SparkER: Scaling Entity Resolution in Spark

Luca Gagliardelli
University of Modena and Reggio Emilia
Modena, Italy
luca.gagliardelli@unimore.it

Giovanni Simonini
MIT CSAIL
Cambridge, MA, USA
giovanni@csail.mit.edu

Domenico Beneventano
University of Modena and Reggio Emilia
Modena, Italy
domenico.beneventano@unimore.it

Sonia Bergamaschi
University of Modena and Reggio Emilia
Modena, Italy
sonia.bergamaschi@unimore.it

ABSTRACT

We present SparkER, an ER tool that can scale practitioners' favorite ER algorithms. SparkER has been devised to take full advantage of parallel and distributed computation as well (running on top of Apache Spark). The first SparkER version was focused on the *blocking* step and implements both *schema-agnostic* and *Blast meta-blocking* approaches (i.e. the state-of-the-art ones); a GUI for SparkER, to let non-expert users to use it in an unsupervised mode, was developed. The new version of SparkER to be shown in this demo, extends significantly the tool. *Entity matching* and *Entity Clustering* modules have been added. Moreover, in addition to the completely unsupervised mode of the first version, a supervised mode has been added. The user can be assisted in supervising the entire process and in injecting his knowledge in order to achieve the best result. During the demonstration, attendees will be shown how SparkER can significantly help in devising and debugging ER algorithms.

1 INTRODUCTION

Entity Resolution (ER) is the task of identifying different representations (*profiles*) that pertain to the same real-world entity. ER is a fundamental and expensive task for Data Integration [2]. The naïve solution of ER (i.e. comparing all profiles to each others) is impracticable when the data volume increases (e.g. Big Data), thus *blocking* techniques are employed to cluster similar records and to limit the number of comparisons only among the profiles contained in the same block.

In a real-world scenario, to identify a blocking strategy (i.e. the *blocking key*) yielding high recall and precision is a hard task [4]. In particular, in the Big Data context, *schema-aware* techniques have two main issues: (i) schema alignment, hardly achievable with a high heterogeneity of the data; (ii) labeled data to train classification algorithms, or human intervention to select which attributes to combine. To overcome these problems, the *schema-agnostic* approach was introduced [10]: each profile is treated as a *bag of words* and schema-information is ignored. For instance, *Schema-Agnostic Token Blocking* considers as blocking key each token that appear in profiles, regardless of the attribute in which it appears (Figure 1(b)). However, *schema-agnostic* methods produce a very low precision.

© 2019 Copyright held by the owner/author(s). Published in Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), March 26-29, 2019, ISBN 978-3-89318-081-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

So, to mitigate this problem, they are typically coupled with *meta-blocking* [6, 10, 13]. 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; each node is connected to another one if the profiles co-occurs in at least one block. Then, the edges are weighted on the basis of the co-occurence of its adjacent profiles and for each profile a threshold is computed. Finally, the graph is pruned removing the edges which have a weight lower than the threshold. A toy example is shown in Figure 1(c): each edge is weighted counting the co-occurring blocks of its adjacent profiles, and is retained if its weight is above the average. The dashed lines are the removed comparisons.

