Repairing Databases over Metric Spaces with Coincidence Constraints

Youri Kaminsky
Hasso Plattner Institute, University of Potsdam, Germany

Benny Kimelfeld
Technion, Haifa, Israel

Ester Livshits
Technion, Haifa, Israel

Felix Naumann
Hasso Plattner Institute, University of Potsdam, Germany

David Wajc
Technion, Haifa, Israel

Abstract

Datasets often contain values that naturally reside in a metric space: numbers, strings, geographical locations, machine-learned embeddings in a vector space, and so on. We study the computational complexity of repairing inconsistent databases that violate integrity constraints, where the database values belong to an underlying metric space. The goal is to update the database values to retain consistency while minimizing the total distance between the original values and the repaired ones. We consider what we refer to as *coincidence constraints*, which include unary key constraints, inclusion constraints, foreign keys, and generally any restriction on the relationship between the numbers of cells of different labels (attributes) coinciding in a single value, for a fixed attribute set.

We begin by showing that the problem is APX-hard for general metric spaces. We then present an algorithm solving the problem optimally for tree metrics, which generalize both the line metric (i.e., where repaired values are numbers) and the discrete metric (i.e., where we simply count the number of changed values). Combining our algorithm for tree metrics and a classic result on probabilistic tree embeddings, we design a (high probability) logarithmic-ratio approximation for general metrics. We also study the variant of the problem where we limit the allowed change of each individual value. In this variant, it is already NP-complete to decide the existence of any legal repair for a general metric, and we present a polynomial-time repairing algorithm for the case of a line metric.

2012 ACM Subject Classification Information systems → Data management systems

Keywords and phrases Database repairs, metric spaces, coincidence constraints, inclusion constraints, foreign-key constraints

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.14

Related Version *Full Version*: `http://arxiv.org/abs/2409.16713` [21]

Funding *Benny Kimelfeld*: Israel Science Foundation grant 768/19, German Research Foundation grant KI 2348/1-1.
David Wajc: Taub Family Foundation "Leader in Science and Technology" fellowship, Israel Science Foundation grant 3200/24.

1 Introduction

A pervasive problem encountered in databases is *inconsistency*, which means that the database violates some integrity constraints that are expected to hold in reality. Such an anomaly can arise due to mistaken data sources (e.g., Web resources), erroneous data recording (e.g., manual form filling), noisy data generation (e.g., machine learning), imprecise

data integration (e.g., faulty entity resolution), and so on. Commercial tools for data cleaning and transformation (e.g., in data warehouse scenarios) are often based on human-specified constraints to guide the detection and repair of errors. Consequently, a well-studied computational problem that arises with inconsistent data is database *repairing* – suggesting a *minimal intervention* needed to correct the database so that all integrity constraints are satisfied. Studies of this general problem differ in their considered (i) types of integrity constraints, (ii) intervention models, and (iii) intervention cost measure.

Common types of integrity constraints include functional dependencies [8, 18, 23, 26, 28] (including key constraints), the more general denial constraints [5, 12, 13], and inclusion dependencies [8, 12, 27]. The intervention model refers to the operations applied to repair the database, and these are typically tuple deletion, tuple insertion, and value updates [4, 17]. In this work, we focus on the third and seek a so-called *update repair*. For this intervention model, finding an optimal repair is almost always shown to be computationally hard and algorithms developed are therefore mostly heuristic or focus on highly specialized cases [8,13,18,23,26,30]. This state of affairs holds for various intervention cost measures studied, including the number of changed cells [23,26], the sum of changes of numerical values [5], or the sum of user-provided costs of value updates [8, 13], often cast as probabilities of (independent) changes [18, 30].

To obtain provable guarantees for repair costs in a wide variety of update cost measures, in this work we exploit the fact that database values commonly belong to a structured domain, namely a *metric space*, where update costs of cells abide by the laws of a metric space (positivity, symmetry, and the triangle inequality). The obvious example of such a metric space is the space of numerical values [5]. Another simple example is the *discrete metric*, where any two different values are a unit distance apart (hence, the cost of a repair is the number of changed cells [23, 26]). A more sophisticated example is the distance or travel time between map locations. Yet another example is textual values and distance given by edit distance, or the distance between textual embeddings in a Euclidean space [29]. In fact, several studies have proposed ways of embedding general database cells (including opaque keys) in Euclidean spaces, with the objective that semantically close cells should be mapped to geometrically close vectors, and vice versa [9, 10, 33]. Hence, there is a wealth of value types corresponding to metrics, so a unifying approach for repairing databases with metric values can apply to a wide range of applications. Motivated by this observation, we introduce the following general computational problem.

Metric Database Repair Problem (informal; see Section 2)

A *metric database* consists of a set of *cells*, each with a label (attribute) and a value from a metric space. A *coincidence constraint* lists, for each value, the allowed combinations of numbers of coincident cells of different labels. A *repair* of an inconsistent database moves cells between points in the metric space (i.e., updates the cells' values). The goal is to compute a repair, if one exists, minimizing the total distance moved over all cells. The attribute set is fixed, but all else is given as input, including the (finite) metric space.

Coincidence constraints generalize (unary) inclusion constraints, key constraints, foreign-key constraints, as well as other constraints on cardinalities (e.g., the number of people associated with a company is at most the number of employees of the company according to some external trusted registry). Moreover, this class of constraints is closed under conjunction, disjunction, and negation. For example, we can phrase a constraint stating that every location of a team includes one driver (Driver.location cell), and either engineers (Eng.location cells) or salespeople (SP.location cells), but not both.

Contributions. Our first contribution is a formalization of the problem introduced informally above, together with illustrative examples (Section 2). We then begin the complexity investigation of this problem by showing that, in the generality presented here, it is NP-hard and even APX-hard (Section 2.5).

Our main technical contribution is a polynomial-time algorithm for finding an optimal repair when the metric is a *tree metric* (Section 3.1). This algorithm implies the tractability of the problem for well-studied special cases of the tree metric, namely the *line metric* (i.e., the usual metric on numeric values) and the *discrete metric* where, as mentioned above, the cost of a repair is the number of changed cells. Moreover, combining our algorithm for tree metrics with classic results on randomly embedding a general metric into a tree metric [3, 16], we establish a (high-probability) logarithmic-factor approximation for general metrics (Section 3.2).

We also investigate two extensions of our work (Section 4). We first show how the model and results can be generalized to an infinite metric, such as the full Euclidean space (e.g., with the metric ℓ_p). We then discuss the implication of imposing a bound on the amount of change allowed for each individual cell; that is, each point can move by at most some given distance τ. We show that the bound restriction makes the problem fundamentally harder for a general (finite) metric, since testing whether *any* repair exists is already NP-complete (even for a single label or two labels with a simple inclusion constraint). On the other hand, we devise a polynomial-time algorithm for finding an optimal repair for the line metric.

Related Work. For repairs, we restrict the discussion to update repairs. For a broader view of database repair, see the excellent books [4, 17]. Upper and lower bounds have been established for the number of changes (which corresponds to the discrete metric) under functional dependencies [23, 26], which are incomparable with our coincidence constraints. Gilad, Imber and Kimelfeld [18] studied a relational model where each cell has a distribution over possible values (and cells are probabilistically independent), and the goal is to find a most probable instantiation that satisfies integrity constraints; one can translate an optimal repair in our model to a most probable world in their model, yet the metric properties are ignored in their work and, again, their study is restricted to functional dependencies.

Several studies investigated the computation of an optimal repair with a general distance function (that does not necessarily obey the distance/metric axioms); yet, their results are restricted to lower bounds (hardness) and heuristic algorithms without quality guarantees [8, 13]. While Chu et al. [13] focused on denial constraints (which are again incomparable to our coincidence constraints), the work of Bohannon et al. [8] is closer to our work, as they considered inclusion dependencies (in addition to functional dependencies). Bertossi et al. [5] studied the complexity of optimal repairs ("Database Fix Problem") for numerical values under the Euclidean space (square of differences), and focused on denial constraints and aggregation constraints; their results for this problem are lower bounds (NP-completeness).

There is a weaker relevance of this work to frameworks where the repairs themselves (rather than the database values) are points of the metric space [1, 2], and work on consistent query answering (i.e., determining whether all repairs agree on a query answer) under keys and foreign keys [20].

