Support for User Involvement in Data Cleaning

Helena Galhardas[1], Antónia Lopes[2], and Emanuel Santos[1]

[1]INESC-ID and Technical University of Lisbon,
`hig@inesc-id.pt, esantos@ist.utl.pt`
[2]Faculty of Sciences, University of Lisbon
`mal@di.fc.ul.pt`

Abstract. Data cleaning and ETL processes are usually modeled as graphs of data transformations. The involvement of the users responsible for executing these graphs over real data is important to tune data transformations and to manually correct data items that cannot be treated automatically. In this paper, in order to better support the user involvement in data cleaning processes, we equip a data cleaning graph with *data quality constraints* to help users identifying the points of the graph and the records that need their attention and *manual data repairs* for representing the way users can provide the feedback required to manually clean some data items. We provide preliminary experimental results that show the significant gains obtained with the use of data cleaning graphs.

1 Introduction

Data cleaning and ETL processes are commonly modeled as workflows or graphs of data transformations. The logic underlying real-world data cleaning processes is usually quite complex. These processes often involve tens of data transformations that are implemented, for instance, by pre-defined operators of the chosen ETL tool, SQL scripts, or procedural code. Moreover, these processes have to deal with large amounts of input data. Therefore, as pointed out in [14], in general it is not easy to devise a graph of data transformations able to always produce accurate data. This happens for two main reasons. First, individual data transformations that consider all possible data quality problems are difficult to write. Consequently, the underlying logic needs to undergo several revisions, in particular when the cleaning process is executed over a new batch of data. Hence, it is important that users responsible for executing the data cleaning processes have adequate support for tuning data transformations. Second, a fully automated solution that meets the quality requirements is not always attainable. In general, a portion of the cleaning work has to be done manually and, hence, it is important to also support the user involvement in this activity.

When using ETL and data cleaning tools, intermediate results obtained after individual data transformations are typically not available for inspection or eventual manual correction — the output of a data transformation is directly pipelined into the input of the transformation that follows in the graph. The solution we envisage for this problem is to support the specification of the points in the graph of data transformations where intermediate results must be available, together with the *quality constraints* that this data should meet, if the upward data transformations correctly transform all the data records as expected. Because assignment of blame is crucial for identifying where the problem is, the records responsible for the violation of quality constraints are highlighted. This information is useful both for tuning data transformations that do not handle the data as expected and for performing the manual cleaning of records not handled automatically by data transformations.

While the tuning of data transformations requires some knowledge about the logic of the cleaning process, it is useful that manual data repairing actions can also be performed in a black-box manner, namely by the application end-users. As already advocated in the context of Information Extraction [4], in many situations, data consumers have knowledge about how to correctly handle the rejected records and, hence, can provide critical feedback into the data cleaning program. Our proposal is that the developer of the cleaning process has the ability to specify, in the points of the graph of data transformations where intermediate results are available, the way users can provide the feedback required to manually clean certain data items. This may serve two different purposes: for guiding the effort of the user that is executing the cleaning process (even if he/she has some knowledge about the underlying logic) and for supporting the feedback of users that are just data consumers.

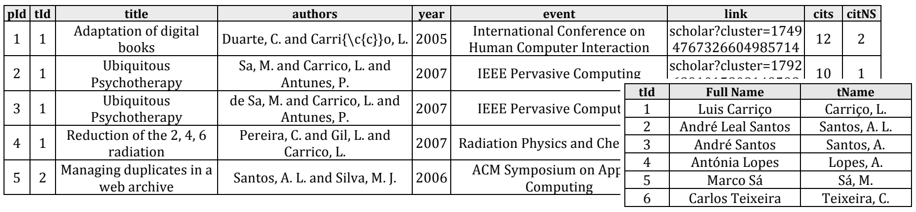
In this paper, we put forward a notion of *data cleaning graph* (DCG, for short) that supports the modeling of data cleaning processes that explicitly define where and how user feedback is expected as well as which data should be inspected by the user. The operational semantics of DCGs formally defines the execution of a data cleaning process over source data and past instances of manual data repairs. With this semantics it is possible to interleave the tuning of data transformations with the manual data correction without requiring that the user repeats his feedback actions. We present experimental results that show, for a real-world data cleaning application modeled as a DCG, the gain in terms of the accuracy of the data produced, and the amount of user work involved.

The paper is organized as follows. Section 2 presents the motivation and an overview of the proposed approach. In Section 3, the elements of the approach are presented in detail. In Section 4, we present a case study of a data cleaning process and, in Section 5, we report on the experimental results obtained that show the usefulness of our approach. In Section 6, we discuss the related work and in Section 7 we summarize the conclusions and future work.

2 Motivation

Let us consider that the information required for computing the research performance metrics for a given team is collected into a database with tables Team and Pub as illustrated in Fig.1 (a simplification of the real database used in the CIDS system [6]). The Team table is manually filled with accurate information about the team members. The Pub table stores the information about the citations of team members obtained through queries posed to Google Scholar.

