Improving Data Quality by Source Analysis

HEIKO MÜLLER, CSIRO
JOHANN-CHRISTOPH FREYTAG, Humboldt-Universität zu Berlin
ULF LESER, Humboldt-Universität zu Berlin

15

In many domains, data cleaning is hampered by our limited ability to specify a comprehensive set of integrity constraints to assist in identification of erroneous data. An alternative approach to improve data quality is to exploit different data sources that contain information about the same set of objects. Such overlapping sources highlight hot-spots of poor data quality through conflicting data values and immediately provide alternative values for conflict resolution. In order to derive a dataset of high quality, we can merge the overlapping sources based on a quality assessment of the conflicting values. The quality of the resulting dataset, however, is highly dependent on our ability to asses the quality of conflicting values effectively.

The main objective of this article is to introduce methods that aid the developer of an integrated system over overlapping, but contradicting sources in the task of improving the quality of data. Value conflicts between contradicting sources are often systematic, caused by some characteristic of the different sources. Our goal is to identify such systematic differences and outline data patterns that occur in conjunction with them. Evaluated by an expert user, the regularities discovered provide insights into possible conflict reasons and help to assess the quality of inconsistent values. The contributions of this article are two concepts of systematic conflicts: contradiction patterns and minimal update sequences. Contradiction patterns resemble a special form of association rules that summarize characteristic data properties for conflict occurrence. We adapt existing association rule mining algorithms for mining contradiction patterns. Contradiction patterns, however, view each class of conflicts in isolation, sometimes leading to largely overlapping patterns. Sequences of set-oriented update operations that transform one data source into the other are compact descriptions for all regular differences among the sources. We consider minimal update sequences as the most likely explanation for observed differences between overlapping data sources. Furthermore, the order of operations within the sequences point out potential dependencies between systematic differences. Finding minimal update sequences, however, is beyond reach in practice. We show that the problem already is NP-complete for a restricted set of operations. In the light of this intractability result, we present heuristics that lead to convincing results for all examples we considered.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous—*Data cleaning*; H.2.8 [**Database Management**]: Database applications—*Data mining; scientific databases*

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Conflict resolution, data cleaning, quality assessment, semantic distance measure

ACM Reference Format:
Müller, H., Freytag, J.-C., and Leser, U. 2012. Improving data quality by source analysis. ACM J. Data Inf. Qual. 2, 4, Article 15 (February 2012), 38 pages.
DOI = 10.1145/2107536.2107538 http://doi.acm.org/10.1145/2107536.2107538

Authors' addresses: H. Müller (corresponding author), Tasmanian ICT Centre, Commonwealth Scientific and Industrial Research Organisation (CSIRO), GPO Box 1538, Hobart TAS 7001, Australia; email: heiko.mueller@csiro.au; J.-C. Freytag and U. Leser, Institut für Informatik, Humbolit-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany.

DOI 10.1145/2107536.2107538 http://doi.acm.org/10.1145/2107536.2107538

1. INTRODUCTION

Decreasing cost of generating and maintaining scientific data has led to an enormous increase in the number of scientific data sources on the Web. For example, the number of entries in the NAR Molecular Biology Database Collection has increased from 202 entries in 1999 [Burks 1999] to 1,107 entries in 2009 [Galperin and Cochrane 2009]. Within this multitude of sources, there are various examples of overlaps between them for the following reasons.

—*Data Replication*. Genome data is frequently copied, and modified by different research projects that publish their results as new data sources. This is especially true for curated databases, where data is constantly copied, altered, extended, and corrected between hundreds of databases [Buneman et al. 2008].
—*Different Groups Analyzing, or Observing, the Same Set of Real-World Objects*. A common practice in scientific research is to distribute the same set of samples, such as clones, proteins, or patient's blood, to different laboratories for analysis to enhance reliability of the final results [Zehetner and Lehrach 1994]. A typical example of the same set of clones being sequenced multiple times is the Human Genome Project [International Human Genome Sequencing Consortium 2004; Venter et al. 2001]. In order to achieve the targeted exactness of 99.99% each base is sequenced six times on average [Dennis and Gallagher 2002].

Whenever data on a given set of objects is distributed or generated without a controlling scheme enforcing consistency, there is a high probability that the actual values will differ, either already from the start due to different measurements, or after changes made independently in the different sources. Other reasons can be errors in the replication mechanism, different levels of actuality of the data, or usage of different vocabularies to describe the same concepts. We refer to these differences as *conflicts* or *contradictions*. Note that conflicts need not be rooted in actual errors, but may as well represent different points-of-view on the same fact [Bleiholder and Naumann 2008].

Example 1.1. Throughout the article, we use an example from structural biology to showcase our contributions. The OpenMMS project [Bhat et al. 2001] and the Macromolecular Structure Relational Database (E-MSD) [Boutselakis et al. 2003] are two data cleaning projects that focus on different quality aspects of the Protein Structure Database (PDB) [Berman et al. 2000]. Both work independently from each other on a copy of the original PDB. When comparing the OpenMMS project data with E-MSD at a given point in time, we found hundreds of thousands of contradicting values between them. These contradictions not only affected their respective focus of cleansing activities, but also such fundamental metadata as the date-of-creation of a PDB entry. Analyzing these contradictions using the algorithms detailed later, we found a regularity in this error and were able to track down its root to a parser error in the OpenMMS project.

Example 1.2. An example of conflicts between independently generated data was published by Steven E. Brenner [Brenner 1999]. He compared functional annotations for the *Mycoplasma genitalium* genome generated independently by three different groups to estimate the accuracy of automatic functional annotation. He found that of the 468 genes of this bacterium only 340 were annotated by two or more groups, and that for about 8% of these genes the functional descriptions of at least two of the groups were completely incompatible.

These examples not only demonstrate the existence of conflicts between overlapping data sources, but also show the importance of spotting them for guiding a systematic data cleaning process. Cleaning scientific data, in general, is hampered by our incomplete or fuzzy knowledge of domain regularities, which limits our ability to specify a

comprehensive set of integrity constraints to identify erroneous data. Integrating data sources with overlapping scope can help to highlight hot-spots of poor data quality as these often appear as clusters of contradicting values [Müller et al. 2003]. Conflicts, once detected, may be resolved by domain experts familiar with domain constraints, regularities, and possible pitfalls in the data generation process; however, these experts are often overwhelmed by the sheer number of conflicts [Baumgartner et al. 2007]. On the other hand, conflicts between overlapping data sources often are systematic, that is, they are produced not at random, but due to some characteristics of the different systems (such as the parser error in the example given before). A frequent reason for conflicts is the usage of different vocabularies or measurement units. For instance, one data source might describe the function of a gene as "*metalloenzyme inhibitor activity*" (term A), while another source uses the term "*metalloenzyme inhibition*" (term B). Assuming identical methods to resolve gene function, we can expect to always find term A in the first source when there appears term B in the second; assuming different methods, we can expect to frequently find term A in the first source when there appears term B in the second. In this article, we develop methods for finding such commonalities in conflicts when faced with thousands of individual contradictions. We contribute by developing and investigating two concepts of systematic conflicts. Both concepts rely on identifying meaningful patterns that occur in conjunction with conflicts between contradicting data sources.

—*Contradiction patterns.* Contradiction patterns summarize data properties that are characteristic for the occurrence of conflicts between overlapping sources for a single attribute. These patterns help in providing answers to questions like "Which are the conflict-causing attributes, values, or value pairs?" or "What kind of dependencies exists between the occurrences of contradictions and certain (conflicting) data values in different attributes?". To this end, we introduce contradiction patterns, which are a special kind of association rule [Agrawal and Srikant 1994].

—*Minimal update sequences*. Contradiction patterns essentially view each conflict in isolation, which often leads to largely redundant and overlapping patterns. We therefore introduce a second way to understand systematic conflicts using a process-oriented model by studying sequences of update operations. Assuming that the divergent sources have a common root (like in Example 1.1), we develop algorithms that find the minimal number of set-oriented update operations that can explain the differences. These sequences are a compact representation for all conflicts between a pair of data sources; furthermore, the order of operations within the sequences point out potential dependencies between systematic differences.

Evaluated by an expert user, contradiction patterns and minimal update sequences act as descriptive information providing insights towards: (i) possible conflict reasons, and (ii) the quality of individual values. Within this space, our main contributions are as follows.

—We describe a model of systematic conflicts based on contradiction patterns. We present an adaptation of existing association rule mining algorithms to allow for efficient contradiction pattern mining over large datasets. Similar to the measures of support and confidence for association rules, we define interestingness measures for contradiction patterns that allow us to restrict the set of patterns returned by our algorithm to those most helpful for the expert user.

—We introduce minimal update sequences as an alternative model for systematic conflicts. Minimal update sequences fully take into account dependencies between conflicts, but are more complex to compute than contradiction patterns.

—We prove that, even for a restricted class of update operations, the problem of finding minimal update sequences is NP-complete.

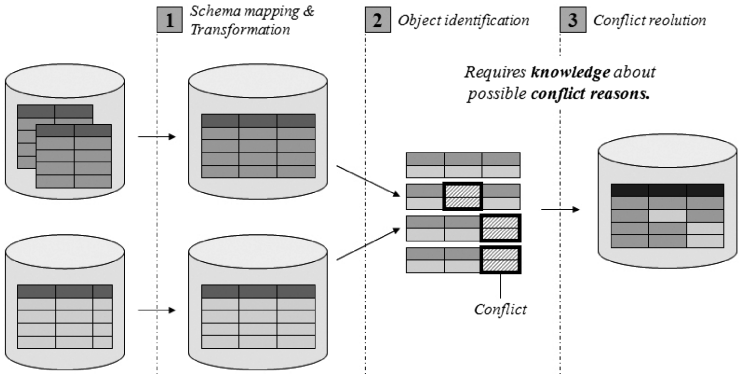


Fig. 1. The three-step data integration process.

—We present different algorithms that allow computation of minimal update sequences. However, in the light of our intractability result, we also present scalable heuristics to compute short (yet not necessary minimal) update sequences between pairs of large data sources.
—We provide experimental evaluations of our models and algorithms using real-world and synthetic data.

The remainder of this article is structured as follows: Section 2 defines the necessary concepts and notation. Section 3 defines contradiction patterns while Section 4 develops algorithms for mining them. Section 5 defines update distance between databases. Section 6 presents algorithms to determine minimal update sequences. Section 7 defines problem variations and establishes complexity bounds for one of these variations. Heuristic algorithms are presented in Section 8. We review related work in Section 9. Section 10 concludes the article and discusses future work.

2. PRELIMINARIES

The methods and algorithms presented in this article assume a three-step data integration process depicted in Figure 1 [Bleiholder and Naumann 2008] (see Stein [2003], Hernandez and Kambhampati [2004], and Louie et al. [2007] for surveys on integrating biological databases). Given a pair of databases, the *schema mapping and transformation* step transforms each database to match a unified global schema (see Abiteboul et al. [1999], Rahm and Bernstein [2001], Lenzerini [2002], and Fagin et al. [2005]). The first step results in a pair of databases structured under the same schema. The *object identification* step assigns a global unique object identifier to each entry in the resulting databases (see Elmagarmid et al. [2007] and Winkler [1999] for surveys). Object identifiers are used to identify matching entries between different databases. The *conflict resolution* step resolves value conflicts between matching tuples to derive an integrated database [Naumann and Häussler 2002; Bleiholder and Naumann 2008]. This work contributes to the conflict resolution step. Information about systematic conflicts is derived after object identification. The results may then be used to guide conflict resolution, or to improve the first two steps of the integration process. We now define the necessary concepts and notations used throughout the article (see Table I for an overview).

2.1. Contradicting Databases

Databases. Within this article, we solely consider relational databases containing a single relation following schema $R(A_1, \ldots, A_n)$. Let $dom(A)$ denote the domain of

Table I. Summary of Notations Used Throughout This Article

	Contradicting Databases
r	Database instance.
$t\{j\}$	Tuple with primary key value j.
$v(r_1, r_2)$	View of matching pairs between r_1, r_2.
$u(r_1, r_2)$	Unmatched tuples from r_1.
C_A	Conflict indicator for attribute A.
$v_C(r_1, r_2, A)$	Set of matching pairs with conflict in A.
$v_N(r_1, r_2, A)$	Set of matching pairs without conflict in A.
$conflicts(r_1, r_2)$	Number of conflicts between r_1 and r_2.
	Data Patterns
τ	Term.
ρ	Pattern.
$sup(\tau, r)$	Support of τ in r.
$sup(\rho, r)$	Support of ρ in r.
$P_C(r)$	Set of closed patterns for r.
	Database Transformer
ψ	Update operation.
ψ_{INS}	Insert operation.
ψ_{DEL}	Delete operation.
ψ_{MOD}	Modification operation.
Ψ	Update sequence.
$\Delta_U(r_1, r_2)$	Update distance between r_1, r_2.

attributes $A \in R$ and $r \subseteq dom(A_1) \times \cdots \times dom(A_n)$ denote database instances. For tuples $t \in r$ and attributes $A \in R$, let $t[A]$ denote the value of tuple t for attribute A. We assume the existence of a primary key constraint for schema R. Without loss of generality, we assume A_1 to be the primary key attribute. We use ID as a synonym for A_1. We refer to the tuple with primary key value j by $t\{j\}$. Note that the equal database schemas result from the mapping and transformation step, and that primary keys are assigned within the object identification step. The primary key constraint further ensures that the individual databases are free of duplicates.

View of Matching Pairs. Given a pair of databases r_1, r_2. A pair of tuples $t_1 \in r_1, t_2 \in r_2$ is called a *matching pair*, if t_1, t_2 possess identical primary key values, that is, $t_1[ID] = t_2[ID]$. A pair of databases is called *overlapping*, if there exists at least one matching pair between them. Conflicts between overlapping databases occur within matching pairs that disagree on certain attribute values, that is, whenever $t_1[ID] = t_2[ID] \wedge t_1[A] \neq t_2[A]$ for $A \in R$. A pair of databases is called *contradicting*, if there exists at least one conflict between them. For mining contradiction patterns, we define a relational representation for the set of matching pairs and conflicts between them. For a pair of databases r_1, r_2, the *view of matching pairs*, denoted by $v(r_1, r_2)$, is a relational instance over schema $V(ID, A_{2_1}, A_{2_2}, C_{A_2}, \ldots, A_{n_1}, A_{n_2}, C_{A_n})$. Attributes A_{i_1} and A_{i_2} are renamed versions of attribute $A_i \in R, 2 \leq i \leq n$, that represent the different values for A_i in matching pairs. Attribute C_{A_i} is called the *conflict indicator* for attribute A_i, and $dom(C_{A_i}) = \{true, false\}$.