From an opposite angle, there has been considerable work on *approximate* variations of database dependencies, where a database satisfies such a constraint if the database values are *close* to satisfying it. For example, if we assume that values belong to a metric space (as we do in this work), then such a constraint, also known as a *metric constraint* [24, 32] or a *differential dependency* [25, 31], may state that if two tuples are close on their X attributes, they should

Patient (P)	
pid	name
437	Anna
487	Bill
719	Carl
799	Darcy

Registration (R)	
pid	time
779	10:00
437	13:00
199	14:00

Vaccine (V)	
pid	nurse
719	018
481	017
987	078

UsedShots (U)	
nurse	#shots
078	5
017	1

Figure 1 Example relations. The violated constraint is V.pid $\sqsubseteq_k$ R.pid $\sqsubseteq_k$ P.pid: attribute pid is a foreign key of V referencing pid of R, which is a foreign key referencing pid of P. The P relation is assumed to be clean.

also be close on their Y attributes, yielding a soft variation of the functional dependency [11]. This line of work is in the vain of *approximate query answering* (or "similarity joins") where value equality (e.g., for equi-joins) is replaced with metric proximity [14, 15, 19, 34]. The work of Kaminsky, Pena, and Naumann [22] combines inclusion constraints and similarity between values within the problem of constraint discovery (with equality replaced by similarity); their work, as well as other work on the discovery of metric based approximate constraints [25, 31], is relevant here in the sense that it can be combined with ours in a larger flow that improves data quality by the discovery of (violated) integrity constraints, which are then handled by a repairing phase.

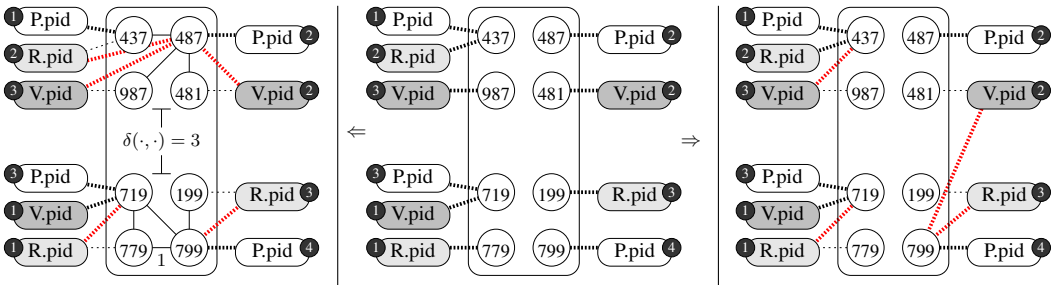
2 Formal Framework

We first describe our formal framework, from the basic concepts to the problem definition.

2.1 Metric Spaces

Recall that a *metric space* is a pair (M, δ) where M is a set of *points* and $\delta : M \times M \to \mathbb{R}$ is a function satisfying the following: *positivity:* $\delta(x, y) \geq 0$ for all $x, y \in M$ and $\delta(x, y) = 0$ if and only if $x = y$; *symmetry:* $\delta(x, y) = \delta(y, x)$ for all $x, y \in M$; and *the triangle inequality:* $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$ for all $x, y, z \in M$. When M is finite, the metric (M, δ) can also be seen as having its distances correspond to the length of the shortest paths in an undirected graph, such as a complete graph on M where each edge (u, v) has weight $\delta(u, v)$.

We will discuss three special cases of metric spaces (M, δ), and we will distinguish between them via a subscript of the distance function δ.



(a) Optimal repair E_1 w.r.t. the string Hamming distance.

(b) Database D_{pid} (for the pid columns of Figure 1).

(c) Optimal repair E_2 w.r.t. the discrete metric.

Figure 2 Optimal repairs of a database D_{pid} (middle) according to two metric spaces (M, δ) (left and right) over the person identifiers (pid).

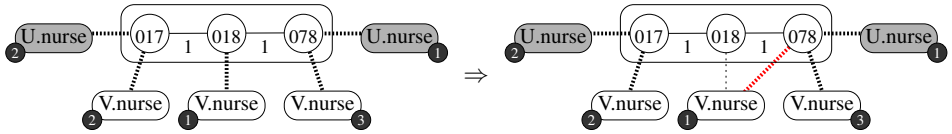


Figure 3 Database D_{nurse} (on the left) and an optimal repairs E according to the Hamming distance and the discrete distance (on the right).

- A *line* metric $(M, \delta_{\mathbb{R}})$, where $M \subseteq \mathbb{R}$ and $\delta_{\mathbb{R}}(x, y) = |x - y|$.
- A *discrete metric* $(M, \delta_{\neq})$, where $\delta_{\neq}(x, y) = 1$ whenever $x \neq y$.
- A *tree* metric (M, δ_T), where T is a weighted tree with the vertex set M, and $\delta_T(x, y)$ is the weighted distance (i.e., sum of weights along the unique path) in T between x and y.

2.2 Databases

We assume three countably infinite sets: Atts is the set of all *attributes*, Cells is a set of all *cells*, and Vals is a set of *values*. Each cell c is labeled with an attribute $\lambda(c) \in \mathsf{Atts}$. If $\lambda(c) = A$, then we call c an *A-cell*. By a *domain* we refer to a set $M \subseteq \mathsf{Vals}$ of values.

A *database* is a mapping D from a finite set of cells, denoted $\mathsf{Cells}(D)$, to values. Hence, we view a database as a function $D : \mathsf{Cells}(D) \to \mathsf{Vals}$. We denote by $\mathsf{Vals}(D)$ the set of values that occur in D, that is, $\mathsf{Vals}(D) := \{D(c) \mid c \in \mathsf{Cells}(D)\}$. We say that D is a database *over* a domain $M \subseteq \mathsf{Vals}$ if $\mathsf{Vals}(D) \subseteq M$. If D is a database and $A \in \mathsf{Atts}$, then we denote by $D[A]$ the restriction of D to its A-cells; hence, $\mathsf{Cells}(D[A]) = \{c \in \mathsf{Cells}(D) \mid \lambda(c) = A\}$, and $D[A](c) = D(c)$ for all $c \in \mathsf{Cells}(D[A])$. If D is a database and $v \in \mathsf{Vals}$, then we denote by $D^{-1}(v)$ the set of cells $c \in \mathsf{Cells}(D)$ with $D(c) = v$. We denote by $\mathsf{Atts}(D)$ the set of attributes that occur in D, that is, $\mathsf{Atts}(D) := \{\lambda(c) \mid c \in \mathsf{Cells}(D)\}$.

A *signature* $\mathbf{A}$ is a sequence $(A_1, \ldots, A_q)$ of attributes. We denote by $\mathsf{Atts}(\mathbf{A})$ the attribute set $\{A_1, \ldots, A_q\}$. An *instance* of $\mathbf{A}$ is a database D such that $\mathsf{Atts}(D) \subseteq \mathsf{Atts}(\mathbf{A})$. In the remainder of this paper, we always assume the presence of a signature $\mathbf{A}$, and a database D that we mention is implicitly assumed to be an instance of $\mathbf{A}$.

▶ **Example 1.** The relational database of Figure 1 consists of a registry of vaccinations in an improvised medical facility. (We later discuss the errors in the relations.) From the "pid" columns we derive the database D_{pid} (in our model) of Figure 2b over the signature $(\text{P.pid}, \text{R.pid}, \text{V.pid})$. Each cell c is depicted by a rounded rectangle with its label $\lambda(c)$ written inside the box and its value $D_{\text{pid}}(c)$ connected to the rectangle by a dotted line. The number attached to each cell denotes the row number of the cell in Figure 1. (It is not part of the formal model and is given here just to help connecting between the figures.) For example, the pid attribute of row 1 of PATIENT gives rise to the left-top cell c with $\lambda(c) = \text{P.pid}$ and $D_{\text{pid}}(c) = 437$. We can similarly obtain the database D_{nurse} over the signature $(\text{V.nurse}, \text{U.nurse})$ from the nurse columns of the relations. This database is depicted on the left side of Figure 3. We use both D_{pid} and D_{nurse} as running examples for this section. ⌟

2.3 Coincidence Constraints

Let $\mathbf{A} = (A_1, \ldots, A_q)$ be a signature. If D is database and $v \in \mathsf{Vals}$, then the sequence $p_D(v) := (|D[A_1]^{-1}(v)|, \ldots, |D[A_q]^{-1}(v)|)$ lists the number of A_i-cells with the value v for $i = 1, \ldots, q$. We call $p_D(v)$ the *coincidence profile* of v (stating how many cells of each attribute coincide at v).

