DuDe: The Duplicate Detection Toolkit

Uwe Draisbach
Hasso Plattner Institute
Potsdam, Germany
uwe.draisbach@hpi.uni-potsdam.de

Felix Naumann
Hasso Plattner Institute
Potsdam, Germany
naumann@hpi.uni-potsdam.de

ABSTRACT

Duplicate detection, also known as entity matching or record linkage, was first defined by Newcombe et al. [19] and has been a research topic for several decades. The challenge is to effectively and efficiently identify pairs of records that represent the same real world entity. Researchers have developed and described a variety of methods to measure the similarity of records and/or to reduce the number of required comparisons. Comparing these methods to each other is essential to assess their quality and efficiency. However, it is still difficult to compare results, as there usually are differences in the evaluated datasets, the similarity measures, the implementation of the algorithms, or simply the hardware on which the code is executed.

To face this challenge, we are developing the comprehensive **du**plicate **de**tection toolkit "DuDe". DuDe already provides multiple methods and datasets for duplicate detection and consists of several components with clear interfaces that can be easily served with individual code. In this paper, we present the DuDe architecture and its workflow for duplicate detection. We show that DuDe allows to easily compare different algorithms and similarity measures, which is an important step towards a duplicate detection benchmark.

1. DUPLICATE DETECTION FRAMEWORKS

The basic problem of duplicate detection has been studied under various names, such as entity matching, record linkage, merge/purge or record reconciliation. Given a set of entities, the goal is to identify the represented set of distinct real-world entities. Proposed algorithms in the area of duplicate detection aim to improve the efficiency or the effectiveness of the duplicate detection process. The goal of efficiency is usually to reduce the number of pairwise comparisons. In a naive approach this is quadratic in the number of records. By making intelligent guesses which records have a high probability of representing the same real-world entity, the search space is reduced with the drawback that some duplicates might be missed. Effectiveness, on the other hand, aims at classifying pairs of records accurately as duplicate or non-duplicate [17].

Elmagarmid et al. have compiled a survey of existing algorithms and techniques for duplicate detection [11]. Köpcke and Rahm give a comprehensive overview about existing duplicate detection *frameworks* [15]. They compare eleven frameworks and distinguish between frameworks without training (BN [16], MOMA [24], SERF [1]), training-based frameworks (Active Atlas [22], [23], MARLIN [2, 3], Multiple Classifier System [27], Operator Trees [4]) and hybrid frameworks (TAILOR [10], FEBRL [6], STEM [14], Context Based Framework [5]). Not included in the overview is STRINGER [12], which deals with approximate string matching in large data sources. Köpcke and Rahm use several comparison criteria, such as supported entity types (e.g. relational entities, XML), availability of partitioning methods to reduce the search space, used matchers to determine whether two entities are similar enough to represent the same real-world entity, the ability to combine several matchers, and, where necessary, the selection of training data.

In their summary, Köpcke and Rahm criticize that the frameworks use diverse methodologies, measures, and test problems for evaluation and that it is therefore difficult to assess the efficiency and effectiveness of each single system. They argue that standardized entity matching benchmarks are needed and that researchers should provide prototype implementations and test data with their algorithms. This agrees with Neiling et al. [18], where desired properties of a test framework for object identification solutions are discussed. Moreover, Weis et al. [25] argue for a duplicate detection benchmark. Both papers see the necessity for standardized data from real-world or artificial datasets, which must also contain information about the real-world pairs. Additionally, clearly defined quality criteria with a description of their computation, and a detailed specification of the test procedure are required. An overview of quality and complexity measures for data linkage and deduplication can be found in Christen and Goiser [7]

With DuDe, we provide a toolkit for duplicate detection that can easily be extended by new algorithms and components. Conducted experiments are comprehensible and can be compared with former ones. Additionally, several algorithms, similarity measures, and datasets with gold standards are provided, which is a requirement for a duplicate detection benchmark. DuDe and several datasets are available for download at `http://www.hpi.uni-potsdam.de/naumann/projekte/dude.html`.

