Semantic Web 1 (2018) 1–5
IOS Press

EMBench^{++}: Benchmark Data for Thorough Evaluation of Matching-Related Methods

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

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*

Abstract. Matching-related methods, i.e., entity resolution, search and evolution, 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 present collections of entity sets generated by EMBench^{++} as an illustration of the benefits provided by our tool.

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. Strongly related to entity matching are the tasks of entity search and evolution, which focus on efficiently and effectively retrieving the entities of a given collection that best match the user's description.

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 [5], [6], [12], [15], and [22]. 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 [23,21], 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 [21] 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 [33,34], applications, especially Web 2.0 applications, focus on enabling and encouraging users to constantly contribute and modify existing content. A DBPedia analysis has revealed that the data describing the entities has been modified overtime time, and only a small fraction of it has remained 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 [40,29].

In this work, we are introducing EMBench^{++}, an extension of our previous system with additional mechanisms aiming at generating benchmark data for the evaluation of matching-related methodologies, which (primarily) include other than entity matching and search, also entity evolution. EMBench^{++} moves towards a 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 with 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 datasets that consider not only individual entities but also sets of entities determined by their schema information.
- We are generating collections of data sets that are able to capture and evaluate specific matching-related aspects.
- 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 evaluating 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 challenged.

2.1. Synthetic Data Generation

On 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 [1].

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. 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 ISLab Instance Matching Benchmark [14]. The particular benchmark, first includes entities from the OKKAM project [30,31]. 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

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

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 [13]. SWING design principles and goals are similar to EMBench but EMBench [23,21] has more expressive power and offers more flexibility in the specification of the testing data. A detailed discussion is available in [21]. 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.

The most recent approach is LANCE [42], 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.

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 [2], 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.

2.2. 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 [6] and [5]), such as Levenshtein distance [28], Jaro [24], Jaro-Winkler [46] and TF/IDF similarity [41]. Note that [7] and [5] describe and discuss an experimental comparison of various basic similarity techniques used for matching names. Merge-purge [18] 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* [44], such as abbreviation, stemming, and initials, or predefined rules [9] 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* [10]. 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 [3,4] 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 [26,25] 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 mechanisms. 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 [39,8,20,29]. More specifically, [39,8] 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 [20] 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

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

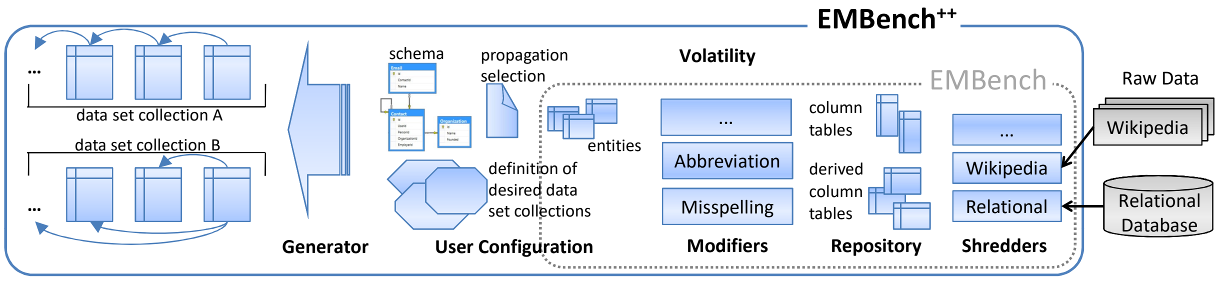


Fig. 1. An illustration of EMBench^{++}'s architecture (grey dotted line denotes the components from the previous version).

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, [45] 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 [16] 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 [17]. 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., [35]. 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 [38] 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 architecture of EMBench^{++} and discuss the additions included from the previous version of the system, which was described in [21] and [23]. Figure 1 provides a graphical illustration of the 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.

(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.

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 identify function rule meaning that the resulted derived table is identical to the column table without any modification. It also supports function rules that combining column tables between them or with 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. 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 [23], EMBench contains implementations for 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, refer 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 particularly, we need the generated entity sets to capture all the aspects required for a complete and extensive evaluation of matching-related methods, which were 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 $I(n_1,...,n_k)$, is a set of entities $\{e_1,...,e_n\}$, where $I \in \mathcal{N}$ and is referred to as the **name** of the entity set, and all the entities in the set are of the same type $\langle n_1,...,n_k \rangle$.* ■

Note that 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 rerate 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 identification name 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 $\langle i, t, l \rangle$.

