A Duplicate Detection Benchmark for XML (and Relational) Data

Melanie Weis, Felix Naumann, and Franziska Brosy
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany

E-mail: {mweis,naumann,brosy}@informatik.hu-berlin.de

Abstract

Duplicate detection, which is an important subtask of data cleaning, is the task of identifying multiple representations of a same real-world object. Numerous approaches both for relational and XML data exist. Their goals are either on improving the quality of the detected duplicates (effectiveness) or on saving computation time (efficiency). In particular for the first goal, the "goodness" of an approach is usually evaluated based on experimental studies. Although some methods and data sets have gained popularity, it is still difficult to compare different approaches or to assess the quality of one own's approach. This difficulty of comparison is mainly due to lack of documentation of algorithms and the data, software and hardware used and/or limited resources not allowing to rebuild systems described by others.

In this paper, we propose a benchmark for duplicate detection, specialized to XML, which can be part of a broader duplicate detection or even data cleansing benchmark. We discuss all necessary elements to make up a benchmark: Data provisioning, clearly defined operations (the benchmark workload), and metrics to evaluate the quality. The proposed benchmark is a step forward to representative comparisons of duplicate detection algorithms. We note that this benchmark is yet to be implemented and this paper is meant to be a starting point for discussion.

1. Motivation

Duplicate detection is an important subtask of data cleaning [16], whose aim is to identify multiple but possibly inconsistent representations of a same real-world object. It is of practical relevance in important and business-relevant applications, such as data warehousing, data mining, or data integration. The problem has been studied extensively under various names, such as record linkage [26], merge/purge [17], entity resolution [5], or reference reconciliation [15], to name but a few. The proposed algorithms most often improve either efficiency or effectiveness. In the first case, the goal is to reduce the number of pairwise comparisons, which is quadratic in the number of elements if all pairs are compared. However, duplicates may be missed when pruning comparisons. The goal of such algorithms is to improve efficiency without reducing effectiveness. Representatives of algorithms concerned primarily in efficiency are the Sorted Neighborhood Method [17], its domain independent version [18], or DELPHI [1]. When concentrating on effectiveness, the goal is to find duplicates more accurately, e.g., using approaches such as [15, 24].

As numerous approaches exist both for increasing efficiency and effectiveness, it is essential to provide some common ground to compare these algorithms with each other. Some data sets (e.g., CORA[1] used in [15, 24, 8] or selected subsets of the Internet Movie Database IMDB[2] [14, 25]) have gained popularity. However, the state of the art does not allow representative comparisons between algorithms yet, for the following main reasons[3].

- **Lack of algorithm documentation.** Many duplicate detection algorithms are described in scientific papers only, and often a 12 page publication cannot cover all details and aspects of an approach. When it comes to re-implementing an existing method, the information provided in a paper is often insufficient.

- **Different testing environments.** When own results are compared to results reported in a paper, the different testing environments may falsify the comparison. That is, when measuring time, the results are only comparable if the hardware and the software are the same. Again, these parameters are often not documented and even if they are, chances

[1] `http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz`
[2] `http://www.imdb.com`
[3] While the following bullets criticize what has become common practice in publications and research, we do not exempt ourselves from some of the misdoings.

are that the systems are not the same, and we cannot up- or downgrade systems every time we want to compare our results with others.

- **Lack of common dataset.** Freely available and simultaneously interesting datasets for duplicate detection are rare. Even more seldom are datasets with true duplicates already marked. As a consequence, even if same or similar datasets were used, the results expressed as precision, recall and runtime measure are not comparable: Two approaches might not agree in what is a correctly detected duplicate and how many duplicates are in fact hidden in the dataset.

  To confront these problems, many publications create their own data sets, inject duplicates and then let their duplicate detection algorithm find them. Often these generators of data are not freely available for legal reasons or simply because the original code is lost once the programmer leaves the organisation.

- **Obscure methodology.** Comparing results to published results is further problematic, because many approaches "‘fudge"’ the original data to meet their needs. In papers, we may read sentences like "We further cleaned [a commonly used data set] up by correcting some labels" or "we used the technique [X]" without mentioning how X's tunable parameters are set. In such cases, the methodology is not reproducible, an essential property if we want to compare approaches.

In this paper, we propose a benchmark for duplicate detection that alleviates the above problems as follows:

- **Standardized data.** By applying different duplicate detection approaches on the same data, comparing efficiency or effectiveness of different approaches is easy. The proposed benchmark proposes data from several different domains. For the artificially generated data the size of the data and error characteristics can be varied in order to evaluate algorithms under varying settings. For the real-world data we merely characterize their properties.

- **Clearly defined operations and metrics.** The problem of lacking documentation about algorithms and experimental methodology is alleviated by defining operations that an algorithm may perform, as well as some clearly defined metrics to evaluate the results of these operations. Our benchmark proposal supports several variants of the duplicate detection problem. For example, the benchmark considers both detection of duplicate pairs as well as detection of duplicate clusters. Another example is the distinction between duplicate detection as a batch process or duplicate search. Results are then evaluated using a set of metrics.

