NADEEF/ER: Generic and Interactive Entity Resolution

Ahmed Elmagarmid[1] Ihab F. Ilyas[2] Mourad Ouzzani[1]
Jorge-Arnulfo Quiané-Ruiz[1] Nan Tang[1] Si Yin[1]

[1]Qatar Computing Research Institute [2]University of Waterloo
{aelmagarmid, mouzzani, jquianeruiz, ntang, siyin}@qf.org.qa ilyas@uwaterloo.ca

ABSTRACT

Entity resolution (ER), the process of identifying and eventually merging records that refer to the same real-world entities, is an important and long-standing problem. We present NADEEF/ER, a generic and interactive entity resolution system, which is built as an extension over our open-source generalized data cleaning system NADEEF. NADEEF/ER provides a rich programming interface for manipulating entities, which allows generic, efficient and extensible ER. In this demo, users will have the opportunity to experience the following features: (1) *Easy specification* – Users can easily define ER rules with a browser-based specification, which will then be automatically transformed to various functions, treated as *black-boxes* by NADEEF; (2) *Generality and extensibility* – Users can customize their ER rules by refining and fine-tuning the above functions to achieve both effective and efficient ER solutions; (3) *Interactivity* – We also extended the existing NADEEF dashboard with summarization and clustering techniques to facilitate understanding problems faced by the ER process as well as to allow users to influence resolution decisions.

Categories and Subject Descriptors

H.2.m [**Database Management**]: Miscellaneous - Data cleaning

Keywords

NADEEF, Entity resolution, Generic, Interactive

1. INTRODUCTION

Entity resolution (ER) (*a.k.a.* record linkage and deduplication) is a well known problem that arises in many applications. The goal is to identify and link different representations (*e.g.,* records in a relational database) of the same real world entity. A large chunk of research has been dedicated to tackle different issues related to ER such as using entity features and relational features for matching, leveraging machine learning techniques, combining crowd sourcing and active learning, taking into different constraints on the data during ER, and so on (see [8] for a tutorial).

ER is usually computationally expensive, since it may require comparing all possible pairs of entities. Multiple techniques such as blocking, sorted neighborhood, clustering, and canopies (see [6] for a survey) have been proposed. In addition, parallelization and Hadoop-based solutions have also been studied to further improve efficiency, *e.g.,* Dedoop [9]. Crowd has been used to improve effectiveness [12]. There have been also some efforts in supporting generality for ER. For example, Swoosh [1] proposes a generic approach where the users implement two user defined functions (UDFs): match() and merge(). Swoosh would then take care of how to invoke these two UDFs.

Despite all of these rich contributions, allowing easy *user-tunable* parameters as well as providing *generic* interfaces for users to implement application specific functions remain hard to achieve in one single system.

We recently presented NADEEF, an open-source[1] data cleaning system [3]. NADEEF distinguishes between a programming interface and a core to achieve generality and extensibility for data cleaning. We demonstrated these two features in [4]. This work inspired us in our solution to the above stated ER issues: *How to allow both easy user-tunable parameters and generic interfaces for ER?* In this demo, we present NADEEF/ER, a generic and interactive ER system. NADEEF/ER, which is built on top of NADEEF, supports a large variety of ER applications by providing a rich programming interface and an interactive dashboard. The demo will demonstrate the following three key features:

(1) *Easy specification.* NADEEF/ER provides a succinct GUI that allows users to declare advanced ER rules. NADEEF/ER, in turn, automatically transforms these rules into various UDFs, following NADEEF model, for execution.

(2) *Generality and extensibility.* Users can modify or override any NADEEF/ER interface functions in order to manipulate entities more effectively and efficiently. For example, users might provide their own matching, merging, blocking, and clustering functionalities. This offers freedom to fully customize NADEEF/ER for specific ER applications. Users can perform any of these code editing/checking via the GUI provided by NADEEF/ER.

(3) *Interactivity.* NADEEF/ER provides a dashboard where users can better visualize duplicated entities. This will not

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2594511.