The modifiers are not executed on the whole entity set but on a subset of it. Initially, the entity set I_{k-1} into two sets: I^m_{k-1} that contains randomly selected entities and I^o_{k-1} that contains the remaining entities, i.e., I^o_{k-1}=$I_{k-1} \setminus I^m_{k-1}$. The entities of I^o_{k-1} are directly included in the entity set I_k whereas the entities of I^m_{k-1} are first modified and then included in I_k.

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

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., $c \leq |I|$. 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 $I_{k-1} \xrightarrow{f_{k-1}} I_k$. Since the result of a modifier is

```
<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.

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

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 scenario** 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 pair $\langle I, e_m \rangle$, 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 $f_1, f_2, \dots, f_m$. 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 $\langle I, e_m, e \rangle$ $\forall e \in I$ 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 e as the best match of entity e_m in the entity set I.

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): $F_a = F_a \cup \{f_j\}$*

C-A0	**C-A1**	**C-A2**
F_{a1}={}	F_{a2}=addition(F_{a1},⟨missp.,10%⟩)	F_{a3}=adjust(F_{a1},missp.,10%,+)
person article	C-**A0**.person + misspelling 10% C-**A0**.article + misspelling 10%	C-**A0**.person + misspelling 20% 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 article	C-**B0**.person + volatility 10% C-**B0**.article + volatility 10%	C-**B1**.person + volatility 20% C-**B1**.article + volatility 20%

Fig. 4. Independent and Sequential propagation.

- *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 $\odot$={+,−} ∎

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, …} 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 containg a configuration setting equal to t by setting the value of this configuration setting, i.e., t, from level to 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, …. 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

```
<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.

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 [33,34], 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 [40].

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 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 " ".

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.

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: that 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., $e_1.n_i$, $e_2.n_i$, $\ldots$, $e_n.n_i$.

B.2) Expansion: that 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 from a small set of works [10,19,36,37] and only few include entities of other types [11,27,32,36]. 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 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 contain any 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 that 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

Publication-related Data Collections							
Dataset	**Types**	**Instances**	**Entities**	**Duplicates**	**Public**	**Evolution**	**Ground-truth**
Cora, e.g., [37,10,20]	citations	6.107	338	1-21	yes	no	included
DBLP/ACM, e.g., [36,19]	citations	4.671	2.552	1-2	yes	no	included
CiteSeer, e.g., [37,19]	citations	1.031	558	1-21	yes	no	included
Four PIMs, i.e., [10]	citations	total=103.435	total=9.989	avg. 10	no	no	manually
KDD Cup 2003, e.g., [38]	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., [10]	business	5.000 each entity	87	avg. 5.000	not stated	artificial	included
IMDB/DBPedia, i.e., [36]	movies	50.797	22.863	0-2	yes	no	included
Amazon/Google, e.g., [32,27,36]	products	4.393	1.104	0-2	yes	no	included
Abt-Buy, e.g., [27]	products	2.173	1.097	0-2	yes	no	included
Wikipedia, DBpedia, e.g., [11]	various	5,48M[3]	5,48M	none	yes	yes	included
LinkedGeoData, e.g., [11]	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.

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 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.

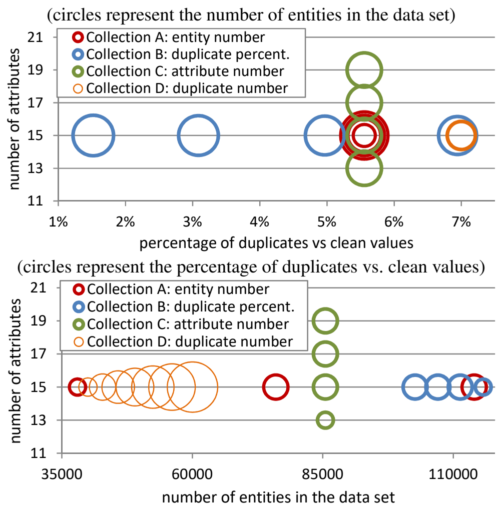
We used EMBench^{++} to generate four collections containing a small number of data sets, with each collection related to one of the four investigated characteristics. The specific characteristic remained identical for all the data sets in the collection. The other characteristics were increased among 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 clear 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 duplication (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.