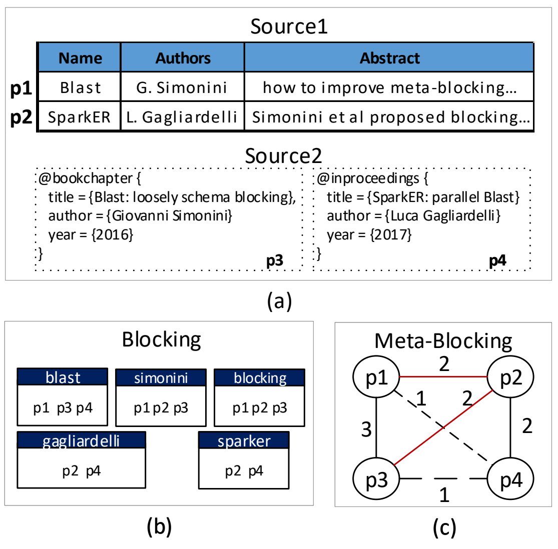

Source1
Name | Authors | Abstract
p1 | Blast | G. Simonini | how to improve meta-blocking…
p2 | SparkER | L. Gagliardelli | Simonini et al proposed blocking…
Source2
@bookchapter { title = {Blast: loosely schema blocking}, author = {Giovanni Simonini} year = {2016} } p3
@inproceedings { title = {SparkER: parallel Blast} author = {Luca Gagliardelli} year = {2017} } p4
(a)
Blocking
blast: p1 p3 p4
simonini: p1 p2 p3
blocking: p1 p2 p3
gagliardelli: p2 p4
sparker: p2 p4
(b)
Meta-Blocking
p1
p2
p3
p4
2
1
2
3
2
1
(c)

Figure 1: Schema-agnostic (meta-)blocking process.

In [13] we proposed *Blast*, which introduces the notion of *loose schema information* extracted from the data and composed of: (i) *attribute partitioning* and (ii) *attribute partition entropy* (Figure 2(a)). The idea beyond *attribute partitioning* is that more values two attributes share, more are similar, thus similar attributes are put together in the same partition. Then, the *meta-blocking* takes into account the generated attributes' partitions: the blocking key is composed by tokens concatenated to partition IDs; in this way, the token "Simonini" (Figure 2(b)) is split into two tokens, disambiguating "Simonini" as author ("Simonini_1"), and

"Simonini" as cited author; note that "Simonini_1" do not generate any block, since it appears only in p_2.

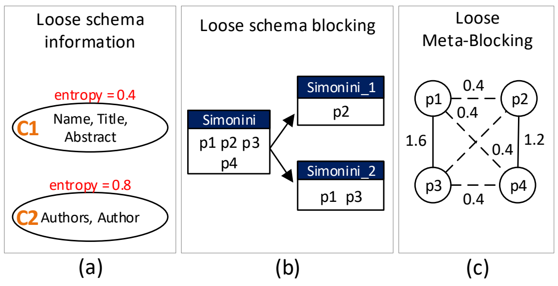


Figure 2: Meta-blocking with loose schema information.

Attribute partition entropy computes the entropy of each cluster and gives more importance to the profiles that co-occurs in blocks generated from clusters with high entropy. The idea is that finding equalities inside a cluster with a high variability of the values (i.e. high entropy) has more value that finding them in a cluster with low variability (i.e. low entropy). The *attribute partition entropy* is used in order to improve the edges weights: each edge of the *meta-blocking* graph is re-weighted according to the entropy associated to the block that generates it (i.e. the entropy of the partition from which the blocking key belongs), as shown in Figure 2(c). This affects the meta-blocking by helping to remove more superfluous comparisons than the ones removed by schema-agnostic blocking (the two retained red edges of Figure 1(c) are now removed).

At the end of the pruning step, the *meta-blocking* produces the *candidate pairs*, i.e. pairs of profiles related to the same entity. Then, these pairs have to be resolved, i.e. it is necessary to decide if a pair is a true match or not, this task is called *entity matching*. Several techniques can be applied to perform this task, e.g. resolution functions, classifiers, crowdsourcing, etc. Finally, , the retained matching pairs are clustered (*entity clustering*) in order to group together all the profiles associated to the same entity.

Several tools were proposed to cover the full Entity Resolution stack [9, 11]. In particular, JedAI [11] is more devoted to work with semi-structured data, a *schema-agnostic* approach, and the *entity matching* phase uses only unsupervised techniques (i.e. no labeled data are required). In contrast, Magellan [9] is meant to work with structured data and a supervised approach, so the user has to align the schema, to provide matches examples to perform *entity matching*, and supervise each step. Moreover, JedAI covers the *entity clustering* step, while Magellan not.

Nevertheless, none of these tools exploits the benefits of distributed computing. Works on the *meta-blocking* parallelization have been proposed [5], but they are implemented using *Hadoop MapReduce*, that is not the best paradigm to exploit modern cluster architectures [3, 12]. SparkER[1] is an Entity Resolution tool for Apache Spark[2] designed to cover the full Entity Resolution stack in a big data context.

