Fuzzy Duplicate Detection on XML Data

Melanie Weis

Humboldt-Universität zu Berlin
Unter den Linden 6, Berlin, Germany
mweis@informatik.hu-berlin.de

Abstract

XML is popular for data exchange and data publishing on the Web, but it comes with errors and inconsistencies inherent to real-world data. Hence, there is a need for XML data cleansing, which requires solutions for fuzzy duplicate detection in XML. The hierarchical and semi-structured nature of XML strongly differs from the flat and structured relational model, which has received the main attention in duplicate detection so far. We consider four major challenges of XML duplicate detection to develop effective, efficient, and scalable solutions to the problem.

1 Introduction

Data cleansing is an issue of critical practical importance. It is required to ensure high data quality in scenarios such as report generation over data warehouses and CRM. Another application is data integration, where data from distributed and heterogeneous data sources should be combined into a unique, complete, and correct representation for every real-world object. A crucial subtask in data cleansing is fuzzy duplicate detection (duplicate detection for short). It resolves which entries in a data source actually represent the same real-world object. Fuzzy duplicates are not trivial to detect, because several inconsistencies, such as spelling errors, missing information, and incomplete information, require more sophisticated techniques than comparisons based on equality.

Past research has mainly focused on duplicate detection on relational data. As XML has become a standard for data exchange and data publishing on the Web, concerns in XML data quality and efforts to integrate XML data are justified. Hence, algorithms for XML duplicate detection are required. We explore solutions to XML duplicate detection, considering the following aspects.

Real-world object vs. Object description. In relations, we can easily distinguish between real-world objects and *object descriptions (ODs)*. A relation represents objects whose description is provided by the attributes of the relation. For example, relation `BOOK(ISBN,TITLE,YEAR)` describes real-world `BOOK` objects by their `ISBN`, `TITLE`, and `YEAR`. In XML, all data is represented by XML elements, so there is no clear structural distinction between objects and their ODs. Consequently, the task of determining ODs, which is refereed to as *description selection*, is challenging in XML.

Structural Heterogeneity. The XML data model allows structural differences for XML elements that represent the same kind of real world object. We distinguish two kinds of structural differences, namely *schematic heterogeneity* and *instance heterogeneity*. Schematic heterogeneity occurs when two different XML elements represent the same real world object. For example, `<MOVIE>` and `<FILM>` both represent the same type of real-world object but are different elements in the schema. Instance heterogeneity signifies that two instances complying to the same schema can differ in their XML representation, e.g., in the number of occurrences of an element, or the order of elements.

Element Dependencies. Due to the hierarchical structure of XML, elements relate to their ancestors and descendants. These relationships can be considered to improve duplicate detection. For instance, two `<CITY>` elements with text `Los Angeles` are nested under different `<COUNTRY>` elements with text `USA` and `Chile`. Although the city names are identical, we can detect that they are not the same city in the real-world, because they are in different countries.

Before we define our four research challenges in Sec. 3, we briefly discuss related work in Sec. 2. In sections 4 to 7, we discuss solutions to each of the four challenges. Sec. 8 describes the data sets used to evaluate our approaches, and we conclude in Sec. 9.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

2 State of the Art

Research on fuzzy duplicate detection has mainly concentrated on efficiently and effectively finding duplicate records in relational data. Due to space limitations, we only highlight a few solutions in this section. We classify approaches into two areas: domain-dependent solutions, such as the ones proposed in [4, 10], assume a certain domain and the help of human experts to calibrate algorithms. Examples of domain-independent approaches are [7, 5]. All these approaches have in common that description selection and structural heterogeneity are not considered because the problems do not arise in relational duplicate detection. The work presented in [1] detects duplicates in hierarchically organized relations by considering data of tables related to the object table through foreign keys. In this context, instance heterogeneity has to be considered but description selection and schematic heterogeneity are still not an issue. There is work on identifying similar hierarchical data and XML data. However, most work does not consider the effectiveness of the similarity join. Rather, the authors concentrate on fast execution of the algorithm [3, 6]. In [3], the focus is on the efficient incorporation of tree edit distance in a framework performing approximate XML joins. The authors present upper and lower bounds for the tree edit distance, which are used as filters to avoid expensive tree edit distance computations. They further introduce a sampling method to effectively reduce the amount of data examined during the join operation. The only approach we are aware of that considers recall and precision of XML similarity joins is [2]. They present four different strategies to define the similarity function using the vector space model.

