Semantic Web 0 (2018) 1
IOS Press

EMBench^{++}: Data for a Thorough Benchmarking of Matching-Related Methods

Homepage *→ https://db.disi.unitn.eu/pages/EMBench/*

Ekaterini Ioannou [a] and Yannis Velegrakis [b]
[a] *Open University of Cyprus. E-mail: ekaterini.ioannou@ouc.ac.cy*
[b] *University of Trento. E-mail: velgias@disi.unitn.eu*

Editors: Axel-Cyrille Ngonga Ngomo, Institute for Applied Informatics, Leipzig, Germany; Irini Fundulaki, ICS-FORTH, Heraklion Greece; Anastasia Krithara, National Center for Scientific Research "Demokritos", Athens, Greece
Solicited reviews: Matteo Palmonari, University of Milan - Bicocca, Milan, Italy; Michelle Cheatham, Wright State University, USA; Matthias Wauer, Universität Leipzig, Leipzig, Germany; One anonymous reviewer

Abstract. Matching-related methods, i.e., entity resolution, entity search, or detecting evolution of entities, are essential parts in a variety of applications. The specific research area contains a plethora of methods focusing on efficiently and effectively detecting whether two different pieces of information describe the same real world object or, in the case of entity search and evolution, retrieving the entities of a given collection that best match the user's description. A primary limitation of the particular research area is the lack of a widely accepted benchmark for performing extensive experimental evaluation of the proposed methods, including not only the accuracy of results but also scalability as well as performance given different data characteristics.

This paper introduces EMBench^{++}, a principled system that can be used for generating benchmark data for the extensive evaluation of matching-related methods. Our tool is a continuation of a previous system, with the primary contributions including: modifiers that consider not only individual entity types but all available types according to the overall schema; techniques supporting the evolution of entities; and mechanisms for controlling the generation of not single data sets but collections of data sets. We also illustrate collections of entity sets generated by EMBench^{++} and discuss the benefits of using our system through the results of an experimental evaluation.

Keywords: Data integration, Matching-Related Methods, Benchmarking Data, Benchmark Tool

1. Introduction

Entity Matching is the task of efficiently and effectively detecting whether two different pieces of information describe the same real world object, such as a conference, a person, or a publication [1]. Strongly related to entity matching are the tasks of (i) *entity search*, focusing on efficiently and effectively retrieving the entities of a given collection that best match the user's description [2, 3], and (ii) *evolution of entities*, aiming at detecting entities that describe the same real world object even if their information is different due to evolution over time [4–7].

Matching-related methods are typically part of data integration or cleaning components that are considered essential in a variety of applications. The research community has already introduced a great deal of matching-related methods. These methods have been discussed in related surveys, such as [8], [9], [1], [10], and [11]. A major limitation of the existing works in the particular research area is the lack of a widely accepted benchmark for performing extensive experimental evaluation of the proposed methods, including not only the accuracy of results but also the scalability and the performance for different data characteristics.

In our previous work, we had developed EMBench [12, 13], a system for benchmarking matching-related methods in a generic, complete, and principled way. The system is based on a series of test cases aiming at capturing the majority of the matching situations. EMBench [13] is fully configurable with users being able to define the desired entities with their attributes as well as the modifications and modification level that will be incorporated in the entities. The system also provides an on-the-fly generation of the different test cases in terms of different sizes and complexities both at the schema and at the instance level.

Currently, we are witnessing research efforts for dealing with a number of new challenges. One such challenge is volatility. As explained elsewhere [14, 15], applications, especially Web 2.0 applications, focus on enabling and encouraging users to constantly contribute and modify existing content. An analysis of different versions of DBPedia revealed that users modify the entity data, and only a small fraction of the entity data remain unchanged. Volatility is the result of many reasons like change on the requirements, on the topics of interest, performance reasons, or even semantic evolution that requires entity merging or splitting [6, 16].

This publication introduces EMBench^{++}, an extension of our previous system with additional mechanisms aiming at generating benchmark data for the evaluation of matching-related methodologies. Primarily, these extensions include the generation of entities with relationships and entities those data have evolved. EMBench^{++} allows thorough experimental evaluations by enabling the assessment of a plethora of aspects that influence quality and performance. Each aspect can be investigated following various scenarios and different assumptions, produced in a controlled and consistent manner. The main contributions we are making here is the set of extensions and new services in our entity matching benchmark. In particular,

– We introduce modification mechanisms on existing data sets that consider not only individual entities but also sets of entities determined by their schema information.
– We provide mechanisms for generating collections of data sets that are able to capture and evaluate specific matching-related aspects.
– We provide mechanisms that allow the users to generate sequences of data sets, with the entities of each data set being the evolved version of the entities from the preceding one.
– We illustrate the abilities of our system by generating collections with appropriate data sets for the thorough experimental evaluation of matching-related methodologies.

2. Related Work & Open Challenges

Existing related works can be separated into two main categories: those related to the generation of synthetic data for matching-related approaches (Section 2.1) and those related to approaches for performing entity matching (Section 2.2). In each category there are a number of open challenges.

2.1. Synthetic Data Generation

In 2004 the Ontology Alignment Evaluation Initiative (OAEI) started working on the controlled experimental evaluation of alignment and matching systems. With respect to EMBench^{++}, the most interesting task is *instance matching* for which, currently, OAEI provides real and synthetic data [17].

Real data are static collections, typically of a much larger size than the synthetic ones. Such collections are extracted from real applications and reflect the matching problems that must be addressed. Thus, real data contain the possible, independent occurrences of the challenges that the matching-related techniques must handle (e.g., heterogeneities, schema absence, etc.) as well as situations in which such challenges appear in combination. Furthermore, we must always keep in mind that such real systems evolve, which means that additional data challenges can appear. Thus, regularly monitoring and extracting data from real applications (e.g., once per month) can assist in detecting such data challenges.

Given the reasons presented above, it is clear that real data should be the first source for the experimental evaluation of matching-related techniques. However, there are some aspects of real data collections that limit their usability. The first limitation is that that ground truth is typically not fully correct. As an example consider the DBLP system that lists publications from researchers. For researchers with common, or even similar names, the system has difficulties separating them[1]. Obviously, computing the quality of a matching technique cannot be accepted as being fully correct when using data with issues related to the ground truth. Another limitation of using real data is the lack of collections focusing on a particular challenge or challenges.

[1]E.g.: http://dblp.uni-trier.de/pers/hd/c/Chen:Lin

For instance, one matching technique might focus on addressing the lack of schema and thus evaluating the technique over data that also include heterogeneities in the values could be considered unfair for the particular technique.

Synthetic data is included in OAEI using the IS-Lab Instance Matching Benchmark [18]. The particular benchmark, includes entities from the OKKAM project [2, 3]. These entities are then modified using: (i) value transformations, such as typographical errors; (ii) structural transformations, such as value deletions; (iii) logical transformations, such as creation of two entities for the same real world object; and (iv) combinations of various transformations.

Synthetic generation of benchmarking entity matching data is also possible with the SWING system [19]. SWING design principles and goals are similar to EMBench but EMBench [12, 13] has more expressive power and offers more flexibility in the specification of the testing data. A detailed discussion is available in [13]. It compares the functionalities provided by EMBench with the ones of the SWING system, grouped according to (1) data acquisition, (2) data generation, and (3) matching scenarios.

One recent approach is LANCE [20], a generator that given a linked data set with its schema creates a new data set with matching tasks of various difficulties. The generator follows standard test cases related to structure and value transformations while also considering expressive OWL constructs. Another recent approach is SPIMBENCH [21] focusing on Semantic Publishing Domain. SPIMBENCH supports transformations of values, structure, semantics, as well as their combinations.

Challenges Related to Benchmarking Systems. It is clear that using synthetic and not static data allows users to control the generated data sets. This is in line with benchmarks in other data domains, such as TPC-H and STBenchmark [22], and stress test tools, such as Siege[2]. EMBench^{++} extends the options that users can control with the most advanced being the ability to control the “modifications” between the generated entity sets. In addition, EMBench^{++} provides mechanisms for generating volatile data, an aspect that existing benchmarking systems have not yet considered.

[2]http://www.joedog.org/siege-home/

2.2. Matching-related Methods

The research area of matching-related methods has been deeply investigated the last couple of decades and a plethora of methods have been suggested. The primary difference between the existing methods is what they consider as an entity representation and which information they use for performing the matching. We discuss these methods next. For ease of comprehension and discussion, we group them into categories according to the data included in the entity representation although there are many methods that span across more than one category.

