J Intell Inf Syst (2016) 46:409–424
DOI 10.1007/s10844-015-0359-2

A systematic approach to reliability assessment in integrated databases

Vladimir Zadorozhny[1] · **John Grant**[2]

Received: 27 March 2014 / Revised: 4 March 2015 / Accepted: 14 April 2015 /
Published online: 12 May 2015

Abstract We provide a novel framework based on a systematic treatment of data inconsistency and the related concept of data reliability in integrated databases. Our main contribution is the formalization of reliability assessment for historical data where redundancy and inconsistency are common. We discover data inconsistency through the analysis of relationships between existing reports in the integrated database. We present a new approach by defining properties (rules) that a good measure of reliability should satisfy. We then propose such measures and show which properties they satisfy. We also report on a simulation-based study of the introduced framework.

Keywords Data integration systems · Data inconsistency · Data reliability assessment

1 Introduction

Continued research on Data Integration Systems aims to provide users with uniform data access and efficient data sharing. The ability to share data is particularly important for interdisciplinary research, where a comprehensive picture of the subject requires large amounts of *historical data* from disparate data sources from a variety of disciplines. For example, epidemiological data analysis often relies upon knowledge of population dynamics, climate change, migration of biological species, drug development, etc. As another example, consider the task of exploring long-term and short-term social changes, which requires consolidation of a comprehensive set of data on social-scientific, health, and environmental dynamics.

Nowadays, there are numerous historical data sets available from various groups worldwide such as the Institute for Quantitative Social Science and the Center for Geographic Analysis at

✉ Vladimir Zadorozhny
vladimir@sis.pitt.edu

John Grant
grant@cs.umd.edu

[1] School of Information Sciences, University of Pittsburgh, Pittsburgh, PA, USA

[2] Department of Computer Science & UMIACS, University of Maryland, College Park, MD, USA

Harvard, Great Britain Historical GIS at Portsmouth, the International Institute of Social History in Amsterdam, the CLIO World Tables at Boston University, and World-Historical Dataverse at the University of Pittsburgh. Notable prior campaigns of data collection and analysis include the Integrated Public Use Microdata Series (IPUMS, at Minnesota), Electronic Cultural Atlas Initiative (ECAI), the Alexandria Digital Library (ADL), and others.

While the aforementioned initiatives indicate a considerable effort to utilize diverse historical data sources, researchers are nowhere near to having a global historical data repository against which to perform comprehensive socio-scientific analysis and to test emerging large-scale theories. The existing data sources are principally oriented toward regional comparative efforts rather than global applications. They vary widely both in content and format. Scalable methods for integration of the existing and emerging historical data sources would considerably advance global data utilization. In addition to resolving data heterogeneity, such methods should support efficient data curation strategies based on *data reliability assessment*. Historical data sources may have different levels of reliability for many reasons, e.g., issues with the primary sources of information, faulty data collection methodology, etc. Integration of the historical data sources may also face *data redundancy*. It is common to have multiple reports about the same event within *overlapping time intervals*. For example, we may have hundreds of reports from different authorities about cases of measles in Los Angeles in 1900. We may also have multiple reports on historical statistics for *overlapping locations*. A cumulative report on the total number of measles cases for the entire state of California may differ considerably from the available reports on the total number of measles cases in individual California cities. Another challenge is *overlapping names*: evolving concepts may be reported under different names and categories co-existing at different time intervals. For example, many 19th century reports on yellow fever were actually referring to cases of hepatitis. Note that historical data redundancy does not necessarily imply data inconsistency. But even if the overlapping historical reports are accurate, data redundancy prevents researchers from obtaining reliable aggregate query results. Meanwhile, data inconsistency is caused by inaccurate reports.

In this paper we provide a systematic treatment of data redundancy and inconsistency and the related concept of data reliability. We explore how data inconsistency can be utilized for data reliability assessment. We consider an approach to discover data inconsistency through the analysis of relationships between existing reports in an integrated database. We present a new approach by defining rules that a good measure of reliability should satisfy. We then propose such measures and show which rules they satisfy. Our main contribution is the formalization of reliability assessment for historical data where redundancy and inconsistency are common. But actually, these issues reoccur in data fusion for any type of data. Hence we believe that our approach has wide applicability.

The plan of the paper is as follows. Section 2 reviews related works with a brief background on inconsistency measurement. In Section 3 we provide a motivating example of historical data redundancy and inconsistency. Section 4 presents our formalization. In Section 5 we introduce our measures of reliability and some desirable rules that any measure of reliability should satisfy. Section 6 contains our simulation-based study illustrating the behavior of our reliability measures. We conclude the paper in Section 7.

2 Related work

The problems of data redundancy and inconsistency considered in this paper are of general applicability to large-scale Data Integration Systems. Data Integration Systems must address

two major challenges: (1) *heterogeneous data* and (2) *conflicting data*. Resolving data heterogeneities has been the focus of active research and development for more than two decades (Brodie 2010; Haas 2007). There are numerous tools for efficient mapping of data sources in a homogenous schema with proper data cleaning (eliminating typos, misspellings, and formatting errors), standardization of names, conversion of data types, duplicate elimination, etc. A separate body of research deals with Web data integration and wrapper/mediator architectures, which includes our work on accessing heterogeneous Web data sources (Zadorozhny et al. 2005, 2008; Zadorozhny and Raschid 2007).

The amount of research in the area of data conflict resolution and querying inconsistent data is also considerable. (Dong and Naumann 2009; Bleiholder and Naumann 2009) and (Bertossi 2006; Bertossi and Chomicki 2003) provide a comprehensive review of the current state of the art. Early research on handling inconsistencies was mostly theoretical and did not relate this problem directly to data integration (Imelinski and Lipski 1984). Data inconsistency as a key integrity constraint violation was considered in (Agarwal et al. 1995). Consistent query answering that ignores inconsistent data, thereby violating integrity constraints, was introduced in (Bry 1997). This approach is related to more recent research on query transformation for consistent query answering (Wijsen 2009). An alternative approach is based on inconsistent database repair, producing a minimally different—yet consistent—database that satisfies integrity constraints (Staworko and Chomicki 2010; Bohannon et al. 2005; Wijsen 2005). Our work on information integration based on crowdsourcing and conflict-aware data fusion represent a new research direction in this area (Zadorozhny et al. 2013; Zadorozhny and Hsu 2011).

