Enhancing Loosely Schema-aware Entity Resolution with User Interaction

Giovanni Simonini, Luca Gagliardelli, Song Zhu, Sonia Bergamaschi
Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
Italy
Email: {giovanni.simonini;luca.gagliardelli;song.zhu;sonia.bergamaschi}@unimore.it

***Abstract*—Entity Resolution (ER) is a fundamental task of data integration: it identifies different representations (i.e., *profiles*) of the same real-world entity in databases. To compare all possible profile pairs through an ER algorithm has a quadratic complexity. *Blocking* is commonly employed to avoid that: profiles are grouped into *blocks* according to some features, and ER is performed only for entities of the same block. Yet, devising blocking criteria and ER algorithms for data with highly schema heterogeneity is a difficult and error-prone task calling for automatic methods and debugging tools.**

In our previous work, we presented *Blast*, an ER system that can scale practitioners' favorite Entity Resolution algorithms. In current version, *Blast* has been devised to take full advantage of parallel and distributed computation as well (running on top of Apache Spark). It implements the state-of-the-art unsupervised blocking method based on automatically extracted *loose* schema information. We build on top of blast a GUI (Graphic User Interface), which allows: *(i)* to visualize, understand, and (optionally) manually modify the loose schema information automatically extracted (i.e., injecting user's knowledge in the system); *(ii)* to retrieve *resolved* entities through a free-text search box, and to visualize the process that lead to that result (i.e., the *provenance*). Experimental results on real-world datasets show that these two functionalities can significantly enhance Entity Resolution results.

***Index Terms*—Entity Resolution; Data Integration; Data Cleaning; Big Data**

I. Introduction

With the increasing of the data (i.e. Big Data), data analysis now drive every aspect of modern society and different tools were proposed in order to integrate, search over and analyze these huge amount of data [1] [2] [3] [4]. In this scenario Entity Resolution (ER) plays a central role, since it is a crucial and expensive task in data integration. ER is the task of identifying different representations (called *profiles*) of the same real-world entity in data sources. The naïve all-pairs comparison solution of ER is unmanageable with large databases, thus *blocking* techniques are typically employed to group similar records and limit the actual comparisons among those records that appear together in *block*. For example, given the dataset in Figure 1a (which is about people information), the initials of the name attribute values might be employed as *keys* to index profiles into blocks. Hence, all the profiles that have the same initials (e.g., p_2 and p_3, which have "*a.l.*" as initials) are placed together in block to be compared. However, this blocking criterion fails to place together the matching profiles p_1 (initials: "*b.d.*") and p_3 (initials: "*r.d.*").

A. Background

In a real world scenario, to identify a blocking criterion (a.k.a. the *blocking key*) yielding high recall and precision is a difficult and critical task [5], [6]. In fact, these *schema-aware* techniques suffers of two main drawbacks when dealing with big data: *(i)* they require schema alignment, which may be very hard to achieve with data characterized by high heterogeneity of schemata and formats; *(ii)* they require either labeled data and classification algorithms, or domain experts to select the attributes to combine. To overcome these limitations, the *schema-agnostic* approach [7] has been proposed: it renounces to exploit schema-information, treating profiles as *bag of words*. For instance, schema-agnostic Standard Blocking considers each token appearing in the value of a record as a blocking key, regardless of the attribute in which it appears (example in Figure 1b). This allows to minimize the number of false negative due to the mismatches of schema attributes or blocking key extraction.

However, *schema-agnostic* methods achieves generally low precision. So, to mitigate this problem, they are typically coupled with *meta-blocking* [7], [8], [9]. The goal of meta-blocking is to restructure a blocking collection by removing least promising comparisons. This is achieved in the following way: profiles and comparisons are represented as nodes and edges of a graph, respectively; then, each edge is weighted on the basis of the co-occurrence of its adjacent nodes in the original blocks; finally, a graph pruning algorithm retains only the highest weighted edges involving each node.

Figure 1 illustrates a toy example of schema-agnostic blocking and meta-blocking.