VLDB '10, September 13-17, 2010, Singapore
Copyright 2010 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

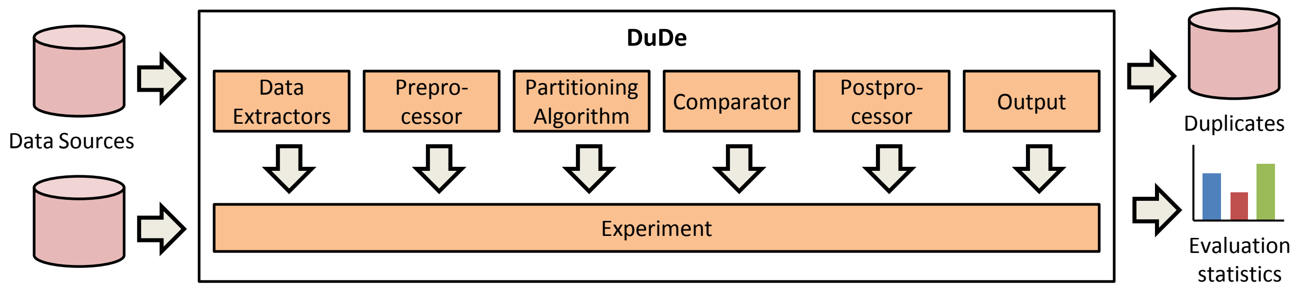


Figure 1: DuDe architecture

Our contributions are:

- A modular duplicate detection toolkit that can easily be extended
- A set of datasets including their gold standard and detailed descriptions to be used as benchmark

In the following Sec. 2 we give an overview of the DuDe architecture and its main components. In Sec. 3, we explain the data flow of an example experiment and demonstrate how easy it is to configure DuDe. Sec. 4 gives an overview of the datasets that are provided with DuDe and might be used for duplicate detection benchmarking. Two of these datasets are then used in Sec. 5, in which compare the performance of DuDe and the record linkage framework Febrl. Finally, we conclude and discuss our future work in Sec. 6.

2. DUDE ARCHITECTURE

In this section, we describe the architecture of the DuDe toolkit and its components. The goal of DuDe is to provide a toolkit for duplicate detection that

- is easy to use,
- is easy to extend,
- supports a large variety of data sources, including nested data,
- allows almost all algorithms to be implemented using the toolkit, and
- provides several basic similarity measures.

Figure 1 gives an overview of the different components used within the DuDe toolkit to conduct experiments. The framework is implemented in Java, which makes it easy to extend. The internal data format for processing records is based on JSON[1] which is a language-independent data-interchange format. In the following subsections, we describe the different components in more detail.

2.1 Data Extractors

The data extractor component is used to extract data from any data source that is supported by the toolkit and to convert the data into the internal JSON format. Currently, we are able to extract records from relational databases (Oracle, DB2, MySQL and PostgreSQL), CSV files, XML documents, and BibTeX bibliographies. For each data extractor, a record identifier, consisting of one or many attributes, can be defined and additionally a global ID is assigned to each data extractor, which is also saved within the extracted records. This allows a comparison of records from different sources without the necessity of an extractor-wide unique identifier.

[1]JavaScript Object Notation, `http://www.json.org`

2.2 Preprocessor

The preprocessor is used to gather statistics while extracting the data, e.g., counting the number of records or (distinct) values. After the extraction phase, each preprocessor instance is accessible within the algorithm and might be used within comparators that need preprocessing information.

2.3 Partitioning Algorithms

Partitioning algorithms are responsible for selecting pairs of records from the extractors that should be classified as duplicate or non-duplicate. In general, DuDe supports all algorithms that follow a pair-wise comparison pattern and it already provides a growing selection of such algorithms. An implemented naive approach to be used as a baseline simply generates all possible pairs of objects that are stored within the data source(s). Each pair is returned only once. So if (a, b) is already returned, (b, a) is not. Most algorithms require some kind of preprocessing, such as sorting for the Sorted Neighborhood Method [13] or partitioning for the Blocking Method [9]. Therefore, each algorithm can execute a preprocessing step before returning record pairs. In case of sorting, DuDe allows the definition of a sorting key. A sorting key collects a list of different subkeys which specify attributes or part of attribute values. The sorting can be executed by an in-memory (for small datasets) or by a file-based sorter.

2.4 Comparators

Comparators are used to compare two records and calculate a similarity. The similarity is a value between 0 and 1, with 1 defined as equality. We distinguish between three types of comparators:

- **Structure-based comparators** can be used to compare objects based on their structure. This is especially interesting if records from different sources with different schemas are compared (e.g. calculating the similarity based on the number of equal attributes in the record schemas).