The idea that a set of formulas may be considered more inconsistent than another set was introduced in the pioneering paper (Grant 1978). This work was then taken up again in the 1990s as the handling of inconsistency became an important issue in databases. In particular, the work was recast as the problem of assigning an inconsistency measure to a set of formulas, a knowledge base. Over the last 20 years researchers have proposed numerous inconsistency measures for knowledge bases. A good review of the research up to 2005 appears in (Hunter and Konieczny 2005). Since then both additional inconsistency measures as well as properties that such an inconsistency measure should satisfy have been studied. The following are some of the important papers in this field: (Grant and Hunter 2006, 2011; Hunter and Konieczny 2010; Mu et al. 2011a, b), and (Grant and Hunter 2013). For our purpose there is no need to give the details of these papers.

In this section we briefly summarize some of this work, using primarily (Grant and Hunter 2011) as the reference. An important conclusion of that paper is that there is no single definition of inconsistency measure that is best in all circumstances. Although this previous inconsistency measure work deals primarily with symbolic, rather than numeric data, its conclusion is applicable to our case as well; namely, that we cannot expect to have a single definition for inconsistency, and hence reliability, that is always best.

A knowledge base is usually represented as a finite set of formulas in some appropriate language such as propositional logic. Let DB$=\{a_1, \ldots, a_n\}$ be a knowledge base. An inconsistent subset of DB is any subset S of DB that is inconsistent in classical 2-valued logic. A minimal inconsistent subset S is one that has no proper inconsistent subset. The free formulas of DB are all the formulas that are not in any minimal inconsistent subset. Intuitively this means that a free formula does not participate in any inconsistency. For a very simple example, let DB$=\{a, b, \sim a \vee \sim b, b \vee c\}$. Then S$=\{a, b, \sim a \vee \sim b\}$ is a minimal inconsistent subset and Free(DB)$=\{b \vee c\}$.

Before defining some inconsistency measures, let us consider what properties such a measure should satisfy. Clearly, an inconsistency measure should be a function that assigns a nonnegative real number to every knowledge base. A consistent knowledge base should be assigned the measure 0 and an inconsistent knowledge base a measure greater than 0. We would like a knowledge base DB that is more inconsistent (whatever that means) than DB' to have a higher measure. The problem is to determine what it means for DB to be more inconsistent than DB'.

Inconsistency measures can be defined in different ways. One approach gives an absolute inconsistency value to each knowledge base, while another approach provides an inconsistency value that is relative to the size of the knowledge base. Consider the following situation. A database DB is inconsistent and has measure *i*. Suppose now that we add a formula *a* that does not cause any new inconsistency, that is *a* is free in DB $\cup$ $\{a\}$. Both DB and DB $\cup$ $\{a\}$ should have the same inconsistency measure because in absolute terms the inconsistency of the database does not change. But the addition of *a* increases the size of the knowledge base; hence relative to the size of the knowledge base the inconsistency of DB union $\{a\}$ should be less than the inconsistency of DB. Most research on inconsistency measures uses the absolute approach, but if we are interested in calculating the reliability of a database, the relative approach, where the database size is taken into consideration, is a better way to go. It seems reasonable to consider a large database with a few inconsistencies to be more reliable than a small database with the same number of inconsistencies.

In this brief review we can only mention a few concepts from the general theory of inconsistency measures that are particularly relevant to our situation. The concept of minimal inconsistent subset is particularly important. Such sets capture the essence of the inconsistencies. One common measure counts the number of minimal inconsistent subsets. But that is an absolute inconsistency measure. Note how this measure does not change the inconsistency when a free formula is added.

The reviewed work does not carry over precisely to our situation because we are interested not so much in the inconsistency of a DB but in the inconsistency of a formula within the DB. However, we will use some of the concepts in our framework with the caveat that measuring inconsistency in a historical database with summary numeric values presents challenges that are different from measuring the inconsistency of a set of propositional logic formulas.

3 Historical data redundancy and inconsistency: a motivating example

Historical data reports on events occurring within various time intervals. As a result, it may include *data redundancy* that prevents researchers from obtaining the correct answers to queries on an integrated historical database. Basically, there are three major types of redundancy that may occur between the historical reports: (1) *temporal redundancy*, (2) *spatial redundancy,* and (3) *naming redundancy.* We use the following examples for illustration purposes only. They do not represent any actual disease occurrences.

> **Temporal Redundancy.** It is possible to have multiple concurrent reports about the same disease in the same location within *overlapping time intervals*. Figure 1 shows an example of a historical epidemiological database including data references for the total number of cases of measles in NYC (tuples *t1, t4*). We cannot simply add the numbers of *t1* and *t4* to find the total number of cases of measles from 1900 to 1930, because *t1* and *t4* have overlapping time intervals. There is a temporal redundancy between *t1* and *t4*.

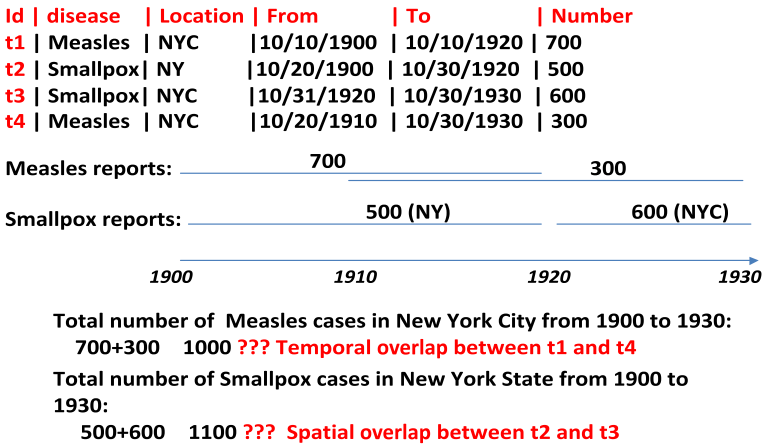


Fig. 1 Example of temporal and spatial redundancy

Spatial Redundancy. We may also have multiple reports on disease statistics for *overlapping locations*. Figure 1 also shows an example of two data references for the total number of cases of smallpox in the state of New York (tuple *t2*), and the corresponding total cases of smallpox in New York City (tuple *t3)*. Although the time intervals of *t2* and *t3* do not overlap, we cannot simply add up their corresponding numbers to obtain the total number of smallpox cases in the state of New York. Tuple *t2* refers to the total number of smallpox cases reported for the state of New York. Meanwhile, it is unknown if this includes all New York City cases reported in *t3*. In any case, *t3* does not include smallpox cases in New York State outside of New York City. There is a spatial redundancy between *t2* and *t3*.

