Adaptive Duplicate Detection in XML Document Based on Hash Function

Thandar Lwin, Thi Thi Soe Nyunt
University of Computer Studies, Yangon, Myanmar
thandarlwin72@gmail.com, ttsoenyunt@gmail.com

Abstract

The task of detecting duplicate records that represents the same real world object in multiple data sources, commonly known as duplicate detection and it is relevant in data cleaning and data integration applications. Numerous approaches both for duplicate detection in relational and XML data exist. As XML becomes increasingly popular for data representation, algorithms to detect duplicates in XML documents are required. Previous domain independent solutions to this problem relied on standard textual similarity functions (e.g., edit distance, cosine metric) between objects. However, such approaches result in large numbers of false positives if we want to identify domain-specific abbreviations and conventions.

In this paper, we present a generalized framework for duplicate detection, specialized to XML. The aim of this research is to develop an efficient algorithm for detecting duplicate in complex XML documents and to reduce number of false positive by using hash function algorithm.

1. Introduction

Duplicate detection, which is an important subtask of data cleaning, is the process of detecting duplicate records corresponding to the same real word object from one or more data sources. It is of practical relevance in important and business-relevant applications, such as data warehousing, data mining, or data integration. The problem was original defined in 1995 by Newcombe et al. [1] and was first formalized by Felligi and Sunter [2] ten years later. The problem was originally called record linkage [11], but ironically, it has appeared under various different names since then, including (in chronological order) entity identification [3],

deduplication [4], duplicate detection [5], entity resolution [6], fuzzy duplicate identification [7], object consolidation [8], reference reconciliation [9], or object identification [10].

The challenge in duplicate detection is to detect duplicate representations that are *not exactly equal* due to errors in the data, and that *cannot be identified using a universal identifier* (e.g., the ISBN of a book). Examples for errors are typos and misspellings or the lack of a standard representation for the data, for instance when a date can be represented both in the European format (day.month.year) or the American format (month/day/year). Further errors are missing, outdated, or contradictory data. As ac consequence, duplicate detection cannot be performed just by checking on the equality of object attributes or global identifiers. Instead, more complex algorithms are required, e.g., objects need to be compared pairwisely using a complex similarity measure. Such algorithms are necessary in both data cleaning and data integration scenarios.

Until recently, research on duplicate detection has focused on detecting duplicates within a *single relational table*. The schema of a table consists of several attributes. Every tuple in a relational table has exactly one value for every attribute. Most duplicate detection approaches designed for a single relation iteratively compare pairs of tuples as follows: They first compare attribute values pairwisely by computing a value similarity, and then combine these similarities to a total tuple similarity. If the similarity is above a specified threshold, tuple pairs represent duplicates, otherwise they represent non-duplicates. We call this comparison approach a *thresholded similarity measure* approach, which is a popular approach for *iterative duplicate detection in relational data*. A good similarity measure is very important to find the correct duplicates, i.e., for high *effectiveness*. Another aspect that has been considered is the time complexity of an approach, i.e., *efficiency*, which is mainly improved by sophisticated tuple pruning techniques that avoid expensive similarity comparisons. To perform duplicate detection on large relational data not fitting in main memory, *scalability* has also been considered. To this end, relational database management systems are commonly used to store, access, and update data efficiently.

The objectives of our proposed system are to detect duplicated XML documents relating to the same entities, to improve data quality and integrity, to allow re-use of exiting data sources for new studies and to reduce costs and effort in data acquisition for research studies. More specifically, our approach detects duplicate objects in a single XML document, assuming the structure of elements with equal name may differ. Without further domain-specific information, we are able to detect duplicate objects with typographical errors (data differs only slightly), equivalence errors (data differs significantly but has same meaning), and missing data.

The outline of this paper is as follows. In Section 2, we present related work. In Section 3, we discuss the main concept of XML and MD5 algorithm on which our approach is based. Section 4 provides a detailed description of our approach. In Section 5, we present experimental study and we conclude in Section 6.

2. Related work

The problem of duplicate detection, originally defined by Newcombe [9] and formalized in the Fellegi-Sunter [10] model for record linkage has received much attention in the relational world and has concentrated on efficiently and effectively finding duplicate records.

Some approaches are specifically geared towards a particular domain, including census, medical, and genealogical data [5 11, 12], and require the help of a human expert for calibration [13]. Other algorithms are domain-independent, e.g., those presented in [15, 22]. 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.

