Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0
M. Peruzzini et al. (Eds.)

BigDedup: A Big Data Integration Toolkit for Duplicate Detection in Industrial Scenarios

Luca GAGLIARDELLI[1], Song ZHU, Giovanni SIMONINI and Sonia BERGAMASCHI

University of Modena and Reggio Emilia, Italy

Abstract. Duplicate detection aims to identify different records in data sources that refer to the same real-world entity. It is a fundamental task for: item catalogs fusion, customer databases integration, fraud detection, and more. In this work we present BigDedup, a toolkit able to detect duplicate records on Big Data sources in an efficient manner. BigDedup makes available the state-of-the-art duplicate detection techniques on Apache Spark, a modern framework for distributed computing in Big Data scenarios. It can be used in two different ways: *(i)* through a simple graphic interface that permit to the user to process structured and unstructured data in a fast and effective way; *(ii)* as a library that provides different components that can be easily extended and customized. In the paper we show how to use BigDedup and its usefulness through some industrial examples.

Keywords. Duplicate detection, Entity Resolution, Data Integration, Record Linkage, Big Data

Introduction

Duplicate detection (also known as Entity Resolution) is the task of identifying if different records pertain to the same real-world object. It is a fundamental and expensive task for Data Integration process. With the advent of Industry 4.0 and the massive employment of smart services, there is an increasing number of application domains where duplicate detection is being required: e-commerce, for catalog fusion; security, to detect frauds; customer databases integration and more. Moreover, Boston Consulting [6] identifies as a fundamental pillar of Industry 4.0 the "Horizontal and Vertical System Integration", i.e., companies, supplier, customers, will be much more cohesive, as a cross-company. Thus, the integration of all these figures will require an intensive use of duplicate detection, in order to integrate all their data.

The main challenge of using duplicate detection in Industry 4.0 applications is related to the huge amount of data to work with, i.e., Big Data. For instance, a product catalog such as Amazon can contain millions of products, and to find the duplicated ones involves billions of comparisons. To manage and analyze these Big Data the distributed computing is the most promising approach [2, 3, 16], and different distributed frameworks has been proposed in the last years [7, 8, 9]. One of the most

[1] Corresponding Author, Mail: luca.gagliardelli@unimore.it.

used framework is Apache Spark[2], that is an Open Source, fast and general engine for Big Data processing.

In this work we present BigDedup a toolkit that makes available the state-of-the-art duplicate detection techniques on Apache Spark. It can be used through a simple web application, or as a library to create more complex applications.

The rest of the paper is structured as follows: in the following Section 1 the background and the related works are presented. Section 2 describes BigDedup architecture. Then, Section 3 exhibits the results of the experiments on different datasets. Finally, Section 4 concludes the paper.

1. Background and related works

This chapter outline the basic concepts of duplicate detection that are necessary to understand the rest of the paper.

