REGULAR PAPER

Eris: efficiently measuring discord in multidimensional sources

Alberto Abelló[1] · **James Cheney[2]**

Abstract

Data integration is a classical problem in databases, typically decomposed into schema matching, entity matching and data fusion. To solve the latter, it is mostly assumed that ground truth can be determined. However, in general, the data gathering processes in the different sources are imperfect and cannot provide an accurate merging of values. Thus, in the absence of ways to determine ground truth, it is important to at least quantify how far from being internally consistent a dataset is. Hence, we propose definitions of *concordant* data and define a *discordance* metric as a way of measuring disagreement to improve decision-making based on trustworthiness. We define the *discord measurement* problem of numerical attributes in which given a set of uncertain raw observations or aggregate results (such as case/hospitalization/death data relevant to COVID-19) and information on the alignment of different conceptualizations of the same reality (e.g., granularities or units), we wish to assess whether the different sources are concordant, or if not, use the discordance metric to quantify how discordant they are. We also define a set of algebraic operators to describe the alignments of different data sources with correctness guarantees, together with two alternative relational database implementations that reduce the problem to linear or quadratic programming. These are evaluated against both COVID-19 and synthetic data, and our experimental results show that discordance measurement can be performed efficiently in realistic situations.

1 Introduction

The focus of this work is decision-making environments performing complex OLAP-like multidimensional queries [2] that extensively use numerical aggregation and involve multiple data sources requiring integration. Data integration traditionally has three steps [14]: (1) schema matching and alignment, which overcomes semantic and structural heterogeneity between attributes and entities from different sources; (2) entity matching (a.k.a. record linkage), which detects records that correspond to the same real-world entity; and (3) data fusion (a.k.a. record merging), which aims to identify the correct one among conflicting values. Thus, data fusion refers to the combination of data from different,

heterogeneous sources in order to provide a more precise understanding of reality than offered by those sources separately [26]. The quality of its result is clearly affected by the disparity of the involved sources. For instance, in data warehousing environments, consistent and well- known data from different sources go through a well-structured cleaning and integration process. In the wild, sources are typically incomplete and not well aligned, and such data cleaning and integration processes are far from trivial, resulting in imperfect comparisons. Like in the parable of the blind men describing an elephant after touching different parts of its body (i.e., touching the trunk, it is like a thick snake; the leg, like a tree stump; the ear, like a sheath of leather; the tail tip, like a furry mouse; etc.), in many areas like epidemiology, social sensing or information extraction, different data sources reflect the same reality in slightly different and partial ways, and there is not any ground truth available, requiring truth discovery processes [30]. For example, during the COVID-19 pandemic, it was problematic to have reliable information on number of cases and deaths, since many different actors were independently gathering data that had later to be integrated to make them globally meaning-

✉ Alberto Abelló
aabello@essi.upc.edu

James Cheney
jcheney@inf.ed.ac.uk

[1] Universitat Politecnica de Catalunya, Jordi Girona, 1-3, 08034 Barcelona, Catalonia, Spain

[2] University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, Scotland, UK

ful. Evaluating the reliability of each source was crucial for decision-making, and we develop ERIS a tool that facilitates doing this.

In such a complex context, where estimating the source reliability and inferring true information is necessary, we require a tool to measure discrepancies for available data. Typically, consistency is used to measure to which degree a dataset is free of contradictions [26], which is done most of the time by simply counting differences in the sources [19], or maybe something more elaborate like using the Shapley value to weight primary key violations as in [31], or based on the number of necessary repairs like [7]. A more complete overview and classification of these kinds of metrics can be found in [38]. Nevertheless, we contend that better measures than counting exist for many scenarios in the case of coincidence of numerical attributes, whose distance can be precisely quantified. Hence, in this case, a binary metric aiming at full consistency does not look realistic and we require a more precise one showing how far values are from each other.

Problem. Thus, having in mind that we often analyze data by placing different indicators/features/measures (e.g., number of patients or deaths) in a multidimensional space (e.g., geography and time), the problem we approach in this work is the measurement of numerical disagreement between different sources. To solve this problem we have to face several difficulties: a) sources need to firstly go through a difficult and error prone alignment to make them comparable, (b) there can be many alternative metrics, and (c) given the amount of data in today's scenarios, these must be computed efficiently. Hence, we aim at defining a (a) **declarative**, (b) **flexible** and (c) **scalable** method to quantify discrepancies in the different numerical attributes.

Relational Database Management System (RDBMS) are scalable and provide a declarative query language, together with a flexible mechanism to deal with uncertain and incomplete information by using NULL values. However, they do not provide any guarantees on alignments and it is well known that NULLs are overloaded with different meanings such as unknown, nonexisting or no-information [3].

Approach. First, we restrict the use of NULL only for nonexisting or no-information and propose to enrich the data model with *symbolic variables* that allow to represent the partial knowledge we might have about uncertain numerical values, and integrate this in an RDBMS whose query results are processed in a solver to generate the desired metric. Our approach is (a) declarative in that it provides a high-level language (based on standard relational query languages) for expressing the intended alignments among sources. This high-level language can be translated down to linear or quadratic programming problems that can be solved efficiently. The translation is proved correct and users do not need to carry it out themselves or be concerned with the

low-level details of the encoding. It is (b) flexible because users can firstly use different distance metrics and also make changes to alignments (e.g., to accommodate changes to source data formats) using the high-level language rather than manually changing a low-level system of equations. It is (c) scalable in that despite the NP-completeness of general quadratic programming problems, our approach can find optimal solutions measuring the discord (i.e., distance away from being consistent) in a dataset quickly using off-the-shelf solvers. This is so, because we only allow linear expressions in the characterization of uncertainty of values and use convex functions in the discord measurement.

Contributions. In this paper, we consider problems we call *concordance checking* and *discordance measurement*. Concordance is the problem of determining whether disparate data sources we wish to integrate are consistent with each other according to some specification (of how they should be related). Discordance measurement is the problem of determining how close or distant the observed data are from being concordant. We define a flexible setting, that can be instantiated with different distance metrics (see [22] for alternatives), for the evaluation of the trustworthiness of different sources of multidimensional data based on their concordancy/discordancy using standard linear or quadratic programming solvers. Moreover, since besides errors and conflicts in data, different conceptualizations are also a problem [34], we define an algebra that allows to easily describe alignments between sources and guarantees the correctness of their symbolic evaluation. While using symbolic variables for NULLs is not a new idea, introduced for example in classical models for incomplete information such as c-tables and v-tables [27] and used more recently in data cleaning systems such as LLUNATIC [23], our approach generalizes unknowns to be arbitrary (linear) expressions.

To our knowledge, there is not any system that can automatically generate the measurement of discordance and even less in the presence of semantic heterogeneities between the sources. More concretely, in this paper, we contribute:

1. A definition and formalization of the problem of **discord measurement** of databases under some merging processes, independently of the concrete distance metric being used.
2. A set of algebraic operations to describe high-level **alignment specifications** that allow to describe merging processes of multidimensional tables with symbolic numerical expressions.
3. An automatic translation from such specifications to low-level linear or quadratic programs with accompanying proofs of correctness.
4. A novel *coalescing* operator that automatically generates **concordancy constraints** over symbolic tables, that

can be efficiently checked with off-the-shelf software, as exemplified in our prototype.

5. A prototype, ERIS,[1] and an experimental comparison of two alternative implementations using an RDBMS (PostgreSQL) and a quadratic programming solver (OSQP[43]). For the sake of prototyping, we assume data reside in a single DBMS, but the same approach applies to virtual data integration given the corresponding wrappers.

6. An analysis of discordances in the epidemiological surveillance systems of six European countries during the COVID-19 pandemic based on EuroStats and Johns Hopkins University (JHU) data.

Organization. Sect. 2 presents a motivational example that helps to identify the problem formally defined in Sect. 3, whose solution based on an algebraic query language for symbolic evaluation is presented in Sect. 4. Section 5 details two alternative relational implementations, which are evaluated in Sect. 6. Our experimental results show that both approaches provide acceptable performance and illustrate the value of our approach for assessing the discordance of COVID-19 epidemiological surveillance data at different times and countries between March 2020 and February 2021. The paper concludes with related work and conclusions in Sects. 7 and 8. A preliminary short paper (presenting the problem statement informally via examples and excluding the main technical content in Sects. 3–6) has been published in [1].

2 Motivating example

We used COVID-19 data in our experiments and examples, which are widely available and of varying quality, making them a good candidate for discordance evaluation. We consider that a network of actors (i.e., governmental institutions) take *primary* measurements of COVID-19 cases and *derive* some aggregates from those. In an idealized setting, we would expect to know all the relationships and have consistent measurements for each primary attribute, and each derived result would be computed exactly with no error. However, some relationships are unknown and both primary and derived attributes are noisy, biased, unknown or imperfect. We illustrate now how to model it using database schemas and views and describe the different problems we need to solve in this scenario.[2]

Example 1 *Spain*, as depicted in Fig. 1, comprises nineteen regions ($R_I, \ldots, R_{XIX}$). In each region, there are several hospitals and a person living in R_i is monitored by at most one

---

[1] Eris is the Greek goddess of discord.

[2] We assume some familiarity with relational model, queries, views, SQL, etc. [3] as well as with the multidimensional data model [2].
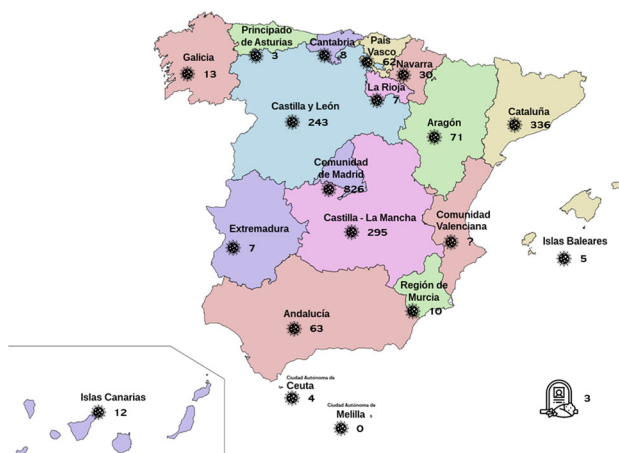

Fig. 1 *Spain*'s map [source: Wikipedia] with epidemiological information reported on week 2020W24 (total of 2,142 cases and three deaths in the whole country)

hospital. Hospitals report their number of cases of COVID-19 to their regional governments, and each regional government reports to the Ministry of Health (MoH).

Given their management autonomy, the different regions in *Spain* use different and imperfect monitoring mechanisms and report separately the COVID-19 cases they detect every week. Suppose that despite being gathered daily at health facilities, *Spain* is only reporting weekly to the European Centre for Disease Prevention and Control (CDC) partial information at the region level and the overall information of the country. We can model this using relational tables with the weekly region and country information, and try to use SQL to **measure discord** between them.

ReportedRegion(region, week, cases)
ReportedCountry(week, cases)   ◊

The first thing that must be done before measuring discrepancies is to overcome semantic and schematic heterogeneity. Thus, in terms of SQL, we can align the schemas through named queries (a.k.a. views).

Example 2 Before making any measurement, we need to align the two sources by describing the **merging process**. In this case, the following view aggregates the regional data for each week, which ought to coincide with the values per country:

```
CREATE VIEW AggReported AS
SELECT week, SUM(cases) AS cases
FROM ReportedRegion GROUP BY week;
```
◊

Once we know that quantities in the attributes are using the same units, scales, etc., and assuming that we already have properly identified the different entities, we can simply count coincidences in the attribute values.

Example 3 Ideally, if all COVID-19 cases were detected and properly reported, the week should unambiguously determine the number of cases (i.e., information derived from reported cases, both at region and country levels, must coincide). In terms of SQL, as in [19], this could be checked with the assertion of a **concordancy constraint** in the form of a simple query like the following.

```
SELECT count(*)
FROM ReportedCountry
NATURAL JOIN AggReported;                    ◇
```

Nevertheless, as explained above, achieving exact consistency seems unlikely in any real setting. Pure coincidence (or even string similarity metrics such as Levenshtein distance [46]) does not give an idea of the magnitude of discrepancies in numeric data. For example, in the case of European countries, and according to the real data used in the experiments in Sect. 6.2, we can see that the reported cases at country and region level only coincide for one country (DE) in week 24. If we use Levenshtein distance with a threshold of 20% of the overall number of digits, we get three more coincidences for UK and one more for IT (still none for ES, NL and SE). Thus, using existing techniques (i.e., assertions) it is possible to check full consistency (i.e., value coincidence) among data sources when there is no uncertainty, but it is not straightforward to quantify to which extent the various data sources are consistent with the expected relationships, in the presence of unknown values or suspected errors in reporting.

Example 4 We can see that the following database is not consistent with the view specification above, in part because the cases of one of the nineteen Spanish regions (i.e., R_{XIX}, which in reality corresponds to *Comunidad Valenciana*) are not declared, but also because the sum of cases per region (1,995) do not add up to the overall amount in the country (2,142). Thus, it is not enough to say that the database is inconsistent, but we can see that there is a discrepancy of $2142 - 1995 = 147$ cases.

```
ReportedRegion("I","2020W24",13)
...
ReportedRegion("XVIII","2020W24",12)
ReportedCountry("2020W24",2142)
AggReported("2020W24",1995)
```

First of all, it is important to realize that the simplistic approach of using NULL just worsens the problem, because replacing any value with it would only result in a loss of information. Instead, we can assign an error factor ϵ_i to every value v_i in the database and measure the average of squared difference from each number of weekly cases v_i to the midpoint m (a.k.a. average) so that $v_i \cdot (1 + \epsilon_i) = m$ with the following query. According to [22], one of the most common goodness of fit measures is least squares error.
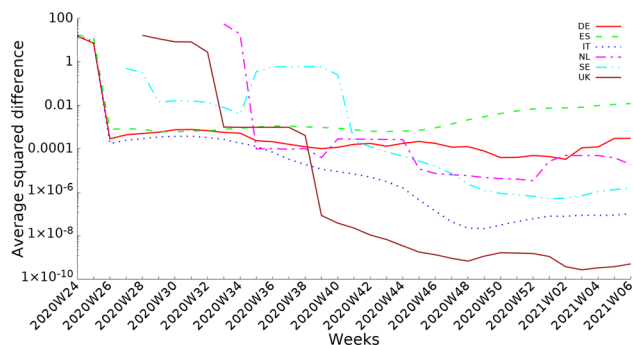

Fig. 2 Average squared error per week (against average)

```
SELECT week,
  (((r.cases+a.cases)/(2.0*r.cases)−1)^2
  +((r.cases+a.cases)/(2.0*a.cases)−1)^2)/2
FROM ReportedCountry r JOIN AggReported a
  ON r.week=a.week;
```

It is important to notice, firstly the dependence of the query on the distance being used and also how its complexity grows with the number of variables and their corresponding alignments. ◇

Example 5 We now consider the same *Spain* scenario, but using the real data of six European countries discussed in detail in Sect. 6.2. Taking as reference value simply the average of the two reported, we apply a similar query, obtain the value for all 39 weeks in our case study and get the line chart in Fig. 2, which shows the average discordance of each dataset along time in a running average of five weeks. We need to use logarithmic scale because the distance (measured as the average squared difference) between values can vary from one week to another in several orders of magnitude. ◇

Even though, as seen in the previous example, we can go beyond simply counting discrepancies (a.k.a. voting) with only SQL, we contend that we require some specific mechanism to properly and flexibly quantify discordancy between different sources. From the query generation point of view, it is easy to realize that having more than two sources, or simply considering potential error variables in all tuples (e.g., those nineteen actually reported at region level) substantially complicates the SQL code. Indeed, if we think of manually generating the formula for any potential alignment (overcoming any semantic heterogeneity) between multiple sources, it is clear that it is not only error prone, but simply unfeasible.[3]

---

[3] Sect. 6.2 presents, in the same COVID-19 case study, a more realistic and complex alignment of 35 algebraic operations on multiple sources (Fig. 13) showing all the potential of our discordance quantification technique. For this case, ERIS automatically generates, with all the correctness guarantees, two SQL queries of 145 and 225 operators in the PostgreSQL access plan, and a Python file of 8 538 characters.