Naming Redundancy. Evolving concepts may be reported under different names and categories co-existing at different time intervals. For example, many 19th century reports on yellow fever were actually referring to cases of hepatitis. Beginning in 1947, viral hepatitis was classified as hepatitis A and hepatitis B; that distinction was not immediately reflected in the epidemiological records.

Historical data redundancy does not necessarily imply *inconsistency*. Data inconsistency is caused by *inaccurate* reports. Analyzing relationships between existing reports in the redundant database can discover such inconsistency. For example, several reports may reflect different numbers of diseases for the same location and time interval. Another case of inconsistency is illustrated in Fig. 2a. Here, report *R1* reflects a larger number of measles

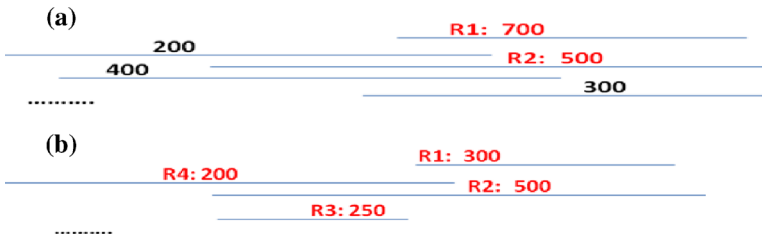


Fig. 2 Redundant and inconsistent databases

cases in NYC (700) for a smaller time interval than report *R2* (reporting only 500 cases). Discovering inconsistency may be challenging in a large-scale historical database. Figure 2b illustrates a more complex case of inconsistency within four reports as shown below. The total number of R1 and R3 (550) should not be greater than the number reported in R2 (500). The number reported in R3 (250) should also be smaller than the number or R4 (200).

In order to utilize the integrated historical data properly we will have to perform efficient *data reliability assessment*. Such assessment is based on an analysis of data inconsistency. In the next section we elaborate on our approach to discover and measure data inconsistency.

4 Formalization of inconsistency

We now define formally the concepts needed to assess reliability via inconsistency. We start by defining the types of tuples we consider. In this framework we proceed to define an important case of inconsistency. We will be dealing with a set of objects that belong to some category, such as Disease, named regions of space, time intervals and real numbers (values).

Category. A category contains a partially ordered set of names of concepts within the category. Consider the category to be Disease. The names then are specific diseases such as *flu, stomach ulcer, GI disease*, etc. We write *stomach ulcer*$\leq$*GI* disease because every stomach ulcer is a GI disease; this is the basis of the partial ordering. We extend the$\leq$ relation to sets of names as follows: If G and H are sets of names then we write $G \leq H$ if for every name $g \in G$ there is a name $h \in H$ such that $g \leq h$.

Space. Space S consists of a finite set of points; every subset R of S is a region but we will use names for the regions such as *Pittsburgh, New York City, Pennsylvania*. The partial ordering here is the subset relation: *Pittsburgh* $\subseteq$ *Pennsylvania*.

Time. We fix a temporal reference system T to consist of nonnegative numbers in a range $[0,M]$. A time point $t \in T$ and a time interval $[s,e] = \{t \in T \mid s \leq t \leq e\}$. The subset ordering on time intervals is as follows: $[s_1,e_1] \subseteq [s_2,e_2]$ if $s_2 \leq s_1$ and $e_1 \leq e_2$. This means that every time point in $[s_1,e_1]$ is also in $[s_2,e_2]$.

This subset ordering can be extended to unions of intervals. Let *I1* and *I2* be such unions. Then *I1*$\leq$*I2* if every point in an interval in *I1* is in an interval in *I2*.

Value. All real numbers.

Next we define the concept of a database that contains specific types of tuples using the object types just given. In order to simplify notation we restrict the concept of tuple to a specific type of 4-tuple.

Tuple. A tuple is a 4-tuple <*category name, region, time interval, value*>. For example:

$$id_1 = < measles, NYC, [1900, 1920], 700 > \text{ is a tuple.}$$

We will sometimes use identifiers to identify tuples, as shown above. We will also identify tuple components by writing abbreviations for the names of components: *id[cat]*, *id[reg]*, *id[int]*, *id[val]*. Hence $id_1[reg] = NYC$.

Database. A database is a finite set of tuples.

For the semantics we define the consistency of a database. We start with the consistency of a pair of tuples.

Pairwise tuple consistency. A pair of tuples id_1=<*c1, r1, i1, v1*> and id_2=<*c2, r2, i2, v2*> is (tuple) consistent if the following condition is satisfied:

$$For\ 1 \le j \ne k \le 2 \text{ if } cj \le ck, rj \subseteq rk, \text{and } ij \subseteq ik \text{ then } vj \le vk.$$

Pairwise database consistency. If every pair of tuples is consistent in *DB* then we say that *DB* is pairwise consistent.

While pairwise database consistency is a useful concept that can be checked relatively fast, it is not the whole story about the consistency of a database. The problem is that inconsistency may occur when multiple tuples are taken together. Consider the following simple example.

Example 1 DB1: id_1=<*c,r,[2,4],20*>, id_2=<*c,r,[5,7],30*>, and id_3=<*c,r,[1,8],40*>.