The relationship that exists between the two tables, through the foreign key tId, associates all the publications to a team member. However, this association may be incorrect, namely due to the existence of homonyms. In our example, the first member in Team refers to a colleague of us and the Pub record with pid 4 is not authored by him, but by a homonym. Another problem that affects these tables is the multitude of variants that author names admit. For instance, the records of Pub shown in Fig.1 contain two synonyms of "Carriço, L.".



pId	tId	title	authors	year	event	link	cits	citNS
1	1	Adaptation of digital books	Duarte, C. and Carri{\c{c}}o, L.	2005	International Conference on Human Computer Interaction	scholar?cluster=17494767326604985714	12	2
2	1	Ubiquitous Psychotherapy	Sa, M. and Carrico, L. and Antunes, P.	2007	IEEE Pervasive Computing	scholar?cluster=1792	10	1
3	1	Ubiquitous Psychotherapy	de Sa, M. and Carrico, L. and Antunes, P.	2007	IEEE Pervasive Comput			
4	1	Reduction of the 2, 4, 6 radiation	Pereira, C. and Gil, L. and Carrico, L.	2007	Radiation Physics and Che			
5	2	Managing duplicates in a web archive	Santos, A. L. and Silva, M. J.	2006	ACM Symposium on App Computing			

tId	Full Name	tName
1	Luis Carriço	Carriço, L.
2	André Leal Santos	Santos, A. L.
3	André Santos	Santos, A.
4	Antónia Lopes	Lopes, A.
5	Marco Sá	Sá, M.
6	Carlos Teixeira	Teixeira, C.

Fig. 1. Pub and Team tables.

The computation of reliable research performance indicators for a team requires a data cleaning process that, among other things, deals with the problems of synonyms and homonyms pointed before. The Team table can be used as reference to identify and correct these problems. State-of-art procedures to solve synonyms are based on the use of approximate string matching [12]. Names as "Carriço, L." in tuple 1 of Team table and "Carrico, L." in tuple 2 of Pub table can easily be found as matches. However, it may also happen that these procedures find several possible correct names for the same author name. For example, "Santos, A." and "Santos, A. L." are the names of two team members and both match the author name "Santos, A. L." encountered in tuple 5 of the Pub table. That is to say, both names in ("Santos, A.","Santos, A. L.") and ("Santos, A. L.", "Santos, A. L.") are similar enough so that both entries of the Team table are considered as potential candidates of team member names for "Santos, A. L.". The problem that remains to be solved is which of the two to choose, or to decide if none of them does in fact correspond to the individual "Santos, A. L.". We believe that this kind of domain knowledge can only be brought by a user that is aware of the team members and their research work. The syntactic similarity value that exists between the two pairs is not enough for automatically taking this decision.

The detection of homonyms in the context of names has been object of active research. For instance, [13] has shown that the detection of homonyms among author names can benefit from the use of knowledge about co-authorship. If this kind of information is available, then a clustering algorithm can be applied with the purpose of putting into the same cluster those author names that share a certain amount of co-authors. In principle, the author names that belong to the same cluster most probably correspond to the same real entity. The problem that remains is how to obtain accurate co-authorship information. Clearly, automatic methods for calculating this information from publications are also subject to the problem of homonyms and, hence, the produced information in general is not accurate. In this case, we believe that the problem of circularity can only be broken by involving the user in the cleaning of the co-authorship information that was automatically obtained.

The example just presented shows the importance of being able to automatically clean data while efficiently employing user's efforts to overcome the problems that were not possible to handle automatically. In this paper, we propose a way of incorporating the user involvement in these processes and present a modeling primitive — the *data cleaning graph*, that supports the description of data cleaning processes that are conceived having user involvement in mind. A DCG encloses a graph of data transformations as used, for instance, in [17, 9]. The output of each transformation is explicitly expressed and associated with a *quality constraint.* This constraint expresses the criteria that data produced by the transformation should obey to and its purpose is to call the user attention for quality problems in the data produced by the transformation. Additionally, the DCG encloses the specification of the points where the manual data repairing actions may take place. The aim of this facility is to guide the intervention of the user (end-users included) and, hence, it is important to define which data records can be subject to manual modifications and how. We have only considered actions that can be applied to individual data records for repairing data. Three types of actions were found useful: remove a tuple, insert a tuple, and modify the values of certain attribute of a tuple.

3 Data cleaning graphs

In this section we present the concept of *data cleaning graph* — the modeling primitive we propose for describing data cleaning processes. We provide its operational semantics through an algorithm that manipulates sets of tuples.

Terminology. We consider a set $\mathcal{R}$ of relations names and, for every $R{\in}\mathcal{R}$, a schema $sch(R)$ constituted by an ordered set of attribute names. An *instance* of a relation R is a finite set of $sch(R)$-tuples. We consider a set $\mathcal{T}$ of *data transformations*. Each $T{\in}\mathcal{T}$ consists of an ordered set $\mathbb{I}_T$ of input relation schemas, an output relation schema $\mathbb{O}_T$ and a total function that maps a sequence of $\mathbb{I}_T$-tuples to $\mathbb{O}_T$-tuples. We use $\mathbb{I}^i_T$ to denote the i-ary element of $\mathbb{I}_T$. If G is a *direct acyclic graph* (DAG), we use ${}^\bullet n$ and $n^\bullet$ to denote, respectively, $\{m :(m,n){\in}edges(G)\}$ and $\{m :(n,m){\in}edges(G)\}$ and $\leq_G$ to denote the partial order on the nodes of G, i.e., $n \leq_G m$ iff there exists a directed path from n to m in G.

3.1 The notion of data cleaning graph

The notion of DCG builds on the notion of data transformation graph introduced in [9]. These graphs are tailored to relational data and include data transformations that can range from relational operators and extensions (like the mapper operator formalized in [3]) to procedural code. The partial order $\leq_G$ on $nodes(G)$ partially dictates the order of execution of the data transformations in the process (transformations not comparable can be executed in any order).