	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).

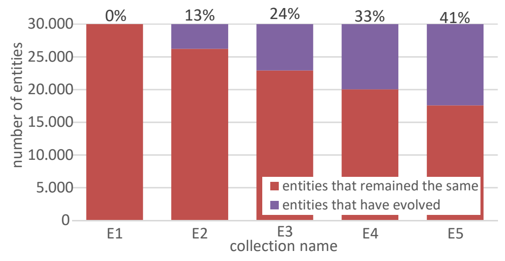


Fig. 8. The number of evolved entities as well as the ones that remained the same in all the data sets in Collection E.

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.

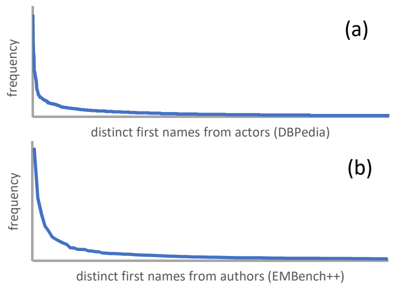


Fig. 9. Distribution of the first names from (a) actors from DBPedia, and (b) authors generated by our system.

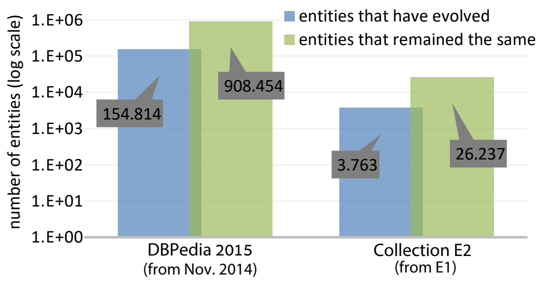


Fig. 10. The percentage of evolved people entities between (a) two DBPedia versions, and (b) the E1-E2 collections.

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

The VLDB 2018 publication [19], provides an example of how EMBench^{++} can be used for generating and using data for the experimental evaluation. 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 tech-

	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

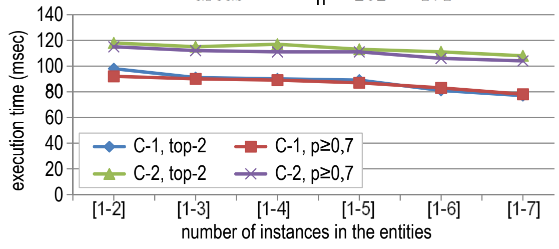


Fig. 11. A plot from the experimental evaluation include in [19], using data generated with EMBench^{++}.

nique. 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 [19]). 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 included 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] 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*, pages 73–129, 2016.

[2] B. Alexe, W. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.

[3] I. Bhattacharya and L. Getoor. Deduplication and group detection using links. In *LinkKDD*, 2004.

[4] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *DMKD*, pages 11–18, 2004.

[5] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[6] M. Cheatham and P. Hitzler. String similarity metrics for ontology alignment. In *ISWC*, pages 294–309, 2013.

[7] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb co-located with IJCAI*, pages 73–78, 2003.

[8] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

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

[10] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[11] K. Dreßler and A. N. Ngomo. On the efficient execution of bounded jaro-winkler distances. *Semantic Web*, 8(2):185–196, 2017.

[12] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[13] A. Ferrara, S. Montanelli, J. Noessner, and H. Stuckenschmidt. Benchmarking matching applications on the semantic web. In *ESWC (2)*, pages 108–122, 2011.

[14] A. Ferrara (contact person). The islab instance matching benchmark. Published online. http://islab.di.unimi.it/iimb/.

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

[16] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, 2014.

[17] M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[18] M. Hernández and S. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[19] E. Ioannou and M. Garofalakis. Holistic query evaluation over information extraction pipelines. *PVLDB*, 11(2):217–229, 2017.

[20] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.

[21] E. Ioannou, N. Rassadko, and Y. Velegrakis. On generating benchmark data for entity matching. *J. Data Semantics*, 2(1):37–56, 2013.