Moreover, the mechanism should be flexible enough to facilitate changing the definition of distance (e.g., from relative to absolute values), or the function (e.g., from sum of squares to simple sum of values), or even assigning weights to the distances depending on their sources. To our knowledge, there is not any other tool that allows to do this.

3 Problem formulation

Given an *idealized* scenario (specified by its schema and views) and a collection of actual observations (both primary and derived), we can still consider two problems:

(A) *Value estimation*: Estimate the values of numerical attributes of interest (e.g., the number of COVID-19 cases across *Spain*) that make the system consistent, a special case of Truth Discovery [30].

(B) *Discord evaluation*: Evaluate how far is the actual, discordant dataset from an idealized concordant one.

Problem (A) is the well-studied statistical estimation problem through numerical fusion operators [11], so this can be very difficult, especially where the interrelationships among different data sources are complex (see [10] for a survey of existing systems). Instead, we consider problem (B), which is roughly analogous to computing the distance function used in truth discovery approaches [30], as an indication of source reliability. However, unlike conventional truth discovery, which considers homogeneous datasets from different sources that might have different reliability but follow a common format, we consider situations where there are heterogeneous data sources providing complementary views of the real phenomenon, but where the available data sources have nontrivial relationships. Traditional truth discovery seeks to identify consensus values that minimize the distance from these to the observations, whereas in our setting, we have a single, heterogeneous dataset and we want to measure how far it is away from being consistent with our expectations. That is, we wish to measure the distance from the observed data to the nearest self-consistent dataset (of which there may be infinitely many), not just a finite number of distances between homogeneous datasets.

Given a (probably incomplete but overlapping) set of instances, we assume only a merging process specification in the form of *expectations* about their alignment, expressed using database queries and views, and try to answer the following questions. Considering on the one hand the queries and views specifying the expected behavior, and on the other the data corresponding to observations of some of the inputs, intermediate results, or (expected) outputs are the observed numeric data complete and concordant considering the alignment specification? If there is missing data, can the existing datasets be extended to some complete instance that is concordant? Finally, how far from being fully consistent is the numerical data?

Consequently, we aim at extending DBMS functionalities for generic concordance evaluation as a way to quantify how far away the data are from being consistent. Although our goal in this paper is not to find a realistic estimate of the true values of unknown or uncertain data, but instead to quantify how close the data are to our expectations under the given alignment, we need to make some assumption on this. As in Example 4 and 5, taking the average of multiple points is always possible, but over-simplistic. Thus, we contend that using the value minimizing the errors of all sources, although more complex, is more principled (e.g., in our case, it gives a more comparable measure and avoids the need of using logarithmic scale, as in Fig. 2). It is important to clarify that while the approach produces estimates for the uncertain values as a side-effect, they may not have any statistical validity unless additional work is done to statistically characterize the sources of uncertainty, which we see as a separate problem.
Notation. We assume some familiarity with foundations of relational databases, as covered for example by textbooks like Abiteboul et al. [3]. We use the following notational conventions for tuples and relations: a tuple t is a finite map from some set of attribute names $\{A_1, \ldots, A_n\}$ to data values $d \in D$. We use letters such as K, U, V, and W to denote sets of attribute names and sometimes write U, V to indicate the union of disjoint attribute sets (i.e., $U \cup V$ when U and V are disjoint) or U, A to indicate addition of new attribute A to attribute set U (i.e., $U \cup \{A\}$ provided $A \notin U$). We also write $U \backslash V$ for the difference of attribute sets. Data values D include real numbers $\mathbb{R}$, and (as discussed below) value attributes are restricted to be real valued. The domain of a tuple $\mathrm{dom}(t)$ is the set of attribute names it maps to some value. We write $t.A$ for the value associated with A by t, and $t[U]$ for the tuple t restricted to domain U. We write $t.A := d$ to indicate the tuple obtained by mapping A to d and mapping all other attributes $B \in \mathrm{dom}(t)$ to $t.B$. Note that $\mathrm{dom}(t.A := d) = \mathrm{dom}(t) \cup \{A\}$ and this operation is defined even if A is not already mapped by t. Furthermore, if $V = \{B_1, \ldots, B_n\}$ is an attribute set and u is a tuple with domain $U \supseteq V$, then we write $t.V := u$ as an abbreviation for $(\cdots (t.B_1 := u.B_1).B_2 := u.B_2 \cdots).B_n := u.B_n$, that is for the result of (re)defining t to match u on the attributes from V. Finally, when the range of t, u happens to be $\mathbb{R}$, that is, t and u are real-valued vectors, we write $t + u$ for the vector sum, $\alpha \cdot t$ for scalar multiplication by $\alpha \in \mathbb{R}$, and when Z is a finite multiset of such real-valued vectors, we write $\sum Z$ for their vector sum.

Relational databases generally have schemas that describe the field names and types of each relation in the database, as well as integrity constraints such as key and foreign key constraints. Our approach assumes data adhering to a sim-

ple multidimensional data model; specifically this means we consider the fields of each relation to be split into two sets, *keys* which identify a particular row uniquely and may be either numeric or descriptive (e.g., geographical location, date) and *values* that must be numeric and give quantitative indicators associated with keys. Relations of this form are essentially partial finite maps from the keys to the values. We define schemas as follows to ensure that they have this form.

Definition 1 (Finite Map Signature) A finite map signature is a relational schema with a primary key annotation, which is written $K \triangleright V$, where K and V are disjoint attribute names. A relation matches such signature if its attributes are $K \cup V$, the key-fields K are elements of the data domain D, the value-fields V are real numbers $\mathbb{R}$, and it satisfies the functional dependency (FD) $K \rightarrow V$ (i.e., any two tuples in the relation that match on the fields in K also match on the fields in V), that is we write $R : K \triangleright V$ for what is conventionally written as $R(\underline{A_1, \ldots, A_m}, B_1, \ldots, B_m)$ when $K = A_1, \ldots, A_m$ and $V = B_1, \ldots, B_m$.

Definition 2 (Finite Map Schema) A *finite map schema* Σ is an assignment of relation names $R_1, \ldots, R_n$ to *finite map signatures*. A database instance I matches Σ if each relation R of I matches the corresponding signature $\Sigma(R)$. We write $Inst(\Sigma)$ for the set of all instances of a schema Σ.

Example 6 Thus, our example in page 3 contains tables

$ReportedRegion : \{region, week\} \triangleright \{cases\}$ and
$ReportedCountry : \{week\} \triangleright \{cases\}$.

We also introduce here (for later use) a new table

$DeclaredDeaths : \{week\} \triangleright \{deaths\}$

reporting the total number of deaths across the whole country, with the following data:

DeclaredDeaths("2020W24", 3) $\qquad \diamond$

Given that our goal is to define and measure the degree of discordance of different data sources with complementary multidimensional information (e.g., *ReportedRegion*, *ReportedCountry*, *DeclaredDeaths*) under a data fusion process, it is important to notice that discrepancies occur when they assign different values to the same key. Consequently, we will work in a setting where not only the source tables, but also query results (and view schemas) need to satisfy such finite map constraints, indicating that different coincident sources quantify features of the same object or event. We introduce a specialized query language and type system to maintain these constraints while dealing with uncertainty, which arises from completely unknown/missing

values, or reported measurements that have some unknown error.

Specifically, we define a high-level alignment definition language and a flexibly configurable metric (see [22] for alternatives) that can be efficiently computed with off-the-shelf software. Indeed, the key contribution of this paper is that both checking concordance and measuring discord can be done by augmenting the data model with *symbolic expressions*, and this in turn can be done consistently and efficiently in an RDBMS with the right set of algebraic operations guaranteeing correctness. We formalize this intuition next.

4 Proposed solution

In the following, we introduce the three mechanisms that constitute our technique, and how to use them together to tackle the problem at hand. Firstly, in Sect. 4.1, we define a variant of relational algebra for queries over tables that are finite maps. This guarantees that the result of any query is still a finite map. Then, in Sect. 4.2, the concept of *symbolic tables* representing uncertainty is defined and the effect of each operator over them formally established and exemplified, together with their correctness proof. Afterward, in Sect. 4.3, a new abstract operator (a.k.a. *fusion*) is introduced to establish the behaviour in finding coincident instances in the presence of an *alignment specification* of different sources and show how it can be implemented by reduction to linear or quadratic programming. Finally, in Sect. 4.4, using the previous toolset, we formally define and exemplify the concordance and discordance problems.

4.1 Restricted algebra

We consider a variation of relational algebra over finite maps, whose type system ensures that the finite map property is preserved in any query result.

$$c ::= A = B \mid A < B \mid \neg c \mid c \wedge c'$$
$$e ::= \alpha \in \mathbb{R} \mid A \mid e + e' \mid e - e' \mid e \cdot e' \mid e/e'$$
$$q ::= R \mid \sigma_c(q) \mid \hat{\pi}_W(q) \mid q \bowtie q' \mid q \uplus_B q' \mid q \backslash q'$$
$$\mid \rho_{B \mapsto B'}(q) \mid \varepsilon_{B:=e}(q) \mid \gamma_{K;V}(q)$$

Conditions c and expressions e are typical sublanguages containing Boolean combinations of (in)equalities among attributes, and real-valued arithmetic operations over attributes, respectively. Queries q are loosely based on the standard relational algebra with extensions for grouping/aggregation and expression evaluation. They include several standard forms such as relation names R, selections $\sigma_c(-)$, projections $\hat{\pi}_W(-)$, set difference $\backslash$, renaming $\rho_{B \mapsto B'}$ and joins ($\bowtie$), as well as discriminated union $\uplus_B$, expression evaluation $\varepsilon_{B:=e}(-)$, and aggregation $\gamma_{K;V}(q)$. Figure 3 defines

$$\frac{R : K \triangleright V \in \Sigma}{\Sigma \vdash R : K \triangleright V} \qquad \frac{\Sigma \vdash q : K \triangleright V \qquad K \vdash c : \mathbb{B}}{\Sigma \vdash \sigma_c(q) : K \triangleright V}$$

$$\frac{\Sigma \vdash q : K \triangleright V \qquad W \subseteq V}{\Sigma \vdash \hat{\pi}_W(q) : K \triangleright V \backslash W}$$

$$\frac{\Sigma \vdash q_1 : K_1 \triangleright V_1 \qquad \Sigma \vdash q_2 : K_2 \triangleright V_2 \qquad V_1 \cap V_2 = \emptyset}{\Sigma \vdash q_1 \bowtie q_2 : K_1 \cup K_2 \triangleright V_1, V_2}$$

$$\frac{\Sigma \vdash q : K \triangleright V \qquad \Sigma \vdash q' : K \triangleright V}{\Sigma \vdash q \uplus_B q' : K, B \triangleright V}$$

$$\frac{\Sigma \vdash q : K \triangleright V}{\Sigma \vdash \rho_{A \mapsto B}(q) : K[A \mapsto B] \triangleright V[A \mapsto B]}$$

$$\frac{\Sigma \vdash q : K \triangleright V \qquad \Sigma \vdash q' : K \triangleright \emptyset}{\Sigma \vdash q \backslash q' : K \triangleright V}$$

$$\frac{\Sigma \vdash q : K \triangleright V \qquad K, V \vdash e : \mathbb{R}}{\Sigma \vdash \varepsilon_{A=e}(q) : K \triangleright V, A}$$

$$\frac{\Sigma \vdash q : K \triangleright V \qquad K' \subseteq K \qquad V' \subseteq V}{\Sigma \vdash \gamma_{K';V'}(q) : K' \triangleright V'}$$

Fig. 3 Well-formed queries

the well-formedness relation for queries. The rules in Fig. 3 define the relations $\Sigma \vdash q : K \triangleright V$ inductively as the least relation satisfying the rules, where each rule is interpreted as an implication "if the hypotheses (shown above the line) hold then the conclusion (shown below the line) holds." For more background on type systems, inference rules and inductive reasoning about them, see a standard textbook such as Pierce [39]. The judgment $\Sigma \vdash q : K \triangleright V$ states that in schema Σ, query q is well-formed and has type $K \triangleright V$. Intuitively, this means that if q is run on an instance of Σ, then it produces a result relation that is a finite map $R : K \triangleright V$. We make use of additional judgments for well-formedness of selection conditions c ($U \vdash c : \mathbb{B}$) and expressions e ($U \vdash e : \mathbb{R}$) which are standard and omitted. Later on, in the next section, a type system is defined that specification queries are required to adhere to, which is illustrated in Fig. 5, page 11.

– Selection $\sigma_c(R)$ behaves as in relational algebra; the selection criterion c is evaluated on each row in the input table and those rows satisfying c are retained in the output, while the rest are discarded. The type system restricts selection criteria to consider only predicates over key values that evaluate to boolean $\mathbb{B} = \{true, false\}$.
– Projection-away $\hat{\pi}_U(R)$ projects the fields in U away (that is, removes them from the rows of its argument). The type system only allows projection away of value-fields, that is, requires $U \subseteq V$; this ensures the results of these operations are still finite maps.
– Join $R \bowtie S$ takes two finite maps, whose key-fields may overlap, and returns tuples formed by fusing all pairs

of tuples whose common fields have the same values. Unlike in general relational algebra, the arguments to a join may only have overlapping key-fields and not shared value-fields.

– Discriminated union $R \uplus_B S$ combines two finite maps, adding a new field B whose value will differentiate the tuples coming from the first or, respectively, second argument. We do not allow arbitrary unions because the union of two finite maps is not a finite map if the domains overlap.
– Difference $R \backslash S$ removes the keys in S from R, where S has no value-fields (i.e., is just a set of keys).
– Renaming $\rho_{B \mapsto B'}(R)$ that changes the name of field B to B'.
– Derivation $\varepsilon_{B:=e}(R)$ performs arithmetic calculations by adding a new value-field B which is initialized by evaluating expression e using the field values in each row.
– Grouping/aggregation $\gamma_{K;V}(R)$ performs grouping on key-fields K and aggregation by summing the value-fields V. The constraint that grouping can only be performed on key-fields and aggregation on value-fields ensures that the results are still finite maps.