- **Central processing.** We envision that the benchmark is executed on a central server to which a duplicate detection algorithm can be submitted, which in turn will be executed on the server. This way, the testing environment is guaranteed to be the same across different approaches.

The duplicate detection benchmark proposed in this paper focuses on XML data because the XML data model is less restrictive than for instance the relational data model, which makes some problems in duplicate detection even more difficult. Nevertheless, duplicate detection algorithms focusing on flat relational data can use the benchmark, because it can produce flat XML data that easily maps to relational data.

Our proposed domains and datasets are described in Sec. 2. Then, we define the operations supported by our benchmark in Sec. 3. Sec. 4 summarizes metrics used to evaluate the performance of the algorithms. Tools supporting the benchmark are described in Sec. 5. Related work is summarized in Sec. 6 before we conclude and point out future work in Sec. 7.

2. Domain and Data

A key point to any data management benchmark is the choice of a suitable data set that is representative of real world data and the tasks to manage, here duplicate detection. In comparison to benchmarks for the efficiency of database systems the choice of data for a duplicate detection benchmark is even more important: While speeding up a database is a purely syntactical task, whose boundaries can be tested by simply producing *more* data, duplicate detection is a semantic task, i.e., a task that can only be solved by understanding (or guessing at) the *meaning* of data. Its boundaries can only be tested by producing more *difficult* data. In the following sections we elaborate on the domain the data of an XML benchmark might come from, on the actual data set and if necessary how to generate it, and finally on the types and numbers of duplicates in the data.

2.1 Domain

The choice of domain for the XML benchmark should reflect the usage of XML data in today's applications. Therefore, we recommend to use domains from real-world applications for benchmarking. Using real-world domains allows to generate or use data that has similar characteristics to real-word data from that domain. Another criterion for selecting a domain is the understandability of the data. Indeed, we have seen that duplicate detection is a semantic task, so understanding the data one is working on is crucial.

As data generation according to different criteria is part of the benchmark we envision, we first consider domains where data generators are readily available. More specifically, we consider the following domains used by the XML benchmarks XMach-1[4] [6] and XMark[5] [23]. Furthermore, to support benchmarking of approaches focusing on flat relational data, we use the popular domain of customer relationship management.

- *XMach-1* simulates a web application, a typical use case of an XML data management system. The system architecture consists of an XML database, application servers, loaders to populate the database, and browser clients. In this domain, the XML data populating the database contains both document-centric and data-centric XML documents.

- *XMark* models an auction website as a typical e-commerce application. Here, the XML documents in the database model the auction web site and include descriptions of items, open auctions, closed auctions, persons and categories.

[4] `http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html`
[5] `http://monetdb.cwi.nl/xml/`

- *Customer Relationship Management* models data that companies store about their customers, e.g., addresses, products bought, ...), needed for analysis, business reports, customer care, etc.

In addition to the above domains for which artificial data can be generated, we consider two other domains for which real-world data is available.

- *Digital libraries*: This domain models an on-line digital library that stores information about publications (e.g., authors, title, conference, proceedings). An example for such a digital library is DBLP[6], where the actual data can be downloaded as XML. Here, duplicates stem from multiple input of data about the same object.

- *Movie databases*: This domain models data related to movies and CDs. The scenario involves several repositories of information about movies, similar to the Internet Movie Database (IMDB). Here, duplicates typically stem from different sources whose integration brings about the duplicates.

We include several different domains in our benchmark to have the possibility to test algorithms on various types and "difficulties" of data.

2.2 Data

Depending on the domain chosen for evaluation, the data differs. For the XMach-1 and XMark domain, we intend to use data produced by their respective XML data generators. For the digital library and the movie domain, both artificially generated and real-world data is provided by the benchmark.

As mentioned earlier, we focus on XML data but the benchmark also offers the possibility to evaluate duplicate detection algorithms for relational data. Hence, both variants need to be provided by the benchmark. More generally, the data provided by the benchmark allows users to experiment on different structures: from flat and same structure for all elements as in relational data to deeply nested and scarcely or heterogeneously structured data.

XML content is produced by the XMach-1 and XMark data generators when the respective domain is chosen. Advantages of using these domains and data are that (i) the XML structure is close to a real-world structure, (ii) data generation scales to large amounts of data, and (iii) data is easily generated with already existing tools. However, the text content is not necessarily representative of the domain. For instance, in the XMark benchmark where auction data is modeled, the actual text data originates from Shakespeare's plays. This is certainly acceptable for XML database benchmarks, but for XML duplicate detection, the actual data values play a more important role. Hence, we plan to offer data of the library and movie domain to create more realistic data. A method to achieve this is to collect lists of names, titles, and other data from the real-world domain (e.g., from DBLP or IMDB), and to create data by randomly combining values of these lists. Additionally, the creation of standardized relational CRM data can be achieved with the UIS Database Generator[7].

