Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Replica Detection and Eliminating XML Duplicates in Hierarchical Data

Padmaasree R
PG Scholar
Department of CSE
Chettinad College of Engineering & Technology, India

Rathika R
PG Scholar
Department of CSE
Chettinad College of Engineering & Technology, India

Rajkumar R
Sr.Assistant Professor
Department of CSE
Chettinad College of Engineering & Technology, India

Abstract - Duplicate Detection consist in detecting multiple representations of a same real - world object. Hierarchical data consist of both tree and graph model. The Novel comparison strategy that uses graph model in terms of relationships proposed for hierarchical data. Instead, pairs of objects at any level of the hierarchy are compared in an order that depends on their relationships. Objects with many dependants influence many other duplicity-decisions and thus it should be decided early if they are duplicates themselves. We present an novel iterative algorithm for duplicate detection system called REVISE. REVISE allows to re-examine an object if its influencing neighbors turn out to be duplicates. Here ordering reduces the number of such re-comparisons so that it improves the efficiency of the system. The overall Complexity of REVISE algorithm is N.

Keywords : record linkage, merge/purge , object matching, object consolidation and reference reconciliation

1. INTRODUCTION

Duplicate detection is the problem of determining that different representations of entities actually represent the same real-world object. Duplicate detection is a necessary task in data cleansing [4, 5] and is relevant for data integration [2], personal information management [3], and many other areas. The problem has been studied extensively for data stored in a single relational table [1] with sufficient attributes to make sensible comparisons. However, much data comes in more complex structures, so conventional approaches cannot be applied. For instance, within XML data[6], XML elements may lack any text. However, the hierarchical relationships with other elements potentially provide enough information for meaningful comparisons. The problem of XML duplicate detection is particularly tackling in applications like catalog integration or online data cleansing.

Basically, we consider an object to depend on another object if the latter helps finding duplicates of the first. For example, actors help find duplicates in movies, so movies depend on actors, and actors influence movies. Due to mutual dependencies that can occur, detecting duplicates of one XML element helps find duplicates of the other, and vice versa. Therefore, algorithms such as [3] use dependencies to increase effectiveness by performing pairwise examining more than once.

Recently, a new class of algorithms for duplicate detection has emerged. We refer to that class of algorithms as *duplicate detection in graphs (DDG)* algorithms[8]. These algorithms detect duplicates between representations of entities in a data source, called *candidates* by using relationships among candidates. Within this class, we focus on those algorithms that iteratively detect duplicates when relationships form a graph.

Fig.1 Many to Many Relationship

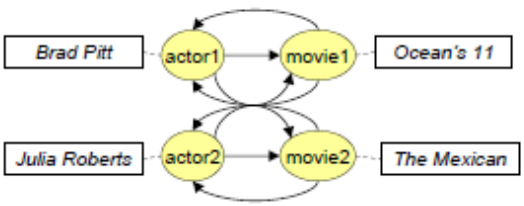


Fig.2 Graph Representation

© 2014, IJARCSSE All Rights Reserved

Fig. 1 shows a many-to-many (M:N) relationship between object types movie and actor. The absence of arrows shows that an actor can play in several movies, and several actors can play in a single movie. On instance level (Fig. 2, we observe that objects and their relationships form a graph.

2. RECENT WORK

In [9, 10], presented similar algorithms to [7], and focused on the impact of comparison order and recomparisons on the efficiency and effectiveness of DDG. The comparison order can significantly compromise efficiency and effectiveness in data graphs with high connectivity. The RC-ER algorithm first introduced in [7] and further described and evaluated in [2, 4] represents candidates and dependencies in a reference graph. The algorithm re-evaluates distances of candidate pairs at each iterative step, and selects the closest pair according to the distance measure. Duplicates are merged together before the next iteration, so effectively clusters of candidates are compared. Dong et al. perform duplicate detection in the PIM domain by using relationships to propagate similarities from one duplicate classification to another [3]. The main focus of their approach is the increase of effectiveness by using relationships. In contrast, we concentrate on increasing efficiency by using relationships.

3. EXISTING SYSTEM

In Existing system , the work was carried out using XML duplicate detection system called XMLDup.XMLDup uses the Bayesian Network to determine the probability of two nodes being the duplicates by using the threshold value and the elimination of those duplicates are carried out using Pruning Technique. By doing this we can able to achieve high precision and recall scores. XMLDup is also able to outperform another state-of-the-art duplicate detection solution, both in terms of efficiency and of effectiveness.