- **Content-based comparators** can be used to compare objects based on concrete attribute values. Right now, we have implemented 19 content-based comparators, most of them using the publicly available library SimMetrics.[2] Examples are Levenshtein distance, Jaro Winkler distance, Smith-Waterman distance, SoundEx, and identity comparators.

- **Multi-comparators** can be used to combine different structure- or content-based comparators. This means that they receive the similarity values of several comparators as input and calculate a combined similarity. At present, multi-comparators for the calculation of the minimum or maximum value, the weighted average, and the harmonic mean are implemented. Multi-comparators can also be nested.

Comparators are used only for calculating the similarity of a candidate pair, but not for finally classifying whether the candidate pair represents the same real-world entity. For the classification, additional thresholds have to be defined within the duplicate detection experiment. Depending on whether the similarity is greater or less than the threshold, the candidate pairs are then forwarded to a postprocessor or an output.

2.5 Postprocessor

The postprocessor receives the classified record pairs and performs additional processing: Two important postprocessors are the transitive closure generator and the statistic component. The former calculates the transitive closure for all classified duplicates. The latter allows the calculation of key performance indicators, such as runtime, number of generated record pairs, reduction ratio, and the number of classified duplicates. If a gold standard exists for a dataset, additionally precision, recall, f-measure, etc. are calculated.

2.6 Output

There are several output formats for the record pairs, including a simple text output that writes each result pair into one line using a specified separator for the attribute values, and a JSON output, which uses JSON syntax. The CSV output allows the output of additional information for record pairs, such as the calculated similarity or whether the pair has been classified as duplicate or not. The statistic component has its own CSV output, which also allows the specification of additional attributes. These attributes can be used to describe the configuration of an experiment. All outputs can be written to the screen or into a file. The offered output components can easily be extended to meet experiment specific requirements.

3. EXPERIMENT CONFIGURATION AND DATA FLOW

In this section, we explain a typical experiment configuration. In our example, we deduplicate audio CD information (e.g., artist, title, tracks) in a CSV-file and use the Sorted Neighborhood Method [13] to search for duplicates. Figure 2 shows the data flow for our configuration.

For each experiment, we create a new Java class. First, we configure the data extractor (see Listing 1). As the data