A data cleaning graph is a DAG, where nodes correspond to data transformations or relations, and edges connect (input and output) relations to data transformations. In order to support the user involvement in the process of data cleaning, each relation R in a cleaning graph has associated a constraint expressing a *data quality criteria*. If the constraint is violated, it means that there is a set of tuples in the current instance of R that needs to be inspected by the user. Quality constraints can include the traditional constraints developed for schema design, such as functional dependencies and inclusion dependencies, as well as constraints specifically developed for data cleaning, such as conditional functional dependencies [7]. Each relation R in a cleaning graph has also associated a set of *manual data repairs*. These represent the actions that can be performed by the user over the instances of that relation in order to repair some quality problems, typically made apparent by one or more quality constraint labelling that relation or a relation "ahead" of R in the graph. For the convenience of the user, it might be helpful to filter the information available in R and, thus, we have considered that data repair actions are defined over updatable views of R[1]. They can range from SQL expressions to relational lenses [2]. The examples of manual data repairs provided in this paper consider an updatable view defined as an SQL expression.

Definition 1. *A* Manual Data Repair *m over a relation $R(A_1, ..., A_n)$ consists of a pair $\langle view(m), action(m)\rangle$, where $view(m)$ is an updatable view over R and $action(m)$ is one of the actions that can be performed over $view(m)$:*

$action ::= \textbf{delete} \mid \textbf{insert} \mid \textbf{update}\ A_i$

In the case where the action is **update** *A_i, we use $attribute(m)$ to refer to A_i.*

Definition 2. *A* Data Cleaning Graph *$\mathcal{G}$ for a set of input relations R_I and a set of output relations R_O is a labelled directed acyclic graph $\langle G, \langle \mathcal{Q}, \mathcal{M}\rangle\rangle$ s.t.:*

- *$nodes(G) \subseteq \mathcal{R} \cup \mathcal{T}$. We denote by $rels(G)$ and $trans(G)$ the set of nodes of G that are, respectively, relations and data transformations.*
- *$R_I \cup R_O \subseteq rels(G)$.*
- *$n \in R_I$ if and only if ${}^\bullet n = \emptyset$, and $n \in R_O$ if and only if $n^\bullet = \emptyset$ and ${}^\bullet n \neq \emptyset$.*
- *if $(n, m) \in edges(G)$, then either ($n \in \mathcal{R}$ and $m \in \mathcal{T}$) or ($n \in \mathcal{T}$ and $m \in \mathcal{R}$).*
- *if $T \in trans(G)$ then $\mathbb{I}_T = \{sch(R) : R \in {}^\bullet T\}$ and $\mathbb{O}_T = \{sch(R) : R \in T^\bullet\}$.*
- *if $R \in rels(G)$ then ${}^\bullet R$ has at most one element.*
- *$\mathcal{Q}$ is a function that assigns to every $R \in rels(G)$, a quality constraint over the set of relations behind R in G or in R_I, i.e., $\mathcal{Q}(R) \in \mathcal{L}(R_I \cup \{R' \in rels(G) : R' \leq_G R\})$ such that $\mathcal{Q}(R)$ is monotonic w.r.t. R, i.e., given a set of relation instances that satisfies $\mathcal{Q}(R)$, the removal of an arbitrary number of tuples form the instance of R does not affect the satisfaction of $\mathcal{Q}(R)$.*
- *$\mathcal{M}$ is a function that assigns to every $R \in rels(G)$, a set of manual data repairs over R.*

The conditions imposed on DCGs ensure that R_I and R_O are input and output relations of the graph; relations are always connected through data transformations; the input and output schemas of a data transformation are those

[1] For a definition of an updatable view, see [11], for instance.

determined by their immediate predecessors and successors nodes in the graph; the instances of a relation in the graph result, at most, from one data transformation; the quality constraints over a relation in the graph can only refer to relations that are either in R_I or behind that node in the graph and must be monotonic w.r.t. to the relation of the node. This last condition is necessary to ensure that quality constraints can be evaluated immediately after the data of the relation is produced, i.e., does not depend on data that will be produced later, by transformations ahead in the graph.

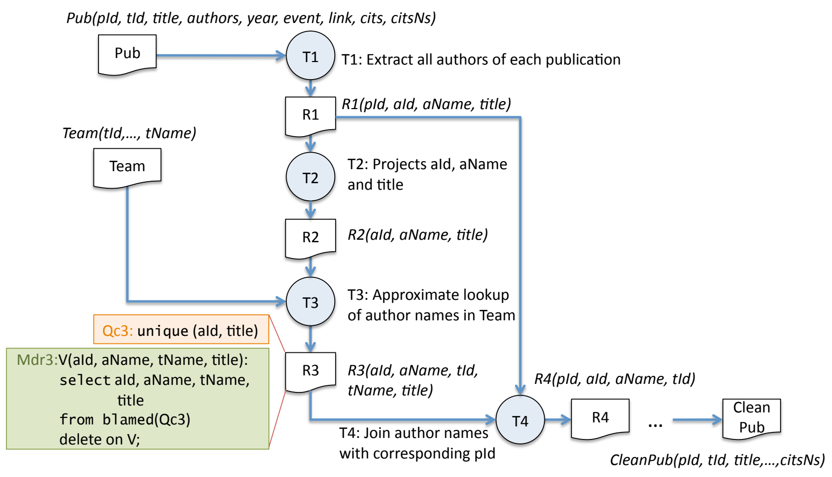

Pub(pId, tId, title, authors, year, event, link, cits, citsNs)
Pub
T1
T1: Extract all authors of each publication
R1
R1(pId, aId, aName, title)
Team(tId,…, tName)
Team
T2
T2: Projects aId, aName and title
R2
R2(aId, aName, title)
T3
T3: Approximate lookup of author names in Team
Qc3: unique (aId, title)
Mdr3:V(aId, aName, tName, title): select aId, aName, tName, title from blamed(Qc3) delete on V;
R3
R3(aId, aName, tId, tName, title)
R4(pId, aId, aName, tId)
T4
R4
…
Clean Pub
T4: Join author names with corresponding pId
CleanPub(pId, tId, title,…,citsNs)