4. PROPOSED SYSTEM

A. Comparison Order:

It is based on the estimated number of re-comparisons of neighboring elements that become necessary if the pair indeed were a duplicate.

B. REVISE Algorithm:

REVISE Algorithm allows a pairwise comparison to be performed more than once, and the proposed comparison order reduces the number of re-comparisons. It is based on the observation that detecting duplicates to an object may affect similarity and duplicate classification on other objects. It is an exact algorithm for solving the duplicate detection problem, in that it guarantees to find *all* duplicates as defined by the similarity measure and the ODs.

The algorithm has two phases . The initialization phase (lines 2-10) defines a priority queue *OPEN*, which contains all pairs of candidates. The priority order of *OPEN* is the ascending order of
r. *DUPS* is a set of candidate pairs and is used to keep track of found duplicates to avoid unnecessary recomparisons, and to compute the rank r where the set of classified neighbor pairs (see Equations 4 and 5)
is the set of neighbor pairs that are in DUPS. *CLOSED* is the set of possibly re-classified pairs, i.e., a set of pairs that have been classified as duplicates. Only these pairs can be added to *OPEN* again. For duplicate
classification, we specify a similarity threshold μ. During the initialization phase, we also initialize the data graph G by reading the XML data.

The comparison phase (10-12) compares pairs of candidates using a similarity measure. As a reminder, to classify pairs of candidates as duplicates, we use a thresholded similarity approach, i.e., if sim(v, v') > μ they are classified as duplicates. Detected duplicates are added to *DUPS* to ensure that duplicates are never revoked in subsequent comparisons. Consequently, similarity can only increase when re-calculated, and the algorithm always terminates. Non-duplicate pairs of dependent neighbors of v and v'are fed back into *OPEN* according to the procedure updateOpen(v, v')
.

The procedure updateOpenRevise(v, v') first determines the set of dependent neighbor pairs of v and V' . Pairs in *DUPS* are not added back to *OPEN*, because they are already known to be duplicates.
If a pair is not in *DUPS*, we distinguish two cases. In the first case, the potentially added pair (n1, n2) is already in *OPEN*, because it has not been classified yet, and we merely update its position in the priority
queue according to the newly calculated rank. This is required because the value of r(n1, n2) depends on duplicates among I(n1) and I(n2), and the neighbor pair (v, v') has just been classified as duplicate. In case (n1, n2) is neither in *OPEN*, nor in *DUPS*, (n1, n2) is pushed into *OPEN*.

1)REVISE Algorithm:

1 G← data graph;
2 *OPEN*←priority queue of candidate pairs ordered in ascending order of r;
3 *DUPS*←set of duplicate pairs;
4 *CLOSED*← set of possibly re-classified pairs;
5 θ← similarity threshold;
6 Initialize G;
7 Add all candidate pairs to *OPEN*;
8 while *OPEN not empty* **do**
9 (ci, cj)← *OPEN*.popFirst ();
10 sim ← sim(ci, cj);