The view of matching pairs is the result of joining r_1, r_2 on the ID attribute and adding a conflict indicator for attributes $A_2, \ldots, A_n$. We use the following SQL query to generate the view of matching pairs.

```
SELECT ID, r1.A2 AS A2_1, r2.A2 AS A2_2,
 CASE r1.A2 WHEN r2.A2 THEN false ELSE true AS C_A2,
 ...,
 r1.An AS An_1, r2.An AS An_2,
 CASE r1.An WHEN r2.An THEN false ELSE true AS C_An
FROM r1, r2 WHERE r1.ID = r2.ID
```

Databases:

r_1

ID	Tissue	Location	Cofactor	Disease
1	Lung	Peroxisome	Fe^+	Lup. Nep.
2	Liver	Peroxisome	FAD	P–NALD
3	Brain	Peroxisome	Zn^+	RCDP2
4	Kidney	Peroxisome	FAD	RCDP3
5	Kidney	Membrane	Fe^+	VDDR–1
6	Heart	Membrane	Zn^+	HSN1
7	Kidney	Cytoplasm	Mg^+	VDDR–1
8	Heart	Cytoplasm	FAD	HSN1

r_2

ID	Tissue	Location	Cofactor	Disease
1	Lung	Peroxisome	Iron	Lup. Nep.
2	Liver	Peroxisome	Zn^+	VDDR–1
3	Brain	Peroxisome	Zn^+	VDDR–1
4	Kidney	Peroxisome	Zn^+	VDDR–1
5	Kidney	Membrane	Iron	VDDR–1
6	Heart	Membrane	Zn^+	VDDR–1
7	Kidney	Cytoplasm	Mg^+	VDDR–1
8	Heart	Cytoplasm	FAD	HSN1

$v(r_1, r_2)$

ID	Tis_1	Tis_2	C_{Tis}	Loc_1	Loc_2	C_{Loc}	Cof_1	Cof_2	C_{Cof}	Dis_1	Dis_2	$\mathsf{C}_{\mathsf{Dis}}$
1	Lung	Lung	false	Peroxisome	Peroxisome	false	Fe^+	Iron	true	Lup. Nep.	Lup. Nep.	false
2	Liver	Liver	false	Peroxisome	Peroxisome	false	FAD	Zn^+	true	P–NALD	VDDR–1	true
3	Brain	Brain	false	Peroxisome	Peroxisome	false	Zn^+	Zn^+	false	RCDP2	VDDR–1	true
4	Kidney	Kidney	false	Peroxisome	Peroxisome	false	FAD	Zn^+	true	RCDP3	VDDR–1	true
5	Kidney	Kidney	false	Membrane	Membrane	false	Fe^+	Iron	true	VDDR–1	VDDR–1	false
6	Heart	Heart	false	Membrane	Membrane	false	Zn^+	Zn^+	false	HSN1	VDDR–1	true
7	Kidney	Kidney	false	Cytoplasm	Cytoplasm	false	Mg^+	Mg^+	false	VDDR–1	VDDR–1	false
8	Heart	Heart	false	Cytoplasm	Cytoplasm	false	FAD	FAD	false	HSN1	HSN1	false

Patterns:

$\rho_1 = \{(\mathsf{Location}, \texttt{'Peroxisome'}), (\mathsf{Cofactor}, \texttt{'Zn}^+\texttt{'})\}$
$\rho_2 = \{(\mathsf{Location}, \texttt{'Peroxisome'}), (\mathsf{Cofactor}, \texttt{'Zn}^+\texttt{'}), (\mathsf{Disease}, \texttt{'VDDR-1'})\}$

Contradiction pattern for Disease:

$\rho_{\mathsf{Disease1}} = \{(\mathsf{C}_{\mathsf{Tis}}, \texttt{false}), (\mathsf{C}_{\mathsf{Loc}}, \texttt{false}), (\mathsf{Cof}_2, \texttt{'Zn}^+\texttt{'}), (\mathsf{Dis}_2, \texttt{'VDDR-1'})\}$
$\rho_{\mathsf{Disease2}} = \{(\mathsf{Loc}_1, \texttt{'Peroxisome'}), (\mathsf{Loc}_2, \texttt{'Peroxisome'})\}$

Update sequence:

(1) `UPDATE` r_1 `SET Cofactor = 'Iron' WHERE Cofactor = 'Fe`$^+$`'`
(2) `UPDATE` r_1 `SET Cofactor = 'Zn`$^+$`' WHERE Location = 'Peroxisome' AND Cofactor = 'FAD'`
(3) `UPDATE` r_1 `SET Disease = 'VDDR-1' WHERE Cofactor = 'Zn`$^+$`'`

Fig. 2. A pair of contradicting databases containing information about the predicted involvement of proteins in certain diseases. Also shown is their view of matching pairs, example patterns, and contradiction patterns, and one of their minimal update sequences.

Example 2.1. Consider the pair of contradicting databases in Figure 2. Each database contains information about the same set of proteins. Individual proteins are identified by their ID. For each protein the databases list the tissue it occurs in, the subcellular location, the cofactor that activates the protein function, and the predicted disease the protein is involved in. Contradicting values are highlighted in gray. Figure 2 also shows the view of matching pairs $v(r_1, r_2)$.

For each attribute, we further define two disjoint subsets of the view of matching pairs: the set of *conflicting matching pairs*, and the set of *nonconflicting matching pairs*. Let $v_C(r_1, r_2, A) = \sigma_{C_A=true}(v(r_1, r_2))$ denote the set of matching pairs that have a conflict in A. Likewise, let $v_N(r_1, r_2, A) = v(r_1, r_2) \setminus v_C(r_1, r_2, A)$ denote the subset of matching pairs without a conflict in A. Based on these definitions, we compute the number of conflicts between the databases, denoted by $conflicts(r_1, r_2)$, as the sum of the number of conflicting matching pairs for each attribute, that is, $conflicts(r_1, r_2) = \Sigma_{A \in R \setminus \{ID\}} |v_C(r_1, r_2, A)|$.

Unmatched Tuples. There may exist tuples in r_1, r_2 without a matching partner. We call these tuples *unmatched*. Given a pair of databases r_1, r_2, the *set of unmatched tuples* from r_1, denoted by $u(r_1, r_2)$, is defined as $r_1 \setminus (r_1 \ltimes_{ID} r_2)$.

2.2. Data Patterns

Terms and Patterns. Terms and patterns are conjunctive queries that form the basis for contradiction patterns and update operations. A *term* τ over schema R is an attribute-value pair (A, x), with $A \in R$ and $x \in dom(A)$. We define $attr(\tau) = A$ and $val(\tau) = x$. A term (A, x) represents a query that returns all tuples t that satisfy the condition $t[A] = x$. We use $\tau(r)$ as a short form of $\sigma_{attr(\tau)=val(\tau)}(r)$. A *pattern* ρ over schema R is a set of terms $\{\tau_1, \ldots, \tau_k\}$ and $\rho(r)$ denotes the query $\sigma_{attr(\tau_1)=val(\tau_1)\wedge\ldots\wedge attr(\tau_k)=val(\tau_k)}(r)$. The empty pattern, denoted by $\rho_\emptyset$, is satisfied by any tuple, that is, $\rho_\emptyset(r) = r$. We further define the *support* that a term or pattern has in a database r as the fraction of tuples that satisfy them, that is, $sup(\tau, r) = \frac{|\tau(r)|}{|r|}$, and $sup(\rho, r) = \frac{|\rho(r)|}{|r|}$, where $|\cdot|$ denotes the size of a set.

Example 2.2. Pattern ρ_1 in Figure 2 is satisfied by tuple $t\{3\}$ in r_1, and by tuples $t\{2\} - t\{4\}$ in r_2. Pattern ρ_2 is only satisfied by tuples $t\{2\}$–$t\{4\}$ in r_2.

Closed Patterns. Patterns are used within this article to summarize sets of tuples, that is, the set of tuples they select. Different patterns, however, may select the same set of tuples. In order to avoid such descriptional redundancies, we define closed patterns. Our definition of closed patterns is in accordance to the definition of closed itemsets for association rule mining [Pasquier et al. 1999]. A pattern ρ with $\rho(r) \neq \emptyset$ is a *closed pattern* for r if there does not exist a pattern $\rho' \supset \rho$ with $\rho(r) = \rho'(r)$, that is, there exists no superset of ρ that selects the same set of tuples from r. Note that a pattern that is closed for database r_1 may not be closed for database r_2, that is, the property of being a closed pattern can only be evaluated for a given database. Furthermore, for each pattern ρ, with $\rho(r) \neq \emptyset$, there exists a corresponding closed pattern ρ' such that $\rho \subseteq \rho'$ and $\rho(r) = \rho'(r)$. From the definition it follows that the corresponding closed pattern is the set of terms that are common to all the tuples in $\rho(r)$. We denote the set of closed patterns that select at least one tuple from a database r by $P_C(r)$.

Example 2.3. Pattern ρ_1 is not closed for either r_1 or r_2 in Figure 2. The corresponding closed pattern in r_1 is the set of terms derivable from $t\{3\}$; the corresponding closed pattern in r_2 is ρ_2.

2.3. Database Transformer

Update Operations. Update operations are used to modify existing databases. For relational databases there are three types of basic update operations, namely insert, delete, and modify [Vossen 1991]. Update operations, denoted by ψ in the following, can be considered as functions that map databases onto each other. An insert operation creates a new tuple. A delete operation removes a set of tuples satisfying a given selection criterion. A modification operation changes the value for an attribute within a set of tuples satisfying a given selection criterion. We use patterns and terms in our definition of update operations.

—*Insert* operation $\psi_{INS} = (\tau_1, \ldots, \tau_n)$, containing exactly one term for each attribute in R. An insert operation adds a new tuple t_{new} to r, with $t_{new}[A_i] = value(\tau_i)$ for $1 \leq i \leq n$, that is, $\psi_{INS}(r) = r \cup \{t_{new}\}$. If there exists a tuple $t \in r$ with $t[ID] = t_{new}[ID]$, then $\psi_{INS}(r) = r$.
—*Delete* operation $\psi_{DEL} = \rho$ removes those tuples from r that are selected bu pattern ρ, that is, $\psi_{DEL}(r) = r \setminus \rho(r)$.
—*Modification* operation $\psi_{MOD} = (\rho, \tau)$ modifies all tuples within r that are selected by ρ. For these tuples, the value for attribute $attr(\tau)$ is set to $value(\tau)$. We exclude the primary key attribute from being modified, that is, $attr(\tau) \in R \setminus \{ID\}$.

Given a modification operation $\psi_{MOD} = (\rho, \tau)$, we refer to τ as the *modification term*, to $val(\tau)$ as the *modification value*, to $attr(\tau)$ as the *modified attribute*, and to ρ as the *modification pattern*. Note that there does not necessarily exist a reverse operation for each modification operation. For example, there is no single reverse operation for delete operations that delete more than one tuple.

Database Transformers. An ordered list of update operations $\Psi = \langle \psi_1, \ldots, \psi_k \rangle$ is called an *update sequence*. Applied on a database r_1, an update sequence generates (or derives) a database $r_2 = \Psi(r_1)$ by executing the update operations in given order on r_1, that is, $\Psi(r_1) = \psi_k(\ldots(\psi_1(r_1))\ldots)$. The databases that are generated by the update operations of an update sequence while transforming r_1 into r_2 are called *intermediate states*. Obviously, the order of operations within an update sequence is important. We call Ψ a *transformer* for databases r_1, r_2, if $\Psi(r_1) = r_2$. The number of update operations within a sequence is called its length and is denoted by $|\Psi|$.

Update Distance. An update sequence Ψ is called a *minimal transformer*, if $\Psi(r_1) = r_2$ and there does not exist another transformer Ψ' with $\Psi'(r_1) = r_2$, and $|\Psi'| < |\Psi|$. There may be several minimal transformers for a pair of databases. For a pair of databases r_1, r_2, the *update distance* $\Delta_U(r_1, r_2)$ is defined as the length of any minimal transformer for r_1 and r_2. Note that the update distance is not a metric as it is not a symmetric relation.

3. PATTERNS IN CONTRADICTORY DATA

Contradicting databases are valuable sources of information for data cleaning, provided that we are able to identify and resolve conflicts effectively. Within this section, we present a model for interesting groups of conflicts, called contradiction patterns. Contradiction patterns describe regularities in conflicts occurring together with certain attribute values. These patterns are therefore a very quick way to find quality hot-spots and help to ignore spurious problems.

Contradiction Patterns. Contradiction patterns highlight characteristic data properties that occur in conjunction with conflicts in attributes $A \in R$. Thus, a contradiction pattern is always interpreted in combination with an attribute A. We define a *contradiction pattern* for attribute A, denoted by ρ_A, as a pattern over schema V for which $\rho_A(v_C(r_1, r_2, A)) \neq \emptyset$ holds, that is, the pattern selects at least one matching pair having a conflict in A. The terms within a contradiction pattern ρ_A highlight data properties that are characteristic for the occurrence of conflicts in A. We exclude the term $(C_A, true)$ from the set of terms that are allowed within ρ_A.

Example 3.1. Tuples $t\{2\} - t\{4\}, t\{6\}$ in databases r_1, r_2 in Figure 2 form matching pairs having a conflict for attribute Disease. Contradiction pattern ρ_{Disease1} selects all of them. The pattern indicates that conflicts in Disease are caused by value `'VDDR-1'` being used by the second source to denote the disease involvement of the respective protein. Furthermore, these conflicts occur whenever the second source has identified the activating cofactor as zinc (`'Zn`$^{+}$`'`). Contradiction pattern ρ_{Disease2} indicates that conflicts in Disease primarily occur for proteins that are located within peroxisomes.

3.1. Interestingness Measures

Our definition considers any pattern that selects at least one matching pair having a conflict in A a contradiction pattern. We therefore define three measures of interest to filter out patterns that are most suitable to represent systematic conflicts. Based on these interestingness measures we then give the problem statement for contradiction pattern mining.

Conflict Relevance. The *conflict relevance* of contradiction pattern ρ_A is defined as the fraction of matching pairs having a conflict in attribute A that are selected by ρ_A, that is, the support $sup(\rho_A, v_C(r_1, r_2, A))$, denoted by $rel(\rho_A, r_1, r_2, A)$. Conflict relevance is a measure for the support (or relevance) of a pattern ρ_A in the set of matching pairs having a conflict in A. The higher the conflict relevance the more conflicts are captured by the pattern. Since contradiction patterns define groups of conflicts assumed to result from the same conflict reason, we are particularly interested in patterns having high conflict relevance.

Conflict Potential. The *conflict potential* of contradiction pattern ρ_A is defined as the ratio of matching pairs in $v_C(r_1, r_2, A)$ that are selected by ρ_A over the total number of matching pairs that ρ_A selects, that is,

$$pot(\rho_A, r_1, r_2, A) = \frac{|\rho_A(v_C(r_1, r_2, A))|}{|\rho_A(v(r_1, r_2))|}.$$

Conflict potential is a measure for the accuracy of a contradiction pattern in "predicting" a conflict in attribute A. The higher the conflict potential of a pattern ρ_A, the higher the probability for a matching pair selected by ρ_A to also possess a conflict in attribute A. Patterns having low conflict potential are therefore almost meaningless in our scenario since their occurrence is unrelated to the occurrence of conflicts.

Relevance Deviation. Conflict relevance and conflict potential are the primary parameters to restrict the set of contradiction patterns. Together, they allow the user to focus on those patterns that frequently and particularly hold in conjunction with conflicts in the attribute under consideration. For attributes with a large number of conflicts, however, the influence of the conflict potential diminishes. For example, using different vocabularies may cause a conflict in every matching pair and every pattern potentially has conflict potential 1. With the conflict relevance being the only significant interestingness measure, we only should combine terms that are highly similar regarding the sets of tuples they select, that is, whose occurrence is characteristic for a particular set of tuples.

Our definition of relevance deviation is motivated by the work on association rule mining on datasets having skewed support distribution [Xiong et al. 2003]. In traditional market basket analysis the frequency of items like milk, bread, and butter is expected to be significantly higher than the frequency of luxury goods like caviar. Therefore, it is not surprising to find milk present in transactions that contain caviar, for example. Thus, patterns involving items with substantially different support levels tend to be uninteresting for analytic purposes. Similarly, we want to be able to exclude combinations of terms with largely differing conflict relevance that do not yield any important information. In accordance with the definitions in Xiong et al. [2003], we therefore define the *relevance deviation* for a pattern ρ_A as

$$reldev(\rho_A, r_1, r_2, A) = 1 - \frac{min_{1 \le i \le |\rho_A|}\{rel(\{\tau_i\}, r_1, r_2, A)\}}{max_{1 \le j \le |\rho_A|}\{rel(\{\tau_j\}, r_1, r_2, A)\})}, \tau_i, \tau_j \in \rho_A.$$

Note that a relevance deviation threshold is especially helpful when mining contradiction patterns for attributes with high conflict frequency. For attributes with low conflict frequency, the relevance deviation allows us to mine for special patterns with highly related terms. For these attributes, however, a low relevance deviation threshold can miss relevant contradiction patterns. We therefore regard the relevance deviation as an additional tool that the user can use in combination with the essential interestingness measures, conflict potential, and conflict relevance.

Procedure CPMINE $(r_1, r_2, A, min_{rel}, min_{pot}, max_{dev})$

1. $CP := \emptyset$;
2. rxlist := FREQUNITERMPATTERNS (r_1, r_2, A, min_{rel});
3. root := *new Node*($\emptyset$, rxlist);
4. itt := *new ITT*(root);
5. CPEXTEND $(itt, root, CP, r_1, r_2, A, min_{rel}, max_{dev})$;
6. **return** REMOVEPATTERNS (CP, min_{pot});

Fig. 3. CPMINE algorithm for mining contradiction patterns for attribute *A*.

3.2. Mining Contradiction Patterns

Closed patterns avoid descriptional redundancies by summarizing sets of patterns that select the same set of tuples. For mining contradiction patterns, we therefore intend to focus on closed patterns. Restricting contradiction pattern mining to closed patterns, however, would not give the expected result. A closed pattern may combine terms having relevance deviation above a given threshold. In order to get the maximum information while avoiding redundancies as much as possible, we define contradiction patterns that are closed under a given relevance deviation threshold. Given databases r_1, r_2, contradiction pattern ρ_A, and relevance deviation threshold max_{dev}, we say that ρ_A is a *closed contradiction pattern under* max_{dev} if $rel(\rho_A, r_1, r_2, A) > 0$, $reldev(\rho_A, r_1, r_2, A) \leq max_{dev}$, and there does not exist another pattern ρ'_A such that $\rho_A \subset \rho'_A$, $\rho_A(v(r_1, r_2)) = \rho'_A(v(r_1, r_2))$, and $reldev(\rho'_A, r_1, r_2, A) \leq max_{dev}$. That is, a contradiction pattern is closed under a given threshold if we cannot add another term such that the resulting pattern still satisfies the relevance deviation threshold and selects the same set of tuples.