A. Similarity Methods. The first category contains methods operating on entities that are either atomic string values or a set of string values. Here, we have various basic similarity techniques (see surveys [9] and [8]), such as Levenshtein distance [23], Jaro [24], Jaro-Winkler [25] and TF/IDF similarity [26]. Note that [27] and [8] describe and discuss an experimental comparison of various basic similarity techniques used for matching names. Merge-purge [28] is another method. It considers every database relation (i.e., record) as a representation and detects if relational records refer to the same real world object. Other methods focus on finding mappings between the representations using either *transformations* [29], such as abbreviation, stemming, and initials, or predefined rules [30] with knowledge about specific representations.

B. Collective Matching. The next category contains methods using collective matching, which means performing the matching using existing or discovered inner-relationships. A well-know method for this category is *Reference Reconciliation* [31]. The method first detects possible associations between the entities by comparing their corresponding attribute values. These associations are propagated to the rest of the entities in order to enrich their information and improve the quality of final matches. Other methods are [32, 33] that use entity inner-relationships to create a graph between entities. Graph nodes are clustered and detected clusters are used to identify the common entities. The methods from [34, 35] follow a similar methodology to create a graph. However, these methods also generate additional possible relationships to represent the candidate matches between entities. Matches between entities are discovered by analyzing the relationships in the graph.

C. Entity Evolution. The third category contains methods that deal with the volatile nature of the data. Handling volatility can be achieved by various mecha-

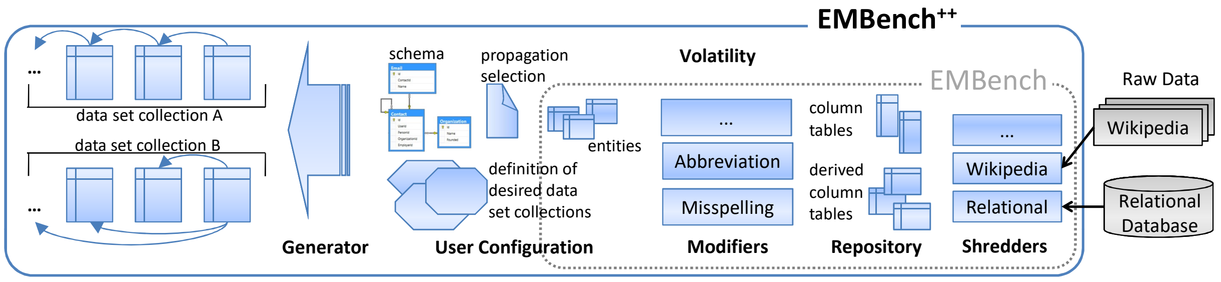


Fig. 1. An illustration of EMBench^{++}'s architecture (grey dotted line denotes the components from the previous version).

nisms. For instance, a portion of the introduced methods handle volatility through probabilities that model the belief related to the current resolution status of the entities [4–7]. More specifically, [4, 7] consider a small set of possible entity alternatives, with each alternative accompanied by a probability that indicates the belief we have that this reflects the correct entity. The approach in [5] addresses many challenges of heterogeneous data. It does not assume that the alternatives are known, but that an entity collection comes with a set of possible linkages between entities. Each linkage represents a possible match between two entities and is accompanied with a probability that indicates the belief we have that the specific representations are for the same real world object. Entities are compiled on-the-fly, by effectively processing the incoming query over representations and linkages, and thus, query answers reflect the most probable solution for the specific query.

Another example of methods aiming at handling volatility, focuses on using the newly arrived data to incrementally and efficiently update the detected entities. For this purpose, [36] focuses on maintaining the matches up-to-date with techniques that do not execute matching from scratch but exploit all previous matches. The approach in [37] considers clustering, i.e., each cluster corresponds to a specific entity. New data can be merged with existing clusters or can be used for correcting previous matching mistakes.

D. Blocking-based Methods. The last category includes blocking-based methods, focusing on processing data sets of large sizes. Instead of comparing each entity with all other entities, blocking-based methods separate entities into blocks, such that entities of the same block are more likely to be a match than entities from different blocks. Thus, only the entities of the same block are compared. The majority of the proposed methods typically associate each entity with a *Blocking Key Value* (BKV) summarizing the values of selected attributes and then operate exclusively based on the BKVs. One such example is [38]. It sorts blocks according to their BKV and then slides a window of fixed size over them, comparing the representations it contains. The most recent methods investigate building the blocks when having heterogeneous semi-structured data with loose schema binding, e.g., [39]. Among other, the authors introduce an attribute-agnostic mechanism for generating the blocks, and explain how efficiency can be improved by scheduling the order of block processing and identifying when to stop the processing. Iteratively block processing [40] provides a principled framework with message passing algorithms for generating a global solution for the resolution over the complete collection.

Challenges Related to Benchmarking Systems. The last three categories have not yet been targeted by benchmarking systems. Extensive evaluation of collective matching methods requires usage of schema information in order to incorporate relationships between the entities. Entity evolution requires generating evolving versions of entities, i.e., the system should be able to include modifications on the already generated entities with the modifications reflecting possible changes due to time. Scaling to a large number of entities is also important, especially for blocking-based methods that aim at processing huge collections. This implies being able to generate a huge number of data while also being able to alter specific aspects, such as the level of inconsistencies.

3. The architecture of EMBench^{++}

We now present the new architecture of EMBench^{++} and discuss the additions and extensions included from the previous version of the system, which was described in [13] and [12].

Overview. The goal is to generate benchmark data for the extensive evaluation of matching-related methods. EMBench^{++} imports data from external applications which are then used/recombined for creating collections of synthetic entities. These entities are then modified by incorporating a particular real world heterogeneity, e.g., abbreviation. The system maintains the gold standard between the modified and the original entities, which is then used for testing matching-related methods.

Figure 1 provides a graphical illustration of the current architecture. The rectangle with dotted grey line denotes the components that were incorporated in EMBench. The remaining components have been included in the system in order to achieve the goals introduced in Sections 1 and 2.

The system includes a set of *Shredders* that are responsible for shredding a given data source (e.g., Wikipedia data, XML files) it into a series of *Column Tables*. The current implementation contains general purpose shredders, such as relational databases and XML files, and shredders that are specifically designed for popular systems, such as Wikipedia and DBLP. Each Column Table contains distinct and clean atomic values of a particular type, for example first names, surnames, cities, and universities. This is achieved through mechanisms that focus on cleaning the repetitive, overlapping, and complementary information in the resulted column tables.

In addition, the system also uses rules that specify how the values of the column tables are to be combined together or modified and guide the creation of a new set of column tables. Data resulted from rules are stored in *Derived Column Tables*, and are actually used by the system in the same way as column tables. Our current implementation, supports an "identity function" rule meaning that the resulted derived table is identical to the column table without any modification. It also supports function rules that can be used to combine column tables and Strings. As an example, consider a derived column table for FullName. The rule for FullName represents the concatenation of values from FirstName with a space character and values from Surname (i.e., Column Table followed by a String and then another Colum Table). This is, for example, expressed in the system as 'FirstName + " " + Surname'.

EMBench^{++} also maintains a *Repository* that maintains internal data, including the *Column Tables*, *Derived Column Tables* as well as generated entities and data sets. Note that the system contains a default repository with a number of *Column Tables*, for example 1,2 million first names, 293,5 thousands surnames, 8,6 thousands universities and 22,5 thousands titles of journal articles.

The system also contains a set of *Entity Modifiers*. Each modifier is responsible for incorporating a particular type of heterogeneity in the specified entities. As explained in [12], EMBench contains implementations for a set of Entity Modifiers, including misspellings, word permutations, acronyms and abbreviations.

In the updated version of the system, i.e., EMBench^{++}, we have incorporated mechanisms for *Volatility*. In short, these mechanisms focus on heterogeneity that appears in entities due to time changes. The developed mechanisms for volatility are presented and discussed in Section 4.3.

User Configuration allows users to configure the parameters related to the generation of data. Primarily, this involves configuring the desired entities and data collections. With respect to the entities, users define the entity types to be generated by specifying the number of entities, the attributes of each entity and the source for the attribute values. The source is a (Derived) Column Table along with a distribution (i.e., normal or Zipf) or a random value within a given range.

