Data Quality Sensitivity Analysis on Aggregate Indicators

Mario Mezzanzanica, Roberto Boselli, Mirko Cesarini, and Fabio Mercorio
Department of Statistics, C.R.I.S.P. Research Centre, University of Milan Bicocca, Italy
{*mario.mezzanzanica, roberto.boselli, mirko.cesarini, fabio.mercorio*}*@unimib.it*

Keywords: Data Quality, Data Cleansing, Sensitivity Analysis, Inconsistent Databases, Aggregate Indicators, Uncertainty Assessment.

Abstract: Decision making activities stress data and information quality requirements. The quality of data sources is frequently very poor, therefore a cleansing process is required before using such data for decision making processes. When alternative (and more trusted) data sources are not available data can be cleansed only using business rules derived from domain knowledge. Business rules focus on fixing inconsistencies, but an inconsistency can be cleansed in different ways (i.e. the correction can be not deterministic), therefore the choice on how to cleanse data can (even strongly) affect the aggregate values computed for decision making purposes. The paper proposes a methodology exploiting Finite State Systems to quantitatively estimate how computed variables and indicators might be affected by the uncertainty related to low data quality, independently from the data cleansing methodology used. The methodology has been implemented and tested on a real case scenario providing effective results.

1 INTRODUCTION

Several studies, e.g. (Batini and Scannapieco, 2006; Strong et al., 1997; Redman, 1998), report that enterprise databases and public administration archives suffer from poor data quality, therefore data assessment and (in case) cleansing activities are required before data can be used for decision making. The best solution for improving data quality would be cleansing a database relying on the comparison with different (and more trusted) data sources. Unfortunately this is rarely feasible. An alternative solution is to collect real data and then to use it for comparison, however this approach may be extremely expensive. Data cleansing using rules derived from domain knowledge (business rules) is the most frequently adopted solution when the previously described approaches are not feasible. Business rules focus on consistency checking and corrective actions are implemented when an inconsistency is found. Such actions may modify, delete or add data to the existing databases.

Data quality is a general concept and it can be described by many dimensions, e.g., *accuracy*, and *consistency*, *accessibility*. The reader is referred to (Batini and Scannapieco, 2006)) for a complete survey. In this paper we focus on *consistency*, which (according to the previously cited survey) is a dimension describing the violation of semantic rules defined over a set of data items, where items can be tuples of relational databases.

Consider the dataset in Tab. 1 showing a cruise ship travel plan. The ship usually travels by sea, then stops at intermediate destinations (port of calls), making a *checkin* when entering the harbour and a *checkout* when exiting from it.

An example of low data quality is the missing departure date from Lisbon, since there is no *checkout* from the Lisbon's harbour.

ShipID	City	Date	Event Type
S01	Venice	12th April 2011	checkin
S01	Venice	15st April 2011	checkout
S01	Lisbon	30th April 2011	checkin
S01	Barcelona	5th May 2011	checkin
S01	Barcelona	8nd May 2011	checkout
…			

Table 1: Travel Plan of a Cruise Ship

Such inconsistencies should be fixed before using data for decision making activities. The departure from Lisbon might be set on May 3rd, e.g. by observing that the ship usually stays in the harbour for 3 days. Of course, there is no certainty of having guessed the real value (unless the real and correct data can be obtained in some way), the ship could have departed either on the 1st, or on the 2nd, or on May the 4th, etc.

Focusing on an indicator like *active travel days* (i.e., the number of day spent by the ship on the sea), missing dates create uncertainty, their impact on the indicator should be carefully evaluated when a lot of wrong dates are detected and (important) business decisions will be based thereon. Hereafter the term indicator will be used to refer to any kind of aggregate information derived from database contents.

The work presented in this paper describes a methodology (focusing on data consistency check based on formal methods) for executing a sensitivity analysis on indicators computed on possibly inconsistent data.

1.1 Related Work

Data quality is a large and complex domain as reported in (Batini et al., 2009). The work presented in this paper can be framed on the data cleansing field (also called data cleaning or scrubbing) which deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data (Rahm and Do, 2000).

Works on *repair* focus on finding a consistent database and minimally different from the original one, however the authors of (Chomicki and Marcinkowski, 2005) state that computational intractability affects algorithms used for performing minimal-change integrity maintenance.

Consistent query answering approaches focus on finding consistent answers from inconsistent databases, whereas an answer is considered consistent when it appears in every possible repair of the original database (Arenas et al., 1999). The authors of (Arenas et al., 2003) investigate how to compute the range of possible value results that an aggregate query can return from an inconsistent database.

The latter work shows a lot of similarities with the one presented in this paper: both focus on evaluating the range of possible values that an aggregate indicator may assume. However (Arenas et al., 2003) focuses only on the SQL aggregate functions (MIN, MAX, COUNT, SUM, and AVG) while the approach described in this paper can be easily generalised to a broader set of indicators/aggregate functions. Furthermore, as the authors state, the computation of aggregate queries in such context is an intractable problem when two or more functional dependencies are used (Arenas et al., 2003). Unfortunately this is the case of many real world domain consistency problem (as the one described in Sec. 4).

In (Embury et al., 2001) the authors identify data integration conflicts exploiting a formal model built upon rules derived from domain knowledge. Integrity checking and repairing techniques are used to detect the data violating the model. However the authors outline that exploiting the latter technique for repairing a complete database (and not a single transaction) requires to look for repairs on a very large combinatorial search space since transaction analysis cannot be used to prune it.

Many cleansing tools and database systems exploit integrity analysis (including relational integrity) for identifying errors. While data integrity analysis can uncover a number of possible errors in a data set, it does not address complex ones (Maletic and Marcus, 2000).

Some research activities, e.g. (Fan et al., 2008), focus on expanding integrity constraints paradigms to deal with a broader set of errors by introducing conditional functional dependencies, i.e. functional dependencies holding only on a subset of tuples identified by a specific condition. However conflicts may arise among the several dependencies, and their discovery may be an intractable problem. Furthermore constraints checking that involves different tuples (e.g. like the examples showed in Tab. 1 and in Sec. 4) can result in a very complex and large set of conditional functional dependencies.

In (Arasu and Kaushik, 2009) “augmented context free grammars” are used to extract information from attributes, and to subsequently cleanse data. Such approach mainly focuses on the attribute level, whilst the work presented in this paper focuses on set-of-events semantics and consistency.

“Learning based methods” can be used for data cleansing. Possible techniques and approaches are: unsupervised learning, statistical methods, data profiling tools exploiting integrity constraints, range and threshold checking, pattern recognition, clustering methodologies, and rule discovery from sequential data (Mayfield et al., 2009; Sang Hyun et al., 2001). Such techniques may guess wrong information e.g. (the next is an overstated example), they might erroneously conclude that double entries are fine in a database containing a lot of incorrect double entries. These methods can improve their performance in response to human feedbacks. However the model that is built during the learning phase by these techniques can’t be easily accessed by domain experts. In this paper we explore a different approach where consistency models are explicitly built and validated by domain experts.