B. Our Approach

We have proposed *Blast* [10], a novel approach to meta-blocking, which goes beyond the *bag-of-word* model employed by the current state-of-the-art. *Blast* introduces the idea of *loose schema information* extracted directly from the data and exploited for both blocking and meta-blocking. Loose schema information is composed of: *(i)* the *attribute partitioning*, which is a surrogate of the schema matching

978-1-5386-7879-4/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCS.2018.00138

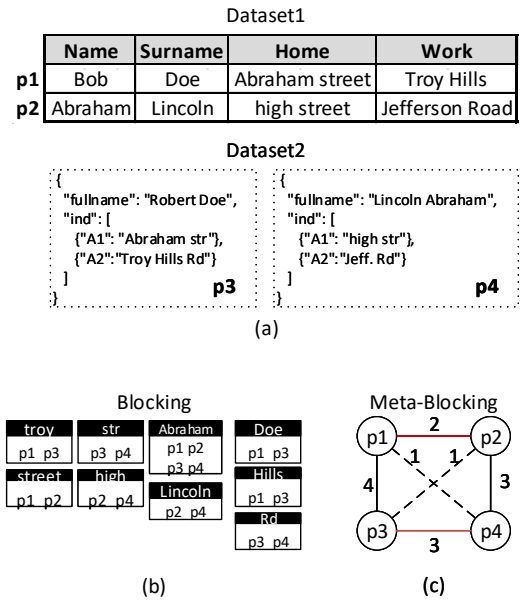

Fig. 1: *Example of schema-agnostic (meta-)blocking process.* (a) A set of profiles P from an imaginary data lake. (b) The set of blocks B derived applying schema-agnostic Standard Blocking on P (i.e., each token is a blocking key). (c) The *blocking graph* derived from the blocks of B, and the effect of the pruning algorithm: dashed lines are the removed comparisons, while red lines are the incorrectly retained ones. For the sake of the example: each edge is weighted counting the blocks that its adjacent profiles have in common, and is retained if its weigh is above the average (more complex weighting and pruning strategies are actually employed [10]).

employed to enhance a schema-agnostic blocking method; and *(ii)* the *attribute partition entropy*, which captures the importance of an attribute partition as blocking key.

The first basic intuition is that similar attributes will have similar values if matching profiles are present in the dataset; thus, this information can be exploited to produce blocking keys. The second basic intuition is that the more *unpredictable* are the values of an attribute, the lower is the probability that two profile will have the same value for that particular attribute by chance. (The unpredictability of an attribute is measured through entropy.)

Figure 2 illustrates the effect of the loose schema information on the toy example of Figure 1.

C. The Blast System.

Blast[1] is a complete ER System built around the parallel and distributed implementation of *Blast* [10] for Apache Spark. Here we present the *Blast* User Interface, which provides functionalities that support practitioners in devising and debugging an ER algorithm/workflow.

[1]https://github.com/Gaglia88/sparker

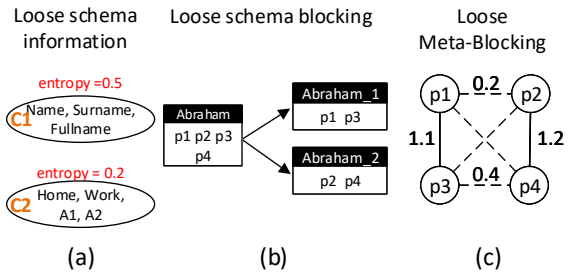

Fig. 2: *Example of (meta-)blocking process employing loose schema information.* (a) The loose schema information extracted from the dataset in Figure 1a: the attribute partitioning and the attribute partition entropies. Attributes referring to the name of a person are grouped together, while all the other attributes are grouped in another cluster (say that there is no pair of attributes that have similar values at some extent). (b) The attribute partitioning is employed to enhance Standard Blocking: by using this information the token "Abraham" is split into two tokens (i.e., disambiguating "Abraham" as a street name, and "Abraham" as person name). (c) The attribute partition entropies are employed as multiplicative factors of edge weights of Figure 1c to capture the importance of the co-occurrence of two profiles in the blocks. This affects the meta-blocking by helping to remove more superfluous comparisons than those removed by the schema-agnostic blocking (the two incorrectly retained edges of Figure 1c are now removed).

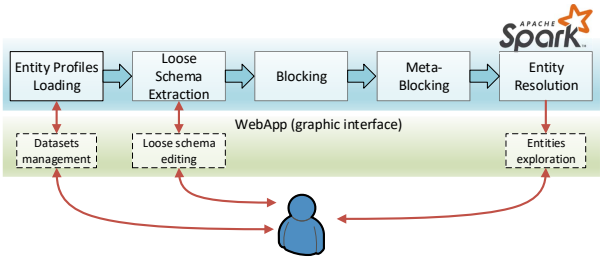

Fig. 3: *Blast* logical architecture.

In the following Section II, we present the modules that compose *Blast*, and the novel GUI. Finally, in Section III we present experimental results showing how ER results can be easily improved employing the Blast GUI.