This DB contains just three tuples where *c* and *r* are fixed values. It is pairwise consistent but intuitively it is not consistent because taken together *20+ 30> 40*, even though [2,4] ∪ [5,7] ⊆ [1,8].

Note how the category and region values are the same for all three tuples. Suppose we change a region so that the new database, DB2 contains a new region *r′* where $r \subseteq r'$.

Example 2 DB2: id_1=<*c,r,[2,4],20*>, id_2=<*c,r,[5,7],30*>, and id_3=<*c,r′,[1,8],40*>.

DB2 is still inconsistent. But if *r* is not a subset of *r′* then DB2 becomes consistent because we cannot compare the values for different regions.

Hence to completely characterize consistency we need to consider all dimensions: category, region, and time interval. Finding all cases of inconsistency is a complex task. In this paper we define only the type of inconsistency that appears to be the most common one in applications, namely the case where the category and region values are the same for all tuples and the consistency issue involves time intervals only. To simplify matters we also avoid a certain kind of overlap on time intervals that we explain later.

We start with some definitions. Let *t1* = <*cat1,reg1,int1,val1*> and *t2* = <*cat2,reg2,int2,val2*> be 2 tuples.

Pairwise non-time overlapping. A set of tuples is pairwise non-time overlapping if for every pair of tuples *t1* and *t2*, *int1* ∩ *int2* = ∅.

Time-included. Let A and B be sets of tuples. A is time-included in B if

$$\cup_{i=1}^{n} int_{a\_i} \subseteq \cup_{j=1}^{m} int_{b\_j}$$

Now we define a special type of inconsistency using the concepts just given. There are other types of inconsistencies as we will discuss later. However, this is the only type of inconsistency that we will deal with; so for the purpose of this paper that is what we define as an inconsistency.

Inconsistent DB. DB is inconsistent if there exist two pairwise non-time overlapping sets of tuples in DB, A and B, such that A is time-included in B, and $\sum_{i=1}^{n} v_{a\_i} > \sum_{j=1}^{m} v_{b\_j}$.

This is exactly the kind of situation we had in DB2. Now, given such an A and B, we say that the pair $<A, B>$ is an inconsistent pair. We call $<A,B>$ a minimal inconsistent pair if there are no sets A′ and B' such that $<A,B> \neq <A',B'>$, $<A',B'>$ is an inconsistent pair, $A' \subseteq A$ and $B' \subseteq B$. We say that $<A,B>$ is a *mip*. We say id *is a member* of a mip <A,B> (we use the notation $id \in <A,B>$) to mean that either $id \in A$ or $id \in B$ for such a minimal inconsistent pair of sets. For a set of ids C, $C \subseteq <A, B>$ if for every $id \in$ C, $id \in <A,B>$.

A crude measure of inconsistency is to count the number of minimal inconsistent subsets of *DB*. If we are interested in the sizes of the sets A and B the appropriate measure is to count $\sum_{<A,B> \in MI(DB)} 1/|A \cup B|$, so that if inconsistencies require larger sets of tuples their measure is diminished.

We conclude with an example of an inconsistency involving time intervals that is not covered by our definition.

Example 3 DB3: $id_1=<c,r,[2,4],20>$, $id_2=<c,r,[3,7],30>$, and $id_3=<c,r,[2,7],60>$.

Intuitively, DB3 is inconsistent because according to the first two tuples, the maximum value for the interval [2,7] cannot be greater than 50. But the first two tuples are time-overlapping; hence our definition of inconsistency does not apply. Note that we also did not consider the case where there are differences in the category or region. A complete analysis of all possible inconsistencies is beyond the scope of this paper where our interest is in finding typical inconsistencies in historical databases and using them to define reliability.

5 Reliability

In this section we define ways of measuring the reliability of tuples in a database. As our measure involves the presence of inconsistencies, it is more convenient to define a measure for the unreliability of a tuple. This will not be a problem because unreliability will be normalized to a value between 0 and 1. Hence writing *r(id)* (resp. *u(id)*) for the reliability (resp. unreliability) of a tuple *id*, knowing *u(id)* we simply assign *r(id)= 1-u(id)*.

We start dealing with reliability by devising *properties* that a reasonable definition of reliability should satisfy. Then we will give a definition for reliability (actually many definitions on account of parameters) that satisfies some of these properties. As mentioned in Section 4 the relevant concept of inconsistency is the *mip* which is what we will use.

5.1 Properties of (un)reliability

In our setup we have a database *DB* and a tuple *id*. The first property deals with the extreme cases: when a tuple is completely reliable or completely unreliable.

P1(a). *u(id)*=0 if and only if *id* ∉ *mip* for any *mip* of *DB*.

Thus internally we consider a tuple completely reliable if and only if it is not in any inconsistency.

P1(b). *u(id)*=1 if and only if *id* ∈ *mip* for every *mip* of *DB*.

A tuple that is a part of every inconsistency has a maximal (normalized) unreliability.

For the following three properties we consider what happens to a tuple *id* in *DB* when a new tuple *id′* is added to *DB* to obtain *DB′*. We write *u′(id)* for the unreliability of *id* in *DB′*.

P2. If *id′* ∉ *mip* for any *mip* of *DB*' then *u(id)=u′(id)*.

Thus the reliability of a tuple does not change when a new tuple is added that is consistent with *DB*.

P3(a). If *id′* ∈ <*A,B*> for some *mip* of *DB′* but *{id,id′}* ⊄ *mip* for every *mip* of *DB′* then *u(id)=u′(id*).

Here the new tuple is inconsistent with *DB* but no *mip* of *DB′* contains both *id* and *id′*. So the inconsistency is not directly related to *id*. In this case the reliability of the tuple does not change.

P3(b). If *id′* ∈ <*A,B*> for some *mip* of *DB′* but *{id,id′}* ⊄ *mip* for every *mip* of *DB′* then *u(id)>u′(id*) unless *u(id)= 0* in which case *u′(id)= 0*.

In this case the unreliability of the tuple decreases relative to the whole database that has just become more inconsistent.

Clearly, P3(a) and P3(b) are incompatible.