Correction steps (also known as imputation or data editing) are performed in the statistical domain without altering the collected data statistical parameters (Fellegi and Holt, 1976). Not altering the statistical parameters is required to make inferences from the sample data about the whole population. However, in a scenario where datasets covering the whole population are available (e.g. public administration databases), inference is no more a strong requirement, whilst data cleansing is paramount.

Many data cleansing toolkits have been proposed for implementing, filtering, and transforming rules over data. A survey can be found in (Galhardas et al., 2000; Müller and Freytag, 2003). A detailed survey of those tools is outside the scope of the paper, here it is enough to summarise that they implement in several ways the theoretical approaches described in this section.

1.2 Contribution

The contribution of our work is twofold: from one side we propose a methodology to quantitatively estimate the impact that uncertainty (due to low data quality) can have on indicators computed on cleansed data. On the other side this methodology has been implemented and validated against a real-world domain application.

For the sake of clarity, in Sec. 2 we briefly introduce how formal methods can be used to verify database consistency. Sec. 3 describes the proposed methodology to perform sensitivity analysis on indicators derived from low quality databases, while in Sec. 4 a real scenario is introduced and the results obtained are outlined. Finally, Sec. 5 draws some conclusion and outlines the future work.

2 DATA CONSISTENCY THROUGH FORMAL METHODS

The idea behind this paper draws on the work of (Mezzanzanica et al., 2011) where a database content is mapped onto a set of events (called Finite State Event Database), a *formal model* modeling the consistent evolution of data is built, and the data is validated against the model by using model checking techniques. Generally speaking, an *event* (or transition) in a system represents a change that may occur in the system state as a consequence of an action. In (Mezzanzanica et al., 2011) the authors map a database record onto an event and the attributes of the record onto the event information. As a consequence, consistency verification can be turned into a model checking problem, which allows one to exploit the formalisation and the computational power of Model Checking techniques. Due to the space limitation we cannot deeply describe the details of their techniques. However, since in the paper we often refer to the concept of Finite State Event Database, we formalise the following:

Definition 1 (Finite State Event Database). *Let $E = \{e_1, \ldots, e_n\}$ be a finite set of events, and let $\lesseqgtr$ a* total order *operator such that $e_1 \lesseqgtr e_2 \lesseqgtr \ldots \lesseqgtr e_n$, then a* Finite State Event Dataset *(FSED) is a dataset S whose content is an $\lesseqgtr$-ordered sequence of events $\varepsilon = e_1, \ldots, e_n$. Then, a Finite State Event Database (FSEDB) is the union of FSEDs as $\bigcup_{i=1}^{k} S_i$ where $k \geq$* 1. *By abuse of notation we also denote $\varepsilon = e_1, \ldots, e_n$ as* FSED.

The following example should clarify the matter. Let us consider the Cruise Ship Travel Plan as presented in Table 1, and let us focus on the *overall cruise duration* indicator. In this case, given a set of events E, a generic event $e_i \in E$ may be mapped to the attributes *ShipID*, *City*, *Date*, and *Event Type*, namely $e_i = (ShipID_i, City_i, Date_i, EType_i)$. Moreover, the binary operator $\leq$ defined over the event's attribute *Date* is used to generate an *ordered* sequence of events, that is $\forall e_i, e_j \in E, e_i \leq e_j$ iff $Date_{e_i} \leq Date_{e_j}$. Finally, a simply consistency property could be *"if a ship checks in to a harbour A, then it must check out before checking in to a different harbour"*. Fig. 1 shows an automaton modelling our example. The system state is composed by the *pos* variable (only one) which describes the ship's position (i.e., *sea* when the ship is travelling, *harbour* when it is in a port of call). Consistency verification for this example can be turned into finding out if an *event-flow-compatible-path* does exist in the automaton of Fig. 1 e.g., *checkin*, *checkin* is not a feasible path for the system.

In this paper we perform a sensitivity analysis (see Sec. 3) on the impact of possible corrections in scenarios similar to the one just introduced.

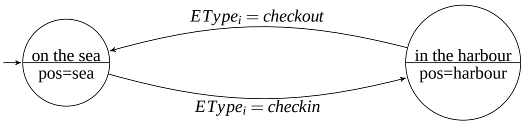


Figure 1: The Automaton for the Travel Plan of a Cruise Ship example.

3 DATA QUALITY SENSITIVITY ANALYSIS

This Section will outline how to carry out indicator sensitivity analysis on Finite State Event Databases.

It is worth noting that a Finite State Event Database is made by one or more Finite State Event Dataset. Data consistency is checked, and for each inconsistent dataset two cleansed versions are derived which respectively minimise and maximise a reference indicator. The analysis is allways performed having an indicator as a reference. Cleansing is not always deterministic, since several cleansed and consistent versions can be derived from an original inconsistent dataset. In the cruise example of Tab. 1, the departure date from Lisbon can be set on any date from 30^{th} April 2011 to 5^{th} May 2011 (for simplicity it is not considered the ship travel time which would narrow the choice).

Informally speaking, the sensitivity analysis on the indicator *active traveldays* is carried out as follows. Two timetables are selected among the set of possible corrections: the former where the missing departure date is set on the 30^{th} April 2011 (this choice maximises the travel days) and the latter where the departure date is set on the 5^{th} May 2011 (this choice minimises the travel days). The two datasets are labelled respectively upper bound case and lower bound case cleansed dataset (UB and LB case respectively).

In our example one intervention is enough to make the sequence consistent, i.e., inserting the event $e_4 = (S01, Lisbon, X, checkout)$ where $X \in [30\text{th April } 2011, \ldots, 5\text{th May } 2011]$. Hence, the Lisbon to Barcelona travel days value (used to compute the overall *active traveldays*) can range between 1 and 6 days.

In this example each ship timetable is a Finite State Event Dataset, and for each of them an upper and lower bound case is generated. For simplicity an upper and lower bound case is generated also for consistent timetable, in this case they are both equals to the original one. The average values of respectively the UB and LB cases (of all the ships travel plans) are computed and then compared. The gap between the two average values tell us how the indicator is sensible to the uncertainty caused by the missing data.

This approach cannot discover all the differences among the database data and the "real world data", indeed a completely missing sequence of data can't be detected by consistency checking. Nevertheless, in the example reported in Tab. 1 the sensitivity analysis based on UB and LB case derivation can provide useful information to quantitative estimate the impact that uncertainty may have on aggregate indicators.

3.1 Exploring the Datasets Tree

We introduce $ccheck(FSED)$: a function that checks the consistency of a Finite State Event Dataset (or a subset thereof). In case of inconsistency the function returns a pair composed by (1) the error code (describing the error type) and (2) the index identifying the minimal sequence of events being consistent. The functions is implemented by making use Model Checking, analogously to the work described in (Mezzanzanica et al., 2011).