[6]`http://www.informatik.uni-trier.de/~ley/db/`

[7]written by Mauricio Hernández and available at http://www.cs.utexas.edu/users/ml/riddle/data/dbgen.tar.gz

To evaluate efficiency and scalability of an approach, the size of the input data needs to be variable by some scale-up factor. The data generators of the existing XML benchmarks and the one we envision to create bibliographical and movie data allow the creation of small to large documents by varying the number of elements in the XML files.

In addition to efficiency and scalability, the quality of the result of duplicate detection needs to be measured. This requires "dirty" data as testing data, a property difficult to implement, as discussed next.

2.3 Data contamination

Data contamination refers to the type and number of duplicates within the data. An XML duplicate detection benchmark should have two modes of contamination: first, existence of duplicates that are already part of the real world data set, and second, insertion of duplicates into the data set given a set of parameters. Both modes are difficult to implement.

Existing duplicates. A contaminated XML data set from the real-world might already contain duplicates. The difficulty of implementing the benchmark using this kind of data and contamination is to actually find and mark these duplicates. Without such a pre-analysis of the data one cannot evaluate precision and recall of a duplicate detection algorithm. To actually determine all true duplicates in a data set without relying on approximate algorithms, one must manually compare all pairs of objects and determine whether they are duplicates—a tedious and possibly error-prone task.

We are currently developing and testing a tool, called *ManDup* for manual duplicate detection, which supports persons in the process of manually iterating through all pairs of objects and their decision if the pair is indeed a duplicate. Each pair is analyzed by at least two persons and there is the possibility to defer the decision to a domain expert.

Inserting duplicates. Artificially contaminating a data set on the other hand gives at least the knowledge of all duplicates, assuming the data set was clean from the outset. Here the challenge is to determine *how* to create duplicates, *how many* to create, and *where* to place them in the data set. We consider the following types of data contaminations when duplicating objects (some contamination types can be combined with others).

- *Exact duplicates*: No change is made to the duplicated object.

- *Contradictory text*: Text includes text nodes and attribute values and is considered contradictory if two or more text values exist and are different. Contradictory text is for example caused by typographical errors, synonyms, abbreviations or incomplete text.

- *Missing data*: Missing data refers to missing structural data (elements and attributes) as well as to missing textual data (text nodes and attribute values).

- *Wrong referential constraints*: Key references may point to keys that do not actually represent the related entity.

- *Different structure*: The text remains the same but structure is changed by renaming labels, by changing the order of subelements, and by moving subelements across levels of the hierarchy.

Contamination Type	How	Where
Exact duplicates	Exact copy of an element The original element and the copy are assigned a common identifier to identify them as duplicates.	Applied on instance of a specified element type. Duplication of an instance occurs with a specified probability. The maximum number of duplicate is parameterized as well. Duplicates can be inserted 1. after the original. 2. at a random offset of the original. 3. at the beginning or at the end of the document.
Contradictory text	Insert, delete, swap, or replace specified number of characters in text.	In text of an instance of a specified element type. The contamination (insert, delete, ...) is applied on a generated duplicate with a specified probability.
Missing data	Delete text or element.	Applied on an instance of a specified element type. Deletion is performed on a generated duplicate with a specified probability.
Wrong referential constraints	Change, add, delete reference element.	Specified on a referencing element type. Reference is contaminated on generated duplicate with a specified probability.
Different structure	Element names are contaminated similarly to text contamination (insert, delete, swap character). Order of sibling elements is changed by swapping siblings. A subelement is moved up in the hierarchy and replaces an ancestor.	Rename is performed on an instance label of a specified element type with a specified probability. Swapping siblings occurs with a specified probability and requires the specification of the element type. Changes in the hierarchy are parameterized by a probability, the element type, and the number of levels an element of that type is shifted upwards.

Table 1. Contamination of Duplicate Elements

In Tab. 1 we describe how these types of contaminations can be implemented and where they are introduced. Basically, contamination is applied on user-specified element types with a specified probability. Duplicate elements are created as copies of elements in the clean data, and the other contaminations are only applied on these duplicated elements. A sample specification of a contamination could read like this. Movie elements should be duplicated at most three times with a probability of 50 %. These duplicates contain the following errors: (i) the actor names include a character swap with a probability of 30 %, and (ii) actor elements are missing with 30% probability. (iii) In 10 % of all cases, references to production studios are corrupted, meaning deleted (80% probability) or swapped (20% probability). (iv) Structure is contaminated by swapping characters of `<title>` with a probability of 50 %. Considering the movie element shown in Tab. 2, a contaminated version according to the above definitions could be the XML element of Fig. 3, in the event that all contaminations apply on this particular duplicate.