Let $M \subseteq \mathsf{Vals}$ be a domain. A *coincidence constraint over* M (*w.r.t.* $\mathbf{A}$) is a function Γ that maps every value $v \in M$ to a subset $\Gamma(v) \subseteq \mathbb{N}^q$, stating the allowed coincidence profiles for every value in M. A database D over M satisfies Γ, denoted $D \models \Gamma$, if $p_D(v) \in \Gamma(v)$ for all $v \in M$; that is, the coincidence profile of every value in M is allowed by Γ. Conversely, D *violates* Γ if $p_D(v) \notin \Gamma(v)$ for at least one value $v \in M$, and then we denote it by $D \not\models \Gamma$.

Note that the class of coincidence constraints is closed under conjunction (intersection), disjunction (union), and negation (complement). That is, if Γ_1 and Γ_2 are coincidence constraints over M, then $\Gamma_1 \cup \Gamma_2$ (that maps every $v \in M$ to $\Gamma_1(v) \cup \Gamma_2(v)$) is a coincidence constraint over M, and so are $\Gamma_1 \cap \Gamma_2$ and $\mathbb{N}^q \setminus \Gamma_2$. Also note that a $\Gamma(v)$ may be infinite, but for a given database, only a finite subset is relevant (see Remark 6 later in the section).

For illustration, let $\mathbf{A} = (A_1, \ldots, A_q)$, and let M be a set of values. The *key constraint* $\mathsf{key}(A_j)$ states that no two A_j-cells can have the same value. Hence, $\mathsf{key}(A_j)$ says that every value can have at most one A_j-cell. The constraint $\mathsf{key}(A_j)$ can be expressed as the following function (which is the same on every point v):

$$\Gamma_{\mathsf{key}(A_j)}(v) := \{(i_1, \ldots, i_q) \mid i_j \leq 1\}$$

The *inclusion constraint* $A_\ell \sqsubseteq A_j$ states that every value in an A_ℓ-cell must also occur in an A_j-cell. Hence, $A_\ell \sqsubseteq A_j$ states that every value that has one or more A_j-cells must also have one or more A_ℓ cells, and can be expressed by the following coincidence constraint:

$$\Gamma_{A_\ell \sqsubseteq A_j}(v) := \{(i_1, \ldots, i_q) \mid i_\ell = 0 \text{ or } i_j > 0\}$$

Finally, the *foreign-key constraint* $A_\ell \sqsubseteq_{\mathsf{k}} A_j$ is the conjunction of $\mathsf{key}(A_j)$ and $A_\ell \sqsubseteq A_j$, hence expressed by the coincidence constraint $\Gamma_{A_\ell \sqsubseteq_{\mathsf{k}} A_j} := \Gamma_{\mathsf{key}(A_j)} \cap \Gamma_{A_\ell \sqsubseteq A_j}$.

A coincidence constraint Γ may, in principle, pose a different restriction on every value of M. We also consider the uniform case where Γ is the same everywhere. Formally, we say that Γ is *uniform* if $\Gamma(v) = \Gamma(u)$ for all pairs u and v of values in M. In this case, we may write just Γ instead of $\Gamma(v)$, and treat Γ simply as a set of coincidence profiles $(i_1, \ldots, i_q)$. For example, each of $\Gamma_{\mathsf{key}(A_j)}$, $\Gamma_{A_\ell \sqsubseteq A_j}$ and $\Gamma_{A_\ell \sqsubseteq_{\mathsf{k}} A_j}$ is uniform since its definition does not depend on v, hence we can write, for instance, $\Gamma_{\mathsf{key}(A_j)} := \{(i_1, \ldots, i_q) \mid i_j \leq 1\}$.

▶ **Example 2.** We continue with our running example. The scenario we consider is that Figure 1 describes a relational database with a combination of clean data in the Patient and UsedShots relations (where administrators insert records), and noisy data in the Registration and Vaccine relations, where records are entered (manually or via OCR) from handwritten forms that may include mistakes or ambiguous letters.

Consider again the databases D_{pid} in Figure 2b. derived from Figure 1. The constraints that we associate to the domain of D_{pid} is V.pid $\sqsubseteq_{\mathsf{k}}$ R.pid $\sqsubseteq_{\mathsf{k}}$ P.pid, stating that every vaccine is given to a registered patient who, in turn, is in the patient registry, and no two registrations, or two patient records, coincide in their value. For the relations of Figure 1 it means that pid is a foreign key from Vaccine to Registration and from Registration to Patient. In our formalism, this translates to the uniform constraint Γ, where $\Gamma(v) := \Gamma_{\text{V.pid} \sqsubseteq_{\mathsf{k}} \text{R.pid}}(v) \cap \Gamma_{\text{R.pid} \sqsubseteq_{\mathsf{k}} \text{P.pid}}(v)$ for very value v. Note that the constraint is violated by D_{pid} since, for example, `719` is a value of a V.pid-cell but not an R.pid-cell, and `779` is a value of an R.pid-cell but not a P.pid-cell. ⌟

▶ **Example 3.** Still within our running example, for the domain of D_{nurse} (Figure 3), recall that our signature is (V.nurse, U.nurse). Our constraint Γ for this domain defined by

$$\Gamma(v) \;:=\; \Gamma_{\text{V.nurse} \sqsubseteq \text{U.nurse}}(v) \;\cap\; \Gamma_{\text{U.nurse} \sqsubseteq \text{V.nurse}}(v) \;\cap\; \{(i_1, i_2) \mid i_1 \leq \text{shots}(v)\}$$

where shots(v) is the number in the row of v in the (clean) relation UsedShots of Figure 1, that is, shots(078) = 5 and shots(017) = 1. Hence, Γ states that every vaccinating nurse should occur in the registry of the used shots (i.e., V.nurse $\sqsubseteq$ U.nurse) and vice versa, and the number of times a nurse is recorded as giving a vaccine (that is, the number of V.nurse cells per nurse identifier) is at most the number of used shots recorded for that nurse. Note that Γ is non-uniform since it differs for the two values 078 and 017. It is violated since the value 018 has a V.pid-cell but not a U.pid-cell. ⌟

2.4 Repairs

Let $\mathbf{A} = (A_1, \ldots, A_q)$ be a signature, (M, δ) be a metric space, and Γ be a coincidence constraint over M. An *inconsistent database* is a database D over M such that $D \not\models \Gamma$. A *repair* of D is a database E such that $\mathsf{Cells}(E) = \mathsf{Cells}(D)$ and $E \models \Gamma$. The *cost* of a repair E is the cumulated distance that the values of D undergo in the transformation to E. We allow to differentiate the cost between different attributes, so we assume a global weight function $w : \mathsf{Atts}(\mathbf{A}) \to \mathbb{R}_{\geq 0}$.[1] Hence, we define the cost of a repair E, denoted $\kappa(D, E, \delta)$, by

$$\kappa(D, E, \delta) := \sum_{c \in \mathsf{Cells}(D)} w(\lambda(c)) \cdot \delta(D(c), E(c)) \,.$$

▶ **Example 4.** Continuing Example 2, assume that w(P.pid) = ∞ (or some large number), since P.pid-cells are assumed to be clean, and that w(R.pid) = 1 and w(V.pid) = 1.1 (since V-pid cells are deemed slightly more reliable than R.pid-cells). Figures 2a and 2c show optimal repairs of D_{pid} of Figure 2b with respect to (M, δ_1) and (M, δ_2), respectively, where

- $M = \{437, 487, 987, 481, 719, 199, 779, 799\}$ (i.e., $\mathsf{Vals}(D_{\text{pid}})$);
- δ_1 is the Hamming distance between strings (e.g., $\delta_1(437, 487) = 1$ and $\delta_1(437, 987) = 2$);
- δ_2 is the discrete distance over M (e.g., $\delta_2(437, 487) = \delta_2(437, 987) = 1$).