Fig. 2. Excerpt of a data cleaning graph for cleaning Pub table

The example sketched in Fig.2 illustrates an excerpt of the DCG required for cleaning the Pub table introduced in Section 2. It mainly makes use of SQL for expressing constraints and updatable views. The input relations of this DCG are Team and Pub and there is a single output relation, CleanPub that contains only publications authored by a member of Team. In the part of the graph that is shown, we can see that the node R3 is labelled with the quality constraint unique(aId, title). It is not difficult to conclude this is indeed a monotonic constraint over relations $\leq_G R$. The reason for imposing this quality constraint, at this point, is because we want to have at most one matching team member, for each author of a publication in Pub. Since transformation T3 applies a string similarity function to decide if two names (one from Pub and the other from Team) are the same, it might happen that some data produced by T3 violates this constraint. For instance, both Team members "Santos, A." and "Santos, A. L." are found similar to Pub author "Santos, A. L.". The quality constraint will call the attention of the user to the tuples blamed for the violation.

Moreover, the function $\mathcal{M}$ of this DCG assigns to the node R3 a single manual data repair, Mdr3, that consists in the view V defined over R3 that returns only the tuples blamed for the violation of Qc3 (this is formally defined in the next section) and the action **delete**. The view V projects almost all the attributes of the relation but we could use the view to exclude non relevant information and, in this way, limit the amount of information the user has to process in order to decide which are the appropriate manual data repairs to apply.

3.2 Operational Semantics

DCGs specify the quality criteria that the instances of each relation should meet. The records responsible for the violation are identified through the notion of blame assignment for quality constraints.

Definition 3. *Let ϕ be a quality constraint over a set of relations $R_1, ..., R_n$ that is assigned to relation R. Let r and $r_1, ..., r_n$ be instances of these relations s.t. $r, r_1, ..., r_n \nvDash \phi$. The blame of the violation is assigned to the set $blamed(\phi)$, which is defined as the union of all subsets rp of r that satisfy: (1) $r \backslash rp, r_1, ..., r_n \vDash \phi$; (2) rp does not have a proper subset o s.t. $r \backslash o, r_1, ..., r_n \vDash \phi$.*

Each subset rp of r that satisfies the two conditions above represents a way of "repairing" r through the removal of a set of tuples that, all together, cause the violation of ϕ (a particular case of data repairs as introduced in [1]). Hence, all tuples in r that have this type of "incompatibility" share the blame for the violation of ϕ. For instance, suppose that R3 in Fig.2 has the tuples (1,"Santos, A. L.","Santos, A. L.", 2,"Managing...") and (1,"Santos, A. L.", "Santos, A.", 3,"Managing..."). These tuples are blamed for the violation of the quality constraint Qc3. Notice that this form of blame assignment is only appropriate if constraints are monotonic in R and this is why we limit constraints to be of this type.

Data cleaning of a source of data tends to be the result of numerous iterations, some involving the tuning of data transformations and others involving manual data repairs. Even if the DCG developed for the problem was subject to a strict validation and verification process, it is normal that when it is executed over the real data, small changes in the DCG, confined to specific data transformations, are needed. Because we do not want to force the user to repeat the data repairs previously done that, in principle, are still valid, we define that the execution of a DCG takes as input not only the data that needs to be cleaned but also collections of instances of manual data repairs (mdr, for short). These represent mdr actions enacted at some point in the past. For convenience, we consider that instances of mdrs keep track of their type.

Definition 4. *Let m be a manual data repair. If $action(m)$ is* **delete** *or* **insert**, *an $m-$instance ι is a pair $\langle m, tuple(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple. If $action(m)$ is* **update** *A, an $m-$*instance *ι is a triple $\langle m, tuple(\iota), value(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple, $value(\iota)$ is a value in $Dom(A)$.*

For instance, still referring to Fig.2, after analyzing the violation of the quality constraint Qc3 and taking the title into account, the user could conclude that the author "Santos, A. L." does not correspond to the team author "Santos, A." and decide to delete the corresponding tuple from 0R3. This would generate the Mdr3-instance ⟨mdr3, (1, "Santos, A. L.", "Santos, A.", "Managing...")⟩.

The execution of a DCG is defined over a source of data (instances of the graph input relations) and what we call a *manual data repair state M* — a state capturing the instances of mdrs that have to be taken into account in the cleaning process. Because the order of actions in this context is obviously relevant, this state registers the order by which the instances of mdrs associated to each relation should be executed (what comes in first is handled first).

The execution of a DCG consists in the sequential execution of each data transformation in accordance with the partial order defined by the graph: if $T <_{\mathcal{G}} T'$, then T' is executed after T. The execution of a data transformation T produces an instance of the relation R in $T^\bullet$. This relation is then subject to the mdr instances in $M(R)$. Then, the set of tuples in the resulting relation instance that are blamed for the violation of the quality constraint associated to R, $\mathcal{Q}(R)$ is calculated. Formally, the execution of a DCG can be defined as follows.