In addition, EMBench^{++} has mechanisms that allow users to use a generated entity value as the source of entities, which basically means that the result will be not independent tables but a complete database with foreign keys among its tables. The details are introduced and discussed in Section 4.1.

EMBench^{++} does not only allow users to generate and apply modifiers over individual entities (as the previous version) but also allows generating collections that contain various data sets of entities. As we later describe (Section 4.2), users can specify a collection with a number of data sets. Each data set can contain a different set of Entity Modifiers or the same set but different levels of destruction. The system also provides different options, referred to as *propagation type*, for generating the data sets within the same collection. The mechanisms related to collection generation are described in Section 4.2.

4. Advanced Generation of Benchmark Data

The primary concern of EMBench^{++} is the generation of data that goes beyond individual entity types. In particular, we need the generated entity sets to capture all the aspects required for a complete and extensive evaluation of matching-related methods, which were

(a)

- Entity type **Person** contains the following data (attribute$\mapsto$source): **id**$\mapsto$*autoincrease*, **full_name**$\mapsto$*FullName*, **university**$\mapsto$*University*, **job_title**$\mapsto$*random*{PhD student, postdoc, lecturer, etc}.
- Entity type **Article** contains the following data (attribute$\mapsto$source): **id**$\mapsto$*autoincrease*, **author_1**$\mapsto$*Person*, ..., **author_10**$\mapsto$*Person*, **title**$\mapsto$*PaperTitle*, **journal**$\mapsto$*JournalName*, **year**$\mapsto$*random[1970-2016]*.

(b)

Person			
id	full_name	university	job_title
e_p1	Noela Kuglen	Goshen College	professor
e_p2	Nikoline Paccini	University of Dubrovnik	postdoc
e_p3	Adwin Bokhri	Columbus State University	professor
e_p3	Oliwer Ellert	Kwan Dong University	PhD student

(c)

Article							
id	author_1	author_2	author_3	author_4...10	title	journal	year
e_a1	e_p1	e_p2	e_p4	null	Social and Human Aspects of Software Engineering	Thomas Wolfe Review	2015
e_a2	e_p3	null	null	null	Road Detection in Panchromatic SPOT Satellite Images	Wire Journal International	2014
e_a3	e_p4	e_p2	null	null	Constraint Based Object State Modeling	Review of Metaphysics	2016

Fig. 2. (a) Definition for two entity types. (b) Data generated for Person entity type. (c) Data generated for Article entity type.

discussed in Section 2. Two important aspects are the existence of inner-relationships between entities (also referred to as correlations) and the incorporation of all possible heterogeneities. To formally incorporate these aspects in the entity sets of our system, we use a model that assumes the existence of an infinite set of entity identifiers $\mathcal{O}$, an infinite set of names $\mathcal{N}$ and an infinite set of atomic values $\mathcal{V}$.

Definition 1. *An **entity** is a tuple that consists of an identifier $o \in \mathcal{O}$ and a finite set of attribute name-value pairs $\langle n, v \rangle$, where $n \in \mathcal{N}$ and $v \in \mathcal{V} \cup \mathcal{O}$. The sequence of attribute names $\langle n_1, n_2, ..., n_k \rangle$ of an entity e is referred to as the **type** of the entity.*

*An **entity set**, denoted as Itname:$(n_1,...,n_k)$, corresponds to a set of entities $\{e_1,...,e_n\}$, where tname$\in \mathcal{N}$ and is referred to as the **name** of the entity set, and all the entities in the set provide data for (or part of the) attributes n_1, ..., n_k.* ■

Note that in the remaining text, we will only use *tname* and attribute values (i.e., n_1, ..., n_k) if these are needed for comprehension. Otherwise, we will denote a set of entities as I.

The fact that a value of an attribute in an entity can be an atomic value or the identifier of another entity, is the main mechanism that allows entities to relate to each other. In what follows, we will use notation $e_i.n_j$ to denote the value v_j corresponding to attribute n_j for entity e_i. For example, $e_1.full\_name$ in Figure 2 returns "Noela Kuglen".

EMBench^{++} includes a set of modifiers responsible for incorporating particular types of heterogeneity (Section 3). A modifier f_i is tuple $\langle i, \{\langle t,l \rangle\}\rangle$ where i is the identifier of the modifier and each pair t-l provides the level l of the particular configuration setting t. Our implementation follows this definition and supports modifiers with a number of configuration settings, i.e., set of t-l pairs. However, for simplicity, in the remaining paper we assume modifiers with only one pair and thus a modifier corresponds to tuple $\langle i, t, l \rangle$.

Modifiers included in the Previous Version. The previous version of the our system [12, 13]included a discussion of the requirements related to the evaluation of matching-related methods as well as the related necessity modifiers. More specifically, in the previous version we considered the following categories: (a) *syntactic variations*, with modifiers focusing on variations of the value or attribute, e.g., misspellings, word permutations, aliases, acronyms, initials, and abbreviations; and (b) *structural variations*, with modifiers focusing on differences of the attributes, e.g., usage of multiple attributes, missing values, underspecification, and overspecification.

The modifiers are not executed on the whole entity set but on a subset of it. Initially, the entity set I is separated into two sets: I^m that contains randomly selected entities and I^o that contains the remaining entities, i.e., I^o=$I \setminus I^m$. We create a new entity set I' by using I^o and I^m. The entities of I^o are directly included in the new entity set I' whereas the entities of I^m are first modified and then included in I'.

Definition 2. *A **modified entity set** is the entity set I_m resulted by a sequential execution of the modifiers f_1, $f_2, \ldots, f_m$ over entity set I, i.e., I_m=$f_m(\, f_{m-1}(\ldots f_1(I)))$.*

Each $f_k(I_{k-1})$ creates sets I^m_{k-1} and I^o_{k-1} such that I_{k-1}=$I^m_{k-1} \cup I^o_{k-1}$ and $|I^m_{k-1}|$=c with c being a constant. Then, $f_k(I_{k-1}) = I^o_{k-1} \cup \{\, f_k(e_i) \mid \forall\, e_i \in I^m_{k-1} \,\}$. ■

The constant c is given to the system for selecting the number of entities from I that should be modified, i.e., $c \leqslant |I|$. Our current implementation also accepts selecting the number of entities using a percentage over I, denoted as p, and computes c as $|I| \times p$.

Consider again a modified entity set, i.e., I_k=$f_{k-1}(I_{k-1})$. It is also a set of entities generated when modifier f_{k-1} is applied on the entities on I_{k-1}. This operation is denoted as $I_{k-1} \overset{f_{k-1}}{\rightarrow} I_k$. Since the result of a modifier is also an entity set, it can also be used as an input to another modifier. Thus, a modified entity set may be the result of a series of different modifiers applied on the

```
<attribute name="author"
           min-occurence="1" max-occurence="10" >
   <namespec>
      <factor type="constant">author</factor>
   </namespec>
   <valuespec>
      <factor type="value-set" distribution="zipf">
         Author
      </factor>
   </valuespec>
</attribute>
```

Fig. 3. Configuration of author attributes in the Person entity set.

original entity set (as given in Definition 2), i.e., $I \xrightarrow{f_1} I_1 \xrightarrow{f_2} I_2 \ldots \xrightarrow{f_{m-1}} I_{m-1} \xrightarrow{f_m} I_m$.

Definition 3. *Let I_m be a modified entity set for I and $e_m \in I_m$ be the modified entity of $e \in I$. An* ***entity matching pair*** *is then tuple $\langle I, e_m, e\rangle$ and it is said to be* ***successfully executed*** *by a matching-related approach if it returns the entity e as a response when provided as input the tuple $\langle I, e_m\rangle$, i.e., returns e as the best match of e_m in the entity set I.* ∎

Once EMBench^{++} generates the entity set I, as specified by the user, it executes the selected and configurated modifiers $f_1, f_2, \ldots, f_m$. The latter creates the modified entity set I_m. The two entity sets are then used for evaluating a matching-related approach by generating an *entity matching scenario*, i.e., a set of matching pairs $\langle I, e_m, e\rangle$ $\forall e \in I$ with e.id=e_m.id where $e_m \in I_m$. The evaluation of the matching-related approach is related to its ability to detect and return the best matches of the requested entities.