Note that $\kappa(D_{\text{pid}}, E_1, \delta_1) = 2{\cdot}1 + 3{\cdot}1.1 = 5.3$, since there are two changes of R.pid and three of V.pid, each of distance 1. (Changes are marked by red lines.) Also note that $\kappa(D_{\text{pid}}, E_2, \delta_2) = 4.2$ as there are two changes of R.pid and two of V.pid, but $\kappa(D_{\text{pid}}, E_2, \delta_1) = 7.5$ since:

$$\text{R.pid}: \quad 1 \cdot \delta_1(779, 719) + 1 \cdot \delta_1(199, 799) = 1 + 1 = 2; \tag{1}$$

$$\text{V.pid}: \quad 1.1 \cdot \delta_1(987, 437) + 1.1 \cdot \delta_1(481, 799) = 2.2 + 3.3 = 5.5\,. \tag{2}$$

In particular, note that E_1 is superior to E_2 for the metric δ_1. ⌟

▶ **Example 5.** Continuing Example 3, recall that D_{nurse} violates Γ because the value 018 is associated with a V.nurse-cell but no U.pid-cells. Assume that w(U.nurse) = ∞ since the U.nurse-cells are assumed to be clean, but w(V.nurse) = 1. Under both the discrete metric and the Hamming distance, an optimal repair is obtained by changing the cell c of 018 to 078, as illustrated at the right of Figure 3. Note that changing the value of c to 017 would not be legal, since it would violate the constraint $\{(i_1, i_2) \mid i_1 \leq \text{shots}(017)\}$, as i_1 would be two (corresponding to two V.nurse-cells with the value 017 while shots(017) = 1. ⌟

2.5 Computational Problem

We study the complexity of computing a low-cost repair. Formally, we assume a fixed signature $\mathbf{A}$. The input consists of a finite metric space (M, δ), a finite coincidence constraint Γ over M, and an inconsistent database D over M. The goal is to compute an *optimal* repair,

[1] For example, it may be the case that we trust A_i-cells more than we trust A_j-cells, so the same movement by distance ϵ can contribute differently to the cost of E; this will be reflected in $w(A_i) > w(A_j)$.

that is, a repair E such that $\kappa(D, E, \delta) \leq \kappa(D, E', \delta)$ for all repairs E', or declare that no repair exists. We will also study the approximate version of finding an α-optimal repair, where α is a number (or a numeric function of the input), which is a repair E such that $\kappa(D, E, \delta) \leq \alpha \cdot \kappa(D, E', \delta)$ for all repairs E'.

▶ Remark 6. The assumption that $\mathbf{A}$ is fixed (i.e., concerning the *data complexity* the problem) is essential in this work, as it means that we can traverse in polynomial time all the profiles that are relevant to a repair: for a given database D, these are the profiles $(i_1, \ldots, i_q)$ where every number i_j is at most $|D[A_j]|$. In particular, our polynomial-time algorithms do not deal with the manner in which the constraints are represented. In fact, for this reason, the constraint $\Gamma(v)$ could even be infinite, and examples of such constraints are shown in Section 2.3; hence, the assumption that Γ is given as part of the input is not a limitation. ⌟

It may be unclear upfront whether we can even test in polynomial time whether *any* repair exists (i.e., our version of the *existence of repair* problem [4]). It follows immediately from our later results on optimal repairs (e.g., Theorem 8) that this problem is, indeed, solvable in polynomial time. Yet, our first result in the next section states hardness of approximation, even when a repair is guaranteed to exist.

2.6 Hardness

The following theorem states that for general input metrics, it is NP-hard to approximate the optimal repair beyond some fixed ratio. Recall that the metric space (M, δ) is given explicitly as part of the input (e.g., as a graph). Be reminded that APX-hardness means that there is some constant $\alpha > 1$ such that there is no polynomial-time algorithm for computing an α-approximation unless P = NP. The proof (in the full version [21]) is via a PTAS reduction from the problem of finding a minimum cover by 3-sets.

▶ **Theorem 7.** *Let $\mathbf{A} = (A_1, \ldots, A_q)$ be a signature with $q \geq 2$. Minimizing the cost of a repair is APX-hard, even if the weight w is uniform, Γ is uniformly the inclusion constraint $A_1 \sqsubseteq A_2$, and a repair is guaranteed to exist.*

The remainder of this paper is therefore dedicated to obtaining optimal algorithms for metrics of interest (notably, the line metric and discrete metric), and providing provable approximation algorithms for general metrics.

3 Algorithms

In this section, we present our repair algorithms. We begin with an exact algorithm for tree metrics. This algorithm immediately implies the tractability of the line and discrete metrics (as we will show in Corollary 9). The exact algorithm for trees will also play a central role in the approximation algorithm for general metrics in the second half of this section.

3.1 Algorithm for Tree Metrics

Many problems in tree metrics are amenable to dynamic programming approaches, allowing for polynomial-time algorithms for problems that might be intractable on general metrics. This is also the case here. In this section, we devise a polynomial-time algorithm for finding an optimal repair in the case of a tree metric. Formally, we will prove that:

▶ **Theorem 8.** *An optimal repair can be found in polynomial time (if exists), given a tree metric space (M, δ_T), a coincidence constraint Γ over M, and an inconsistent database D.*

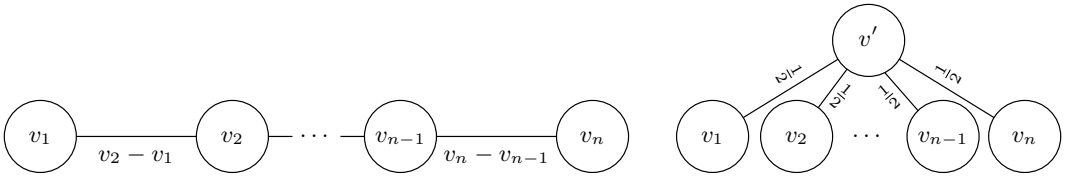


Figure 4 The line metric $(M, \delta_{\mathbb{R}})$ (left) and discrete metric $(M, \delta_{\neq})$ (right) cast as tree metrics, with $M = \{v_1, \ldots, v_n\}$.

Theorem 8 implies, in particular, that we can test in polynomial time whether a repair exists (regardless of the metric): apply the theorem to an arbitrary tree metric over the points. Moreover, the tractability of tree metrics implies the same for other basic metrics:

▶ **Corollary 9.** *An optimal repair can be found in polynomial time (if exists) in the case of a line metric $(M, \delta_{\mathbb{R}})$, and in the case of the discrete metric $(M, \delta_{\neq})$ over M.*

Proof. Let $M = \{v_1, \ldots, v_n\}$. The corollary follows from Theorem 8 by casting the two metrics as tree metrics, as illustrated in Figure 4. The case of a line metric $(M, \delta_{\mathbb{R}})$ is straightforward; if (w.l.o.g.) $v_1 < v_2 < \cdots < v_n$, then $(M, \delta_{\mathbb{R}})$ is the same as the tree metric (M, δ_T) where T is the path $v_1 — \ldots — v_n$ where the weight of each edge $\{v_i, v_{i+1}\}$ is $v_{i+1} - v_i$. In the case of the discrete metric $(M, \delta_{\neq})$ over M, the tree T is a star with the leaves $v_1, \ldots, v_n$ and a new vertex v' as the center. The weight of every edge is $1/2$. To make sure that (optimal) repairs do not place any cell in v', we define $\Gamma(v') = (0, \ldots, 0)$. ◀

Next, we prove Theorem 8 by showing the repairing algorithm. Throughout this section, we assume the signature $\mathbf{A} = (A_1, \ldots, A_q)$ and a given input (M, δ_T, Γ, D). We further assume that the tree T itself is given (or, otherwise, we can reconstruct it from δ_T).

General idea. Before we delve into the details of the algorithm, we give the general intuition behind it. The initial direction is to process the tree bottom-up, starting from the leaves, and compute the best repair for each subtree we process. However, an optimal repair for the subtree T_v rooted at a vertex v (if any exists) is not necessarily useful when processing the ancestors u of v, since an optimal solution for T_u may need to transfer cells from T_v to vertices outside of T_v, or cells from outside of T_v into T_v (as illustrated in Figure 5). Nevertheless, due to the uniformity of the cost of moving cells of each type A_j, it would suffice to find the best repair knowing just the *number* of cells moved inside/outside T_v without knowing the identity of these cells. So, we compute the best repair under the assumption that we need to

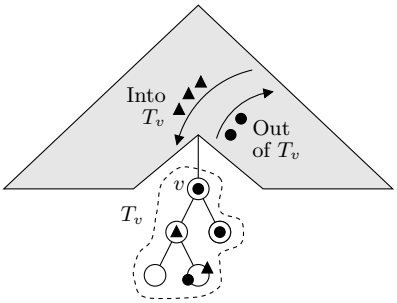


Figure 5 Illustration of the algorithm for a tree metric: the dynamic program accounts for ▲-cells and ●-cells moved into and out of the subtree.

handle a specific number t_j of A_j-cells (which can be lower or higher than the number of A_j-cells in T_v), for every attribute A_j. Moreover, we consider all possible (yet polynomially many) combinations $(t_1, \ldots, t_q)$ for the q attributes in the signature $\mathbf{A} = (A_1, \ldots, A_q)$.