Problem Statement. Based on the preceding definitions, the problem of *contradiction pattern mining* is defined as follows: Given a pair of databases r_1, r_2, and attribute *A*, the goal is to find the set of contradiction patterns for *A* in $v(r_1, r_2)$ that are closed under the given relevance deviation threshold, and that have conflict potential and conflict relevance above given thresholds. Note that the case of contradiction patterns being closed patterns is a special case of our problem statement if the relevance deviation threshold is 1.

4. MINING FOR PATTERNS IN CONTRADICTORY DATA

We now present our algorithm for contradiction pattern mining and discuss the results of an experimental evaluation using a real-world dataset.

4.1. CPMine - Closed Pattern Mining

In the following, we use the term *closed contradiction pattern* to refer to contradiction patterns that are closed under a given relevance deviation threshold. Our algorithm for mining closed contradiction patterns, called CPMINE, is an adoption of the closed frequent itemset mining algorithm CHARM [Zaki and Hsiao 2002]. CPMINE (shown in Figure 3) takes two databases r_1, r_2, attribute *A*, and conflict potential, conflict relevance, and relevance deviation thresholds as parameters. The algorithm returns the set of closed contradiction patterns for attribute *A* that satisfy the given thresholds. CPMINE first determines in a single pass over the data the set of frequent terms (subroutine FREQUNITERMPATTERNS not shown). Terms are called *frequent* if they satisfy the given conflict relevance threshold in $v_C(r_1, r_2, A)$. Each distinct attribute-value pair in $v(r_1, r_2)$ (excluding terms for conflict indicator C_A) represents a valid term. Terms not having

sufficient conflict relevance are pruned. For each frequent term FREQUNITERMPATTERNS generates a pattern of length 1.

Example 4.1. When mining contradiction patterns having a conflict relevance of over 0.7 the set of frequent terms for attribute Disease for the databases of Figure 2 contains the following six elements (also shown are the overall support, conflict relevance, and conflict potential of the resulting pattern).

	Term	Overall Support	Conflict Relevance	Conflict Potential
τ_1:	(C_{Tis}, `false`)	1.00	1.00	0.50
τ_2:	(C_{Loc}, `false`)	1.00	1.00	0.50
τ_3:	(Dis_2, `'VDDR-1'`)	0.75	1.00	0.67
τ_4:	(Cof_2, `'Zn`$^+$`'`)	0.50	1.00	1.00
τ_5:	(Loc_1, `'Peroxisome'`)	0.50	0.75	0.75
τ_6:	(Loc_2, `'Peroxisome'`)	0.50	0.75	0.75

Similar to CHARM, CPMINE uses a tree structure (called IT-tree) to maintain candidate patterns. Each node of the tree is a prefix-list pair. The prefix is a candidate pattern and the list contains valid extensions of the prefix. CPMINE uses the IT-tree to enumerate closed contradiction patterns that satisfy given conflict relevance and relevance deviation thresholds. The set of frequent patterns returned by FREQUNITERMPATTERNS forms the extension list for the root of the IT-tree. The prefix pattern of the root node is empty. The main computation is performed in subroutine CPEXTEND that returns the set of candidate patterns satisfying the given conflict relevance and relevance deviation thresholds. The last step of CPMINE then removes those patterns having conflict potential below min_{pot}.

Subroutine CPEXTEND (shown in Figure 4) recursively builds the IT-tree in depth-first order. We denote the extension list of a node n by $list(n)$. For each element ρ_i in the extension list of the current node n_{curr} a child node n_{child} is added to the tree with ρ_i being the prefix of n_{child}. We then generate the extension list for n_{child}. Elements for the extension list are generated from pattern pairs (ρ_i, ρ_j), where $\rho_j \in list(n_{curr})$ and ρ_j appears later in order "$>_\rho$" than ρ_i. Different ordering criteria are possible for the elements in the extension list. We use the support of a pattern in increasing order, that is, if $sup(\rho_i, v(r_1, r_2)) < sup(\rho_j, v(r_1, r_2)) \Leftrightarrow \rho_i >_\rho \rho_j$. This definition of pattern order brings patterns with equal support together and thereby allows CPMINE to combine patterns having the same corresponding closed pattern at an earlier stage (see the following). For each pair (ρ_i, ρ_j) we first check whether ρ_i and ρ_j are compatible. Two patterns are compatible if:

(1) the resulting pattern $\rho_i \cup \rho_j$ is satisfiable, that is, there are no contradicting terms $\tau_i, \tau_j \in \rho_i \cup \rho_j$ such that $attr(\tau_i) = attr(\tau_j) \wedge value(\tau_i) \neq value(\tau_j)$,
(2) the relevance deviation of $\rho_i \cup \rho_j$ is below the given threshold.

Compatible patterns are merged. The resulting pattern $\rho_m = \rho_i \cup \rho_j$ is added to the extension list of n_{child} if ρ_m has sufficient conflict relevance. Note that this procedure ensures that elements in the extension list of a node: (a) are supersets of the node's prefix, and (b) satisfy conflict relevance and relevance deviation thresholds.

Similar to CHARM, CPMINE leverages two basic properties of patterns for efficient closed pattern enumeration. Recall from Section 2.2 that the corresponding closed pattern for a pattern ρ is the set of terms that are common to all the tuples in $\rho(r)$. Given two patterns ρ_1, ρ_2, and database r with $\rho_1(r) \neq \emptyset$ and $\rho_2(r) \neq \emptyset$. The following holds.

```
Procedure CPExtend (itt, n_curr, CP, r_1, r_2, A, min_rel, max_dev)
1.  for each ρ_i ∈ list(n_curr) do
2.      n_child := AddChild (n_curr, (ρ_i, ∅));
3.      for each ρ_j ∈ list(n_curr) with ρ_j >_ρ ρ_i do
4.          if (Compatible (ρ_i, ρ_j, max_dev))
5.              ρ_m := ρ_i ∪ ρ_j;
6.              if (rel(ρ_m, r_1, r_2, A) > min_rel)
7.                  if (ρ_i(r) = ρ_j(r)) /* Case 1 */
8.                      remove ρ_j from list(n_curr);
9.                      replace ρ_i with ρ_m in itt;
10.                 else if (ρ_i(r) ⊂ ρ_j(r)) /* Case 2 */
11.                     replace ρ_i with ρ_m in itt;
12.                 else if (ρ_i(r) ⊃ ρ_j(r)) /* Case 3 */
13.                     remove ρ_j from list(n_curr);
14.                     add ρ_m to list(n_child);
15.                 else /* Case 4 */
16.                     add ρ_m to list(n_child);
17.     if (list(n_child) ≠ ∅)
18.         CPExtend (itt, n_child, CP, r_1, r_2, A, min_rel, max_dev);
19.     RemoveChild (n_curr, n_child);
20.     if (¬ Subsumed (prefix(n_child, CP)))
21.         CP := CP ∪ prefix(n_child);
```

Fig. 4. CPExtend recursively enumerates the complete set of contradiction patterns that satisfy the conflict relevance and relevance deviation thresholds.

(1) Two patterns that select the same set of tuples have the same corresponding closed pattern, which in turn is a superset of the union $\rho_1 \cup \rho_2$.
(2) A pattern ρ_1 that selects a subset of the tuples selected by a pattern ρ_2 has the same corresponding closed pattern to the union $\rho_1 \cup \rho_2$. The corresponding closed pattern for ρ_2, however, is different.

The first property implies that we replace both patterns ρ_1, ρ_2 with $\rho_1 \cup \rho_2$. The second property implies that we only replace ρ_1 with $\rho_1 \cup \rho_2$ while retaining ρ_2 as a separate pattern. We use these properties to modify nodes in the IT-tree by replacing or removing patterns while generating the extension list of n_{child}. Whenever we replace or remove a pattern we traverse the tree from the root to the leaf nodes and replace or remove each occurrence of the pattern in the prefix pattern of a node and in its extension list. In CPMine, we distinguish four cases.

(1) $\rho_i(r) = \rho_j(r)$: Both patterns have the same corresponding closed pattern as ρ_m. We remove ρ_j from the extension list of node n_{curr} and replace every occurrence of ρ_i in the IT-tree with ρ_m.
(2) $\rho_i(r) \subset \rho_j(r)$: The corresponding closed pattern for ρ_i is the same as for ρ_m. We therefore replace every occurrence of ρ_i in the IT-tree with ρ_m.
(3) $\rho_i(r) \supset \rho_j(r)$: The corresponding closed pattern for ρ_j is the same as for ρ_m. We remove ρ_j from the extension list of node n_{curr} and continue with ρ_m instead by adding it to the extension list of node n_{child}.
(4) Else: We add ρ_m to the extension list of node n_{child}.

Clearly, the first case is the most desirable one regarding reduction of the number of patterns in the IT-tree. We allow for this by sorting the patterns according to their support. In addition, pruning incompatible patterns and patterns having relevance deviation above the given threshold eliminates many of the other three cases. After

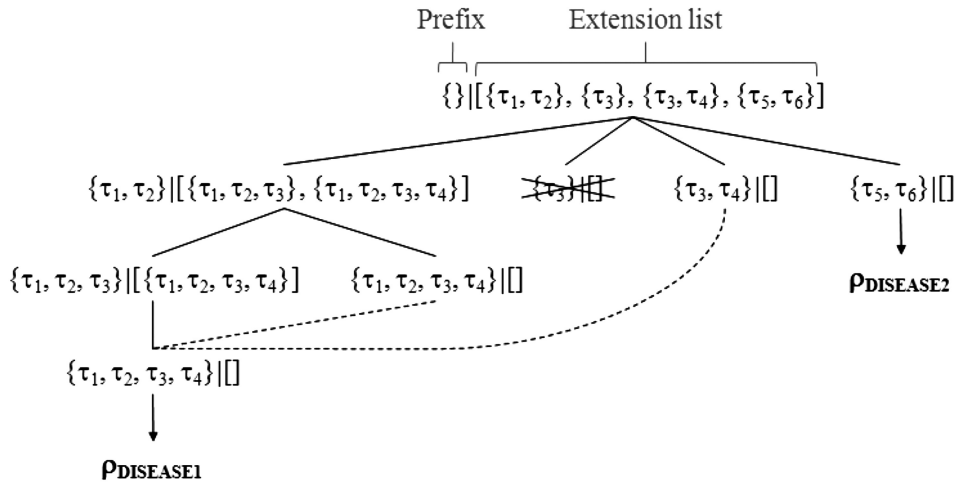


Fig. 5. Example IT-tree when mining closed contradiction patterns for attribute Disease.

generating the extension list of n_{child}, we call CPEXTEND recursively if the extension list is not empty. Once all children of n_{child} have been processed the prefix of n_{child} is added to the result set of contradiction patterns. Note that the prefix may have changed due to replacements performed. We have to check whether the prefix is subsumed by existing patterns in the result set that have been added while traversing the children of n_{child} in recursive calls of CPEXTEND. If the prefix of n_{child} is subsumed by existing patterns it will not be added to the result set.

Example 4.2. Figure 5 shows the resulting IT-tree when mining contradiction patterns for attribute Disease using thresholds $min_{rel} = 0.7$, $min_{pot} = 0.7$, and $max_{dev} = 0.0$. From the initial set of terms shown in Example 4.1, τ_1 and τ_2, and τ_5 and τ_6 are merged and replaced due to case 1 during IT-tree enumeration. Term τ_3 is merged with τ_4 and the latter is replaced by the resulting pattern $\{\tau_3, \tau_4\}$ due to case 2. The relevance deviation threshold prevents $\{\tau_5, \tau_6\}$ to be extended with any other pattern, thus resulting in contradiction pattern ρ_{Disease2} (see Figure 2). The contradiction pattern containing only term τ_3 does not satisfy the conflict potential threshold, and all other patterns are subsumed by ρ_{Disease1}.

CHARM maintains tid-lists for items to perform fast checking of itemset support. In CPMINE, we use disjunctive lists $tidlist_C$ and $tidlist_N$ that represent the primary keys of tuples in $v_C(r_1, r_2, A)$ and $v_N(r_1, r_2, A)$. The tid-list of a pattern is the intersection of the according tid-lists of all of the terms in the pattern. The primary keys in a tid-list are sorted in ascending order to allow efficient intersection. Conflict potential and conflict relevance for candidate patterns can be determined using the length of $tidlist_C$ and $tidlist_N$, that is,

$$rel(\rho_A, r_1, r_2, A) = \frac{|tidlist_C(\rho_A)|}{|v_C(r_1, r_2, A)|}, \quad \text{and} \quad pot(\rho_A, r_1, r_2, A) = \frac{|tidlist_C|}{|tidlist_C| + |tidlist_N|}.$$

With respect to time complexity, consider the general case of mining contradiction patterns for a database instance with n attributes, v tuples, and k different values for each attribute. The total number of terms for such a database is $n \times k$. To generate the list of frequent terms we need to count the frequency for each term. We currently maintain a term index for each attribute. FREQUNITERMPATTERNS therefore performs $n \times v \times log_2(k)$ index lookup operations. In CPEXTEND we enumerate all contradiction patterns

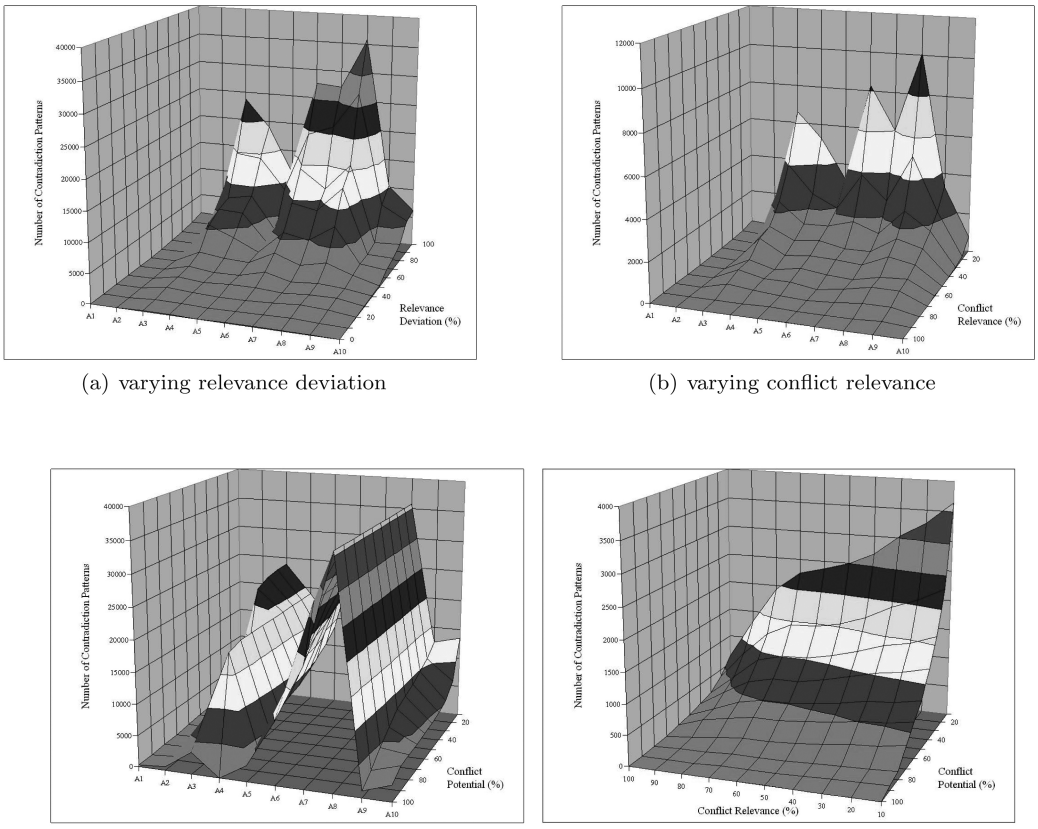
using the set of frequent terms. In the worst case, each term is frequent and overlaps with each of the k terms for each of the compatible attributes. Thus, the total number of contradiction patterns generated by CPExtend is $(k+1)^n$. For each of these patterns we need to compute the conflict relevance and check subsumption. Computing the conflict relevance requires $n \times v$ comparisons to compute the size of the intersection of tid-lists of terms within a pattern. Checking for subsumption, in the worst case, requires to compare the set of terms of each pattern against the set of terms of all other patterns in the current result set. Thus, the overall worst-case time complexity for CPMine is $O(n \times v \times log_2(k)) + O((k+1)^n \times ((n \times v) + (n \times \frac{((k+1)^n+1)\times(k+1)^n}{2})))$.