Our approach. The first SparkER version [14] was focused on the *blocking* step and implements using *Apache Spark* both *schema-agnostic* [10] and *Blast* [13] *meta-blocking* approaches (i.e. the

[1]https://github.com/Gaglia88/sparker
[2]http://spark.apache.org

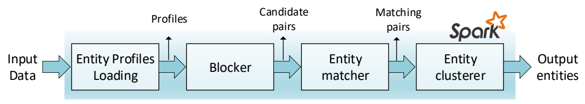


Figure 3: SparkER architecture.

state-of-the-art ones). The description of the algorithms that we devised for Apache Spark (and any MapReduce-like system) can be found in our technical report [15]. Also, we developed a GUI for SparkER to let non-expert users to use it in an unsupervised mode.

The new version of SparkER that will be shown in this demo, extends significantly the tool. *Entity matching* and *entity clustering* modules have been added. Moreover, in addition to the completely unsupervised mode of the first version, a supervised mode has been added. The user can be assisted in supervising the entire process and in injecting his knowledge in order to achieve the best result.

In the following Section 2, we present the main modules that compose SparkER and in Section 3 the process debugging. Finally, in Section 4 we present the demonstration for the EDBT attendees.

2 SPARKER

SparkER is a distributed entity resolution tool, composed by different modules designed to be parallelizable on Apache Spark. Figure 3 shows the architecture of our system. There are 3 main modules: (1) **blocker**: takes the input profiles and performs the blocking phase, providing as output the candidate pairs; (2) **entity matcher** takes the candidate pairs generated by the blocker and label them as match or no match; (3) **entity clusterer** takes the matched pairs and groups them into clusters that represents the same entity. Each of these modules can be seen as black box: each one is independent from the other.

2.1 Blocker

Figure 4 shows the blocker' sub-modules implementing the *Loose-Schema Meta-Blocking* method described in the introduction.

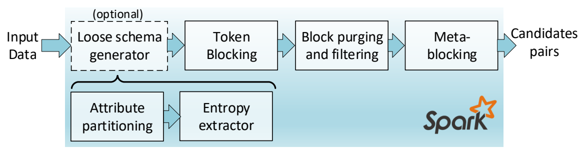


Figure 4: Blocker module

Loose Schema Generator-Attribute Partitioning: attributes are partitioned in cluster using a *Locality-sensitive Hashing* (LSH) based algorithm. Initially, 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 compare in multiple clusters. Then, for each attribute only the most similar one is kept, obtaining pairs of similar attributes. Finally, the transitive closure is applied to such attributes pairs and then attributes are partitioned into nonoverlapping clusters (Figure 2(a)). All the attributes that do not appear in any cluster are put in a *blob* partition.

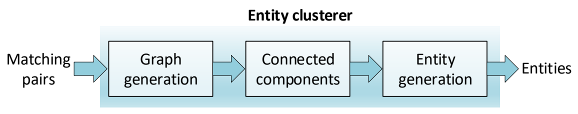


Figure 5: Entity clusterer.

Loose Schema Generator-Entropy Extractor: computes the Shannon entropy for each cluster.

Block Purging and Filtering : the block collection is processed to remove/shrink its largest blocks [10]. *Block Purging* discards all the blocks that contain more than half of the profiles in the collection, corresponding to highly frequent blocking keys (e.g. stop-words). *Block Filtering* removes each profile from the largest 20% blocks in which it appears, increasing the precision without affects the recall.

Meta-Blocking: Finally, the *meta-blocking* method [10, 13] introduced in the introduction is applied. The parallel *meta-blocking*, implemented on Apache Spark, 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.

The output of the `blocker` module are profile pairs connected by an edge, which represent candidate pairs that will be processed by the *entity matcher* module.

2.2 Entity Matcher and Clusterer

Regarding Entity Matching, any existing tool can be used. In the demo we will show the one implemented in Magellan [9]. The `Entity Matcher` produces *matching pairs* of similar profiles with their similarity score (*similarity graph*). The user can select from a wide range of similarity (or distance) scores, e.g.: Jaccard similarity, Edit Distance, CSA [1].