Translation into a binary tree. We first transform the tree T into a binary tree where every internal vertex has precisely two children. We do so by introducing new vertices with distance zero to their parents. While this construction violates the property of non-zeroness of the metric, this does not matter for the algorithm, which is optimal even for a pseudometric (where distinct points can be of distance 0). The end result is that every point in M is a vertex of T, and some vertices of T are not in M (as we introduced them for the construction). Let M' be the set of new vertices (hence, $M \subseteq M'$). We extend Γ to M' by defining $\Gamma'(v) = \Gamma(v)$ for every $v \in M$ and $\Gamma'(v) = (0, \ldots, 0)$ for every $v \in M' \setminus M$. This ensures that our solution places no cells in the new vertices, and so, the result is a legal repair.

Hereon, we will assume that T is binary to begin with, and that M and Γ are, to begin with, the above constructed M' and Γ'.

Placement. The algorithm deploys dynamic programming that processes T bottom-up, leaf to root. For a vertex v of T, we denote by T_v the subtree of T rooted at v, and by D_v the subset of D with cells having points inside T_v.

Let $\mathbf{A} = (A_1, \ldots, A_q)$. For $j = 1, \ldots, q$, let n_j be the number of A_j-cells of D, that is, $n_j := |D[A_j]|$. Let $t_1, \ldots, t_q$ be integers, where each t_j is in $[0, n_j]$. A $(t_1, \ldots, t_q)$-*placement* in T_v is a *consistent* database E that is obtained by positioning $t_1 + \cdots + t_q$ cells of D in T_v, where the number of A_j-cells is t_j. Some of these cells may belong to D_v and others may be outside of D_v. Cells of the former kind are called *resident cells* and those of the latter kind are *visitor cells*. Note that the consistency of a $(t_1, \ldots, t_q)$-placement considers only T_v and the resident and visitor cells, and ignores the rest of T and the remaining cells of D; in particular, it may be the case that a $(t_1, \ldots, t_q)$-placement exists, but it cannot be extended to a full repair (e.g., since the other metric points cannot contain the remaining cells).

The *cost* of a $(t_1, \ldots, t_q)$-placement E is the sum $\sum_{c \in \mathsf{Cells}(E) \cup \mathsf{Cells}(D_v)} w(\lambda(c)) \cdot \Delta(c)$ where:

$$\Delta(c) := \begin{cases} \delta_T(D(c), E(c)) & \text{if } c \in \mathsf{Cells}(D_v) \cap \mathsf{Cells}(E); \\ \delta_T(D(c), v) & \text{if } c \in \mathsf{Cells}(D_v) \setminus \mathsf{Cells}(E); \\ \delta_T(v, E(c)) & \text{if } c \in \mathsf{Cells}(E) \setminus \mathsf{Cells}(D_v). \end{cases}$$

In words, we consider the transformation of E from D_v; if c is a cell that moves from one vertex of T_v to another, then $\Delta(c)$ is the distance between the two vertices. If c is a resident cell that disappears, then $\Delta(c)$ is the cost of moving c to the root. If c is a visitor cell that occurs in E, then $\Delta(c)$ is the cost of moving c from the root to its vertex.

Given $t_1, \ldots, t_q$, we will compute, for each vertex v, the minimum-cost $(t_1, \ldots, t_q)$-placement (among all sets of cells) in T_v. This value will be stored as $Opt_v(t_1, \ldots, t_q)$ where, as a special case, the cost of the best repair of D is $Opt_r(n_1, \ldots, n_q)$, where r is the root vertex of T. If no $(t_1, \ldots, t_q)$-placement exists, then $Opt_v(t_1, \ldots, t_q)$ is ∞. (As often done in dynamic programming, the actual repair is obtained by restoring the optimal placements that produce the least cost $Opt_r(n_1, \ldots, n_q)$.)

Handling leaves. When v is a leaf, we define $Opt_v(t_1, \ldots, t_q) = 0$ if $(t_1, \ldots, t_q) \in \Gamma(v)$ (hence, the coincidence profile of v is legal); otherwise, $Opt_v(t_1, \ldots, t_q) = \infty$.

Handling internal vertices. We now consider the case where v is an internal vertex. For a vertex u of T and $j = 1, \ldots, q$, we use $n_j[T_u]$ to denote the number of A_j-cells in the tree T_u (as positioned in D), that is, the sum of $|(D^{-1}(u'))[A_j]|$ over all vertices u' in the subtree T_u.

To find an optimal $(t_1, \ldots, t_q)$-placement for setting $Opt_v(t_1, \ldots, t_q)$, we compute the minimum cost for every coincidence profile $(i_1, \ldots, i_q) \in \Gamma(v)$ of the vertex v, and then take the least-cost entry across all profiles. In the remainder of this part, we fix $(i_1, \ldots, i_q)$.

Recall that v is an internal vertex, and so, has precisely two children. Let us denote them by v_1 and v_2. To obtain our optimal $(t_1, \ldots, t_q)$-placement, we can *pull* from T_{v_ℓ} (where $\ell \in \{1, 2\}$) a set of A_j-cells of size p for $p \in \{0, \ldots, n_j[T_{v_\ell}]\}$, or *push* to T_{v_ℓ} a set A_j-cells of size p for $p \in \{0, \ldots, t_j - n_j[T_{v_\ell}]\}$. (Note that there is no gain in pulling A_j-cells and pushing A_j-cells at the same time since the placement can use the pulled cell instead of the pushed cell with no additional cost, or even a lower cost.) We say uniformly that we pull from T_{v_ℓ} a set of A_j-cells of size $p_{\ell,j}$ for $p_{\ell,j} \in \{-(t_j - n_j[T_{v_\ell}]), \ldots, n_j[T_{v_\ell}]\}$, where the meaning of pulling $-p$ cells, for $p \geq 0$, is pushing p cells. Once we pull A_j-cells from (or push A_j-cells to) T_{v_ℓ}, we place the cells optimally in T_{v_ℓ}. To find the cost of that, we use a previously computed $Opt_{v_\ell}(t^\ell_1, \ldots, t^\ell_q)$ where $t^\ell_j = n_j[T_{v_\ell}] - p_{\ell,j}$ (i.e., the number of A_j-cells that remain in T_{v_ℓ}). Hence, the total cost is given by:

$$\sum_{\ell=1,2} \Big(\sum_{j=1}^{q} \big(w(A_j) \cdot p_{\ell,j} \cdot \delta_T(v, v_\ell) \big) + Opt_{v_\ell}(t^\ell_1, \ldots, t^\ell_q) \Big) \tag{3}$$

We will then iterate over all *legal* combinations of $p_{\ell,j}$ and take the minimal cost according to Equation (3). The combination is legal if, for all $j = 1, \ldots, q$, the number of cells that we position in T_v is indeed t_j. This number consists of the number of A_j-cells that remain in each T_{v_ℓ}, namely $n_k[T_{v_\ell}] - p_{\ell,j}$, plus the number i_j of A_j-cells that remain in v; hence, $i_j + p_{1,j} + p_{2,j} = t_j$.

This concludes the description of the dynamic program. Clearly, the execution of the program terminates in polynomial time. (Recall that the signature $\mathbf{A} = (A_1, \ldots, A_q)$ is fixed and, in particular, q is treated as a constant.) The dynamic program is correct in the sense that it constructs an optimal repair if any repair exists. This is proved by showing that the program is indeed solving the generalized optimization problem:

▶ **Lemma 10.** *Let v be a vertex of T. When we process v with $t_1, \ldots, t_q$, we compute the least cost of a $(t_1, \ldots, t_q)$-placement in T_v, or ∞ if none exists.*

The proof is straightforward from the description of the algorithm in this section.

We remark here that the algorithm is exponential in the size q of the (fixed) signature $\mathbf{A} = (A_1, \ldots, A_q)$, and the exponent is manifested in two parts of the algorithm: first, we assume that we can explicitly enumerate all sequences of each $\Gamma(v)$; second, the dynamic program handles every combination $(t_1, \ldots, t_q)$ for its placement optimization.