[2]http://www.dcs.shef.ac.uk/~sam/simmetrics.html

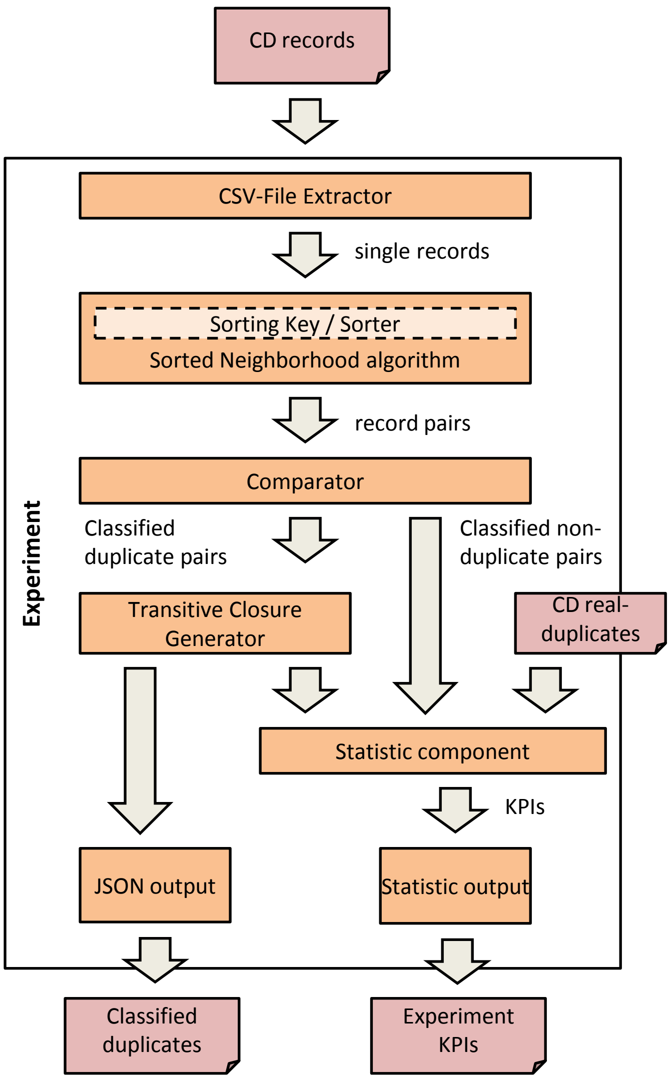


Figure 2: DuDe example experiment workflow

is stored in a file, we need only an identifier for the extractor and the file name. Then we set the separator character for the attributes and enable the header function. The attribute names are then read from the first row in the file. Afterwards, we define attribute "pk" as the unique identifier.

```
// configuration CSV extractor
extractor = new CSVExtractor("cd_records",
  new File("./res/cd_records.csv"));
extractor.setSeparatorCharacter(';');
extractor.enableHeader();
extractor.addIdAttributes("pk");
```

Listing 1: Configuration of the data extractor

As our data extractor extracts single records, we need an algorithm that creates record pairs for comparison. In this example, we use the Sorted Neighborhood Method, which needs sorted records. Therefore, we create a sorting key, which is the combination of one or several subkeys (attributes “artist” and “category”, see Listing 2). Then the algorithm is instantiated with a window size of 20 and in-memory processing is selected, which improves performance, but can be used only for smaller datasets.

```
// defines sub−keys used to generate the sorting key
artistSubkey = new TextBasedSubkey("artist");
titleSubkey = new TextBasedSubkey("title");
// key generator uses sub−key selectors (order matters)
sortKey = new SortingKey();
sortKey.addSubkey(artistSubkey);
sortKey.addSubkey(titleSubkey);

// new SNM algorithm with window size 20 and
// in−memory processing enabled
algorithm = new SortedNeighborhoodMethod(sortKey, 20);
algorithm.enableInMemoryProcessing();
algorithm.addDataExtractor(extractor);
```

Listing 2: Configuration of the SNM algorithm

The algorithm returns record pairs to be classified by our weighted average comparator that uses the Levenshtein distance for the disc title and SoundEx for the artist name (see Listing 3). Weighting is possible, but not used in our example.

```
levComp = new LevenshteinDistanceComparator("title");
sndComp = new SoundExComparator("artist");
comparator =
  new WeightedAverageComparator(levComp, sndComp);
```

Listing 3: Configuration of the comparator

For classified duplicate pairs, we configure a postprocessor, which calculates the transitive closure and an output that writes the finally classified duplicate pairs in a file (see Listing 4).

```
// transitive closure generator
transClosure = new NaiveTransitiveClosureGenerator();

// output for classified duplicates
jsonOutput = new JsonOutput(
  new FileOutputStream("./res/duplicates.json"));
```

Listing 4: Configuration of the transitive closure generator and output for classified duplicates

To assess the quality of the duplicate detection process, we instantiate a statistic component. This component first reads the gold standard from a file. Both the classified duplicate pairs from the transitive closure and the non-duplicate pairs from the comparator are then added to the statistic component and compared with the gold standard. The non-duplicate pairs are needed, because the statistic component additionally counts true-negatives and false-negatives only for the actual classified pairs to get an additional measure of the comparator quality. At the end of the experiment, the statistic component calculates several key figures including runtime, precision, and recall. By using the statistic output, these key figures can then be written to a CSV-file. Such a file can contain results from several experiments. Listing 5 shows the configuration of the experiment data flow, and Figure 3 shows an extract from the resulting statistics file.

```
double threshold = 0.99;
statisticComponent.setBeginningTime();

for (DuDeObjectPair pair : algorithm) {
  if (comparator.compareObjects(pair) > threshold) {
    transClosure.add(pair);
  } else {
    statisticComponent.addNonDuplicate(pair);
  }
}

// read all classified records including those
// from the transitive closure
for (DuDeObjectPair pair : transClosure) {
  statisticComponent.addDuplicate(pair);
  jsonOutput.write(pair);
}

statisticComponent.setFinishingTime();
statisticOutput. writeStatistics ();
```

Listing 5: Experiment execution

Please note that our algorithm does not select all duplicate pairs at once, but rather selects a candidate pair that is pipelined to the comparator and then classified before the next candidate pair is selected. Therefore, it is possible that algorithms get notified whether the former pairs have been classified as duplicate or non-duplicate, before the algorithm selects the next pair. This is necessary for algorithms that select pairs in dependency to previous classifications.