[22] E. Ioannou and S. Staworko. Management of inconsistencies in data integration. In *Data Exchange, Integration, and Streams*, pages 217–225. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[23] E. Ioannou and Y. Velegrakis. Embench: Generating entity-related benchmark data. In *ISWC*, pages 113–116, 2014.

[24] M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *American Statistical Association*, 84, 1989.

[25] D. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *TODS*, 31(2):716–767, 2006.

[26] D. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM SDM*, 2005.

[27] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[28] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[29] P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.

[30] 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*, pages 302–316, 2010.

[31] A. Morris, Y. Velegrakis, and P. Bouquet. Entity identification on the semantic web. In *SWAP*, 2008.

[32] A. N. Ngomo, M. A. Sherif, and K. Lyko. Unsupervised link discovery through knowledge base repair. In *ESWC*, pages 380–394, 2014.

[33] G. Papadakis, G. Giannakopoulos, C. Niederée, T. Palpanas, and W. Nejdl. Detecting and exploiting stability in evolving heterogeneous information spaces. In *JCDL*, pages 95–104, 2011.

[34] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, pages 85–94, 2011.

[35] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.

[36] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolution to the next level. *TKDE*, 26(8):1946–1960, 2014.

[37] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI*, pages 913–918, 2007.

[38] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.

[39] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[40] F. Rizzolo, Y. Velegrakis, J. Mylopoulos, and S. Bykau. Modeling Concept Evolution: A Historical Perspective. In *ER*, pages 331–345, 2009.

[41] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[42] T. Saveta, E. Daskalaki, G. Flouris, I. Fundulaki, M. Herschel, and A. N. Ngomo. LANCE: piercing to the heart of instance matching tools. In *ISWC*, pages 375–391, 2015.

[43] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.

[44] S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.

[45] S. E. Whang and H. Garcia-Molina. Incremental entity resolution on rules and data. *VLDB J.*, 23(1):77–102, 2014.

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

EMBench++: Benchmark Data for Thorough Evaluation of Matching-Related Methods, by E. Ioannou and Y. Velegrakis

We thank the editor and reviewers for the effort they devoted on our submission, and the positive feedback that they have provided, as well as the fact that they have appreciated the usefulness of our work. We have carefully revised our manuscript in order to improve its quality. All the comments and suggestions have been addressed. We truly believe that the review comments were instrumental to improve several aspects of the manuscript, and especially the empirical evaluation section (which has been extended with additional evaluations, suggested by the reviewers, as well as clarifications on the previous evaluations). The following paragraphs present the actions taken for each of the reviewer's comments.

Reviewer 1 (Recommendation: Major Revision)

The paper describes EMBench++, an extension of a tool to generate benchmarks for entity resolution. The tool (significantly) extends a previous tool (EMBench) with more features, in particular supporting evaluation in contexts where entity schemas are different and entities evolve along time.

The tool and its new features provide several functionalities and control mechanisms over the generated datasets. In addition the feature to generate evolving entity sets is to me extremely useful to evaluate approaches that tackle the important topic of matching representations that change along time. Thus, the paper contains a contribution that is original compared to previous work.

- - - - - - - - - - - -
The paper is reasonably understandable, although it contains several typos and some paragraphs require further explanations. In particular, the explanation of Figure 5 and concepts used therein is not clear and need to be improved.

We have addressed the raised concerns in the revised manuscript, i.e., corrected the typos, extended the text for incorporating additional explanations, and improved the explanation of Figure 5. The following paragraphs provide more details on the actions taken with respect to the incorporated improvements and extensions.

- - - - - - - - - - - -
I am quite convinced about the usefulness of the tool. Section 5 provides good arguments for the effectiveness of control mechanisms for the generation of datasets with desired features. As a small remark, Section 5 does not discuss a proper usability evaluation. Thus I suggest changing its title.

Thank you for the suggestion. The title of the particular section in the revised version is "Empirical Evaluation", which we also feel that it better represents the new content of the section.

- - - - - - - - - - - -
What instead is missing in my opinion is an equally convincing argument for the usefulness and representativeness of the datasets that are generated.

Following this comment, we extended the section's content with respect to the usefulness and representativeness of the generated data sets. Detailed comments for these extensions are given in the answers of the next paragraphs.

- - - - - - - - - - - -
I can better explain these concerns with a set of questions that the authors should address in the new version of the paper (in case of major revision).
• Have datasets generated with EMBench and EMBench++ been used to evaluate some entity resolution approach so far? If yes, to what extent?