4.1. Foreign Key Relationships

The data models followed by RDF and relational databases support internal references (i.e., namely foreign keys in databases), which is considered as an essential aspect. The primary reason is that foreign keys model the real world relationships as references in the data. Also, they are especially useful for encoding cascading relationships, i.e., having multiple foreign keys in tables with each foreign key referring a different parent table. In addition, satisfying *referential integrity*, i.e., ensuring that foreign keys agree with the primary key that the foreign keys refer to, enforces data consistency. For being able to support foreign keys we have included special mechanisms in EMBench^{++}.

The first mechanism is in the configuration. In the previous version of our benchmarking system, users could define entity sets using attributes that are either Column Tables or Derived Column Tables. EMBench^{++} enhanced this part and also allows defining entity sets in which the values of the entity attributes are identifiers to entities, either of the same set (i.e., type) or to entities from other entity sets. In other words, entities can now have references to other entities.

The second mechanism for enabling usage of foreign key relationships is incorporated in the Generator. More specifically, the Generator uses the configuration of each Entity Type and of each of its included attributes to create the entities. The foreign keys mechanism is applied when the configuration detects that the values of a particular attribute are another Entity Type. In this case, the Generator does not include an actual value, i.e., from a (Derived) Column table, but identifiers from the specified Entity Type.

The selection of identifiers from the specified Entity Type can be also influenced by the users (through configuration). More specifically, users can choose between a random or Zipfian option (Figure 3). The random option will do a random selection among all the identifiers of the given Entity Type without repetitions. The Zipfian option will do the selection based on a Zipfian distribution of all identifiers of the given Entity Type. The latter implies that the majority of the selected identifiers would appear few times and only a small number of the select identifiers would appear many times.

4.2. Generating Collections

As discussed in Section 2.1, to have an an extensive evaluation we need to examine how the matching-related methods behave when modifying important data characteristics, such as the size of the data set or the destruction level of the modifiers. For example, with respect to the level of the modifiers, it would be beneficial to apply the modifiers with the different level on the original entity set. On the contrary, with respect to the size of the data, it would reasonable to start from the original entity set and keep incrementally including entities, thus each time we use the previously created entity set.

Definition 4. *Let F_a denote a set of modifiers, i.e., F_a={f_1, f_2, ...}. We allow the following operations:*
- *addition(F_a,f_j): $F_a = F_a \cup \{f_j\}$*
- *deletion(F_a,f_j): $F_a = F_a \setminus \{f_j\}$*
- *adjust(F_a,t,l,$\odot$): $\forall f_j \in F_a$ with f_j.type=t $\mapsto f_j$.level=f_j.level$\odot$l, where $\odot$={+,−}* ∎

C-A0	**C-A1**	**C-A2**
F_{a1}={}	F_{a2}=addition(F_{a1},⟨missp.,10%⟩)	F_{a3}=adjust(F_{a1},missp.,10%,+)
person	C-**A0**.person + misspelling 10%	C-**A0**.person + misspelling 20%
article	C-**A0**.article + misspelling 10%	C-**A0**.article + misspelling 20%

C-B0	**C-B1**	**C-B2**
F_{b1}={}	F_{b2}=addition(F_{b1},⟨volat.,10%⟩)	F_{b3}=adjust(F_{b2},volat.,10%,+)
person	C-**B0**.person + volatility 10%	C-**B1**.person + volatility 20%
article	C-**B0**.article + volatility 10%	C-**B1**.article + volatility 20%

Fig. 4. Independent and Sequential propagation.

```
<volatility source-template="Author" rounds=3>
   <class type="addition" attribute="fullname"
      source="Surname" separator="-; "></class>
   <class type="replacement" attribute="fullname"
      source="Surname"></class>
 <class type="replacement" attribute="university"
      source="University"></class>
   <class type="continuous" attribute="job_title"
      deviation="-1,+2"></class>
</volatility>
```

Fig. 5. Configuration of the volatility modifications.

As explained in Definition 2, a modified entity set results when we execute modifiers $f_1, f_2, \ldots, f_m$ over an entity set. We now use symbol F_a to denote a sequence of modifiers, i.e., F_a={$f_1, f_2, \ldots$} and examine the generation of collections that contain data sets on which we execute different modifier sequences.

As shown in Definition 4, a sequence of modifiers F_a can either be extended with another modifier (i.e., addition operation), condensed by removing one of the modifiers (i.e., deletion operation), or adjusted by altering properties of selected modifiers (i.e., adjust operation). The adjust operation is given a configuration setting t, the level l, and the activity $\odot$. The result is to alter all modifiers containing a configuration setting equal to t by setting the value of this configuration setting, i.e., the level for t becomes (level$\odot l$). If, for example, we have F_a={f_1} with f_1=⟨missp.,attr.-per.,10%⟩ and we apply adjust(F_a,attr.-per.,5%,+) then the f_1 in F_a becomes ⟨missp.,attr.-per.,15%⟩.

Definition 5. *Given an entity set I and a collection of modifier sets F_a, F_b, F_c, ..., then:*
(i) an ***independent propagation*** *results in sets*
I^a=$F_a(I)$, I^b=$F_b(I)$, I^c=$F_c(I)$, ...
(i) a ***sequential propagation*** *results in sets*
I^a=$F_a(I)$, I^b=$F_b(I^a)$, I^c=$F_c(I^b)$, ... ∎

According to the above definition, we consider collections as follows: starting by a modifier sequence F_a and a set of operators, we first apply the operators over F_a and generate $F_b, F_c, \ldots$. These modifier sequences are then executed over the entity sets, starting from the original entity set I, and considering the requested propagation.

Figure 4 shows two example collections, each with three data sets. The data sets of the first collection, i.e., C-A, include an increasing level of misspelling in their entity sets. The first data set (i.e., C-A0) has a zero level of misspelling, the next (i.e., C-A1) has 10%, and the third one (i.e., C-A2) has 20%. C-A is an independent propagation with both C-A1 and C-A2 generated from C-A0. Note that the requested modifiers and level are executed on all entity types of the data sets, in this situation on Person as well as Article. C-B0 is similar example with one volatility modifier. As shown, C-B is a sequential propagation with C-B1 generated from C-B0 and C-B2 generated from C-B1.

4.3. Volatility

Applications, especially Web 2.0 applications, focus on enabling and encouraging users to constantly contribute and to modify existing content. For example, an analysis of DBPedia [14, 15] revealed that the data describing the entities were modified in time, with only some of the data remaining the same. Changes affect not only values but might also involve entities splitting or being merged, a form of semantic evolution [16]. As discussed in [16], an entity can either evolve into another entity, split into several other entities, or merge into another entity.

EMBench^{++} provides mechanisms that allow users to generate sequential data sets, with the entities of each succeeding data set being the evolved version of the entities from the preceding data set. Volatility mechanisms are either value-level or attribute-level, as follows:

A. Value-level Mechanisms. The first group of mechanisms incorporate modifications in the entity values. Given an entity set $I(n_1,...,n_k)$={$e_1,...,e_n$}, and some attribute name n_i, the value-level mechanisms will modify the value corresponding to n_i in the entities, i.e., $e_1.n_i$, $e_2.n_i$, ..., $e_n.n_i$. The modification can be of three different kinds.

A.1) Replacement: that substitutes the value of the attribute $e_j.n_i$ with another value selected from a given (Derived) Column Table. As shown in the Figure 5, values of attribute "university" will be replaced with values from the University Column Table.

A.2) Continuous: that is used on attributes that take values from a restricted set, e.g., job_title takes values from {PhD student, postdoc, lecturer, ...}, and replaces the existing value with the next one in the sequence of the allowed values. It is also possible to change the

Person :: C-B0			
id	full_name	university	job_title
e_p1	Noela Kuglen	Goshen College	professor
e_p2	Nikoline Paccini	University of Dubrovnik	postdoc
e_p3	Adwin Bokhri	Columbus State University	professor
e_p3	Oliwer Ellert	Kwan Dong University	PhD student

Person :: C-B1			
id	full_name	university	job_title
e_p1	Noela Kuglen-Airta	Goshen College	professor
e_p2	Nikoline Paccini	Goshen College	lecturer
e_p3	Adwin Bokhri	University of Vermont	professor
e_p3	Oliwer Ellert	Presidency University	researcher

Person :: C-B2			
id	full_name	university	job_title
e_p1	Noela Kuglen-Airta	Goshen College	professor
e_p2	Nikoline Saro	Keele University	senior lect.
e_p3	Adwin Bokhri	University of Vermont	professor
e_p3	Oliwer Ellert	Presidency University	researcher

Fig. 6. A collection with data volatility.

selection mechanism and instead of selecting the one immediatelly after, select the one before or the one k positions after.