Recent projects consider detecting duplicates in hierarchical and XML data. This includes DELPHI [5], which identifies duplicates in hierarchically organized tables of a data warehouse using a top-down approach along a single data warehouse dimension. The algorithm is efficient because it compares only children with same or similar ancestors. 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. To minimize the number of pairwise element comparisons, an appropriate filter function is used [23], where three filtering methods is used.

Work presented in [15] describes efficient identification of similar hierarchical data, but it does not describe how effective the approaches are for XML duplicate detection. For example, Dong et al. present duplicate detection for complex data in the context of personal information management [7], where different kinds of entities such as conferences, authors, and publications are related to each other, giving a graph like structure. The algorithm propagates similarities of entity pairs through the graph. Any similarity score above a given threshold can trigger a new propagation of similarities, meaning that similarities for pairs of entities may be computed more than once. Although this improves effectiveness, efficiency is compromised. The domain-independent DogmatiX algorithm [8], which considers both effectiveness by defining a suited domain-independent similarity measure using information in ancestors and descendants of an XML element, and efficiency by defining a filter to prune comparisons. However, in the worst case, all pairs of elements need to be compared, unlike the sorted neighborhood method that has a lower complexity. The word presented in [22] which uses the sorted neighborhood method, which trades off effectiveness for higher efficiency by comparing only objects within a certain window.

However, most work based on similarity matrix to classify pairs of objects as duplicates or non-duplicates using an appropriate threshold value which may cause false positives.

3. XML and MD5

3.1. XML

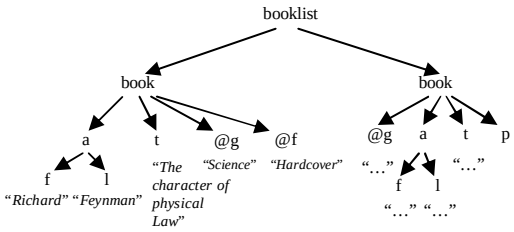
The Extensible Markup Language (XML) has been proposed by the World Wide Web Consortium (W3C) in 1999, and its definition has evolved ever since. The resulting W3C Recommendation is provided in [19]. XML is used both for largescale electronic publishing of data, and for the exchange of data on the Web and elsewhere. The two main features of XML are that the data is organized hierarchically, and is semi-structured, mixing content, e.g., text and structure, using so called XML tags. A file conforming to the XML format is called an XML document.

An XML document includes (but is not limited to) a set of nodes in the document (a root node, element nodes, attribute nodes, and value nodes) all having an identity, as well as a set of edges between nodes such that a tree is obtained. The root node has no ancestors, an has a set of children elements that can be element nodes, attribute nodes, and value nodes. Element nodes are delimited by an opening and a closing tag (<.> and </.>). The content of the node (between the opening and closing tag) is either (i) complex if an XML node only has element nodes as children (ii) simple if it only has value nodes as children, or (iii) mixed if it has both element and