Using data generated according to our proposal, the benchmark can ask duplicate detection algorithms to solve several different tasks, which we describe next.

```
<movie>
  <title> Troy </title>
  <actor>Pitt</actor>
  <actor>Bana</actor>
  <actor>Cox</actor>
  <prodCom>
    refWB
  </prodCom>
</movie>
```

Table 2. Duplicate Contamination Example

```
<movie>
  <titel> Troy  </titel>
  <actor>iPtt</actor>

  <actor>Cox</actor>
  <prodCom>
    refRP
  </prodCom>
</movie>
```

Table 3. Duplicate Contamination Example

3. Benchmark Tasks

As duplicate detection approaches specialize on different aspects, our XML duplicate detection benchmark supports several tasks (the “workload”), as given in Tab. 4. The different tasks are described in more detail in the following subsections.

3.1 Pairs vs. Cluster

The goal of duplicate detection is to identify multiple representations of a same real-world object. Clearly, the number of representations is not limited, therefore, the final goal is to identify clusters of duplicates. Two techniques have been used. Algorithms such as [17, 20] detect duplicates pairwisely and then cluster duplicate pairs by computing the transitive closure over these pairs. Opposed to that, algorithms such as [5, 12] apply clustering algorithms to directly identify clusters of duplicates, i.e., they skip the intermediate step of finding duplicate pairs. The benchmark we propose in this paper supports both the task of pairwise duplicate detection and of clustered duplicate detection:

Task name	Description
Pairwise d.d.*	Detects pairs of duplicates.
Clustered d.d.	Detects clusters of duplicates.
Batch d.d.	Detects all duplicates in a source.
Duplicate search	Detects duplicates to a particular element.
Algorithm-centric	Concentrates on efficiency, given a similarity measure.
Similarity-centric focused	Uses a provided comparison algorithm to evaluate a similarity measure.
Mixed focus	Detects duplicates without using any provided technique.

*d.d. = duplicate detection

Table 4. Task overview

Task 1 Pairwise duplicate detection*: Within a given data set find duplicate pairs such that all pairs within a cluster are detected. For instance, consider duplicates a, b, and c, the pairs to detect as duplicates are (a, b), (a, c), and (b, c).*

Task 2 Clustered duplicate detection*: Within a data set find duplicate clusters such that (i) all elements within a cluster are true duplicates and (ii) no true duplicates span two different clusters.*

3.2 Batch vs. Search

Duplicate detection can be considered as batch process, where all pairs of duplicates are determined in a single process (e.g., [22, 24]), or it can be considered a search problem, i.e., given a particular element, find its duplicates in a given data set (e.g.,[10, 11]). The latter is particularly important to support during data input, so that users can warned of a possibly existing entry for the particular real-world object. For batch duplicate detection, both pairwise and clustered duplicate detection are possible. For duplicate search, the goal is to find a subset of elements in a data set that includes all duplicates of the searched element and excludes any non-duplicate.

Task 3 Batch duplicate detection*: Find all duplicate pairs or clusters in a data set.*

Task 4 Duplicate search*: Given an element e, find duplicates of e in a given data set. The result is expressed as a cluster of duplicates. That is, if $P = \{p_1, ..., p_n\}$ is the set of positive matches returned by the search, then the result is $\{e, p_1, ..., p_n\}$.*

3.3 Efficiency vs. Effectiveness

As mentioned earlier, approaches generally concentrate either on efficiency or on effectiveness. Depending on the goal of the algorithm, we propose different tasks. When the primary goal is to increase efficiency, our benchmark provides a given similarity measure to the algorithm, so that no effort needs to be put into developing the similarity measure. Effectiveness can be increased using various techniques, e.g., new similarity measures [1, 25] or new algorithms [15, 24]. In the first case, comparability is obtained by applying the novel similarity measure to a reference algorithm, whereas in the second case, it is possible to use a readily available similarity measure provided by the benchmark. Essentially, depending on the goal of the algorithm, one of the tasks defined below can be used.

Task 5 Algorithm-centric. *Apply duplicate detection algorithm using a similarity measure sim provided by the benchmark.*

Task 6 Similarity-centric. *Using a reference comparison algorithm, apply similarity measure for comparisons.*

Task 7 Mixed focus. *Neither similarity measure nor algorithm are provided by benchmark.*

4. Metrics

The results obtained for one of the above operations must be evaluated using clearly defined metrics. Effectiveness is most commonly evaluated using precision, recall, and the f-measure. As for efficiency evaluation, both the number of comparisons (computational complexity) and the observed runtime can be used.

4.1 Metrics for Effectiveness

To measure effectiveness, we distinguish metrics that apply to duplicate pairs and metrics used to evaluate approaches that detect duplicate clusters.