3.2 General (Finite) Metrics

Leveraging our algorithm for tree metrics and classic results for probabilistic tree embeddings, in this section we devise a logarithmic-ratio approximation for general (finite) metrics. Formally, we will prove that:

▶ **Theorem 11.** *There is a polynomial-time randomized algorithm that, given a metric (M, δ), a coincidence constraint Γ, an inconsistent database D, and an error probability $\epsilon > 0$, finds an $O(\log |M|)$-optimal repair with probability at least $1 - \epsilon$, if any repair exists.*

In the remainder of this section, we prove Theorem 11. We fix the input (M, δ, Γ, D) for the rest of this section. The proof is based on the following classic *tree embedding* lemma [16] that allows for the translation of exact algorithms for tree metrics to $O(\log |M|)$-approximation algorithms for arbitrary metrics.

▶ **Lemma 12** ([16]). *Given a metric (M, δ), there exists a polynomial-time samplable distribution $\mathcal{P}$ over weighted trees T defining tree metrics (M, δ_T) such that for every $u, v \in M$:*
1. $\delta(u, v) \leq \delta_T(u, v)$ *for every T in the support of $\mathcal{P}$.*
2. $\mathbb{E}_{T\sim\mathcal{P}}[\delta_T(u, v)] = O(\log |M|) \cdot \delta(u, v)$.

Blelloch, Gu, and Sun [6] show a near-linear-time counterpart of Lemma 12. We note that randomization and expectation are crucial for this theorem; for example, if we embed an n-point cycle with unit-distance edges in any tree, then at least one pair of vertices will suffer an $\Omega(n)$ distortion [3, Theorem 7].

By combining Theorem 8 and Lemma 12, we will show how we obtain an $O(\log |M|)$ approximation, with high probability, in polynomial time. We do so by repeatedly finding an optimal repair for multiple random trees T, and taking the best outcome. More precisely, to obtain an $O(\log |M|)$-approximation with probability at least $1 - \epsilon$, we take the best outcome out of the $O(\log \frac{1}{\epsilon})$ repetitions of the following procedure:
1. Select a random tree T according to Lemma 12.
2. Find an optimal repair E_T for (M, δ_T), Γ and D.

We show that the process gives an $O(\log |M|)$-approximation with probability at least $1 - \epsilon$. Let us denote by E_{opt} an optimal repair for the original metric (M, δ). We use the following two lemmas.

▶ **Lemma 13.** $\mathbb{E}_{T\sim\mathcal{P}}[\kappa(D, E_T, \delta)] \leq O(\log |M|) \cdot \kappa(D, E_{\text{opt}}, \delta)$.

Proof (Sketch). We prove the lemma (in the archive version of this paper [21]) by analyzing $\mathbb{E}_{T\sim\mathcal{P}}[\kappa(D, E_T, \delta)]$, applying the linearity of expectation and Lemma 12. ◀

▶ **Lemma 14.** $\Pr_{T\sim\mathcal{P}}\Big[\kappa(D, E_T, \delta) \leq C \cdot \log |M| \cdot \kappa(D, E_{\text{opt}}, \delta)\Big] \geq \frac{1}{2}$ *for some constant $C > 0$. In words, E_T is $O(\log |M|)$-optimal with probability at least $1/2$.*

Proof. It follows immediately from the inequality of Lemma 13 and Markov's inequality. ◀

The two lemmas suffice to prove Theorem 11, since the randomized procedure can be seen as a Bernoulli trial that succeeds (i.e., produces a good approximation) with a probability of at least $1/2$; hence, we see at least one success with probability $1 - \epsilon$ after $\log_2 \frac{1}{\epsilon} = O(\log \frac{1}{\epsilon})$ independent trials.

4 Extensions

In this section, we study two extensions of our study: the case of an infinite metric, and bound restriction on the movement of each individual cell.

4.1 Infinite Metrics

Up to now, we have considered databases over a finite metric (M, δ) that is given explicitly as part of the input. In particular, a repair could use only values from the given point set M. There are, however, natural situations where the metric is a known *infinite* metric that can provide additional points for repairs. We wish to be able to repair an inconsistent database by using arbitrary values from the infinite metric. An example is the ℓ_p-metric (M, δ) where M is the Euclidian space $\mathbb{R}^k$ and δ is the norm $\|\cdot\|_p$ over M, that is, $\delta(v, u) = \|v - u\|_p$.

To model the computational problem in the case of infinite metrics, we consider the case where the metric (M, δ) is fixed and infinite. Moreover, the coincidence constraint Γ is fixed, and we restrict the discussion to a uniform Γ (that maps every point in M to the same, possibly infinite, set of coincidence profiles). We will further assume that Γ contains the profile $(0, \ldots, 0)$ since, otherwise, it is impossible to satisfy Γ using any database, as our databases are finite. Computationally, we only require polynomial-time computation of distances $\delta(u, v)$, given u and v, and membership testing in Γ, given $(i_1, \ldots, i_q)$.

The following theorem states that we need not use points outside of D if we are satisfied with a 2-approximation and the coincidence constraint is closed under addition. Note that a uniform coincidence constraint Γ over M is *closed under addition* if for every pair $(i_1, \ldots, i_q)$ and $(i'_1, \ldots, i'_q)$ of profiles in Γ, the profile $(i_1 + i'_1, , \ldots, i_q + i'_1,)$ is also in Γ. For illustration, referring to Section 2.3, the inclusion constraint $\Gamma_{A_j \sqsubseteq A_\ell}$ is closed under addition, but the key constraint $\Gamma_{\mathsf{key}(A_j)}$ and the foreign-key constraint $\Gamma_{A_j \sqsubseteq_{\mathsf{k}} A_\ell}$ are *not* closed under addition.

▶ **Proposition 15.** *Let (M, δ) be an infinite metric space, Γ a uniform coincidence constraint, and D an inconsistent database. If Γ is closed under addition, then there exists a 2-optimal repair E such that $\mathsf{Vals}(E) \subseteq \mathsf{Vals}(D)$.*

Proof (Sketch). To establish E from an optimal repair E_0, we take every point in $\mathsf{Vals}(E_0) \setminus \mathsf{Vals}(D)$ and move all cells in that point to the nearest cell in $\mathsf{Vals}(D)$. The distance to each moved cell can then grow at most twice. See the archive version of the paper [21]. ◀

Note that it is necessary to make an assumption on the coincidence constraint in Proposition 15. If we remove the assumption, the statement is false simply because there may be no repair at all over the domain of D. For example, if Γ is the key constraint $\mathsf{key}(A_j)$, then it may be necessary to introduce new metric points if D has fewer points than A_j-cells.

Proposition 15 implies that we can reduce the case of an infinite metric to the case of a finite one (assuming that the coincidence constraint is closed under addition). In particular, we can apply Theorem 11 to conclude a logarithmic approximation in the infinite case.

▶ **Theorem 16.** *Let (M, δ) be an infinite metric space, and let Γ be a uniform coincidence constraint that is closed under addition. There is a polynomial-time randomized algorithm that, given an inconsistent database D and an error probability $\eta > 0$, finds an $O(\log |M'|)$-optimal repair for $M' = |\mathsf{Vals}(D)|$, with probability at least $1 - \eta$.*

The assumption on Γ is crucial for the correctness of Theorem 16. For example, suppose that (M, δ) is the ℓ_p-metric and Γ is a key constraint. In that case, we can construct a repair with an arbitrarily small cost, by generating enough points around those of D. Hence, any α-optimal solution must be an optimal solution, since our approximation ratio is multiplicative. In particular, we cannot get any approximation guarantee by using the tree embedding of Lemma 12.

4.2 Bound Restriction

Note that a low cost of a repair E does not necessarily imply that an individual cell is moved to a point that is close to its origin. This is due to our choice to measure the cost of a repair as the sum of cell movements. It is clearly of interest to consider the variant of the problem where we limit the movement of individual cells by a threshold.

In this section, we consider the extension of our repairing problem where a bound is posed on the maximal movement of a cell. Formally, consider a metric space (M, δ), a coincidence constraint Γ, and an inconsistent database D. For a threshold $\tau > 0$, a $(\delta \leq \tau)$-*repair* is a repair E such that $\delta(D(c), E(c)) \leq \tau$ for every $c \in \mathsf{Cells}(D)$. Our goal is now to find a $(\delta \leq \tau)$-repair E with a minimal $\kappa(D, E, \delta)$.

However, the bound restriction is nontrivial to deal with. Our algorithms inherently fail to deal with this restriction. The dynamic-programming algorithm of Theorem 8 is based on the fact that we can remember the total movement of cells from/to the root, but not any information about individual cells. Moreover, the approximation using the random tree embedding of Lemma 12 cannot lead to the support of a bound restriction, since every random tree may (unavoidably) have a high distortion, meaning that two individual points u and v can be way farther in δ_T than in δ. In fact, even the existence of a repair (regardless of its cost) becomes an intractable problem in the presence of a bound constraint.