Now we are in state to introduce how the UB (upper bound) and LB (lower bound) case cleansed dataset are generated. For each sequence where an inconsistency is detected, the event subsequent to the detection point is labelled as Consistency Failure Point (CFP). The CFP event is not necessarily the responsible of the consistency failure, but it is the point where the failure emerges.

For the sake of clarity, the semantics of *ccheck* and *CFP* are formally defined as follows.

Definition 2 (*ccheck*). *Let S be a FSED and let* $\varepsilon = e_1, \ldots, e_n$ *be a sequence of events according to Definition 1, then* $ccheck : FSED \rightarrow \mathbb{N} \times \mathbb{N}$ *returns the pair* $< i, er >$ *where:*

1. *i is the index of a minimal subsequence* $\varepsilon_{1,i} = e_1, \ldots, e_i$ *such that* $\varepsilon_{1,i+1}$ *is inconsistent while* $\forall j : j \leq i \leq n$*, the subsequences* $\varepsilon_{1,j}$ *are consistent.*
2. *er is zero if* $\varepsilon_{1,n}$ *is consistent, otherwise it is a natural number which uniquely describes the inconsistency error code of the sequence* $\varepsilon_{1,i+1}$*.*

Then, we can define CFP_S *as the index of the event* after *the shortest consistent subsequence. By abuse of notation, we denote the index i of the pair returned as* $first\{ccheck(S)\}$ *whilst* $second\{ccheck(S)\}$ *denotes the element er. According to this notation* $CFP_S = first\{ccheck(S)\} + 1$.

The error code previously introduced is used to select the corrective actions to be executed. The semantic of the error codes and the implementation of corrective actions are domain dependent. Indeed, one can insert new events, delete, or change existing ones or a combination thereof. For each CFP several corrective actions can be implemented to fix the inconsistency, generating in this way several cleansed versions of the original dataset. For each of them, the consistency check has to be repeated again, since other CFPs can be detected in the remainder of the sequence. The process is executed recursively until a (most probably very large) set of completely cleansed datasets are generated. The case that maximises (minimises) the reference indicator is selected as the UB (LB) case,

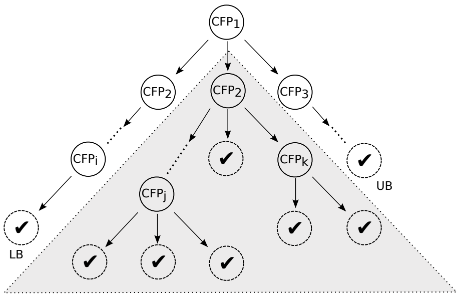


Figure 2: Example of dataset correction tree exploration. Leaf nodes represent a cleansed generated dataset whilst non-leaf nodes represent datasets where CFPs still need to be addressed.

the remaining cases are discarded. An example of this process is showed in Fig. 2.

The tree of cleansed datasets showed in Fig. 2 can grow exponentially, thus making the problem intractable. Nevertheless, several simplifications both domain independent and dependent can be used to make the algorithm more scalable. An example is provided in Subsec. 3.2. More precisely, a Corrective Event for a CFP can be defined as follows.

Definition 3 (Corrective Event for a CFP)**.** *Let S be a FSED whose content is a sequence of events $\varepsilon = e_1,\dots,e_n$ according to definition 1. Moreover, let CFP_S be a Consistency Failure Point for S. Then e_j is a* corrective event *for CFP_S if the new dataset $S' = S \cup \{e_j\}$ satisfies $CFP_{S'} > CFP_S$. That is either $\varepsilon' = e_1,\dots,e_j,\dots,e_i,\dots,e_n$ is consistent (i.e., $first\{ccheck(S')\} = n$) or S' has a $CFP_{S'}$ that appears later than CFP_S.*

It is worth noting that some domains may require to define a *set* of corrective events for a CFP (i.e., more than one event could be required to fix an inconsistency). To this regard, the Def. 3 can be easily adapted to deal with a set of corrective events instead of a single one.

The process of deriving a consistent sequence of events from an inconsistent one is largely domain dependent. However some general practice can be identified. Fixing an inconsistency may require to add or delete events. Focusing on adding events to a sequence, a trivial implementation could be to try adding events in all the possible places and then to check for each attempt if the resulting sequence has been fixed, i.e. if the new sequence is consistent or a new CFP is found after the original one (who disappeared). Another common techniques (which is used when the time dimension has to be estimated for the event to be inserted) is to try placing an event at the beginning and at the end of a time slot (this is the approach used for estimating the cruise boat UB and LB departure date in the example previously described).

Now it will be illustrated how the sensitivity analysis is performed and how the possible correction tree is explored (an example of correction tree is depicted in Figure 2), the pseudo-code of the sensitivity algorithm is split in Algorithms 1 and 2 for the sake of clarity.

Algorithm 1 focuses on a FSEDB S and, for each inconsistent FSED S_i, it calls Algorithm 2 to cleanse the dataset (lines 6-8). The Algorithm 2 takes as input the dataset S_i and its CFP_{S_i}. Then it generates the possible instances S'_i as a result of all the possible intervention activities. The consistency of each generated S'_i is checked as follows:

- S'_i is completely consistent (line 4). In this case it is added to $S_i^{consistent}$.
- S'_i is still inconsistent but a new emerged CFP occurs *later* than the previous one. That is the previous inconsistency has been fixed, but a new one has emerged in the remainder of the sequence (line 6). Then the algorithm recursively calls SENSITIVITY on the new instance S'_i trying to fix it.
- S'_i is still inconsistent but the correction has generated a new CFP before the previous one (i.e., the corrective action has made the situation worse). In this case the corrective action is discarded.

At the end of this computation, the set $S_i^{consistent}$ contains all the consistent instances generated for S_i (i.e., the leaf nodes of Figure 2). Then, Algorithm 2 chooses the LB (UB) cleansed version by looking for the lower (upper) indicator value (lines 11-12).

Algorithm 1 MAIN ALGORITHM

```
Input: $S$ //A FSEDB according to Def 1
       $ccheck$ //According to Def 2
       $ind$ //An indicator satisfying Property 1
1:  for all $S_i \in S$ do
2:    $lower_{S_i} \leftarrow \emptyset$; //global lower value for $S_i$
3:    $upper_{S_i} \leftarrow \emptyset$; //global upper value for $S_i$
4:    $S_i^{consistent} \leftarrow \emptyset$; //global set of consist. $S_i$
5:    $CFP_{S_i} \leftarrow first\{ccheck(S_i)\} + 1$;
6:    if ($CFP_{S_i} \leq n$) //$S_i$ is not consistent then
7:      if (do not use the simplification of Sec. 3.2) then
8:        SENSITIVITY($S_i$,$CFP_{S_i}$);
9:      else
10:       SENSSIMPLIFIED($S_i$,$CFP_{S_i}$,LB case);
11:       SENSSIMPLIFIED($S_i$,$CFP_{S_i}$,UB case);
```

3.2 An Algorithm Simplification

A simplified version of the previously mentioned algorithm can be used if two conditions are met on both