3 Research Tasks

The general problem of duplicate detection, be it in relational or XML data, can be divided into four subtasks, and in consequence into four research questions. In this section, we discuss why these challenges are particularly tackling in XML.

Challenge 1: Description Selection defines which information describes an XML element that is considered as an object. As we have seen in Section 1, this task is not trivial for XML data because we cannot distinguish schema elements representing objects from schema elements describing objects. We consider two ways of obtaining object description selections. The first is manual definition, the second is automation. We have followed both approaches and have developed both a GUI to specify object descriptions and heuristics that automatically come up with object descriptions of their own. While the first solution is merely an engineering feat, the second is an interesting research challenge that is addressed in Sec. 4.

Challenge 2: Duplicate Classifier. Generally, a duplicate classifier takes a pair of objects as input and classifies them as duplicates or non-duplicates, based on their ODs. Domain-dependent classifiers have the advantage of experts defining thresholds for similarity values and there are many commercial products making use of this. A more interesting challenge is to find a domain-independent classifier that performs well in many different scenarios. The problem becomes even more challenging when regarding XML data, because classifiers not only have to deal with data, but also with instance heterogeneity and schematic heterogeneity of objects. We discuss possible solutions in Sec. 5.

Challenge 3: Comparison Strategy. We need to perform duplicate detection efficiently. Efficient solutions for relational data take into account a single relation - for XML data things become more complex: A full duplicate detection run must detect duplicates at every level of the hierarchy. The research challenges we address to increase efficiency are strategies to traverse this hierarchy and methods of altogether avoiding comparisons while only slightly compromising the quality of the result, which is presented in Sec. 6.

Challenge 4: Scalability. Automatic duplicate detection is particularly relevant for large data sets. Here the research challenge is to minimize the number of runs across the data and thus the number of disk accesses. One question to be answered is to what degree relational and XML databases can support the duplicate detection process and to what degree calculations are performed by applications on top. Ideas are presented in Sec. 7.

Good solutions to these problems already exist for relational data. We address them in the XML scenario considering the additional problems described above.

4 XML Description Selection

Knowing ODs is necessary for duplicate detection, because they represent the information that is used to classify objects as duplicates or non-duplicates. Ideally, an OD includes information that characterizes a particular object, such as a book's ISBN, and does not vary depending on external influence (e.g., who wrote the book's review). In this section, we describe how ODs can be determined automatically in XML.

4.1 Using Schema Information

The concepts described in this subsection are presented in [9], so we only give a condensed description here and refer readers to [9] for details. We introduce two heuristics to select XML elements as part of an OD of an XML object, based on the XML schema the data complies to. The two heuristics are based on the observation that attributes describing an object are usually defined in proximity of the XML element that represents the object, called the *object element*. The two heuristics are illustrated in Fig. 1,

where we select description elements for the object element <MOVIE>. The first heuristic h_k selects the next k schema elements in breadth-first order (top figure, where $k = 5$). The second heuristic uses a radius r that selects all schema elements on the ancestor axis (h_{ra}) or the descendants axis (h_{rd}) that do not differ in depth to <MOVIE> by more than r.

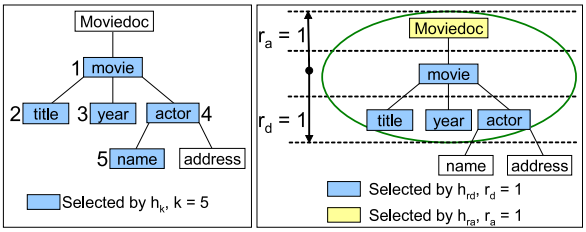

Figure 1: Two heuristics for description selection

We further introduce conditions that prune schema elements selected by the heuristic from the OD. The conditions are based on content model, data type, and cardinality of XML elements (minOccurs, maxOccurs) as defined by an XML Schema.