value nodes as children. Attribute nodes can only nest a single value node, hence they always have simple content type. Value nodes have types such as string, date, double, and integer and represent the leafs of an XML tree. Fig. 1(a) shows an excerpt of a sample XML document. Fig. 1(b) shows the tree representation of the complete XML document, together with node ids, which we introduce to reference a particular node. Our sample XML document consists of root node r, e.g., the <booklist> element. The set of element nodes is the union of the set of <book> elements, the set of <author> elements, the set of <firstname> elements, the set of <lastname>, as well as sets <title>, and <published> respectively. The set of attribute nodes consists of the *genre* attributes and the set of value nodes consists of all values represented in italic in the tree representation and connected to nodes by dashed lines. The set of edges between nodes is the union of edges depicted as straight lines and edges depicted as dashed lines.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<booklist>
<book genre="Science" format="Hardcover">
        <author>
        <firstname>Richard</firstname>
        <lastname>Feynman</lastname>
        </author>
        <title>The character of Physical Law</title>
</book>
<book genre="Fiction">
        <author>
        <firstname>R.K.</firstname>
        <lastname>Narayan</lastname>
        </author>
        <title>Waiting for the Mahatma</title>
        <published>1981</published>
</book>
</booklist>
```

(a) Sample XML document



(b) Tree representation of sample XML data

Figure 1. Different representations of sample XML data

3.2. MD5

MD5 stands for Message Digest 5, which is developed by Ron Rivest of the MIT Laboratory for Computer Science and RSA Data Security, Inc. MD5 is an algorithm which takes an input of any length and outputs a message digest of a fixed length. The generation of a digest is very fast and the digest itself is very small and can easily be encrypted and transmitted over the Internet. It is very easy and fast (and therefore cheap) to check some data for validity. The algorithms are well known and implemented in most major programming languages, so they can be used in almost all environments. MD5 uses the same algorithm every time. Hence it will always generate the same message digest for the string (data).

3.2.1 Message Padding

The message to be fed into the message digest computation must be a multiple of 512 bits (sixteen 32-bit words). The original message is padded by adding a 1 bit, followed by enough 0 bits to leave the message 64 bits less than a multiple of 512 bits.

3.2.2 Overview of MD5 Computation

In MD5 the message is processed in 512-bit blocks (sixteen 32-bit words). (See Figure2.) The message digest is a 128-bit quantity (four 32-bit words). Each stage consists of computing a function based on the 512-bit message chunk and the message digest to produce a new intermediate value for the message digest. The value of the message digest is the result of the output of the final block of the message.

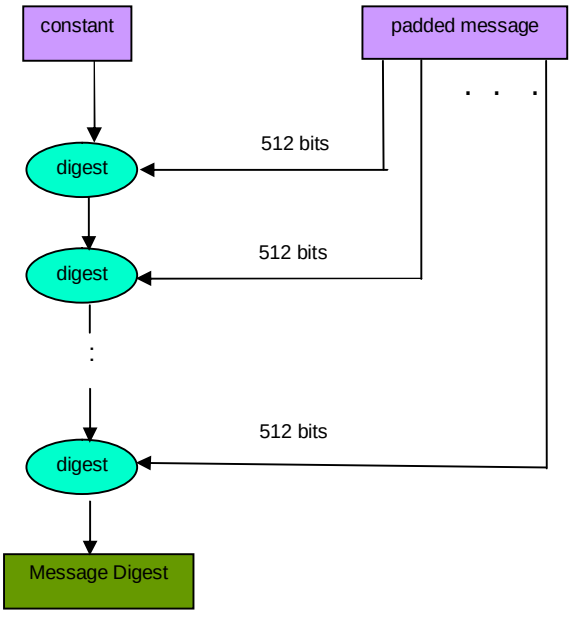


Figure 2. Overview of MD5.

3.2.3 MD5 Algorithm

MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. *F* is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit little endian integers); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message, in bits.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *A*, *B*, *C* and *D*. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function *F*, modular addition, and left rotation.

3.2.4 MD5 hashes

The 128-bit (16-byte) MD5 hashes (also termed *message digests*) are typically represented as a sequence of 32 hexadecimal digits. The following demonstrates a 43-byte ASCII input and the corresponding MD5 hash:

MD5("The quick brown fox jumps over the lazy dog") = 9e107d9d372bb6826bd81d3542a419d6

Even a small change in the message will (with overwhelming probability) result in a completely different hash, due to the avalanche effect. For example, adding a period to the end of the sentence:

MD5("The quick brown fox jumps over the lazy dog.")
= e4d909c290d0fb1ca068ffaddf22cbd0

The hash of the zero-length string is:

MD5("")= d41d8cd98f00b204e9800998ecf8427e

4. Duplicate Detection

The duplicate detection algorithm is introduced in this section. Section A describes the overall architecture which consists of several phases, which are discussed in detail in the Sections B through C.

4.1Outline of the Duplicate Detection

The architecture of duplicate detection is show in Figure 4. The system begins with the XML data and candidate definitions as input. The Selector module selects candidate objects from the XML documents using candidate definition. The candidate objects are then preprocessed to get standardized objects. The outputs of Preprocessor are stored in a relational database to identify duplicate objects. The Duplicate identifier uses the output of Preprocessor to compute their corresponding hash codes by using MD5 algorithm. Finally, the duplicate objects can be identified according to their same hash values.

4.2 Candidate Definition