Algorithm 2 SENSITIVITY

```
Input: $S_i, CFP_{S_i}$
for all $a_i$ corrective action for $S_i$ do
  $S'_i \leftarrow$ apply($a_i, S_i$);
  $CFP_{S'_i} \leftarrow first\{ccheck(S'_i)\} + 1$;
  if $(CFP_{S'_i} = n+1)$ then
    $S_i^{consistent} \leftarrow S_i^{consistent} \cup S'_i$;
  else if $(CFP_{S'_i} > CFP_{S_i})$ then
    SENSITIVITY($S'_i$,$CFP_{S'_i}$);
  else
    //Discard the corrective action
for all $S'_i \in S_i^{consistent}$ do
  $lower_{S_i} \leftarrow eval\_lower\_ind(S'_i, ind)$;
  $upper_{S_i} \leftarrow eval\_upper\_ind(S'_i, ind)$;
```

Algorithm 3 SENSSIMPLIFIED

```
Input: $S_i, CFP_{S_i}, modality$
$S_i^{LB} \leftarrow \emptyset$;  //the $S_i$ instance minimising ind.
$S_i^{UB} \leftarrow \emptyset$;  //the $S_i$ instance maximising ind.
for all $a_i$ corrective action for $S'_i$ do
  $S'_i \leftarrow$ apply($a_i, S_i$);
  $CFP_{S'_i} \leftarrow first\{ccheck(S'_i)\} + 1$;
  if $(CFP_{S'_i} = n+1 \vee CFP_{S'_i} > CFP_{S_i})$ then
    if $S'_i$ minimises the indicator then
      $S_i^{LB} \leftarrow S'_i$;
      $lower_{S_i} \leftarrow eval\_lower\_ind(S'_i, ind)$;
    else if $S'_i$ maximises the indicator then
      $S_i^{UB} \leftarrow S'_i$;
      $upper_{S_i} \leftarrow eval\_upper\_ind(S'_i, ind)$;
  else
    //Discard the corrective action
if ($modality$ = LB case $\wedge$ $S_i^{LB} \neq \emptyset$) then
  $CFP_{S_i^{LB}} \leftarrow first\{ccheck(S_i^{LB})\} + 1$;
  SENSSIMPLIFIED($S_i^{LB}$,$CFP_{S_i^{LB}}$, modality);
else if ($modality$ = UB case $\wedge$ $S_i^{UB} \neq \emptyset$) then
  $CFP_{S_i^{UB}} \leftarrow first\{ccheck(S_i^{UB})\} + 1$;
  SENSSIMPLIFIED($S_i^{UB}$,$CFP_{S_i^{UB}}$, modality);
```

indicator and correction activities: 1) the sequence being evaluated can be partitioned into non-overlapping subsequences which fully cover the original sequence (e.g., like the subsequences $\varepsilon_1, \varepsilon_2, \ldots$ in Fig. 3); 2) the computation of the indicator being evaluated can be parallelised over the subsequences identified in the previous step. The latter is feasible when 2a) the indicator is an additive function with respect to the subsequences; and 2b) the corrections carried out within a subsequence do not affect the corrections executed on the other subsequences.

If all the two conditions hold then the inconsistency fixing and the indicator computation tasks can be independently executed within the boundaries of each identified subsequence, thus greatly reducing the computational effort. The condition 2a holds under the hypothesis showed below:

Property 1 (Additive Function)**.** *Let $\varepsilon = e_1, \ldots, e_n$ be a FSED (according to Def. 1) which can be partitioned into event subsequences such that $\cup_{i=1}^{m} \varepsilon_i = \varepsilon$, and $\forall i, j$ with $i \neq j$, $\varepsilon_i \cap \varepsilon_j = \emptyset$, then a function $F : FSED \rightarrow \mathbb{Z}^+$ is* additive *if $F(\varepsilon) = \sum_{i=1}^{m} F(\varepsilon_i)$ with $m < n$.*

In order to define the next properties, we introduce a relation between subsequences $\varepsilon_1, \varepsilon_2$ such that $\varepsilon_1 < \varepsilon_2$ iff $\forall e_i \in \varepsilon_1, e_j \in \varepsilon_2, e_i < e_j$.

Then condition 2b can be rephrased as follows: cleansing a subsequence ε_i has either no impact both on the previous (i.e. the antecedents) and on the following (i.e. the subsequent) subsequences, or every possible corrections inside a subsequence has the same impact on the remaining future subsequences. These can be summarised by saying that a cleansing intervention should satisfy the properties "Preserving a common future" and "not altering the past" more formally defined below.

Property 2 (Preserving a common future)**.** *Every correction of a CFP found inside a subsequence ε_i must share a common future in all the subsequences ε_k where $\varepsilon_k > \varepsilon_i$. Note that this condition can be easily satisfied if the FSS modelling the FSED is a memoryless systems (Csiszar and Körner, 1981) e.g., a Markov process (Iosifescu, 1980). Under this hypothesis, all the corrections done on a subsequence will have a common future, if all of them will lead to the same final state at the end of the sequence.*

Property 3 (Not altering the past)**.** *Every correction of Consistency Failure Points (in ε_i) will not modify prior subsequences ε_p where $\varepsilon_p < \varepsilon_i$.*

Claim 1 (Pruning Conditions)**.** *If Properties 1,2 and 3 hold for an indicator to be evaluated on a FSE Dataset, then the UB / LB case search can exploit a "divide et impera" paradigm (i.e. it can be computed independently on each ε_i).*

Proof. Let ε_0 being a subsequence of S from the beginning to the first CFP met. Only the cleansed dataset of ε_0 that minimises the indicator (computed on ε_0) need to be expanded. Indeed Properties 2 and 3 ensure that not overlapping subsequences can be cleansed independently, therefore the cleansed solution that minimises the global indicator must contain the cleansed version of ε_0 that minimises the indicator locally. To this regard, suppose, without loss of generality, that the cleansed dataset that minimises the indicator computed for ε_0 is the leftmost child of the top node in Fig. 2. All its siblings (successors of the top node) and their related subtrees can therefore be pruned.

Going further, let us suppose that a second CFP is found. Let ε_1 be the sequence of events between the first and the second CFP. Suppose, that the cleansed version of ε_1 that minimises the indicator computed locally is the leftmost successor node. This is the only node that needs to be expanded, all its siblings can be pruned. Without loss of generality, we can identify the cleansed version of S_i that minimises the indicator as being always the leftmost node. Hence, the completely cleansed dataset that minimises the indicator is the leftmost leaf, and the path to reach it goes through all the leftmost nodes of the tree. Therefore, the path to the cleansed dataset that minimises the indicator is the path where each indicator computed locally on ε_i is minimised. Note that the cleansed version of S_i that maximises the indicator can be obtained in a specular way. More generally, every node of the leftmost (rightmost) path is the cleansed version of S_i that minimises (maximises) the indicator. If the CFPs location would be known a priory, the cleansed version of every S_i could be computed in parallel. □

3.2.1 The Indicator Estimation

The simplified algorithm can be applied only to the datasets where the corrections don't violate properties 2 and 3. Conversely, a dataset violating properties 2 and 3 requires to explore all the feasible corrected instances in search of the UB and LB values. Unfortunately, this can lead to computational problems. To avoid this, a coarse-grained estimation of the indicator can be executed, namely the *estimation* approach. The *estimation* approach does not try to make the sequences consistent but estimates the UB and LB values with a trivial method (domain dependent) requiring a reasonable effort but at the price of providing a less precise estimation (i.e., enlarging the value bounds). For the sake of clarity the two approaches will be called respectively *computation* approach and *estimation* approach.