4.2 Using Instance Statistics

We consider using statistics on the actual XML data in addition to schema information. Statistics can be collected either on the structure of the instance XML document or on the content of XML elements. In the following, we denote by *od* an XML element included by a heuristic in the OD of an object element *o*. For instance, in the bottom figure of Fig. 1, *o* is the <MOVIE> element and *od* can be any of the dark shaded elements.

Statistics on XML Element Structure. Our first statistic captures how often an instance of *o* includes a particular number of instances of *od*. Examples are shown in Fig. 2. The left histogram shows the frequencies of the number of <TITLE> elements per <MOVIE>. 8,000 movies have exactly one title, opposed to 2,000 movies nesting two title elements. When a histogram shows that a majority of instances have one occurrence, although the element is declared with `maxOccurs=unbounded`, we declare it as *loose 1:1 relationship*. We prefer elements with loose 1:1 relationships over elements with 1:N relationships for ODs because this leverages the problem of instance heterogeneity for comparisons. The second histogram of Fig. 2 shows that most movies do not have a <YEAR> element at all, indicating a *tight optional relationship*. Such elements can be pruned from the OD selection because they will rarely provide useable information for comparisons.

Statistics on XML Element Content. We can collect statistics on the actual content to prune elements that are unlikely to be useful in later steps from an OD selection. More specifically, we collect the *element frequency* of values for every XML element *od*. The element frequency for the content of *od* is defined as the number of elements of type *o* in which *od* occurs with same content. For instance, in Fig. 3, the left histogram shows that the <YEAR> element has two distinct values that each occur in roughly 50% of all <MOVIE> elements. The good news is that the value is never null, so when the element exists it always has a value that we can use for comparisons. However, the value is common to so many movies that it cannot be considered as very descriptive of a particular movie. Therefore, it is not suited in an OD. On the other hand, <TITLE> has a wide range of values that all have low frequencies. Consequently, titles distinguish movies fairly well and are suited as OD.

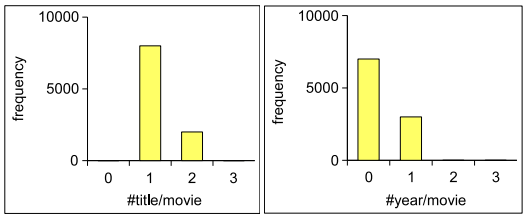

Figure 2: Statistics on Structure

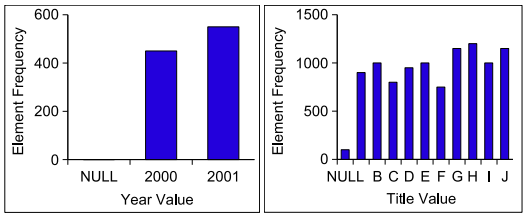

Figure 3: Statistics on Content

5 XML Duplicate Classifier

Once ODs are available, we need to define how objects are compared based on their ODs. Several duplicate classifiers that consider data exist for relational approaches. In XML, instance and schematic heterogeneity need to be considered as well. We distinguish three approaches for defining domain-independent XML classifiers that all use a similarity measure to classify pairs of objects as duplicates or non-duplicates. First, we investigate how classifiers solely based on data perform in XML. Second, we define classifiers that consider instance heterogeneity and data. The third kind of classifiers takes into account schematic heterogeneity as well.

Considering Data. In relations where only tuples are compared with each other, the structure is uniform across all tuples, and attributes are single valued and hence there is only one possible comparison for each attribute. In this scenario, which can be mapped to some XML scenarios such as XML data generated from relations, classifiers based on data similarity perform well. A commonly used similarity measure for

string data is the edit distance (the minimum number of insert, delete and change operations necessary to transform one string into another). When applied to tuples, a sample classifier compares attribute data pairwisely using edit distance, and if 75% of all attributes between two tuples are similar according to edit distance, they are considered duplicates. This easily maps to XML where attributes are XML elements that are part of ODs and attribute values are the corresponding text nodes.