The `Entity Clusterer` receives as input the similarity graph, in which the profiles are the nodes and the matching pairs represent the edges, and partition its nodes into *equivalence clusters* such that every cluster contains all profiles that correspond to the same entity. Several entity clustering algorithms have been proposed in literature [8]; at the moment, we use the *connected component* algorithm[3], based on the the assumption of *transitivity*, i.e., if p_1 matches with p_2, p_2 matches with p_3, then p_1 matches with p_3. At the end of this step, the system produces clusters of profiles: the profiles in the same cluster refer to the same real-world entity.

3 PROCESS DEBUGGING

The tool can work in a completely unsupervised mode, i.e. the user can use a default configuration and performs the process on its data without taking care of the parameters tuning. Otherwise, the user can supervise the entire process, in order to determine which are the best parameters for its data, producing a custom configuration. Given the iterative nature of this process (e.g. the user try a configuration, if it is not satisfied changes it, and repeat the step again), it is not feasible to process the entire input data, as the user should waste too much time. Thus, it is necessary to sample the input data, reducing the size. The main problem is to take a sample that represents the original data, and also contains matching and non matching profiles. This problem was already addressed in [9], where the authors proposed to pick up some random K profiles P^K, then for each profile $p_i \in P^K$ pick up $k/2$ profiles that could be a match (i.e. shares a high number of token with p_i) and $k/2$ profiles randomly. K and k are two parameters that can be set by the user based on the time that she wants to spend (e.g. selecting more records requires a higher computation time).

Each step can be assessed using precision and recall, if a ground-truth is available; otherwise the system selects a sample of the generated profile pairs (e.g. pairs after blocking, matching pairs after matching, etc.) and shows them to the user who, on the basis of his experience evaluates whether the system is working well or not.

In the `blocker` each operation (blocking, purging, filtering, and meta-blocking) can be fine tuned in order to obtain better performances, e.g. the purging/filtering are controlled through parameters that can change the aggressiveness of filters, or the *meta-blocking* can use different types of pruning strategies, etc. Moreover, if the *Loose Schema Blocking* is used, it is possible to see how the attributes are clustered together, and how to change the clustering parameters in order to obtain better clusters.

In the entity matching phase, it is possible to try different similarity techniques (e.g. Jaccard, cosine, etc.) with different thresholds.

At present no tuning activity is possible in the *clustering* step since the connected component algorithm used does not have any parameters. At the end of the process, the system allows to explore the generated entities and to store the obtained configuration. Then, the optimized configuration can be applied to the whole data in a batch mode, in order to obtain the final result.

4 DEMONSTRATION OVERVIEW

During the demonstration, participants will explore the features of our system on the `Abt-Buy` dataset[4]. It contains 2,000 products extracted from *Abt.com* and *Buy.com* catalogs, denoted respectively in red and blue. The dataset comes with a ground-truth that allows to analyze the performances of each `SparkER` step. Also, different datasets can be used[5] during the demonstration.

In this demo we focus on showing the attribute partitioning unsupervised/supervised step, the use of Attribute Partition Entropy was illustrated in our previous paper [7] and the meta-blocking step including entropy.