4.2. Experiments

The algorithms presented in this section were motivated by Columba, a database of integrated protein annotations [Trißl et al. 2005]. Columba pays special attention to the aspect of data quality in merging overlapping databases. The developers of Columba quickly found that the number of inconsistencies are overwhelming and that they therefore need to focus on the most interesting (= most annoying) ones. Using our algorithm CPMine, this focus was achieved leading to the detection of various parser errors, different understanding of data in the databases, and, especially, truly conflicting data. In the following, we show how variations to the interestingness measures used in contradiction pattern mining affect the number of identified patterns and discuss example contradiction patterns found by CPMine.

Data. For our experiments, we use two relational instances of protein structure data. The first relation is directly derived from flat-files of the Protein Structure Database (PDB) [Berman et al. 2000]. The second relation results from parsing mmCIF-files using the OpenMMS Toolkit, a data cleaning project aiming at removing inconsistencies in PDB data [Bhat et al. 2001]. We refer to these datasets as PDB and OpenMMS, respectively. The schema consists of 10 attributes containing the PDB entry identifier, information about the deposition date and year of an entry, the resolution of the protein structure described, a measure of the quality of the atomic model obtained from the crystallographic data (R_Free), as well as the experimental methods used for determining resolution and atomic structures. The relational instance resulting from the PDB flat-files has 26,764 tuples. The instance resulting from OpenMMS contains 24,202 tuples. Identification of matching tuples is trivial using the original ID for all PDB entries. The number of matching pairs between PDB and OpenMMS is 23,614. None of the attributes is free of conflicts, with conflict rates between 0.003% to 100%.

Parameter Values. Figure 6(a) shows the influence of the relevance deviation on the number of patterns returned by CPMine. We fix conflict relevance and conflict potential to 0.02 and 0.25, respectively. For relevance deviation threshold between 0 (no deviation allowed at all) and 0.4 the number of patterns remains almost constant at an overall low level. The number of patterns starts to increase significantly for threshold values that are above 0.5, that is, with decreasing influence of the relevance deviation threshold. Figure 6(b) shows the influence of the conflict relevance parameter using conflict potential and relevance deviation thresholds of 0.25 and 1, respectively. In general, for attributes with a large number of conflicts (e.g., A_7, A_8) the number of patterns increases significantly when lowering the conflict relevance threshold. For attributes with a small number of conflicts (e.g., A_2, A_5) the overall number of terms occurring in conjunction with these conflicts in general is small and so is the number of patterns. The inability of conflict potential to restrict the number of patterns for attributes containing a large number of conflicts is shown in Figure 6(c). For attributes A_7 and A_8 the number of patterns remains (almost) constant. For all other attributes, increasing the conflict potential threshold (with conflict relevance and relevance deviations being



(a) varying relevance deviation

(b) varying conflict relevance

(c) varying conflict potential

(d) varying conflict potential and conflict relevance

Fig. 6. Experimental study showing the influence of varying the threshold values for conflict relevance, conflict potential, and relevance derivation on the number of returned closed contradiction patterns.

0.25 and 1, respectively) leads to a decrease in the number of contradiction patterns. We also performed experiments that vary the conflict relevance and conflict potential thresholds for a fixed relevance deviation of 0.5. Figure 6(d) shows the results for attribute A_5 for which there are 2,853 conflicts between PDB and OpenMMS. Especially for small conflict relevance values the potential significantly constrains the number of patterns. This behavior is desirable when searching for patterns that describe smaller subsets of conflicts in an attribute with high accuracy. Only in a few rare cases we expect to be able to find a single pattern that describes all the conflicts in an attribute. Instead, we are often looking for patterns that are closely related to the occurrence of conflicts.

Pattern Evaluation. In the following, we present a selection of contradiction patterns returned by CPMine for PDB and OpenMMS. These patterns were helpful to the developers of Columba to identify data quality problems. Contradiction patterns summarize and highlight groups of conflicts. Discussion of the common pattern within each group of conflicts with structural biology experts lead to the detection of various errors in the PDB and OpenMMS instances and parsers. This information was used in Columba to resolve data conflicts, and to improve the data parsers. Thus, instead of having to

inspect each conflict individually, sets of conflicts could be solved at once by evaluating each contradiction pattern.

—$\rho_{\text{R\_Free}}$: {(R_Free$_{\text{PDB}}$, '+0.00000E+000')}
Looking at dataset PDB shows that '+0.00000E+000' is the only value that occurs in attribute R_Free. Accordingly, there is a contradiction pattern stating that this value occurs in all conflicts since tuples in OpenMMS have nonzero values for attribute R_Free. The conflicts were a consequence of the flat-file parser for PDB files not considering the value appropriately. The same was true for conflicts in other attributes.

—ρ_{Year} : {(Year$_{\text{PDB}}$, '2000'), (Year$_{\text{OpenMMS}}$, '1900')}
Each time the value '1900' occurs as deposition year in OpenMMS the according value in PDB is '2000'. This pattern has a conflict relevance of about 50%. Conflict potential and relevance deviation are 100% and 0% respectively. Deposition years are extracted from deposition dates of PDB entries. Looking at the PDB flat-file format reveals that the conflicts presumably result from parsing error in the OpenMMS parser in conjunction with the Y2K problem. The deposition date is represented in format DD-MM-YY in flat files. Thus, in some cases a year '00' is transformed into year '1900' by the parser. Note that this is not true for all deposition dates having '00' as their year value. However, knowing that PDB was established in the 1970s the solution for these conflicts is obvious.

—$\rho_{\text{Resolution}}$: {(StructMethod$_{\text{OpenMMS}}$, 'NMR'), (Resolution$_{\text{PDB}}$, '0.0'),
(Resolution$_{\text{OpenMMS}}$, NULL)}
When taking a closer look at attribute Resolution we see that either source contains only a single unique value if the method used for determining atomic structures (StructMethod) is 'NMR' (Nuclear Magnetic Resonance). This fact is not surprising as resolution is not recorded for method NMR. However, the sources represent this fact differently: PDB uses '0.0' while OpenMMS uses NULL.

—$\rho_{\text{NMR\_Structs}}$: {(StructMethod$_{\text{PDB}}$, 'NMR')
(StructMethod$_{\text{OpenMMS}}$, 'NMR, n STRUCTs'), (NMR_Structs$_{\text{PDB}}$, 'n')}
This is actually a group of patterns for different values of n (e.g., 9, 20). Each pattern reveals that in OpenMMS for structure method NMR the number of identified structures is encoded in the method name while PDB separately lists the number of structures in attribute NMR_Structs. Note that the first term of these patterns, that is, (StructMethod$_{\text{PDB}}$, 'NMR'), is not included in patterns for small relevance deviation values due to its frequent occurrence with many of these (syntactically) different conflicts.

5. MINIMAL UPDATE SEQUENCES

Contradiction patterns provide valuable information about systematic differences within a single attribute. The full set of contradiction patterns, however, often contains highly similar and overlapping patterns. Within this section, we introduce a second concept for understanding systematic differences based on sequences of update operations.

Example 5.1. Consider the patterns ρ_{Disease1} and ρ_{Disease2} in Figure 2. Both patterns overlap in 3 out of 4 tuples they select from $v(r_1, r_2)$, namely tuples $t\{2\} - t\{4\}$. Furthermore, the union $\rho_{\text{Disease1}} \cup \rho_{\text{Disease2}}$ may also (depending on thresholds) be included in the set of contradiction patterns for Disease.

In many cases, contradicting databases are (or can be seen as) modified copies of a common ancestor (e.g., Example 1.1). Update sequences that transform one database into the other have to undo and redo update operations that have been applied on the

respective database copies. Update sequences thereby form a compact representation of all the differences between a pair of databases. Furthermore, the order of operations within an update sequence enables identification of dependencies between conflicts in different attributes that are not revealed by contradiction patterns. Such dependencies arise for example in data cleaning workflows [Rahm and Do 2000; Galhardas et al. 2001; Raman and Hellerstein 2001] or in annotation pipelines [Curwen et al. 2004]. Each update sequence is as long as the update distance is one of the simplest possible explanations for the observed differences. Following the "Occams Razor" principle, we conclude that the simplest explanations are usually the most likely.

Example 5.2. Consider the sequence of update operations in Figure 2 that transforms r_1 into r_2. The first operation indicates the different representations for iron cofactors in r_1 and r_2. The second operation suggests erroneous cofactor identification by the second database for peroxisome proteins when the actual cofactor is *FAD*. The third operation outlines a possible dependency between conflicts in Cofactor and Disease. After erroneously determining cofactors Zn^+ the second source may, again in error, have assigned *vitamin D-dependent rickets type 1* (`'VDDR-1'`) involvement to all proteins with previously identified cofactor Zn^+.

Since we consider minimal transformers as explanations for systematic differences between two databases, we want to avoid meaningless (or trivial) update sequences like: (1) delete all tuples in r_1, and then (2) for each tuple in r_2 perform an insert operation. We therefore allow only valid update operations within minimal transformers. For any intermediate state r_i in the process of transforming r_1 into r_2 an operation ψ is *valid*, if $\psi(r_i) \neq r_i$ and:

—ψ is an insert operation and $t_{new} \in u(r_2, r_1)$,
—ψ is a delete operation and $\psi_{DEL}(r_i) \subseteq u(r_1, r_2)$, or
—ψ is a modification operation.

That is, we allow inserts only for tuples from r_2 that do not have a matching partner in r_1, and deletions for tuples in r_1 that haven't got a matching partner in r_2. Modification operations are unrestricted. Note that our algorithms for computing minimal update sequences work for an unrestricted set of update operations. Our definitions for the upper and lower bound of the update distance, however, will have to be modified in order to account for the trivial transformer that is of length $|r_2| + 1$. In the following, we consider valid update operations only.

5.1. Upper and Lower Bounds

In the following, we derive upper and lower bounds that are important for optimization. Based on these definitions, we describe our algorithm for calculating minimal update sequences for a given pair of databases in Section 6. Problem variations and heuristic algorithms are discussed in Section 7 and Section 8.

Example 5.3. To give an idea of the complexity of the problem, consider the databases r_1, r_2 in Figure 7. Clearly, their update distance can be determined by enumerating update sequences of increasing length until one sequence is found that implements all necessary changes. This procedure, however, generates 294,998 intermediate states. An intuitive idea to prune the search space would be to use a greedy strategy, that is, to select at each stage the operation that solves the most conflicts. The shortest sequence found using such an approach has four elements (update sequence a)), although the update distance between databases r_1 and r_2 is three (update sequence c)). Another pruning idea might be to avoid modification operations that introduce new conflicts. This results in only 32 generated intermediate states. However, using this

Databases:

r_1

A_1	A_2	A_3
1	1	1
2	1	1
3	1	1
4	2	1
5	3	1
6	4	1
7	5	1
8	6	1
9	1	0
10	1	0

r_2

A_1	A_2	A_3
1	1	1
2	1	1
3	1	1
4	2	0
5	3	0
6	4	0
7	5	0
8	6	0
9	1	0
10	1	0

Update sequences:

a) `UPDATE` r_1 `SET` A_3 `= 0`
`UPDATE` r_1 `SET` A_3 `= 1 WHERE` A_1 `= 1`
`UPDATE` r_1 `SET` A_3 `= 1 WHERE` A_1 `= 2`
`UPDATE` r_1 `SET` A_3 `= 1 WHERE` A_1 `= 3`

b) `UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_1 `= 4`
`UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_1 `= 5`
`UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_1 `= 6`
`UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_1 `= 7`
`UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_1 `= 8`

c) `UPDATE` r_1 `SET` A_3 `= 2 WHERE` A_2 `= 1`
`AND` A_3 `= 1`
`UPDATE` r_1 `SET` A_3 `= 0 WHERE` A_3 `= 1`
`UPDATE` r_1 `SET` A_3 `= 1 WHERE` A_3 `= 2`

Fig. 7. An example for the need to introduce conflicts in order to find an optimal solution.

heuristic worsens the result, as now the shortest sequence is of length five (update sequence b)). Intuitively, it is often necessary to use operations that in first place introduce new values (and thereby new conflicts) that can be used as discriminating conditions in later update operations. The first operation in update sequence c) temporarily increases the total number of conflicts, but this is compensated in later operations that are now able to solve more conflicts within one statement.

We prove in Section 7.2 that computing the set of minimal transformers is NP-complete when using only a restricted set of operations. While the calculation of the update distance is nontrivial, we can define upper and lower bounds.

LEMMA 5.4. *An upper bound for the update distance between databases r_1, r_2, denoted by $ub(r_1, r_2)$, is given by $|u(r_1, r_2)| + |u(r_2, r_1)|$+$conflicts(r_1, r_2)$.*

PROOF. In order to transform r_1 into r_2, we have to: (i) remove the tuples from r_1 without a matching partner in r_2, (ii) solve the conflicts within the matching pairs, and (iii) insert those tuples that exist in r_2 but not r_1. Due to the primary key and our restriction of modification terms every tuple $t \in r$ is always individually selectable by a pattern $\rho = \{(ID, t[ID])\}$. The deletions are accomplished using a single delete operation for every unmatched tuple in r_1. Each conflict is individually solved using a single modification operation. The inserts are performed by executing an insert operation for every unmatched tuple from r_2. Overall, this requires $|u(r_1, r_2)|$ delete operations, $conflicts(r_1, r_2)$ modification operations, and $|u(r_2, r_1)|$ insert operations. Any sequence of these operations is a transformer for r_1 and r_2. □

To define a lower bound for the update distance, we make use of the following observation. Consider again the databases in Figure 2. For attribute Cofactor, values `'Iron'` and `'Zn`$^+$`'` in r_2 appear in conflicts between r_1 and r_2. Transforming r_1 into r_2 requires at least two modification operations, one having modification term (Cofactor, `'Iron'`) and another having modification term (Cofactor, `'Zn`$^+$`'`). For attribute Disease, `'VDDR-1'` is the only value in r_2 that occurs in conjunction with conflicts. We therefore need at least one more modification operation having modification term (Disease, `'VDDR-1'`). In general, when transforming r_1 into r_2 we need at least one modification operation for each attribute A and each distinct value in r_2 that occurs within a conflict with r_1. That is, for each value $c \in \pi_{A_{i_2}}(v_C(r_1, r_2, A_i))$, $2 \leq i \leq n$, we need at least one modification operation with (A_i, c) as the modification term. Let $NMT(r_1, r_2) = \bigcup_{A_i, 2 \leq i \leq n} A_i \times \pi_{A_{i_2}}(v_C(r_1, r_2, A_i))$ denote the set of *necessary modification terms* when transforming r_1 into r_2.

LEMMA 5.5. *The lower bound for the update distance between a pair of databases r_1, r_2, denoted by $lb(r_1, r_2)$, is given by*

$$lb(r_1, r_2) = |u(r_2, r_1)| + |NMT(r_1, r_2)| + \begin{cases} 1 & \text{if } u(r_1, r_2) \neq \emptyset \\ 0 & \text{else} \end{cases}.$$

PROOF. The proof follows directly from our definition of update operations and the observation that we need at least one modification operation per term in $NMT(r_1, r_2)$. To transform r_1 into r_2 we need to: (a) insert all unmatched tuples in $u(r_2, r_1)$ into r_1 using a single insert operations for each tuple, (b) delete all unmatched tuples in r_1 requiring at least one delete operation if $u(r_1, r_2)$ is not empty, and (c) solve all conflicts between r_1 and r_2 requiring at least $NMT(r_1, r_2)$ modification operations. Any update sequence shorter than $lb(r_1, r_2)$ would either have to insert more than one tuple per insert operation, solve conflicts in more than one attribute per modification operation, or modify values in the ID attribute. This, however, violates our definition of update operations. Likewise, an update sequence that first deletes all tuples in r_1 and then inserts all tuples from r_2 would violate our definition of valid update operations. □

6. TRANSIT - FINDING MINIMAL TRANSFORMERS

This section describes the TRANSIT algorithms to determine the set of minimal transformers for contradicting databases. Given a pair of databases r_o and r_t, called *origin* and *target*, the TRANSIT algorithms enumerate the space of databases reachable by applying sequences of modification operations to r_o. Doing so efficiently poses several challenges for which we describe solutions. We describe two algorithms that resemble a branch-and-bound approach [Land and Doig 1960] based on the bounds defined in Section 5.1. These algorithms differ in the way they explore the search space, that is, in breadth-first or in depth-first manner.

In general, an early execution of insert operations does not provide any benefit regarding the minimization of transformer length. Instead, early inserts bear the chance that following modification or delete operations affect the inserted tuples and cause additional conflicts. Inserts are therefore delayed until the end. Delete operations can be handled as special cases of conflict resolution with modification operations. We postpone a separate treatment of deletes to Section 6.4. Throughout the other sections of this article, we only consider modification operations and restrict the algorithms to database pairs without unmatched tuples.