P3(c) If there exists *id″* such that *{id,id″}* ⊆ <*A,B*> for some *mip* of *DB*, and *{id′,id″}* ⊆ <*A′,B′*> for some *mip* of *DB′* and *{id,id′}* ⊄ *mip* for any *mip* of *DB′* then *u(id)>u′(id).*

In this case the inconsistency relationship between *id* and *id′* is due to a third tuple in an indirect way. In a sense P3(c) is between P3(a) and P3(b). It states that the unreliability of a tuple decreases in this type of indirect inconsistency between *id* and *id′*.

P4. If *{id,id′}* ⊆ <*A,B*> for some *mip* of *DB′* then *u(id)<u′(id) unless u(id)= 1* in which case *u′(id)= 1*.

So if the inconsistency of a new tuple is related to *id* then the unreliability of *id* increases.

P5. The order in which the tuples are inserted makes no difference in the calculation of unreliability.

As we introduce a measure for unreliability we wish to give the user choices about how to do the calculation. Those choices are based on application requirements. For example, a tuple may be a member of several *mip*s; the more *mip*s there are the higher the unreliability should be. However, we may want to count not just the number of *mip*s but also the amount of inconsistency in each *mip*. Consider, for example the *DB1* introduced in Example 1. The inconsistency is due to the fact that

$$id_1[val] + id_2[val] = 50 > id_3[val] = 40.$$

Suppose we replace id_3 by id_4=<*c,r,[1,8],35*> and let *DB′= {id*$_1$*, id*$_2$*, id*$_4$*}*.

In the same way as before, choosing *A′= {id*$_1$*,id*$_2$*}* and *B′= {id*$_4$*}* produces the

minimal inconsistent pair $<A',B'>$, but now $id_1[val]+id_2[val]=50>id_3[val]=35$. The difference, *50–35,* is bigger than the difference, *50–40*. So, intuitively, the inconsistency of $<A',B'>$ should be greater than the inconsistency of $<A,B>$ even though there is only a single *mip* in both cases. . Recall now that an inconsistency is caused by the condition that $\sum_{i=1}^{n} v_{a\_i} - \sum_{j=1}^{m} v_{b\_j} > 0$. We will use the notation shorthand *A-B* for this difference and we call it the *size* of the inconsistency for that *mip*.

5.2 Methods to calculate unreliability

We now suggest several different ways of calculating the unreliability of a tuple. We use the following notation:

TotalMips	the number of *mip*s in *DB*
idTotMip	the number of *mip*s of which *id* is a member
MIP	is a user-defined maximal value for the number of *mip*s;
TotalDiffs	the sum of the sizes of the *mip*s in *DB*
idTotDiff	the sum of the sizes of the *mip*s for *id*
DIFF	is a user-defined maximum value for the sum of all sizes of the inconsistencies of the *mip*s.

In general we allow the user to determine the relative importance of the number of *mip*s that a tuple participates in and the sizes of the inconsistencies by choosing two nonnegative parameters α and β such $\alpha+\beta=1$.

Method 1 User defined maximal values.
Consider a tuple *id*. We define

$$u(id) = \alpha\, min(1, idTotMip/MIP)) \; + \beta(min(1, idTotDiff\,/DIFF)). \tag{1}$$

Next we show that this definition satisfies some of the rules given above.

Theorem 1 The definition of unreliability given in *(1)* satisfies the properties P1(a), P2, P3(a), P4, and P5 independently of the chosen parameters.

Proof P1(a). Clearly, $u(id)=0$ if and only if $idTotMip=0$ and $idTotDiff=0$ which is the case only if $id \notin mip$ for any *mip*.
P2. Adding an $id' \notin mip$ for any *mip* of *DB'* does not change any value in *(1)*.
P3(a). If $id' \in <A,B>$ for some *mip* of *DB'* but $\{id,id'\} \not\subset mip$ for any *mip* of *DB'* again, no value changes in *(1)*.
P4. If $\{id,id'\} \subseteq <A,B>$ for some *mip* of *DB'* then both *idTotMip* and *idTotDiff* increase when *u'(id)* is calculated, hence $u(id)<u'(id)$.
P5. Clear from the definition.

The previous method requires the user to choose maximum values. There is no such requirement for the other method.

Method 2 The values for the number of *mips* and the size of each are used separately.

This method is similar to Method 1; however, we use the maximal values from the *DB* itself. The formula corresponding to *(1)* is

$$u(id) = \alpha(idTotMip/TotalMips) + \beta(idTotDiff/TotalDiff). \quad (2)$$

This is a normalized definition. Now we show which rules are satisfied by this definition.

Theorem 2 The definition of unreliability given in *(2)* satisfies the rules P1(a), P1(b), P2, P3(b), and P5 independently of the chosen parameters.

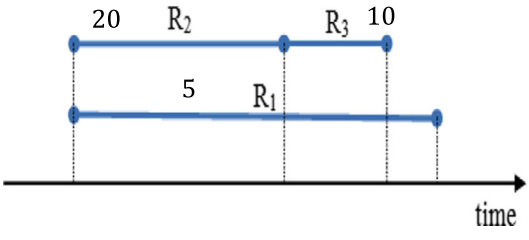
Proof:

P1(a). Same as for Method 1.
P1(b). Both fractions equal *1* and $\alpha+\beta=1$.
P2. Similar to the proof for Method 1.
P3(b). If $id' \in <A,B>$ for some *mip* of *DB′* but $\{id,id'\} \not\subset mip$ for any *mip* of *DB′*, then both *TotalMips* and *TotalDiffs* increase.
P5. Same as for Method 1.