Example 7 We now illustrate the query language above on the running example scenario introduced in page 3. Thus, we can get the sum of all cases reported by region in a given week using our query language as $\gamma_{week;cases}(ReportedRegion)$. As discussed earlier, this results in a sum of 1,995 cases (not the 2,142 one would expect). We also expect that the number of deaths is 0.015 times the number of reported cases, which would be written as $\varepsilon_{deaths=0.015*cases}(ReportedCountry)$, but again does not coincide with the three declared deaths. $\Diamond$

Some of these restrictions help ensure that the query operations preserve the finite map property described by the typing rules. Others, though not necessary for this purpose, are nevertheless helpful later when we generalize the queries to evaluate over symbolic values. In particular forbidding selections or joins that involve comparing value-fields will help avoid the need for some of the complexities encountered in c-tables [27] or work on provenance for aggregate queries [4]. These restrictions have not posed problems in part because, especially in the multidimensional model, it is usually not necessary or desirable to perform exact comparisons on (continuous-valued) value-fields when describing how different sources are aligned. Instead, we often do want to express that different sources should be close together, but we can do this by introducing symbolic variables that represent unknown errors distorting the true value and imposing equational constraints using fusion and coalescing operators, as explained later.

To allow for better understanding of how well or badly the data conforms to our expectations, expressed using queries,

we next consider symbolic evaluation of queries over tables in which some values can be variables (or more generally, expressions).

4.2 Symbolic evaluation

The basic idea is to represent *unknown* real values with variables $x \in X$. Variables can occur multiple times in a table, or in different tables, representing the same unknown value, and more generally unknown values can be represented by (linear) expressions. However, key values $d \in D$ used in key-fields are required to be known. This reflects the assumption that the source database is *partially closed* [20], that is, we assume the existence of master data for the keys (i.e., all potential keys are coincident and known).

Definition 3 (Symbolic Expression) Let X be some fixed set of variables. A *symbolic expression e over X* is a real-valued expression in $\mathbb{R}[X]$, the set of polynomials in $\mathbb{R}$ with variables from X. We normally consider only linear expressions (e.g., $a_0 + a_1 x_1 + \cdots + a_n x_n$).

Definition 4 (Symbolic Table) A *symbolic table*, or *s-table (over X) R : K ▷ V* is a table (with the name prepended with $\mathbb{S}$) in which attributes from K are mapped to discrete non-null values in D (as before) and value attributes in V are mapped to symbolic expressions in $\mathbb{R}[X]$.

We define the *domain* of an s-table $dom(R)$ to be the set of values of its key attributes. We say that an s-table is *linear* if each value attribute is a linear expression in variables X.

An s-table is *ground* if all of the value entries are actually constants, i.e., if it is an ordinary database table (though still satisfying the functional dependency $K \rightarrow V$). The restrictions we have placed on symbolic tables are sufficient to ensure that symbolic evaluation is correct with respect to evaluation over ground tables, as we explain below.

We now clarify how real-world uncertain data (e.g., containing NULLs for unknown values instead of variables, or containing values that might be wrong) can be mapped to s-tables.

Suppose we are given an ordinary database instance $I \in Inst(\Sigma)$, which may have missing values (a.k.a., NULLs) and uncertain values (i.e., reported values which we do not believe to be exactly correct). To use the s-table framework, we need to translate such instances to s-instances I' that represent the set of possible complete database instances that match observed data I. In doing this and to allow for the possibility that some reported values present in the data might be inaccurate, we replace such values with symbolic expressions containing variables from X. We restrict ourselves to linear expressions for the sake of efficiency, but still our approach allows to do this in many ways, with different justifications

based on the application domain, and consequently different quantifications of discordance that should be interpreted accordingly. To exemplify it, in this paper, we replace uncertain values v with $v \cdot (1 + x)$ (or simply x if $v = 0$) where x is an error variable. The reason for doing this and not simply $v + x$ is that the weight we associate in our experiments with an error variable x is x^2, so the cost of errors is scaled to some extent by the magnitude of the value (e.g., it should be easier to turn 1,000,000 into 2,000,000 than 1 into 10). On the other hand, a natural way to handle missing values in s-tables is to replace each one in the original relational table with a distinct variable with no associated cost. In particular scenarios, we might instead prefer to replace each NULL with an expression $c \cdot (1 + x)$ where c is an educated guess as to the likely value, but here we consider only the simple approach of a NULL mapped to a variable. In general, we can assign to attributes any linear expression, with any number of variables and reuse these variables in any number of attributes or tables.

Example 8 It is easy to see that, in our example on page 3, there are many possibilities of assigning cases of COVID-19 to the different regions of *Spain* that add up to 2,142 in the studied week and consequently improve the consistency of our database, which may be easily represented by replacing constants by symbolic expressions "$v_i(1 + x_i)$," where v_i is the corresponding value and x_i is an error parameter representing that cases may be missed or overreported in every region. The cases for region R_{XIX}, that were not reported at all, could then be simply represented by a variable x_{XIX}. Nevertheless, this may not completely explain the mismatch between cases reported at the country and region levels, and there might also be some doubly counted or hidden cases in *Spain* (for example, in the *Ciudad Autonoma de Melilla*, which this week declared not to have any cases), which we represent by variable $(1 + y)$. On the other hand, we can also consider census data and try to attribute all the excess deaths to COVID-19, which clearly involves some imprecision, too. So we should apply some error term $(1 + z)$ to the declared number of deaths coming from the census, as well. Therefore, s-tables $\mathbb{S}ReportedRegion : \{region, week\} ▷ \{cases\}$, $\mathbb{S}ReportedCountry : \{week\} ▷ \{cases\}$ and $\mathbb{S}Declared\text{-}Deaths : \{week\} ▷ \{deaths\}$ would contain:

$\mathbb{S}ReportedRegion("I", "2020W24", 13 * (1 + x_I))$
$\cdots$
$\mathbb{S}ReportedRegion("XVIII", "2020W24", 12 * (1 + x_{XVIII}))$
$\mathbb{S}ReportedRegion("XIX", "2020W24", x_{XIX})$
$\mathbb{S}ReportedCountry("2020W24", 2142 * (1 + y))$
$\mathbb{S}DeclaredDeaths("2020W24", 3 * (1 + z))$ ◊

We now make the semantics of our query operations over s-tables precise in Fig. 4. An essential property of this semantics is that (for both ground and symbolic tables) the typing rules ensure that a well-formed query evaluated on a valid

$$\sigma_c(R) = \{t \mid t \in R, c(t) = true\}$$
$$\hat{\pi}_W(R) = \{t[K, V \setminus W] \mid t \in R\}$$
$$R \bowtie S = \{t[K_R \cup K_S, V_R, V_S]$$
$$\mid t[K_R, V_R] \in R, t[K_S, V_S] \in S\}$$
$$R \uplus_D S = \{t.D := 0 \mid t \in R\} \cup \{t.D := 1 \mid t \in S\}$$
$$R \setminus S = \{t \mid t \in R, t[K] \notin S\}$$
$$\rho_{B \mapsto B'}(R) = \{t[K \setminus B, V \setminus B].B' := t.B \mid t \in R\}$$
$$\varepsilon_{B:=e}(R) = \{t.B := e(t) \mid t \in R\}$$
$$\gamma_{K';V'}(R) = \{t[K'].V' := sum(R, t, K', V') \mid t \in R\}$$
$$sum(R, t, K, V) = \sum(t'[V] \mid t' \in R, t[K] = t'[K])$$

Fig. 4 Symbolic Evaluation

instance of the input schema yields a valid result table, preserving the desired properties, that is, ensuring that the resulting tables are valid s-tables, and moreover ensuring that the semantics of query operations applied to s-tables is consistent with their behavior on ground tables. Moreover, symbolic evaluation preserves linearity, which is critical for ensuring that the constrained optimization problems arising from symbolic evaluation fit standard frameworks and can be efficiently solved.

The following paragraphs describe and motivate the behavior of each operator and informally explain and justify the correctness of the well-formedness rules ensuring that the result of (symbolic) evaluation is a valid (symbolic) table.

– Selection ($\sigma_c(R) : K \triangleright V$ where $R : K \triangleright V$). We permit $\sigma_c(R)$ when c is a Boolean formula referring only to fields $A, B, \ldots \in K$. If comparisons involving symbolic values were allowed, then the existence of some rows in the output could depend on unknown variable values, so would not be representable just using s-tables.
– Projection-away ($\hat{\pi}_W(R) : K \triangleright V \setminus W$ where $R : K \triangleright V$ and $W \subseteq V$). The projection operator projects-away only value-fields. Discarding key-fields could break the finite map property by leaving behind tuples with the same keys and different values.
– Join ($R \bowtie S : K_1 \cup K_2 \triangleright V_1 \cup V_2$ where $R : K_1 \triangleright V_1$, $S : K_2 \triangleright V_2$ and $V_1 \cap V_2 = \emptyset$). Joins can only overlap on key-fields, for the same reason that selection predicates can only select on keys: if we allowed joins on value-fields, then the result of a join would not be representable as an s-table.
– Discriminated union ($R \uplus_D S : K, B \triangleright V$ where $R : K \triangleright V$ and $S : K \triangleright V$). The union of two finite maps may not satisfy the functional dependency from keys to values. We instead provide a discriminated union that tags the tuples in R and S with a new key-field B to distinguish the origin.
– Renaming ($\rho_{B \mapsto B'}(R) : K[B \mapsto B'] \triangleright V[B \mapsto B']$ where $R : K \triangleright V$). Note that since K and V are disjoint, the renaming applies to either a key-field or a value-field,

but not both. In any case, this clearly preserves the finite map property.
– Difference ($R \setminus S$ where $R : K \triangleright V$ and $S : K \triangleright \emptyset$). The difference of two maps discards from R all tuples whose key-fields are present in S. The result is a subset of R hence still a valid finite map. We assume S has no value components; if not, this can be arranged by projecting them away in advance.
– Derivation ($\varepsilon_{B:=e}(R) : K \triangleright V$, B where $R : K \triangleright V$ and e is a linear expression over value-fields V). No new keys are introduced so the finite map property still holds.
– Aggregation ($\gamma_{K';V'}(R)$ where $R : K \triangleright V$ and $K' \subseteq K$ and $V' \subseteq V$). We allow grouping on key-fields and aggregation of value-fields (possibly discarding some of each). We consider SUM as the only primitive form of aggregation; COUNT and AVERAGE can be easily defined from it.

Example 9 Given $\mathⓈ{Reported Region} : \{region, week\} \triangleright \{cases\}$ and $\mathⓈ{ReportedCountry} : \{week\} \triangleright \{cases\}$, we can define views:

$$\mathⓈ{AggReported} := \gamma_{\{week\};\{cases\}}(\mathⓈ{ReportedRegion})$$
$$\mathⓈ{InferredDeaths} := \hat{\pi}_{cases}($$
$$\varepsilon_{deaths:=0.015*cases}(\mathⓈ{ReportedCountry}))$$

The first one corresponds to the SQL in Example 2, while the second assumes an average case-fatality ratio (CFR) of 1.5% to estimate the number of deaths based on the number of reported COVID-19 cases in the country. Notice the projection is necessary, because the expression evaluation operation adds a new field, so we must get rid of the *cases* for the resulting table's signature to match that of *DeclaredDeaths*. Regarding CFR, we use a single value for simplicity, but it could be declared in an auxiliary table containing different ones per week, as long as these do not contain any variable, which would break the linearity of the expression. ◇

The above discussion gives a high-level argument that if the input tables are finite maps (satisfying their respective functional dependencies as specified by the schema) then the result table will also be a finite map that satisfies its specified functional dependency. More importantly, linearity is preserved: if the s-table inputs to an operation are linear, and all expressions in the operation itself are linear, then the resulting s-table is also linear.

Correctness We interpret s-tables as mappings from valuations to ground tables, obtained by evaluating all symbolic expressions in them with respect to a global valuation $h : \mathbb{R}^X$.

Definition 5 (Valuation) A *valuation* is a function $h : \mathbb{R}^X$ assigning constant values to variables. Given a symbolic expression e, we write $h(e)$ for the result of evaluating e with variables x replaced with $h(x)$. We then write $h(t)$

for the tuple obtained by replacing each symbolic expression e in t with $h(e)$ and write $h(R)$ to indicate the result of evaluating the expressions in R to their values according to h, that is, $h(R) = \{h(t) \mid t \in R\}$. Likewise for an instance I, we write $h(I)$ for the ground instance obtained by replacing each R with $h(R)$. An s-table R *represents* the set $(\!|R|\!) = \{h(R) \mid h : \mathbb{R}^X\}$ of ground tables obtained by applying all possible h to R. We write $(\!|I|\!)$ for the set of all ground instances obtainable from an s-instance I by some $h \in \mathbb{R}^X$, defined as $(\!|I|\!) = \{h(I) \mid h \in \mathbb{R}^X\}$, and we write $q(\!|I|\!)$ for $\{q(I') \mid I' \in (\!|I|\!)\}$, the set of all possible results of evaluating q on a ground table represented by I.

Given a query expression in our algebra, we can evaluate it on a ground instance since every ground instance is an s-instance and s-table operations do not introduce variables that were not present in the input. Further, given a set of ground instances, we can (conceptually) evaluate a query on each element of the set.

Theorem 1 *Let q be a well-formed query mapping instances of Σ to relations matching $K \triangleright V$, and let I be an s-instance of Σ. Then $q(\!|I|\!) = (\!|q(I)|\!)$.*

The proof is in Appendix A and is similar to correctness proofs for c-tables [27]; the main complication is that in s-instances, variables occurring in different tables are intended to refer to the same unknown values, whereas in c-tables such variables are scoped only at the table level.

4.3 Fusion, alignment, and coalescing

We now consider how s-tables and symbolic evaluation can be used to reduce concordance checking and discordance measurement to linear programming and quadratic programming problems, respectively, when we find more than one tuple with the same key and multiple (symbolic) values for the same attribute. We first consider an abstract *fusion* operator:

Definition 6 (Fusion) Given two ground relations $R, S : K \triangleright V$, their *fusion* $R \sqcup S$ is defined as $R \cup S$, provided it satisfies the functional dependency $K \to V$, otherwise the fusion is undefined.

Thus, our goal is to fuse different sources. However, since they are independent, their concrete values can come in a variety of formats, being expressed in different units, scales or even computation stages (e.g., benefit vs income and expenses). As explained in [33], until the conflicts at the representation level have been resolved, those at the data level cannot be resolved (or even measured), either. Therefore, we represent the expected relationships between source and derived data using a generalization of view specifications called *alignment specifications*. The alignments must always

$$\overline{\Sigma \vdash \cdot : \cdot}$$

$$\cdots \quad \frac{\Sigma \vdash q_1 : K \triangleright V}{\Sigma \vdash q_n : K \triangleright V \quad \Sigma, S : K \triangleright V \vdash \Delta : \Omega} \over \Sigma \vdash S := q_1 \sqcup \cdots \sqcup q_n, \Delta : (S{:}K \triangleright V, \Omega)$$

Fig. 5 Well-formed alignment specifications $[\Sigma, \Omega, \Delta]$

be defined by the user, considering the domain knowledge, but our set of algebraic operators guarantees the correctness from a computational point of view (i.e., identity of tuples is preserved and expressions are guaranteed to be linear). We generalize fusion to many sources and actually allow alignment specifications to define derived tables as the fusion of multiple views.