Metrics for duplicate pairs. Pairwise duplicate detection is evaluated using recall, precision and f-measure. They have been used extensively to evaluate duplicate detection algorithms and originate from information retrieval [2]. Basically, the goal is to obtain high precision for high recall, resulting in a high f-measure.

Opposed to information retrieval, duplicate detection approaches do not provide a ranking or order of duplicate pairs as their output, but merely the *set* of duplicate pairs. Thus, precision and recall of only the final result can be computed. While some approaches do provide a ranking function, e.g., a similarity score or a confidence for duplicates, the benchmark requires to also produce a threshold. This similarity threshold ultimately decides upon the result set of duplicates.

Definition 1 ***Set definitions for pairs.*** *Let $S_{all} = \{(e_1, e'_1), ..., (e_n, e'_n)\}$ be the set of* all duplicate pairs *in a data set. A duplicate detection algorithm detecting pairs of duplicates returns a set of* positives $S_{pos} = \{(e_i, e'_i)...(e_j, e'_j)\}$. *Then, the set S_{true} of* true positives *is defined as $S_{true} = \{(e_i, e'_i)|(e_i, e'_i) \in S_{pos} \wedge ((e_i, e'_i) \in S_{all} \vee (e_i, e'_i) \in S_{all})\}$.*

Metric 1 ***Recall.*** *Recall is the fraction of true positives over all duplicate pairs.*

$$Recall = \frac{|S_{true}|}{|S_{all}|}$$

Metric 2 ***Precision.*** *Precision is the fraction of true positives over positives.*

$$Precision = \frac{|S_{true}|}{|S_{pos}|}$$

If S_{pos} is empty, precision is set to 0.

These metrics are applicable both on pairs resulting from batch duplicate detection, as well as pairs resulting from duplicate search.

Metric 3 ***F-measure.*** *The f-measure is the harmonic mean between recall and precision.*

$$F - measure = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

Metrics for duplicate clusters. To evaluate approaches that find duplicate clusters, we again use recall, precision, and the f-measure. Whereas the formal definitions for recall, precision, and f-measure remain unchanged, the sets considered in the formulas are different.

Definition 2 ***Set definitions for clusters.*** *Let S_{all} be the set of* all duplicate clusters *in a data set, such that no two clusters contain duplicates of the same entity and all elements in a cluster are duplicates. A duplicate detection algorithm detecting duplicate clusters returns a set of* positive clusters $S_{pos} = \{C_1 ... C_n)\}$*, where C_i is a set of elements and corresponds to a duplicate cluster. We define two clusters $C = \{e_1, ..., e_k\}$ and $C' = \{e'_1, ..., e'_k\}$ as equal if for all $e_i \in C, 1 \leq i \leq k$ there exists exactly one$e_j \in C', 1 \leq j \leq k$ such that $e_i = e_j$. Then, the set S_{true} of* true positives *is defined as $S_{true} = S_{all} \cap S_{pos}$*

Using the above definitions of recall and precision for clustering algorithms is strict in the sense that a cluster $C_P \in P$ needs to exactly match a cluster within $C_D \in D$ in order to be considered as true positive. That is, in cases where clusters are incomplete or contain some non-duplicates the metrics are overly pessimistic. For example, consider

$$S_{all} = \{(a_1, a_2, a_3), (b_1, b_2, b_3)\}$$

and

$$S_{pos} = \{(a_1, a_2, a_3, b_1), (b_2, b_3)\}$$

Using the above definitions, recall and precision both equal 0. We address this issue by considering a cluster as a set of pairs. More specifically, a cluster {a,b,c} consists of three pairs, namely {(a,b), (a,c), (b,c)}. Once clusters have been divided into pairs, recall and precision as defined on pairs is applied for evaluation. Considering the above example, we obtain

$$S_{all}^{pairs} = \{(a_1, a_2), (a_2, a_3), (a_1, a_3), (b_1, b_2), (b_2, b_3), (b_1, b_3)\}$$

and

$$S_{pos}^{pairs} = \{(a_1, a_2), (a_2, a_3), (a_1, a_3), (a_1, b_1), (a_2, b_1), (a_3, b_1), (b_2, b_3)\}$$

This results in a precision of $\frac{4}{7}$ at a recall of $\frac{2}{3}$.

We considered other measures for evaluating duplicate clustering approaches, including *entity dispersion* and *cluster diversity* proposed in [4]. However, as pointed out by Chen et al. [12], these metrics do not always reflect the quality of a duplicate detection result. Chen et al. propose a new entropy-based quality measure to evaluate duplicate clustering algorithms that overcome the limitations of dispersion and diversity. However, we prefer recall and precision over pairs that form clusters because it allows direct comparisons among duplicate detection algorithms that detect pairs, and algorithms that detect clusters.

4.2 Metrics for Efficiency

To measure the efficiency of an approach, we propose two metrics in our benchmark. The first one is the number of pairwise comparisons performed by an algorithm. Therefore, it only applies to pairwise duplicate detection. The second metric is the runtime of the algorithm and applies to all types of tasks.