Considering Instance Heterogeneity. Well-structured XML data is not the general case, so we have to deal with differences in the structure of two XML elements' ODs. In data integration, data coming from different data sources is usually mapped to a global schema. If the global schema is XML, the mapped XML objects have the same structure on schema level, but may differ in their instance structure. In this case, classifiers that consider data and instance heterogeneity are required. We propose such a classifier in [9]. It addresses the problem of multiple occurrences of a description element within an OD by computing the best similarity match between multiple occurrences in the two ODs. For example, let two `<COUNTRY>` elements c_1 and c_2 nest `<CITY>` elements. The three text nodes of `<CITY>` elements under c_1 contain `San Jose`, `Gilroy`, `San Francisco`, and c_2 nests cities `Gillroy`, and `San Francesco`. The best match is (`Gilroy`,`Gillroy`), (`San Francisco`, `San Francesco`) and the overall similarity is high because all cities from c_2 match cities from c_1. Determining the best match among all possible matches is not trivial. One possibility is to use stable marriage.

Considering Schematic Heterogeneity. The third and most general class of classifiers considers schematic heterogeneity in addition to data and instance heterogeneity. A similarity measure considering schematic heterogeneity can use tree edit distance to calculate the minimum cost of transforming the schema of one object into the schema of the other object. Data and instance heterogeneity are then computed on the transformed representation and are weighed according to the tree edit distance.

We consider using statistics to support or even automate the choice of a classifier for a given application. For instance, we can derive from statistics on the value distribution (see Fig. 3) which data type prevails. Depending on whether data is mainly text, date/time, or numerical, we choose an appropriate similarity measure. Considering instance heterogeneity is only necessary if the histograms of the occurrence frequency of elements (see Fig. 2) show a high variance in the distribution.

6 Comparison Strategy

A full duplicate detection run on XML data requires the detection of duplicates at every level of the hierarchy. We present three traversal strategies that we develop with efficiency in mind. The goal is to prune expensive computations (mainly pairwise comparisons) to reduce the overall duration of duplicate detection, either by *exact pruning* and *heuristical pruning*.

Exact pruning saves expensive operations and provably does not alter the final result of duplicate detection. An example for exact pruning is the filter f to the similarity measure sim that we define in [8]. Two objects are considered duplicates if sim is above a given threshold θ. The filter f is defined as an upper bound to sim. If $f < \theta$, it follows that $sim < \theta$, so sim is not computed because it will provably not find a duplicate. Of course, such a filter only makes sense when the complexity of the filtering step is below the complexity for computing the similarity of pruned pairs.

Heuristical pruning does not guarantee that saving comparisons does not alter the final result of duplicate detection. We consider three comparison strategies that classify as heuristical pruning, namely a bottom-up, a top-down, and a relationship-aware strategy.

Top-down strategy. Inspired by work presented in [1], we developed a bottom-up comparison strategy, published in [8]. We limit pairwise comparisons to XML elements that have the same or similar ancestors. This works well when nesting represents a 1:N relationship where there is exactly one possible parent for an element. By comparing elements in a breadth-first order from the root to the leaves, we detect duplicates in ancestors of every element first. This maintains good effectiveness because we limit comparisons to elements with equal ancestors, and at the same time, efficiency is greatly improved because we prune all pairwise comparisons of elements with different ancestors.

Bottom-up strategy. The drawback of the top-down approach is that its effectiveness degrades when the strict 1:N relationship does not hold, a common case in XML data. Furthermore, XML data is often stored in nested leaf nodes, and the data required for comparing ancestors is precisely the data in these leaf nodes. We consider a traversal strategy that first compares all leaf nodes and then only compares ancestors that have at least one child in common. This way, we save comparisons on ancestors and effectiveness remains good even if strict 1:N relationships do not hold.