A.3) Addition: which maintains the existing value but adds to it another one from a specific (Derived) Column Table. The existing and new value are separated using a given character, for example "-" or " ".

Figure 6 illustrates a collection with evolving data sets following the configuration shown in Figure 5. Thus, the entities of C-B1 are the evolved version of the entities of C-B0, and the entities of C-B2 are the evolved version of the entities of C-B1. Consider again the configuration of the volatility modifications. It includes four modifications. The first is the addition of values from column table Surname to attribute fullname with "-" as the separator, e.g., the "Noela Kuglen" from C-B0 becomes "Noela Kuglen-Airta" in C-B1. The second modification is replacement of the value of fullname with a value from Surname, e.g., "Nikoline Paccini" from C-B1 appears as "Nikoline Saro" in C-B2. The third modification is similar to the second one. It involves the replacement of the value of university with a value from University, e.g., the university of entity with id "e_p2" is "University of Dubrovnik" in C-B0, changes to "Goshen College" in C-B1, and to "Keele University" in C-B2. The last modification is for the job_title attribute. This was originally an ordered list of values and now the modifications is for taking the value either to the value found one place before in the list or up to two places afterwards. In the figure, this is present in entity with id "e_p3" that was a "PhD student" in C-B0 and a "researcher" in C-B1.

B. Attribute-level Mechanisms. The second group contains mechanisms that operate on the attribute-level. Thus, for a given entity set $I(n_1,...,n_k)$={$e_1,...,e_n$} the result involves modifications on the attribute names as well as the reflection of these modification on the entities. More specifically, the attribute-level mechanisms can be:

B.1) Elimination: this removes selected attributes from the entity set and thus eliminates the corresponding values from all entities. For example, if a user removes attribute n_i then the system (i) will convert I into $I(n_1, ..., n_{i-1}, n_{i+1}, ..., n_k)$ and (ii) will remove n_i from all entities of I, i.e., $e_1.n_i$, $e_2.n_i$, ..., $e_n.n_i$.

B.2) Expansion: this includes additional attribute names in an entity set while also generating the values for these attributes in all the corresponding entities. Expansion of I with attribute names n_{k+1}, ..., n_{k+j}, implies that the system (i) will now use $I(n_1,...,n_k, n_{k+1}, ..., n_{k+j})$ and (ii) will generate values for these attributes for all entities, i.e., $e_1.n_{k+1}$, ..., $e_1.n_{k+j}$, ..., $e_n.n_{k+1}$, ..., $e_n.n_{k+j}$.

5. Empirical Evaluation

In this section we illustrate an empirical use and evaluation of the introduced benchmarking system. We aim at examining different aspects of EMBench^{++} and describe and report our different assessments.

More specifically, we start with an overview of the collections used in existing publications (i.e., static data) and discuss the advances offered by EMBench^{++} (Section 5.1). Next is an illustration of generated data collections focusing on collective resolution and entity evolution (Section 5.2). We then continue with a comparison between collections used in the literature with the data sets that EMBench^{++} can generate (Section 5.3). Finally, we provide an example illustration on how data collections generated by our system have been used for testing a real matching-related technique (Section 5.4).

5.1. Advances over Static Collections

The majority of data collections used in the literature are related to publications (i.e., include data of authors and citations) [31, 41–43] and only few collections include entities of other types [42, 44–46]. We follow this separation in our comparison. Table 1 focuses on one publication-related collections while Table 2 on collections of various entity types.

As Table 1 shows, most publication-related collections are of a small size. For instance, Cora contains

Publication-related Data Collections							
Dataset	**Types**	**Instances**	**Entities**	**Duplicates**	**Public**	**Evolution**	**Ground-truth**
Cora, e.g., [5, 31, 41]	citations	6.107	338	1-21	yes	no	included
DBLP/ACM, e.g., [42, 43]	citations	4.671	2.552	1-2	yes	no	included
CiteSeer, e.g., [41, 43]	citations	1.031	558	1-21	yes	no	included
Four PIMs, i.e., [31]	citations	total=103.435	total=9.989	avg. 10	no	no	manually
KDD Cup 2003, e.g., [40]	papers, authors	58.515 authors	29.555 papers & 13.092 authors	not given	yes	no	included
EMBench^{++}	citations, authors, affiliations, etc.	*	*	*	yes	yes	included

Table 1

Comparison with publication-related collections.

Collections of Various Types							
Dataset	**Types**	**Instances**	**Entities**	**Duplicates**	**Public**	**Evolution**	**Ground-truth**
Biz, i.e., [31]	business	5.000 each entity	87	avg. 5.000	not stated	artificial	included
IMDB/DBPedia, i.e., [42]	movies	50.797	22.863	0-2	yes	no	included
Amazon/Google, e.g., [42, 44, 45]	products	4.393	1.104	0-2	yes	no	included
Abt-Buy, e.g., [45]	products	2.173	1.097	0-2	yes	no	included
Wikipedia, DBpedia, e.g., [46]	various	5,48M[3]	5,48M	none	yes	yes	included
LinkedGeoData, e.g., [46]	location-related	1.073M[4]	1.073M	none	yes	no	to other systems
EMBench^{++}	various	*	*	*	yes	yes	generated

Table 2

Comparison with collections of various entity types.

6.107 citations and DBLP/ACM contains 4.671. The largest collections are the KDD Cup 2003 and the Four PIMs. Unfortunately, the latter is from personal data and not publicly available. Furthermore, all these data collections do not contain evolution data.

The situation is a little better with respect to collections containing different entity types (i.e., Table 2). Here we have collections of larger sizes, for example IMDB/DBPedia containing 50.797 movies and Wikipedia containing 5,48M entities. Although this is a positive aspect, there are other issues when evaluating algorithms using these collections. The first is that these collections have a low number of duplicates (i.e., from 0 to 2 instances per entity). This is because the collections were typically created by merging two sources. For example, in the IMDB/DBPedia collection we would have one instance from IMDB and one instance from DBPedia describing the same real world object. Wikipedia, which is among the largest collections, does not have any duplicates, i.e., we see only one instance per entity. Another issue with these collections is the absence of evolution data. The only exception is with Wikipedia, since one can use the previous versions of the collection.

As previously discussed, EMBench^{++} is able to alleviate the aspects of existing collections that put limits on the possible evaluations. These aspects are the capability of generating collections of large sizes, containing various entity types, including various number of duplicates, and capturing evolution.

5.2. *Illustration of Generated Data Collections*

5.2.1. *Collective Resolution*

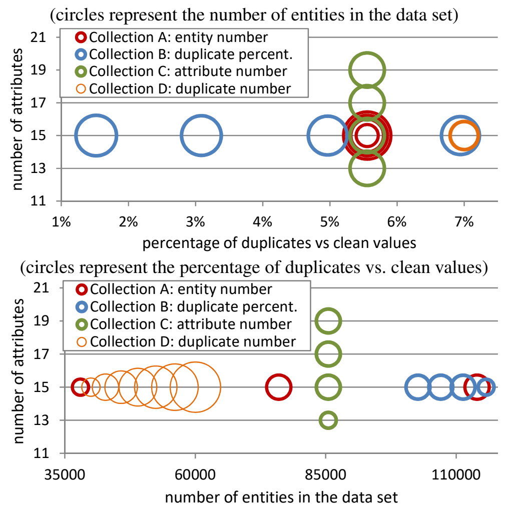
Evaluating collective matching methods, for example those briefly discussed in part B of Section 2.2, requires investigating the behavior of the methods under various data sets characteristics. We now explain how EMBench^{++} can be used for generating collections that allow investigating such characteristics.

For example, consider that we have a technique and would like to examine how it behaves when altering the following characteristics:

(a) *collection size* defined as the total number of entities in the data set on which the technique is executed. This would help to verify that the technique is scalable and thus able to efficiently process collection of various sizes, e.g., collections with 1.000 as well as collections with 1 million entities.

(b) *cleanliness*, which is the percentage of the total number of duplicates with respect to the total number of clean entities in the data set. For example, we would like to check what happens when only 1% of the entities in the collection are duplicates and what happens when 50% are duplicates.

