Redblock: a tool for online deduplication on large datasets

Luan Félix Pimentel [1]
Igor Lemos Vicente [1]
Guilherme Dal Bianco [1]

Abstract: *Online data deduplication aims to identify records that represent the same purpose on a continuous data flow environment. It must be able to process a range of information with high effectiveness and no delays. The purpose of this paper is to introduce a developed tool entitled Redblock, for real-time data deduplication, using a distributed platform for online processing combined with an Inverted Index. During the experimental evaluation, Redblock managed to provide good preliminary results in terms of efficiency and effectiveness in a database.*

***Keywords*:** *Data Integration, Online Deduplication, Blocking.*

Resumo: A deduplicação online tem como propósito identificar registros que representam um mesmo objetivo em ambientes com fluxo contínuo de dados. A deduplicação online deve ser capaz de processar volumes variados de informações, sem atrasos e com uma alta eficácia. Este trabalho, propõe uma ferramenta intitulada Redblock para a deduplicação de dados em tempo real. A ferramenta utiliza uma plataforma distribuída de processamento online em conjunto com um método de blocagem utilizando índice invertido. Na experimentação, Redblock demonstrou resultados preliminares promissores em relação a eficácia e a eficiência em uma base de dados.

***Palavras-chave*:** Integração de Dados, Deduplicação Online, Blocagem.

1 Introduction

Data integration provides information in an easy and accessible way by consolidating different sources of data into a single repository. Services such as virtual libraries, media streaming and social media need high-quality integration processes. In terms of feasibility, a fundamental task is identifying entities (i.e., records, documents or texts) that are already inserted in the database so it is not necessary for them to be allocated again. This process is known as data deduplication. The goal involves the constant search for removing duplication within data, without compromising its integrity.

Data deduplication has three main stages: blocking, comparison and classification[1]. Blocking corresponds to the process of generating pair candidates. It means that all records must be analyzed in the search of potential duplicates. Only the records that belong to the same blocks are used for creating pair candidates with quadratic processing cost. In the comparison stage, it is computed the similarity between a pair using similarity functions (i.e., Jaccard, Jaro, and Ngram) [2]. Finally, during the classification stage, some algorithms as decision trees and Naive Bayes are used [3]) or configured heuristics based on thresholds manually defined for identifying among the pairs which ones are duplicates.

Deduplication is traditionally performed in static (or batch) databases. It means that the database is analyzed in its entirety to search records that represent duplicate data. However, applications and services growth on a continuous flow of data in real-time, drives the demand for solutions capable of supporting such data flow. Online deduplication, different than the static version, must be able to handle processing peaks without bottlenecks being detected and must be flexible to possible changes in data patterns.

[1]Laboratório de Análise de Dados (LAD), UFFS, Campus Chapecó - BR 285 - Rodovia SC 484 Km 02, Fronteira Sul - Brasil
{luanfelixpimentel@gmail.com, igorlemosvicente@gmail.com, guilherme.dalbianco@uffs.edu.br}

http://dx.doi.org/10.5335/rbca.v9i2.7143

This paper proposes a tool named Redblock for online deduplication using a distributed computing platform that allows to process data in real time. Redblock main challenge is to ensure that as many duplicate pairs as possible are identified. The second challenge is to allow process large datasets in a short time. Although Redblock implements the main steps of deduplication (blocking and classification), the main contribution is on the blocking process. It takes advantage of an inverted index structure that is stored on a key-value database. The purpose of this database is to optimize the process of searching for the information needed for data deduplication. For the classification process, Redblock uses a supervised decision tree based algorithm. This algorithm is supervised so it depends on a training set. This activity should be minimized as much as possible due to the labeling cost. Thus, Redblock uses random sampling to compose the training method.

The experimental evaluation shows that Reblock was able to correctly group matching pairs and subsequently constructing the candidate pairs for online data deduplication maintaining a high quality efficiency. Also, we show the time-consuming of each bolt to identify the individual workload.

The present paper is organized as follows: In the second section we provide the basis for understanding Redblock's processes. Section 3 describes the tool operation. In the fourth section some related works involving online blocking process are exemplified. In section 5 we present the experiments and results obtained by the third section. Finally, the final considerations are presented and what the next steps will be for the evolution and continuous improvement of the tool.

2 Background

In order to understand our proposed approach, this section presents main methods involving data blocking and the Apache Storm platform which turns possible to develop applications that require massive data processing in real time.

2.1 Blocking Methods

A traditional method of producing data blocks is defining one of the attributes as the blocking criterion or key [4]. Only records that have the same blocking key will be inserted into the same block, reducing substantially the number of comparisons. However, in situations where the key attribute exhibits variations or errors, duplicate pairs may not be grouped correctly, thereby reducing the quality of the method [5].