Yes, there are authors that used our system to generate data for evaluating their entity-related techniques. A recent VLDB publication (from one of the authors), used data generated from EMBench++ for performing experiments in order to compliment the ones performed using real world data sets. For example, test the introduced technique when increasing the number of instances in the collection and when having collections with a larger ratio of duplicated vs. clean instances. For performing the evaluations, the authors used our system to generate three collections with a total of 12 data sets.

In the revised version of our manuscript we have included a short description of the particular experimental setup along with a discussion on tests performed using these collections and data sets. The description is included under "Demonstration of Usage" in Section 5.

- - - - - - - - - - - -

• How the data generated with the tool relate to other datasets used to evaluate entity resolution approaches? The relation to OAEI is quite clear, but other datasets have been used e.g., DBpedia and LinkedGeoData [1], or ACM-DBLP, Amazon-Google and Abt-Buy datasets used in [2]. Can you generate datasets of the same size of datasets used so far? Is there an upper bound to the size of datasets that you can generate? More specifically, these questions become important in relation to datasets used to evaluate approaches for collective matching, entity evolution (a missing reference for this body of work is to temporal record linkage [3]), and blocking-based methods. In other words, arguments should be given (after the description of the tool, for example in a subsection of Section 5) to convince that the generated datasets have the features that are required to test these approaches. To this end, you may compare datasets that you can create with EMBench++ to datasets used in these domains (e.g., making a table that list datasets used so far for evaluation and their features, e.g., synthetic vs non synthetic, size, etc..), or to sum up main challenges found for these approaches in the sate-of-the-art and discuss how these challenges are well represented in datasets built with EMBench++.

In order to address these issues, we have followed the reviewer's suggestions and compared the data that one can create with EMBench++ to data sets used in existing publications.

The related information is included in part "Advances over Static Collections" of Section 5 and is summarized in Tables 1 and 2. Comparisons are discussed by separating the data sets into two categories: (1) collections containing data related to publications and (2) collections containing other entity types. In all cases, we provide also the capabilities of generating corresponding data sets using the EMBench++ system.

- - - - - - - - - - - -

• Will the tool be publicly available? (I consider this as an important factor for acceptance)

Absolutely! We had done that with the first version of our tool, the EMBench (http://db.disi.unitn.eu/pages/EMBench/), and we will do the the same for the EMBench++. We will include the source code, example configurations (e.g., the ones used for the data sets generated for this manuscript), and values for loading the repository. These will also be accompanied by an online tool that will allow users to easily execute the functionalities of EMBench++. We have not made it public yet, first because the paper is not yet accepted, and second, because we need to polish it so that it will be easier for people to use it, as we had done with the EMBench. We are working towards this goal.

- - - - - - - - - - - -

For the above-mentioned reasons, and despite the merit of the work described in the paper, I think that the paper needs to be revised before acceptance. I add more detailed comments below the reference.

We have carefully gone through all provided comments and revised our manuscript in order to capture the given suggestions and raised questions. Please see the actions taken in the following paragraphs.

- - - - - - - - - - - -

P. 2

The benefit of using synthetic data … → Yes, but real-world data may reflect more heterogeneities that can be found in real-world matching problems. I suggest discussing pros and cons of synthetic datasets.

We have rechecked the paragraph from the manuscript that was mentioned in the comment and agree with the reviewer that a more complete discussion should be included. Thus, we have updated the particular part based on the provided suggestion, i.e., a discussion on the pros and cons of using synthetic/real data sets for the evaluation of matching-related techniques. The principal advantage of the synthetic data is that we have a better control of the test cases we want to run so that we get a better understanding of the capabilities of the system.

- - - - - - - - - - - -
P. 4
It took me a while to understand if derived column tables contain again one column each (which results from rules applied to column tables) or contain more columns. I suggest stating this more explicitly.

We have revised the corresponding paragraph to making this explicit. We have also included a small example of a rule.

- - - - - - - - - - - -
"for for the attribute values" → for the attribute values
"These sources is" → These sources are

We have corrected these mistakes in the revised version. Thanks for pointing them.