The number of comparisons is an important measure for duplicate detection algorithms, because the difficulty of the problem lies in the fact that in principle all pairs of objects should be compared. A main feature of most algorithms is to reduce precisely this number. However, as the provider of a benchmark we cannot hope to actually determine the number of comparisons but must rely on statements of the submitters.

Metric 4 ***Pairwise comparison count.*** *This metric measures the number of pairwise comparisons performed by the algorithm. The lower the count, the more efficient the algorithm is.*

Metric 5 ***Runtime.*** *Runtime of an algorithm is the time needed from start to end of a duplicate detection algorithm.*

Runtime as defined above includes all phases of an algorithm, e.g., reading the data, performing duplicate detection, and returning a result. Runtime measurements in finer granularity that measure runtime for different phases should be included and its output produced by the program itself. All runtime measurements are suited for comparing different duplicate detection approaches, because we envision a central benchmark server where all applications are executed. That is, the system is the same for all algorithms, so time measurements are indeed comparable.

Up to this point, we have defined the data used by the benchmark and tasks that can be addressed and evaluated using metrics. To support these features, it is essential to have a system that supports data creation and pollution, lets users upload programs that can be executed and that returns evaluation results that can be analyzed by users. Additionally, the system should support easy comparisons of approaches. In the following section, we describe a possible system to run the benchmark.

5. Tools

For the duplicate detection benchmark for XML Data a standardized data basis is needed on which the algorithms can be tested. With tools like the XMLgen from XMark or with ToXGene[8] this basis can be achieved. To contaminate data we propose a tool – the Dirty XML Generator. A first version can be downloaded from the project's web site[9]. Currently, the Dirty XML Generator supports contradictory text, missing data, and duplicate elements contamination, as described in Tab. 1. Duplicate elements are only appended at the end of the document. The alteration of key references is part of future work. Other dirty data generators exist [17, 3] that actually perform both data creation and data contamination, but to the best of our knowledge, they only create relational data.

The benchmark should run on a centralized system to guarantee the comparability of time measurements. Thus, a server needs

[8]http://www.cs.toronto.edu/tox/toxgene/

[9]http://www.informatik.hu-berlin.de/mac/dirtyxml/

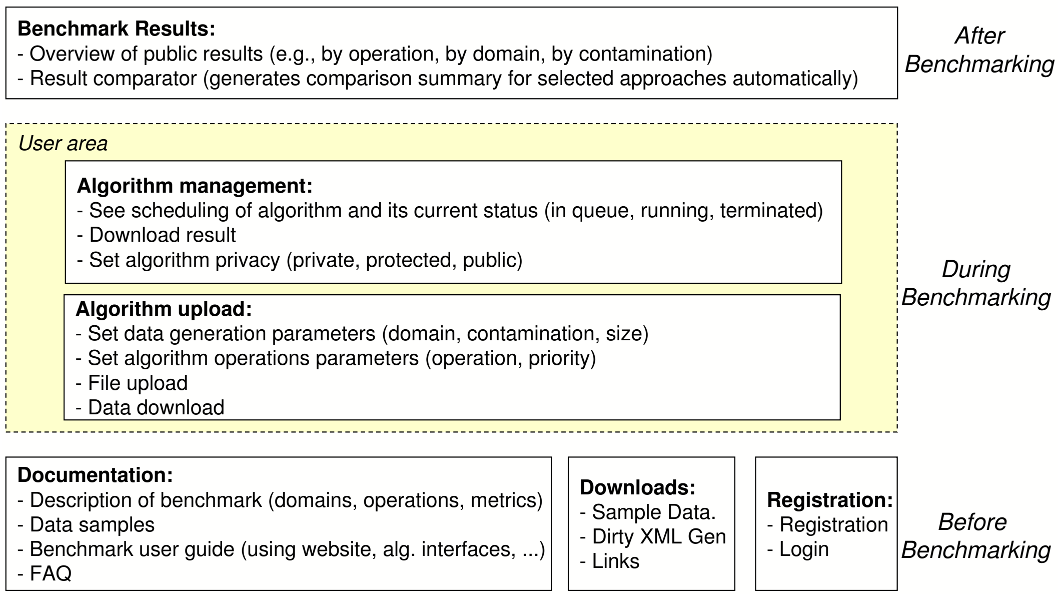
Benchmark Results:
- Overview of public results (e.g., by operation, by domain, by contamination)
- Result comparator (generates comparison summary for selected approaches automatically)
After Benchmarking
User area
Algorithm management:
- See scheduling of algorithm and its current status (in queue, running, terminated)
- Download result
- Set algorithm privacy (private, protected, public)
Algorithm upload:
- Set data generation parameters (domain, contamination, size)
- Set algorithm operations parameters (operation, priority)
- File upload
- Data download
During Benchmarking
Documentation:
- Description of benchmark (domains, operations, metrics)
- Data samples
- Benchmark user guide (using website, alg. interfaces, ...)
- FAQ
Downloads:
- Sample Data.
- Dirty XML Gen
- Links
Registration:
- Registration
- Login
Before Benchmarking

Figure 1. Web site Overview

to be set up, and interfaces need to be published. We propose two options for submitting algorithms to the benchmark server: The first option is completely file-based. That is, executable code is submitted to the benchmark server and is scheduled for execution. The second option is to submit java packages that implement predefined java interfaces, so that the algorithm can be executed on the server. The two options require dealing with security issues, e.g., the submitted executables or class files may include some malicious code.