[1]https://github.com/Qatar-Computing-Research-Institute/NADEEF

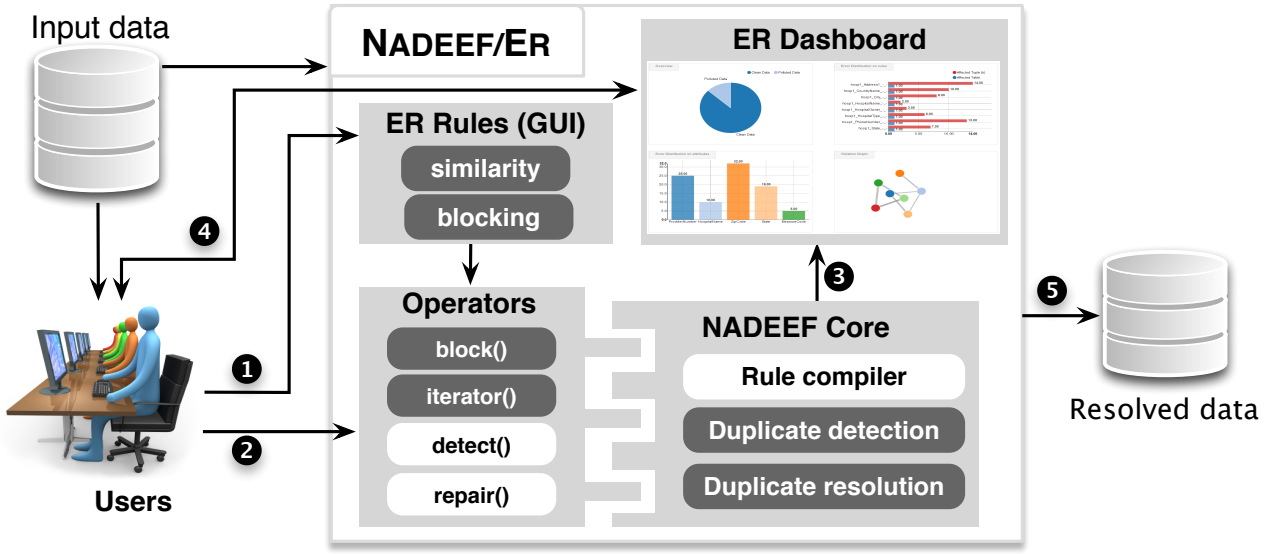

Figure 1: Architecture of NADEEF/ER

only help users to understand resolution problems, but also permit users to influence resolution decisions.

Related work. Matching techniques (see [6] for a more comprehensive survey) are a crucial tool for ER. These techniques range from character-based similarity metrics (*e.g.,* edit distance, Jaro distance, Q-gram distance) and token-based similarity metrics (*e.g., tf.idf*) to phonetic similarity metrics (*e.g.,* Soundex). In NADEEF/ER, we use the SimMetrics library[2] that supports many of these techniques. Moreover, using NADEEF/ER generic interface, users can implement other functions for numeric metrics and probabilistic matching models (*e.g.,* Bayes decision rules).

Achieving efficiency in ER aims to reduce the number of entity comparisons and has been tackled with techniques such as blocking, sorted neighborhood, clustering and canopies. NADEEF/ER supports these techniques through two functions, block() and iterator(). The former partitions entities into blocks, while the latter specifies how entities are actually compared within a block. Other researchers have also studied different indexing techniques for improving efficiency (*e.g.,* inverted indices, suffix arrays, see [2] for a comprehensive study). Supporting these indices is not the focus of NADEEF/ER.

There have also been many ER systems in the past. Some provide a declarative language (*e.g.,* AJAX [7]) or a GUI for rich user-tunable parameters (*e.g.,* TAILOR [5], Febrl and FRIL). Others target generic ER solutions *e.g.,* Swoosh [1]. In contrast, NADEEF/ER aims at a generic yet efficient system, by providing various functions to customize the system, to achieve both effective (as match() and merge() in Swoosh) and efficient (via block() and iterator()) ER solutions.

Another line of work is to learn ER rules with training datasets (*e.g.,* ALIAS [11]). These are essentially orthogonal, but complementary, to NADEEF/ER.

2. NADEEF/ER ARCHITECTURE

In this section, we discuss the architecture of NADEEF/ER (Fig. 1), an ER system built on top of NADEEF [3]. NADEEF/ER consists of the following components: a GUI for defining ER rules, a programming interface for defining data quality rules, NADEEF core for duplicate detection and resolution, and an ER dashboard for user interaction.