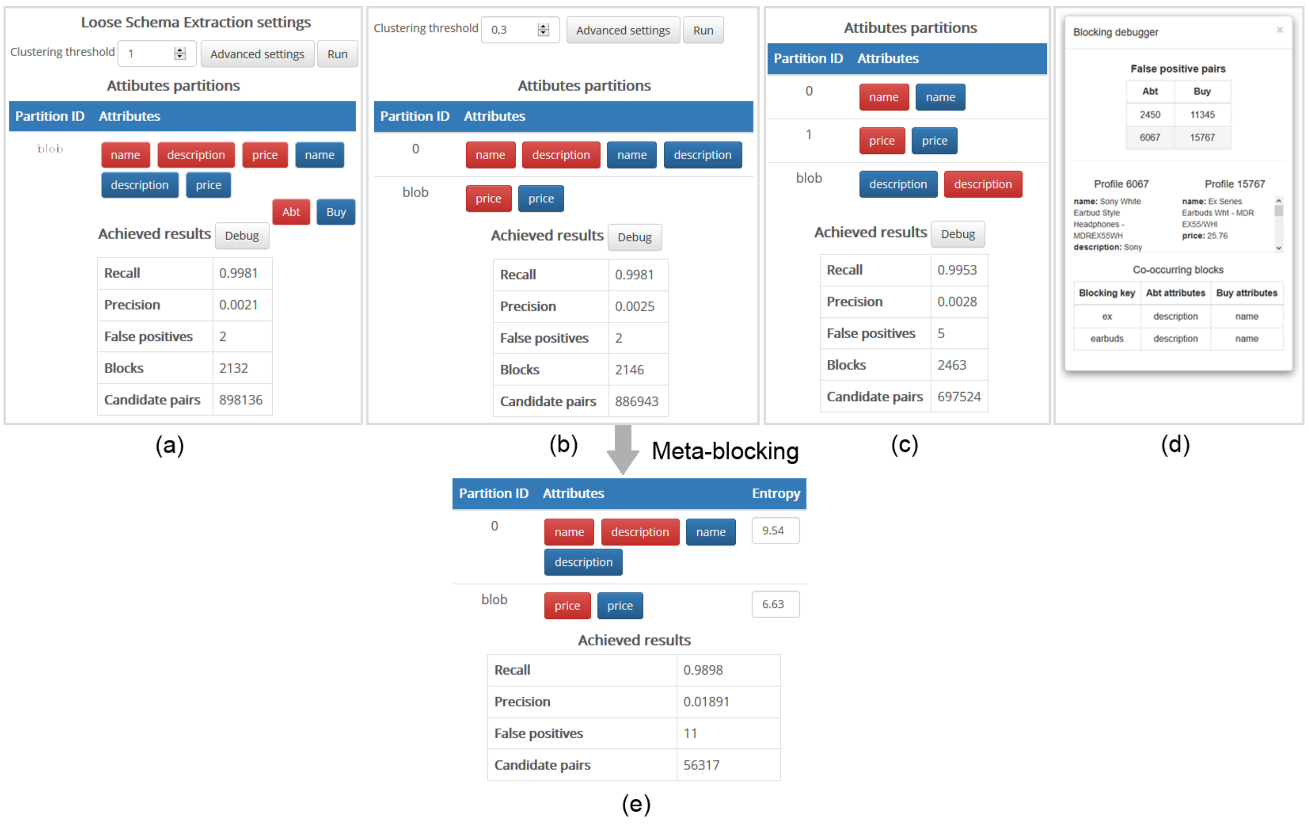
The tool displays the attributes partitions, recall/precision, the number of blocks (blocking keys) generated, the number of candidate pairs in the blocks, and the number of false positives (i.e. the pairs that are in the ground-truth but are lost during the blocking process) obtained after blocking. Through the interface it is possible to modify the clustering threshold and other parameters (*Advanced settings*) which influence the algorithm in a more marginal way.

We start setting the threshold to the maximum value (1) e.g a schema-agnostic token blocking is applied and all the attributes fall in the same *blob* cluster (Figure 6(a)). Then the user decreases the threshold (0.3) and looks at what happens (Figure 6(b)). Two clusters are created, representing, respectively the name with the

[3]This approach is implemented by using the GraphX library of Spark (https://spark.apache.org/graphx/) that natively implement the *connected component* approach.

[4]https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution

[5]The datasets are available at: https://sourceforge.net/projects/sparker/files/datasets/



(a)

Clustering threshold: 1

Atributes partitions

Partition ID	Attributes
blob	name, description, price, name, description, price

Abt, Buy

Achieved results

Recall	0.9981
Precision	0.0021
False positives	2
Blocks	2132
Candidate pairs	898136

(b)

Clustering threshold: 0,3

Atibutes partitions

Partition ID	Attributes
0	name, description, name, description
blob	price, price

Achieved results

Recall	0.9981
Precision	0.0025
False positives	2
Blocks	2146
Candidate pairs	886943

(c)

Atibutes partitions

Partition ID	Attributes
0	name, name
1	price, price
blob	description, description

Achieved results

Recall	0.9953
Precision	0.0028
False positives	5
Blocks	2463
Candidate pairs	697524

(d)