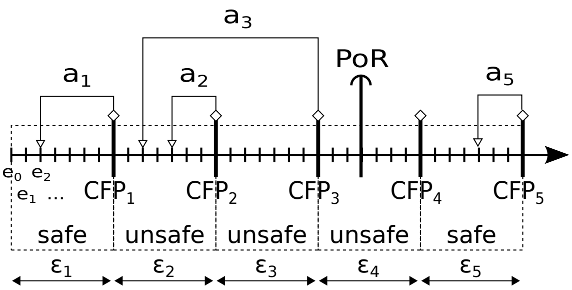


Figure 3: An example of correction violating the property "not altering the past".

Consider the sequence of events reported in Fig. 3 which can be split into subsequences bounded by CFPs (e.g., ε_1, ε_2, ..., ε_5). The *computation* and the *estimation* approaches can be jointly used, as described in what follows (the analysis focuses on computing the LB indicator value, the UB value can be obtained specularly).

1) The *computation* approach is executed on the subsequences and the results are investigated. The subsequences where the *computation approach* can be applied are marked as *safe* whilst the others are marked as *unsafe*.
2) In subsequence ε_1 action a_1 is the correction (out of several others) that minimises the indicator value in the subsequence ε_1. All the corrections of CFP_1 do not violate neither the property 2 nor the property 3, therefore this subsequence is marked as *safe*.
3) Focusing on ε_3 (the over next subsequence) the correction of CFP_3 using action a_3 violates the property 3 (not altering the past). Consequently the scenario where the correction a_2 had been previously selected and executed in ε_2 has changed and action a_2 might be no more the optimal choice for the CFP_2. The subsequence ε_2 is marked as *unsafe* and is no more investigated, i.e. no effort is spent for finding a corrective action that might fix ε_2 and CFP_2. The uncertainty that CFP_3 creates in its past can make very difficult identifying the correction that minimises the indicator value in ε_3 (a_3 may result not being anymore the best choice), therefore ε_3 is no more investigated and it is marked as unsafe too.
4) The uncertainty originated by ε_3 will propagate in the remainder sequences until a *point-of-reset* is found, i.e. an event from which the simplifying conditions are satisfied again (properties 2 and 3) and ends the uncertainty created by unsafe subsequences. Looking at Fig. 2 this will happen when all the nodes of a sub-tree (that can't be pruned) will share a common future from the point-of-reset onward, therefore it is enough to evaluate only one sub-subtree since all the other siblings are similar. The criteria for identifying points-of-reset are domain dependent, however as a general rule, good candidates are points for which a common future holds for every possible prior correction.
5) Subsequence ε_4 is marked as *unsafe* because the events before the *point-of-reset* are affected by uncertainty too, whilst ε_5 is marked as *safe* since it doesn't violate neither the property 2 nor the property 3 and no uncertainty propagates from previous subsequences.
6) Subsequences violating the property 2 can be managed similarly: the subsequences will be marked as *unsafe* until a *point-of-reset* is found. An example of such error is showed in Fig. 5.

7) The indicator LB value is computed according to the *computation* approach for the safe subsequences, whilst the *estimation* approach is used for unsafe ones. We recall that the indicator value can be computed for the whole sequence by summing the values computed in each subsequence (i.e. the indicator is addictive).

In a similar way The UB and LB values are computed for each FSED composing the FSEDB. Then, the average values of all the $FSED$ UB and LB values are computed and the gap is analyzed as described in the example of Sec. 4.

The methodology introduced in this paper can be used for estimating an indicator possible value ranges on an inconsistent dataset. It is worth noting that the term *indicator* refers to a general and domain-dependent concept (e.g., they are widely used in many fields as statistics, economics and business intelligence). To give an example, in the context of data quality an indicator can be used to measure and improve the quality of the data (Wang et al., 1993) as well as to assess the reliability of the results arising from the data uncertainty (Weidema and Wesns, 1996). Roughly speaking, an indicator is a function able to represent or manipulate data so that operators can assess and gain confidence with data (Weidema and Wesns, 1996). As a consequence, an indicator should have the same characteristics of a function, e.g. being meaningful for the domain, having a reasonable computation complexity, etc.

every possible inconsistency there is a finite set of possible corrections). Moreover, if the indicator satisfies the properties 1, 2, and 3 then the simplified algorithm described in this subsection can be applied.

4 THE CASE OF “THE WORKERS CAREER ADMINISTRATIVE ARCHIVE”

The Italian labour law states that every time an employer hires or dismisses an employee, or a contract of employment is modified (e.g. from part-time to full-time, or from fixed-term to unlimited-term) a communication (Mandatory Communication) has to be sent to a job registry. The registry is managed at *provincial level* for several administrative tasks, every Italian province has its own job registry recording the working history of its inhabitants (as a side effect). In this scenario, the job registry data is analysed to provide information to civil servants and policy makers. See (Martini and Mezzanzanica, 2009) for further details.

For each worker, a mandatory notification (representing an *event* in our context) contains several data:

w_id: it represents an id identifying the person involved in the event;

e_id: it represents an id identifying the communication;

e_date: it is the event date;

e_type: it describes the event type occurring to the worker career. The event types are: the *start* or the *cessation* of a working contract, the *extension* of a fixed-term contract or the *conversion* from a contract type to a different one;

c_flag: it states whether the event is related to a full-time (FT) or a part-time contract (PT);

empr_id: it is used to uniquely identify the employer involved in the event.

The evolution of a consistent worker’s career along the time is described by a *sequence* of events ordered with respect to *e_date*. More precisely, in this settings the FSED is the ordered set of events for a given *w_id*, whose union composes the FSEDB. Then, *consistency* is related to the “correct” evolution of a worker career, which can be inferred by the Italian Labour Law and common practice. To this regard, an employee can have only one full-time contract active at a the same time; or an employee can not have more than two part-time contracts (signed with different companies).

For the sake of simplicity, we omit to describe some trivial constraints which can be derived from the ones above (e.g., a employee cannot have a *cessation* event for a company for which he does not work, an event can not be recorded twice, etc).

4.1 The Formal Model

The first step when dealing with FSS is the definition of the *system state*, which in our settings represents the worker career at a given time point.

It is composed by two elements: the list of companies for which the worker has an active contract ($C[]$) and the list of modalities for each contract ($M[]$). To give an example, $C[0] = 12, M[0] = PT$ models that the worker has an active part-time contract with company 12.