NADEEF/ER works as follows. (1) Users specify ER rules via a GUI. (2) User-specified ER rules via the GUI will be automatically transformed into data quality rules, expressed in a set of basic functions: block(), iterator(), detect(), and repair(). Users can also customize these rules by refining any of these four functions. (3) NADEEF core then compiles these rules and uses them to detect duplicated entities and eventually merge them. NADEEF treats these four functions as black-boxes. (4) Users can interact with the dashboard to explore and visualize information for duplicated entities. This allows users to refine ER rules as in steps (1) and (2), or to influence resolution decisions (by confirming duplicates/non-duplicates). (5) After iterations of the above process, NADEEF returns all detected duplicates, or produces a duplicate-free dataset if a repair() function for merging duplicates is provided by the users.

ER rules. This component collects user-specified ER rules via a GUI, as shown in Fig. 2. Given two tables (which might be the same table), users can drag lines to define attributes to be compared and define the similarity functions as well as corresponding thresholds. For each ER rule, the users can also specify the blocking strategy along with the generation function for its blocking key. NADEEF/ER will then automatically transform rules into four basic functions (or operators), as we explain below.

Operators. The unified programming interface for a data quality rule in NADEEF is as follows:

```
class Rule {
 Collection⟨Table⟩ block(Collection⟨Table⟩ tb);
 void iterator(Collection⟨Table⟩ tbs, IteratorStream⟨TuplePair⟩ is);
 Collection⟨Violation⟩ detect(TuplePair tp);
 Collection⟨Fix⟩ repair(Violation v);
}
```

While one can use these four functions for generalized data cleaning tasks, in the following, we explain them in the context of entity resolution only.

(1) block() takes a set of tables as input and returns partitions for each table. One can use this function to implement

[2] https://github.com/Simmetrics/simmetrics

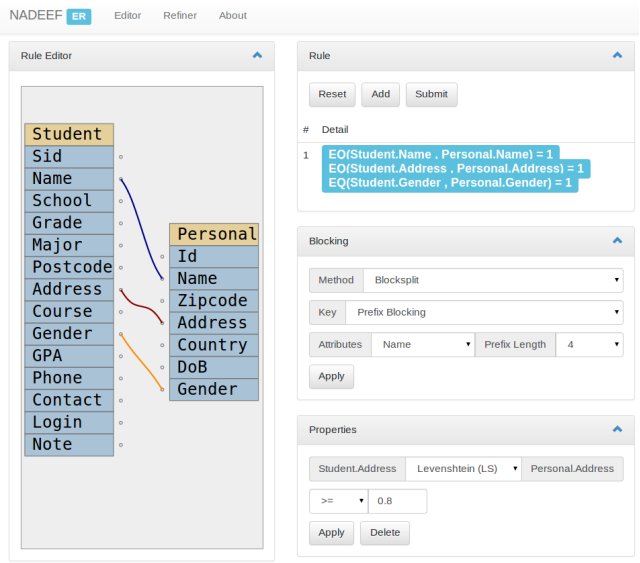

Figure 2: ER rule specification

most blocking/clustering algorithms, allowing a more generalized blocking strategy beyond key-based blocking.

(2) iterator() takes a set of tables which could be base tables or blocks from the function block() as input and outputs pairs of tuples. Intuitively, this function allows users to implement smart ways to populate pairs of entities to be compared, *e.g.,* sorted neighborhood, in contrast to enumerating all pairs of entities within a given block.

(3) detect() takes a pair of tuples as input. If they are duplicates, it will return both tuples, referred to as a *Violation*. Otherwise, it returns an empty result, indicating that the two input tuples are not duplicates.

(4) repair() takes a pair of duplicated tuples (*i.e.,* a *Violation*) as input and specifies how to merge them.

When users specify ER rules via GUI, NADEEF/ER will transform them to the above four functions as follows. The similarity metrics on how to decide whether two entities are duplicates will be transformed into the detect() function. The blocking strategy will be transformed into the block() function. The other two functions, repair() (for merging entities) and iterator() (for smartly populating entity pairs), will have to be implemented by users. In Section 3, we show how these can be easily done.

NADEEF core. This module takes all user provided rules as input and treats all UDFs as *black-boxes*. NADEEF is responsible for efficiently invoking these black-boxes to identify and resolve duplicates.

ER dashboard. The ER dashboard (by extending NADEEF dashboard [4]) helps users understand the results of the ER process, by showing summarized, sampled, ranked and clustered duplicates. This facilitates the solicitation of user feedback for refining ER rules.