Definition 7 (Alignment Specification) Let Σ and Ω be finite map schemas with disjoint table names $R_1, \ldots, R_n$ and $T_1, \ldots, T_m$, respectively. Let Δ be a sequence of view definitions, one for each T_i, of the form $T_i := q_1 \sqcup \cdots \sqcup q_k$, where each q_i is a query over finite maps, that refers only to table names in Σ and $T_1, \ldots, T_{i-1}$. The triple $Spec = [\Sigma, \Omega, \Delta]$ is called an *alignment specification*.

Alignment specifications are considered well-formed according to the rules in Fig. 5. The first rule handles the base case where the Δ and Ω parts of the specification are empty. The second rule says that a specification $[\Sigma, (S : K \triangleright V, \Omega), (S := q_1 \sqcup \cdots \sqcup q_n, \Delta)]$ is well-formed provided each q_i is a query producing an output matching $K \triangleright V$ and provided the rest of Δ is well-formed with respect to Ω if the type of S is added to Σ. The purpose of adding S to Σ here is to ensure later view definitions may refer both to the tables initially in Σ and to earlier view definitions, but view definitions cannot be cyclic (for example S cannot refer to itself or to a view defined later).

Example 10 Given the s-tables in Example 8 and views in Example 9, we can specify the alignment of COVID-19 cases (corresponding to Example 3) and deaths by the following views:

$$\text{\textcircled{s}}SumOfCases := \text{\textcircled{s}}ReportedCountry \sqcup \text{\textcircled{s}}AggReported$$
$$\text{\textcircled{s}}NumberOfDeaths := \text{\textcircled{s}}DeclaredDeaths \sqcup \text{\textcircled{s}}InferredDeaths$$

These view definitions express our intention that in concordant data, the country-level reported data would correspond to the sum of the region-level reports and the deaths according to the census data would be 1.5% of reported cases, as shown in the equations in Example 7. However, importantly, the fusion operator is not restricted to two arguments, it can express simultaneous coincidence among multiple inputs. ◊

We implement the abstract fusion operation on s-tables by first making the discriminated union of the input relations $(R \uplus_B S)$ and then using a unary operation, called *coalescing*,

whose behavior on sets $\mathcal{R}$ of ground tables $R : K, B \triangleright V$ is $\kappa_B(\mathcal{R}) = \{\hat{\pi}_B(R) \in \mathcal{R} \mid \hat{\pi}_B(R) \text{ satisfies } K \rightarrow V\}$. Intuitively, coalescing of a set of tables $R \in \mathcal{R}$ applies a projection $\hat{\pi}_B$ to each R and returns those projected tables that still satisfy the FD $K \rightarrow V$. These are the rows where the values associated with the corresponding keys are consistent across all inputs in which the keys are present. To represent the result of coalescing using s-tables, we augment them with constraints. A constraint ϕ is simply a conjunction of linear equations; a constrained s-table is a pair (R, ϕ) that represents the set of possible ground tables $\langle\!\langle R, \phi \rangle\!\rangle = \{h(R) \mid h : \mathbb{R}^X, h \models \phi\}$; finally a constrained s-instance (I, ϕ) likewise represents a set of ground instances of I obtained by valuations satisfying ϕ. We can implement coalescing as an operation on constrained s-tables as follows:

$$\kappa_D(R, \phi) = (T, \phi \wedge \psi)$$
$$R^+ = \{t \in R \mid \exists t' \in R, t[K \backslash D] = t'[K \backslash D] \wedge t[D] \neq t'[D]\}$$
$$T = \{t[K \backslash D, B_1 := L_{t[K \backslash D], B_1}, \dots] \mid t \in R^+\}$$
$$\cup \{t[K \backslash D, V] \mid t \in R \backslash R^+\}$$
$$\psi \equiv \bigwedge_{t \in R^+} \bigwedge_{B_i \in V} L_{t[K \backslash D], B_i} = t[B_i]$$

That is, let R^+ be the set of tuples in R for which there exists another tuple that has the same values on $K \backslash D$ but differs on D, and let $R \backslash R^+$ be the remaining tuples (for which there are no such sibling tuples). Thus, R^+ is the set of tuples of R potentially violating the FD $K \backslash D \rightarrow V$, and $R \backslash R^+$ is the largest subset of R that satisfies this FD. Then $\kappa_D(R)$ consists of table T obtained by filling in new variables $L_{t[K \backslash D], B_i}$. L-values are used only where there may be disagreement, and we use the value from R otherwise. The constraint ψ consists of equations between the observed values of each attribute and the corresponding L-value. No constraints are introduced for tuples in $R \backslash R^+$, where there is no possibility of disagreement.

Example 11 Thus, $\circledS Sum Of Cases$ is implemented in terms of our algebra as

$$\kappa_D(\circledS ReportedCountry \uplus_D \circledS AggReported),$$

and $\circledS Number Of Deaths$ as

$$\kappa_D(\circledS DeclaredDeaths \uplus_D \circledS InferredDeaths).$$

As a result, we introduce two new variables (i.e., respectively $L_{2020W24, cases}$ and $L_{2020W24, deaths}$) and would obtain the following constrained s-tables:

$\circledS$SumOfCases("2020W24", $L_{2020W24, cases}$)

$$\psi_{Sum Of Cases} \equiv L_{2020W24, cases} = 2142(1 + y)$$
$$\wedge\ L_{2020W24, cases} = 13(1 + x_{\mathrm{I}}) + \dots$$
$$+ 12(1 + x_{\mathrm{XVIII}}) + x_{\mathrm{XIX}}$$

$\circledS$NumberOfDeaths("2020W24", $L_{2020W24, deaths}$)

$$\psi_{Number Of Deaths} \equiv L_{2020W24, deaths} = 0.015 * 1995(1 + y)$$
$$\wedge\ L_{2020W24, deaths} = 3 * (1 + z)$$

Notice that if another source had provided data at region level, two possible alignments would appear: (a) aggregating also the new source (as done before with $\circledS ReportedRegion$) and making the correspondence also through the same existing fusion $\circledS Sum Of Cases$ (i.e., $L_{2020W24, cases}$ would be simply reused with another conjunct clause in ψ), or (b) creating a new independent fusion of this new source and $\circledS ReportedRegion$ which would generate thirteen new L-values (one per region) and the corresponding thirteen logic clauses to be added to ψ. Obviously, (b) is more restrictive than (a), but the choice would depend on the user defining the most appropriate alignment to the use case.

$\diamond$

4.4 Discord measurement

Having introduced (constrained) s-tables, and evaluation for query operations and coalescing over them, we finally show how these technical devices allow us to define concord and measure discord.

Definition 8 (Concordant Instance) Given a specification $Spec = [\Sigma, \Omega, \Delta]$, an instance I of schema Σ is *concordant* if there exists an instance J of Ω such that for each view definition $T_i := q_1 \sqcup \dots \sqcup q_n$ in Δ, we have $J(T_i) = q_1(I, J) \sqcup \dots \sqcup q_n(I, J)$ where $q(I, J)$ is the result of evaluating q on the combined instance I, J and all of the fusion operations involved are defined. The concordant instances of Σ with respect to $Spec$ are written $Conc(Spec)$.

Definition 9 (Concordance) Given $[\Sigma, \Omega, \Delta]$, let I be an s-instance. We say I is *concordant* if there exists a concordant instance $C \in \langle\!\langle I \rangle\!\rangle$.

Given an alignment specification $Spec = [\Sigma, \Omega, \Delta]$ and an s-instance I, we can check concordance by symbolically evaluating Δ on I to obtain an s-instance J as follows. For each view definition $T_i := q_1 \sqcup \dots \sqcup q_n$ in Δ in order, evaluate $q_1, \dots, q_n$ to their s-table results and fuse them using the coalescing operator (repeatedly if $n > 2$). This produces a new s-table T_i' and a constraint ϕ_i. Add $T_i := T_i'$ to J and continue until all of the table definitions in Δ have been symbolically evaluated (i.e., $J = [T_1 := T_1', \dots, T_m := T_m']$). Thus, the constrained s-instance (I', ϕ) where $\phi = \bigwedge_{i=1}^m \phi_i$ characterizes the set of possible concordant instances based on I', and in particular I is concordant if ϕ is satisfiable.

Example 12 From the constrained s-tables in Example 11, obtained by the corresponding coalescing operation, we get the next intertwined system of equations:

$$2142(1 + y) = 13(1 + x_{\mathrm{I}}) + \dots + 12(1 + x_{\mathrm{XVIII}}) + x_{\mathrm{XIX}}$$

$$0.015 * 1995(1 + y) = 3 * (1 + z)$$

Obviously, this system has many solutions. One solution S_1 consists of taking all x_i to be zero, $y = -0.07$ and $z = 8.975$. This corresponds to assuming there is no error in the eighteen regions' reports and there are no cases in Region XIX. Another solution S_2 sets $x_I = ... = x_{XVIII} = 0$ and $x_{XIX} = 147$, then $y = 0$ and $z = 9.71$ which corresponds to assuming R_{XIX} had all of the missing COVID-19 cases in *Spain* that week. Of course, whether S_1 or S_2 (or some other) is more plausible depends on domain-specific knowledge. $\Diamond$

Given a *cost function* δ assigning a cost to each solution, we can compare different solutions in terms of how much correction is needed (or discord exists). Thus, the discord is, intuitively, the shortest δ-distance between the actual observed, uncertain data (represented as a set of possible instances) and a hypothetical concordant database instance that is consistent with the alignment specification. The more distant from any such concordant instance, the more discordant our data are.

Definition 10 (Discordance) Given $Spec = [\Sigma, \Omega, \Delta]$, let $\delta : Inst(\Sigma) \times Inst(\Sigma) \to \mathbb{R}$ be a measure of distance between pairs of elements of $Inst(\Sigma)$. The *discordance* of a (constrained) s-instance I is

$$\inf_{J \in (\![I]\!), C \in Conc(Spec)} \delta(J, C)$$

Then, the degree of discordance of I given the alignment Δ and according to δ (i.e., $disc_\delta(I, \phi)$) equals the solution to the quadratic programming problem formed by minimizing δ subject to ϕ. Depending on the choices of metrics, this leads to well-understood constrained optimization problems such as linear programming or least squares optimization. Linear programming has polynomial time complexity, and the popular simplex algorithm is worst-case exponential but has good behavior in practice [42]. Quadratic programming is NP-Complete in general, but with good computational behavior in many practical applications [43]. Moreover in the *convex* case when the cost function is based on a *positive semidefinite* matrix, quadratic programming is very well behaved, having a single global solution that can be found in polynomial time [45]. In our experiments, we used the convex metric δ defined as the sum of squares of the error variables in I.

Like the alignment, the cost function evaluating the discordance is also strongly domain knowledge dependent, but our system is flexible enough to consider different alternatives (e.g., different weights; if the weights are non-negative then the problem remains convex). Nevertheless, in the rest of this paper, we will only use simplistic (and convex) cost functions like the sum of squares suggested in [30]. This guarantees that

the problems we generate are solvable in polynomial time; we also establish feasibility in practice for datasets of moderate size. Although the underlying quadratic solver we use can accommodate more general quadratic functions of the symbolic variables, we leave exploration of more sophisticated cost functions to future work.

Example 13 Considering simply the sum of the squares of the variables:

$$c_1(\vec{x}, y, z) = \left(\sum_{i \in \{I, ..., XIX\}} x_i^2 \right) + y^2 + z^2$$

For the two solutions in Example 12, S_1 has cost ≈ 80.56, while S_2 has cost 21,703.28, so with the above cost function the first one is much closer to being concordant, because a large change to x_{XIX} is not needed. Alternatively, we might give the unknown number of cases in R_{XIX} no weight, reflecting that we have no knowledge about what it might be, corresponding to the cost function

$$c_2(\vec{x}, y, z) = \left(\sum_{i \in \{I, ..., XVIII\}} x_i^2 \right) + y^2 + z^2$$

that assigns the same cost to S_1 but assigns cost 94.28 to S_2, indicating that if we are free to assign all unaccounted cases to x_{XIX} then the second solution is closer to concordance.

Furthermore, we could also weight variables considering the reliability of the different regions as well as the central government, and the historical information of the census, but it is important to notice that such weights would depend on knowledge of the domain. $\Diamond$

It is important to mention also that as a side-product, the instanciations of the L-values introduced by coalescing could be used to obtain a concordant instance, but this instance is not guaranteed to provide the true values of the uncertain indicators, it is only an estimate minimizing the distance metric. Thus domain-specific knowledge or statistical techniques also need to be applied to characterize the quality of these estimates.

5 Implementation

We now describe the techniques employed in ERIS, an implementation of our approach. The systems of equations resulting from constraints generated on coalescing tables or instances are linear, so they can be solved using linear algebra solvers. However, it may not be immediately obvious how to evaluate queries over s-tables to obtain the resulting systems of equations efficiently. One strategy would simply be

Non-First Normal Form (NF^2)

ReportedRegion			
dist.	week	cases	deaths
I	2020W24	$\{\{13, x_I\}, 13\}$	. . .
. . .	. . .		. . .
XVII	2020W24	$\{\{1, x_{XVII}\}, 0\}$	. . .
XVIII	2020W24	$\{\{12, x_{XVIII}\}, 12\}$	. . .
XIX	2020W24	$\{\{1, x_{XIX}\}, 0\}$	. . .

Normalized (Partitioning)

ReportedRegion			
dist.	week	cases	deaths
I	2020W24	13	. . .
. . .	. . .	. . .	. . .
XVII	2020W24	0	. . .
XVIII	2020W24	12	. . .
XIX	2020W24	0	. . .

ReportedRegion_cases			
dist.	week	var.	coeff.
I	2020W24	x_I	13
. . .	. . .	. . .	. . .
XVII	2020W24	x_{XVII}	1
XVIII	2020W24	x_{XVIII}	12
XIX	2020W24	x_{XIX}	1

ReportedRegion_deaths			
dist.	week	var.	coeff.
. . .	. . .	. . .	. . .

ReportedRegion			
dist.	week	cases	deaths
I	2020W24	$13x_I + 13$	. . .
II	2020W24	$3x_{II} + 3$	. . .
III	2020W24	$8x_{III} + 8$	. . .
IV	2020W24	$62x_{IV} + 62$	. . .
V	2020W24	$30x_V + 30$	. . .
VI	2020W24	$7x_{VI} + 7$	. . .
VII	2020W24	$71x_{VII} + 71$	. . .
VIII	2020W24	$826x_{VIII} + 826$	. . .
IX	2020W24	$243x_{IX} + 243$	. . .
X	2020W24	$295x_X + 295$	. . .
XI	2020W24	$7x_{XI} + 7$	. . .
XII	2020W24	$336x_{XII} + 336$	. . .
XIII	2020W24	$5x_{XIII} + 5$	. . .
XIV	2020W24	$63x_{XIV} + 63$	. . .
XV	2020W24	$10x_{XV} + 10$	. . .
XVI	2020W24	$4x_{XVI} + 4$	. . .
XVII	2020W24	x_{XVII}	. . .
XVIII	2020W24	$12x_{XVIII} + 12$	. . .