Relationship-aware strategy. Both previous approaches consider relationships between nested elements in one direction only. However, it is possible that elements influence each other in both directions. For example, movies with same actors are likely to be duplicates, and actors playing in the same movies are also likely to be duplicates. So detecting duplicates in movies influences duplicate detection in actor, and vice versa. We cannot afford recomputing similarities and perform duplicate detection for both movies and actors alternately until the result converges, which is in principle possible, although this algorithm may be best in

terms of effectiveness. A compromise is to develop an algorithm that, based on the influence that elements have on each other determines an order of comparisons where each pairwise similarity is computed at most once, while maximizing the benefit of element dependency.

7 Scalability

In order to efficiently detect duplicates on large data sets, we have to consider scalability. One question that we address is to what degree relational and XML databases can support the duplicate detection process and to what degree calculations are performed by applications on top. Databases can be used as storage for information necessary for comparisons. Currently, we use a relational database to store ODs. We need to define a retrieval strategy for objects that minimizes database accesses when we perform pruning and pairwise comparisons in main-memory. Today's databases can perform much more than storing and retrieving large amounts of data. A database can support automatic description selection by gathering the necessary statistics. Collecting statistics is commonly used for query optimization, so we will explore which statistics are collected for XQuery optimization and whether we can use them. Pruning of comparisons as well as pairwise comparisons can be performed by a database, e.g., by defining filters and similarity measures as user defined functions (UDFs) in relational databases. An interesting issue is how to optimize the underlying calculations.

8 Experiments

We need to evaluate the proposed measures, heuristics and methods against large, realistic, and interesting data sets. We conduct experiments on synthetic as well as real world data. We use synthetic data to control several parameters, such as types and amount of errors, and probability of duplicates. We generate duplicates from clean data using a tool developed in our group[1]. Our real world scenarios currently include large real-world data sets from the CD domain[2] and movie domain[3] ,[4]. We plan to classify duplicates by hand on the FILMDIENST movie dataset, containing data about roughly 56000 movies and 180000 actors, using a tool that supports manual duplicate detection and which is currently under development.

9 Conclusion

Our research in XML duplicate detection addresses four major challenges. First, we investigate on how object descriptions can be selected automatically, a difficult task in XML where objects and object descriptions are both represented by XML elements. Second, we define new domain-independent duplicate classifiers that take into account not only data, but also structural diversity of XML objects. Third, we define comparison strategies that make use of element dependencies to improve efficiency without jeopardizing effectiveness. Finally, we consider scalability by investigating how relational and XML databases can support the duplicate detection process. By considering the problem of XML duplicate detection under the aspects of effectiveness, efficiency and scalability, we believe that our insights and solutions will significantly contribute to solving XML duplicate detection for a wide range of applications.

Acknowledgments. This research is supported by the German Research Society (DFG grant no. NA 432). Thanks to my thesis advisor Felix Naumann for helpful discussions.

[1] http://www.informatik.hu-berlin.de/mac/dirtyxml/
[2] FREEDB: http://www.freedb.org/
[3] IMDB: http://www.imdb.com/
[4] FILMDIENST: http://film-dienst.kim-info.de/

References

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. of VLDB*, Hong Kong, China, 2002.

[2] J. C. Carvalho and A. S. da Silva. Finding similar identities among objects from multiple web sources. In *CIKM-2003 Workshop on Web Information and Data Management*, pages 90–93, New Orleans, LA, 2003.

[3] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *Proc. of SIGMOD*, pages 287–298, Madison, WI, 2002.

[4] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of SIGMOD*, pages 127–138, San Jose, CA, 1995.

[5] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proc. of DASFAA*, Kyoto, Japan, 2003.

[6] K. Kailing, H.-P. Kriegel, S. Schnauer, and T. Seidel. Efficient similarity search for hierarchical data in large databases. In *Proc. of EDBT*, pages 676–693, Heraclion, Crete, 2004.

[7] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD-1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, 1997.

[8] M. Weis and F. Naumann. Duplicate detection in XML. In *SIGMOD-2004 Workshop on Information Quality in Information Systems*, pages 10–19, Paris, France, 2004.

[9] M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML (to appear). In *Proc. of SIGMOD*, Baltimore, MD, 2005.

[10] W. E. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC, 1994.