Note that P4 need not hold because the new tuple may be in several new *mips*, in addition to the one with *id*, so *TotalMips* and *TotalDiffs* may increase substantially. Figure 3 shows an example of unreliability assessment using Method 2 for a simple inconsistent database with three tuples *R1, R2* and *R3.*

5.3 Internal versus external reliability

In general, we may consider two types of reliability: internal and external. The latter deals with issues outside of the database, such as prior knowledge concerning the reliability of the source providing the tuple. In this paper we deal only with internal reliability: how a tuple is involved in inconsistencies. Here we outline how both internal and external reliabilities could be used. If



TotalMips = 2
TotalDiffs = (20-5)+(10-5) = 20
R1TotMip = 2
R2TotMip = 1
R3TotMip = 1
R1TotDiff = 20
R2TotDiff = 15
R3TotDiff = 5
Alpha = 0.5
Beta = 0.5
u(R1) = 0.5(2/2)+0.5(20/20) = 1
u(R2) = 0.5(1/2)+0.5(15/20) = 0.625
u(R3) = 0.5(1/2)+0.5(5/20) = 0.375

Fig. 3 An example of unreliability evaluation

the external reliabilities for the tuples are predefined, then again we may allow a user to determine the importance of each. By using both reliabilities and nonnegative parameters γ and δ such that $\gamma+\delta=1$, we define

$$comb\_u(id) = \gamma(int\_u(id)) + \delta(ext\_u(id)).$$

In fact, if the external reliabilities are given first, these values may be used to compute the internal reliabilities, as long as it is made clear that this is not "pure" internal reliability. The way to use external reliabilities in this sense is to apply them in the computation of *idTotDiff*. We demonstrate how this can be done in the example *DB1* of Example 1. Suppose that the external unreliabilities are as follows: $eu(id_1)=.1$, $eu(id_2)=.5$, $eu(id_3)=.9$. So id_1 is more reliable than id_2 which is more reliable than id_3. Then when we compute the size of a *mip* for a tuple, we multiply it by the external unreliability of the tuple. So for id_1 the size will be $.1\times 10=1$, but for id_2 the size will be $.5\times 10=5$, and for id_3 the size will be $.9\times 10=9$.

A thorough study of the proper utilization of external reliability is outside of the scope of this paper.

6 Simulation-based study

We performed a simulation-based study to explore the behavior of the inconsistency and unreliability measures introduced in previous sections. For this study we used MATLAB 2014 running on MacBook Pro with 2.7 Ghz Intel Core i7 and 16GB 1600 MHz DDR3 memory. Figure 4 explains the study set up. We used report redundancy configuration with full subsumption, i.e., assuming that in the case of overlapping reports, the time interval of one report lies within the time interval of another report. Figure 4 (upper left part) shows an example of such a configuration with a report R1 subsuming reports R2 and R3. This configuration can be represented as a subsumption tree with the root at the bottom to follow the report configuration, as shown in Fig. 4 (upper right part). It is a binary tree (fanout =2) with one layer of non-root nodes. For each report corresponding to a root node in a subsumption tree (R1 in our example) we set a base reported value (700 in Fig. 4). Values of the subsumed reports (R2 and R3 in Fig. 4) were set as base value plus a step value; for example, the step value for R2 is 750–700=50. While performing the experiments we

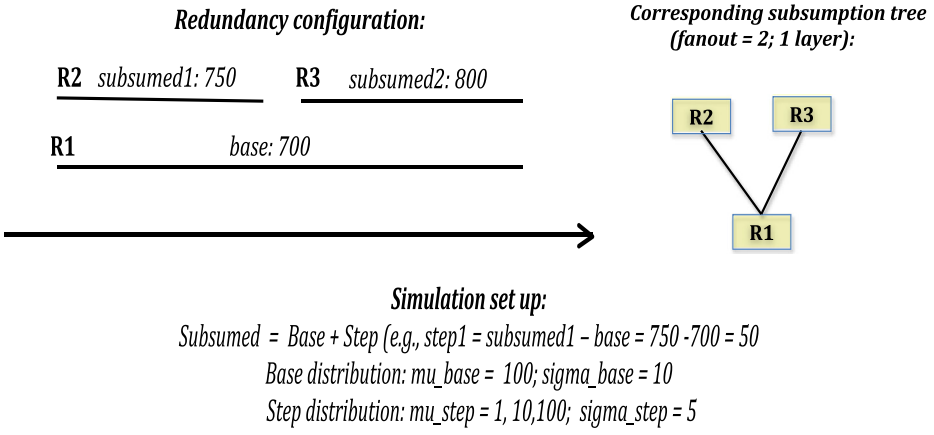


Fig. 4 Explanation of configurations for experiments

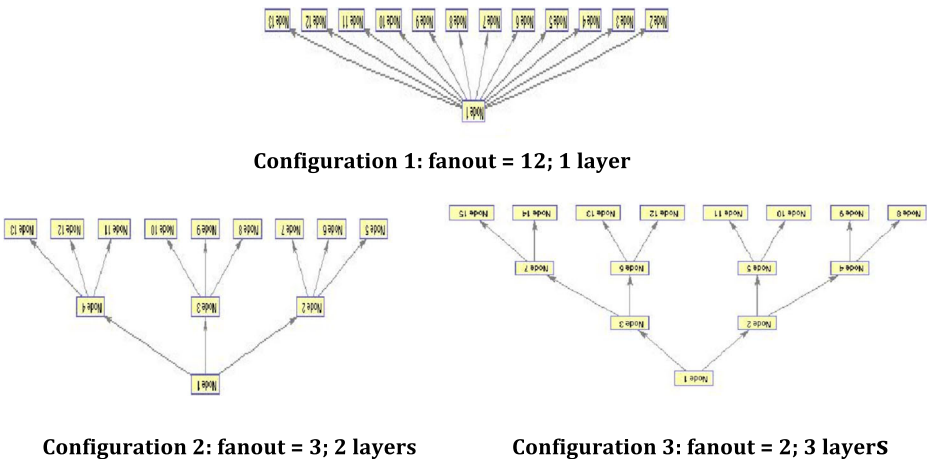


Fig. 5 Redundancy configurations for experiments

generated base and step values for different redundancy configurations using a normal distribution. Figure 4 (lower part) shows the parameters of the normal distributions that we used in our experiments.