4. DATASETS

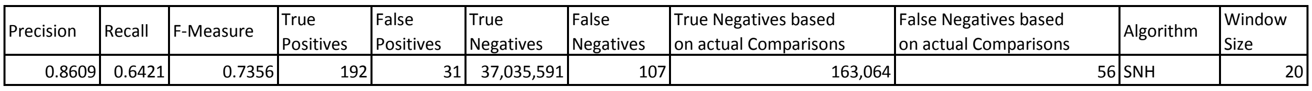
To develop a duplicate detection benchmark, it is necessary to make generally accepted datasets available. There is no single dataset that is commonly used for benchmarking in the duplicate detection community; rather, there is a variety of more or less useful datasets that have been used to evaluate algorithms.

Real-world data is preferable over synthetic data, as it is difficult to simulate all types of errors that might occur during data entry or data processing. On the other hand, legal regulations or privacy concerns often prevent data exchange between organizations and the scientific community.

We have prepared three real-world datasets that have been used in several papers. Required transformation steps for loading these datasets in DuDe are documented on our website[3] along with example code for the extractors. As frequently described, some massaging of the data was necessary. To make this process transparent, we diligently describe the individual corrections and customizations on the download page.

To evaluate the results of an experiment, we additionally offer a gold standard for each of the datasets. The gold standard was created manually in an arduous process involving distributed manual checking and cross-checking of all candidate pairs. Table 1 gives an overview of the number of records and duplicates in the datasets.

[3]`http://www.hpi.uni-potsdam.de/naumann/projekte/dude.html`



Precision	Recall	F-Measure	True Positives	False Positives	True Negatives	False Negatives	True Negatives based on actual Comparisons	False Negatives based on actual Comparisons	Algorithm	Window Size
0.8609	0.6421	0.7356	192	31	37,035,591	107	163,064	56	SNH	20

Figure 3: Extract from statistics file formatted in MS Excel

The **Restaurant data**[4] were extracted from the RIDDLE repository, which is a valuable source for datasets. The dataset comprises names and addresses of restaurants. For DuDe we have converted the file format from ARFF into CSV. The gold standard was extracted from the included attribute "class". As for the CORA dataset, we have added a unique identifier for each record to be able to easily represent duplicate pairs.

The **CD dataset**[5] is a randomly selected extract from freeDB[6]. It contains information about CDs including artist, title, and songs and has been used in several papers [16, 9].

Finally, the **CORA Citation Matching** dataset[7] lists groups of differently represented references to the same paper and is used in several approaches to evaluate duplicate detection [2, 8, 21]. A disadvantage of this dataset is the missing unique identifier for each record: A deeper look at the records revealed that the reference ID (the BibTeX key) is unfortunately not always faultless. In particular, we discovered two problems: two references have the same reference ID, but do not represent the same paper (see Listing 6). And vice versa, there are references that in fact represent the same paper but have different reference IDs (see Listing 7). To make the CORA dataset readable for the DuDe toolkit, we have transformed the three original files into one XML document. Therefore, it was necessary to make minor changes within the new file, e.g., adding closing tags for the references or repairing broken tags. Additionally, we have added a unique identifier for each record, which is a prerequisite to define a gold standard.

```
<NEWREFERENCE id="968"> 438 aha1991
  <author>D.W. Aha, D. Kibler and M.K. Albert</author>
  <year>(1991)</year>
  <title>. Instance-Based Learning Algorithms.</title>
  <journal>Machine Learning,</journal>
  <volume>6</volume>
  <pages>37-66.</pages>
</NEWREFERENCE>
<NEWREFERENCE id="969"> 439 aha1991
  <author>D. Aha and D. Kibler.</author>
  <title>Noise-tolerant instance-based learning
        algorithms.</title>
  <journal>Machine Learning,</journal>
  <volume>8:</volume>
  <pages>794-799,</pages>
  <year>1991.</year>
</NEWREFERENCE>
```