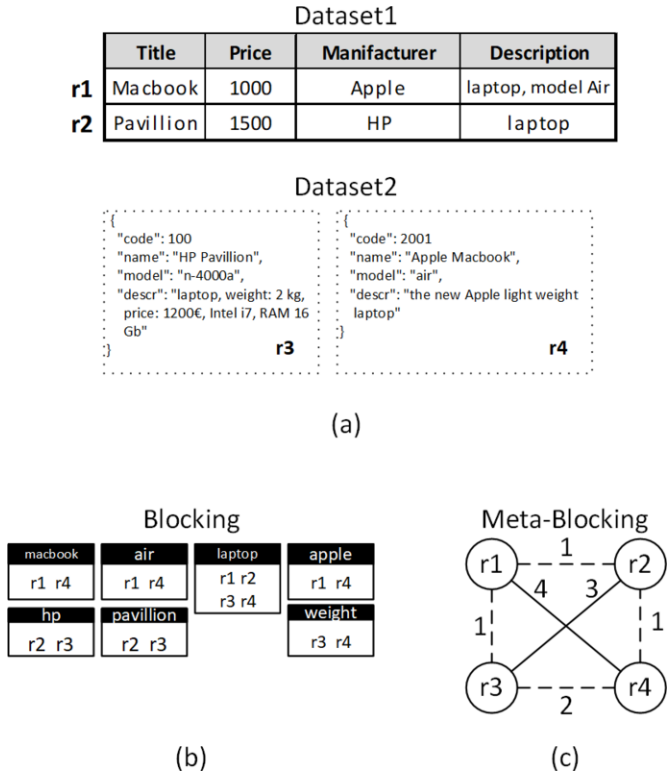

Figure 1. Example of schema-agnostic (meta-)blocking process. (a) A set of records R from an imaginary data lake. (b) The set of blocks B derived applying schema-agnostic Token Blocking on R (i.e., each token is a blocking key). (c) The blocking graph derived from the blocks of B, and the effect of the pruning algorithm, dashed lines are the removed comparisons. 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 [4]).

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

1.1. Blocking

The naïve solution of duplicate detection consists in comparing each record to all the other records. This approach has a quadratic complexity, thus unmanageable in practice, especially in a Big Data context. To overcome this issue, i.e., to reduce the number of comparisons, *blocking* techniques [10] have been introduced. Blocking techniques extract features from the records (*blocking keys*), and clusters together records that have similar features. Then, the all-pair comparison is limited within each block. This limits the number of comparisons, because two records being compared only if they co-occur in the same block. An example of blocking is reported in Figure 1b.

To define good blocking keys, the schema of the data has to be known. However, typically the schema alignment of different datasets is a heavy task, and usually requires the intervention of domain experts. Thus, schema-agnostic blocking is employed to avoid schema-alignment (they do not consider the schema). For example, the *token blocking*, that is one of the most used one, uses only the values to generate the blocking keys. This, will result in a higher recall (i.e. finds almost all the existing duplicates), but with a very low precision (i.e. retains a high number of superfluous comparison). To overcome this problem, and improve the precision, Simonini et al. [1] propose a schema aware version of token blocking called *Loose Schema-aware Blocking* (LSB). LSB generates clusters of similar attributes by applying LSH on their values, also give them a weight in order to change their relevance in the meta-blocking phase, then apply *token blocking* taking into account the generated clusters (i.e. if two records have the same blocking key they are clustered together only if the blocking key belongs to the same cluster of attributes). However, the use of the blocking is not sufficient to limit the number of comparisons, more sophisticated techniques are required.

1.2. Meta-blocking

Given that the *blocking* is not able to reduce enough the number of comparisons, Papadakis at all. [4] have proposed the *Meta-Blocking*. *Meta-Blocking* aims to restructure the block collection obtained with the blocking by removing the superfluous comparisons, in order to obtain almost the same level of *recall* (the fraction of detected duplicates correctly identified), but a higher *precision* (the average number of comparisons executed to find each duplicate). *Meta-Blocking* generates a graph where the records are the nodes, and two records are connected by an edge only if they co-occur in at least one block. The edges are weighted using the idea that more blocks two records share, more they are similar. Then, for each profile is calculated a threshold (e.g. the average value of all its edges), and all the edges that has a weight lower than the threshold are pruned. The *Meta-Blocking* produce as result a list of record pairs that have to be resolved with a resolution process (e.g. resolution function, crowdsourcing, etc.). A toy example of Meta-Blocking is presented in Figure 1c, as it is possible to see the superfluous edges (*r1-r2*, *r1-r3*, *r2-r3*, *r2-r4*) are pruned, and only the relevant ones are retained (*r1-r4*, *r2-r3*).

1.3. Entity Resolution tools

There are other data deduplication tools, among which the most similar to BigDedup is JedAI [14]. JedAI exploits the meta-blocking [4] to perform the Entity Resolution as

BigDedup. However, JedAI does not integrate the BLAST method [1] that is the state-of-the-art one for the Entity Resolution. A full comparison between BLAST and the standard meta-blocking can be found in [1]. Most importantly, BigDedup is developed for Apache Spark and can manage big datasets using the distributed computing, JedAI is developed for Java and can be executed only on a single machine, which is limiting: to manage high quantity of data can be problematic.

2. BigDedup

BigDedup is divided in two separated components: the *core* that is organized in modules, each performing a specific task, devised to be parallelizable on Apache Spark; the *GUI* that allow to manage the *core* operations, and analyze the results through a powerful multi-device web interface.