Defining which object data values actually describe an object, i.e., which values to consider when comparing two objects. It is based on three observations.

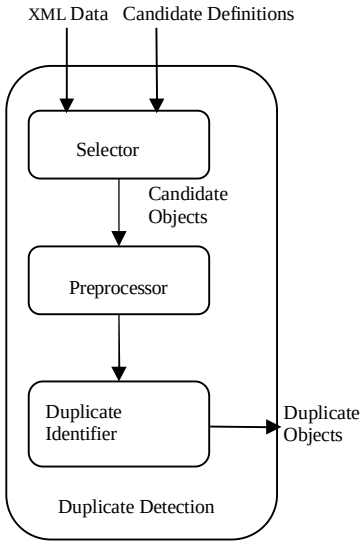


Figure 4. Duplicate Detection Architecture

(i) A data source may store information about various types of real-world objects, not all of which need to be considered for duplicate detection.

(ii) Among the elements relevant to object identification, some may represent the same type of real-world object, just represented differently. These should be compared with each other.

(iii) On the other hand, it makes no sense to compare objects of different real-world type, as they cannot be duplicates of each other.

4.3 Data Preprocessing

The data preprocessing module includes a parsing, a data transformation step and a standardization step all of which are performed first

for improving the quality of in-flow data and second for making the data comparable and more usable.

Parsing is the first critical component in the data preparation stage of record matching. This process locates, identifies and isolates individual data elements in the source files. Parsing makes it easier to correct, standardize, and match data because it allows comparing individual components, rather than long strings of data. For example, the appropriate parsing of name and address components is a crucial part in the duplicate detecting process.

Data transformation refers to simple conversions that can be applied to the data in order for them to conform to the types of their corresponding domains. The most common form of a simple transformation is the conversion of a data element from one data type to another.

Data standardization refers to the process of standardization the information represented in certain fields with some special content. This is used for information that can be stored in many different ways in different data sources and must be converted to a uniform presentation before the record matching process starts. One of the most common formatting needs is for address information. Without standardization, many true matches could erroneously be designated as non-matches, based on the fact that common identifying information can not be compared.

4.4 Duplicate Detection

Duplicate detection is achieved by calculating a unique hash value for each document. Each document is then examined for duplication by looking up the value (hash) in either an in-memory hash or persistent lookup system. Several common hash functions used are MD2, MD5, or SHA (1995). These functions or types of functions are used because they have three desirable properties, can be calculated on arbitrary data / document lengths, easy to compute, and have very low probabilities of collisions. In this paper, we use MD5 hash function for duplicate detection in XML document.

4.4.1 Similarity Metrics

While it is unclear at which point a document is no longer a duplicate of another, researchers have examined several metrics for determining the similarity of a document to another. The first is *resemblance* (Broder, Glassman et al. 1997), where the general notion is that if a document contains roughly the same semantic content it is a duplicate whether or not it is a precise syntactic match.

$$resembalance(D_j,D_k)=\frac{|S(D_j)\cap S(D_k)|}{|S(D_j)\cup S(D_k)|} \quad \text{----(1)}$$

Where the resemblance of two documents is the intersection of features over the union of features from two documents. This metric can be used to calculate a fuzzy similarity between two documents. For example, D_1 and D_2 have 50% of their terms the same and each document has 10 terms. Their resemblance would be .33 (5/15). By extending what features are used as the unit of comparison, terms, sentences, or any other document information can be used for comparison.

Many researchers have looked at using resemblance given some threshold *t* to find duplicate documents (Brin, Davis et al. 1995; Shivakumar and Garcia-Molina 1995; Garcia-Molina, Gravano et al. 1996; Shivakumar and Garcia-Molina 1996; Broder, Glassman et al. 1997; Shivakumar and Garcia-Molina 1998; Fetterly, Manasse et al. 2003), where if *t* was exceeded the documents would be considered duplicate. Two general issues have been explored when using resemblance approaches. The first is what features and threshold *t* should be used when calculating resemblance of documents. The second is efficiency issues that come into play with large collections and the optimizations that can be applied (Broder 1998).

Next we examine the efficiency issues that are involved in calculating similarity of a document to a collection. We partition the work into two categories: shingling techniques, similarity measures calculations. Shingling techniques, such as COPS (Brin, Davis et al. 1995), KOALA (Heintze 1996), and DSC (Broder 1998), take a set of contiguous terms or *shingles* of documents and compare the number of matching shingles, or others (Finkel, Zaslavsky et al. 2001). The comparison of document subsets allows the algorithms to calculate a percentage of overlap between two documents using resemblance, Equation (1). This type of approach relies on hash values for each document subsection/feature set and filters those hash values to reduce the number of comparisons the algorithm must perform. This filtration of features, therefore, improves the runtime performance. Note that the simplest filter is strictly a syntactic filter based on simple syntactic rules, and the trivial subset is the entire collection. We illustrate later why such a naive approach is not generally acceptable. In the shingling approaches, rather than comparing elements, subelements are compared, and thus, each element may produce many potential duplicates.

