Detecting Duplicates in Complex XML Data

Melanie Weis, Felix Naumann
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin
{mweis,naumann}@informatik.hu-berlin.de

Abstract

Recent work both in the relational and the XML world have shown that the efficacy and efficiency of duplicate detection is enhanced by regarding relationships between entities. However, most approaches for XML data rely on 1:n parent/child relationships, and do not apply to XML data that represents m:n relationships.

We present a novel comparison strategy, which performs duplicate detection effectively for all kinds of parent/child relationships, given dependencies between different XML elements. Due to cyclic dependencies, it is possible that a pairwise classification is performed more than once, which compromises efficiency. We propose an order that reduces the number of such reclassifications and apply it to two algorithms. The first algorithm performs reclassifications, and efficiency is increased by using the order reducing the number of reclassifications. The second algorithm does not perform a comparison more than once, and the order is used to miss few reclassifications and hence few potential duplicates.

1. State of the Art

Duplicate detection is the problem of identifying multiple representations of a same real-world object. It is a crucial task in data cleansing and has applications in scenarios such as data integration, customer relationship management, and personal information management. With the popularity of XML, there is a growing need for duplicate detection algorithms specifically geared towards the XML data model. Indeed, most algorithms developed for relational data, such as those presented in [4, 5, 6, **?**, 7] apply to a single relation with sufficient attributes. However, in the case of XML data, we observe that XML elements representing objects have few attributes and instead have related XML elements describing them. We call XML data complex, because we have to consider a schema with complex XML elements that is more complex than a single relation for duplicate detection.

Recently, algorithms that exploit relationships between entities were developed. In [1], the authors consider parent/child relationships in relations of a data warehouse hierarchy in a top-down traversal of the hierarchy and thereby increase efficiency and effectiveness. We proposed a similar approach for XML data in [9]. However, as shown in [8], the top-down approach, as well as a bottom-up approach we envisioned, rely on the fact that parent and child elements are in a 1:n relationship, meaning that a parent can have several different children but a child is associated to a unique parent. This is for example the case for `<movie>` elements nesting `<title>` elements, because a single title can only belong to a single movie, but a movie can have alternative titles. However, the assumption is not valid for movies nesting actors, because an actor can star in different movies. In [2], the approach views data sets as attributed relational graphs of object representations (nodes), connected via relationships (edges). The approach then applies graph partitioning techniques to cluster object representations. Our classification technique relies on pairwise comparisons and classifications. Another publication on this topic is [3], where duplicate detection is performed for a personal information management (PIM) application. Dong et al.[3] propose an algorithm that propagates similarities from one duplicate classification to another. Our basic idea of using references resembles the idea proposed in [3], but we consider different aspects of the problem. We are interested both in efficiency and effectiveness, whereas for low-volume PIM data efficiency is not the crucial issue. Furthermore, we set an additional focus on a comparison order.

Here, we introduce a novel comparison strategy that applies on all kinds of parent/child relationships, not only 1:n. We show that the comparison order influences our strategy and present a suited order in Sec. 3. In Sec. 4, we apply the comparison strategy and the order to two algorithms. The first algorithm uses the order to reduce the number of possible reclassifications whereas the second one uses the order to reduce the number of missed comparisons and hence potentially missed duplicates. Experiments support our belief that the defined order achieves these goals better than other orders.

2. Classification Strategies

Before demonstrating our comparison strategy by example, we need to clarify terms. An XML element can play the role of being the XML element representing an object to which we detect duplicates, i.e., a *duplicate candidate*, or it can be an element describing a candidate, which we call an *object description (OD)*. Consequently, duplicate detection of one candidate is not independent of the duplicate detection of another candidate, because candidates are compared based on similarities in their ODs. The comparison strategy we present is based on candidate and OD definitions provided by an expert. We say that a candidate c depends on another candidate c' if c' is part of the OD of c. As we will see, due to cyclic dependencies between candidates it is useful to classify pairs of objects more than once.