- - - - - - - - - - - -
P. 5
Figure 2: Do you support URIs as identifiers? If yes, do you support URI generation schemes (e.g., www.example.org./person/Noela_Kulgen)? It may be helpful for benchmarking entity resolution for RDF.

Our current implementation does not explicitly support URIs as identifiers. However, we do see the usefulness and are considering them for the next versions of the system. Our goal is to continue improving it as we did with the EMBench, that we extended it to EMBench++.

- - - - - - - - - - - -
"or the same set but different level of destruction. " → please, revise "levels of destruction".
"Advance Generation of Benchmark Data " → Advanced Generation of Benchmark Data (?)
"either data among the different entities or the heterogeneity level. " → please, revise.
"incorporating a particular type of " → incorporating particular types of
"I_m=f_m(f_m−1(... f(I))). " → shouldn't be I_m=f_m(f_m−1(...f_1(I))) ?

We have revised and corrected these sentences.

- - - - - - - - - - - -
I suggest revise the explanation of Definition 2. It is a very intuitive concept (and definition) if the intuition that each modified entity set contains a subset of unmodified entities along with a set of modified entities is explained before the formal definition (e.g., using explanations in the paragraph "EMBench++ uses c to split […]".

As an action to the comment we improved the explanation of Definition 2. More specifically, we followed the suggestion and explained before the definition that each modified entity set contains a subset of unmodified entities along with a set of modified entities. We have also modified Definition 1 to make it more complete.

- - - - - - - - - - - -
P. 6
"related to its ability of the approach to " → related to its ability to
"ensuring foreign keys agree " → ensuring that foreign keys agree
"using attributes that are other entity types " → using attributes that have entities of other types as values (?)

We have revised these sentences in the updated version.

- - - - - - - - - - - -
Use listings for pieces of code, add a caption and reference them (similarly o Figures) in the text.

We modified the manuscript and now all pieces of code are illustrated as figures, as suggested.

- - - - - - - - - - - -
P. 7
Fig. 3.: you have “F_a,” for C-A1, but “F_a,” for C-A2 (similarly for C-B1 and C-B2). Harmonize the notation in the figure.
“and keep incrementally include entities” → please, revise.
The adjust function in Definition 4 should be explained better. I suggest also using an example.
“we first apply the operators over F_a and generated F_b” → we first apply the operators over F_a and generate F_b (?)
“C-A is a independent” → C-A is an independent

We revisited these sentences. After a close consideration of the unclarity commented for Definition 4, we updated the definition for simplifying the description of the particular operation, added a paragraph to describe the content of the definition, and included an example.

- - - - - - - - - - - -
P. 8
I like Figure 4 but all mechanisms should be exemplified (listing are helpful but less than example of transformations).

We have included an extensive explanation of the data included in the entity sets of this figure. We also related them to the corresponding configuration and the volatility mechanisms.

- - - - - - - -
“four investigated characteristic” → four investigated characteristics

The sentence has been revised.

- - - - - - - -
P. 9
The plots and the tables in Fig. 5 should be explained more in details, as well as concepts needed to understand them. For example, percentage of clean vs duplicated entities is not clear (what is the total in 5.6%?). Why entities are always 12.000 in Collection B is not clear. In Collection C, I guess that it sould be 90000 instead of 9000. The semantics of circles in the plots is also unclear, because it centers of the circles seem to represent points in the plane. I guess you want to represent the size of the datasets, but size seems to be represented by the x axis.

We have revised the particular part of the paper. Some numbers shown in the figure were incorrectly transferred from the excel sheet to latex and this caused part of the confusion. Overall, we made the following modifications:

- Updated the figure, including the two plots and all tables. The circles in the top plot represent the number of entities in the data set. The circles in the bottom plot represent the percentage of duplicates vs. clean values. This information is now included in the figures.
- We revised the description of the used characteristics and better explained how these are defined. For example, *cleanliness* is the percentage of the total number of duplicates with respect to the total number of clean entities in the data set. Cleanliness for A-1 is 5.6% since this data set contains 38.000 entities, out of which 36.000 are clean and the remaining 2.000 refer to the same real world objects (i.e., cleanliness=2.000/36.000*100).
- The collections are now better explained. Especially the statistics related to Collection A have been described in more details in order to provide the clear setting followed for all the collections. For the other collections we simply provided an overview.