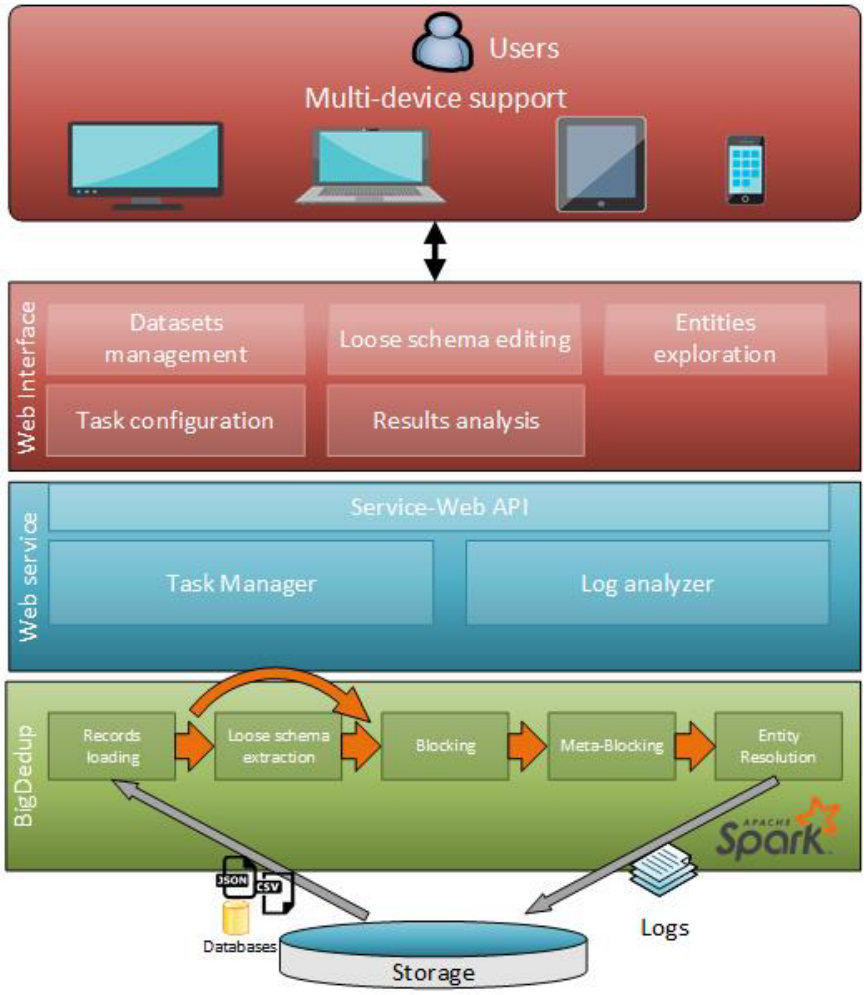

Figure 2. BigDedup architecture.

2.1. Core

The core is composed by different modules that are combined together in order to perform the Entity Resolution process, as outlined in Figure 2. The core of BigDedup is available as Scala library from our repository[3].

BigDedup can ingest different kind of data, structured (databases) and unstructured (CSV, JSON, RDF, etc.). The data are loaded into RDDs [7] that can be processed with Apache Spark. It supports different *blocking* types, implementing the schema-agnostic token blocking [10], and the loose-schema aware blocking [1]. Moreover, in case of loose-schema aware blocking the user can improve the autogenerated attribute clusters modifying them.

The *meta-blocking* module let to restructure the blocks collection, in order to increase the *precision*. It implements all the *meta-blocking* methods described in [1, 4]. Also, we developed a specific strategy to perform the meta-blocking on Spark that is inspired to the broadcast join [11], that allows to not materialize the whole blocking graph, but only a portion of it is materialized in parallel. This will result in a faster and less memory expensive algorithm, that let to process big datasets in a fast and efficient manner. M*eta-blocking* produces as result a list of record pairs that are candidates to be true matching. These pairs have to be validated through an Entity Resolution process, a user defined function can be used, or other systems such as Magellan [12].