(c) *entity size*, i.e., the number of attributes included in the entities. This could assist in testing whether the



	Collection A			
	A-1	A-2	A-3	A-4
• **Collection Size**	38.000	76.000	114.000	152.000
Person	28.500	57.000	85.500	114.000
Article	9.500	19.000	28.500	38.000
• **Cleanliness**	5,6%			
Person clean	27.000	54.000	81.000	108.000
Article clean	9.000	18.000	27.000	36.000
• **Entity Size**	15 attributes			
Person attr.	6	6	6	6
Article attr.	9	9	9	9

	Collection B			
	B-1	B-2	B-3	B-4
• **Collection Size**	107.000-115.000			
Person	86.800	83.500	80.300	77.000
Article	28.930	27.830	26.750	25.680
• **Cleanliness**	2%	3%	5,0%	7%
Person clean	85.500	81.000	76.500	72.000
Article clean	28.500	27.000	25.500	24.000
• **Entity Size**	15 attributes			
Person attr.	6	6	6	6
Article attr.	9	9	9	9

	Collection C			
	C-1	C-2	C-3	C-4
• **Collection Size**	85.500			
Person	57.000	64.125	68.400	71.250
Article	30.000	22.500	18.000	15.000
• **Cleanliness**	5,6%			
Person clean	54.000	60.750	64.800	67.500
Article clean	28.500	21.357	17.100	14.250
• **Entity Size**	13	15	17	19
Person attr.	6	6	6	6
Article attr.	7	9	11	13

	Collection D			
	D-1	D-3	D-5	D-7
• **Entities**	40.000	45.796	52.432	60.029
Person	30.000	34.347	39324	45022
Article	10.000	11.449	13.108	15.007
• **Cleanliness**	0,7% (for all Person & Article data sets)			
• **Entity Size**	15 (6 attr. for Person & 9 for Article)			
• **Duplication**	[1-1]	[1-3]	[1-5]	[1-7]

Fig. 7. Two illustrations and statistics for the data sets in Collections A, B, C and D (can be used for evaluating collective resolution).

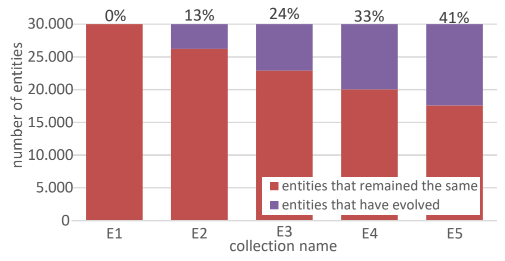


Fig. 8. The number of evolved entities as well as the ones that remained the same in all the data sets in Collection E.

technique can handle small entities, e.g., composed by 1-2 attributes, as well as large entities, i.e., composed by 8-10 attributes.
(d) *duplication*, given the number of duplicates describing the same real world entity. For example, test if the technique can handle collections in which a maximum of 2 entities can refer to the same real world object as well as collections in which up to 25 entities might refer to the same real world entity.

We used EMBench^{++} to generate four collections containing a small number of data sets, with each collection related to one of the four investigated characteristics. The specific characteristic remained identical for all the data sets in the collection. The other characteristics were increased among the data sets of the collection. For the particular generation we focused on Person and Article entity sets, which are the most commonly used types in existing works (Section 5.1).

Figure 7 provides two illustrations of the collections (i.e., the two plots on top of the figure) as well as detailed statistics (i.e., the four tables) for their data sets. *Collection A* is related to the collection size, i.e., characteristic a. As shown in the figure, data set A-1 contains 38.000 entities, A-2 contains 76.000, A-3 contains 114.000 entities, and A-4 contains 152.000. Consider now data set A-1. It contains 38.000, out of which 36.000 are clean and 2.000 refer to the same real object. Thus, cleanliness is 5,6% (i.e., cleanliness=2.000/36.000*100) and it is the same for all data sets of collection A. entity size is 15 (i.e., the total number of attributes used in the entities) and duplication is 2 (i.e., maximum of 2 entities refer the same object).

Collection B is related to the cleanliness, meaning that only this increases among the data sets of the collection whereas the other characteristics remain the same. Note that in this situation the entity size could not be exactly the same but it is almost the same. *Collection C* is related to entity size and thus we see the same value for collection size, cleanliness, and du-

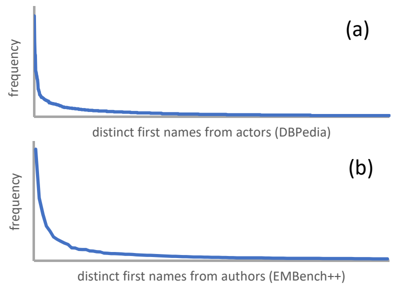


Fig. 9. Distribution of the first names from (a) actors from DBPedia, and (b) authors generated by our system.

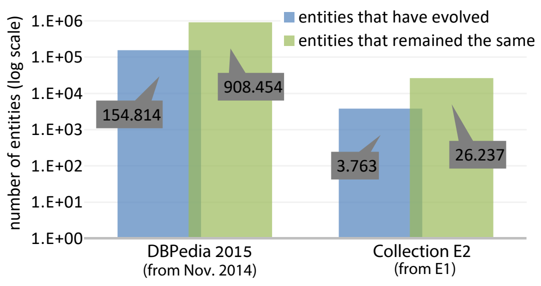


Fig. 10. The percentage of evolved people entities between (a) two DBPedia versions, and (b) the E1-E2 collections.

plication (i.e., 85.500, 5,5%, and 2). Lastly, *Collection D* is related to duplication. Thus, D-1 to D-7 data sets contain an increasing number of duplicates for the same real world entity while having the same cleanliness and entity size. Generating identical collection sizes for all data sets of collection D was not possible, so we see an only slightly increasing value.

5.2.2. *Entity Evolution*

As explained in part C of Section 2.2, a recently appeared research area focuses on dealing with the volatile nature of the data. Our second usability investigation generates data suitable for evaluating such methods.

Using EMBench^{++} we created a collection that contains data sets with entities evolved in time. We used Person entities with the configuration shown in Section 4.3. Starting from a data set of 30.000 Person entities, we created a total of five data sets (named E1, E2, E3, E4 and E5) with each data set containing an evolved version of 3.000 entities from the previous data set.

The resulted data sets are exactly the same except for the number of duplicated entities. More specifically, the total number of entities is 30.000 and the number of entity attributes is 5. Figure 8 provides a graphical illustration of the data sets in Collection E.

5.3. *Representation of Real World Situations*

We now continue with a comparison between data collections generated by EMBench^{++} with real world data from existing collections. The particular evaluation aims at illustrating that the introduced mechanisms are able to generate data that actually represent real world situations.

The first aspect we investigated is the distribution of the values generated by EMBench^{++}. For this evaluation we retrieved people included in two data collections. The first corresponds to actors included in DBPedia movies and the second to authors included in publications generated by our system (e.g., publication collections shown in Figure 7). From these two collections, we used the initial 15.500 distinct names, i.e., DBPedia actors and EMBench^{++} publication authors. We then extracted the first names and computed the appearance frequency of each first name. Figure 9 provides the distribution of the first names for the two collections. The plots illustrate the resemblance between the frequency of first name appearance and actually illustrate that for both collections the first names follow the Zipfian distribution.

We also examined whether our system can capture evolution as this occurs in the real world applications. To examine this we used data from two different versions of DBPedia and in particular data from DBPedia November 2014 and from 2015. We then analyzed the "Persondata" data sets from both versions. More specifically, we computed the number of entities from the 2015 version that had different values than the November 2014 version. This showed that 154.814 entities evolved and 908.454 remained the same, giving a percentage of 14,5% of evolved entities.

Figure 10 shows the number of entities that have evolved as well as the entities that remained the same between the particular DBPedia versions. Furthermore, the plot also shows the same information for the entities evolved from the E1 to the E2 collection. It can be seen that the percentage of evolved entities in E2 is 13%, which is similar to the DBPedia data. In addition, our system is capable of going to larger percentages as for example the ones illustrated in Figure 6.

5.4. *Demonstration of Testing a Real Matching-Related Technique*

EMBench^{++} has been used for generating data to evaluating a real technique for holistic in-database

	C-1	C-2
size (#instances)	20.000	
duplic. vs. clean	10%	6%
max inst. per entity	7	6
number of entities	18.528	19.048
canopies	17.739	18.115
areas	262	171

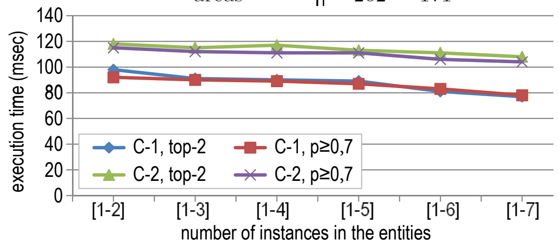


Fig. 11. A plot from the experimental evaluation include in [43], using data generated with EMBench^{++}.

query processing over information extraction pipelines [43]. The authors evaluated their technique on the real data sets of Cora, CiteSeer, DBLP/ACM, which we discussed in Section 5.1. The authors have also used generated data for being able to study the influence of a small set of particular characteristics, such as the number of instances in the collection.