6.1. Search Space Exploration

We represent the search space using a directed labeled graph, called a *transition graph*. Vertices of this graph are databases connected by directed edges representing modification operations. Let $G_T = (V, E)$ denote the transition graph, with vertices V and edges E. For each edge $e = (r_1, r_2, \psi) \in E, r_1, r_2 \in V$, the edge label ψ represents the modification operation $\psi(r_1) = r_2$. Figure 8 shows an example transition graph. The different levels of update distance are outlined by horizontal lines.

Breadth-First Algorithm. The TRANSIT algorithms iteratively construct the transition graph starting with r_o as the only vertex. In the breadth-first algorithm, we enumerate databases reachable from r_o at levels of increasing update distance. After finishing enumeration of databases at level k we continue by enumerating all databases at level $k + 1$. The procedure ensures that the level at which a database r is first generated represents the update distance $\Delta_U(r_o, r)$.

We use the upper and lower bounds for pruning. Let β denote the current upper bound for the update distance between r_o and r_t. This bound is initialized as $ub(r_o, r_t)$.

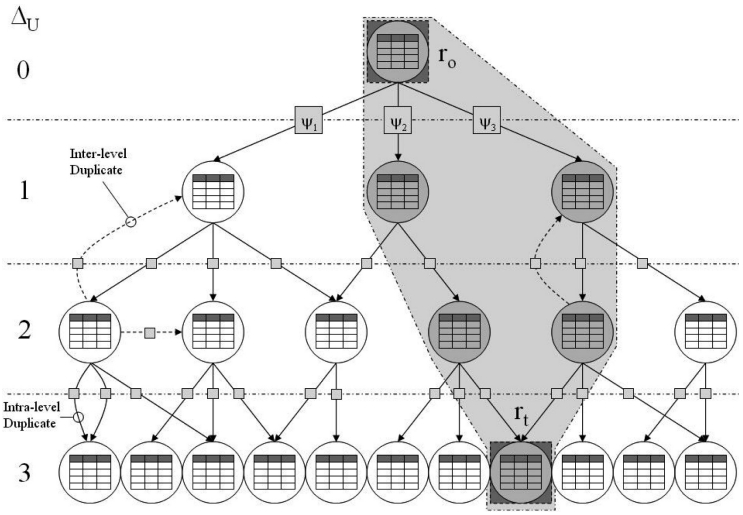


Fig. 8. An exemplified transition graph as generated by the TRANSIT algorithm without pruning.

Procedure TRANSIT-BFS (r_o, r_t)

1. $G_T := (\{r_o\}, \emptyset)$;
2. $V_P := V(G_T)$;
3. $\Delta_U := 0$;
4. $\beta := ub(r_o, r_t)$;
5. **while**$(r_t \notin V_P)$
6. $\quad \Delta_U := \Delta_U + 1$;
7. $\quad V_C := \emptyset$;
8. $\quad$ **for each** $r_i \in V_P$ **do**
9. $\quad\quad$ **for each** $\psi \in$ ***modifier***(r_i, r_t) **do**
10. $\quad\quad\quad r_{new} := \psi(clone(r_i))$;
11. $\quad\quad\quad$ **if** $((lb(r_{new}, r_t) + \Delta_U) \leq \beta)$
12. $\quad\quad\quad\quad$ **if** $(r_{new} \notin V(G_T))$
13. $\quad\quad\quad\quad\quad V(G_T) := V(G_T) \cup r_{new}$;
14. $\quad\quad\quad\quad\quad E(G_T) := E(G_T) \cup \{(r_i, r_{new}, \psi)\}$;
15. $\quad\quad\quad\quad\quad V_C := VC \cup r_{new}$;
16. $\quad\quad\quad\quad\quad$ **if** $((ub(r_{new}, r_t) + \Delta_U) < \beta)$
17. $\quad\quad\quad\quad\quad\quad \beta := ub(r_{new}, r_t) + \Delta_U$;
18. $\quad\quad\quad\quad\quad\quad$ *prune* V_P, V_C, G_T, β;
19. $\quad\quad\quad\quad\quad$ **else if** $(r_{new} \in V_C)$
20. $\quad\quad\quad\quad\quad\quad E(G_T) := E(G_T) \cup \{(r_i, r_{new}, \psi)\}$;
21. $\quad V_P :=$ SORT (V_C);
22. *output* MINPATHS (G_T, r_o, r_t);

Fig. 9. The breadth-first algorithm TRANSIT-BFS.

Databases r with insufficient bound, that is, $\Delta_U(r_o, r) + lb(r, r_t) > \beta$ are excluded from the transition graph. We decrease β whenever generating a database r having an upper bound below the current best bound, that is, $(\Delta_U(r_o, r) + ub(r, r_t)) < \beta$. Such a database ensures that there is a transformer $\Psi(r_o, r_t)$ of length $\Delta_U(r_o, r) + ub(r, r_t)$. Each time β is decreased, we remove all databases from the transition graph with insufficient bound.

The corresponding algorithm TRANSIT-BFS is shown in Figure 9. Let V_P and V_C denote the set of databases from the previous and the current level, respectively. Variable Δ_U

stores the current level of enumeration. We initialize V_P with $\{r_o\}$. Each database in V_P is processed to enumerate V_C. Databases in V_C then become the candidates for enumeration of the next level. We sort the candidates in ascending order of their lower and upper bounds. This is done with the intention of being able to decrease the current bound β as soon as possible. After reaching the target the algorithm returns the set of minimal paths in the transition graph from the origin to the target.

Processing a database r_i from V_P starts by determining the set of possible modification operations, denoted by $modifier(r_i, r_t)$ (see Section 6.3 for details). Each operation is applied to a copy of the database, as modification operations alter the given database. The resulting database r_{new} is added to the transition graph if its lower bound with r_t does not exceed the current bound. Since different update sequences may generate the same database, we have to check for duplicates in the transition graph (see Section 6.2 for details). If r_{new} does not occur within G_T it is added to V_C. If r_{new} does occur within G_T, we have to check at which level it occurs. If the database has been derived before at the current distance level, we add an additional edge to G_T. Otherwise, no changes occur.

Depth-First Algorithm. We refer to the algorithm that constructs the transition graph in depth-first manner as TRANSIT-DFS. Within this algorithm, after finishing the processing of the current database, we immediately proceed to the next distance level. From all generated databases, we chose the one with the smallest lower bound as the new current database. Pruning is performed as described earlier. The depth-first approach finds a first solution after processing fewer databases then the breadth-first approach. Although this solution is not necessarily minimal in the number of modification operations, it often is helpful for pruning. After reaching the target database, TRANSIT-DFS needs to return to the previous databases and process them as candidates, again in a depth-first manner. This is continued until all databases that have not been pruned by the bounding step have been tested. In TRANSIT-DFS we maintain the databases processed and generated on the current path on a stack to enable upward traversal. Compared to TRANSIT-BFS search space enumeration is complicated by the fact that identical databases may be generated multiple times at decreasing levels. Every time a database is repeatedly derived at a lower level, it has to be considered as a candidate again.

6.2. Duplicate Detection

Duplicates in transition graph construction occur whenever the same database is derived by different update sequences. We distinguish between interlevel and intralevel duplicates. *Interlevel duplicates* occur, if update sequences of different length derive the same database, that is, the same database is derived at different levels. Duplicates at different levels of the graph may introduce cycles. Since the corresponding edges (delineated by dotted lines for clarity in Figure 8) cannot be part of a minimal transformer, they are not included in the graph. Thus, the transition graph is acyclic. *Intralevel duplicates* result from different update sequences of equal length that derive the same database. These duplicate databases result in multiple edges between two vertices on adjacent distance levels.

Example 6.1. The operations $\psi_1 = (\{(A_3, 1)\}, (A_3, 0))$ and $\psi_2 = ((\{\}, A_3, 0))$ derive the same result when applied to database r_1 of Figure 7. Update sequences may also derive a database from itself. The update sequence $\langle(\{(A_1, 1)\}, (A_2, 0)), (\{(A_1, 1)\}, (A_2, 1))\rangle$ derives r_1 from r_1 using two update operations.

A large portion of databases generated in transition graph enumeration are duplicates. We need to detect duplicates efficiently to avoid unnecessary explosion of the

search space. Duplicate detection requires comparison of entire databases. To reduce the number of duplicate checks, we compute a hash value for each database and maintain a hash table for generated databases. While there are many possible hashing schemes for databases, we found the following one to be most useful in our case. Without a loss of generality we assume the primary keys to be integers in the range $1, \ldots, m$. Let $vlist(r) = \langle t\{1\}[A_2], \ldots, t\{1\}[A_n], t\{2\}[A_2], \ldots, t\{m\}[A_n]\rangle$ be an ordered list of nonkey values. Let $vlist(r)[i]$ denote the i^{th} element of the list. Given a derived database r and the target r_t, we compute a hash value as follows.

(1) From $vlist(r)$, we remove all elements for which $vlist(r)[i] = vlist(r_t)[i]$. The resulting list, denoted by $vlist(r)^{-r_t}$, is an ordered list of all conflicting values from r.
(2) We select k values $v_1, \ldots, v_k$ from $vlist(r)^{-r_t}$ at equal distant positions, that is, $v_i = vlist(r)^{-r_t}[(i-1) \times (|vlist(r)^{-r_t}|/k)]$, for $1 \leq i \leq k$.
(3) The final hash value is an integer with k digits, where the i^{th}-digit is the result of v_i modulo 10.

During search space exploration we check for equality of a pair of databases r_1, r_2 only if: (i) their hash values are equal, and (ii) $ub(r_1, r_t) = ub(r_2, r_t)$. Overall, this approach drastically reduces the number of full database comparisons.

6.3. Enumeration of Valid Modification Operations

For a database r the set of valid modification operations is a subset of the Cartesian product of the set of possible modification terms and the set of possible modification patterns.

Modification Terms. The set of modification terms is the union of modification terms for each nonkey attribute. For each attribute $A \in R \setminus \{ID\}$ this set is $A \times dom(A)$. A problem is the infinite size of $dom(A)$ that leads to an infinite number of modification terms. Almost all of the generated terms, however, are isomorphic with respect to their ability to participate in a shortest update sequence. We therefore constrain the set of possible modification values. For a pair of databases r, r_t, and attribute A, we permit only the following values from $dom(A)$ for modification terms.

—All values occurring for attribute A in the current database r, that is, $\pi_A(r)$. In some situations increasing the selectivity of individual values enables us to solve more conflicts using a single modification operation afterward.
—All values from r_t that occur within attribute A, that is, $\pi_A(r_t)$. Some of these values are already included in $\pi_A(r)$. The others are needed as modification values for solving conflicts.
—Any of the remaining values from $dom(A) \setminus (\pi_A(r) \cup \pi_A(r_t))$ is a potential modification value to serve as a unique selection criterion in later stages of the algorithm. Thus, the actual value does not matter, as long as it is different from all other currently used values. We choose one value using a random function. We call these values *Skolem constants*.

For example, the first modification operation in update sequence c) in Figure 7 uses Skolem constant 2 as a modification value. The Skolem constants are maintained within a separate list for each attribute, called $skolem(A)$. Within the final modification sequence, the occurring Skolem constants can be replaced by any valid subset of $dom(A)$ of size $|skolem(A)|$ that is disjoint with $\pi_A(r_o) \cup \pi_A(r_t)$.

Selection Patterns. For every pattern ρ, with $\rho(r) \neq \emptyset$, there has to be a modification operation that allows us to modify the tuples in $\rho(r)$. To avoid redundancies in the set of selection patterns, we restrict the set of modification patterns to the set of closed

patterns $P_C(r) \cup \rho_\emptyset$. We add the empty pattern to $P_C(r)$ to allow modifications of the complete database at once. In our algorithms, we use an implementation of CHARM to enumerate the set of closed patterns for a database.

Filtering Modification Operations. When enumerating modification operations for a given database, only operations that actually change the database state are valid (see Section 5). A modification operation has no effect, that is, is invalid, if the modification term τ also occurs within the modification pattern. In this case, all selected tuples already possess the new value in the modified attribute. We remove these operations during enumeration.

6.4. Handling Delete Operations

Delete operations are handled as special cases of modification operations. We therefore need to slightly alter a given database. The set of tuples from r_o to be deleted is given by $u(r_o, r_t)$. We add a special attribute A_D, a delete flag, to schema R setting $t[A_D] = 0$ for each $t \in u(r_o, r_t)$. We also insert the tuples from $u(r_o, r_t)$ to r_t with value $t[A_D] = 1$. For all other tuples in r_o and r_t attribute A_D has the same value $x \neq [0, 1]$. Terms for attribute A_D are excluded when enumerating valid selection patterns (see Section 6.3). However, the term $(A_D, 1)$ is allowed as a modification term in operation enumeration. A modification operation $\psi_{MOD} = (\rho, (A_D, 1))$ represents a delete operation $\psi_{DEL} = (\rho)$. A tuple t with $t[A_D] = 1$ then represents a deleted tuple. Following the definition of valid update operations, we additionally have to restrict ρ in $(\rho, (A_D, 1))$ to select only tuples from $u(r_o, r_t)$.

7. PROBLEM VARIATIONS AND COMPLEXITY

The complexity of the presented TRANSIT algorithms limits their applicability to small databases. In this section, we present problem variations for finding minimal update sequences that intend to reduce the search space. We start by giving a classification of modification operations based on how they change the set of conflicts between a pair of databases. In a variation of the original problem, we then consider only certain classes of modification operations in our algorithms for finding minimal update sequences. Based on our classification, we prove that computing minimal update sequences is NP-complete for a certain class of operations.

7.1. A Classification of Modification Operations

The search space for minimal update sequences is enormous due to the large number of valid modification operations. In a variation of the update distance problem, we reduce the search space by restricting the set of modification operations. While the size of the search space is reduced significantly by this restriction, the update distance will differ depending on the class of modification operations used.

Given databases r, r_t, and modification operation $\psi_{MOD} = (\tau, \rho)$. Let $t \in \rho(r)$ be a tuple affected when applying ψ_{MOD} to r. Let $t_t \in r_t$ be the respective matching partner for t, that is, $t[ID] = t_t[ID]$. Furthermore, let $A = attr(\tau)$ and $v = val(\tau)$ denote the modification attribute and modification value. We distinguish four cases regarding the effects that ψ_{MOD} has on t and conflicts between t and t_t.

—$t[A] = v$: ψ_{MOD} has no effect.
—$t[A] \neq v \wedge t_t[A] = v$: ψ_{MOD} solves an existing conflict.
—$t[A] \neq v \wedge t_t[A] \neq v \wedge t[A] = t_t[A]$: ψ_{MOD} introduces a new conflict.
—$t[A] \neq v \wedge t_t[A] \neq v \wedge t[A] \neq t_t[A]$: ψ_{MOD} alters an existing conflict.

Table II. Using TRANSIT-DFS to Compute $\Delta_U(r_1, r_2)$ for the Databases in Figure 7 with Different Classes of Modification Operations

Operation Class	Databases Tested	Operations Executed	Databases Added	Intra-Duplicates	Inter-Duplicates	Δ_U
CLASS0+Skolems	4,275	603,971	4,204	4,417	4,483	3
CLASS0-Skolems	1,384	134,906	1,384	1,578	1,504	4
CLASS1	36	104	36	38	10	4
CLASS2	31	80	31	49	0	5
CLASS3	31	80	31	49	0	5

Based on these four cases we define the following disjoint subsets of $\rho(r)$.

—NEUTRAL$^+ = \{t|t \in \rho(r) \wedge t[A] = v\}$, that is, the set of tuples for which ψ_{MOD} has no effect.
—NEUTRAL$^- = \{t|t \in \rho(r) \wedge t[A] \neq v \wedge t_t[A] \neq v \wedge t[A] \neq t_t[A]\}$, that is, the set of tuples for which ψ_{MOD} alters an existing conflict.
—NEW $= \{t|t \in \rho(r) \wedge t[A] \neq v \wedge t_t[A] \neq v \wedge t[A] = t_t[A]\}$, that is, the set of tuples for which ψ_{MOD} introduces new conflicts.
—SOLVED $= \{t|t \in \rho(r) \wedge t[A] \neq v \wedge t_t[A] = v\}$, that is, the set of tuples for which ψ_{MOD} solves a conflict.

We call the 4-tuple of sets (NEUTRAL$^+$, NEUTRAL$^-$, NEW, SOLVED) the *modification fingerprint* of ψ_{MOD} for r, r_t. Tuples in NEUTRAL$^+$ and NEUTRAL$^-$ do not represent changes to the set of conflicts other than a possible change of conflicting values. We include these sets for the definition of CLASS3 operations in the following. Based on the modification fingerprint we define four different classes of modification operations.