The Sorted Neighborhood Indexing technique sorts the database according to the key of each block and sequentially moves a fixed size window that will create the data groups on the values. Initially, the database is sorted by the selected key (i.e., the "name" attribute in a table composed of client information). The pair candidates are generated using a fixed-size window. In Table 1 an example is presented containing four registers that when applied a sliding window with size three, it produces a first block containing the first three records of the database (R1, R2, and R3) and another block containing the records (R2, R3, and R4), as shown in table 2.

Table 1: Sorted Neighborhood Indexing example.

Identifier	Name
R1	João Silva
R2	João
R3	Jo
R4	Teresinha Souza

Table 2: Sorted Neighborhood Indexing output for table 1.

Window	Sorted Neighborhood Indexing
R1,R2, R3	(R1, R2), (R2,R3), (R1,R3)
R2,R3, R4	(R2,R3), (R3,R4), (R2,R4)

A problem identified by [1] with this technique is that the ordering of the keys of the blocks is sensitive to

errors and variations in the first positions of the values. If the dataset is large, the names "Christina" and "Kristina" would be far from each other on the list arranged in alphabetical order even though they are very similar names that might refer to the same person.

To address this problem, the Q-Gram method uses as a blocking key an attribute's substrings of size Q. The blocking key (strings) produced using the grams will become the key-value in an index. All records that belong to the same index are inserted into the same block. Although the Q-Gram method obtains good results, the high cost of indexing substrings [6] limits the application of the method to a reduced number of attributes.

The Inverted Index method uses words as a way of indexing a collection of records. The nomenclatures used by this method are separated by terms (different words present in the document) and occurrences (the frequency that a specific word appears in the document). We exemplify in Table 3 how the received data would be arranged with a record separated by commas.

In Table 4, we present a simple output example of the inverted index. As you see, the method was able to group the records together, using the "street name" as the criterion.

Table 3: Data received separated by commas.

Docs	Text
1	João Silva, (49) 987453214, Avenida Gusmão Freire
2	João, (49) 987453214, Avenida Gusmão Freire
3	Jo Silvo, (49) 987453218, Avenida Freire

Table 4: Inverted Index result.

Number	Term	Docs
1	João	1, 2
2	Silva	2
3	(49)987453214	1,2
4	Avenida	1,2, 3
5	Gusmão	1,2
6	Freire	1,2,3

The inverted index has its methodology similar to the traditional method of blocking. Using the inverted index, we do not choose a particular field to produce the blocking keys. Instead, we use all fields and generate keys for each record that is processed [1]. Due to this flexibility, such blocking method was exploited to Redblock as described in Section 3.

2.2 Apache Storm

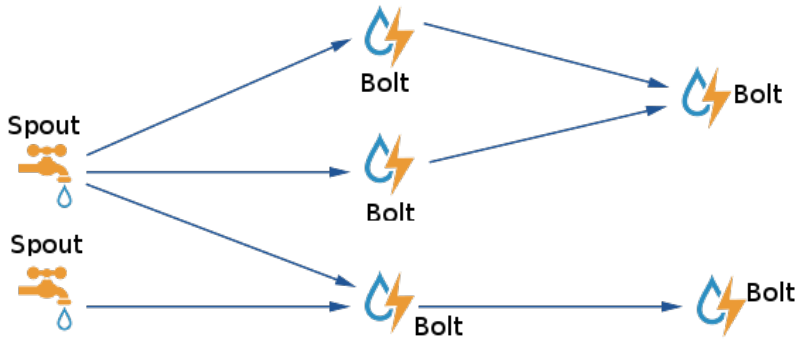
The Apache Storm platform makes possible to develop applications that require massive data processing in real time. The available methodology allows a collection of tuples to be distributed and processed by spouts and bolts. Spouts are similar to systems processes with the function of reading a data source. Bolts are responsible for processing the tuple to be distributed to other bolts or to store the information in external sources.

Bolts and spouts are integrated together into a topology that enables both to be connected in order to form an architecture. As it keeps continuously processing data, the topology will be running until the user stops the processing.

Figure 1 illustrates an example of topology with two spouts and five bolts. In the example, spouts receive the data and distributes it to the next three bolts. They do the processing on the data and will be forwarding tuples to the final two bolts. They are responsible for transforming and processing the data received from previous bolt and saving the required information [2].

[2]Information provided by Apache Storm. Available at: <http://storm.apache.org/>. Accessed on February 6, 2017.