More specifically, EMBench^{++} was used for generating 3 collections with a total of 12 data sets. Each collection had a fixed value on one of the investigated characteristics and an increased number for the other characteristics, similar to the ones discussed in Figure 7. The data sets from these collections were used for evaluating various aspects of the introduced technique. The following list provides some of the performed tests:

- Collection Size. The test examined efficiency and effectiveness of the technique when increasing the number of entities in the collection, for example on collections with 2.000 entities until 20.000 entities.
- Number of Instances in Entities. The test investigated the technique with entities consisting of a different number of instances, for example with up to 2 instances match the same real world entities or with up to 7 instances match the same real world entities.
- Ratio of Duplicates. The test valuated the technique on collections with different ratio of duplicated vs. clean entities. For example, only 4% of the collection instances can refer to the same real world entities, or 10% of the collection instances refer to the same real world entities.

As an example, consider the evaluation result shown in Figure 11 (originally shown in [43]). This uses two of the synthetic data sets, namely the C-1 and C-2 data sets. Both data sets contain 20.000 entities but a different number of duplicates with a different ration of duplicated vs. clean entities (i.e., 10% C-1 and 6% for C-2) and different number of maximum instances per entity (i.e., 7 for C-1 and 6 for C-2). The authors performed evaluations for each of these data sets and for each of the two supported query types, which are top-k and threshold. Then, they reported execution time (i.e., efficiency) according to the number of maximum instances per entity.

6. Conclusions

We have introduced a system for generating benchmark data that can be used for the extensive evaluation of matching-related methods. Our main contributions include the usage of the available schema information during the modification of entities, generating data sets with evolved versions of entities, and controlling not just the generation of single data sets but collections of data sets. Note that the implementation of EMBench^{++} with the default repository data as well as the configuration and collections involved in the usability experiments will be made available in the final version of the journal.

References

[1] A. Elmagarmid, P. Ipeirotis and V. Verykios, Duplicate Record Detection: A Survey, *Transactions on Knowledge and Data Engineering (TKDE)* **19**(1) (2007), 1–16. doi:10.1109/TKDE.2007.250581.

[2] Z. Miklós, N. Bonvin, P. Bouquet, M. Catasta, D. Cordioli, P. Fankhauser, J. Gaugaz, E. Ioannou, H. Koshutanski, A. Maña, T. Palpanas and H. Stoermer, From Web Data to Entities and Back, in: *CAiSE*, B. Pernici, ed., Lecture Notes in Computer Science, Vol. 6051, Springer, 2010, pp. 302–316. ISBN 978-3-642-13093-9. doi:10.1007/978-3-642-13094-6. https://doi.org/10.1007/978-3-642-13094-6.

[3] A. Morris, Y. Velegrakis and P. Bouquet, Entity Identification on the Semantic Web, in: *Proceedings of the Workshop on Semantic Web Applications and Perspectives (SWAP)*, A. Gangemi, J. Keizer, V. Presutti and H. Stoermer, eds, CEUR Workshop Proceedings, Vol. 426, CEUR-WS.org, 2008. http://ceur-ws.org/Vol-426.

[4] N. Dalvi and D. Suciu, Management of probabilistic data: foundations and challenges, in: *Proceedings of the SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, L. Libkin, ed., ACM, 2007, pp. 1–12. ISBN 978-1-59593-685-1. doi:10.1145/1265530.1265531. http://dl.acm.org/citation.cfm?id=1265530.

[5] E. Ioannou, W. Nejdl, C. Niederée and Y. Velegrakis, On-the-Fly Entity-Aware Query Processing in the Presence of Linkage, *Proceedings of the Very Large Database Endowment (PVLDB)* **3**(1) (2010), 429–438. doi:10.14778/1920841.1920898.

[6] P. Li, X.L. Dong, A. Maurino and D. Srivastava, Linking Temporal Records, *Proceedings of the Very Large Database Endowment (PVLDB)* **4**(11) (2011), 956–967.

[7] C. Re, N. Dalvi and D. Suciu, Efficient Top-k Query Evaluation on Probabilistic Data, in: *Proceedings of the International Conference on Data Engineering (ICDE)*, R. Chirkova, A. Dogac, M.T. Özsu and T.K. Sellis, eds, IEEE Computer Society, 2007, pp. 886–895. ISBN 1-4244-0802-4. doi:10.1109/ICDE.2007.367934. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4221634.

[8] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar and S. Fienberg, Adaptive Name Matching in Information Integration, *Intelligent Systems* **18**(5) (2003), 16–23. doi:10.1109/MIS.2003.1234765.

[9] M. Cheatham and P. Hitzler, String Similarity Metrics for Ontology Alignment, in: *Proceedings of the International Semantic Web Conference (ISWC)*, H. Alani, L. Kagal, A. Fokoue, P.T. Groth, C. Biemann, J.X. Parreira, L. Aroyo, N.F. Noy, C. Welty and K. Janowicz, eds, Lecture Notes in Computer Science, Vol. 8219, Springer, 2013, pp. 294–309. ISBN 978-3-642-41337-7. doi:10.1007/978-3-642-41338-4_19. https://doi.org/10.1007/978-3-642-41338-4.

[10] L. Getoor and C. Diehl, Link mining: a survey, *SIGKDD Explorations* **7**(2) (2005), 3–12. doi:10.1145/1117454.1117456.

[11] E. Ioannou and S. Staworko, Management of Inconsistencies in Data Integration, in: *Data Exchange, Integration, and Streams*, P.G. Kolaitis, M. Lenzerini and N. Schweikardt, eds, Dagstuhl Follow-Ups, Vol. 5, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 217–225. ISBN 978-3-939897-61-3. doi:10.4230/DFU.Vol5.10452.217. http://www.dagstuhl.de/dagpub/978-3-939897-61-3.

[12] E. Ioannou and Y. Velegrakis, EMBench: Generating Entity-Related Benchmark Data, in: *Proceedings of the Posters & Demonstrations Track, a track within the International Semantic Web Conferene (ISWC)*, M. Horridge, M. Rospocher and J. van Ossenbruggen, eds, CEUR Workshop Proceedings, Vol. 1272, CEUR-WS.org, 2014, pp. 113–116. http://ceur-ws.org/Vol-1272.

[13] E. Ioannou, N. Rassadko and Y. Velegrakis, On Generating Benchmark Data for Entity Matching, *Journal of Data Semantics* **2**(1) (2013), 37–56. doi:10.1007/s13740-012-0015-8.

[14] G. Papadakis, G. Giannakopoulos, C. Niederée, T. Palpanas and W. Nejdl, Detecting and Exploiting Stability in Evolving Heterogeneous Information Spaces, in: *Proceedings of the Joint International Conference on Digital Libraries (JCDL)*, G. Newton, M.J. Wright and L.N. Cassel, eds, ACM, 2011, pp. 95–104. ISBN 978-1-4503-0744-4. doi:10.1145/1998076.1998094. https://doi.org/10.1145/1998076.

[15] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas and W. Nejdl, Eliminating the redundancy in blocking-based entity resolution methods, in: *Proceedings of the Joint International Conference on Digital Libraries (JCDL)*, G. Newton, M.J. Wright and L.N. Cassel, eds, ACM, 2011, pp. 85–94. ISBN 978-1-4503-0744-4. doi:10.1145/1998076.1998093. https://doi.org/10.1145/1998076.

[16] F. Rizzolo, Y. Velegrakis, J. Mylopoulos and S. Bykau, Modeling Concept Evolution: A Historical Perspective, in: *Proceedings of the International Conference on Conceptual Modeling (ER)*, A.H.F. Laender, S. Castano, U. Dayal, F. Casati and J.P.M. de Oliveira, eds, Lecture Notes in Computer Science, Vol. 5829, Springer, 2009, pp. 331–345. ISBN 978-3-642-04839-5. doi:10.1007/978-3-642-04840-1_25. https://doi.org/10.1007/978-3-642-04840-1.