Blocking debugger

False positive pairs

Abt	Buy
2450	11345
6067	15767

Profile 6067
name: Sony White Earbud Style Headphones - MDREX55WH
description: Sony

Profile 15767
name: Ex Series Earbuds Wht - MDR EX55/WHI
price: 25.76

Co-occurring blocks

Blocking key	Abt attributes	Buy attributes
ex	description	name
earbuds	description	name

(e)

Partition ID	Attributes	Entropy
0	name, description, name, description	9.54
blob	price, price	6.63

Achieved results

Recall	0.9898
Precision	0.01891
False positives	11
Candidate pairs	56317

Figure 6: Process debugging. The figure shows how it is possible to debug the blocking phase.

description, and the prices of the products. However, we see how precision slightly increases but the number of candidate pairs has been reduced.

Now, the user try to modify the clusters, as apparently separating the attributes that refer to the name from those which refer to the description of products (Figure 6(c)) seems a good idea. He looks at the result and sees that unfortunately the number of false positive increases.

By the *Debug* button it is possible to understand where the false positives come from (Figure 6(d)). The tool shows the list of false positive pairs (i.e. pairs that are in the ground-truth but are not present after the blocking). By clicking on a pair, its profiles and shared *blocking key* are shown and the user can understand why this pair was lost. In the example we can see that the lost pairs match on blocking keys referring to the name and description attributes. So, partitioning descriptions and names was a wrong choice and the automatic solution proposed by the tool was better (Figure 6(b)). Moreover, it suggests that the choice of partitioning the attributes on the bases of their names (i.e. exploting schema information) can be wrong.

Finally, Figure 6(e) shows the debugging of the meta-blocking phase, with the Entropy's values obtained by the Entropy Extractor module. We can see a large decrease in the number of candidate pairs w.r.t. 6(b) thus proving the effectiveness of our technique.

REFERENCES

[1] Fabio Benedetti, Domenico Beneventano, Sonia Bergamaschi, and Giovanni Simonini. 2019. Computing inter-document similarity with context semantic analysis. *Information Systems* 80 (2019), 136–147.
[2] Sonia Bergamaschi, Domenico Beneventano, Francesco Guerra, and Mirko Orsini. 2011. Data Integration. In *Handbook of Conceptual Modeling - Theory, Practice, and Research Challenges*. 441–476.
[3] Sonia Bergamaschi, Luca Gagliardelli, Giovanni Simonini, and Song Zhu. 2017. BigBench workload executed by using Apache Flink. *Procedia Manufacturing* 11 (2017), 695–702.
[4] P. Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering* 24, 9 (2012), 1537–1555.
[5] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas. 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems* 65 (2017), 137–157.
[6] Simonini G., Papadakis G., Palpanas T., and Bergamaschi S. 2018. Schema-Agnostic Progressive Entity Resolution. In *ICDE 2018*. 53–64.
[7] Simonini G., Gagliardelli L., Zhu S., and Bergamaschi S. 2018. Enhancing Loosely Schema-aware Entity Resolution with User Interaction. In *HPCS 2018, July 16-20, 2018*. 860–864.
[8] O. Hassanzadeh, F. Chiang, H. C. Lee, and R.ée J Miller. 2009. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1282–1293.
[9] P. Konda, S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *VLDB Endowment* 9, 12 (2016), 1197–1208.
[10] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. 2016. Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-blocking.. In *EDBT*. 221–232.
[11] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. 2018. The return of jedAI: end-to-end entity resolution for structured and semi-structured data. *VLDB Endowment* 11, 12 (2018), 1950–1953.
[12] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan. 2015. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2110–2121.
[13] G. Simonini, S. Bergamaschi, and HV Jagadish. 2016. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *VLDB Endowment* 9, 12 (2016), 1173–1184.
[14] G. Simonini, L. Gagliardelli, S. Zhu, and S. Bergamaschi. 2018. Enhancing Loosely Schema-aware Entity Resolution with User Interaction. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 860–864.
[15] G. Simonini, Gagliardelli L., Bergamaschi S., and Jagadish H.V. 2019. Technical Report. (2019). http://dbgroup.unimo.it/paper/g/scaling_er_report.pdf