EMBench: Generating Entity-Related Benchmark Data

Ekaterini Ioannou[1] and Yannis Velegrakis[2]

[1] Technical University of Crete, Greece, `ioannou@softnet.tuc.gr`
[2] University of Trento, Italy, `velgias@disi.unitn.eu`

Abstract. The *entity matching* task aims at identifying whether instances are referring to the same real world entity. It is considered as a fundamental task in data integration and cleaning techniques. More recently, the entity matching task has also become a vital part in techniques focusing on *entity search* and *entity evolution*. Unfortunately, the existing data sets and benchmarking systems are not able to cover the related evaluation requirements. In this demonstration, we present EMBench; a system for benchmarking entity matching, search or evolution systems in a generic, complete, and principled way. We will discuss the technical challenges for generating benchmark data for these tasks, the novelties of our system with respect to existing similar efforts, and explain how EMBench can be used for generating benchmarking data.

1 Introduction

The entity matching task aims at identifying instances representing the same real world entity, such as an author or a conference [6]. Existing matching approaches are typically based on some similarity function that measures syntactic and semantic proximity of two instances. Depending on the results of this comparison, it is decided whether the two instances are matching or not. More advance matching approaches exploit relationships between instances [1,2], the use of blocking for reducing the required processing time [7,8], and using information encoded in the available schemata [3,4].

Despite the many different techniques for entity matching there is no evaluation methodology that covers all the aspects of matching tasks or at least giving the user the ability to test the aspects of interest. Most matching techniques have followed their own ad-hoc evaluation approach, tailored to their own specific goals. Comparison among entity matching systems and selection of the best system for a specific task at hand is becoming a challenge. Developers can not easily test the new features of the products they develop against competitors, practitioners can not make informative choices for the most suitable tool to use, and researchers can neither compare the techniques they are developing against those already existing, neither identify existing limitations that can serve as potential research directions.

In this demonstration, we will present and discuss the EMBench system for benchmarking entity matching systems in a generic, complete, and principled way [5]. The system provides a series of scenarios that cover the majority of the matching situations that are met in practice and which the existing matching systems are expected to support. EMBench is fully configurable and allows the dynamic (i.e., 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. The fact that the entity matching scenarios are created in a principled way, allows the identification of the actual type of heterogeneities that the under evaluation matching system does not support. This is a fundamental difference from other existing benchmark or competition-based approaches that come with a static set of cases that do not always apply in all the real world scenarios.

The following URL provides an online access to the system as well as the sources code and binary file, and the details can be found in the full version of the paper [5]:

```
http://db.disi.unitn.eu/pages/EMBench/
```

2 Entity Matching Scenarios

To generate test cases in a systematic way, we introduce the notion of a *scenario*. A scenario is a tuple $\langle e_n, I, e_r\rangle$ where e_n is an entity, I is an entity collection, and e_r an entity from I referred to as the *ground truth*. The scenario is said to be *successfully executed* by an entity matching technique if the technique returns the entity e_r as a response when provided as input the pair $\langle e_n, I\rangle$, i.e., returns e_r as the best match of e_n in the entity collection I.

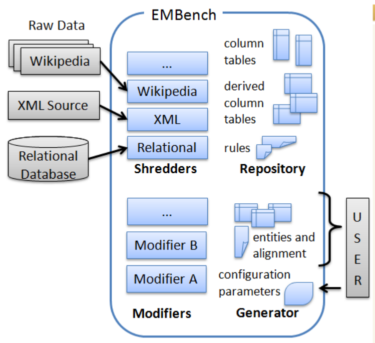
EMBench creates a scenario by first selecting an entity e_r from the collection I and a series of modifiers f_1, $f_2, \ldots, f_n$. It then applies the modifiers over the selected entity, i.e., $e_r \xrightarrow{f_1} e_1, \xrightarrow{f_2} \ldots \xrightarrow{f_n} e_n$, and generates as a scenario the triple $\langle e_n, I, e_r\rangle$.

Each modifier reflects a specific heterogeneity that matching tasks are frequently requested to detect. An example of such a category of modifiers is *Syntactic Variations* and it includes modifiers such as misspellings, word permutations, aliases, abbreviations, and homonymity. *Structural Variations* is another category of modifiers. These modifiers exploit variations on the attribute level. For example, we might have entities that use a set of attributes to describe some information while others entities use just one attribute (e.g., human names might be split into first name and last name, or may not). Another category is *Entity Evolution* simulating scenarios in which the entities have modifications due to time. These modifications can be, for example, changes in the attribute values, elimination of attributes, or addition of new attributes.

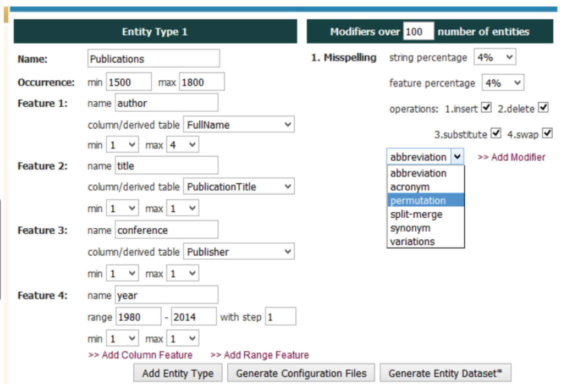
An important feature of the system is that the data engineer that created the scenarios can choose not only the case but also the size of data instance to generate. In this way the matching algorithm is tested not only in terms of effectiveness but also in terms of efficiency (scalability).