- - - - - - - -
“sets have everything the same except the number of attribute entities.” → Please revise “sets have everything the same”, and I think it should be “number of entity attributes”.
“data sets those entities evolved in time.“ → please, revise.

We have revised these sentences.

Reviewer 2 (Recommendation: Major Revision)

The paper introduces EMBench++, a framework to generate benchmark data. In addition to the previous system, modifiers are introduced which modify data across entity types, a technique is implemented to simulate the evolution of entities and mechanisms are included to generate not only one single data set but whole collections of data sets.

Altogether, the paper is well written and nicely describes the framework as well as the enhancements of the systems. It is getting clear why there is a need for benchmarks supporting for example the evolution of entities because that is an emerging challenge for entity linking systems.

- - - - - - - - - - - -

It would be good to know if the previous system has found widespread support in the entity linking community and whether systems are actually using the generated data sets. The main criticism is a missing evaluation to assess whether the data sets represent a realistic scenarios and to show the behavior of matching systems on the data sets. The characteristics of one data set are investigated but only regarding indicators like the number of entities or the number of duplicates. It would be important to see whether the data sets present realistic matching scenarios. For example, it is not clear if the implemented mechanism to simulate the evolution of the entities, generates data sets that actually represent this scenario. An application of state-of-the-art entity linking systems could show the applicability of the generated data sets and in turn of the EMBench++ framework. Such experiments could be similar to the ones presented by the same authors in the paper "On generating benchmark data for entity matching", where the influence on the modification level on the effectiveness and execution time of one matching system is depicted. To review whether a realistic scenario is for example presented by the data generated for entity evolution, characteristics of entities in a knowledge base that actually evolve could be gathered and compared to the characteristics of the created data set. By applying a matching system, the influence of the evolution on the performance etc. can be shown.

We performed new experiments focusing on investigating whether the data sets generated represent realistic matching scenarios. As commented by the reviewer, one aspect should be testing/verifying if "the implemented mechanism to simulate the evolution of the entities, generates data sets that actually represent this scenario". To investigate this, we used two versions of DBPedia (i.e., data from DBPedia taken at two different times, the first from 2014 and the second from 2015). More specifically, we analyzed the people entities and detected the percentage of entities that evolved. We then used data sets that we generated using EMBench++ to also detected the percentage of evolved entities. EMBench++ can indeed generate data with the same percentage, and thus, it can capture realistic matching scenarios. In addition, we also compared the distribution of the entity values generated by our system against the values of real world entities taken from DBPedia. We found out that both follow the Zipfian distribution. These experiments are presented and discussed in Section 5, under the "Representation of Real World Situations" part.

We also included a discussion with respect to the comment that an "application of state-of-the-art entity linking systems could show the applicability of the generated data sets". For this, we discuss a real technique that used our system to generate benchmarking data for its experimental evaluation. The authors generated 3 collections with a total of 12 data sets, which they used to compliment the experiments performed using real world data sets, i.e., for testing aspects of the technique that could not have been tested using the available real data collections.

In addition to the experiments described above, we also revised and further extended Section 5 "Empirical Evaluation". Particularly, the revised version includes a more clear description of the part related to the illustration of generated data collections and a new part including a discussion of the static data sets that are used in existing publications along with the advances users can get by using EMBench++.

Reviewer 3 (Recommendation: Minor Revision)

This paper describes a system called EMBench++, which generates synthetic tests sets for evaluation of entity matching algorithms. The paper is in general well-written and easy to follow, and the tool is likely of practical interest to researchers in this field.

- - - - - - - - - - - -

I have two primary concerns with the paper. One is related to the related work section. This section describes both synthetic data generation systems and entity matching algorithms. In my view, the description of other synthetic data generation systems (with the exception of ISLab) is somewhat lacking, in that it does not clearly indicate the functionality of all systems mentioned and how that compares to EMBench++. For instance, the paper states that when compared to SWING "EMBench has more expressive power and offers more flexibility in the specification of the testing data." More detail is needed here in order to fully establish the novelty of EMBench++. It might be helpful to include a table comparing the available features of all of the synthetic data generations systems mentioned.

We see the point raised by the reviewer. We did not include such a discussion in the manuscript since this was covered by the previous journal. More specifically, the previous journal included a discussion (along with a table providing an overview of the differences) of the functionalities provided by our system in comparison to the functionalities offered by the SWING system. These functionalities were grouped and discussed according to (1) data acquisition, (2) data generation, and (3) matching scenarios.