After running a duplicate detection algorithm on the server, results are communicated to the authors. We believe that it is sufficient to return a result as a file of ID-pairs or ID-clusters, as well as a file providing some information about resources used by the algorithm (time and space). When duplicates in the input data are known, i.e., they have been generated, recall and precision can be calculated on server side, which is returned in an effectiveness summary file in addition to the other two files.

The entire benchmarking process is presented and guided through a web site. That is, researchers can communicate with the benchmark server via this web site to upload their algorithm with a certain priority, see when the algorithm execution is scheduled, download their results and publish them on the web site, or look at results of other approaches.

A proposed overview of the web site is shown in Fig. 1. The overview divides functionality into three categories: before, during, and after benchmarking. Before using the benchmark, documentation is available so that users can learn about the purpose of the benchmark, what data to expect, and how to use it. For private testing, the dirty XML generator is available for download, as well as some data samples. Furthermore, links to other used tools (e.g., ToxGene, XMLGen, XMark data generator) are provided as well. To use the benchmark, users must register and login to get access to a private user space, where the algorithm can be uploaded and managed. Before uploading an algorithm, data needs to be generated. This generation requires setting up some parameters. The generated data can be downloaded so that results can be analyzed more accurately. Users further have to set up their algorithm task, i.e., its operation. After uploading files (either java classes or executables), the server schedules the algorithm. Users can keep track of their algorithm's status looking at scheduling and status information. Once the algorithm is terminated, results can be downloaded. Results can be classified as private (the default), protected (only visible to selected users), or public (visible for all). Public results appear in the benchmark overview, that can be used to compare results to each other. The comparison may be grouped or sorted by domain, by operation, or by contamination, to name just a few possibilities. A result comparator is available that allows the selection of public approaches and that automatically generates a comparison summary. We include this result comparator to motivate users to make their results public by leveraging the task of algorithm comparison.

6. Related Work

Benchmarks have been used in various areas to provide standardized evaluation for solutions, so that they can be compared.

In this section, we do not provide a survey on benchmarks, instead, we mention a few that have been used throughout database publications. The first example of benchmarks that are widely used are the benchmarks proposed by the Transaction Processing Performance Council (TPC)[10] that focus on evaluating transaction processing and databases. For example, it provides the TPC-H benchmark that models complex queries for decision support systems, e.g., data warehouses. These benchmarks are widely used both in industry and research to evaluate and rank systems against each other (see for example the top-ten TPC-H results by performance[11]).

[10] `http://www.tpc.org`
[11] `http://www.tpc.org/tpch/results/tpch\_perf\`

Another group of benchmarks we want to point out are XML Database benchmarks used to evaluate XML database systems. Examples are XMach-1 [6], XMark [23], and XOO7[12] [9], a comparison of which is provided in [21]. Whereas XMach-1 and XMark have an application scenario and domain, XOO7 does not provide an actual application. Instead, it uses generic descriptions based on an ER model. In our choice of domains, we opted for the domains of XMach-1 and XMark because of their relative simplicity compared to XOO7. Simplicity is one of the four criteria for a domain-specific benchmark [21]. The other criteria are relevance, portability, and scalability. Our benchmark meets relevance, because it covers several domains to capture a wide range of problem difficulty. Portability is satisfied because exchanging XML data is easy. Scalability is achieved by varying data sizes from small to large.

To the best of our knowledge, no benchmark for duplicate detection is currently used. Throughout the literature, several data sets have been used by more than one approach. For instance, the CORA data set that consists of bibliographical data proposed by McCallum has been used for evaluation in [15, 24, 8]. Another popular data source is the internet movie database IMDB. Data sets extracted from IMDB have been used in [14, 25]. A repository of other available data sets (both for relational and XML data) and other valuable resources on duplicate detection is the RIDDLE repository[13]. However, most data sets require some further processing before they can be used for experiments. For example, the CORA data set includes annotations for duplicate publications, but not for authors or venues, so whenever duplicates are detected among these entities, there is no guarantee for a common data set anymore. When extracting data from IMDB, different sampling techniques (and possibly different error introduction techniques) result in data sets with different characteristics. Our benchmark leverages these problems because its centralized processing guarantees that the data is not modified and all created data sets have similar characteristics.