To demonstrate our comparison strategy, we consider the XML elements of Fig. 1. The candidates and OD definitions, which are necessary input to the algorithm, are provided in Tab. 1. For instance, the candidate `<movie>` is described by its `<title>` and `<actor>` children. The `<title>` candidate depends on its text node and on the `<movie>` element it is child of. We observe a cyclic dependency between `<movie>` and `<title>` candidates.

```
<movie>
<title> Troy </title>
  <actors>
    <actor> Brad Pitt </actor>
    <actor> Eric Bana </actor>

  </actors>
</movie>
```

```
<movie>
  <title> Troja </title>
  <actors>
    <actor> Brad Pit </actor>
    <actor> Erik Bana</actor>
    <actor> Brian Cox</actor>
  </actors>
</movie>
```

```
<movie>
  <title> The Illiad Project </title>
  <actors>
    <actor> Prad Pitt </actor>

    <actor> Brian Cox </actor>
  </actors>
</movie>
```

Figure 1. Sample XML elements

Candidate	OD
`movie`	`title`, `actor`
`title`	`movie`, *textnode*
`actor`	*textnode*

Table 1. Sample OD definition

For ease of presentation, we identify XML element as follows. The three `<movie>` elements are duplicates, which we denote $m1$, $m1'$, and $m1''$. Their `<title>` elements are not duplicates, so we denoted them as $t1$, $t2$, and $t3$, respectively. Brad Pitt and its obvious duplicates are denoted $a1$, $a1'$, and $a1''$. Similarly, Eric Bana is $a2$ when nested under $m1$ and $a2'$ when nested under $m1'$. Brian Cox and its duplicate are denoted $a3$ and $a3'$. For pairwise classification, we arbitrarily decide to consider pairs in the order $\{(m1, m1'), (m1, m1''), (m1', m1''), (t1, t2), (t1, t3),$ $(t2, t3), ((a1, a2), ...\}$. When comparing $m1$ and $m1'$, they appear to have no related object in common because actors and titles have not yet been compared. We conclude for now that they are not duplicates. The same is true for all other comparisons between movies and between titles. Continuing along the list we start to compare actors and find duplicates $(a1, a1')$, $(a1, a1'')$, $(a1', a1'')$, $(a2, a2')$, and $(a3, a3')$. Knowing that movies depend on their actors, we compare movies again, with the additional knowledge of duplicates among actors.We find that they are duplicates because they now share several actors. Titles being related to movies, we compare titles again, but do not find further duplicates. The point is that by reclassifying movies after duplicates in related objects have been detected, we were able to find duplicates where we could not before. Consequently, reclassifying pairs can increase effectiveness. However, classifications being an expensive operation, we should avoid to perform a classification too often. In the above example, it is easy to see that if we had started by comparing actors, we would have saved reclassifying movies and titles a second time. Next, we introduce an order that reduces this number of reclassifications.

3. Comparison Order

For every candidate pair (v, v'), we compute a rank $r(v, v')$ whose ascending order is used as comparison order. Rank $r(v, v')$ estimates for every pair of candidates v and v' the *number of reclassifications* necessary if the similarity was calculated at the current processing state. A low rank implies few reclassifications, so the pair with low rank is classified early. The estimation of r takes into account both an estimate of how often a pair (v, v') is reclassified, and an estimate of how many classifications of other pairs are triggered by pair (v, v') if v and v' are classified as duplicates. The value of $r(v, v')$ depends on duplicates detected among elements composing v and v''s ODs. Consequently, $r(v, v')$ needs to be recomputed whenever a duplicate is detected among their ODs. These computations can be saved using $r - static(v, v')$, a version of r that does not take into account duplicates in ODs.

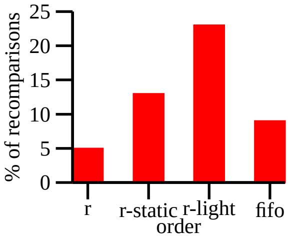
4. Application and Evaluation

The comparison order as defined by rank r finds application in two algorithms. The first algorithm, which we call the *reconsidering algorithm (ReconA)*, is the algorithm illustrated by the example in Sec. 2. It reclassifies pairs of candidates when there is a chance that their similarity has increased due to a duplicate classification of an element in their ODs. The order helps to reduce the number of reclassifications performed by ReconA.