Definition 5. *Let $\mathcal{G} = \langle G, \langle \mathcal{Q}, \mathcal{M} \rangle \rangle$ be a data cleaning graph for a set $R_1, ..., R_n$ of input relations. Let $r_1, ..., r_n$ be instances of these relations and M be a* manual data repair state *for $\mathcal{G}$, i.e., a function that assigns to every relation $R \in rels(G)$, a list of instances of manual data repairs over R. The result of executing $\mathcal{G}$ over $r_1, ..., r_n$ and M is $\{\langle tuples(R), tuples^{bl}(R) \rangle : R \in rels(G)\}$ calculated as follows:*

```
for $i = 1$ to $n$ do
    for each** $\iota \in M(R_i)$ do
        $vr \leftarrow compute\_view(view(\iota), tuples(R_i))$
        $apply\_mdr(\iota, vr)$
        $tuples(R_i) \leftarrow propagate(vr)$
    end for
end for
for $i = 1$ to $n$ do
    $tuples^{bl}(R_i) \leftarrow blamed(tuples(r_i))$
end for
for each* $T \in trans(G)$ do
    let $\{R'_1, ..., R'_k\} = {}^\bullet T$
    $tuples(T^\bullet) \leftarrow T(tuples(R'_1), ..., tuples(R'_k))$
    for each** $\iota \in M(T^\bullet)$ do
        $vr \leftarrow compute\_view(view(\iota), tuples(T^\bullet))$
        $apply\_mdr(\iota, vr)$
        $tuples(T^\bullet) \leftarrow propagate(vr)$
    end for
    $tuples^{bl}(T^\bullet) \leftarrow blamed(tuples(T^\bullet))$
end for

$apply\_mdr(mdrInstances, vr)$
for each** $\iota \in mdrInstances$ do
    if $action(mdr(\iota)) =$ delete then
        $vr \leftarrow vr \setminus \{tuple(\iota)\}$
    else if $action(mdr(\iota)) =$ insert then
        $vr \leftarrow r \cup \{tuple(\iota)\}$
    else if $action(mdr(\iota)) =$ update then
        $newt \leftarrow tuple(\iota)$
        $newt[attribute(action(mdr(\iota)))] \leftarrow value(\iota)$
        $vr \leftarrow (vr \setminus \{tuple(\iota)\}) \cup \{newt\}$
    end if
end for
```

** Assuming that the underlying iteration will traverse the set in ascending element order. ** Assuming that the underlying iteration will traverse the list in proper sequence.*

The procedure *compute_view(view,setOfTuples)* encodes the application of the *view* to the base table constituted by the *setOfTuples* whereas *propagate(view)* encodes the propagation of the updates applied to the tuples returned by *view* to the base table. Although this algorithm defines an operational semantics for DCGs, it must not be regarded as a proposal for the implementation of an engine that supports the execution of DCGs. The sole purpose of this algorithm is to formally define what is the result of executing a DCG over a source of data and a manual data repair state.

4 Case Study

We have developed and implemented in full depth the process to clean publication citation data retrieved from the web, introduced in Section 2. The goal of this process is to clean the Pub table and produce a table containing only the publications authored by at least one team member, with duplicate entries for the same real world publication organized in clusters. The process: (i) extracts the author names independently of the publication they are associated to; (ii) matches each of these author names against the names stored in the Team table, and tries to find synonyms (i.e., approximate similar names); (iii) builds the list of co-authors for each author; (iv) removes those publications that are not authored by any team member; and (v) detects and clusters approximate duplicate publication records.

The DCG that models this process is presented in Fig. 3 and in the two tables presented in Fig. 4. It presents slight differences with respect to the excerpt presented in Fig. 2, because therein we made some simplifications (more details can be found in [10]). The condition that an author of each publication can only match one team member is now checked through the quality constraint Qc6 that is imposed after the user gives feedback about the co-authorship tuples (through Mdr5). The data transformation T5 was introduced for gathering the co-authorship information about each author. The co-authorship information, after being validated by the user, can provide additional knowledge that is helpful for automatically deciding whether an author name in a publication refers to a team member.

Based on the matching name pairs produced by T3 and T4, and on the co-authorship tuples produced by T5, the transformation T6 is able to distinguish, among the set of authors for each publication, those who belong to the team from those who do not. The user feedback provided through Mdr6 confirms whether the information automatically produced is true. Finally, T7 discards the publication records whose list of authors does not contain a team member. Besides producing Pub records that concern only team members, the goal of the graph is also to put together Pub records that concern the same real world publication. To this end, the publication records must be compared in order to identify entries that constitute approximate duplicates. For this purpose, transformations T9 and T10 match pairs of publications, and cluster the matched publications.

Other quality constraints were introduced in the graph to call the user's attention for anticipated data problems. Qc0 and Qc8 call the user attention for analyzing and correcting tuples that have the word "others" in its authors attribute value, and tuples that correspond to single-author publications (i.e., by checking if the author attribute value does not contain the conjunction "and", which connects two or more authors names), respectively. Quality constraints Qc3 and Qc9 are imposed on the result of the matching operations encoded in T3 and T9, respectively, that consider the existence of two threshold values. Pairs of records whose computed similarity is below the inferior threshold are considered as non-matches and discarded by the transformations. Pairs of records whose similarity is above the inferior threshold are considered as candidate matches and returned as a result of the data transformations. Those resulting records whose similarity value (stored in the sim attribute) is inferior to the superior threshold violate the corresponding quality constraints (Qc3 and Qc9). These

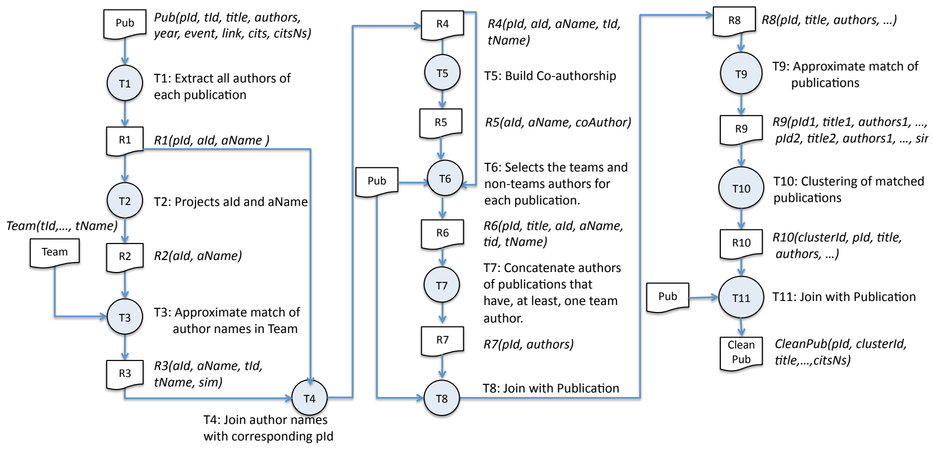
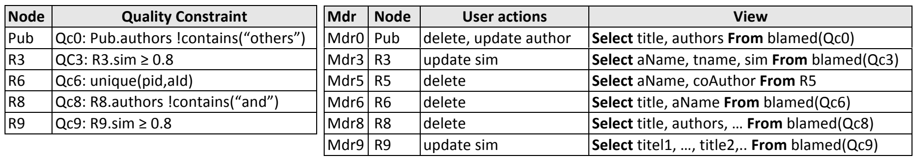