Providing data is only the first step towards a benchmark. It leverages the problem of using different evaluation environments, but the problems of insufficient documentation and obscure methodology remain. A benchmark framework for relational record linkage using a single relation has been proposed in [19]. More complex scenarios, e.g., XML duplicate detection, where we focus on, or relationship-based duplicate detection [15, 24] cannot be evaluated using this framework. Additionally, the framework does not provide actual data, it essentially provides criteria for data sets, which the data sets of our benchmark satisfy (e.g., provide both artificial and real-world data). The authors of [7] present a study on evaluation and training-set construction for adaptive duplicate detection, which is a necessary step towards a benchmark supporting adaptive duplicate detection methods such as [13, 22]. The authors conclude that recall-precision curves are the most appropriate method to adopt when evaluating effectiveness, which is considered in our benchmark, as well.

7. Conclusion

In this paper, we presented a benchmark for XML duplicate detection, which is easily applicable to duplicate detection in relational data as well. The benchmark facilitates comparisons between duplicate detection algorithms, which is currently a tedious task because of lack of algorithm documentation, system heterogeneity and undocumented evaluation methodologies. Indeed, the benchmark allows to apply algorithms to the same data, leveraging the problem of system heterogeneity. By defining operations that algorithms should perform, a common goal is clearly defined and achievement of that goal can be evaluated using suited metrics, again provided by the benchmark. By choosing a centralized benchmark server where all applications are run, the system environment is guaranteed to be the same. Using such centralized evaluation, there is no need for re-implementation, so algorithm documentation and experimental methodology are not as relevant for comparisons of approaches anymore. As we have seen, the proposed benchmark requires several tools and administration. Some of these tools are already available, whereas the environment and administration is still an open question.

We believe that putting some effort in realizing this benchmark will significantly improve the quality of scientific publications in the area of duplicate detection. And eventually, this benchmark may become part of a more general data cleaning benchmark. We hope that this contribution will spark fruitful discussions leading to a wide acceptance of a new data cleaning benchmark.

Acknowledgment. This research was supported by the German Research Society (DFG grant no. NA 432).

`_results.asp`

[12]`http://www.comp.nus.edu.sg/~ebh/XOO7.html`

[13]`http://www.cs.utexas.edu/users/ml/riddle/`

8. 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] R. A. Baeza-Yates and B. A. Ribeiro-Neto. Modern information retrieval. *ACM Press / Addison-Wesley*, 1999.

[3] P. Bertolazzi, L. D. Santis, and M. Scannapieco. Automatic record matching in cooperative information systems. In *Workshop on Data Quality in Cooperative Information Systems*, pages 13–20, Siena, Italy, January 2003.

[4] 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.

[5] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. *Workshop on Multi-Relational Data Mining (MRDM)*, 2005.

[6] T. Böhme and E. Rahm. XMach-1: A benchmark for XML data management. In *Proceedings of BTW 2001*, pages 264–273, Oldenburg, Germany, March 2001.

[7] M. Bilenko and R. Mooney. On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD 2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003.

[8] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, Washington, DC, 2003.

[9] S. Bressan, M.-L. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar. The XOO7 benchmark. In *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques*

and Data Integration over the Web-Revised Papers, pages 146–147, London, UK, 2003. Springer-Verlag.

[10] E. Cesario, F. Folino, G. Manco, and L. Pontieri. An incremental clustering scheme for duplicate detection in large databases. In *Proceedings of the International Database Engineering Application Symposium (IDEAS)*, pages 89–95, Montreal, Canada, 2005.

[11] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *International Conference on the Management of Data (SIGMOD)*, pages 313–324, San Diego, California, 2003.

[12] 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.

[13] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[14] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: A profiler-based approach. In *IIWeb*, pages 53–58, 2003.

[15] 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.

[16] 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.

[17] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *International Conference on Management of Data (SIGMOD)*, pages 127–138, San Jose, CA, May 1995.

[18] 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.

[19] M. Neiling, S. Jurk, H.-J. Lenz, and F. Naumann. Object identification quality. In *Workshop on Data Quality in Cooperative Information Systems 2003 (DQCIS)*, Siena, Italy, 2003.

[20] S. Puhlmann, M. Weis, and F. Naumann. XML duplicate detection using sorted neigborhoods. *International Conference on Extending Database Technology (EDBT)*, 2006.

[21] E. Rahm and G. Vossen. Web und Datenbanken. *dpunkt.verlag*, 2002.

[22] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, Edmonton, Alberta, 2002.

[23] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, Hong Kong, China, August 2002.

[24] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, Porto, Portugal, 2005.

[25] M. Weis and F. Naumann. DogmatiX tracks down duplicates in XML. In *International Conference on the Management of Data (SIGMOD)*, Baltimore, MD, 2005.

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