In the revised version, we included a small summarization of the particular discussion and we also included a sentence stating that more information can be found in the description of the previous version of the system.

- - - - - - - - - - - -

My second concern is that there is no grounding of the synthetic datasets produced by the tool. In order to establish the utility of EMBench, it would be helpful to provide an argument (either logical or empirical) that the synthetic datasets it produces reflect the types of datasets found in the real world.

We followed the suggestion and include an additional part, named "Representation of Real World Situations", in Section 5 that focuses on the particular aspect of our system.

In short, we performed two new experiments. In the first experiment we compared the distribution of the entity values generated by EMBench++ against the values from real world entities. The second experiment compares the ratio of evolved entities vs. to the ones that did not change. This comparison was also done with respect to entities from a real data set (i.e., DBPedia) and illustrate the resemblances in the specific aspects.

- - - - - - - - - - - -

Minor issues:

The references given on page one for matching-related methods (10, 18) are very narrow. This is a very established field, so it would be helpful to either refer to more systems or to a survey paper.

The citations 10 and 18 are surveys but this was not clear from the given text. We thus revised the sentence for clarifying that the references are surveys for methods related to the studied domain. In addition, we also included other citations that are also surveys.

- - - - - - - - - - - -

When mentioning string similarity metrics in the context of entity matching, it might be helpful to reference Cheatham, Michelle, and Pascal Hitzler. "String similarity metrics for ontology alignment." International Semantic Web Conference. Springer, Berlin, Heidelberg, 2013. (Full disclosure: this is my own paper, but it does seem particularly relevant here.)

We found the given citation to be a nice paper and included it in our manuscript as it is indeed strongly related.

- - - - - - - - - - - -

In Section 2.2, it might be helpful to mention that techniques A through D are not mutually exclusive.

Yes, the techniques A through D are not mutually exclusive. The revised version clarifies that we followed the particular categorization for ease of comprehension and discussion and that there actually exist methods that span more than one category.

- - - - - - - - - - - -

The paper seems unnecessarily verbose in some places. For example, the description of foreign key relationships in Section 4.1, particularly the description of Figure 2(a), is somewhat wordy. The descriptions of the various transformations are also rather in-depth, considering that they are fairly straightforward in most cases.

We carefully revisited the whole text for improving the paper and also used the suggestions included in this comment to shorten and make more concise particular parts of the sections.

- - - - - - - - - - - -

Typos:
“with the most advance being the ability” -> “with the most advanced being the ability”
“can be used for correcting previous matching mistakes errors” -> only need either mistakes or errors, rather than both
“the systems also uses rules” -> “the system also uses rules”
“the mechanisms focuses on” -> “these mechanisms focus on”
“these sources is a (Derived)” -> “these sources are a (Derived)”
“and keep incrementally include entities” -> “and keep incrementally including entities”?
“C-B2generated” -> “C-B2 generated”
“contains data sets those entities evolved in time” -> “contains data sets whose entities evolved over time”
“exactly the same except the number of duplicated entities” -> “exactly the same except for the number of duplicated entities”
“the number of entity attributes to 5” -> “the number of entity attributes is 5”

We have corrected all the listed typos in the revised version. We also rechecked the text to ensure that there are no other typos and syntax errors.

- - - - - - - - - - - -

Definition 1 should probably mention that e sub 1 through e sub n are drawn from O.

An entity is tuple < v_1, v_2, …, v_k > and it provides a value for each attribute of the entity set, i.e., v_i gives the value for attribute n_i. Each v_i can be either an atomic value from V or an entity identifier from O. Having entity identifiers as values allows relationships between the entities. For example, an article entity that include authors that are “references” to people entities. Thus, it’s the value of the entities that can be drawn from O (and not the entities themselves).

We thank the reviewer for pointing out that this was unclear. We have now revised the related parts for clarifying these issues. Please see the updated Definition 1 as well as the paragraphs following the particular definition.

- - - - - - - - - - - -

The numbers in should have commas instead of decimals.

We revised the manuscript (i.e., text, figures, and tables) and now use “,” to indicate the decimal place and a “.” to separate groups of thousands.