Implementation details. We have developed NADEEF/ER as a single-page application (SPA) with modern web application stack. The HTML5 client provides GUI and visualization to help users build ER rules, set a blocking strategy, and view the output matrix. Client requests for rule execution are processed asynchronously by a thin Java based server layer. The server wraps the requests and sends them to the NADEEF service for execution. NADEEF/ER's architecture

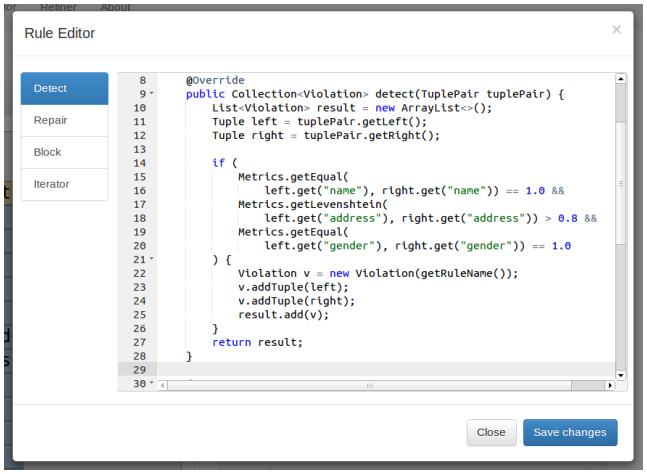

Figure 3: ER rule editor

is designed to be scalable to process multiple requests and multiple users at the same time. The communication among different services is built using Apache Thrift infrastructure, which is able to make requests execution scalable to many compute nodes.

3. DEMONSTRATION OVERVIEW

In this demonstration, we will use real-life datasets to give users the opportunity to experience the features provided by NADEEF/ER. Notice that this demonstration significantly differs from the one presented in [4] in two main aspects: (1) we will show the audience how easily they can specify ER rules via the NADEEF/ER GUI; and (2) we will also show the audience how to modify the four provided functions for customized ER rules via a web-browser. Additionally, we will show: (3) how the data quality dashboard can help users understand duplicate data, and how users can interact with NADEEF/ER to further refine ER rules; and (4) how existing ER algorithms can be easily implemented in NADEEF/ER.

(1) ER rule specification. Figure 2 displays the NADEEF/ER GUI for specifying ER rules. For each ER rule, the audience will be able to specify, by dragging lines between two attributes, the attributes to compare and the similarity (or equality) metrics to employ. Moreover, the audience can select pre-defined blocking functions for each ER rule. After specifying the parameters of one ER rule, the audience can press the "Add" button (on the top-right of Fig. 2) to register this rule in NADEEF/ER. Similarly, the audience will be able to specify more ER rules. Each ER rule will be automatically transformed into the four functions provided by NADEEF/ER.

(2) ER rule customization editor. We will demonstrate how to customize an ER rule by modifying any of the four functions provided by NADEEF/ER. For each ER rule, we will invite the audience to select one of these functions to revise, as shown in Fig. 3. After revision, NADEEF/ER check and highlight syntactic errors in the audience's provided code. If the code is syntactically correct, the audience can submit their changes by pressing the "Save changes" button.

Notice that in the context of ER, one can consider NADEEF detect() (resp. repair ()) having the same functionality as Swoosh match() (resp. merge()) function. In general, how-