5. Experimental Study

In the current state of our work, we have implemented a prototype to evaluate our proposal for duplicate detection in complex XML data. We used a Pentium(R) 4 computer with 3.20 GHz processor and 1GB main memory for the experiments. The experiments were tested using Java 1.6 software development kit. We performed some experiments with this prototype, by using a real world bibliography datasets such as DBLP, and SIGMOD.

Then, we compared the elements from these document sets. We hope that the experiments produced good results, showing that our approach is capable of dealing with divergences in the documents' contents and also in their structures.

6. Conclusion

In this paper, we presented an approach to detect duplicate objects in an XML document, which is an important data cleansing task necessary to improve data quality. Our approach overcomes the problems of element scope and structural diversity within an XML document, and efficiently identifies duplicate elements by using MD5 algorithm. We tend to address the optimality of our algorithm by comparing its effectiveness to other similarity measures when applied to XML and explore the automation of the selection of candidates, so that no domain-knowledge whatsoever is required.

References

[1] Newcombe, H.; Kennedy, J. ; Axford, S. ; James, A.: "Automatic linkage of vital records." In: *Science 130* , Nr. 3381.

[2] FELLEGI, I. P. ; SUNTER, A. B.: A theory for record linkage. In: *Journal ofthe American Statistical Association* (1969)

[3] LIM, Ee-Peng ; SRIVASTAVA, Jaideep ; PRABHAKAR, Satya ; RICHARDSON,James: Entity Identification in Database Integration. In: *InternationalConference on Data Engineering (ICDE)*. Vienna, Austria, April 1993, S.294–301

[4] SARAWAGI, Sunita ; BHAMIDIPATY, Anuradha: Interactive Deduplicationusing Active Learning. In: *Conference on Knowledge Discovery and Data Mining (KDD)*. Edmonton, Alberta, 2002, S. 269–278

[5] BILENKO, Mikhail ; MOONEY, Raymond J.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: *Conference on Knowledge Discovery and Data Mining (KDD)*. Washington, DC, 2003, S. 39–48

[6] BHATTACHARYA, Indrajit ; GETOOR, Lise: Relational Clustering forMultitype Entity Resolution. In: *Workshop on Multi-Relational Data Mining (MRDM)* (2005)

[7] CHAUDHURI, Surajit ; GANTI, Venkatesh ; MOTWANI, Rajeev: Robust Identification of Fuzzy Duplicates. In: *International Conference on Data Engineering (ICDE)*. Tokyo, Japan, 2005, S. 865–876

[8] CHEN, Zhaoqi ; KALASHNIKOV, Dmitri V. ; MEHROTRA, Sharad: Exploiting Relationships for Object Consolidation. In: *SIGMOD Workshop on Information Quality in Information Systems (IQIS)*. Baltimore, MD, 2005

[9] DONG, Xin ; HALEVY, Alon ; MADHAVAN, Jayant: Reference Reconciliation
in Complex Information Spaces. In: *Conference on the Management of Data (SIGMOD)*. Baltimore, MD, 2005, S. 85–96

[10] SINGLA, Parag ; DOMINGOS, Pedro: Object Identification with Attribute-Mediated Dependences. In: *Conference on Principals and Practice of Knowledge Discovery in Databases (PKDD)*. Porto, Portugal, 2005, S. 297– 308

[11] W. E. Winkler. The state of record linkage and current research problems. Technical report, U. S. Bureau of the Census, 1999

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

[13]Hern´andez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: SIGMOD Conference, San Jose, CA (1995) 127–138

[14]Lim, E.P., Srivastava, J., Prabhakar, S., Richardson, J.: Entity identification in database integration. In: ICDE Conference, Vienna, Austria (1993) 294–301

[15]Doan, A., Lu, Y., Lee, Y., Han, J.: Object matching for information integration: A profiler-based approach. IEEE Intelligent Systems, pages 54-59 (2003)

[16]Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: International Conference on VLDB, Hong Kong, China (2002)

[17]Weis, M., Naumann, F.: DogmatiX Tracks down Duplicates in XML. In: SIGMOD Conference, Baltimore, MD (2005)

[18]Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association (1969)

Jaro, M.A.: Probabilistic linkage of large public health data files. Statistics in Medicine **14** (1995) 491–498

[18]Hern´andez, M.A., Stolfo, S.J.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery **2(1)** (1998) 9–37

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

[20]Jin, L., Li, C., and Mehrotra, S.: Efficient record linkage in large data sets. In Proc. Of DASFAA, Kyoto, Japan(2003).

[21]Puhlmanu, S., Weis, M., and Naumann, F.: XML Duplicate Detection Using Sorted Neighborhoods. International conference on Extending Database Technology (EDBT), (2006)

[22]Elmagarmid, A., K.: Duplicate Record Detection: A Survey. IEEE Transactions on Knowledge and Data Engineering,Vol.19, No.1, January(2007)