3 The EMBench System

Figure 1(a) illustrates the architecture of the system. As shown, EMBench maintains a *Repository* that contains the data used during the collection generation. The synthetic data generated by EMBench are not completely random strings but are based on real world values following realistic scenarios. This is achieved by *Shredders*, i.e., software components that receive a source and shreds it into a series of *Column Tables*. The system incorporates general purpose shredders (e.g., relational databases, XML files) as well as shredders specifically designed for popular systems (e.g., Wikipedia, DBPedia, Amazon, IMDb, DBLP, OKKAM).



(a)



Entity Type 1

Name: Publications

Occurrence: min 1500 max 1800

Feature 1: name author
column/derived table FullName
min 1 max 4

Feature 2: name title
column/derived table PublicationTitle
min 1 max 1

Feature 3: name conference
column/derived table Publisher
min 1 max 1

Feature 4: name year
range 1980 - 2014 with step 1
min 1 max 1

>> Add Column Feature >> Add Range Feature

Modifiers over 100 **number of entities**

1. Misspelling string percentage 4%
feature percentage 4%
operations: 1.insert ✓ 2.delete ✓
3.substitute ✓ 4.swap ✓

abbreviation >> Add Modifier
abbreviation
acronym
permutation
split-merge
synonym
variations

(b)

Fig. 1. (a) An illustration of the EMBench's architecture. (b) A screenshot of the EMBench GUI for creating an entity collection.

The system also supports cleaning the repetitive, overlapping, or complementary information in the resulted column tables. Among the processes incorporated for this functionality, we have 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, referred to as the *Derived Column Tables*. Note that a derived column table may be created through an identify function rule, meaning that it is considered a derived table without any modification.

There is no need to shred the original sources or to create the derived column tables every time the benchmark needs to run. Once they are created, they remain in the repository until deleted or overwritten. Actually, the current version of EMBench contains a *Default Data Collection* that is considered sufficient for the realistic evaluation of matching tasks. For instance, it contains 49299 feminine names, 74079 masculine names, 4003 diseases, 84847 companies, and 11817 universities.

The *Entity Generator* creates an entity collection I of N entities by constructing an entity for every tuple of the populated table R. Each such entity will have M attributes, one for every of the M attributes of the table R. EMBench provides two options for selecting the N values from the derived column table: (i) a random selection with or without repetitions, and (ii) select values following the Zipfian distribution.

As mentioned in Section 2, EMBench includes a set of *Entity Modifiers* that modify in various ways the data of an entity collection and construct a new entity collection with a high degree of heterogeneity. The used modifiers, their order and the modification degree is something that is specified by a set of configuration parameters. These parameters have some default values in the system but can also be modified by the user.

Overall, EMBench offers three main functionalities. The first is to create a source repository by importing data using shredders. The second is to generate entity collections using the data from the source repository. The third functionality is to evaluate matching algorithms. To ease the use of these functionalities, EMBench is in general fully parametrized through a configuration file. In addition, EMBench is accompanied with a user interface that allows the specification of the parameters that build the configuration file on-the-fly and run EMBench (shown in Figure 1(b)).

4 Demonstration Highlights

In the proposed demonstration we will discuss with the audience the functionalities and abilities of EMBench. We will particularly focus on the following four parts.

A. Using EMBench. During the first part we will discuss the two available ways for using EMBench. The first is the usage through a configuration file, which allows providing a description of the functionallities that can be executed, for example which shredders to run, or which matching tasks to evaluate. The second usage is through the EMBench GUI (shown in Figure 1(b)). The GUI provides an alternative mechanism for selecting EMBench's configuration and executing the functionalities of EMBench.

B. Repository and Default Data Collection. The second part of the demonstration focuses on the repository. We will present the data included in the default data collection, and illustrate how to use existing EMBench shredders for importing additional data. We will also explain how to create, configure, and execute new shredders.

C. Creating Entity Collections. In the subsequent part of the demonstration we will present the creation of collections. This includes describing the schema for the entities to be generated (e.g., maximum number of entity attributes, value distribution, column tables). It also includes the specification and configuration of the modifiers.

D. Evaluating Algorithms using EMBench. The last part of the demonstration focuses on illustrating how EMBench can be used for evaluating algorithms. We will discuss the metrics that are currently incorporated in EMBench and how additional ones can be easily implemented. Furthermore, we will present and illustrate the supported matching-related tasks (i.e., one-to-one matching and blocking).

The demonstration is intended for researchers and practitioners alike. The conference participant will have the opportunity to understand the principles behind the benchmark. This will help the participants in evaluating and testing new matching systems in order to select the one that bests fits a task at hand, but will also give valuable insight on how to design and improve matching systems.

References

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

[2] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.

[3] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, 2007.

[4] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Semantic Interoperability and Integration*, 2005.

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

[6] E. Ioannou and S. Staworko. Management of inconsistencies in data integration. In *Data Exchange, Information, and Streams*, 2013.

[7] G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *WSDM*, 2012.

[8] S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, 2009.