$\Longrightarrow$

Fig. 6 Implemented encodings of s-tables

to load all of the data from the database into main memory and evaluate the s-table query operations in-memory. While straightforward, this may result in poor performance or duplication of effort, as many query operations which are efficiently executed within the database (such as joins) may need to be hand-coded in order to obtain acceptable performance. Instead, we propose a relational representation of s-table queries such that the s-table operations can be implemented as ordinary (extended) relational algebra queries over the representation. Thus, we consider two in-database representations of s-tables, illustrated in Fig. 6: A denormalized sparse vector representation using nested user-defined data types (NF^2) and a normalized representation using multiple flat relations (Partitioning). Hence, whichever representation we choose, we need to transform the original ground tables into s-tables with variables. This could be done by creating simply a copy, but we found this would materialize a great deal of intermediate data that is not ultimately needed. Instead, the most efficient approach we found is defining the s-tables as *views* over the sources. These views are straightforward, except for the generation of the variables (using SQL:1999 features such as ROW_NUMBER to create unique ids for each of them), and the consideration of special cases (i.e., NULL and zero), that is done by means of standard SQL CASE clauses to distinguish them. Notice that this approach based on views and CASE clauses avoids the manual definition of expressions for every value and allows the definitions of general rules based on the table name, attribute name, or even concrete attribute values to do it. For example, a rule could indicate the usage of a given expression for some concrete region or change the expression from a given point in time on.

In the NF^2 approach, we add a user-defined type for the symbolic (linear) expression fields. There are several ways of doing this, like for example using arrays to represent vectors of coefficients for a (small) fixed number of variables, or using a sparse representation that can deal with arbitrary numbers of variables efficiently when most coefficients are zero. Having experimented with several options, we chose a representation in which symbolic expressions $\sum_i a_i \cdot x_i + b$ are represented as sparse vectors, specifically as a pair of the value of b and an array of pairs $[(a_i, x_i), \ldots]$ of coefficients and variable names. Thus, $\circledS ReportedRegion$ NF^2 implementation would correspond to the following SQL view.

```
CREATE VIEW ReportedRegion_NF2 AS
SELECT region, week,
 ROW(ARRAY[ROW(
  CASE
   WHEN cases IS NULL OR cases = 0 THEN 1
   ELSE cases
  END, (
  CASE
   WHEN cases IS NULL THEN 'MarkNULL_'
   ELSE ''
  END||'ReportedRegion_cases_')||row_number()
   OVER (ORDER BY region, week))::term],
  CASE
   WHEN cases IS NULL THEN 0.0
   ELSE cases
  END)::sparsevec AS cases
 FROM ReportedRegion;
```

The keys remain unchanged, and for the value we define a row of type sparsevec, which is actually an array of terms, represented by pairs "(coefficient,variable)." The coefficient

Fig. 7 Translation of queries to partitioning implementation

$$\sigma_P(R_0, \vec{R}) = (\sigma_P(R_0), \sigma_P(\vec{R}))$$
$$\hat{\pi}_W(R_0, \vec{R}) = (\hat{\pi}_W(R_0), \vec{R}[V \backslash W])$$
$$\rho_{[B \mapsto B']}(R_0, \vec{R}) = (\rho_{[B \mapsto B']}(R_0), \vec{R}[B \mapsto B']))$$
$$(R_0, \vec{R}) \uplus_D (S_0, \vec{S}) = (R_0 \uplus_D S_0, (B := R.B \uplus_D S.B)_{B \in V})$$
$$\varepsilon_{B:=c}(R_0, \vec{R}) = (\varepsilon_{B:=c}(R_0), (\vec{R}, B := \emptyset))$$
$$\varepsilon_{B:=B_i+B_j}(R_0, \vec{R}) = (\varepsilon_{B:=B_i+B_j}(R_0), (\vec{R}, B := \tilde{\gamma}_{K,X;C}(R.B_i \uplus_D R.B_j)))$$
$$\varepsilon_{B:=\alpha \cdot B_i}(R_0, \vec{R}) = (\varepsilon_{B:=\alpha \cdot B_i}(R_0), (\vec{R}, B := \tilde{\varepsilon}_{C:=\alpha \cdot C}(R_{B_i})))$$
$$(R_0, \vec{R}) \backslash (S_0, ()) = (R_0 \bowtie (\pi_K(R_0) \backslash S_0), (B = R.B \bowtie \pi_K(R_0) \backslash S_0)_{B \in V})$$
$$(R_0, \vec{R}) \bowtie (S_0, \vec{S}) = (R_0 \bowtie S_0, (B := R.B_R \bowtie (\pi_{K_S}(S_0)), B' := S.B_S \bowtie (\pi_{K_R}(R_0)))_{B_R \in V_R, B_S \in V_S})$$
$$\gamma_{K';V'}(R_0, \vec{R}) = (\gamma_{K';V'}(R_0), (B := \tilde{\gamma}_{K',X;C}(R.B))_{B \in V'})$$

is "1" if the reported value was either NULL or zero, or the reported value otherwise. Regarding the variable, we create a different one for every tuple using the "row_number," and for the sake of traceability we concatenate to it both the table and attribute names. Moreover, we also add a special mark in case the reported value was NULL, so this can be considered in the cost function. We implemented addition, scalar multiplication, and aggregation (sum) of linear expressions using PostgreSQL's user-defined function facility. With this encoding, the SQL queries corresponding to our algebra are straightforward by applying the standard translation and inserting user-defined functions. We therefore do not present this translation in detail.

Though many RDBMSs support similar mechanisms to create user-defined functions and aggregates, they are not standard and so NF^2 is not very portable. Thus, the alternative approach we present, Partitioning, relies only on standard SQL:1999. In this approach, we represent an s-table $R : K \triangleright V$ with n symbolic value-fields $B_1, \ldots, B_n$ using $n+1$ relational tables, as follows:

– $R_0 : K \triangleright V$ is a ground table with all constant terms.
– For each symbolic field $B_i \in V$, $R_{B_i} : K, X \triangleright C$ is a ground table mapping keys K and an additional key X (corresponding to the variables) to a real value-field C, so that $(k, x, c) \in R_{B_i}$ when c is the coefficient of x in the B_i-attribute of $R(k)$.

We consider the relations corresponding to the symbolic value-fields to be collected in a record $\vec{R} = (B_1 = R_{B_1}, \ldots)$, and we write $(R_0, \vec{R})$ for the full representation. This representation admits relatively simple translations of each of the s-table query operations, as shown in Fig. 7.

The operations $\tilde{\gamma}$ and $\tilde{\varepsilon}$ are zero-filtering versions of aggregation and derivation, respectively, which remove any rows whose C-value is zero. Filtering out zero coefficients is not essential but avoids retaining unnecessary records since absent coefficients are assumed to be zero. In the rule for selection, recall predicate P only mentions key attributes; we write $\sigma_P(\vec{R})$ for the result of applying the selection to each table in $\vec{R}$. In the rule for projection-away, we assume $R : K \triangleright V$ and $W \subseteq V$, and $\vec{R}[V \backslash W]$ is the record resulting from discarding the fields corresponding to attributes

in W. Likewise in the rule for renaming, $\vec{R}[B \mapsto B']$ stands for renaming field B of $\vec{R}$ to B' if present, otherwise applying $\rho_{[B \mapsto B']}(-)$ to each table in $\vec{R}$. In the rule for addition, we introduce a dummy discriminant in the union and just use zero-filtering aggregation $\tilde{\gamma}_{K,X;C}$ to sum coefficients grouped by key and variable name (i.e., getting rid of the dummy discriminant). Likewise, in the case for scalar multiplication, $\tilde{\varepsilon}_{C:=\alpha \cdot C}(R_{B_0})$ operation does an in-place update and finally filters out any zero coefficients. Note that for the sake of understanding, we provide separate rules for assigning a field a constant value, adding two fields, and scalar multiplication of a field, while the query language given earlier allows derivations to assign a new field the result of any linear expression. Arbitrary linear expressions can be handled by introducing intermediate fields and projecting them away at the end, for example $\varepsilon_{B:=C+42}(q) = \hat{\pi}_D(\varepsilon_{B:=C+E}(\varepsilon_{E:=42}(q)))$. The rule for difference is slightly tricky because since S_0 does not have value attributes in the key, so just subtracting it from each of the $R.B$ would not work. Instead, we compute the set of keys present after the difference and restrict each $R.B$ to that set of keys (using a join). The rule for join is likewise a little more involved: given $R : K_R \triangleright V_R$ and $S : K_S \triangleright V_S$, since V_R and V_S are disjoint, it suffices for each field B_R of V_R to join the corresponding table $R.B_R$ with the keys of S_0, i.e., $\pi_{K_S}(S_0)$, and symmetrically for S's value-fields B_S. Finally, for aggregation we assume $R : K \triangleright V$ with $K' \subseteq K$ and $V' \subseteq V$ and again use $\tilde{\gamma}$.

Finally, we comment on constraint generation performed by the coalescing operator. It simply detects repeated values after projecting out the discriminant and generates the corresponding constraints as an extra query (i.e., a query over an s-table is actually a pair of queries: one retrieving the data I' without repetitions in the key and fresh L-values in the values and another independent query creating the constraints ϕ over those L-values). With this approach, coalescing (hence fusion) becomes first-class and can be freely composed with the other operations, so we can convert a specification into a single composed query (referring to the views that define the s-tables) whose translation generates the equations directly. This is the approach we have evaluated, which significantly improves the naive materialization
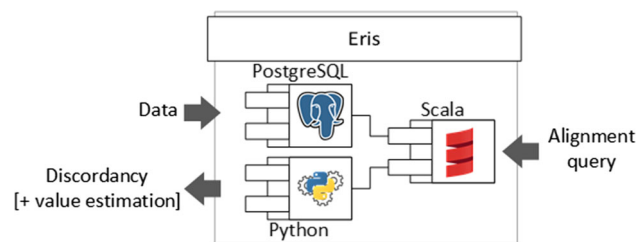
Fig. 8 Eris architecture

approach, because it avoids the need to load and scan numerous intermediate s-tables.

6 Evaluation

ERIS (whose components are depicted in Fig. 8) was implemented[4] in approximately 4000 lines of Scala code, with approximately 100 lines of SQL defining auxiliary operations, user-defined types, and functions involving sparse vectors.

Before launching anything, all the user data need to be uploaded into regular PostgreSQL tables. Then, on choosing the preferred representation of s-tables (either NF^2 or Partitioning), the corresponding views are created to virtually generate the variables. Once this is done, the input of the system is any alignment specification expressed in our algebra. Our Scala code transforms such a specification into regular SQL that returns the requested data from user ground tables. As soon as some of the s-tables are coalesced and some potential violations of the corresponding FDs appear, L-values are automatically created, and this triggers another SQL query for the generation of the constraints and the specific Python code to find the values of the involved variables from the retrieved information. The corresponding linear and quadratic programming subproblems are solved using version 0.6.1 of OSQP [43], called as a Python library with the default configuration and no parameter tuning.

Since, to our knowledge, there is not any other system that can automatically generate a (configurable) measurement of discordance in the presence of semantic heterogeneities between the sources, we cannot make any meaningful comparison to show that this is faster or can do things that the others can not. Instead, we show its scalability in terms of query performance, and the expressive power and usefulness by means of a use case. We do not try to justify the goodness of the sum of squares as an indicator of discordance, because this is absolutely configurable in ERIS, hence, justifying its use is out of the scope of this work.

Experiments were run on a workstation equipped with an Intel Xeon E5-1650 with 6 cores, 32 GB RAM, running Ubuntu 16.10, and using a standard installation of PostgreSQL 9.5. They evaluate ERIS from the perspective of both performance and usefulness.

6.1 Performance microbenchmarks

We considered the following questions:

Q1. How does the time taken for symbolic query evaluation using NF^2 and Partitioning vary depending on data size?
Q2. How does the time taken for equation generation vary depending on data size?
Q3. How does the time taken by OSQP for solving compare to that needed for equation generation?
Q4. How does overall time taken vary depending on the number of variables?
Q5. How does the measured discordance vary depending on the amount of distortion in the data?

Q1 and Q2 measure the performance of our system without considering the time taken by OSQP. Q3 determines whether our system produces QP problems that are feasible for OSQP to solve, because such problems could be encoded in several different ways. Q4 assesses whether and how performance depends on the amount of source data being symbolic, while Q5 investigates how discordance behaves when data that we know to be consistent is distorted to different degrees.

Although there are several benchmarks for entity resolution and evaluation of the distance between descriptive data, there is not any available benchmark with multiple sources of overlapping numerical data suitable for our system, so we adopted a microbenchmarking approach with synthetic data and simple queries. We defined a simple schema with tables $R : A, B \triangleright C, D$ and $S : B \triangleright E, F$ and a random data generator that populates this schema, for a given parameter n, by generating n rows for S and for each such row (b, e, f), generating between 0 and $\sqrt{n}$ rows for R with the same B field. Thus on average the resulting database contains $n + \frac{n}{2}\sqrt{n}$ rows in total. We generated databases for $n \in \{100; 1,000; 10,000\}$; note that $n = 10,000$ actually corresponds to approximately 510,000 rows. For each n, we performed five trials using five different randomly generated datasets and took the median running time (or for Q5, median distortion) over these five runs. We consider the following queries to exercise the most complex cases of the translation of Sect. 5:
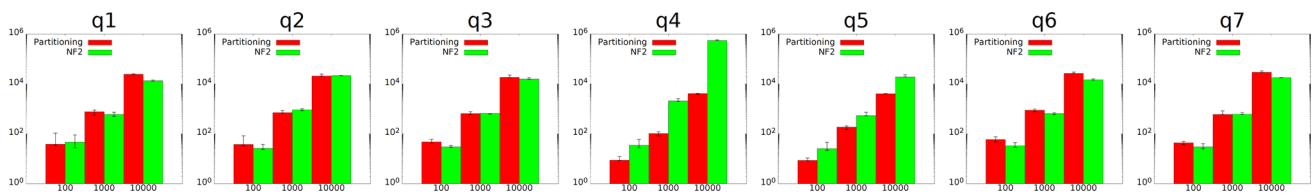
---

[4] https://github.com/dtim-upc/Eris.