Pub
Pub(pId, tId, title, authors, year, event, link, cits, citsNs)
T1
T1: Extract all authors of each publication
R1
R1(pId, aId, aName)
T2
T2: Projects aId and aName
Team(tId,..., tName)
Team
R2
R2(aId, aName)
T3
T3: Approximate match of author names in Team
R3
R3(aId, aName, tId, tName, sim)
T4
T4: Join author names with corresponding pId
R4
R4(pId, aId, aName, tId, tName)
T5
T5: Build Co-authorship
R5
R5(aId, aName, coAuthor)
Pub
T6
T6: Selects the teams and non-teams authors for each publication.
R6
R6(pId, title, aId, aName, tId, tName)
T7
T7: Concatenate authors of publications that have, at least, one team author.
R7
R7(pId, authors)
T8
T8: Join with Publication
R8
R8(pId, title, authors, ...)
T9
T9: Approximate match of publications
R9
R9(pId1, title1, authors1, ..., pId2, title2, authors1, ..., sir
T10
T10: Clustering of matched publications
R10
R10(clusterId, pId, title, authors, ...)
Pub
T11
T11: Join with Publication
Clean Pub
CleanPub(pId, clusterId, title,...,citsNs)

Fig. 3. Data cleaning graph for the case study.

records do not have a sufficiently high value nor a sufficiently low value of the sim attribute, so the user must analyze them. Then, through Mdr3 and Mdr9, the user may decide whether the corresponding pairs of author names or publications are considered as matches, by modifying the sim value accordingly (1 for matches, and 0 for no matches).



Node	Quality Constraint
Pub	Qc0: Pub.authors !contains("others")
R3	QC3: R3.sim ≥ 0.8
R6	Qc6: unique(pid,aId)
R8	Qc8: R8.authors !contains("and")
R9	Qc9: R9.sim ≥ 0.8

Mdr	Node	User actions	View
Mdr0	Pub	delete, update author	**Select** title, authors **From** blamed(Qc0)
Mdr3	R3	update sim	**Select** aName, tname, sim **From** blamed(Qc3)
Mdr5	R5	delete	**Select** aName, coAuthor **From** R5
Mdr6	R6	delete	**Select** title, aName **From** blamed(Qc6)
Mdr8	R8	delete	**Select** title, authors, … **From** blamed(Qc8)
Mdr9	R9	update sim	**Select** titel1, …, title2,.. **From** blamed(Qc9)

Fig. 4. Quality constraints and manual data repairs of the DCG.

5 Experiments

We performed a set of experiments to evaluate the benefits of involving the user in the data cleaning process described in Section 4. We focused on two different aspects: the data quality obtained at the end of the data cleaning process and the cost of the manual activities that have to be performed by the user.

The experiments were performed with the AJAX data cleaning prototype[8], over a subset of the database of the CIDS[6]. These experiments required to implement two data cleaning programs: P_1 complying with the data transformation graph presented in Fig.3 and P_2 complying with the data transformation graph presented in Fig.3 and capturing, as closest as possible with the means available, the quality constraints presented in the first table presented in Fig.4. Quality constraints in P_2 were encoded inside transformations, making use of exceptions as supported by AJAX. As a result, the tuples available for user inspection are not those blamed for the violation but those that originate a blamed tuple. Moreover, the tuples that raise exceptions are not available as input for the transformations ahead in the graph. However, for the evaluation purpose at hand, these differences were considered to be neglectable.

We performed the following cleaning tasks. $Task_1$: the manual cleaning of the Pub table. $Task_2$: the execution of P_1 and the manual intervention of the user over the produced data in the output CleanPub table so that it contains all publications that are authored by at least one team member with duplicates organized in clusters. $Task_3$: the execution of the P_2 and the manual intervention of the user over the produced data in the CleanPub table guided by the rejected tuples in the different points of the program. $Task_4$: the execution of the data cleaning program and, after receiving user feedback, the re-execution of parts of it — with the user involvement guided by the rejected tuples and the mdrs presented in the second table presented in Fig.4.

The metrics used to evaluate the quality of the CleanPub records produced are recall and precision. *TD Recall* (TD R) is given by the number of CleanPub tuples that are authored by the team (i.e., authored by at least one team member) divided by the number of CleanPub tuples authored by the team that should have been produced. *TD Precision* (TD P) is given by the number of CleanPub tuples that are authored by the team divided by the number of CleanPub tuples that were produced. *DD Recall* (DD R) is given by the number of pairs of CleanPub tuples that were correctly identified as duplicates (i.e., the ones with the same value of the clusterId attribute and that correspond to the same real publication) divided by the total number of pairs of CleanPub tuples that should have been identified as duplicates. *DD Precision* (DD P) is given by the number of pairs of CleanPub tuples that were correctly identified as duplicates divided by the number of pairs of CleanPub tuples that were identified as duplicates.