▶ **Proposition 17.** *It is NP-complete to determine whether any $(\delta \leq \tau)$-repair exists, given a (finite) metric (M, δ), a coincidence constraint Γ, an inconsistent database D, and threshold τ. The problem remains NP-hard in each of the following cases:*

1. *The signature $\mathbf{A}$ consists of a single attribute and Γ is uniform.*
2. *The signature is $\mathbf{A} = (A_1, A_2)$ and Γ is the inclusion constraint $\Gamma_{A_1 \sqsubseteq A_2}$.*

Proof (Sketch). For the first case we devise a reduction from exact cover by 3-sets, and for the second we use CNF satisfiability. See the archive version of the paper [21]. ◀

Algorithm for the Line Domain

In contrast to the hardness shown in Proposition 17, we can efficiently force a bound restriction in the case of a line metric.

▶ **Theorem 18.** *An optimal $(\delta \leq \tau)$-repair can be found in polynomial time, given Γ, D, τ and a line metric (M, δ). The same holds true if $(M, \delta) = (\mathbb{R}, \delta_{\mathbb{R}})$ is fixed and Γ is fixed, uniform, and closed under addition.*

In the remainder of this section, we prove Theorem 18 by presenting an algorithm for computing an optimal $(\delta \leq \tau)$-repair. Throughout the section, we assume that $M \subseteq \mathbb{R}$ is a set of numbers and δ is $\delta_{\mathbb{R}}$. We consider separately the cases where M is finite and given as part of the input, and where M is $\mathbb{R}$ itself.

Given (finite) metric. We first introduce some notation. Let D be a database.

A *subset* of D is a database is D' such that $\mathsf{Cells}(D') \subseteq \mathsf{Cells}(D)$ and $D'(c) = D(c)$ for all $c \in \mathsf{Cells}(D')$. We write $D' \subseteq D$ to state that D' is a subset of D, and $D \setminus D'$ to denote the complement of D', that is, the subset D'' of D with $\mathsf{Cells}(D'') = \mathsf{Cells}(D) \setminus \mathsf{Cells}(D')$. If $D' \subseteq D$ and E is a repair of D, then $E_{|D'}$ denotes the subset E' of E with $\mathsf{Cells}(E') = \mathsf{Cells}(D')$.

A *prefix* of D is a database that comprises a prefix of D's cells for each attribute; that is, it is a database $D' \subseteq D$ such that if $c \in \mathsf{Cells}(D'[A_j])$ for some attribute A_j, then for all $c' \in \mathsf{Cells}(D[A_j])$ with $D(c') < D(c)$, it holds that $c' \in \mathsf{Cells}(D'[A_j])$. We write $D' \subseteq_{\mathsf{p}} D$ to denote that D' is a prefix of D. We say that D' is a *strict* prefix of D if $D' \subseteq_{\mathsf{p}} D$ and $\mathsf{Cells}(D') \subsetneq \mathsf{Cells}(D)$. If D' is a prefix of D, then the complement $D \setminus D'$ is a *suffix* of D.

We say that D is *contracted* if D consists of a single point, that is, $D(c) = D(c')$ for all c and c' in $\mathsf{Cells}(D)$. For a set $C \subseteq \mathsf{Cells}(D)$, we say that C *satisfies* Γ (denoted $C \models \Gamma$) if $D \models \Gamma$ for a contracted database D with $\mathsf{Cells}(D) = C$. (Note that either all such contracted D or none of them satisfy Γ, since the common cell value has no impact on satisfying Γ.)

The following lemma implies that, without loss of generality, we can assume that an optimal repair consists of two parts, as illustrated in Figure 6: one is an optimal repair of a strict prefix, and the other is a contraction of the suffix.

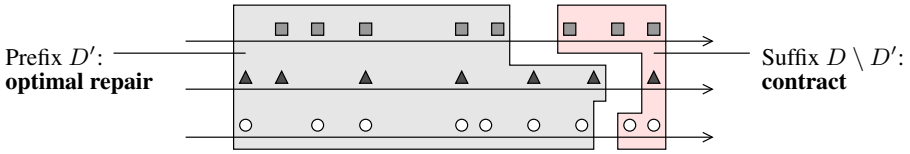


Figure 6 A database D over the line metric. Each shape corresponds to a cell of one of three labels (circle, triangle, square). The structure of an optimal repair by Lemma 19 comprises of two: an optimal repair for a strict prefix, and a contracted suffix.

▶ **Lemma 19.** *Let D be an inconsistent database over a line metric (M, δ). If any $(\delta \leq \tau)$-repair exists, then there is an optimal $(\delta \leq \tau)$-repair E and a strict prefix D' of D with the following properties.*

1. *$E_{|D'}$ is an optimal $(\delta \leq \tau)$-repair of D'.*
2. *$E_{|D \setminus D'}$ is contracted.*

Proof (Sketch). The proof (in the archive version [21]) is based on showing that two A-cells cannot cross each other in their movement from D to E. ◀

Note that D has a polynomial number of prefixes. Also note that $\subseteq_{\mathsf{p}}$ is a partial order over the prefixes of D. Hence, due to Lemma 19, we can apply dynamic programming to compute an optimal $(\delta \leq \tau)$-repair of every prefix D' of D, where we traverse the prefixes in a topological order according to $\subseteq_{\mathsf{p}}$, starting with the empty set.

We compute as follows the cost $\kappa(D', E', \delta)$ of an optimal $(\delta \leq \tau)$-repair E' of D'. (By recording the decisions throughout the procedure, we can derive the repair itself.) Let $v_1, \ldots, v_n$ be the values of M, with $v_i < v_j$ for all $i < j$. For a prefix D' of D, we denote by $V_r^{D'}$ the cost of a $(\delta \leq \tau)$-repair E' of D' such that $E'(c) \in \{v_1, \ldots, v_r\}$ for all $c \in \mathsf{Cells}(D)$, and $\kappa(D', E', \delta)$ is minimal among all $(\delta \leq \tau)$-repairs of D' satisfying this property. Our final goal is to compute the value V_n^D. We show how to compute $V_r^{D'}$ using dynamic programming.

1. If $\mathsf{Cells}(D') = \emptyset$, then $V_r^{D'} = 0$.
2. If $\mathsf{Cells}(D') \neq \emptyset$ and $r = 0$, then $V_r^{D'} = \infty$.
3. Otherwise, let $\mathcal{P}$ be the set of all prefixes D'' of D' such that $\mathsf{Cells}(D' \setminus D'') \models \Gamma$ and $\delta(D'(c), v_r) \leq \tau$ for all $c \in \mathsf{Cells}(D' \setminus D'')$. Then:

$$V_r^{D'} = \min_{D'' \in \mathcal{P}} \left(V_{r-1}^{D''} + \sum_{c \in \mathsf{Cells}(D' \setminus D'')} w(\lambda(c)) \cdot \delta(D'(c), v_r) \right).$$

The first case refers to the situation where D' is empty; hence, the cost is zero. The second case refers to the situation where we have cells but no points to locate cells in (hence, there is no $(\delta \leq \tau)$-repair), so the cost is infinite. In the third case, we go over all possible prefixes D'' of D' with $\mathsf{Cells}(D' \setminus D'') \models \Gamma$. In this case, the repair E' is such that $E'(c) \in \{v_1, \ldots, v_{r-1}\}$ for all $c \in \mathsf{Cells}(D'')$, while $E'(c') = v_r$ for every cell $c' \in \mathsf{Cells}(D' \setminus D'')$. In this case, $V_{r-1}^{D''}$ is the minimal cost for D'', and to that we add the cost of changing the cells of $D' \setminus D''$ to v_r. Since $E'_{|D' \setminus D''}$ is contracted, by checking that $\mathsf{Cells}(D' \setminus D'') \models \Gamma$, we ensure that we do not violate consistency by placing all cells of $D' \setminus D''$ together. Then, $V_{r-1}^{D''}$ is responsible for checking that $E'_{|D''} \models \Gamma$. This guarantees that we obtain a repair. Furthermore, we only consider the prefixes D'' such that $\delta(D'(c), v_r) \leq \tau$ for all $c \in \mathsf{Cells}(D' \setminus D'')$, and $V_{r-1}^{D''}$ is responsible for checking that $\delta(D'(c), E'(c)) \leq \tau$ for all $c \in \mathsf{Cells}(D'')$, which guarantees that we obtain a $(\delta \leq \tau)$-repair.