—CLASS0: The set of all valid modification operations.
—CLASS1: The set of valid modification operations that decrease the overall number of conflicts, that is, |SOLVED|>|NEW|. We call CLASS1 modification operations *conflict reducer*.
—CLASS2: The set of conflict reducers that decrease the overall number of conflicts and do not introduce any new conflicts, that is, SOLVED $\neq \emptyset$ and NEW $= \emptyset$. We call these operations *conflict solver*.
—CLASS3: The set of conflict solvers that solely solve existing conflicts, that is, SOLVED $\neq \emptyset$, NEW $= \emptyset$, and NEUTRAL$^- = \emptyset$. We call these operations *pure conflict solver*.

From our classification it follows that CLASS3 $\subseteq$ CLASS2 $\subseteq$ CLASS1 $\subseteq$ CLASS0. Due to the primary key property and our restriction of modification operations, there always exists a transformer of CLASS3 operations for any pair of databases. It follows that for any of the defined classes there exists a transformer using only operations from that particular class. Our hierarchical classification ensures that the number of modification operations for a given pair of databases is reduced by allowing only operations of a certain class. To reduce the size of generated transition graphs, we change the problem definition to only allow operations of a certain class in minimal transformers. Table II indicates the decrease in the number of databases tested and added to the graph when restricting the set of valid modification operations for TRANSIT-DFS on the databases of Figure 7. The results show that there is already a significant drop-off in the number of databases tested and added when disabling the insertion of Skolem constants (CLASS0-Skolems). Disallowing the insertion of Skolem constants reduces for each attribute the number of valid modification operations by approximately the number of closed patterns. The biggest reduction, however, comes from disallowing CLASS0 operations that potentially increase the number of conflicts. Any further restriction is only marginal in our experiments.

r_o

ID	B_1	B_2	B_3	C
1	1	2	2	2
2	1	1	3	3
3	4	1	1	4
4	5	1	5	5
5	1	6	6	6
6	1	7	1	7
7	8	8	8	8

r_t

ID	B_1	B_2	B_3	C
1	1	2	2	0
2	1	1	3	0
3	4	1	1	0
4	5	1	5	0
5	1	6	6	0
6	1	7	1	0
7	8	8	8	8

Fig. 10. An instance of TRANSIT3 for $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ and $S = \{s_1, s_2, s_3\}$ with $s_1 = \{u_1, u_2, u_5, u_6\}$, $s_2 = \{u_2, u_3, u_4\}$, and $s_3 = \{u_3, u_6\}$.

We described in Section 6.3 how to determine the set of CLASS0 operations. To determine whether an operation is of CLASS1– CLASS3, however, requires significantly more effort. We actually have to access each affected tuple and their respective matching partner. Therefore, we have to execute the operation until a modification occurs that violates the given class definition and revoke the changes. In the worst case, we have to execute the operation completely before being able to decide which class it is in.

7.2. Complexity of Computing Minimal Transformers

We now prove that computing the set of minimal update sequences for a pair of databases using only CLASS3 operations is NP-complete. We refer to the problem as TRANSIT3. The problem is stated as follows: Given a pair of databases r_o, r_t, determine whether there exists a transformer Ψ of CLASS3 operations such that $\Psi(r_o) = r_t$, and $|\Psi| \leq K$.

THEOREM 7.1. TRANSIT3 *is* NP*-complete.*

PROOF. We start by showing that TRANSIT3 is in NP. For a given pair of databases r_o, r_t, the number of different terms that are used for enumerating modification operations is finite due to the finite number of different values for each attribute. Recall that $A \times \pi_A(r_t)$ defines the set of possible terms for attribute A at any point during transformation using CLASS3 operations. The length of modification sequences is bounded by the upper bound and we can guess a modification sequence of length less or equal to K by picking for each operation a valid subset from $A \times \pi_A(r_t)$ as a modification pattern and one term as a modification term. We then test in polynomial time whether the sequence defines a valid TRANSIT3 transformer for the given pair of databases by executing the modification sequence. Therefore, TRANSIT3 is in NP. We now reduce the set cover decision problem to TRANSIT3. The set cover decision problem has been shown to be NP-complete in Garey [1979]. The remaining proof follows directly from Lemma 7.2 through Lemma 7.5. □

The set cover decision problem is defined as follows: Given a universe of elements $U = \{u_1, \ldots, u_n\}$, and a set of subsets $S = \{s_1, \ldots, s_k\}$, with $s_i \subseteq U, 1 \leq i \leq k$. The goal is to decide whether there exists a set cover $C \subseteq S$ such that $U = \bigcup_{s \in C} s$, and $|C| \leq K$. Given an instance (U, S) of the set cover decision problem, we define an instance of TRANSIT3 as follows (see Figure 10 for an example): We represent (U, S) as a database r_o that contains a tuple for each element of U and an attribute for each element of S. The schema of r_o is $R(ID, B_1, \ldots, B_k, C)$ where ID is the primary key, attributes B represent the elements in S, and C is used to introduce conflicts between r_o and a database r_t as described shortly. Let $encode : S \rightarrow \{B_1, \ldots, B_k\}$ be a mapping that maps each element $s \in S$ to the attribute $B \in R$ that encodes the membership of elements in s. For each $u_i \in U, 1 \leq i \leq n$, we add a tuple t to r_o. We set $t[ID] = i$, and $t[C] = i + 1$. For each $s \in S$ we set $t[encode(s)] = 1$, if $u \in s$, or $t[encode(s)] = i + 1$ otherwise. We further add

a tuple t with $t[ID] = n + 1$, and $t[B_j] = t[C] = n + 2, 1 \leq j \leq k$. We further create a database r_t that is equal to r_o except that $t[C] = 0$ for tuples $t\{1\} - t\{n\}$.

We now show the correctness of this reduction. Let Ψ_{C3} denote any minimal transformer $\Psi_{C3}(r_o) = r_t$. The only valid modification term in Ψ_{C3} is $(C, 0)$, denoted by τ_{C0}, that is, $\Psi_{C3} = \langle(\rho_1, \tau_{C0}), \ldots, (\rho_l, \tau_{C0})\rangle$. Any other term introduces or alters conflicts, thus violating the definition of CLASS3 operations. Furthermore, any pattern selecting $t\{n + 1\}$, for example, the empty pattern $\rho_\emptyset$, cannot be used as modification pattern in any operation within Ψ_{C3}. Such an operation would lead to a conflict in $t\{n+1\}$ violating the definition of CLASS3 operations. Let $\Psi^i, 1 \leq i \leq |\Psi|$ denote the prefix of Ψ of length i. Two observations are of importance in the following.

(1) Values in attributes $ID, B_1, \ldots, B_K$ are not changed during any step of executing $\Psi_{C3}(r_o)$, that is, $\pi_{ID,B_1,\ldots,B_k}(\Psi^i_{C3}(r_o)) = \pi_{ID,B_1,\ldots,B_k}(r_o), 1 \leq i \leq |\Psi_{C3}|$. It follows that any term $(A, x), A \in \{ID, B_1, \ldots, B_k\}$ will always select the same set of tuples in any intermediate state $\Psi^i_{C3}(r_o)$.
(2) Every modification operation within Ψ_{C3} selects (and modifies) only a nonempty subset of $\{t\{1\}, \ldots, t\{n\}\}$.

We can immediately describe how to construct a transformer $\Psi(r_o) = r_t$ from a given set cover C. We do so by using patterns $\{(encode(s), 1)\}$ that select the tuples contained in elements $s \in C$ as modification patterns. Let $P(C) = \bigcup_{s\in C}\{(encode(s), 1)\}$ denote the set of patterns selecting subsets of r_o that represent the elements in C. We construct a transformer $\Psi(r_o) = r_t$ from the set of modification operations $P(C) \times \{\tau_{C0}\}$.

LEMMA 7.2. *Any permutation of $P(C) \times \{\tau_{C0}\}$ defines a transformer $\Psi(r_o) = r_t$ of length $|\Psi_{C3}| \leq |C|$.*

PROOF. Following observation (1), every pattern $\rho \in P(C)$ will always select the same set of tuples in any execution of $\Psi(r_o)$. Together, all patterns select the tuples $\{t\{1\}, \ldots\ t\{n\}\}$. A modification operation within a constructed transformer may however be invalid, that is, it does not change the database state it is applied on. This happens if all the tuples selected by its modification pattern have already been modified by previous modification operations, that is, C is not minimal. In this case, however, we can just remove the operation from the sequence. Thus, any permutation of $P(C)\times\{\tau_{C0}\}$ defines a transformer $\Psi(r_o) = r_t$ of valid modification operations of length $|\Psi_{C3}| \leq |C|$. □

We now describe how to derive a set cover from a minimal transformer Ψ_{C3}. Let $P(\Psi_{C3})$ denote the set of modification patterns in Ψ_{C3}. Following observation (2), only patterns with terms (C, x) may select a different set of tuples in $\Psi^i_{C3}(r_o), 1 \leq i \leq |\Psi_{C3}|$ than in r_o. Term $(C, 0)$ is not a valid term in any modification operation of Ψ_{C3}. Any other term $(C, x), x \neq 0$ may at most select a single tuple in r_o and possibly an empty set of tuples in $\Psi^i_{C3}(r_o)$. In the latter case, however, the modification pattern containing (C, x) would result in an invalid modification operation being contained in Ψ_{C3}. Each pattern $\rho \in P(\Psi_{C3})$, therefore, selects the same tuples in r_o as it does when applied as a modification pattern during execution of Ψ_{C3}. In the following, we only consider database r_o.

An important property of the patterns in $P(\Psi_{C3})$ is that each of them selects a set of tuples representing a subset of items in some element $s \in S$. Recall that membership of items in sets $s \in S$ is encoded in tuples $t \in r_o$ by $t[encode(s)] = 1$.

LEMMA 7.3. *For every $\rho \in P(\Psi_{C3})$ there exists at least one $s \in S$ such that $\rho(r_o) \subseteq (encode(s), 1)(r_o)$.*

Procedure GREEDY-TRANSIT (r_o, r_t)

1. $\Psi_T := <>$;
2. $r_s := r_o$;
3. **while**$(r_s \neq r_t)$
4. $\quad \psi_{next} :=$ GREEDYNEXT (r_s, r_t);
5. $\quad$ **if** $(\psi_{next} = \perp)$
6. $\quad\quad \psi_{next} :=$ *pick conflict randomly from* r_s, r_t;
7. $\quad r_s := \psi_{next}(r_s)$;
8. $\quad$ *append* ψ_{next} *to* Ψ_T;
9. **return** Ψ_T;

Fig. 11. The greedy algorithm to calculate the update distance of a pair of databases.

PROOF. If $\rho(r_o)$ contains only one tuple the lemma is true, as every tuple represents an item from U which has to be contained in at least one $s \in S$. If $\rho(r_o)$ contains more than one tuple correctness follows directly from the terms that may occur within patterns. The only terms that are satisfied by more than one tuple are $(encode(s), 1), s \in S$. For ρ to select more than one tuple, it may only contain (one or more) terms $(encode(s), 1)$. For each of these terms $\rho(r_o) \subseteq (encode(s), 1)(r_o)$ is true. □

Let $S|\rho = \{s|s \in S \wedge \rho(r_o) \subseteq (encode(s), 1)(r_o)\}$ denote the elements from S for which their respective tuples in r_o form a superset of $\rho(r_o)$. For each $s \in S|\rho$ we can replace ρ by $\{(encode(s), 1)\}$ in the respective modification operation in Ψ_{C3}.

LEMMA 7.4. *None of the elements in S can occur in more than one set $S|\rho$ for the patterns in $P(\Psi_{C3})$.*

PROOF. Assume that for any $s \in S$ it holds that $s \in S|\rho_1$ and $s \in S|\rho_2$, for $\rho_1, \rho_2 \in P(\Psi_{C3})$. It follows that $\rho_1(r_o), \rho_2(r_o) \subseteq (encode(s), 1)(r_o)$ In this case, we can replace (ρ_1, τ_{C0}) and (ρ_2, τ_{C0}) with $(\{(encode(s), 1)\}, \tau_{C0})$ and Ψ_{C3} would not be minimal. □

Based on Lemma 7.4, we can derive a set cover for U from a given Ψ_{C3} as follows: for each $\rho \in P(\Psi_{C3})$ randomly pick one of the elements in $S|\rho$. Let $C(\Psi_{C3})$ denote such a set cover derived from Ψ_{C3}.

LEMMA 7.5. *$C(\Psi_{C3})$ defines a set cover for U of size $|\Psi_{C3}|$.*

PROOF. Clearly the elements in $C(\Psi_{C3})$ fully cover U as each element represents a superset of a pattern in Ψ_{C3}. Lemma 7.4 ensures that $C(\Psi_{C3})$ contains $|P(\Psi_{C3})| = |\Psi_{C3}|$ distinct elements. □

8. UPDATE DISTANCE FOR LARGE DATABASES

In this section, we describe two different heuristics that allow computation of update sequences for large databases while not necessarily finding the best (exact) solution. In our experiments, we analyze the quality of the computed results and show that even a simple greedy approach gives results of good accuracy.

8.1. Greedy TRANSIT

A first heuristic to cope with the computational complexity of the problem is to apply a greedy algorithm. Figure 11 shows the algorithm GREEDY-TRANSIT that returns a single transformer for a given pair of databases r_o, r_t. Let r_s denote the current starting point. GREEDY-TRANSIT selects at each level the modification operation ψ_{next} that reduces the number of conflicts between r_s and r_t most (algorithm GREEDYNEXT described shortly).

```
Procedure GreedyNext (r_s, r_t)
1.  ψ_max := ⊥;
2.  min_sup := 2;
3.  while (ρ = NextPattern (r_s, min_sup))
4.     for each τ ∈ NMT(r_s, r_t) do
5.        if (|π_ID(ρ(r_s)) ∩ π_ID(τ(r_t))| ≥ min_sup)
6.           r_c := (τ,ρ)(r);
7.           if (ub(r_s, r_t) − ub(r_c, r_t) ≥ min_sup)
8.              ψ_max := (τ,ρ);
9.              min_sup := (ub(r_s, r_t) - ub(r_c, r_t)) + 1;
10. return ψ_max;
```

Fig. 12. GreedyNext avoids enumerating the complete set of modification operations by interleaving pattern generation and operation enumeration.

The starting point for the next level is then set to $\psi_{next}(r_s)$. The algorithm terminates when r_t is reached. The database chosen as the starting point always contains fewer conflicts with r_t than any of the previous databases. Therefore, duplicated databases cannot occur.

Example 8.1. For the databases of Figure 7 the transformer returned by Greedy-Transit, that is, update sequence a), has a length of 4, whereas the update distance $\Delta_U(r_1, r_2)$ is actually 3.

The main challenge for the greedy algorithm is the enumeration of modification operations to determine ψ_{next}. Enumerating the complete set of modification operations is infeasible for large databases due to the large number of closed patterns. However, it is also not necessary. We avoid enumerating modification operations that are no candidate for the final transformer by interleaving closed pattern mining with operation enumeration. The algorithm, called GreedyNext, is outlined in Figure 12. Let min_{sup} denote the minimal support (in number of tuples) a closed pattern has to satisfy in order to be considered in modification operation enumeration. We use min_{sup} as support constraint for pattern mining (NextPattern). Whenever a modification operation is enumerated that performs better, that is, solves more conflicts, than the currently best operation ψ_{max}, we are able to increase min_{sup} according to the number of conflicts solved. That is, min_{sup} is set to *number of conflicts solved by* $\psi_{max} + 1$. We thereby avoid further enumeration of patterns that cannot solve more conflicts than the new ψ_{max}.

For each closed pattern ρ (satisfying min_{sup}) that is returned by the mining algorithm NextPattern (not shown), we enumerate all modification operations that are able to reduce the number of conflicts more than the current ψ_{max}. Recall that $NMR(r_s, r_t)$ denotes the set of necessary modification terms when transforming database r_s into r_t (Section 5.1). When enumerating modification operations for ρ only operations having a modification term $\tau \in NMR(r_s, r_t)$ can reduce the number of conflicts. Furthermore, at least min_{sup} of the tuples in $\rho(r_s)$ have to have a matching partner $t \in r_t$ with $t[attr(\tau)] = val(\tau)$. That is, the matching partners have to possess the modification value in order to solve conflicts. Thus, the operation (τ, ρ) can only solve more conflicts than ψ_{max} if $|\pi_{ID}(\rho(r_s)) \cap \pi_{ID}(\tau(r_t))| \geq min_{sup}$ holds.

Greedy-Transit calls GreedyNext for each database r_s. Note that the result of GreedyNext can be empty as we use 2 as the initial min_{sup}. In this case, we solve one of the existing conflicts randomly using the *ID* of the tuple where the conflict occurs as selection criteria.