II. Architecture

Blast is organized in modules, each performing a specific task, devised to be parallelizable on Apache Spark. These modules are combined together in order to perform the ER process, as outlined in Figure 3: firstly, the set of profiles is

loaded into a Spark *RDD*[2]; then, the loose-schema information is extracted and employed by the blocking and meta-blocking methods; finally, the candidate set of matching profiles are actually compared (i.e., *resolved*) using an UDF (User Defined Function) that takes as input two entity profiles and decides whether they are matching or not, i.e., an ER algorithm (e.g., by calculating a similarity score and a similarity threshold).

A. Loose Schema Information Extractor

The loose-schema information automatically extracted is composed by: the *attribute partitioning* and the *attribute partition entropies*.

Attribute Partitioning. The aim of this operation is to partition together attributes that have similar values. To do that in a fast and efficient manner, an algorithm based on *Locality-sensitive Hashing* (LSH) is employed.

In details: firstly, LSH is applied to the attributes values, in order to group them, according to their similarity. These groups are overlapping, i.e., each attribute can co-occur with different other attributes. Then, for each attribute only the most similar one is kept in order to obtain pairs of similar attributes. Finally, a transitive closure algorithm is applied, to guarantee that each group of attributes, which are transitively obtained, represent an *attribute partitioning*. All the attributes that have not been partitioned, with any other attribute are put together in a *blob* partition

Attribute Partition Entropy. Once the attribute partitions have been generated, each partition is weighted to quantify the relevance of blocking keys derived from the values of its attributes. The Shannon entropy is employed to assess this relevance. The intuition is that the more informative an attribute is, the more effective are the blocking keys extracted from it. Since each partition is composed by different attributes, their entropies are aggregated, and the resulting value is assigned as weight of the partition.

GUI for Loose Schema Information Exploration and Editing. In [10], we have shown that by employing the *Blast* approach for blocking and meta-blocking no human intervention is required to achieve high quality results. Yet, domain knowledge can be employed to refine the automatically extracted loose schema information, so to enhance even further the final ER results. Thus, *Blast* provides an intuitive user interface to visualize and modify the loose schema information. The GUI also provides free-text search functionalities to conveniently retrieve desired attributes by their names, and functionalities for sorting and filtering them by their entropies.

B. Blocking and Meta-Blocking

The blocking is performed using schema-agnostic blocking techniques (e.g. Standard Blocking), but considering the loose schema information. This allows to disambiguate blocking keys according to the attribute group from which they are derived. An example is provided in Figure 2b. Each block inherit the weight of the partition to which its attribute belongs.

Meta-blocking is performed taking into account the weight assigned to each block, in particular, the χ^2 test is employed to measure the strength of co-occurrences. Then, each edge weight is re-weighted according to the weight associated to the block that generates it. For example, in Figure 2c the weight of the edge p_1-p_2 is 0.2 because they co-occurs in the "street" block that belongs from the $C2$ partition, since its a value of the "Address" attribute. Also, this example gives an intuition on how the partitions weights highlights the correct edges. It is indeed possible to see that the higher weight of $C1$ increases the strength of the edges that connects the right entity profiles (p_1-p_3, p_2-p_4), and lowers the weight of the wrong edges (p_1-p_2, p_3-p_4), that will be discarded during the pruning phase.

The meta-blocking algorithm for parallel meta-blocking is inspired by the broadcast join: it partitions the nodes of the blocking graph and sends in broadcast (i.e., to each partition) all the information needed to materialize the neighborhood of each node one at a time. Once the neighborhood of a node is materialized, the pruning function is applied.

C. Entity Resolution algorithm.

The output of a (meta-)blocking method is a list of profile pairs, which are candidate matches. An ER algorithm is then required to decide whether two profiles are matching or not, e.g.: by employing similarity functions [11] combined with hand-tuned thresholds; by asking to the human judges in a *crowd sourcing* setting; or by training a classification algorithm when labeled data are available.

GUI for ER Result Exploration. *Blast* allows to debug entity resolution results by selecting set of (possibly non-correctly) resolved entities and visualize their *provenance* through the workflow, i.e., how they ended up to be (or not) matches. Such a functionality can help in devising/debugging ER algorithms and in refining the loose schema information (by means of the functionality introduced in the previous section).