Fig. 9 Query evaluation performance (milliseconds)

(a) Equation generation performance (milliseconds)
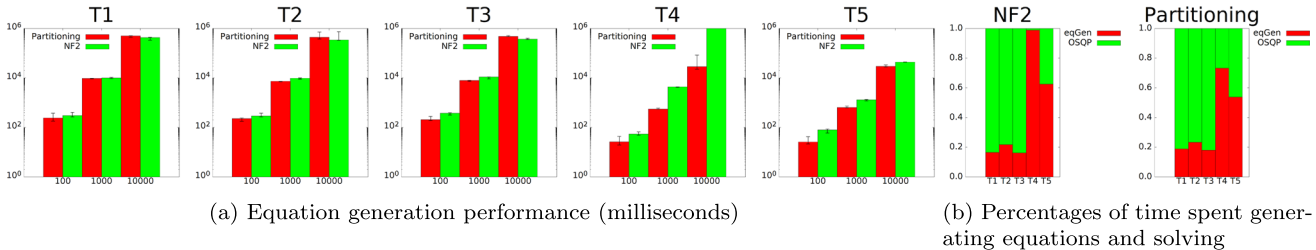
(b) Percentages of time spent generating equations and solving

Fig. 10 Equation generation and solving

$q_1 = R' \bowtie S'$

$q_2 = \varepsilon_{W:=C+D}(R')$ $\quad T_1 = \kappa_Z(q_1 \uplus_Z (R \bowtie S))$

$q_3 = \varepsilon_{X:=W*C}(\varepsilon_{W:=1}(R'))$ $\quad T_2 = \kappa_Z(q_2 \uplus_Z \varepsilon_{W:=C+D}(R))$

$q_4 = \gamma_{A;C}(R')$ $\quad T_3 = \kappa_Z(q_3 \uplus_Z \varepsilon_{X:=W*1}(\varepsilon_{W:=1}(R)))$

$q_5 = \gamma_{B;C}(R')$ $\quad T_4 = \kappa_Z(q_4 \uplus_Z \gamma_{A;C}(R))$

$q_6 = R \uplus_Z R'$ $\quad T_5 = \kappa_Z(q_5 \uplus_Z \gamma_{B;C}(R))$

$q_7 = \kappa_Z(q_6)$

Given two source tables R, S, in a database generated as explained above, we create *observation tables* R_o, S_o by distorting them as follows: For each row, we randomly replace each value-field with NULL with some probability (i.e., $p = 0.01$) and otherwise add a normally distributed distortion. Next, symbolic views R', S' of both distorted tables are defined, as outlined in Sect. 4.2. Once we have these two versions of the tables (i.e., the source R, S and the distorted, symbolic one R', S'), we considered two modes of execution of these queries: in the first mode ($q_1 \dots q_7$), we simply evaluate the query over a symbolic input (i.e., R' and S') and construct the result; in the second mode ($T_1 \dots T_5$), we evaluate the result of aligning the distorted query result with the result over the original source tables (i.e., R and S). Thus, for example, for T_1 we generate the equations resulting from the fusion expression $q_1 \sqcup (R \bowtie S)$, actually implemented like $\kappa_Z(q_1 \uplus_Z (R \bowtie S))$. Finally, the resulting system of equations is solved, subject to the metric giving each error variable x a weight of x^2 and each null variable a weight of 0.

For Q1, executions are summarized in Fig. 9, where reported times include the time to receive the symbolic query results. These show that the Partitioning and NF^2 have broadly similar performance; despite NF^2's comparative simplicity, its running time is often faster with the exceptions being q_4 and q_5, the two aggregation queries. Particularly for q_4, aggregation can result in large symbolic expressions which are not always handled efficiently by the NF^2 sparse vector operations using PostgreSQL arrays; we experimented

with several alternative approaches to try to improve performance without success. Thus, in cases where the symbolic expressions do not grow large, NF^2 seems preferable.

For Q2 and Q3, we measured the time taken for equation generation and for OSQP solving for each query, using different database sizes as described above. The results are shown in Fig. 10. In Fig. 10a, the time taken for equation generation, including querying and serializing the resulting OSQP problem instances, is shown (again in logarithmic scale). The OSQP solving times for Partitioning and NF^2 are coincident and so not shown. In Fig. 10b, the percentage of time spent on equation generation and on OSQP solving for the largest database instance ($n = 10,000$) is shown, and we can appreciate that they are always in a similar order of magnitude so neither can be claimed to be a bottleneck in front of the other.

For Q4, we considered a fixed database size (n=1000) and modified the data generation process and specifications so that for each input table, each row was treated as symbolic with some probability p_{sym}. We considered $p_{sym} \in \{0.01, 0.25, 0.5, 0.75, 0.99\}$. Only the values in these symbolic rows were augmented with variables and only these rows were distorted. We reran the evaluation for Q2 and Q3 to compute the total time in each case, for both encodings, in order to assess how the performance varies as the number of variables/symbolic fields in the input increases. Figure 11 shows the results, in each case reporting the median time observed out of five runs. For both Partitioning and NF^2 strategies, the total time increases roughly linearly. We further inspected the results for equation generation and solving time and found that generally the solving times for problems generated by Partitioning and NF^2 were close to each other, thus the difference in performance (especially in the case of T_4) is mostly due to difference in query evaluation times for equation generation, in line with the general trends noticed in
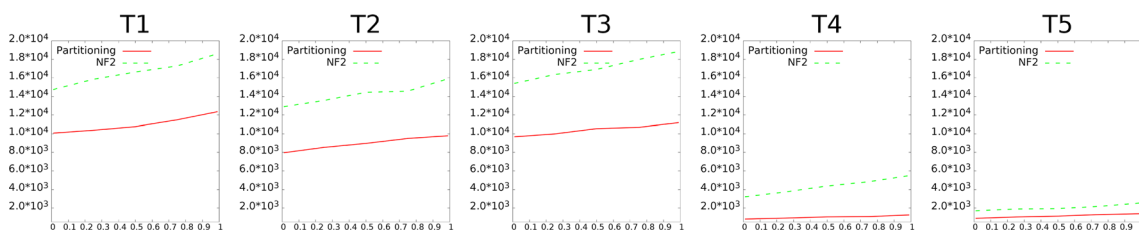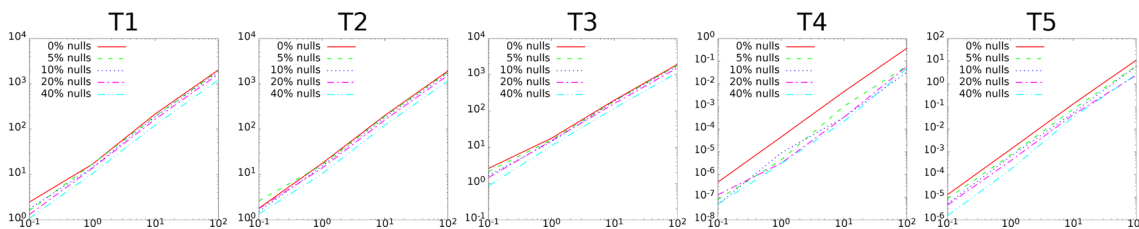
Fig. 11 Number of variables evaluation impact

Fig. 12 Distortion evaluation impact

Fig. 10a. Thus, the scalability of the approach is not compromised by our addition to the model, but follows the expected behaviour of regular ground queries (without variables).

For Q5 we again considered a fixed database size (n=1000) and separately varied the probability of replacing a value with null (p_{null}) and the standard deviation of the normally distributed noise (σ). We would expect increasing the number of NULLs to decrease the discordance (all else being equal) because null variables carry no weight, while increasing the standard deviation of the distortion should increase the discordance. We considered $p_{null} \in \{0, 0.05, 0.1, 0.2, 0.4\}$ and $\sigma \in \{0.1, 1, 10, 100\}$. For each combination of parameters, we evaluated five randomly generated inputs and computed the distance found by OSQP, taking the median discordance in each case. The results are shown in Fig. 12. We report only once the results obtained, because the discordance value found does not depend on the implementation strategy. These results confirm that increasing the amount of distortion (σ) generally increases the discordance, while increasing the number of NULLs (p_{null}) tends to decrease discordance (because it introduces degrees of freedom to the problem that do not incur any penalty in the cost metric).

6.2 Case study

We might use our tool to get a best-fit database. However, this would only be useful if sources are close to each other (and hence to reality). If they are relatively discordant (like the blind men describing the elephant), all we can aim at is to measure and study the evolution of such discordancy. Thus, we applied our prototype to the study of challenging COVID-19 data, which is publicly available, and see from that the improvement of reporting in different countries during the

pandemic. More specifically, we considered two different sources:

– Johns Hopkins University (JHU) The Center for Systems Science and Engineering (CSSE) at JHU was gathering COVID-19 data since the very beginning of the pandemic and became a referent worldwide [17]. On the one hand, we have used its daily time series at country level[5] containing both cases and deaths. Unfortunately, on the other hand, regional data are scattered in different files in the JHU repository, so we used a more compact version.[6]
– EuroStats As second data source for comparison, we used the weekly European mortality by EuroStats,[7] following the Nomenclature of Territorial Units for Statistics (NUTS).[8]

JHU was going through a continuous consolidation and cleaning process, but still resulted in quite poor quality. Obviously, EuroStats data are of much higher quality and more reliable. Indeed, the weekly mortality per country appears to be historically quite stable (less than 5.5% coefficient of variation for the six countries of our study). Hence, we took the weekly mortality of the five years previous to the pandemic as ground truth. However, for some countries, most recent figures were either tagged as provisional or estimated. While we considered the former to be an administrative issue and still part of the error-free ground truth, we put the latter

---

[5] https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series

[6] https://github.com/coviddata/coviddata

[7] https://ec.europa.eu/eurostat/databrowser/view/demo_r_mwk2_ts/default/table

[8] https://ec.europa.eu/eurostat/web/nuts/background

Table 1 Summary of the tables in the experiment

Table	Loc.	Times	*Rows*	First	Last
EU(r,w,#d)	222	1043	152, 938	2000W01	2019W52
EUe(r,w,#d)	222	73	15, 001	2000W01	2021W20
EU(c,w,#d)	33	1043	26, 177	2000W01	2019W52
EUe(c,w,#d)	34	1116	4125	2000W01	2021W20
JHU(c,d,#c,#d)	197	479	94, 363	20200119	20210521
JHU(r,d,#c,#d)	550	385	211, 365	20200129	20210216

together with the mortality of 2020/2021 in an *s-table* and treated those data in the same way as the ones coming from JHU.

We loaded the different data in a PostgreSQL database with Pentaho Data Integration. These were divided in the six tables shown in Table 1, together with the counters of different locations and times, number of rows, and first and last time point available. Data were split firstly according to the source (namely EuroStats or JHU). Ground truth mortality (i.e., until the end of 2019 and free of errors) is in *ground* tables *EU*, while estimates and data of 2020/2021 are in *s-tables EUe*. Different *s-tables* are also generated for different geographic granularities (namely region *r* or country *c*), and relevantly, data from Eurostats are available per week *w*, while data from JHU are available daily *d*. Both location and temporal dimensions result in different (underlined) key attributes for the corresponding tables. From EuroStats, we only used the number of deaths #*d*, while from JHU we took both COVID-19 cases #*c* and deaths #*d*. Attribute #*d* is declared as free of variables in both *EU ground* tables and its instances are consequently constants. Values coming from EuroStats correspond exactly to the reported ones, but to mitigate the noise (e.g., cases not reported during weekends being moved to the next week by some regions) in those coming from JHU, we followed the common practice of taking the average in the previous seven days for both cases and deaths.

Figure 13 shows a logical representation of our alignment of the sources. Notational elements are introduced to facilitate the understanding, like "avg" instead of the "sum/count" actually used in the current prototype. Dimensional tables like *date* and *firstadminunit* and their corresponding joins to facilitate selections over *year* and week of year (*woy*), or the relationships between countries and regions, are omitted for the sake of simplicity. This alignment reflects the knowledge about the behaviour of COVID-19 pandemic, but other alternative alignments could have been easily explored with ERIS. On the first hand, we take *EU* and *EUe* tables and generate the weekly surplus of deaths after the sixth week of 2020 by subtracting from the declared amounts, the average deaths in the last five years for the same week. This is done both per region and country, since these values are not

always concordant (even if coming from the same source). Then, regional results are aggregated per country and merged in the same table with the information provided already at that level using a discriminated union to keep track of the different origins. On the other hand, looking now at JHU tables, we aggregate regional data in three different ways: deaths per country and day, also deaths per region and week, and finally cases per region and week with a lag of three weeks (we will empirically justify this concrete value later). Under the assumption of case-fatality ratio of 1.5% (observed median on June 22nd, 2021 is 1.7% according to JHU[9]), such transformation is applied to the cases before merging and coalescing the weekly regional cases and deaths. Daily deaths reported per country and those obtained after aggregating regions are also coalesced and then aggregated per week. Both branches of JHU data are finally merged with a discriminated union into a single table. Finally, the four branches (namely EuroStats regional data, EuroStats country data, JHU regional data aligning cases and deaths, and JHU regional data coalesced with JHU country data) are merged into a single table with a discriminated union and finally coalesced to generate the overall set of equations.

We restricted our analysis to only the six countries in Table 2, chosen because of their relevance in the pandemic and availability of regional data in both EuroStats and JHU. Regarding the time, we only considered until February 2021, to avoid the impact of vaccination. As previously explained, to avoid the cost of errors is scaled to some extent by the magnitude of the value, we replaced uncertain values *v* in the ground tables with $v \cdot (1 + x)$ (or simply *x* if $v = 0$) where *x* is an error variable. For each country and week, our alignment generates a different system of intertwined equations, which is solved minimizing the discord (i.e., sum of squared error variables as proposed in [30]).

In the table, we can see for each country, the number of systems of equations with the maximum number of variables[10] (i.e., all possible data are available, what happens between weeks 2020W26 and 2021W06, except for UK whose data are only available in EuroStats until 2020W51), as well as

---

[9] https://coronavirus.jhu.edu/data/mortality

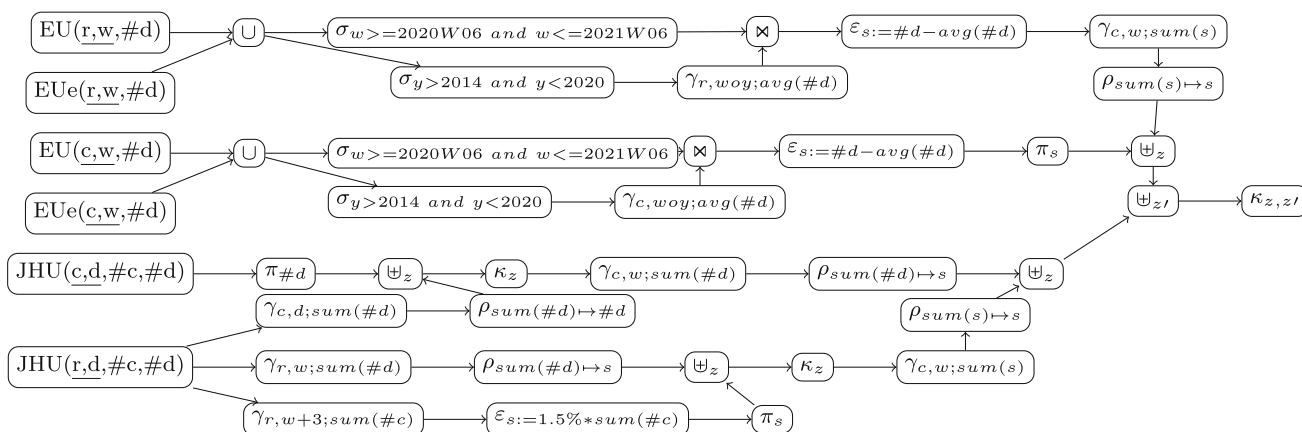[10] We ignored 2020W53, because of its exceptional nature (nonexistent for other years).