The FSS describing a consistent career evolution is showed in Figure 4. Note that, for the sake of clarity, we omit to represent *conversion* events as well as inconsistent states/transitions (e.g., the worker activating two full-time contracts), which are considered in the consistency verification.

To give an example, a consistent career can evolve signing a part-time contract with company i, then activating a second part-time contract with company j, continuing by closing the second part-time and then reactivating the latter again (i.e., $unemp, emp_i, emp_{i,j}, emp_i, emp_{i,j}$).

4.2 The Sensitivity Analysis

In this section we describe how the Algorithm presented in Sec. 3 is applied to the domain just introduced. For the sake of clarity, we map the domain topics on the concepts introduced in Sec. 3.

- An FSE Dataset is a sequence of Mandatory Communications representing a worker career. Hence the FSE Database is a collection of careers. We refer to a worker career as S_i.
- $ccheck(S_i)$ is a function that implements the FSS depicted in Fig. 4 according to Def. 2. The function checks for career consistency and returns both (1) the CFP and (2) the *error code* if an inconsistency is found.
- $ind(S_i)$ is an indicator function which returns the number of worked days for the career S_i.

The consistency check showed that several errors affect the career's data (as illustrated in Sec. 4.3). We identified a set of error types, and for each of them we have implemented the respective corrective actions. Recall that each inconsistency is an inadmissible transition in the automaton of Fig. 4. In order to clarify the matter, we provide some examples of corrective actions.

Er1 Two different full-time-job-start events are found with no cessation in between. The corrective action is to put a cessation event (for the first job) in between.

Er2 A part-time-job-cessation event is found, but the corresponding job-start event is missing. The corrective action is to place a corresponding job-start event somewhere before the cessation event.

Er3 A part-time-job-start ($PT3$) is found but two different part time jobs are ongoing ($PT1$ and $PT2$). Two corrections are available: either to close $PT1$ or to close $PT2$ in both cases before the start of $PT3$. These corrective actions are deeply analyzed in Fig. 5.

The sensitivity analysis is executed according to the simplified algorithm described in Algorithm 3 which is more scalable with respect to the one described in Sec. 3.1. Roughly speaking, for each career S_i the Algorithm MAIN executes twice SENSSIMPLIFIED to obtain the S_i UB and LB indicator values (upper and lower bound respectively). More precisely:

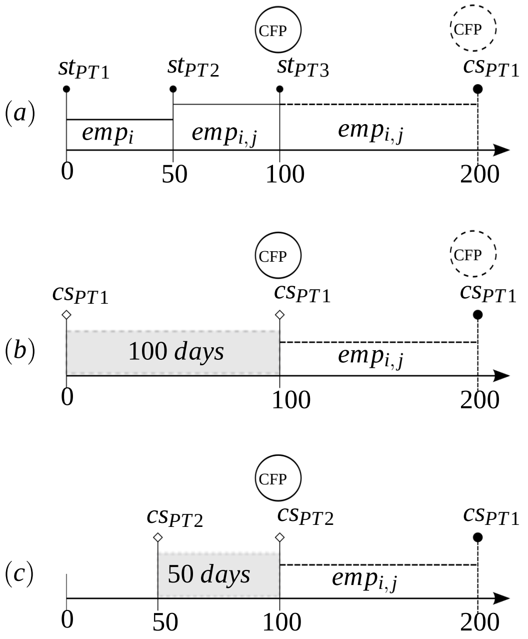
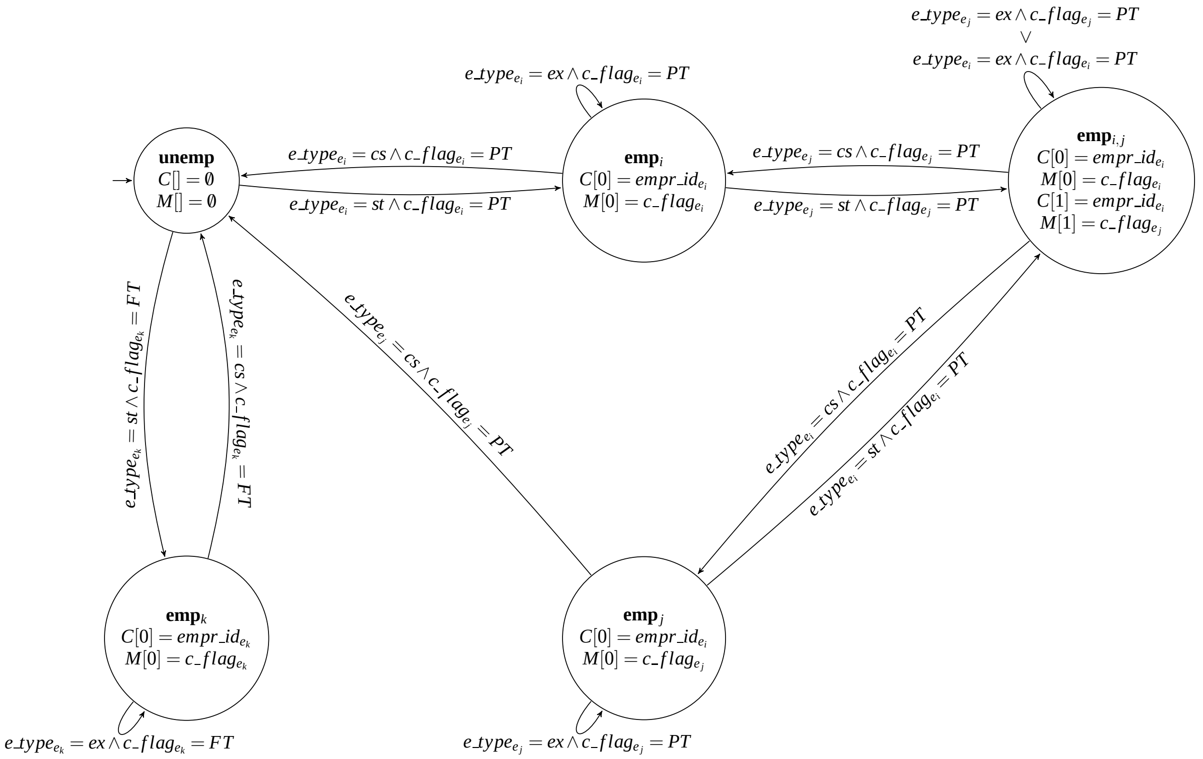


Figure 5: (a) A career affected by Er3, (b) Possible insertion place of event closing first part time (PT_1), (c) Possible insertion place of event closing second part time (PT_2). Note that closing PT_2 will create no problem in the future while closing PT_1 will originate an inconsistency when the real PT_1 cessation will be met.

