Scaling up Duplicate Detection in Graph Data[*]

Melanie Herschel
Hasso-Plattner-Institut
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
melanie.herschel@hpi.uni-potsdam.de

Felix Naumann
Hasso-Plattner-Institut
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
felix.naumann@hpi.uni-potsdam.de

ABSTRACT

Duplicate detection determines *different* representations of real-world objects in a database. Recent research has considered the use of relationships among object representations to improve duplicate detection. In the general case where relationships form a graph, research has mainly focused on duplicate detection quality/effectiveness. Scalability has been neglected so far, even though it is crucial for large real-world duplicate detection tasks.

We scale up duplicate detection in graph data (DDG) to large amounts of data using the support of a relational database system. We first generalize the process of DDG and then present how to scale DDG in space (amount of data processed with limited main memory) and in time. Finally, we explore how complex similarity computation can be performed efficiently. Experiments on data an order of magnitude larger than data considered so far in DDG clearly show that our methods scale to large amounts of data.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Data quality, duplicate detection, entity resolution, scalability

1. INTRODUCTION

Duplicate detection has been addressed in a large body of work [3]. We classify duplicate detection research along three dimensions, namely *data*, *approach*, and *algorithm focus*. For data, we distinguish (i) data in a single *table*, without multi-valued attributes, (ii) *tree* data, such as data warehouse hierarchies or XML data, and (iii) data represented as a *graph*, e.g., data for personal information management. The second dimension discerns between three approaches used for duplicate detection: (i) machine *learning*, where models and similarity measures are learned, (ii) the use of *clustering* techniques, and (iii) *iterative* algorithms, which classify one pair of candidates at every iteration. Considering the algorithm focus dimension, we observe that most articles focus on either effectiveness, efficiency, or scalability. Research on effectiveness is concerned with improving precision and recall, for instance by developing sophisticated similarity measures [2] or by considering relationships [6]. Research on efficiency assumes a given similarity measure and develops algorithms that avoid applying the measure to *all pairs* of objects [4]. To apply methods to very large data sets, it is essential to scale not only in time but also to scale in space, for which relational databases are commonly used [4].

We observe that for duplicate detection in graph data, no methods for scalable iterative duplicate detection have been proposed, a shortcoming we address in this paper.

[*]A full version of this paper is available. See [8] for details.

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.
ACM 978-1-59593-991-3/08/10.

2. DDG AS SEEN FROM OUTER SPACE

All iterative DDG algorithms proposed so far adhere to the following framework: During *initialization*, a graph representation of the considered data is created, precomputations are performed, and a priority queue of pairs of candidates (object representations among which duplicates should be detected) is set up. During the *iterative phase*, pairs of candidates are retrieved from a priority queue, are then classified as duplicates or non-duplicates, and potentially trigger some update in the priority queue. The most significant update is to sort the priority queue when a duplicate was classified. This is done to reduce the number of classifications performed for a candidate pair (which can be more than one in DDG). Scaling up initialization is discussed in the extended version of this paper [8] and we only summarize scaling up the iterative phase here, which is summarized in Fig. 1. We show how a pair of actor candidates (a1,a1') is retrieved from the priority queue residing in main memory. As it is classified as a duplicate, the movie candidate pair (m1,m1') rises in the order of the priority queue, because its similarity increases based on the fact that actor a1 playing in m1 is the same as actor a1' playing in m1'.

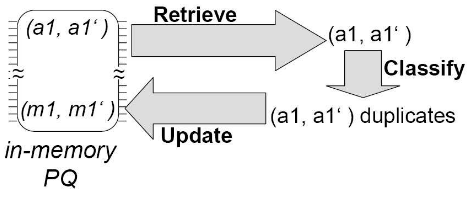

Figure 1: Sample iteration for DDG

3. SCALING UP ITERATIVE PHASE

To scale up the iterative phase of DDG, we scale up the individual steps. To this end, we assume that all necessary data structures, in particular the graph and the priority queue are stored in a relational database. Scalable DDG is summarized in Fig. 2.