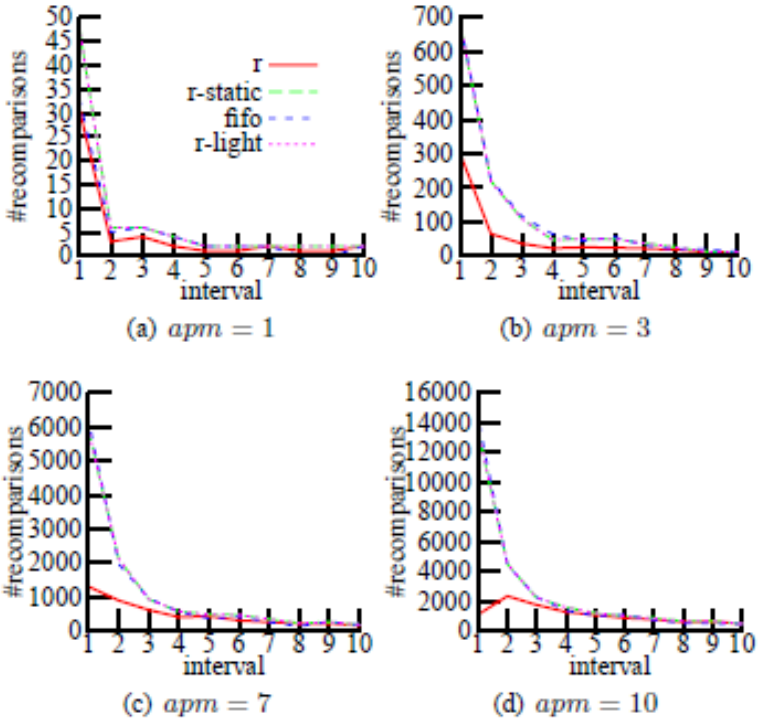
© 2014, IJARCSSE All Rights Reserved

11 if sim > θ **then**
12 *DUPS* ← *DUPS* ∪ {(ci, cj)};
13 updateOpenRevise(ci,cj);

2) Updating Open in REVISE:

1 Input : Candidate c, candidates c′
2 D(v, v′) = {(n1, n2) |n1 ∈ D(c) ∧ n2 ∈ D(c′) ∧ n1 6= n2};
3 foreach (n1, n2) ∈ D(c, c′) **do**
4 if (n1, n2) *not* ∈ *DUPS* **then**
5 rupdate := r(n1, n2);
6 if (n1, n2) ∈ *OPEN* **then**
7 *OPEN*.updateRank ((n1,n2),rupdate);
8 else if (n1, n2) ∈ *CLOSED* **then**
9 *OPEN*.push ((n1, n2),rupdate);

5. RE-EXAMINE RESULTS



Order	Description
r	the order obtained using rank r
r-static	order r at initialization
r-light	order defined by $\|D(v)\| * \|D(v')\|$
fifo	first-in-first-out

Fig .3 Re-Examine Using REVISE

The graphs show the number of re-comparisons necessary for a given connection degree and a varying interval i. We observe that for large values of i, the number of re-comparisons is generally low but increases with increasing connection degree, meaning with decreasing i (as more <m> get actors) and increasing apm. This is easily explained by the fact that the higher i, the less mutual dependencies we have, so the less re-comparisons are possible. Similarly, the higher the connection degree, the higher the number of re-comparisons necessary when duplicates are detected.

6. CONCLUSION

Novel duplicate detection approach for XML data, which, unlike the common top-down and bottom-up approaches, performs well in presence of all kinds of relationships,i.e., m:n. The comparison strategy we presented considers pairwise comparisons in ascending order of a rank, which estimates how many pairs must be reconsidered if the original pair was classified at the current processing state. We applied this strategy to algorithms: REVISE uses the order to classify only

© 2014, IJARCSSE All Rights Reserved

few pairs more than once. Experiments showed that the order obtained using ascending r is very effective in reducing the number of re-comparisons for REVISE, given a high interdependency between entities. For low interdependency there are only few possible re-comparisons, so the difference between the orders is not significant for efficiency.

ACKNOWLEDGMENTS

This work was supported by national funds through FCT Fundac¸a˜o para a Ciencia e a Tecnologia, under project PEst-OE/EEI/LA0021/2011, and PhD grant SFRH/BD/41450/ 2007 (Lu´s Leita˜o). Part of this work was done while Melanie Herschel was at the University of Tu¨ bingen.

REFERENCES

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

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

[3] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *International Conference on the Management of Data (SIGMOD)*, Baltimore, MD, 2005.

[4] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *International Conference on Very Large Databases (VLDB)*, pages 371–380, Rome, Italy, 2001.

[5] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin, Volume 23, pages 3-13*, 2000.

[6] I. Bhattachary, L. Getoor, and L. Licamele. Query-time entity resolution. In *Conference on Knowledge Discovery and Data Mining (KDD)*, Philadelphia, PA, 2006. Poster.

[7] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 2004.

[8] I. Bhattacharya and L. Getoor. Entity resolution in graph data. Technical Report CS-TR- 4758, University of Maryland, 2006.

[9] M. Weis and F. Naumann. Detecting duplicates in complex XML data. In *Conference on Data Engineering (ICDE)*, Atlanta, Georgia, 2006. Poster.

[10] M. Weis and F. Naumann. Relationship-based duplicate detection. Technical Report HUIB-206, Humboldt University Berlin, 2006.

© 2014, IJARCSSE All Rights Reserved