III. Experiments

To showcase our system we use the `movies` dataset, which compares movies extracted from imdb.com and dbpedia.org. It contains 48,000 profiles described by two different schemas, which cannot be perfectly aligned. The dataset comes with a ground-truth that allows to analyze the meta-blocking performance in terms of `precision` and `recall`. Other three datasets are employed in this experiment[3]: `Walmart-Amazon`, `Google-Amazon`, and `Abt-Buy`. They cover a wide range of scenario (e.g., scientific papers, e-commerce products, and generic knowledge bases), have a huge variety of schemata (the largest datasets have thousands of different attributes), and volume (up to millions of entities). Yet, for sake of presentation, here we describe the experiment using the `movies` dataset.

[2]https://spark.apache.org

[3]Datasets characteristics: https://sourceforge.net/projects/sparker/files/

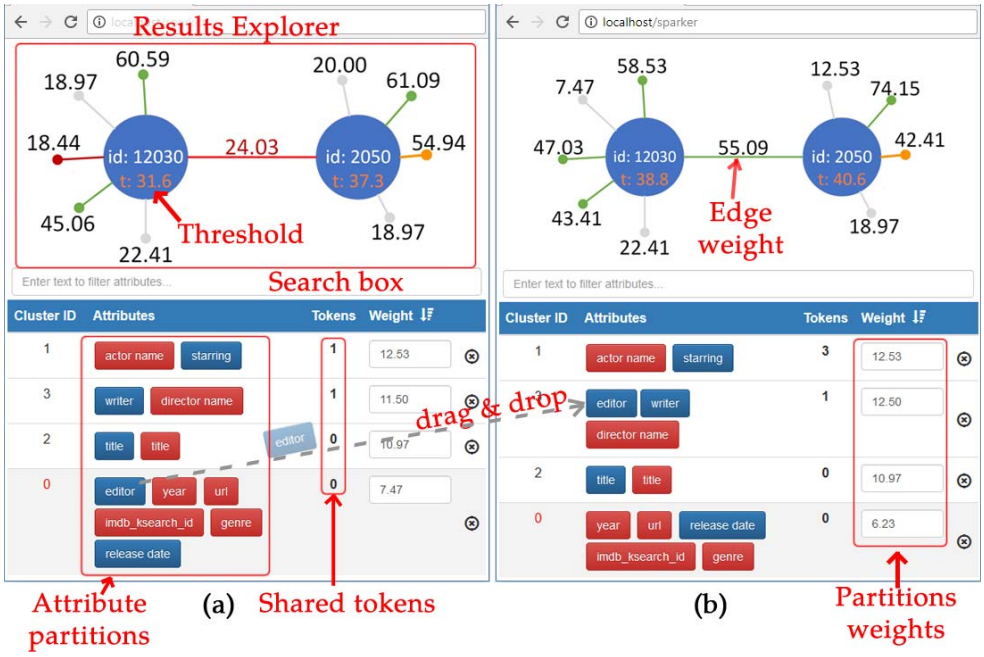

Fig. 4: *Blast* GUI screenshots.

We asked to 4 group of undergraduate and graduate CS students (3 student in each group; each group with a different dataset; no prior knowledge of the dataset schema): *(i)* to explore and modify the automatically extracted loose schema information; *(ii)* to analyze the results in order to improve the discovering of true matches. For brevity we show here only the most relevant phases, omitting the straightforward preliminary ones, such as: datasets management and loading, task settings, etc.

A. Exploring and Editing the Loose Schema Information

During the setups of a new task each user was able to choose the datasets on which operate and a pre-defined matching algorithm (based on string similarity measures). The system elaborates the datasets and generates the loose schema information, which is presented to the user through the GUI. For example, Figure 4b shows the attribute partitions automatically generated for the `movies` dataset. The user can edit the partition of attributes. For each partition, the attributes are highlighted with different colors according to which datasets they belong. The weight assigned to the attribute partition is shown as well: initially they are set equal to the corresponding attribute partition entropy. The interface also allows: to search for a specific partition/attribute by using the search box; to remove/create a partition (not shown in Figure 4); to edit the partition weights; and to drag&drop attributes from a partition to another.

The users were asked to launch different run editing of the loose schema information. This, in order to see each time how this affect the final result. In fact, the system provides all the details extracted from the executions logs (e.g. execution time of each phase, statistics on the datasets, etc.), and also a summary view through four charts: recall, precision, F1-score and the execution time (see Figure 4a).