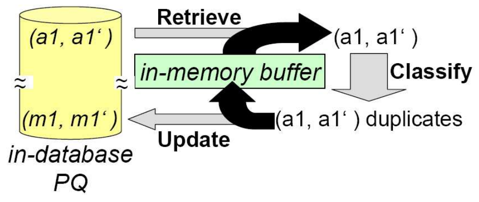

Figure 2: Sample RECUS/BUFF iteration

3.1 Scaling up Retrieval and Update

To scale up retrieval and update, we propose our RECUS/BUFF algorithm. It uses an in-memory buffer B_s of fixed size to avoid sorting the priority queue each time a duplicate is found. The intuition behind RECUS/BUFF is that although ranks of several pairs may change after a duplicate has been found, sorting the priority queue immediately after finding the duplicate is not always necessary and may actually occur several times before an affected pair is retrieved. For instance, when a1 and a1' are detected to be duplicates, the pair consisting of the movies they respectively play in, i.e., (m1,m1') rises in the priority queue, e.g., from position 7 to position 5. Although it is now closer to the head of the priority queue, any pair at position 4 or less will be compared first. Hence, sorting the priority queue does not immediately affect the comparison order and should therefore be avoided. To this end, we use B_s to temporarily store candidate pairs whose position in the priority queue decreased. At each retrieval step, we check if the pair coming from the in-database priority queue has a lower or equal position to the pair in the buffer. Depending on the result, either the pair from the database or the pair from the buffer is compared first. When the buffer overflows, we update the in-database priority queue and sort it again. Until this happens, using the internal buffer avoids sorting the potentially large priority queue, which significantly improves runtime while scaling DDG to large amounts of data.

3.2 Scaling up Classification

To classify pairs as duplicates or non-duplicates, a similarity-based approach is often used: If the similarity is above a specified threshold, the pair is classified as a duplicate, otherwise it is classified as a non-duplicate. We investigate three variants of efficiently computing the similarity of a pair.

SQL. As the data is stored in a database, we can in principal compute the similarity of a pair using SQL. However, the type of similarity function is then limited by the expressive power, which is especially a problem when the similarity measures use other aggregate functions to aggregate attribute similarities than those specified in SQL.

Hybrid/Complete. To overcome the limit of expressive power of the SQL variant, we define the Hybrid/Complete variant that essentially retrieves all necessary data for the computation of the similarity from the database, which is then processed outside the database to obtain the final similarity.

Hybrid/Optimized. Having control over the processing of the data, we can further optimize the similarity computation by retrieving and processing the data in such a way that we can abort retrieval and computation as soon as it is mathematically impossible that the similarity exceeds the predefined threshold. We call this technique early classification, because it classifies a pair as non-duplicate before the actual similarity is computed.

4. EVALUATION

Table 1(a) summarizes how the different phases of DDG scale in time depending on the size of the data s, the duplicate ratio dr, and the connectivity c, which in total amounts to a linear behavior. Table 1(b) summarizes results *reported* for other DDG algorithms. Tab. 1(b) reports on the data set size, runtime (without initialization time) and the parameters for which the algorithms do *not* scale linearly (s, dr, and c are considered). We observe that RECUS/BUFF takes comparably long, but this comes as no surprise as DB communication overhead and network latency add to the runtime. More interestingly, none of the DDG algorithms except RECUS/BUFF scales linearly in time with all three parameters s, dr, and c. Indeed, all algorithms but RECUS/BUFF do not scale linearly in time with the data set size s, which compromises scaling up DDG to large amounts of data. Note that further experiments are reported in [8].

Parameter	Retrieval & update	Classification
PQT size s	linear	linear
duplicate ratio $dr < 0.8$	linear	constant
connectivity c	linear	constant
Overall	**linear**	**linear**

(a) DDG scalability using RECUS/BUFF and HYB/O

Approach	# candidates	Runtime (s)	Not linear in
RC-ER [1]	68,000	890	s, c
RelDC [5]	75,000	180 - 13,000 [a]	s, c
LinkClus [9]	100,000	900	s
RECUS/BUFF	**1,000,000**	**24,433**	-

[a] depending on connectivity

(b) Comparison time for different approaches

Table 1: Comparative evaluation

5. CONCLUSION

This paper is the first to consider scalability of duplicate detection in graphs (DDG). We showed that using RECUS/BUFF and a suited classification strategy such as Hybrid/Optimized, we can scale up DDG to large amounts of data not fitting in main memory. Part of the research presented here was successfully applied to an industry project [7].

6. REFERENCES

[1] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1), March 2007.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD Conference*, Washington, DC, 2003.

[3] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1), 2007.

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

[5] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst.*, 31(2), 2006.

[6] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *PKDD Conference*, Porto, Portugal, 2005.

[7] M. Weis and F. Naumann. Industry-scale duplicate detection. In *VLDB Conference*, Auckland, New Zealand, 2008.

[8] M. Weis and F. Naumann. Space and time scalability of duplicate detection in graph data. Technical Report 25, Hasso-Plattner-Institut, 2008.

[9] X. Yin, J. Han, and P. S. Yu. LinkClus: Efficient clustering via heterogeneous semantic links. In *VLDB Conference*, Seoul, Korea.