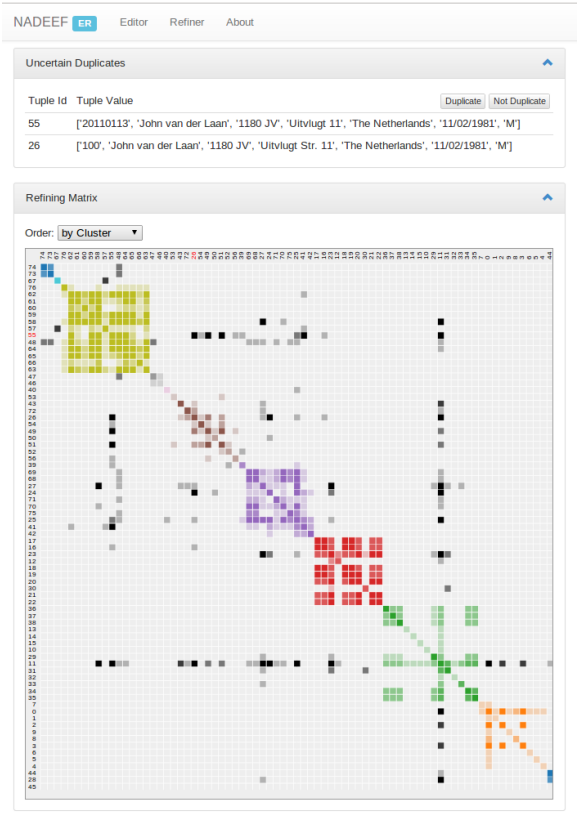

Figure 4: Co-occurrence matrix of duplicates

ever, the detect() (resp. repair ()) function is more general than their Swoosh counterparts (see [3] for more details).

Moreover, we will demonstrate two functions that are not present, neither in NADEEF [3,4] nor Swoosh [1]: the block() and iterator() functions.

- block() function: besides standard blocking functions, we will show how users can implement clustering-based method *canopies* [10].
- iterator() function: we will show how users can implement a sorted neighborhood approach, assuming duplicated records will be close at a certain sorted order. Beyond this, we also want to show that for a specific application and under certain assumptions, users can write a better iterator() function that would help reduce the number of entity pairs to be examined.

(3) Dashboard and user interaction. NADEEF/ER comes with a dashboard to allows user to better understand ER results. This dashboard differs from the dashboard presented in [4] as it provides a new graph that is especially designed for entity resolution: the *co-occurrence matrix of duplicates*. But also, NADEEF/ER dashboard inherits the existing visualization tools from NADEEF: a pie chart to show the overall number of tuples that have duplicates (*overview*); a bar chart to show the number of duplicates for each ER rule (*duplicate distribution*); a bar chart to show, for each attribute, the number of different values involved in duplicate entities (*duplicate distribution on attributes*); and a graph to visualize tuples with duplicates, where each node is a tuple and an edge between two tuples means that they are duplicates (*violation graph*).

Figure 4 depicts the *co-occurrence matrix of duplicates*. Each coloured cell in the matrix diagram represents two entities that have been detected as duplicates; darker cells indicate duplicates that co-occurred more frequently, *i.e.,* detected as duplicates by more ER rules. Clustering algorithms are used to group these cells. Cells with the same color indicate that they belong to the same cluster, *e.g.,* all yellow cells. A grey cell means that this cell is an island, *i.e.,* not in any cluster. By clicking a cell, NADEEF/ER provides its details, shown on top of Fig. 4. The users can investigate and decide to include/exclude them as duplicates. Based on user decisions, NADEEF/ER automatically adds special conditions to refine corresponding functions (operators). With the drop-down menu, users can select how to re-order the matrix, by *e.g.,* sorting on Name attribute to further explore the data. It is worth noting that this matrix only shows tuples that are detected as duplicates. When the number of (possible) duplicated tuples is large, partitioning techniques are used to show this matrix. It is also worth noting that the result about including/excluding uncertain duplicates can be used as training datasets for learning-based algorithms.

(4) Extensibility. Besides showing that NADEEF/ER can support generic entity matching and various optimization techniques, we will also show how various algorithms can be easily implemented in NADEEF/ER.

Summary. This demonstration aims at exhibiting the features of NADEEF/ER. We focus on: (*i*) A GUI that provides a rich set of user-tunable parameters for defining ER rules ((1) above); (*ii*) The NADEEF programming interfaces that helps users specify highly customized rules ((2) above); (*iii*) The data quality dashboard that can help users effectively interact with the system to refine ER rules((3) above); (*iv*) NADEEF/ER is generic and extensible ((4) above); (*v*) A real application exploiting all features of NADEEF/ER.

4. REFERENCES

[1] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1), 2009.

[2] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9), 2012.

[3] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, 2013.

[4] A. Ebaid, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. NADEEF: A generalized data cleaning system. *PVLDB*, 2013.

[5] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. Tailor: A record linkage tool box. In *ICDE*, 2002.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 2007.

[7] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In *SIGMOD*, 2000.

[8] L. Getoor and A. Machanavajjhala. Entity resolution for big data. In *KDD*, 2013.

[9] L. Kolb, A. Thor, and E. Rahm. Dedoop: Efficient deduplication with hadoop. *PVLDB*, 2012.

[10] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, 2000.

[11] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, 2002.

[12] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 2012.