Procedure GREEDY-SOLUTION-COST (r_o, r_t, τ)

```
1.  cost := 0;
2.  s_t := s_target(r_o, r_t, τ);
3.  s_n := s_neutral(r_o, r_t, τ);
4.  P_valid := P_C(r_o) ∪ {ρ_∅};
5.  for each ρ ∈ P_valid do
6.     if ((ρ(r_o) ⊂ s_n)||(ρ(r_o) \ (s_t ∪ s_n) ≠ ∅))
7.        P_valid := P_valid \ ρ;
8.  while (s_t ≠ ∅)
9.     ρ_max := MaxSelect (P_valid, s_t);
10.    s_t := s_t \ ρ_max(s_t)
11.    P_valid := P_valid \ ρ_max;
12.    cost := cost + 1;
13. return cost;
```

Fig. 13. A greedy algorithm for calculating the solution cost.

8.2. Approximation of Update Distance

Our second heuristic algorithm approximates the update distance based on solving groups of conflicts independently. We thereby completely disregard the possible impact that the modification of values for some of the tuples may have on solving conflicts for other tuples. The sum of operations needed for solving all conflicts then represents an approximation for the update distance.

Given a pair of databases r_o, r_t, we consider conflicts that are solvable using the same modification term independently. For each term $\tau \in NMT(r_o, r_t)$, let $s_{target}(r_o, r_t, \tau) = (r_o \setminus \tau(r_o)) \ltimes_{ID} \tau(r_t) = \sigma_{attr(\tau) \neq val(\tau)}(r_o) \ltimes_{ID} \sigma_{attr(\tau)=val(\tau)}(r_t)$ denote the set of tuples in r_o that have a conflict in $attr(\tau)$ with their matching partner in r_t. We call s_{target} the *solution target set* as it defines the set of tuples whose conflict in $attr(\tau)$ can be solved using τ as modification term. Furthermore, let $s_{neutral}(r_o, r_t, \tau) = \tau(r_o) \ltimes_{ID} \tau(r_t)$ denote the set of tuples that are unaffected by a modification operation having τ as modification term. We call $s_{neutral}$ the *solution neutral set*. When approximating the update distance, for each group of conflicts defined by $s_{target}(r_o, r_t, \tau)$ we want to find a minimal number of modification operations using τ to solve the conflicts in $s_{target}(r_o, r_t, \tau)$. However, finding such a minimal number of modification operations is still expensive (as shown for example in Section 7.2 when using CLASS3 operations). We therefore implement a greedy approach called GREEDY-SOLUTION-COST that is shown in Figure 13. The algorithm starts by determining the set P_{valid} of modification patterns that: (i) select at least one tuple from $s_{target}(r_o, r_t, \tau)$, and (ii) only select tuples from $s_{target}(r_o, r_t, \tau) \cup s_{neutral}(r_o, r_t, \tau)$. We then choose the pattern ρ_{max} that selects the largest subset from s_t, which initially contains the tuples in $s_{target}(r_o, r_t, \tau)$. We remove ρ_{max} from P_{valid} and $\rho_{max}(s_t)$ from s_t. The algorithm terminates when s_t is empty, that is, all tuples in $s_{target}(r_o, r_t, \tau)$ have been selected.

Our algorithm for approximating the update distance, called TRANSIT-APPROX, calls GREEDY-SOLUTION-COST for each $\tau \in NMT(r_o, t_r)$. Since $NMT(r_o, t_r)$ represents the set of necessary modification terms when transforming r_o into r_t, we ensure that our approximation considers all the conflicts that exist between r_o and r_t. Note that this approximation can be greater or smaller than the actual update distance. The first case occurs whenever there are positive side-effects of solving conflicts in one attribute for solving conflicts in other attributes. The latter occurs whenever the respective modification operations interfere with each other, that is, after executing one of them, the other is no longer executable or has a different result.

8.3. Experimental Results

We now discuss the results of our experimental evaluation of the described heuristics. The main aim of our experiments is to assess the accuracy of GREEDY-TRANSIT and TRANSIT-APPROX in determining the update distance. All experiments were performed on a Citrix MetaFrame Server containing two Intel Xenon 2,4 GHz processors and 4GB main memory.

Data. For the first series of experiments we extracted three datasets from the Ensembl public MySQL Server[1] having 10 attributes and 100, 1,000, and 10,000 tuples respectively. These databases are used as origins r_o in our experiments. For each of the databases we generate modified copies (target databases) r_t using update sequences of randomly picked modification operations of length $5, 10, \ldots, 50$. We refer to the update sequence that generated r_t from r_o as the *generating sequence*. For each pair of databases r_o, r_t we then compute the update distance $\Delta_U(r_o, r_t)$ using GREEDY-TRANSIT and TRANSIT-APPROX. The chosen setup provides us with a tighter upper bound for the update distance, that is, the length of the generating sequence, in order to assess the accuracy of our algorithms.

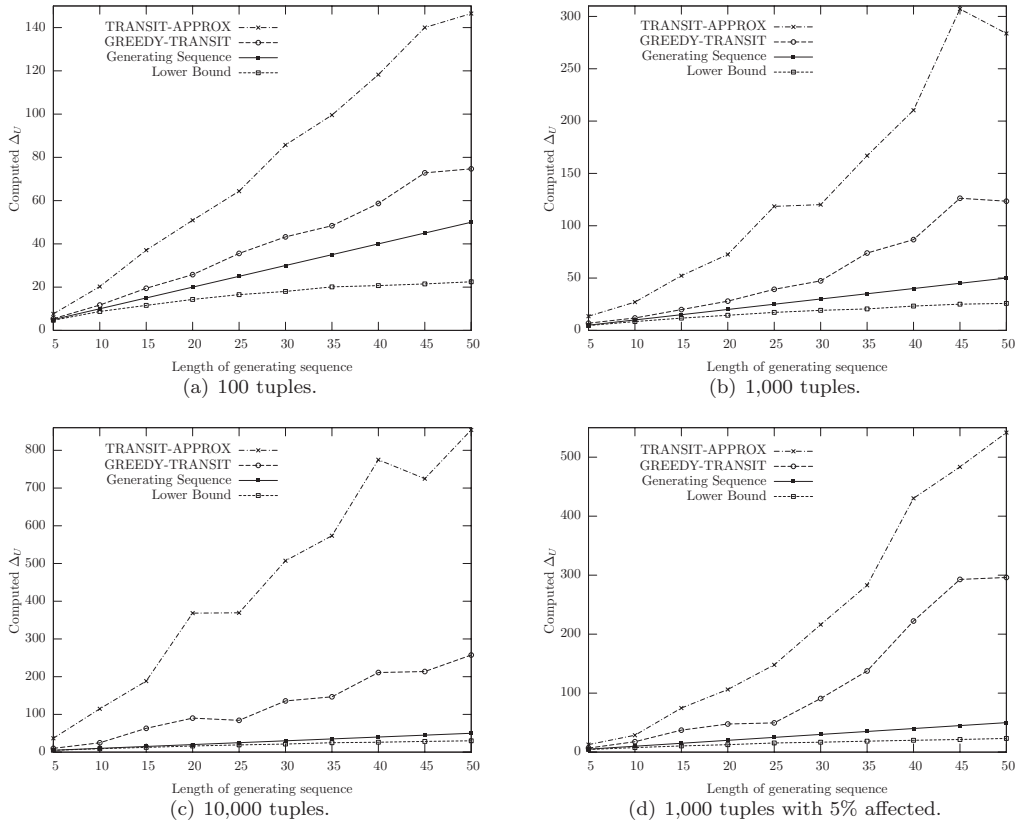
Results. The resulting update distances computed by GREEDY-TRANSIT and TRANSIT-APPROX for the three databases are shown in Figure 14(a) through Figure 14(c). All values are averaged over ten runs. The actual update distance for the databases is between the length of the generating sequence (shown as a solid line) and the lower bound. Note that the generating sequence is not necessarily minimal. The greedy approach and the approximation are both surprisingly accurate for short update sequences. For longer update sequences the accuracy decreases but remains in reasonable bounds. The main reason for this increase is: (a) in the increasing number of conflicts, and (b) in the higher probability of dependencies between operations in the generating sequence. Overall, GREEDY-TRANSIT performs better than TRANSIT-APPROX due to the advantage of taking the order of operations into account. Recall that TRANSIT-APPROX regards all conflicts independently. Thus, it does not benefit from previously executed modification operations as GREEDY-TRANSIT does. As a result, TRANSIT-APPROX more often has to solve remaining conflicts randomly one by one, leading to higher update distances.

When generating the contradicting databases for the accuracy experiments, we randomly chose one operation from the set of valid modification operations for the current database. The accuracy of GREEDY-TRANSIT and TRANSIT-APPROX decreases if we restrict the chosen modification operation to affect a minimum of n tuples. Figure 14 shows the update distances computed by both algorithms when allowing only modification operations whose patterns select at least 5% of tuples (for the database having 1,000 tuples in total). Using patterns that affect more tuples increases the number of conflicts between the resulting databases without increasing the length of the generating sequences. Again, the limited ability of the heuristic algorithms to detect update operations that influenced each other leads to a decrease in accuracy with increasing number of conflicts between the databases. The update distance computed by GREEDY-TRANSIT, however, is still less than 10% of the overall number of conflicts.

Overall, the greedy approach outperforms the approximation in accuracy. On the other hand, the execution time for TRANSIT-APPROX is only a few milliseconds for the tested database while for the GREEDY-TRANSIT it is between 875 and 74,000 ms.

We further applied GREEDY-TRANSIT on the protein structure databases OPENMMS and PDB having nearly 100,000 conflicts between them. The resulting update sequence contained 15,267 update operations and computation took more than 24 hours. The result in Figure 15 shows that over 97% of the operations in the sequences solved

[1]http://www.ensembl.org/info/data/mysql.html.



(a) 100 tuples. (b) 1,000 tuples.

(c) 10,000 tuples. (d) 1,000 tuples with 5% affected.

Fig. 14. Experimental study showing the accuracy of GREEDY-TRANSIT and TRANSIT-APPROX for databases of different size.

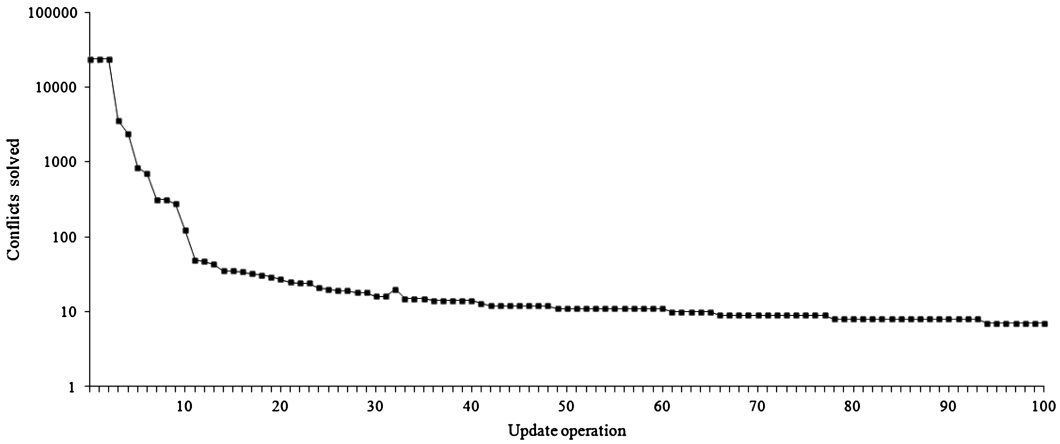


Fig. 15. The figure shows the number of conflicts solved by the first 100 operations in the update sequence for databases OpenMMS and PDB. The total sequence contains 15,267 operations.

less than 10 conflicts. This effect might be an argument for the disadvantage of the greedy approach. However, it more likely points toward the fact that a large number of arbitrary conflicts exists between the databases that do not follow a systematic reason. By interpreting the operations at the start of the sequence, we discovered update operations that describe the commonly known systematic differences between the databases, like usage of different value representations or vocabularies in some of the attributes.

9. RELATED WORK

Within this section we review existing work related to conflict resolution, contradiction pattern mining, and database update distance.

9.1. Conflict Resolution

While there has been a large body of work on the first two steps in data integration, that is, schema mapping and transformation, and object identification, conflict resolution has received little research attention so far. In fact, many data integration platforms rely on the user to decide and specify how to cope with data value conflicts. Bleiholder and Naumann were the first to give a classification of conflict handling strategies [Bleiholder and Naumann 2006]. According to their classification of conflict ignorance, conflict avoidance, and conflict resolution, our work focuses on the third strategy. Conflict resolution in general is done using a set of resolution functions [Naumann and Häussler 2002]. These functions are applied for individual attributes under certain conditions in a specified order. Recent work proposes a declarative specification of conflict resolution strategies [Naumann and Häussler 2002; Bleiholder and Naumann 2005]. Our work is orthogonal to this body of work. We aim to assist the user in identifying conflict reasons, providing information which then is used to specify a conflict resolution strategy.

In Fan et al. [2001], the authors discern between context-dependent and context-independent conflicts. Context-dependent conflicts represent systematic disparities, which are consequences of conflicting assumptions or interpretations. Context-independent conflicts are idiosyncratic in nature and are consequences of random events, human errors, or imperfect instrumentation. According to their separation, we consider context-dependent conflicts. However, in contrast to Fan et al. [2001], we do not discover complex data conversion rules for conflict resolution, but only assist in assessing the quality of contradicting values. Discovering conflict conversion rules is considered part of future work.

Recently, there has been work on identifying true values from contradicting values provided by different data sources on the Web [Dong et al. 2009; Galland et al. 2010; Yin et al. 2007]. Based on different probabilistic models, each of the approaches estimates the most likely true state of the data from a given set of contradicting sources. All approaches rely heavily on the fact that there exists a large number of contradicting sources. The algorithms presented in this article, on the other hand, are designed for pairs of data sources, as in the structural biology example (Example 1.1). Furthermore, our work intends to outline possible reasons for contradictions in order to improve the data generation process and avoid such problems in the future.

9.2. Contradiction Pattern Mining

Contradiction patterns were initially defined in Müller et al. [2004] based on an a priori approach for pattern mining [Agrawal and Srikant 1994]. A priori mining approaches usually generate a huge number of redundant rules, thus overwhelming the user with redundant patterns. Within this article, we present a new algorithm (CPMine) for contradiction pattern mining that is based on closed contradiction patterns. Closed

patterns, in general, avoid descriptional redundancies by summarizing sets of patterns that select the same set of tuples. The set of closed contradiction patterns returned by CPMine in general is orders of magnitude smaller than the set of patterns in Müller et al. [2004] while still containing all the information needed by the expert user to identify possible reasons for systematic differences.

There is a close relationship between contradiction patterns and class association rules [Liu et al. 1998]. In general, an association rule $\rho_1 \Rightarrow \rho_2$ in a database r is a statement about the co-occurrence of patterns ρ_1, ρ_2 in tuples of r, that is, tuples that satisfy ρ_1 also satisfy ρ_2 with a certain confidence and support [Agrawal and Srikant 1994]. A class association rule is an association rule having a right-hand side that contains a single item. Accordingly, a contradiction pattern ρ_A can be interpreted as a class association rule $\rho_A \Rightarrow \{(C_A, \texttt{'true'})\}$ for $v(r_1, r_2)$. It is easy to show the relationship between conflict potential and conflict relevance for contradiction patterns and support and confidence for class association rules. However, there is no adequate parameter to specify a relevance deviation threshold in class association mining algorithm CBA-RG presented in Liu et al. [1998]. Similar to contradiction patterns defined in Müller et al. [2004], CBA-RG is based on an a priori approach for pattern mining [Agrawal and Srikant 1994].

The comparison of datasets using statistical methods is also described in Bay and Pazzani [2001] and Webb et al. [2003]. In contrast to our approach, the authors do not compare overlapping data sources. Instead, the authors try to identify trends or other noticeable differences between datasets. For example, comparing customer data from different branches can reveal customer preferences and behavior by region, group, or month. In Bay and Pazzani [2001] the authors present STUCCO, an algorithm for mining contrast sets. A contrast set is basically a pattern having a meaningful different distribution in different datasets. Contradiction patterns can be seen as a special case of contrast sets by considering the set of conflicting and nonconflicting matching pairs for an attribute as the different sets to be compared. The algorithm presented in Bay and Pazzani [2001], however, is designed to identify patterns that show any kind of difference in their distribution. For contradiction patterns we are mainly interested in patterns that have significantly large difference in their distribution between the two datasets. Thus, the set of contradiction patterns is significantly smaller than the number of contrast sets. In Webb et al. [2003] contrast set mining is compared with decision tree induction and class association rule mining in their ability to describe differences between two datasets. The patterns found by each of the methods were assessed regarding their potential usefulness. The results for two different retailer datasets show that class association rule mining has the overall best ability to identify potentially surprising and useful patterns. Our approach is closely related to class association rule mining. However, we use a closed pattern mining approach to reduce redundancies in the set of contradiction patterns. Furthermore, we define an additional measure of interestingness to help focus on those patterns that contain only terms that occur in close conjunction with conflicts between the overlapping databases.