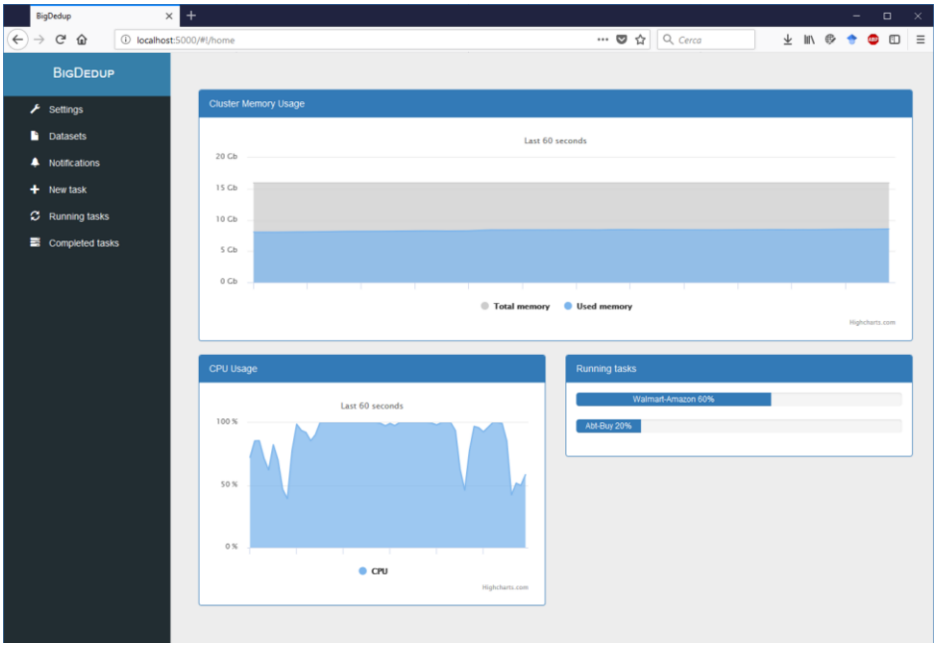

Figure 3. BigDedup GUI.

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

2.2. GUI

The GUI allows to manage the *core* operations thought a simple web interface (Figure 3). The main page of the *GUI* shows the status of the server (CPU and memory usage) and the list of running tasks. Also, it is possible to: configure the Spark settings (memory usage, level of parallelism, etc.); manage the datasets; manage the notifications (the application can send Telegram [4] notifications when a task is completed or if there are errors in the executions); launch new tasks; monitor the running tasks, analyze the results, and compare the results of multiple tasks.

The task analyzer resumes the main results in four charts that shows the *recall*, *precision*, *F1 score* and *execution time* for each *meta-blocking* method employed in the task (Figure 4a). Also, it permits to analyze in detail all the steps, for example to see how many blocks were generated, or the execution time of a certain step, and so on. If the task performed the *loose schema-aware* token blocking it is possible to see the generated cluster of attributes (Figure 4b). For each cluster are showed the contained attributes, highlighted in different colors depending from which dataset they belong. Also, the score calculated for each cluster (i.e. the entropy) is showed. All the attributes that was not clustered together, are put in a *blob* cluster. The user can change the clusters moving the attributes from a cluster to another, or creating a new one, and restart the task in order to improve the results. It is also possible to explore the records identified as duplicate by the system (Figure 4c).

Figure 4. (a) the task analysis presents a summary of the results throught four charts (precision, recall, F1 score, execution time), also let to see the detailed information about all steps; (b) if the loose schema-aware token blocking is used, it is possibile to see and edit the generated attribute clusters; (c) it is possibile to explore the identified as duplicate records.

[4] https://telegram.org

3. Experiments

We tested BigDedup on three real-world datasets, measuring it performances in term of *recall*, *precision* and *execution time*. In these experiments are used both schema-agnostic meta-blocking and loose-schema aware meta-blocking, in order to compare them. The *recall* measures the fraction of existing duplicates that are discovered, while the *precision* measures the number of executed comparison per duplicate.