The correctness of the above algorithm is a direct consequence of Lemma 19, and it is rather straightforward to show that its running time is polynomial in the size of the input.

The full line. When $M = \mathbb{R}$, we can use the following lemma, which gives a reduction to the finite case that we discussed in the previous part.

▶ **Lemma 20.** *Suppose that $M = \mathbb{R}$ and Γ is closed under addition. Let D be an inconsistent database. If any $(\delta \leq \tau)$-repair exists, then there is an optimal $(\delta \leq \tau)$-repair E such that* $\mathsf{Vals}(E) \subseteq (\mathsf{Vals}(D) \cup \{v \pm \tau \mid v \in \mathsf{Vals}(D)\})$.

Proof (Sketch). We show (in the archive [21]) that if the repair E uses a point outside the stated domain, then the entire set of cells in that point can be moved to a point in the domain without increasing the cost; this move is possible since Γ is closed under addition. ◀

5 Conclusions

We studied the problem of finding an optimal repair of an inconsistent database (i.e., a set of labeled cells) with respect to a coincidence constraint. We established that incorporating the metric space underlying the domain of values can lead to algorithms with efficiency and quality guarantees. In summary: the problem is APX-hard for general metrics but logarithmically approximable in polynomial time, and moreover the problem is solvable optimally in polynomial time for a tree metric (hence, for the common line and discrete metrics). We also discussed the case of an infinite metric and the addition of bound restrictions. The addition of the bound restrictions makes it NP-hard to test whether any legal repair exists, but an optimal repair can be found in polynomial time for the line metric.

Many directions are left for future work. First, for general metrics our lower bound is APX-hardness (i.e., some constant ratio) while the upper bound is logarithmic; hence, a gap remains. Next, as mentioned in Section 4, we left open the case of a tree metric with bound restrictions. Note, however, that even if this case is solved optimally in polynomial time, it is not at all clear that this tractability has implications on other metrics (e.g., via embedding) as it has in the absence of bound restrictions. Still, it is an important challenge to find natural metrics, beyond the line metric, where nontrivial upper bounds can be established.

Another direction for future work is to extend our work to constraints besides coincidence constraints, including others commonly studied for data quality management: functional dependencies (and their conditional enrichment [7]), denial constraints, non-unary inclusion constraints, and so on. Another direction is the extension to coincidence constraints with a non-fixed set of labels given as part of the input (i.e., the "combined complexity" variant of this problem), which requires a formalism for compactly expressing coincidence constraints, such as a set of inclusion (or key or foreign-key) constraints.

Finally, it is important to investigate the practical aspects of our work. How efficiently do the algorithms of this paper perform on common datasets? How practical is the dynamic program for the tree metric? How can we optimize it? What is the actual approximation ratio that takes place in a general metric? How does it change from one metric to another? How close is an (approximately) optimal repair to the correct database instance? These questions call for a careful implementation and experimental investigation as the next steps.

References

1 Ofer Arieli, Marc Denecker, and Maurice Bruynooghe. Distance semantics for database repair. *Ann. Math. Artif. Intell.*, 50(3-4):389–415, 2007. doi:10.1007/S10472-007-9074-1.

2 Ofer Arieli and Anna Zamansky. A graded approach to database repair by context-aware distance semantics. *Fuzzy Sets Syst.*, 298:4–21, 2016. doi:10.1016/J.FSS.2015.06.007.

3 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, pages 184–193. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548477.

4 Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

5 Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008. doi:10.1016/J.IS.2008.01.005.

6 Guy E. Blelloch, Yan Gu, and Yihan Sun. Efficient construction of probabilistic tree embeddings. In *ICALP*, volume 80 of *LIPIcs*, pages 26:1–26:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.26.

7 Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755. IEEE Computer Society, 2007. doi:10.1109/ICDE.2007.367920.

8 Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154. ACM, 2005. doi:10.1145/1066157.1066175.

9 Rajesh Bordawekar and Oded Shmueli. Using word embedding to enable semantic queries in relational databases. In *DEEM@SIGMOD*, pages 5:1–5:4. ACM, 2017. doi:10.1145/3076246.3076251.

10 Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *SIGMOD Conference*, pages 1335–1349. ACM, 2020. doi:10.1145/3318464.3389742.

11 Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. Relaxed functional dependencies - a survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016. doi:10.1109/TKDE.2015.2472010.

12 Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005. doi:10.1016/J.IC.2004.04.007.

13 Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE Computer Society, 2013. doi:10.1109/ICDE.2013.6544847.

14 Vlastislav Dohnal, Claudio Gennaro, and Pavel Zezula. A metric index for approximate text management. In *ISDB*, pages 37–42. Acta Press, 2002.

15 Vlastislav Dohnal, Claudio Gennaro, and Pavel Zezula. Similarity join in metric spaces using ed-index. In *DEXA*, volume 2736 of *Lecture Notes in Computer Science*, pages 484–493. Springer, 2003. doi:10.1007/978-3-540-45227-0_48.

16 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/J.JCSS.2004.04.011.

17 Wenfei Fan and Floris Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012. doi:10.2200/S00439ED1V01Y201207DTM030.

18 Amir Gilad, Aviram Imber, and Benny Kimelfeld. The consistency of probabilistic databases with independent cells. In *ICDT*, volume 255 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICDT.2023.22.

19 Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500. Morgan Kaufmann, 2001. URL: http://www.vldb.org/conf/2001/P491.pdf.

20 Miika Hannula and Jef Wijsen. A dichotomy in consistent query answering for primary keys and unary foreign keys. In *PODS*, pages 437–449. ACM, 2022. doi:10.1145/3517804.3524157.

21 Youri Kaminsky, Benny Kimelfeld, Ester Livshits, Felix Naumann, and David Wajc. Repairing databases over metric spaces with coincidence constraints. *CoRR*, abs/2409.16713, 2024. doi:10.48550/arXiv.2409.16713.

22 Youri Kaminsky, Eduardo H. M. Pena, and Felix Naumann. Discovering similarity inclusion dependencies. *Proc. ACM Manag. Data*, 1(1):75:1–75:24, 2023. doi:10.1145/3588929.

23 Solmaz Kolahi and Laks 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. doi:10.1145/1514894.1514901.

24 Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278. IEEE Computer Society, 2009. doi:10.1109/ICDE.2009.219.

25 Selasi Kwashie, Jixue Liu, Jiuyong Li, and Feiyue Ye. Efficient discovery of differential dependencies through association rules mining. In *ADC*, volume 9093 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2015. doi:10.1007/978-3-319-19548-3_1.

26 Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020. doi:10.1145/3360904.

27 Yasir Mahmood, Jonni Virtema, Timon Barlag, and Axel-Cyrille Ngonga Ngomo. Computing repairs under functional and inclusion dependencies via argumentation. In *FoIKS*, volume 14589 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2024. doi:10.1007/978-3-031-56940-1_2.

28 Dongjing Miao, Pengfei Zhang, Jianzhong Li, Ye Wang, and Zhipeng Cai. Approximation and inapproximability results on computing optimal repairs. *VLDB J.*, 32(1):173–197, 2023. doi:10.1007/S00778-022-00738-0.

29 Rajvardhan Patil, Sorio Boit, Venkat N. Gudivada, and Jagadeesh Nandigam. A survey of text representation and embedding techniques in NLP. *IEEE Access*, 11:36120–36146, 2023. doi:10.1109/ACCESS.2023.3266377.

30 Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A formal framework for probabilistic unclean databases. In *ICDT*, volume 127 of *LIPIcs*, pages 6:1–6:18, 2019. doi:10.4230/LIPICS.ICDT.2019.6.

31 Shaoxu Song and Lei Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16:1–16:41, 2011. doi:10.1145/2000824.2000826.

32 Shaoxu Song, Lei Chen, and Hong Cheng. Parameter-free determination of distance thresholds for metric distance constraints. In *ICDE*, pages 846–857. IEEE Computer Society, 2012. doi:10.1109/ICDE.2012.46.

33 Jan Tönshoff, Neta Friedman, Martin Grohe, and Benny Kimelfeld. Stable tuple embeddings for dynamic databases. In *ICDE*, pages 1286–1299. IEEE, 2023. doi:10.1109/ICDE55515.2023.00103.

34 Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String similarity search and join: a survey. *Frontiers Comput. Sci.*, 10(3):399–417, 2016. doi:10.1007/S11704-015-5900-5.