1. $ccheck(S_i)$ is evaluated on the career S_i; if an inconsistency is found then the SENSSIMPLIFIED on S_i and the respective CFP_{S_i} is called (for both S_i LB and UB cases). Algorithm 1 lines 10-11.
2. Looking at SENSSIMPLIFIED, all the feasible corrective actions (suitable to fix the inconsistency) are applied, generating either a cleansed S_i' (if any) or obtaining a new CFP for S_i' (lines 3-6).
3. The corrected instance that minimises (maximises) the *working_days* indicator computed on the sub-sequence before the CFP_{S_i} is selected (lines 7-12).
4. If we are looking for the LB (UB) case of S_i then the new S_i^{LB} (S_i^{UB}) case will be expanded, i.e. *ccheck* on S_i^{LB} (S_i^{UB}) will be executed looking for further inconsistencies. This operation corresponds to expanding the leftmost (rightmost) subtree of Figure 2 (lines 15-20).

The hypothesis listed in Sec. 3.2 should be verified in order to guarantee that the simplified algorithm produces the correct result. The career working days indicator is an additive function (the demonstration is trivial and therefore is omitted).

Given an error, every corrective action tries to make the career subsequence before the CFP consistent and leading to the same state, the automaton de-



$e\_type_{e_j} = ex \wedge c\_flag_{e_j} = PT$
$\vee$
$e\_type_{e_i} = ex \wedge c\_flag_{e_i} = PT$
$e\_type_{e_i} = ex \wedge c\_flag_{e_i} = PT$
unemp
$C[] = \emptyset$
$M[] = \emptyset$
$e\_type_{e_i} = cs \wedge c\_flag_{e_i} = PT$
$e\_type_{e_i} = st \wedge c\_flag_{e_i} = PT$
emp$_i$
$C[0] = empr\_id_{e_i}$
$M[0] = c\_flag_{e_i}$
$e\_type_{e_j} = cs \wedge c\_flag_{e_j} = PT$
$e\_type_{e_j} = st \wedge c\_flag_{e_j} = PT$
emp$_{i,j}$
$C[0] = empr\_id_{e_i}$
$M[0] = c\_flag_{e_i}$
$C[1] = empr\_id_{e_i}$
$M[1] = c\_flag_{e_j}$
$e\_type_{e_k} = st \wedge c\_flag_{e_k} = FT$
$e\_type_{e_k} = cs \wedge c\_flag_{e_k} = FT$
$e\_type_{e_j} = cs \wedge c\_flag_{e_j} = PT$
$e\_type_{e_i} = cs \wedge c\_flag_{e_i} = PT$
$e\_type_{e_i} = st \wedge c\_flag_{e_i} = PT$
emp$_k$
$C[0] = empr\_id_{e_k}$
$M[0] = c\_flag_{e_k}$
emp$_j$
$C[0] = empr\_id_{e_i}$
$M[0] = c\_flag_{e_j}$
$e\_type_{e_k} = ex \wedge c\_flag_{e_k} = FT$
$e\_type_{e_j} = ex \wedge c\_flag_{e_j} = PT$

Figure 4: The FSS of a valid worker's career where $st = start$, $cs = cessation$, $cn = conversion$ and $ex = extension$.

scribing the sequence (Fig. 4) is memoryless (if we don't consider the state variables $c[]$ and $m[]$). Therefore all the corrections will have the same future, excluding one correction type that will be deeply described later (it is the "Er3" and in this case the variables $c[]$ and $m[]$ are the responsible of violating the property 2). Sequences affected by the latter corrections are marked as *unsafe*. The "not altering the past" property violation has been detected as follow. Every time a correction has to be made to previous sequences containing (fixed) CFPs, the subsequences affected are marked as *unsafe*.

Subsequences following *unsafe* ones are checked to verify if they have still to be considered *unsafe*. The presence of Full Time contract start or cessation events act as *point-of-reset* and they can be used to bound the extent of *unsafe* sub-sequences. The rationale is that a full time contract prevents the contextual presence of other full time or part time contracts, therefore 1) it ends the uncertainty caused by previous errors and 2) forces all possible prior corrections to share the same future. After all career subsequences have been corrected and marked either as *safe* or as *unsafe*, the indicator Upper and Lower bounds are computed using the *computation* or the *estimation* approach respectively. Given a sub-sequence covering (for example) 100 days, the *estimation* approach will estimate the *working_days* as follows: 0 is assumed as lower bound and 100 is assumed as upper bound. The approach is very coarse grained but it is also very fast. Hopefully the unsafe subsequences are few with respect to the safe ones. The indicator value for safe subsequences is computed by counting the actual worked days. The UB and LB values are generated in this way. The whole career upper and lower bound indicator values can be computed by summing the subsequences corresponding values; then, the average value of all the careers UB (LB) cases is computed. The more are the inconsistencies found in the database and the larger will be the gap between the UB and the LB average values. This gap can be interpreted as a measure of the uncertainty related to the database inconsistency.

Now let's consider the previously introduced Er3 error. As showed in Fig. 5, choosing to create the cessation event either of Part Time 1 (Fig. 5(b)) or of Part Time 2 (Fig. 5(c)) will create different versions of the future. Indeed related corrective actions violate the condition "preserving a common future".

We will call unsafe careers hereafter to refer to all the careers having at least one unsafe subsequence. The unsafe careers are not negligible (in our case are about 10% of the total), and then it is important to evaluate the impact they can have on the indicator.

Next section will illustrate some resuls.

4.3 Experimental Results

	Average Working Days			*Results*		
	Layer	$\overline{LB}$	$\overline{UB}$	Modified	Consistent	Num
A	(a)	1,377	1,979	YES	YES	82,989
	(b)	1,383	1,383	NO	YES	130,500
	(c)	0	4,017	NO	NO	56
	(d)	0	4,017	YES	NO	887
B	(a)	1,633	1,942	YES	YES	61,301
	(b)	1,383	1,383	NO	YES	130,500
	(c)	0	4,017	NO	NO	56
	(d)	0	4,017	YES	NO	384

Table 2: Data Quality Sensitivity Analysis Experimental Results considering (A) *all* careers i.e. using both the *computation* and the *estimation* approaches and (B) *safe only* careers using only the *computation* approach. Note that 4017 are the days between the 1^{st} January 2000 and the 31^{st} December 2010.

We tested the *Data Quality Sensitivity Analysis* on an Italian Province Administrative Database having $214,432$ careers (composed by $1,248,751$ Mandatory Notifications) observed between the 1^{st} January 2000 and the 31^{st} December 2010.

Tab. 2 summarises the sensitivity analysis statistics. The part A considers all the careers (both *safe* and *unsafe*) and the results are computed applying both the *computation* and the *estimation* approach. Part B of Tab. 2 excludes from the sensitivity analysis all the *unsafe* careers (i.e. careers having at least one *unsafe sub-sequence*).