The datasets [12, 13, 15, 19] used for the tests contain different products collected from different data sources, and all of them have a ground truth that provides the existing duplicates. *Abt-Buy* matches products from Abt.com and Buy.com; *Google-Amazon* matches products from GoogleProducts and Amazon.com. *Walmart-Amazon* matches products from Walmart.com and Amazon.com. The datasets charachteristics are described in Table 1.

Since these datasets have a size that do not requires the use of a cluster, all the tests were performed on a single machine with 16 Gb of RAM and an i7-5500 CPU, using Apache Spark 2.0.2.

Table 1. The charachteristics of the datasets employed in the experiments. The first column indicats the number of records; the second the number of attributes, and the last the number of existing duplicates.

	#r1 - #r2	#A1 - #A2	#D
Abt-Buy	1.1k – 1.1k	3 – 3	1.1k
Google-Amazon	1.4k – 3.0k	4 – 4	1.1k
Walmart-Amazon	2.6k – 22k	16 – 21	1.2k

Figure 5 presents the results in term of *recall* and *precision* of both schema-agnostic and loose-schema aware meta-blocking (i.e. meta-blocking applied on blocks generated with loose-schema aware blocking). The use of loose-schema aware blocking improves the *precision*, but achieves a lower *recall*, w.r.t the schema-agnostic blocking. On *Abt-Buy* and *Walmart-Amazon* the *recall* of both meta-blocking it is almost the same (Figure 5a-b), while on *Google-Amazon* the schema-agnostic one achieves a result $1.2x$ higher (Figure 5c). In term of *precision* the schema-aware meta-blocking always outperforms the other, obtaining a *precision* from 2.14 (Figure 5d) to 189.9 (Figure 5f) times higher.

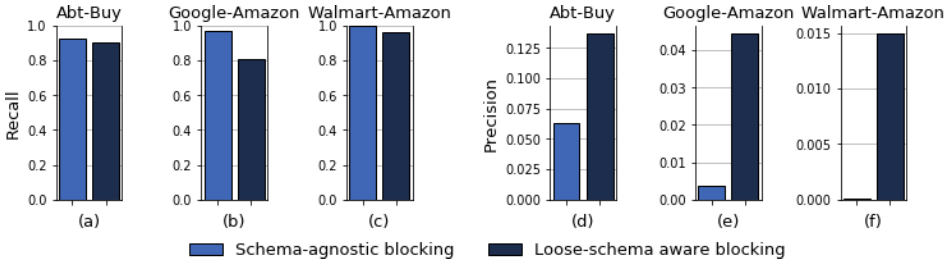

Figure 5. *Recall* and *precision* achieved by considered methods on all datasets.

In Figure 6 are showed the execution times obtained on all the datasets from both meta-blocking techniques, and from a resolution function that employs the Jaccard

Similarity[5] to compare the meta-blocking output pairs. The schema-agnostic process is always faster than loose-schema aware one, this because the latter has to extract the loose-schema information. Despite that, considering the global execution time (meta-blocking + entity resolution) the loose-schema meta-blocking will result faster than the other (Figure 6b-c), the only exception is *Abt-Buy* (Figure 6a). That happens because the loose-schema meta-blocking produces less pairs for the entity resolution function, so the extra-time required during the meta-blocking is rewarded in the entity resolution step. On *Abt-Buy* the execution times are similar because it is a very small dataset, and both methods produce a similar number of pairs.

Figure 6. Execution times of considered methods on all datasets. In green is showed the whole meta-blocking time, and in orange the time requested by the entity resolution function.

4. Conclusion

In this paper we presented BigDedup, a toolkit for Apache Spark that implements the state-of-the-art duplicate detection techniques, able to work efficiently in Big Data scenarios, fulfilling the needs of Industry 4.0 that produces a huge amount of data. It can exploit both schema-agnostic and schema-aware *(meta-)blocking* techniques, that are able to automatically align the schema of heterogeneous data sources. Finally, we show BigDedup performances testing it on several real-world datasets, showing the differences in term of *recall*, *precision* and *execution time* of the different *(meta-)blocking* techniques.

As future work, we are planning to extend the framework to support other applications, such as: keyword search [20] in large corpus of documents, and web pages tagging systems [18].