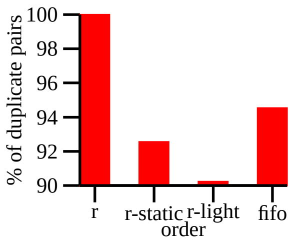
The second algorithm is called *adamant algorithm (AdamA)* and will never perform a classification more than once. In this case, it is crucial to perform a classification when its chance of finding a duplicate (if there is one) is high. This case is achieved when the number of reclassifications associated with this pair is low. This situation corresponds to a low value of r for that candidate pair. Consequently, the ascending order of r is used by AdamA to reduce the number of missed reclassifications and hence the number of missed duplicates.

Preliminary experiments show that the order defined by r achieves its goal for both algorithms, compared to other orders. We show results for *r* and *r-static*, as well as two other orders named *r-light* and *fifo*. *r-light* is a simplified version of *r* that does not consider duplicates in the two candidates' ODs and is only based on the number of comparisons the candidate pair may trigger. *fifo* considers candidates in the order they appear in the XML data.

The experiments are based on a data set obtained by integrating data from two movie data sources [1]. Duplicates are due to two representations of a same movie, one from each source. The data set consists of 2140 candidates and 8796 dependencies. We have mutual dependencies between actors and movies, as well as between movies and titles. Note that these dependencies can be exploited in both directions because key and keyref were used.



(a) Results for ReconA



(b) Results for AdamA

Figure 2. Impact of order

For ReconA, we measure the percentage of reclassifications performed using different orders. The result shown in Fig. 2(a) clearly shows that r results in the least reclassifications. Its static version *r-static*, which does not reestimate r during the comparison phase performs worse, showing that reestimation is appropriate. We observed that the computational overhead of `r` is negligible using a careful implementation, and is dominated by the savings of reclassifications obtained by reestimates. `r-light` shows that minor simplifications to `r` drastically deteriorate the result. Indeed, the number of reclassifications is even higher than the naïve `fifo` order. Using AdamA, no classification of a pair of elements is performed more than once. We measure the goodness of r by counting the number of duplicates found. We have already evaluated the similarity measure used in these experiments in [9], where it obtained high recall and precision. Here, we are only interested in the influence of the order. Fig. 2(b) shows the result normalized by the maximum number of duplicates found among the different orders. We see that the amount of duplicates found is highest when using `r` and lowest when using `r-light`. As `r` misses the fewest duplicates that can be found by our similarity measure, the effectiveness obtained using AdamA with order `r` is best. It is interesting to note the correlation of the amount of reclassifications in ReconA and the number of duplicates found in ReconA. Clearly, the less reclassifications we miss in AdamA, the more duplicates we find.

[1] http://www.imdb.com and http://film-dienst.kim-info.de/

5. Conclusion

We have briefly discussed a new comparison strategy that performs well for all types of relationships, not only 1:n as previous approaches mostly did. It is based on the definition of dependencies in the form of candidates and object descriptions provided by an expert. Due to cyclic dependencies, it is possible that a pairwise classification is performed more than once, which compromises efficiency. To reduce this effect, we proposes an order that reduces the number of reclassifications. We applied it to two algorithms, namely ReconA and AdamA. Whereas ReconA uses the order to increase efficiency by reducing reclassifications, AdamA, which performs a classification at most once, uses the order to maintain good effectiveness by missing few reclassifications and hence few potential duplicates. Preliminary experiments showed that the order achieves these goals better than some other orders. In the future, we will consider how relationships can be exploited when less than O(n^2) comparisons are feasible due to efficiency concerns.

References

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

[2] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting relationships for object consolidation. In *SIGMOD-2005 Workshop on Information Quality in Information Systems*, Baltimore, MD, 2005.

[3] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, Baltimore, MD, 2005.

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

[5] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications*, Kyoto, Japan, 2003.

[6] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *ICDE Conference*, pages 294–301, April 1993.

[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, May 1997.

[8] M. Weis. Fuzzy duplicate detection on XML. In *VLDB PhD Workshop*, Trondheim, Norway, 2005.

[9] 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.