Altogether we used the three redundancy configurations (subsumption trees) shown in Fig. 5. Each of the configurations is a report (tuple) subsumption tree with different fanouts and number of layers. Each root tuple was assigned a normally distributed value with mean of 100 and standard deviation of 10. Values of non-root tuples were generated adding a step value

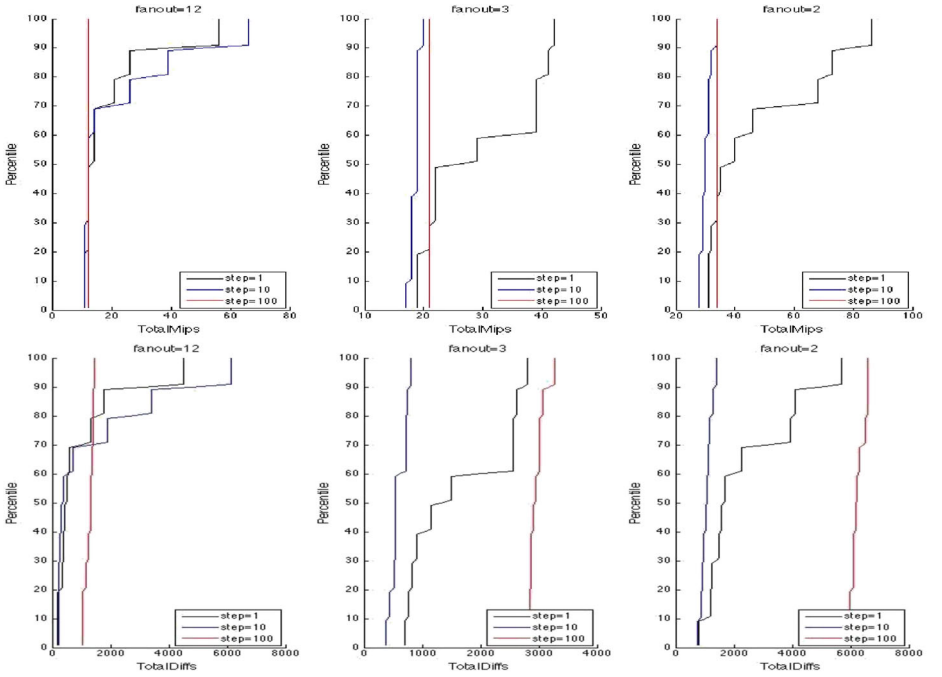


Fig. 6 Aggregate database inconsistency dynamics

to the root value. Step values were also distributed normally with mean ranging from 1 to 10 to 100 and with standard deviation 5. This way we introduced different probabilities of inconsistencies among the tuples: a larger step increases the probability of an inconsistency to occur. For each setting we ran 100 experiments to assess aggregate system dynamics.

Figure 6 shows the behavior of *TotalMips* and *TotalDiffs* measures for different configurations from Fig. 5 and different step distributions. As we observe from the percentile plots, the first configuration (*fanout= 12*) demonstrates very stable behavior for *step= 100.* This can be credited to the fact that each subsumed report combined with the root report forms a *mip*. Thus, practically each random run resulted in 12 *mip*s. For smaller steps we observe a higher number of *mip*s due to various combinations of reports contributing to longer *mip*s. Similar trends are observed for the other two configurations with a more stable behavior for *step =10,* which indicates that shorter *mip*s are generated due to step accumulation via several layers in the subsumption tree. This trend is consistent with *TotalDiffs* dynamics reflecting the impact of larger steps and multiple layers in the tuple subsumption hierarchy on the distribution of *mip* sizes.

Figure 7 illustrates the behavior of the two unreliability measures (Method 1 and Method 2 from Section 5.1) versus α and β parameters ($\alpha+\beta=1$) for different configurations and step distributions. For Method 1 (upper three plots) as we increase α (i.e., increasing the impact of the number of *mip*s and reducing the impact of their sizes) for configuration 1 (*fanout= 12*) we observe a sharp increase of unreliability for the smaller steps, which is consistent with our observations in Fig. 7, since smaller steps contribute a higher number of *mip*s due to a larger

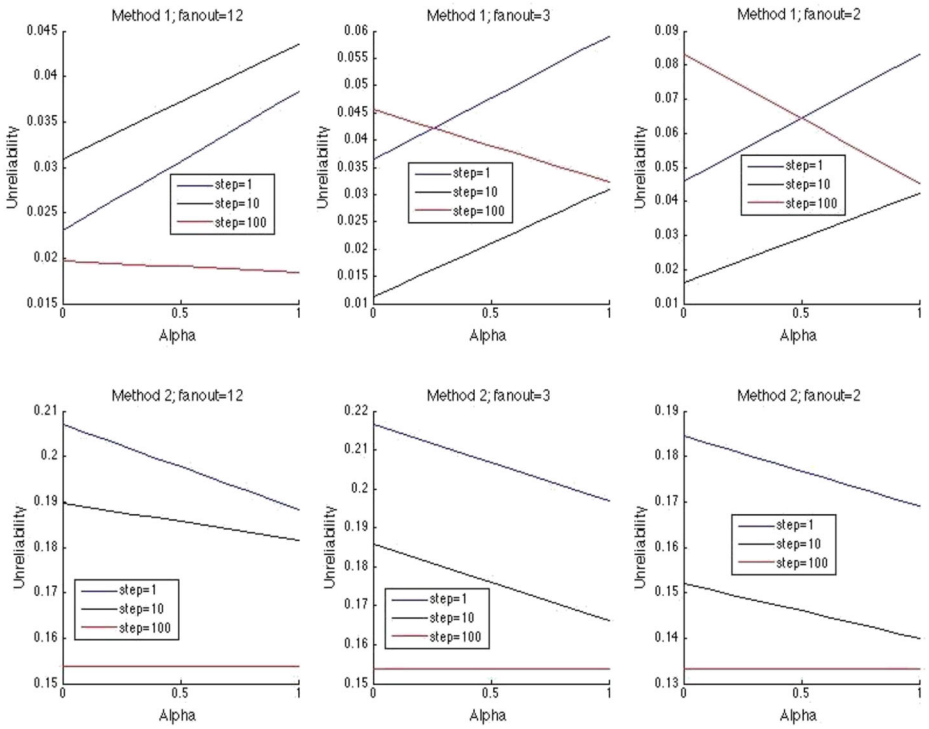


Fig. 7 Unreliability dynamics

number of reports involved in a higher number of longer *mip*s. Meanwhile, this trend is opposite for *step=100,* which becomes even more obvious in configurations with a larger number of layers. For Method 2 (lower three plots) the trends are quite different. Besides, the unreliability values are also smaller, leading to the conclusion that Method 2 is more tolerant of inconsistency and can be used in less sensitive unreliability scenarios. In future work we will explore the applicability limits of different unreliability assessment methods in more detail.