To evaluate the cost associated to the user feedback, we consider the following metrics that we believe can capture the most relevant aspects of user interaction: the number of characters the user needs to visualize in order to decide which data corrections need to be undertaken; the maximum number of characters that may need to be updated, when attribute values are modified; the maximum number of characters that may need to be deleted or inserted, when tuples are deleted or inserted; and the number of tuples that need to be updated, deleted or inserted. The number of characters is given by the multiplication of the number of tuples by the sum of the sizes of each attribute.

We used an instance of the CIDS database, that contains 509 and 24 tuples in the tables Pub and Team, respectively. It includes all the publication records returned by Google Scholar for five members of the team, chosen beforehand. First, we performed $Task_1$ and obtained the cleaned version of this instance by manually cleaning it. This process was performed by retrieving information from the member's home pages and DBLP. Then, the cleaned Pub table obtained was checked and eventually corrected by each team member. The manually cleaned publication table, named $\mathsf{CleanPub}_1$, was used as a reference for computing the quality of the data cleaned automatically and the impact of user feedback.

Data accuracy. To compute the gain of data quality obtained when incorporating the user feedback, we performed $Task_2$, $Task_3$ and $Task_4$. The resulting publication records obtained in each of these cases were stored in tables named, $\mathsf{CleanPub}_2$, $\mathsf{CleanPub}_3$, and $\mathsf{CleanPub}_4$, respectively. The recall and precision (both TD and DD) of the $\mathsf{CleanPub}_3$ and $\mathsf{CleanPub}_4$ tables were 100%. We recall that, in both cases, the manual corrections applied by the user are guided by rejected tuples. In the case of $\mathsf{CleanPub}_2$, 70% of TD R, 78% of DD R and 100% of precision were obtained. In fact, in $Task_2$, the user only has access to the data produced at the end of the data cleaning process and so there is no way of recovering the data tuples that were not properly handled by some data transformations. Overall, these data accuracy values can be considered as good, but there is a trade-off between data accuracy and the cost of user feedback required.

In the case of $Task_4$, to analyze the effect of the different mdrs in the final result, we measured the values of precision and recall after applying each mdr. We considered that after the mdr instances were applied, the remaining of the DCG was re-executed and the precision and recall of $\mathsf{CleanPub}_4$ data was re-computed. The results obtained are summarized in Table 1. We notice that the precision and recall values greatly improved with the user's feedback via mdrs. The non-increasing values of DD P when Mdr8 is applied are justified by the existence of pairs of tuples that correspond to the same single-author publication but whose similarity is inferior to 0.8. These pairs of tuples violated Qc8 and, because we use AJAX exception mechanism for "simulating" quality constraint violation, they were not delivered to transformation T9.

mdr	**TD P**	**TD R**	**DD P**	**DD R**
none	0.83	0.70	0.98	0.76
Mdr0	0.83	0.70	0.98	0.76
Mdr3	0.85	0.80	0.98	0.91
Mdr5	1	0.92	0.98	0.91
Mdr6	1	0.92	0.98	0.91
Mdr8	1	1	0.93	0.93
Mdr9	1	1	1	1

Table 1. Precision and Recall for $\mathsf{CleanPub}_4$ table.

Cost/Task	$Task_1$	$Task_2$	$Task_3$	$Task_4$
Visualization	200,000	137,000	115,000	32,000
# deleted tuples	164	56	56	134
Deletion	33,500	11,500	11,500	7,500
# updated tuples	121	2	32	21
Updating	2,600	40	800	150
# inserted tuples	0	0	68	0
Insertion	0	0	14,000	0

Table 2. Cost of user feedback.

Cost of user feedback. We also wanted to find out whether the approach of incorporating the user feedback into the DCG (embodied by $Task_4$) facilitates the work of the user when compared to other approaches. For this purpose, we measured the cost associated to the user actions performed in the four tasks referred above. The results obtained are presented in Table 2. The cost of data visualization, updating, deletion and insertion are approximate values.

In Table 2, we observe that the use of quality constraints and mdrs in $Task_4$ greatly decreases the cost of data visualization with respect to the other tasks. Notice that this result is even true when comparing the cost of data visualization incurred in $Task_2$, which only considers the data produced at the end of the data cleaning process. This result can be explained by the existence of quality constraints that were specified in such a way that only the set of tuples blamed by constraint violations are shown to the user. In other cases, the mdrs define judiciously the data the user needs to analyze in order to decide which action must be applied.

In what concerns the cost of the user feedback incurred in each task, we also observe that the use of mdrs also decreases substantially the number and cost of user actions that must be applied to manually correct data. In comparison to $Task_1$ and $Task_3$, the results obtained by $Task_4$ are significantly improved. Although in $Task_4$ the user deletes a higher number of tuples than in $Task_3$, the cost of delete in $Task_4$ is lower than the corresponding cost in $Task_3$ because the user has to analyse a smaller amount of data in order to apply each delete action. With respect to $Task_2$, the obtained results are slightly better than $Task_4$ because in $Task_2$ the user actions are only applied over data produced at the end of the data cleaning process and, therefore, the rejected tuples are not analyzed, resulting in significantly worst recall values (70% of TD R and a 78% of DD R). Overall, the results show that the use of the new primitives addressing the user feedback ($Task_4$) may significantly improve a data cleaning process.

6 Related work

Error handling in ETL and data cleaning tools. In current commercial ETL and data cleaning tools, the developer can specify that input records not handled by some pre-defined operators are written into a *log file* whose contents can be later analyzed by the user. However, no user feedback provided on the data stored in these files can be re-integrated in the flow of data transformations. In some tools (e.g. SQL Server Integration Services), it is possible to partially overcome this limitation, by explicitly specifying an error output flow for some data operators that can be later analyzed by the user or considered as input of further data operators.