Fig. 13 COVID data alignment ($\kappa_z(R \uplus_z S)$ is the implementation of fusion operation $R \sqcup S$ in Definition 6)

Table 2 Characteristics of the equations per country

Country	#Sys	#Eqs	#Vars	Gener.	Solve
DE	37	50	247	2.77s	0.24s
ES	37	54	278	2.79s	0.24s
IT	37	60	322	2.80s	0.24s
NL	37	42	187	2.73s	0.24s
SE	37	59	306	2.68s	0.25s
UK	30	26	75	2.73s	0.24s
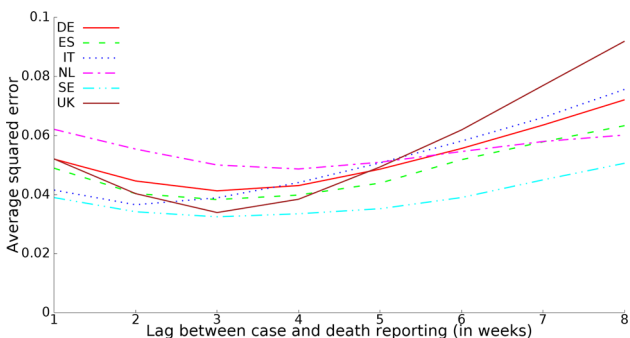
Fig. 14 Error per different alignments of $\#c$ and $\#d$

the equations and variables per system in those cases. The average time in seconds to generate each system of equations as a Python input to OSQP and solve it are also reported.

The line charts in Figs. 14 and 15 plot in the vertical axis discordance (i.e., sum of squared errors in our case) divided by the number of variables (i.e., average squared error per variable), to make them more comparable, since depending on the number of regions, different countries generate more or less variables (see Table 2). Firstly, Fig. 14 varies the lag between reported cases and deaths, for values from one to eight weeks. We can see that the average discordance is minimized in all cases between two and four weeks (three in the average). Thus, in the other charts, we used a lag of three

weeks between cases and deaths, which minimizes the average squared error of all six countries.

Figure 15a shows the evolution of the discordance since 2020W26 until the last week being considered. We can appreciate that during the first weeks reporting regional data, countries adjusted and eventually improved their COVID-19 surveillance mechanisms. However, all of them except UK are too sensitive to the increase of cases and their concordancy with real deaths is clearly affected by the arrival of the second wave after summer and the third one at the end of the year (we can clearly appreciate the two peaks in the five other countries). Unfortunately the UK data reporting to Eurostats stopped on December 31, 2020 due to Brexit, so we cannot see from the Eurostats data whether the UK's error remained low during the rest of the infection wave in early 2021.

Finally, Fig. 15b shows the clearer but less computationally challenging evolution of discordancy without considering regionally reported data (the small pointer in the horizontal axis indicates the horizontal coincidence of both charts). We can see a clear peak of discordancy during the first wave that eventually improves, just to be more or less affected again by the second and third waves depending on the country. As a derived calculation of the observed discordancy, we can take the Pearson correlation coefficient between those and the running average of the number of cases (i.e., DE: 36%; ES: 80%; IT: 50%; NL: 73%; SE: 23%; UK: 93%). Thus, we can observe that in the case of ES, NL and UK, more than 70% of variation in the discordancy can be explained by changes in the number of cases.

Without ERIS, this study had been simply impossible, because manually generating the corresponding SQL and Python code had been too difficult for a human being. Instead, we generated them automatically and with all the correctness guarantees from a relatively simple algebraic sequence of operations. Alternatively, we could have also somehow eas-
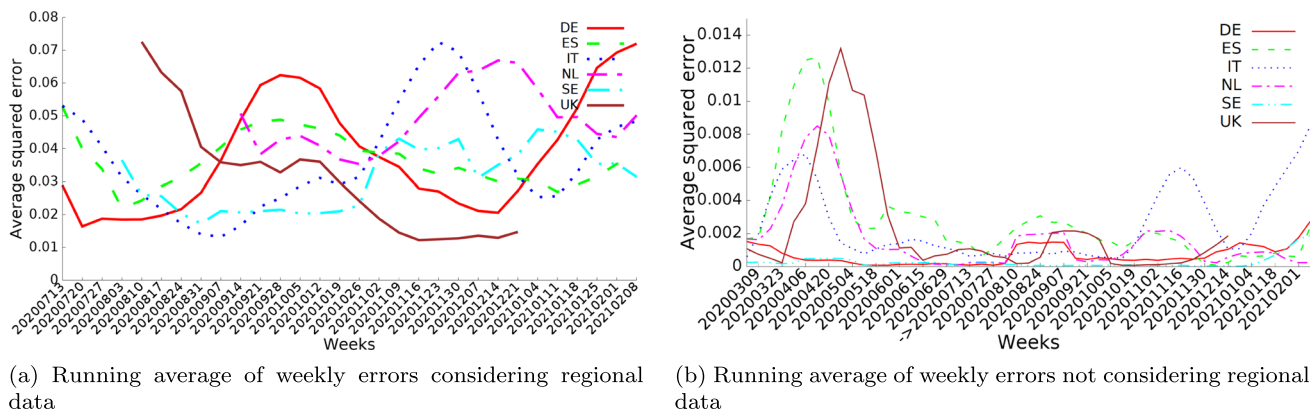
(a) Running average of weekly errors considering regional data

(b) Running average of weekly errors not considering regional data

Fig. 15 Discordancy analysis of COVID-19 data aligning #c and #d with a lag of three weeks

ily defined the corresponding static assertions over JHU data (indicating that they must exactly coincide) and count how many times they were violated. Nevertheless, given the poor quality of the source, all we had gotten is a flat line indicating that all values are simply inconsistent in each and every week with regard to those from EuroStats. Thus, our novel discord measuring mechanism could be easily further integrated in a more complex truth discovery iterative method to obtain the trustworthiness weight of each source. We consider this to be a potential direction for future work.

7 Related work

Out of the many papers identified in a recent systematic literature review on information fusion techniques [26], just one [44] was found making use of consistency to evaluate the quality of input data. However, it only uses the difference between pairs of numbers as the basis for this evaluation. Moreover, this is not actually done with the purpose of evaluating the overall quality of the sources, but rather the coincidence between instances with matching purpose.

There is existing work such as [18], [6] and [24] on measuring differences in the descriptive multidimensional data and their structure. Instead, we aim at evaluating the reliability of the numerical indicators, given some required alignment declaration (e.g., aggregation or scale correction). At this point, it is important to highlight that, even if some work like [36] proposes to treat textual data as indicators (allowing to aggregate them too), we restricted ourselves to numerical measures, whose discrepancies cannot be evaluated using string similarity metrics. The latter would instead be part of a preliminary step of entity matching over dimensional descriptors.

Thus, the problems defined above are related to Consistent Query Answering (CQA) [16], which tries to identify the subset of a database that fulfills some integrity constraints and corresponds to the problem of identifying certain answers under open world assumption [5]. In CQA, distance between two database instances is captured by symmetric difference of tuples. However, in our case, the effects of an alignment are not only reflected in the presence/absence of a tuple, but also in the values it contains. This leads to the much closer Database Fix Problem (DFP) [8, 12], which aims at determining the existence of a fix at a bounded distance measuring variations in the numerical values. Both DFP as well as CQA become undecidable in the presence of aggregation constraints. Nonetheless, these have been used to drive deduplication [15]. However, our case is different since we are not questioning correspondences between entities to create aggregation groups, but instead trying to quantify their (in)consistency in the presence of complex transformations.

Another known result in the area of DFP is that prioritizing the repairs by considering preferences or priorities (like the data sources in our case) just increases complexity. An already explored idea is the use of where-provenance in the justification of the new value [23], but with pure direct value imputation (without any data transformation). In contrast, we consider that there is not any master data, but multiple contradictory sources, and we allow aggregates, while [23] only uses equalities (neither aggregation nor any real arithmetic) between master and target DBs. Related to this, in the area of machine learning, we have [40], which aims at finding counterfactual explanations for a prediction. The purpose in this case is not to directly change the data, but to tell the user what should have been done to get a different prediction. Like in our case, this is treated as an optimization problem.

From the perspective of incompleteness in multidimensional databases, attention is paid to missing values in the measures. [37] presents an approach to maximize entropy, and [9] a linear programming-based framework to impute missing values under constraints generated by sibling data at the same aggregation level, and parent data in higher levels.

We could consider the later a special case of our approach, with a single data source and predefined alignment.

In the context of data fusion, although our purpose is not to merge records, but only evaluate how far they are from one another, we can still position our work according to the characteristics in [14] as follows:

- Data types we consider are continuous (a.k.a. quantitative);
- Heterogeneity of data types is not considered in our work, as we focus on dealing with purely numerical attributes;
- Single-truths (i.e., each attribute has a single value in reality) is assumed;
- Source quality is the focus of our work by measuring their discrepancies;
- Copying between sources is not considered (i.e., instead, we consider they provide their values independently);
- Object relationships can be naturally expressed in the form of symbolic expressions sharing variables;
- Object popularity and difficulty can be considered in the cost functions.

The setting we have described in the context of data fusion and truth discovery shares also many motivations with previous work on provenance. The semiring provenance model [25] is particularly related, explaining why why-provenance [13] is not enough (e.g., in the case of alternative sources for the same data) and we need how-provenance to really understand how different inputs contribute to the result. They propose the use of polynomials to capture such kind of provenance. Further, [4] extended the semiring provenance model to aggregations by mixing together annotations and values, but the fine-grained provenance information may become large. However, to the best of our knowledge no practical implementations of it exist. In contrast, our approach does not have row-level annotations recording the conditions that make a row present in the result, limits aggregation to value- fields, and considers only sum and averaging forms of aggregation, but we have provided practical implementations of this more limited model. As noted earlier, our s-tables are similar in some respects to c-tables studied in incomplete information databases [27]. Our data model and queries are more restricted in some ways, due to the restriction to finite maps, and because we do not allow for conditions affecting the presence of entire rows, but our approach supports aggregation, which is critical for our application area and which was not handled in the original work on c-tables. Similarly, *attribute-level uncertainty bounds* (AU-BDs) in [21] allow to annotate values with intervals (and a selected guess) encoding a set of possible worlds. However, this does not help to find the most likely world (beyond the provided selected guess).

There have been implementations of semiring provenance or c-tables in systems such as Orchestra [28], ProQL [29], ProvSQL [41], and Mimir [35]. In Orchestra, provenance annotations were used for update propagation in a distributed data integration setting. ProQL and ProvSQL implement the semiring model but do not allow for symbolic expressions in data or support aggregation. Mimir is a system for querying uncertain and probabilistic data based on c-tables; however, in Mimir symbolic expressions and conditions are not actually materialized as results, instead the system fills in their values with guesses in order to make queries executable on standard RDBMSs. Thus, Mimir's approach to c-tables would not suffice for our needs since we need to generate the symbolic constraints for the QP solver to solve. On the other hand, our work shows how some of the symbolic computation involved in c-tables can be implemented in-database and it would be interesting to consider whether this approach can be extended to support full c-tables in future work.

We have reduced the concordancy evaluation problem to quadratic programming, a well-studied optimization problem. Solvers such as OSQP [43] can handle systems with thousands of equations and variables. However, we have not made full use of the power of linear/quadratic programming. For example, we could impose additional inequalities on unknowns, to ensure that certain error or null values have to be positive or within some range. Likewise, we have defined the cost function in one specific way but quadratic programming permits many other cost functions, like using different weights for each variable or with additional linear cost factors. As suggested at the end of the last section, it may be worthwhile to combine ERIS with other truth discovery techniques to simultaneously estimate the weights needed for the cost function and the guessed true values of the uncertain data.

As noted in Sect. 2, we have focused on the problem of evaluating concord/discord among data sources and not on using the different data sources to estimate the actual values being measured (like [33]). It would be interesting to extend our framework by augmenting s-tables and queries with a probabilistic interpretation, so that the optimal solution found by quadratic programming produces statistically meaningful consensus values (similarly to [32]).

8 Conclusions

In most data integration and cleaning scenarios, it is assumed that there is some source of ground truth available (i.e., master data or user input). However, in many realistic data fusion settings, such as epidemiological surveillance, ground truth is not obtainable and we need to integrate discordant data sources with different levels of trustworthiness, completeness and self-consistency. In such scenarios, we still would

like to be able to flexibly and efficiently measure how close the observed data are to our idealized expectations. Thus, we proposed definitions of *concordance* and *discordance* capturing, respectively, when data sources we wish to fuse are compatible with one another, and measuring how far away the observed data are from being concordant. Consequently, we can compare measurements over time to understand whether the different sources are becoming more or less consistent with one another. We showed how to flexibly and efficiently solve this problem by extending multidimensional relational queries with symbolic evaluation and gave two relational implementations of this approach reducing it to linear programming or quadratic programming problems that can be solved by an off-the-shelf library. We explored the performance of the two approaches via microbenchmarks to assess the scalability in data size and number of variables, illustrated the value of this information using a case study based on COVID-19 case and death reporting from 2020-2021 and found that the error calculated for six European countries at different times correlates with intuition.

Different cost functions, alternatives in the management of NULL values and zeros, as well as alternatives for variables generation need to be carefully analyzed. However, the most appropriate one will be case-dependent and so we plan to do this separately. Moreover, our approach to symbolic evaluation of multidimensional queries appears to have further applications which we plan to explore next, such supporting other forms of uncertainty expressible as linear constraints, and adapting our approach to produce statistically meaningful estimates of the consensus values.

A Proofs

We state the desired correctness property for the type system formally as follows:

Theorem 2 *Suppose $\Sigma \vdash q : K \triangleright V$ is derivable using the rules in Fig. 3. Then q denotes a function from $Inst(\Sigma)$ to relations $R : K \triangleright V$.*

Proof The proof is by induction on the structure of derivations of $\Sigma \vdash q : K \triangleright V$. Most cases are standard or straightforward. The interesting cases are those where constraints are necessary to preserve the finite map property: for example, projection-away and discriminated union. We sketch the reasoning in each case.

For projection-away, we may only discard value-fields, so the key-fields in the result are the same as those in the input relation resulting from evaluating the subquery. Hence, the finite map property is preserved.

For discriminated union, it is clear from the definition of the semantics that the keys of the result are a tagged disjoint union of the keys of the two input relations, which are both finite maps. Hence the result is a finite map satisfying the FD $K, B \to V$.

For aggregation, the result may drop both key and value-fields, but the value-fields will be aggregated (summed) according to whatever keys remain, so the result will be a finite map $K' \triangleright V'$ by construction. $\square$

Note that some of the constraints on queries are not necessary to ensure well-formed queries produce valid finite maps, but are only needed to ensure that symbolic evaluation is correct on s-tables. For example, if selection conditions were allowed on value-fields (that might be symbolic), then the presence of a tuple with symbolic tested fields in the output would depend on the unknown variable values. This conditional membership is not supported in s-tables, but was considered in c-tables in which the presence of a tuple in the output can be constrained by a formula. While this would be an interesting extension, we do not have a pressing need for this capability in OLAP tools whereas it would significantly complicate the formalism and implementation.