References

[1] G. Simonini, S. Bergamaschi, and H.V. Jagadish, BLAST: a loosely schema-aware meta-blocking approach for entity resolution, *Proceedings of the VLDB Endowment*, 9(12), 2016, pp. 1173-1184.

[2] S. Bergamaschi, L. Gagliardelli, G. Simonini and S. Zhu, BigBench workload executed by using Apache Flink, *Procedia Manufacturing*, Vol. 11, 2017, pp. 695-702.

[3] S. Bergamaschi, D. Beneventano, F. Mandreoli, R. Martoglia, F. Guerra and M. Orsini, From Data Integration to Big Data Integration. In S. Flesca et al. (eds.) *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, Springer International Publishing, 2018, pp. 43-59.

[5] More compex functions might be employed, such as CSA [17].

[4] G. Papadakis, G. Koutrika, T. Palpanas and W. Nejdl, Meta-blocking: Taking entity resolution to the next level, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26(8), 2014, pp. 1946-1960.
[5] S. Das, A. Doan, P.S.G.C.C. Gokhale and P. Konda, *The Magellan data repository*, https://sites.google.com/site/anhaidgroup/projects/data.
[6] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel and M Harnisch, Industry 4.0: *The future of productivity and growth in manufacturing industries*, Boston Consulting Group, 2015.
[7] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation 2012, USENIX Association, pp. 2-2.
[8] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony and R.Murthy, Hive: a warehousing solution over a map-reduce framework, *Proceedings of the VLDB Endowment*, 2009, 2(2), pp. 1626-1629.
[9] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi and K.Tzoumas, Apache flink: Stream and batch processing in a single engine, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Vol. 36(4), 2015, pp. 28-38.
[10] P. Christen, A survey of indexing techniques for scalable record linkage and deduplication, *IEEE transactions on knowledge and data engineering*, Vol. 24(9), 2012, pp. 1537-1555.
[11] S. Blanas, J.M. Patel, V. Ercegovac, J. Rao, E.J. Shekita and Y. Tian, A comparison of join algorithms for log processing in mapreduce, *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ACM, 2010, June, pp. 975-986.
[12] P. Konda, S. Das, P.G.C. Suganthan, A. Doan, A. Ardalan, J.R. Ballard and S. Prasad, Magellan: Toward building entity matching management systems, *Proceedings of the VLDB Endowment*, Vol. 9(12), 2016, pp. 1197-1208.
[13] H. Köpcke, A. Thor and E. Rahm, Evaluation of entity resolution approaches on real-world match problems, *Proceedings of the VLDB Endowment*, Vol. 3(1-2), 2010, pp. 484-493.
[14] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas and M. Koubarakis, JedAI: The Force behind Entity Resolution, *European Semantic Web Conference*, Springer, Cham, 2017, pp. 161-166.
[15] S. Bergamaschi, D. Ferrari, F. Guerra, G. Simonini and Y. Velegrakis, Providing insight into data source topics, *Journal on Data Semantics*, Vol. 5(4), 2016, pp. 211-228.
[16] G. Simonini and S. Zhu, Big data exploration with faceted browsing, *High Performance Computing & Simulation (HPCS), 2015 International Conference on*, IEEE, 2015, pp. 541-544.
[17] F. Benedetti, D. Beneventano, S. Bergamaschi and G. Simonini, Computing inter-document similarity with Context Semantic Analysis, *Information Systems*, 2018, https://doi.org/10.1016/j.is.2018.02.009.
[18] F. Guerra, G. Simonini and M. Vincini, Supporting image search with tag clouds: a preliminary approach, *Advances in Multimedia*, 2015, http://dx.doi.org/10.1155/2015/439020.
[19] G. Simonini, G. Papadakis, T. Palpanas and S. Bergamaschi, Schema-agnostic Progressive Entity Resolution, *IEEE International Conference on Data Engineering*, 2018, pp. 53-64.
[20] S. Bergamaschi, F. Guerra and G. Simonini, Keyword search over relational databases: Issues, approaches and open challenges. In: N. Ferro (ed.) *Bridging Between Information Retrieval and Databases*, Springer, Berlin, Heidelberg, 2014, pp. 54-73.