Support for error handling in the context of data cleaning was investigated in the context of prototypes AJAX [9] and ARKTOS [18] through the notion of, respectively, *exception* and *rejection*. Both notions correspond to input tuples that are not properly handled by a given data transformation. Rejected tuples and exceptions are stored in a specific table whose schema is the same as the input schema of the transformation (in ARKTOS) or contains the key of the input tuples (in AJAX). The purpose of this information is to call the user's attention for data items not correctly handled in specific points of the graph of data transformations. However, these solutions do not provide the support we believe should be available at the modelling level of data cleaning processes. For instance, AJAX exceptions rely on relational technology to detect the occurrence of integrity constraint violations. As a result, in many situations it is not possible to predict which are the tuples that will be identified as exceptions because it will depend on the order in which tuples of the input tables are processed (typically not under the control of the developer). Other initiatives to encode data quality rules and store the records that violate them have taken place (e.g., [16]).

User feedback. The incorporation of user feedback has shown to be useful in several automatic tasks. For example, Chai et al [4] propose a solution to incorporate the end-user feedback into Information Extraction programs. An

Information Extraction program is composed by a set of declarative rules. The developer writes some of these rules with the purpose of specifying the items of data the users can edit and the user interfaces that can be used. Analogously, we are proposing a way of specifying the exact points in the graph of data transformations where the user can provide feedback to improve the quality of the produced data. Moreover, we are limiting the amount of information the user can visualize and provide some guidance for the manual modification of data. In the context of data cleaning, Potter's Wheel [15] offers a graphical interface through which the developer can specify and quickly debug data cleaning rules that are applied to samples of data.

Data repairs. In [5], Cong and colleagues propose a framework for data cleaning that supports algorithms for finding repairs for a database and a statistical method to guarantee the accuracy of the repairs found. As noted in Section 3, the notion of blamed tuples introduced in this paper is based on the concept of database repair (considering that repair operations are limited to deletion of tuples). We consider as blamed for the violation of a data quality constraint associated to a relation of a database, those tuples in the relation instance that belong to some repair of the database.

Recently, [19] puts forward a system for guiding data repairing that explicitly involves the user in the process of checking the data repairs automatically produced by the algorithms introduced in [5]. In particular, the authors focused on ranking the repairs in such a way that the user effort spent in analyzing useless information is minimized. In this paper, we aim at reaching the same goal: to minimize the user effort when providing feedback in a data cleaning process. However, in the current version of our research, we do not provide any method for clustering or ranking the tuples that violate constraints. For the moment, we claim that by disclosing a limited set of records to the user, we are able to reduce the amount of data that he/she needs to analyze and eventually modify.

7 Conclusions

In this paper, we address the problem of integrating the user feedback in an automatic data cleaning process. We propose the notion of *data quality constraint* that may be associated to any of the intermediate relations produced by data transformations in a DCG. We also propose that a DCG specifies *manual data repairs*, that to some extend can be regarded as a kind of wizard-based form that limits the amount of data that can be visualized and modified. We have performed preliminary experiments with a real-world data set that show the gain of data quality achieved when the user feedback is incorporated and that the overhead incurred by the user, when providing feedback guided by quality constraints and mdrs, is significantly inferior to the effort involved in cleaning rejected records in an ad-hoc manner.

As future work, we plan to modify the definition of updatable view that is used in the definition of mdrs so that the join of base relations is possible. Special care must be taken so that the view remains updatable in the sense that the updates can always be propagated to the base relations. In addition, the concept of DCG and corresponding operational semantics must be adequately supported by a software platform that should efficiently compute the set of blamed tuples for a given quality constraint violation, enable the automatic re-application of past user actions, and support the incremental execution of data transformations.

References

1. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.
2. A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational lenses: a language for updatable views. In *PODS*, pages 338–347, New York, NY, USA, 2006. ACM.
3. P. Carreira, H. Galhardas, A. Lopes, and J. Pereira. One-to-many data transformations through data mappers. *Data Knowl. Eng.*, 62(3):483–503, 2007.
4. X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, pages 87–100, 2009.
5. G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.
6. F. M. Couto, C. Pesquita, T. Grego, and P. Verissimo. Handling self-citations using google scholar. *Cybermetrics*, 13(1), 2009.
7. W. Fan, F. Geerts, and X. Jia. Conditional dependencies: A principled approach to improving data quality. In *BNCOD*, pages 8–20, 2009.
8. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In *SIGMOD*, page 590, 2000.
9. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.
10. H. Galhardas, A. Lopes, and E. Santos. Support for user involvement in data cleaning applications. DI/FCUL TR 2010-03, Faculty of Sciences, University of Lisbon, 2010. URL `http://hdl.handle.net/10455/6674`.
11. H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2008.
12. P. A. V. Hall and G. R. Dowling. Approximate string matching. *ACM Comput. Surv.*, 12(4):381–402, 1980.
13. I.-S. Kang, S.-H. Na, S. Lee, H. Jung, P. Kim, W.-K. Sung, and J.-H. Lee. On co-authorship for author disambiguation. *Inf. Process. Manage.*, 45(1):84–97, 2009.
14. E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
15. V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.
16. J. Rodic and M. Baranovic. Generating data quality rules and integration into etl process. In *DOLAP*, pages 65–72, 2009.
17. A. Simitsis, P. Vassiliadis, M. Terrovitis, and S. Skiadopoulos. Graph-based modeling of etl activities with multi-level transformations and updates. In *DaWaK*, pages 43–52, 2005.
18. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Inf. Syst.*, 30(7):492–525, 2005.
19. M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.