B. Debugging Entity Resolution Workflows.

By employing a subset of labeled data (coming from the known ground-truth of the datasets), the users were asked to analyse missed matches. According to that, they were asked to inject useful knowledge by editing the loose schema information and to change employed ER algorithm, in order to improve the final result. This is possible by using the results explorer (e.g., by searching for a movie name among the incorrectly resolved ones), which shows how the profiles are connected in the meta-blocking graph and the choice made by the ER algorithm. An example is provided in Figure 4b: for each profile its pruning threshold is listed in orange, and all its edges with their weights are shown. (Recall that meta-blocking retains only edges above the threshold.) Grey edges are the correctly discarded ones (i.e., true negative); green edges are the correctly kept ones (i.e., true positive); red edges are the incorrectly discarded ones (i.e., false negative); finally, the orange edges are the incorrectly maintained ones (false positive). By changing the partition parameters, it is possible to see in real-time how the weights change. The number of tokens the two profiles share is listed for each attribute partition as well. Hence, following the example, users were asked to drag&drop attributes. For instance the editor attribute can be moved from the blob partition 0 (4a) to the attribute partition 1 (4b). By doing that, the edge weights change and the inspected match is now correctly retained (4b), since the editor of a

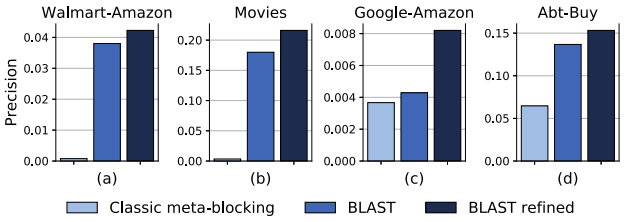

Fig. 5: Precision.

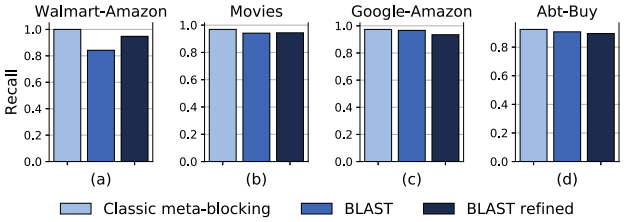

Fig. 6: Recall.

movie in the blue dataset (IMDB) sometimes matches director name of the red one (DBPedia).

C. Quantitative Results.

Figure 5 and Figure 6 report, respectively, the precision and recall of the results obtained by the 4 groups. The results has been obtained averaging the result of the components of each group. In the figures, the results obtained by the users, who interact with the *Blast* GUI (*Blast* refined) are compared to two baselines: *(i)* the classical Meta-blocking [7] and *(ii) Blast* (without user intervention). Overall, the results obtained by the users interacting with the GUI show almost the same recall, with a significant improve in precision.

REFERENCES

[1] F. Guerra, G. Simonini, and M. Vincini, "Supporting image search with tag clouds: a preliminary approach," *Advances in Multimedia*, vol. 2015, p. 4, 2015.

[2] S. Bergamaschi, F. Guerra, and G. Simonini, "Keyword search over relational databases: Issues, approaches and open challenges," in *Bridging Between Information Retrieval and Databases*. Springer, 2014, pp. 54–73.

[3] S. Bergamaschi, D. Ferrari, F. Guerra, G. Simonini, and Y. Velegrakis, "Providing insight into data source topics," *Journal on Data Semantics*, vol. 5, no. 4, pp. 211–228, 2016.

[4] G. Simonini and S. Zhu, "Big data exploration with faceted browsing," in *High Performance Computing & Simulation (HPCS), 2015 International Conference on*. IEEE, 2015, pp. 541–544.

[5] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *TKDE*, vol. 24, no. 9, pp. 1537–1555, 2012.

[6] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park, "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," *SIGMOD '17*, pp. 1431–1446, 2017.

[7] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl, "Meta-blocking: Taking entity resolutionto the next level," *TKDE*, vol. 26, no. 8, pp. 1946–1960, 2014.

[8] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas, "Parallel meta-blocking for scaling entity resolution over big heterogeneous data," *Inf. Syst.*, vol. 65, pp. 137–157, 2017.

[9] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi, "Schema-agnostic Progressive Entity Resolution." *ICDE '16*, 2016.

[10] G. Simonini, S. Bergamaschi, and H. V. Jagadish, "BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution." *PVLDB '16*, vol. 9, no. 12, pp. 1173–1184, 2016.

[11] F. Benedetti, D. Beneventano, S. Bergamaschi, and G. Simonini, "Computing inter-document similarity with context semantic analysis," *Inf. Syst.*, 2018, ISSN: 03064379.