7 Conclusion

We formalized reliability assessment for historical data where redundancy and inconsistency are common. We performed integrated data reliability analysis exploring data redundancy and to discover data inconsistencies so as to provide automatic reliability assessment. We introduced a new approach to discover data inconsistency through the analysis of relationships between reports in the integrated database. First, we defined properties that a good measure of reliability should satisfy. We then proposed such measures and showed which properties they satisfy.

Our simulation-based study demonstrated trends in the behavior of different unreliability and inconsistency measures. We observed that different methods can be used and tuned up for application scenarios that vary with respect to their tolerance to data inconsistency and unreliability in an integrated information repository. We plan to explore the applicability limits of different unreliability assessment methods.

In future work we also plan to extend the proposed framework to handle more general cases of inconsistency and unreliability. This, in particular, includes situations with the subsumption of overlapping reports, as well as overlaps in other categories (such as any combination of time, space and names).

Acknowledgments This research was partially supported by NSF BCS-1244672 grant. We wish to thank the referees for many useful comments and suggestions.

Ethical statement

Funding This study was partially funded by NSF (grant number BCS-1244672).

Conflict of interest The authors declare that they have no conflict of interest.

References

Agarwal, S., Keller, A., Wiederhold, G., Saraswat, K. (1995). Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. *Proc. of International Conference on Data Engineering.*

Bertossi, L. (2006). Consistent Query Answering in Databases. *Proc. of ACM SIGMOD Record, 35*(2)

Bertossi, L.,& Chomicki, J. (2003). Query Answering in Inconsistent Databases. *Logics for Emerging Applications of Databases*, *Springer.*

Bleiholder, J., & Naumann, F. (2009). Data Fusion. *ACM Computing Surveys, 41*(1). doi:10.1145/1456650.1456651.

Bohannon, P., Flaster, M., Fan, W., & Rastorgi, R. (2005). A Cost-based Model and Effective Heuristic for Repairing Constraints by Value Modification. *Proc. of ACM SIGMOD.*

Brodie, M. (2010). Data Integration at Scale: From Relational Data Integration to Information Ecosystems. *Proc. of International Conference on Advanced Information Networking and Applications.*

Bry, F. (1997). Query Answering in Information Systems with Integrity Constraints. *Proc. of International Conference on Integrity and Control in Information Systems*.

Dong, X., & Naumann, F. (2009). Data Fusion—Resolving Data Conflicts for Integration. *Proc. of the VLDB Endowment, 2*(2).

Grant, J., & Hunter, A. (2013). Distance-Based measures of inconsistency. *Proceedings of ECSQARU, Lecture Notes in Artificial Intelligence* Vol. 7958. Springer.

Grant, J., & Hunter, A. (2011) Measuring consistency gain and information loss in stepwise inconsistency resolution. *Proceedings of ECSQARU, Lecture Notes in Artificial Intelligence* Vol. 6717. Springer.

Grant, J., & Hunter, A. (2006). Measuring inconsistency in knowledge bases. *Journal of Intelligent Information Systems, 27*, 159–184.

Grant, J. (1978). Classifications for inconsistent theories. *Notre Dame Journal of Formal Logic, 19*, 435–444.

Haas, L. (2007). Beauty and the Beast: The Theory and Practice of Information Integration. *Proc. of International Conference on Database Theory.*

Hunter, A., & Konieczny, S. (2005). Approaches to measuring inconsistent information. *Inconsistency Tolerance, Lecture Notes in Computer Science* Vol. 3300. Springer.

Hunter, A., & Konieczny, S. (2010). On the measure of conflicts: Shapley inconsistency values. *Artificial Intelligence, 174*, 1007–1026.

Imelinski, T., & Lipski, W. (1984). Incomplete information in relational databases. *Journal of ACM, 31*(4), 761–791.

Mu, K., Liu, W., & Jin, Z. (2011a). A general framework for measuring inconsistency through minimal inconsistent sets. *Knowledge and Information Systems: An International Journal, 27*, 85–114.

Mu, K., Liu, W., Jin, Z., & Bell, D. (2011b). A syntax-based approach to measuring the degree of inconsistency for belief bases. *International Journal of Approximate Reasoning, 52*, 978–999.

Staworko, S., & Chomicki, J. (2010). Consistent query answers in the presence of universal constraints. *Information Systems, 35*(1), 1–22.

Wijsen, J. (2009). Consistent Query Answering under Primary Keys: A Characterization of Tractable Queries. *Proc. of International Conference on Database Theory.*

Wijsen, J. (2005). Database repairing using updates. *ACM TODS, 30*(3), 722–768.

Zadorozhny, V., Manning, P., Bain, D., & Mostern, R. (2013). Collaborative for Historical Information and Analysis: Vision and Work Plan. *Journal of World-Historical Information*, *1*(1), 1–14.

Zadorozhny, V., & Hsu, Y.-F. (2011). Conflict-aware Fusion of Historical Data. *Proc. of the 5th International Conference on Scalable Uncertainty Management*.

Zadorozhny, V., Raschid, L., & Gal, A. (2008). Scalable catalog infrastructure for managing access costs and source selection in wide area networks. *International Journal of Cooperative Information Systems*, *17*(1).

Zadorozhny, V., & Raschid, L. (2007). Alternative path selection in resilient Web infrastructure using performance dependencies. *Journal of Web Engineering, 6*(2), 121–130.

Zadorozhny, V. Gal, A., Raschid, L., & Ye, Q. (2005). AReNA: Adaptive Distributed Catalog Infrastructure Based On Relevance Networks. *Proc. of International Conference on Very Large Data Bases*.