[17] M. Achichi, M. Cheatham, Z. Dragisic, J. Euzenat, D. Faria, A. Ferrara, G. Flouris, I. Fundulaki, I. Harrow, V. Ivanova, E. Jiménez-Ruiz, E. Kuss, P. Lambrix, H. Leopold, H. Li, C. Meilicke, S. Montanelli, C. Pesquita, T. Saveta, P. Shvaiko, A. Splendiani, H. Stuckenschmidt, K. Todorov, C.T. dos Santos and O. Zamazal, Results of the Ontology Alignment Evaluation Initiative 2016, in: *OM workshop co-located with ISWC*, 2016, pp. 73–129.

[18] A. Ferrara (contact person), The ISLab Instance Matching Benchmark, http://islab.di.unimi.it/iimb/.

[19] A. Ferrara, S. Montanelli, J. Noessner and H. Stuckenschmidt, Benchmarking Matching Applications on the Semantic Web, in: *Proceedings of the Extended Semantic Web Conference (ESWC), Part II*, G. Antoniou, M. Grobelnik, E.P.B. Simperl, B. Parsia, D. Plexousakis, P.D. Leenheer and J.Z. Pan, eds, Lecture Notes in Computer Science, Vol. 6644, Springer, 2011, pp. 108–122. ISBN 978-3-642-21063-1. doi:10.1007/978-3-642-21064-8_8. https://doi.org/10.1007/978-3-642-21064-8.

[20] T. Saveta, E. Daskalaki, G. Flouris, I. Fundulaki, M. Herschel and A.N. Ngomo, LANCE: Piercing to the Heart of Instance Matching Tools, in: *Proceedings of the International Semantic Web Conference (ISWC), Part I*, M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P.T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan and S. Staab, eds, Lecture Notes in Computer Science, Vol. 9366, Springer, 2015, pp. 375–391. ISBN 978-3-319-25006-9. doi:10.1007/978-3-319-25007-6_22. https://doi.org/10.1007/978-3-319-25007-6.

[21] T. Saveta, SPIMBench: A Scalable, Schema-Aware Instance Matching Benchmark for the Semantic Publishing Domain, Master's thesis, University of Crete, Greece, 2014.

[22] B. Alexe, W. Tan and Y. Velegrakis, STBenchmark: towards a benchmark for mapping systems, *Proceedings of the Very Large Database Endowment (PVLDB)* **1**(1) (2008), 230–244. doi:10.14778/1453856.1453886.

[23] V. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Physics Doklady* **10**(8) (1966), 707–710.

[24] M. Jaro, Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida, *Journal of the American Statistical Association* **84** (1989), ISSN 01621459.

[25] W. Winkler, The state of record linkage and current research problems, 1999.

[26] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, Inc., New York, NY, USA, 1984. ISBN 0-07-054484-0.

[27] W. Cohen, P. Ravikumar and S. Fienberg, A Comparison of String Distance Metrics for Name-Matching Tasks, in: *Proceedings of the International Workshop on Information Integration on the Web (IIWeb), co-located with the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 73–78.

[28] M. Hernández and S. Stolfo, Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem, *Data Mining and Knowledge Discovery* **2**(1) (1998), 9–37. doi:10.1023/A:1009761603038.

[29] S. Tejada, C. Knoblock and S. Minton, Learning domain-independent string transformation weights for high accuracy object identification, in: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, ACM, New York, NY, USA, 2002, pp. 350–359. doi:10.1145/775047.775099.

[30] A. Doan, Y. Lu, Y. Lee and J. Han, Object Matching for Information Integration: A Profiler-Based Approach, in: *Proceedings of the International Workshop on Information Integration on the Web (IIWeb), co-located with the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 53–58.

[31] X. Dong, A. Halevy and J. Madhavan, Reference Reconciliation in Complex Information Spaces, in: *Proceedings of the International Conference on Management of Data (SIGMOD)*, ACM, New York, NY, USA, 2005, pp. 85–96. doi:10.1145/1066157.1066168.

[32] I. Bhattacharya and L. Getoor, Deduplication and Group Detection Using Links, in: *Proceedings of the Workshop on Link Analysis and Group Detection (LinkKDD), co-located with the International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, 2004.

[33] I. Bhattacharya and L. Getoor, Iterative record linkage for cleaning and integration, in: *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), co-located with the International Conference on Management of Data (SIGMOD)*, ACM, New York, NY, USA, 2004, pp. 11–18. doi:10.1145/1008694.1008697.

[34] D. Kalashnikov, S. Mehrotra and Z. Chen, Exploiting Relationships for Domain-independent Data Cleaning, in: *Proceedings of the International Conference on Data Mining (SIAM SDM)*, H. Kargupta, J. Srivastava, C. Kamath and A. Goodman, eds, SIAM, 2005. ISBN 978-0-89871-593-4. doi:10.1137/1.9781611972757.24. https://doi.org/10.1137/1.9781611972757.

[35] D. Kalashnikov and S. Mehrotra, Domain-independent Data Cleaning via Analysis of Entity-relationship Graph, *Transactions on Database Systems (TODS)* **31**(2) (2006), 716–767. doi:10.1145/1138394.1138401.

[36] S.E. Whang and H. Garcia-Molina, Incremental entity resolution on rules and data, *The International Journal on Very Large Data Bases* **23**(1) (2014), 77–102. doi:10.1007/s00778-013-0315-0.

[37] A. Gruenheid, X.L. Dong and D. Srivastava, Incremental Record Linkage, *Proceedings of the Very Large Database Endowment (PVLDB)* **7**(9) (2014), 697–708. doi:10.14778/2732939.2732943.

[38] M. Hernández and S. Stolfo, The Merge/Purge Problem for Large Databases, in: *Proceedings of the International Conference on Management of Data (SIGMOD)*, ACM, New York, NY, USA, 1995, pp. 127–138. doi:10.1145/223784.223807.

[39] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée and W. Nejdl, A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces, *Transactions on Knowledge and Data Engineering* **25**(12) (2013), 2665–2682. doi:10.1109/TKDE.2012.150.

[40] V. Rastogi, N. Dalvi and M. Garofalakis, Large-Scale Collective Entity Matching, *Proceedings of the Very Large Database Endowment (PVLDB)* **4**(4) (2011), 208–218. doi:10.14778/1938545.1938546.

[41] H. Poon and P. Domingos, Joint Inference in Information Extraction, in: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, AAAI Press, 2007, pp. 913–918. ISBN 978-1-57735-323-2.

[42] G. Papadakis, G. Koutrika, T. Palpanas and W. Nejdl, Meta-Blocking: Taking Entity Resolution to the Next Level, *Transactions on Knowledge and Data Engineering (TKDE)* **26**(8) (2014), 1946–1960. doi:10.1109/TKDE.2013.54.

[43] E. Ioannou and M. Garofalakis, Holistic Query Evaluation over Information Extraction Pipelines, *Proceedings of the Very Large Database Endowment (PVLDB)* **11**(2) (2017), 217–229. doi:10.14778/3149193.3149201.

[44] A.N. Ngomo, M.A. Sherif and K. Lyko, Unsupervised Link Discovery through Knowledge Base Repair, in: *Proceedings of the International Conference of the Semantic Web (ESWC)*, Springer International Publishing, Cham, 2014, pp. 380–394. doi:10.1007/978-3-319-07443-6_26.

[45] H. Köpcke, A. Thor and E. Rahm, Evaluation of entity resolution approaches on real-world match problems, *Proceedings of the Very Large Database Endowment (PVLDB)* **3**(1) (2010), 484–493. doi:10.14778/1920841.1920904.

[46] K. Dreßler and A.N. Ngomo, On the efficient execution of bounded Jaro-Winkler distances, *Semantic Web* **8**(2) (2017), 185–196. doi:10.3233/SW-150209.

[47] C. Stadler, J. Lehmann, K. Höffner and S. Auer, LinkedGeoData: A core for a web of spatial open data, *Semantic Web* **3**(4) (2012), 333–354. doi:10.3233/SW-2011-0052.

[48] G. Newton, M.J. Wright and L.N. Cassel (eds), Proceedings of the 2011 Joint International Conference on Digital Libraries, JCDL 2011, Ottawa, ON, Canada, June 13-17, 2011, ACM, 2011. ISBN 978-1-4503-0744-4. doi:10.1145/1998076. https://doi.org/10.1145/1998076.