Listing 6: CORA example of two different papers with same reference ID "aha1991"

Dataset	Format	Records	Duplicate pairs
Restaurant	CSV	864	112
CD data	CSV	9,763	299
CORA	XML	1,879	64,386

Table 1: Overview datasets

```
<NEWREFERENCE id="1034"> 504 pazzani1992
  <author>Michael Pazzani and Dennis Kibler.</author>
  <title>The role of prior knowledge in inductive
        learning.</title>
  <journal>Machine Learning,</journal>
  <volume>9</volume>
  <pages>57-94,</pages>
  <year>1992.</year>
</NEWREFERENCE>
<NEWREFERENCE id="1035"> 505 kibler1992
  <author>Michael Pazzani and Dennis Kibler.</author>
  <title>The role of prior knowledge in inductive
        learning.</title>
  <journal>Machine Learning,</journal>
  <volume>9</volume>
  <pages>57-94,</pages>
  <year>1992.</year>
</NEWREFERENCE>
```

Listing 7: CORA example of two different reference IDs for the same paper

Beside these real-world datasets, we are planning to create larger datasets using data generators. Possible data generators are the UIS Database Generator[8], the Febrl generator[9], and the Dirty XML Generator[10]. The problem of creating a realistic synthetic dataset of personal information is discussed in [20].

5. EXPERIMENTAL COMPARISON

As mentioned in Sec. 1, a variety of duplicate detection frameworks exists and they are compared in detail in [15]. In this section we want to compare DuDe and Febrl [6] both experimentally and qualitatively. Table 2 summarizes our findings.

Therefore, we conducted two experiments. The first uses the naive approach for the restaurant dataset from Sec. 4 and applies the Levenshstein distance for the restaurant name as comparison function. The second experiment uses the CD dataset for the Sorted Neighborhood method with a window size of 20 and the same settings as described for the example experiment in Sec. 3. As we are mainly interested in comparing the runtime performance of both frameworks, a sophisticated similarity function is not necessary, as long

[4]Originally from `http://www.cs.utexas.edu/users/ml/riddle/`

[5]Originally from `http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets`

[6]`http://www.freedb.org`

[7]Originally from `http://www.cs.umass.edu/~mccallum/code-data.html`

[8]`http://userweb.cs.utexas.edu/users/ml/riddle/data/dbgen.tar.gz`

[9]`http://sourceforge.net/projects/febrl/`

[10]`http://www.hpi.uni-potsdam.de/naumann/projekte/completed_projects/dirtyxml.html`

as both use the same settings. For both experiments, DuDe uses in-memory processing.

The results of our comparison are shown in Tab. 2. The table shows if the frameworks are publicly available, in which programming language they were developed, if they support the calculation of key indicators, such as precision and recall, the supported data formats, and the execution time for the two experiments. As we can see, DuDe performs better for both experiments. Please note that for Febrl we used the algorithm that is closest to the Sorted Neighborhood method, namely the SortingIndex algorithm. To ensure the same set of comparisons, we have therefore added the primary key at the end of the sorting key to ensure unique values.

	DuDe	**Febrl**
Publicly available	Yes	Yes
Extensible	Yes	Yes
Programming language	Java	Python
Graphical user interface	No	Yes
Key figure calculation	Yes	Yes
Supported data formats	Text file, XML, database, BibTeX, JSON	Text file
Restaurant experiment (372,816 comparisons)	6.5 sec	95 sec
CD experiment (185,307 comparisons)	21 sec	78 sec

Table 2: Framework comparison

We did not describe precision and recall, since a major difference between DuDe and Febrl is the calculation of key figures. In Febrl, they are calculated based only on all compared record pairs. In contrast, DuDe uses *all* candidate pairs (as in the naive approach). An example for this different behavior is the calculation of the recall value for an experiment using the Sorted Neighborhood method. Due to erroneous attribute values, it is possible that the records of a real duplicate pair are not in the same window and consequently are not compared. While DuDe considers this to be a false-negative, Febrl does not. As false-negatives reduce the recall, DuDe achieves lower recall values than Febrl. Therefore, experiment results from Febrl and DuDe are only comparable to a limited extent.