In general, statistical methods are becoming a popular tool to assist in quality assessment and data cleaning. For example, association rule mining is used to derive integrity constraints [Maletic and Marcus 2000; Hrycej and Hipp 2004]. Recently, theses ideas have been revived for mining conditional functional dependencies [Chiang and Miller 2008; Golab et al. 2008; Fan et al. 2009]. Conditional Functional Dependencies (CFDs) are traditional functional dependencies that hold on a database under conditions [Fan et al. 2008]. The common goal of all these mining approaches is to find a set of patterns that hold with high confidence over (part of) a given database. Tuples that fail the mined association rules or CFDs are considered violations, that is, values of poor data quality. Contrary to our mining approach, in Maletic and Marcus [2000], Hrycej

and Hipp [2004], Chiang and Miller [2008], Golab et al. [2008], and Fan et al. [2009] the violations (or conflicts) are not known a priori, that is, the goal is to identify the violations and not to describe characteristics for a given set of conflicts. One interesting aspect, however, arises from the work in Golab et al. [2008] where the goal is to find an optimal (nonoverlapping) set of patterns. We currently consider mining a minimal set of nonoverlapping contradiction pattern one possible direction of future work.

9.3. Update Distance

The problem of finding minimal sequences of set-oriented operations for relational databases was first considered in Müller et al. [2006]. Within this article, we extend the work of Müller et al. [2006] by presenting variations of the original problem and establishing complexity bounds for one of these variations. Furthermore, we describe an approximation algorithm for the general update distance problem.

There exist various distance measures for objects other than databases, like the well-known Hamming distance [Hamming 1950] or the Levenshtein distance [Levenshtein 1966] for binary codes and strings. The updated distance for databases follows the Levenshtein distance, defined as the minimal number of edit operations necessary to transform one string into another. The only other distance measure for databases, which is related to this definition, is defined in Arenas et al. [1999]. Here, the distance of two databases is defined as the number of tuples from each of the databases without a matching partner in the other database. The definition is used in the area of computing consistent query answers for inconsistent databases [Arenas et al. 1999; Chomicki and Marcinkowski 2005; Wijsen 2002]. Given a query Q, a set of integrity constraints IC, and a database r, which violates IC, the consistent query answering problem is to determine the set of tuples that satisfy Q and are contained in each possible repair for database r. A repair for database r is defined as a database r, which satisfies IC and is minimal in distance to r in the class of all databases satisfying IC [Arenas et al. 1999]. While the approaches Arenas et al. [1999] and Chomicki and Marcinkowski [2005] only allow insertion and deletion of tuples in order to find the repairs, Wijsen [2002] also considers the modification of existing values. All these approaches rely on integrity constraints to identify contradicting values. For the update distance problem, on the other hand, the repair is already given by the target database. Therefore, one is not interested in finding the nearest database in a plethora of possible repairs for an inconsistent database, but in identifying update sequences that transform given databases into each other.

The manipulation of existing database values to satisfy a given set of integrity constraints is also considered in Bohannon et al. [2005]. In their approach, modification as well as insertion of tuples is allowed. A certain cost is assigned with each modification and insertion operation. For a given database and a set of integrity constraints, which are violated by the database, the problem then is to find a repair, that is, a database satisfying a given set of constraints, with minimal cost. Again, the update distance problem is not about determining the optimal value modifications in order to solve a set of conflicts, as the solutions of existing conflicts are predetermined by the target database. The focus is rather on how to perform the (a priori known) necessary modifications with minimal effort in terms of the number of SQL-like update operations. All other approaches described so far do not consider this problem, as they implicitly expect to modify the values one at a time after they determine a conflict solution.

So-called ”update deltas” are used in several applications to represent differences between databases. In database versioning they are used as memory effective representation of different database versions [Dadam et al. 1984]. However, versioning collects the actual operations during execution instead of having to re-engineer them

from two given versions. In Labio and Garcia-Molina [1996] sequences of insert, delete, and update operations are used to represent differences between database snapshots. However, only operations that affect a single tuple are considered. Since databases are manipulated with set-oriented SQL commands, we consider an approach using set-oriented update operations as more natural than a tuple-at-a-time approach. The detection of minimal sequences of update operations is considered in Chawathe and Garcia-Molina [1997] for hierarchically structured data. The authors consider an extended set of update operations to meet the requirements of the manipulation of hierarchically structured data. The data is represented as a tree structure and there are operations that delete, copy, or move complete subtrees. However, the corresponding update operation, that is, to manipulate single data values, considered in Chawathe and Garcia-Molina [1997] is tuple (or node)-at-a-time.

Buneman et al. study the complexity of updating relational databases through views [Buneman et al. 2002]. The view deletion problem is to find a minimal set of tuples to be deleted from a database in order to delete a tuple t from a view defined by a conjunctive query. Buneman et al. show a dichotomy in the complexity, that is, the problem is either in P or is NP-hard for queries in the same class. We consider our work as being orthogonal to the problems considered in Buneman et al. [2002]. That is, we are interested in finding minimal sequences of update operations rather that minimal sets of tuples that need to be updated (or deleted). The complexity in our case is caused by the *order* of update operations as operations may influence each other. Buneman et al. [2002], on the other hand, try to identify a minimal set of tuples to be modified. They do not, however, consider the sequence of update operations that updates the tuples.

10. CONCLUSION

Contradicting databases provide valuable information for data cleaning, provided that we are able to assess the quality of conflicting values effectively. Within this article we present two concepts of systematic conflicts intended to assist the task of quality assessment for data sources and data values. Contradiction patterns are a quick way to highlight interesting groups of systematic conflicts. We define interestingness measures for contradiction patterns and adopt existing association mining algorithms for mining contradiction patterns. The effectiveness of contradiction patterns has been shown in COLUMBA, where the algorithms presented in this article helped to detect and resolve various errors and differences between a pair of real-world databases. Minimal update sequences, on the other hand, are a compact representation for all contradictions between a pair of databases. Furthermore, the order of operations within an update sequence enables identification of dependencies between conflicts that, for example, arise from data cleaning workflows or in genome annotation pipelines. We describe algorithms and problem variations for finding minimal update sequences. While the problem of finding such sequences is expensive, we define heuristic algorithms and demonstrate their high accuracy within our experimental evaluation.

All algorithms presented within this article operate on relational databases consisting of a single relation. While the problem of mining contradiction patterns can easily be extended to databases with multiple relations, the problem of finding minimal update sequences for such databases remains an open problem. We also consider several other extensions to our minimal update sequence approach. First, enhancing the expressiveness of modification operations, including modifications like `SET A = f(A)` as described in Fan et al. [2001], would be very important; yet the cost of finding such functions is probably prohibitive. Second, assuming that two contradicting databases have been derived from a single ancestor database, it is natural to ask the following question (studied in biology under the term phylogenetics): Given a pair of databases r_1, r_2, compute the database r whose update distance to r_1 plus its update distance to r_2

is minimal. We refer to database r as the database with minimal *phylogenetic distance*. Solving the minimal phylogenetic distance problem would allow us to reconstruct the original database.

ACKNOWLEDGMENTS

The authors would like to thank Floris Geerts for his valuable suggestions and remarks that greatly improved this article, and Silke Trissl and Kristian Rother from the Columba Team for providing access to the Columba dataset and for their assistance in contradiction pattern evaluation.

REFERENCES

Abiteboul, S., Cluet, S., Milo, T., Mogilevsky, P., Simon, J., and Zohar, S. 1999. Tools for data translation and integration. *IEEE Data Engin. Bull. 22,* 1, 3–8.

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*.

Arenas, M., Bertossi, L., and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'99)*.

Baumgartner, W. A., Cohen, K. B., Fox, L. M., Acquaah-Mensah, G., and Hunter, L. 2007. Manual curation is not sufficient for annotation of genomic databases. *Bioinformatics 23,* 13, i41–i48.

Bay, S. D. and Pazzani, M. J. 2001. Detecting group differences: Mining contrast sets. *Data Min. Knowl. Discov. 5,* 3, 213–246.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., and Bhat, T. N. e. a. 2000. The protein data bank. *Nucl. Acids Res. 28,* 1, 235–242.

Bhat, T. N., Bourne, P., Feng, Z., Gilliland, G., and Jain, S. e. a. 2001. The PDB data uniformity project. *Nucl. Acids Res. 29,* 1, 214–218.

Bleiholder, J. and Naumann, F. 2005. Declarative data fusion—Syntax, semantics, and implementation. In *Proceedings of the 9th East European Conference on Advances in Databases and Information Systems*.

Bleiholder, J. and Naumann, F. 2006. Conflict handling strategies in an integrated information system. In *Proceedings of the IJCAI Workshop on Information on the Web (IIWeb)*.

Bleiholder, J. and Naumann, F. 2008. Data fusion. *ACM Comput. Surv. 41,* 1, 1–41.

Bohannon, P., Fan, W., Flaster, M., and Rastogi, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*.

Boutselakis, H., Dimitropoulos, D., Fillon, J., Golovin, A., and et al., K. H. 2003. E-msd: the european bioinformatics institute macromolecular structure database. *Nucl. Acids Res. 31,* 1, 458–462.

Brenner, S. E. 1999. Errors in genome annotation. *Trends Genet. 15,* 4, 132–133.

Buneman, P., Cheney, J., Tan, W.-C., and Vansummeren, S. 2008. Curated databases. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'08)*.

Buneman, P., Khanna, S., and Tan, W.-C. 2002. On propagation of deletions and annotations through views. In *Proceedings of the PODS Conference.* 150–158.

Burks, C. 1999. Molecular biology database list. *Nucl. Acids Res. 27,* 1, 1–9.

Chawathe, S. S. and Garcia-Molina, H. 1997. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'97)*.

Chiang, F. and Miller, R. J. 2008. Discovering data quality rules. *Proc. VLDB Endow. 1,* 1, 1166–1177.

Chomicki, J. and Marcinkowski, J. 2005. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput. 197,* 1/2, 90–121.

Curwen, V., Eyras, E., Andrews, T. D., Clarke, L., and Mongin, E. e. a. 2004. The ensembl automatic gene annotation system. *Genome Res 14,* 5, 942–950.

Dadam, P., Lum, V. Y., and Werner, H. D. In *Proceedings of the 10th International Conference on Very Large Data Bases (VLDB'84)*.

Dennis, C. and Gallagher, R., Eds. 2002. *The Human Genome*. Palgrave, McMillan.

Dong, X. L., Berti-Equille, L., and Srivastava, D. 2009. Integrating conflicting data: The role of source dependence. *Proc. VLDB Endow. 2,* 1, 550–561.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. 2007. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Engin. 19,* 1, 1–16.

FAGIN, R., KOLAITIS, P. G., MILLER, R. J., AND POPA, L. 2005. Data exchange: Semantics and query answering. *Theor. Comput. Sci. 336,* 1, 89–124.

FAN, W., GEERTS, F., JIA, X., AND KEMENTSIETSIDIS, A. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Datab. Syst. 33,* 2.

FAN, W., GEERTS, F., LAKSHMANAN, L. V. S., AND XIONG, M. 2009. Discovering conditional functional dependencies. In *Proceedings of the 25th International Conference on Data Engineering (ICDE'09)*.

FAN, W., LU, H., MADNICK, S. E., AND CHEUNG, D. 2001. Discovering and reconciling value conflicts for numerical data integration. *Inf. Syst. 26,* 8, 635–656.

GALHARDAS, H., FLORESCU, D., SHASHA, D., SIMON, E., AND SAITA, C.-A. 2001. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*.

GALLAND, A., ABITEBOUL, S., MARIAN, A., AND SENELLART, P. 2010. Corroborating information from disagreeing views. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*.

GALPERIN, M. Y. AND COCHRANE, G. R. 2009. Nucleic acids research annual database issue and the NAR online molecular biology database collection in 2009. *Nucl. Acids Res. 37, suppl*$_1$, D1–4.

GOLAB, L., KARLOFF, H., KORN, F., SRIVASTAVA, D., AND YU, B. 2008. On generating near-optimal tableaux for conditional functional dependencies. *Proc. VLDB Endow. 1,* 1, 376–390.

HAMMING, R. 1950. Error detecting and error correcting codes. *Bell Syst. Techn. J. 26,* 2, 147–160.

HERNANDEZ, T. AND KAMBHAMPATI, S. 2004. Integration of biological sources: current systems and challenges ahead. *SIGMOD Rec. 33,* 3, 51–60.

HRYCEJ, T. AND HIPP, J. 2004. Outlier detection by rareness assumption. In *34. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*.

INTERNATIONAL HUMAN GENOME SEQUENCING CONSORTIUM. 2004. Finishing the euchromatic sequence of the human genome. *Nature 431,* 7011, 931–945.

LABIO, W. AND GARCIA-MOLINA, H. 1996. Efficient snapshot differential algorithms for data warehousing. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*.

LAND, A. H. AND DOIG, A. G. 1960. An automatic method of solving discrete programming problems. *Econometrica 28,* 3, 497–520.

LENZERINI, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'02)*.

LEVENSHTEIN, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10,* 8, 707–710.

LIU, B., HSU, W., AND MA, Y. 1998. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*.

LOUIE, B., MORK, P., MARTÍN-SÁNCHEZ, F., HALEVY, A. Y., AND TARCZY-HORNOCH, P. 2007. Data integration and genomic medicine. *J. Biomed. Inf. 40,* 1, 5–16.

MALETIC, J. I. AND MARCUS, A. 2000. Data cleansing: Beyond integrity analysis. In *Proceedings of the 5th Conference on Information Quality (IQ'00)*.

MICHAEL R. GAREY, D. S. J. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co.

MÜLLER, H., FREYTAG, J.-C., AND LESER, U. 2006. Describing differences between databases. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM'06)*.

MÜLLER, H., LESER, U., AND FREYTAG, J.-C. 2004. Mining for patterns in contradictory data. In *Proceedings of the International Workshop on Information Quality in Information Systems (IQIS'04)*.

MÜLLER, H., NAUMANN, F., AND FREYTAG, J.-C. 2003. Data quality in genome databases. In *Proceedings of the 8th International Conference on Information Quality (IQ 2003)*.

NAUMANN, F. AND HÄUSSLER, M. 2002. Declarative data merging with conflict resolution. In *Proceedings of the 7th International Conference on Information Quality (IQ 2002)*.

PASQUIER, N., BASTIDE, Y., TAOUIL, R., AND LAKHAL, L. 1999. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*.

RAHM, E. AND BERNSTEIN, P. A. 2001. A survey of approaches to automatic schema matching. *VLDB J. 10,* 4, 334–350.

RAHM, E. AND DO, H. H. 2000. Data cleaning: Problems and current approaches. *IEEE Data Engin. Bull. 23,* 4, 3–13.

RAMAN, V. AND HELLERSTEIN, J. M. 2001. Potter's wheel: An interactive data cleaning system. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*.

STEIN, L. D. 2003. Integrating biological databases. *Nat. Rev. Genet. 4,* 5, 337–345.

TRISSL, S., ROTHER, K., MÜLLER, H., STEINKE, T., AND ET AL., I. K. 2005. Columba: An integrated database of proteins, structures, and annotations. *BMC Bioinformatics 6*, 81.

VENTER, J. C., ADAMS, M. D., MYERS, E. W., LI, P. W., AND MURAL, R. J. E. A. 2001. The sequence of the human genome. *Science 291,* 5507, 1304–1351.

VOSSEN, G. 1991. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley Longman Publishing Co.

WEBB, G. I., BUTLER, S., AND NEWLANDS, D. 2003. On detecting differences between groups. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*.

WIJSEN, J. 2002. Condensed representation of database repairs for consistent query answering. In *Proceedings of the 9th International Conference on Database Theory (ICDT'03)*.

WINKLER, W. E. 1999. The state of record linkage and current research problems. Tech. rep. RR/1999/04, Statistics Research Division, U.S. Bureau of the Census.

XIONG, H., TAN, P.-N., AND KUMAR, V. 2003. Mining strong affinity association patterns in data sets with skewed support distribution. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*.

YIN, X., HAN, J., AND YU, P. S. 2007. Truth discovery with multiple conflicting information providers on the web. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*.

ZAKI, M. J. AND HSIAO, C.-J. 2002. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining*.

ZEHETNER, G. AND LEHRACH, H. 1994. The reference library system sharing biological material and experimental data. *Nature 367*, 489–491.

Received May 2011; revised November 2011; accepted December 2011