Tables 2 report the average value of the working days variable for the best and worst cases layered by the *Consistent* and *Modified* flags. The former means that a career is consistent at the end of the correction process cause it was either initially consistent or a consistent version has been generated. The *Modified* flag states that at least one corrective action has been applied on the career. Hence, looking at Table 2 row (a) represents careers made consistent while (b) are careers already consistent. Row (c) represents careers for which no corrective action has been found whilst the corrective actions were ineffective for careers in row (d). It is worth to note that (b) refers to consistent careers which don't need to be fixed, consequently there is no source of uncertainty and for this reason Upper and Lower bound are equals. We computed a sensitivity index *SI* of the working days variable as $SI = \frac{\sum((\overline{UB}-\overline{LB})\cdot Num)}{\sum(\overline{LB}\cdot Num)}$. For the dataset which includes *unsafe careers* we obtained a $SI = 18.2\%$. Differently, the *SI* for the dataset which excludes *unsafe careers* is $SI = 7.3\%$. The *SI* value could have been narrowed by exploiting a more fine grained approach for estimating the unsafe subsequences, but we won't deeper this topic. These results show that:

- the *SI* excluding unsafe careers is less than the one which considers unsafe careers. This is an expected result since the estimation approach greatly contribute to enlarge the *working days* indicator upper and lower bound distance.
- The impact of unsafe careers on the indicator *working days* cannot be neglected. Furthermore, if we apply the *estimation approach* without considering the point-of-resets, the *SI* grows up to 26%.
- Due to the inconsistency of the careers, data indicators computed on the cleansed dataset are affected by uncertainty. Nevertheless the sensitivity analysis can be helpful to bound this uncertainty. In our case, regardless the value of the *SI*, which could be acceptable or not (according to the domain requirements), the SI represents a useful information for labour market analysts about the reliability of the *working days* indicator.

We implemented the Algorithms 1,2 and 3 in C++, using the BOOST library to interact with the DBMS. All the experiments were performed on a 32 bits 2.2Ghz CPU in about 1 hour using 200 MB of RAM.

5 CONCLUSION AND FUTURE WORK

We have described how to perform the "Data Quality Sensitivity Analysis" to assess how uncertainty (due to not deterministic corrections of inconsistent data) can affect indicators computed on cleansed data. This methodology has been implemented to quantitatively estimate how the inconsistencies present in a real database can affect an aggregate indicator (i.e. working days on a observed time slot) that is relevant for statistics on the labour market place, since several indicators are based on it. Although in our work we have focused only on a specific indicator, our approach can be easily extended to several ones (compliant with the requirements showed in the paper). The possibility to estimate how data cleansing can affect indicators used for decision making is extremely valuable for decision makers. Our methodology exploits the Finite State Systems and they proved to be useful for modelling the domain rules and to automatically check data consistency. Furthermore the approach presented in this paper can avoid the computational issues affecting works dealing with indicators computed upon inconsistent data (at the price of more coarse grained evaluation). Currently the ongoing research goes into the direction of exploiting model checking to automatically identify possible corrective actions starting from a Finite State System that models a domain. As a future work we are going to investigate the issues arising when the methodology proposed in this paper have to deal with indicators and scenarios requiring an estimation of several indicators to address inconsistencies (including possible complexity issues). We are also considering how to exploit statistical information about the consistent part of a dataset to narrow the bounds of the estimated indicator. The dataset that has been described in this paper cannot be shared due to privacy related issues. We are working on building a fictitious dataset that can be used as a testbed for comparing formal methods with other approaches (e.g. learning based methods) in the context of inconsistency detection and resolution.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

Arasu, A. and Kaushik, R. (2009). A grammar-based entity representation framework for data cleaning. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 233–244.

Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V., and Spinrad, J. (2003). Scalar aggregation in inconsistent databases. *Theoretical Computer Science*, 296(3):405–434.

Arenas, M., Bertossi, L. E., and Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *ACM Symp. on Principles of Database Systems*, pages 68–79. ACM Press.

Batini, C., Cappiello, C., Francalanci, C., and Maurino, A. (2009). Methodologies for Data Quality Assessment and Improvement. *ACM Comput. Surv.*, 41:16:1–16:52.

Batini, C. and Scannapieco, M. (2006). *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer.

Chomicki, J. and Marcinkowski, J. (2005). Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121.

Csiszar, I. and Körner, J. (1981). *Information theory: coding theorems for discrete memoryless systems*, volume 244. Academic press.

Embury, S., Brandt, S., Robinson, J., Sutherland, I., Bisby, F., Gray, W., Jones, A., and White, R. (2001). Adapting integrity enforcement techniques for data reconciliation. *Information Systems*, 26(8):657–689.

Fan, W., Geerts, F., and Jia, X. (2008). A Revival of Integrity Constraints for Data Cleaning. *Proc. VLDB Endow.*, 1:1522–1523.

Fellegi, I. and Holt, D. (1976). A systematic approach to automatic edit and inputation. *Journal of the American Statistical association*, 71(353):17–35.

Galhardas, H., Florescuand, D., Simon, E., and Shasha, D. (2000). An extensible framework for data cleaning. In *Proceedings of ICDE '00*, pages 312–. IEEE Computer Society.

Iosifescu, M. (1980). *Finite Markov processes and their applications*. Wiley.

Maletic, J. and Marcus, A. (2000). Data cleansing: beyond Integrity Analysis. In *Proceedings of the Conference on Information Quality*, pages 200–209.

Martini, M. and Mezzanzanica, M. (2009). The Federal Observatory of the Labour Market in Lombardy: Models and Methods for the Costruction of a Statistical Information System for Data Analysis. In Larsen, C., Mevius, M., Kipper, J., and Schmid, A., editors, *Information Systems for Regional Labour Market Monitoring - State of the Art and Prospectives*. Rainer Hampp Verlag.

Mayfield, C., Neville, J., and Prabhakar, S. (2009). A Statistical Method for Integrated Data Cleaning and Imputation. Technical Report CSD TR-09-008, Purdue University.

Mezzanzanica, M., Boselli, R., Cesarini, M., and Mercorio, F. (2011). Data quality through model checking techniques. In Gama, J., Bradley, E., and Hollmén, J., editors, *IDA*, volume 7014 of *Lecture Notes in Computer Science*, pages 270–281. Springer.

Müller, H. and Freytag, J.-C. (2003). Problems, Methods and Challenges in Comprehensive Data Cleansing. Technical Report HUB-IB-164, Humboldt-Universität zu Berlin, Institut für Informatik.

Rahm, E. and Do, H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13.

Redman, T. C. (1998). The impact of poor data quality on the typical enterprise. *Commun. ACM*, 41:79–82.

Sang Hyun, P., Wesley, W., et al. (2001). Discovering and matching elastic rules from sequence databases. *Fundamenta Informaticae*, 47(1-2):75–90.

Strong, D. M., Lee, Y. W., and Wang, R. Y. (1997). Data quality in context. *Commun. ACM*, 40(5):103–110.

Wang, R., Kon, H., and Madnick, S. (1993). Data quality requirements analysis and modeling. In *Data Engineering, 1993. Proceedings. Ninth International Conference on*, pages 670–677.

Weidema, B. P. and Wesns, M. S. (1996). Data quality management for life cycle inventoriesan example of using data quality indicators. *Journal of Cleaner Production*, 4(34):167 – 174.