6. CONCLUSION AND FUTURE WORK

With DuDe, we are developing a new duplicate detection toolkit. Due to the component-based architecture, it is possible for researchers to implement their new ideas and compare them with existing approaches. Additionally, DuDe offers several real-world datasets and respective gold standards. Therefore, DuDe is suitable for duplicate detection benchmarking.

The currently available toolkit offers the basic framework to execute duplicate detection experiments on different data sources with an acceptable performance. Our focus for the next months is to increase the number of available data extractors, algorithms, comparators, and outputs. In parallel, we will expand the number of available How-To guides, which serve as starting points for new users to develop their own components.

Regarding the classification, whether a pair of records represents the same real-world object or not, we plan to extend our toolkit in a way that not only a single similarity function can be used, but also a set of rules can be applied. This approach is described in [26].

As it is our goal to provide a toolkit that can be used for duplicate detection benchmarking, we will also publish results of our experiments on our website and try to extend the number of available datasets. We encourage the data cleansing community to send us new datasets or newly developed components and the results of their experiments.

7. ACKNOWLEDGMENTS

The authors would like to thank Fabian Lindenberg, Matthias Pohl, and Ziawasch Abedjan who have made a great effort to implement parts of the framework.

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

8. REFERENCES

[1] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.

[2] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 39–48, 2003.

[3] Mikhail Bilenko and Raymond J. 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.

[4] Surajit Chaudhuri, Bee-Chung Chen, Venkatesh Ganti, and Raghav Kaushik. Example-driven design of efficient record matching queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 327–338. VLDB Endowment, 2007.

[5] Zhaoqi Chen, Dmitri V. Kalashnikov, and Sharad Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 207–218, 2009.

[6] Peter Christen. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management*, pages 17–25, 2008.

[7] Peter Christen and Karl Goiser. Quality and complexity measures for data linkage and deduplication. In Fabrice Guillet and Howard J. Hamilton, editors, *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 127–151. Springer, 2007.

[8] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 85–96, 2005.

[9] Uwe Draisbach and Felix Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *Proceedings of*

the International Workshop on Quality in Databases (QDB), pages 51–56, 2009.

[10] Mohamed Elfeky, Vassilios Verykios, and Ahmed Elmagarmid. Tailor: A record linkage tool box. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 17, 2002.

[11] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[12] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293, 2009.

[13] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 127–138. ACM Press, 1995.

[14] H. Köpcke and E. Rahm. Training selection for tuning entity matching. In *Proceedings of the Sixth International Workshop on Quality in Databases and Management of Uncertain Data (QDB/MUD '08)*, pages 3–12, 2008.

[15] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data and Knowledge Engineering (DKE)*, 69(2):197–210, 2010.

[16] Luís Leitão, Pável Calado, and Melanie Weis. Structure-based inference of XML similarity for fuzzy duplicate detection. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 293–302, 2007.

[17] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection (Synthesis Lectures on Data Management)*. Morgan and Claypool Publishers, 2010.

[18] Mattis Neiling, Steffen Jurk, Hans-J. Lenz, and Felix Naumann. Object identification quality. In *in Proceedings of the International Workshop on Data Quality in Cooperative Information Systsems (DQCIS*, pages 187–198, 2003.

[19] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, 1959.

[20] Agus Pudjijono and Peter Christen. Accurate synthetic generation of realistic personal information. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 2009.

[21] Parag Singla and Pedro Domingos. Object identification with attribute-mediated dependences. In Alípio Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, volume 3721 of *Lecture Notes in Computer Science*, pages 297–308. Springer, 2005.

[22] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[23] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 350–359, 2002.

[24] Andreas Thor and Erhard Rahm. Moma - a mapping-based object matching system. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 247–258. www.crdrdb.org, 2007.

[25] Melanie Weis, Felix Naumann, and Franziska Brosy. A duplicate detection benchmark for XML (and relational) data. In *Proceedings of the SIGMOD International Workshop on Information Quality for Information Systems (IQIS)*, 2006.

[26] Melanie Weis, Felix Naumann, Ulrich Jehle, Jens Lufter, and Holger Schuster. Industry-scale duplicate detection. *Proceedings of the VLDB Endowment*, 1(2):1253–1264, 2008.

[27] Huimin Zhao and Sudha Ram. Entity identification for heterogeneous database integration: a multiple classifier system approach and empirical evaluation. *Information Systems*, 30(2):119–132, 2005.