Figure 5 presents additional well-formedness rules for alignment specifications. The judgment $\Sigma \vdash \Delta : \Omega$ says that in schema Σ, the definition of views Δ is well-formed and produces a result matching schema Ω; that is, the new tables defined in Δ are as specified in Ω. This well-formedness judgment satisfies the following correctness property:

Theorem 3 *Suppose $Spec = [\Sigma, \Omega, \Delta]$ is an alignment specification and $\Sigma \vdash \Delta : \Omega$ holds. Then we may interpret Δ as a partial function from instances of Σ to instances of Ω.*

The proof is straightforward; the interpretation of Δ attempts to construct the instance of Ω one relation at a time, using the (partial) fusion operation in each step. Fusion is associative and commutative, so the result is well defined.

A.1 Linearity

In order to ensure that constraints generated by symbolic evaluation and fusion/coalescing are valid linear programming problems, we need to restrict the s-tables/s-instances to include linear expressions only and also restrict queries so that derivation steps only involve linear combinations of attributes. Subject to these restrictions, we can verify that for well-formed queries and alignment specifications, the resulting s-tables, s-instances, and constraints are linear as well.

Theorem 4 *Suppose q is well-formed, satisfying $\Sigma \vdash q : K \triangleright V$, and all occurrences of derivations in q are linear expressions. Suppose in addition I is a well-formed linear s-instance. Then, $q(I)$ is a well-formed linear s-table $q(I) : K \triangleright V$.*

Proof The proof is by induction on the structure/well-formedness derivation of q. Many cases, e.g., the base case $q = R$, selection, projection, join, renaming, etc., are straightforward. The case of derivation follows because derivations are required to be linear expressions over fields. The case of aggregation follows because the only kind of aggregation allowed is SUM, and adding together any number of linear expressions is still a linear expression. □

Theorem 5 *Suppose $Spec = [\Sigma, \Omega, \Delta]$ is well-formed, satisfying $\Sigma \vdash \Delta : \Omega$, and all occurrences of derivations in Δ are linear expressions. Suppose in addition I is a well-formed linear s-instance matching Σ. Then, the result of evaluating Δ on I, a constrained s-instance (J, ϕ), is linear, that is, J is a linear s-instance matching Ω, and ϕ is a conjunction of linear constraints.*

Proof The proof is by induction on the structure/well-formedness derivation of $\Sigma \vdash \Delta : \Omega$. The base case is immediate. For the inductive step where Δ consists of a binding $T := q_1 \sqcup \cdots \sqcup q_n$ followed by another sequence Δ', note that in this case Ω must be of the form $T : K \triangleright V, \Omega'$ and the well-formedness relations $\Sigma \vdash q_i : K \triangleright V$ (for each i) and $\Sigma, T : K \triangleright V \vdash \Delta' : \Omega'$ must hold. Using Theorem 4, since each q_i is well-formed in Σ satisfying $\Sigma \vdash q_i : K \triangleright V$, we know that each q_i preserves linearity, so $q_i(I)$ is linear for each i. Moreover, the fusion of all of the q_i's can be expressed as an n-way coalescing of $q_i(I)$ and we can inspect the definition of coalescing to check that its result (T', ϕ) is a linear s-instance and a conjunction of linear constraints. Since $\Sigma, T : K \triangleright V \vdash \Delta' : \Omega'$ holds, we can apply the induction hypothesis using the specification $[(\Sigma, T : K \triangleright V), \Delta', \Omega']$ and to I extended with $T = T'$, since I and T' are both linear. Thus, we can conclude that the final s-instance and constraint (J, ϕ') obtained are linear also. □

We assume from now on that the s-tables are linear and the queries only involve linear expressions in derivation steps.

A.2 Correctness of symbolic evaluation

We require that symbolic evaluation correctly abstracts ground evaluation, in the sense that evaluating symbolically and then filling in ground values yields the same results as evaluating on fully ground input tables. We also expect that, as s-tables represent sets of ground tables, the symbolic evaluation of query operations over tables correctly reflects the possible sets of ground tables resulting from the query operation.

These properties are closely related and similar to the standard correctness properties used for incomplete information representations such as c-tables [27]. However, there is an important difference: in the classical setting, the variables representing unknown values are "scoped" at the level of tables. That is, if table R and S both mention some variable x, the occurrences in R and, respectively, S are *local* to the respective table, and the value of x in R may not have anything to do with that in S. In our case, however, we wish to reason about situations where unknown values propagate from source tables in I through view definitions in J, and we definitely do *not* want the variables appearing in different tables to be unrelated; instead, we want the variables to have *global* scope.

To prove the main correctness property, we first need a lemma concerning the behavior of the individual operators.

Lemma 1 *Each s-table operation commutes with valuations:*

1. $\sigma_c(h(R)) = h(\sigma_c(R))$
2. $\hat{\pi}_W(h(R)) = h(\hat{\pi}_W(R))$
3. $h(R) \bowtie h(S) = h(R \bowtie S)$
4. $h(R) \uplus_B h(S) = h(R \uplus_B S)$
5. $h(R) \backslash h(S) = h(R \backslash S)$
6. $\rho_{B \mapsto B'}(h(R)) = h(\rho_{B \mapsto B'}(R))$
7. $\varepsilon_{B := e}(h(R)) = h(\varepsilon_{B := e}(R))$
8. $\gamma_{K'; V'}(h(R)) = h(\gamma_{K'; V'}(R))$

Proof We consider selected cases; the rest are straightforward.

For part (1), we need to show that the result of a selection applied to a grounded symbolic table $h(R)$ is the same as performing the selection symbolically and then applying the grounding valuation. This is the case because the selection condition cannot mention value-fields, and so the decision whether to select a tuple cannot depend on symbolic fields that might be affected by h.

For part (2), again since projection-away can only affect value-fields, the key-fields are unaffected so performing the projection-away on the grounded table $h(R)$ is the same as grounding the projected-away symbolic table.

For part (3) the reasoning is similar to that for selection, since joins can only involve common key-fields and not value-fields.

For part (4) again since the discriminant field B is a key-field which cannot be affected by h, the result is immediate.

Parts (5) and (6) are likewise straightforward.

For part (7), the expression e used in the derivation will not mention any variables in X (but only attributes of R), so h will commute with e, and the desired result follows by calculation.

Finally, for part (8) we again note that since key-fields may not be symbolic, the keys of the result of aggregating the grounded $h(R)$ are the same as for aggregating the symbolic table R, and by unfolding definitions we can check that the results obtained are the same.   □

We now prove a slight generalization of Theorem 1:

Theorem 6 *For any well-formed query q such that $\Sigma \vdash q : K \triangleright V$, any s-instance $I \in Inst(\Sigma)$ with variables X and any valuation $h : \mathbb{R}^X$, we have $h(q(I)) = q(h(I))$. Moreover, $q(\!|I|\!) = (\!|q(I)|\!)$.*

Proof The second part (corresponding to Theorem 1) follows from the first. The proof of the first part is by induction on the structure/ well-formedness judgment of q. Each case (except for the base case of a relation name) corresponds to part of Lemma 1. For the second part, we reason as follows:

$$q(\!|I|\!) = \{q(J) \mid J \in (\!|I|\!)\} = \{q(h(I)) \mid h : \mathbb{R}^X\} = \{h(q(I)) \mid h : \mathbb{R}^X\} = (\!|q(I)|\!)$$

□

This is similar to the standard results about c-tables showing that they form a strong representation system for relational queries over incomplete databases [27]. The main difference is that, as previously said, in our setting, the variables mapped by h are globally scoped. This means that to correctly simulate operations that take multiple tables, such as joins and unions, we do not need to rename the variables to avoid unintended overlap. In fact, this would be incorrect: suppose $I(R) = \{(a : 1, b : x)\}$ and $I(S) = \{(a : 1, c : x)\}$. Then using our semantics, the join $R \bowtie S$ evaluated in I is $\{(a = 1, b = x, c = x)\}$ which represents all tuples where $a = 1$ and the b and c fields are equal real numbers. In contrast, using c-table semantics, the variables in R and S would be renamed so the join result would be some variant of $\{(a : 1, b : x', c : y')\}$ which represents all tuples whose a component is 1, with no constraint relating b and c.

References

1. Abelló, A., Cheney, J.: Measuring discord among multidimensional data sources. In: DOLAP, pp. 96–100. ACM (2022)

2. Abelló, A., Romero, O.: Online analytical processing. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, 2nd edn., pp. 2558–2563. Springer, New York (2018)

3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)

4. Amsterdamer, Y., Deutch, D., Tannen, V.: Provenance for aggregate queries. In: PODS, pp. 153–164. ACM (2011)

5. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)

6. Baikousi, E., Rogkakos, G., Vassiliadis, P.: Similarity measures for multidimensional data. In: ICDE, pp. 171–182. IEEE (2011)

7. Bertossi, L.E.: Repair-based degrees of database inconsistency. In: LPNMR, LNCS, vol. 11481, pp. 195–209. Springer (2019)

8. Bertossi, L.E., Bravo, L., Franconi, E., Lopatenko, A.: Complexity and approximation of fixing numerical attributes in databases under integrity constraints. In: DBPL, LNCS, vol. 3774, pp. 262–278. Springer (2005)

9. Bimonte, S., Ren, L., Koueya, N.: A linear programming-based framework for handling missing data in multi-granular data warehouses. Data Knowl Eng **128**(101), 832 (2020)

10. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput Surv **41**(1), 1:1-1:41 (2008)

11. Bloch, I.: Information combination operators for data fusion: a comparative review with classification. IEEE Trans Syst Man Cybern Part A **26**(1), 52–67 (1996)

12. Bohannon, P., Flaster, M., Fan, W., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD, pp. 143–154. ACM (2005)

13. Buneman, P., Khanna, S., Tan, W.C.: Why and where: a characterization of data provenance. In: ICDT, LNCS, vol. 1973, pp. 316–330. Springer (2001)

14. Canalle, G.K., Salgado, A.C., Lóscio, B.F.: A survey on data fusion: what for? in what form? what is next? J Intell Inf Syst **57**(1), 25–50 (2021)

15. Chaudhuri, S., Sarma, A.D., Ganti, V., Kaushik, R.: Leveraging aggregate constraints for deduplication. In: SIGMOD, pp. 437–448. ACM (2007)

16. Chomicki, J.: Consistent query answering: five easy pieces. In: ICDT, LNCS , vol. 4353, pp. 1–17. Springer (2007)

17. Dong, E., Du, H., Gardner, L.: An interactive web-based dashboard to track COVID-19 in real time. Lancet **20**, 533–534 (2020)

18. Dyreson, C.E., Pedersen, T.B., Jensen, C.S.: Incomplete information in multidimensional databases. In: Rafanelli, M. (ed.) Multidimensional Databases: Problems and Solutions, pp. 282–309. Idea Group, Pennsylvania (2003)

19. Esteva, M., Xu, W., Simone, N., Gupta, A., Jah, M.: Modeling data curation to scientific inquiry: a case study for multimodal data integration. In: JCDL, pp. 235–242. ACM (2020)

20. Fan, W., Geerts, F.: Relative information completeness. In: PODS, pp. 97–106. ACM (2009)

21. Feng, S., Glavic, B., Huber, A., Kennedy, O.A.: Efficient uncertainty tracking for complex queries with attribute-level bounds. In: SIGMOD, pp. 528–540. ACM (2021)

22. Gagolewski, M.: Data fusion: theory, methods, and applications. CoRR arXiv:2208.01644 (2022)

23. Geerts, F., Mecca, G., Papotti, P., Santoro, D.: The LLUNATIC Data-Cleaning Framework. In: PVLDB **6**(9), 625–636 (2013)

24. Golfarelli, M., Turricchia, E.: A characterization of hierarchical computable distance functions for data warehouse systems. Decis Support Syst **62**, 144–157 (2014)

25. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS, pp. 31–40. ACM (2007)

26. Gutiérrez, R., Rampérez, V., Paggi, H., Lara, J.A., Soriano, J.: On the use of information fusion techniques to improve information

quality: taxonomy, opportunities and challenges. Inf Fusion **78**, 102–137 (2022)

27. Imielinski, T., Lipski, W., Jr.: Incomplete information in relational databases. J ACM **31**(4), 761–791 (1984)

28. Ives, Z.G., Green, T.J., Karvounarakis, G., Taylor, N.E., Tannen, V., Talukdar, P.P., Jacob, M., Pereira, F.C.N.: The ORCHESTRA collaborative data sharing system. SIGMOD Rec **37**(3), 26–32 (2008)

29. Karvounarakis, G., Ives, Z.G., Tannen, V.: Querying data provenance. In: SIGMOD, pp. 951–962. ACM (2010)

30. Li, Y., Gao, J., Meng, C., Li, Q., Su, L., Zhao, B., Fan, W., Han, J.: A survey on truth discovery. SIGKDD Explor **17**(2), 1–16 (2015)

31. Livshits, E., Kimelfeld, B.: The Shapley value of inconsistency measures for functional dependencies. Log Methods Comput Sci **18**(2), 20:1-20:33 (2022)

32. Mayfield, C., Neville, J., Prabhakar, S.: ERACER: a database approach for statistical inference and data cleaning. In: SIGMOD, pp. 75–86. ACM (2010)

33. Motro, A., Anokhin, P.: Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. Inf. Fusion **7**(2), 176–196 (2006)

34. Mountantonakis, M., Tzitzikas, Y.: Large-scale semantic integration of linked data: A survey. ACM Comput. Surv. **52**(5), 103:1-103:40 (2019)

35. Nandi, A., Yang, Y., Kennedy, O., Glavic, B., Fehling, R., Liu, Z.H., Gawlick, D.: Mimir: bringing CTables into practice. CoRR arXiv:1601.00073 (2016)

36. Oukid, L., Boussaid, O., Benblidia, N., Bentayeb, F.: TLabel: a new OLAP aggregation operator in text cubes. Int. J. Data Warehous. Min. **12**(4), 54–74 (2016)

37. Palpanas, T., Koudas, N., Mendelzon, A.O.: Using datacube aggregates for approximate querying and deviation detection. IEEE Trans. Knowl. Data Eng. **17**(11), 1465–1477 (2005)

38. Parisi, F., Grant, J.: On measuring inconsistency in relational databases with denial constraints. In: ECAI, vol. 325, pp. 857–864. IOS Press (2020)

39. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)

40. Schleich, M., Geng, Z., Zhang, Y., Suciu, D.: Geco: quality counterfactual explanations in real time. In: PVLDB, vol. 14(9), pp. 1681–1693 (2021)

41. Senellart, P., Jachiet, L., Maniu, S., Ramusat, Y.: ProvSQL: provenance and probability management in PostgreSQL. In: PVLDB, vol. 11(12), pp. 2034–2037 (2018)

42. Spielman, D., Teng, S.H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In: STOC, pp. 296–305. ACM (2001) https://doi.org/10.1145/380752.380813

43. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. Math. Program. Comput. **12**(4), 637–672 (2020)

44. Tré, G.D., Britsom, D.V., Matthé, T., Bronselaer, A.: Automated cleansing of POI databases. In: Quality Issues in the Management of Web Information, Intelligent Systems Reference Library, vol. 50, pp. 55–91. Springer (2013)

45. Vavasis, S.A.: Complexity theory: quadratic programming. In: Encyclopedia of Optimization, pp. 304–307. Springer (2001) https://doi.org/10.1007/0-306-48332-7_65

46. Yu, M., Li, G., Deng, D., Feng, J.: String similarity search and join: a survey. Front. Comput. Sci. **10**(3), 399–417 (2016)