Figure 1: *Bolts* and *Spouts* in *Apache Storm*

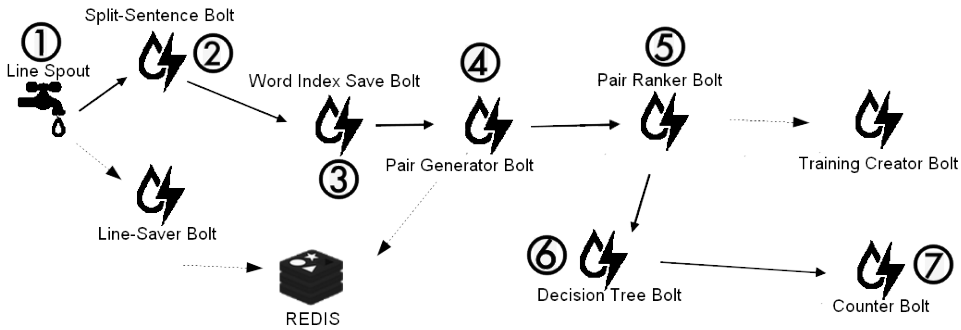


3 Proposed approach - *Redblock*

In this section, the proposed tool for online deduplication entitled Redblock is described to achieve an effective and efficient method. Redblock combines the inverted index method with the online processing platform, Apache Storm, obtaining elasticity on large data sets.

In Figure 2 we illustrate the topology of Redblock that consists of seven main steps. It allows identify bottlenecks and treat them by increasing the level of parallelism. Supposing the bolt that is dealing with the blocking process is overloaded. It is possible to increase the number of bolts for this task, avoiding process delays. The dotted lines represent processes occurring in parallel during the operation of the tool i.e., bolts saving tuples in the database memory while the tuple is assigned to next Bolt in parallel. Each step is detailed below.

Figure 2: Redblock Topology.



3.1 Spout - *Line Spout*

Line Spout (1) represents the initial step of the topology. It starts the database read. To develop a test environment, the continuous flow of data is simulated by loading a static dataset.

After reading the file, the line spout processes line per line of the dataset and sends it to the next bolt recipient. The dataset of table 5 is loaded and sent respectively to the Bolts Line Saver and Split Sentence. In other words, one copy of each record is sent to the Bolt Line Saver and another copy to the Bolt Split Sentence.

Table 5: Explaining how records are loaded by Line Spout.

01, joão, 6562348, estudante
02, joão, 98542138, professor
03, roberto, 52132157, estudante

3.2 Bolt - *Line Saver*

Data processes occurs in real-time. This means that one process (or one bolt) does not freeze or stop while another is running. Therefore, the Line Saver performs its function in parallel to the Split Sentence Bolt receiving the tuple and doing its task.

The line saver bolt aims to save the line for later recovery optimization. The tuple sent by this Spout is stored in [ID][Line] format.

The data is stored in memory on the non-relational database Redis that provides a mechanism for storage and retrieval of data. It is open source (BSD licensed) and stores the tuple on disk which means that it is fast but that it is also non-volatile. In order to achieve its performance, it works with an in-memory key-value dataset, making persistence available[3]

3.3 Bolt - *Split Sentence*

The Split Sentence Bolt (2) receives as input a unique line (according to the table 5) and starts the fragmentation process. Each of the attributes is assigned to the next step for further indexing, i.e., the first record of the table 5 is split into three pieces: "joão", "6562348" and "estudante" and sent to the next step along with the reference ID. All fields will be assigned to the next bolt in the format [ID][field].

3.4 Bolt - *WordIndex Save*

In the WordIndex Save bolt (3), the data produced by Split sentence bolt is entered into an inverted index structure (as described in Section 2). Each record attribute value is looking for in the inverted index structure if the value is already presented then only the record ID is attached to the Ids list.

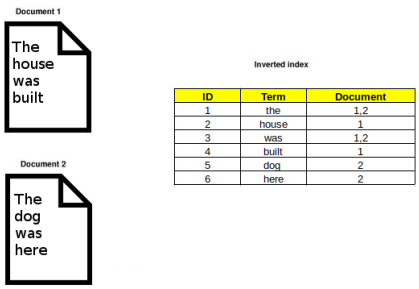
However, if the value is missing in the index structure a new entry is created. A simplified example of the inverted index structure is presented on table 6, with the words "João" e "Silva". Note that the term "João" appears in the records with IDs 01 and 02, the term "6562348" is already present in the record with ID 01.

Table 6: Bolt WordIndex Save processing.

Field	Ids list with the term
João	01, 02
6562348	01
estudante	01,03

One more example on how the inverted index works is illustrated below:

Figure 3: How the inverted index works.



It is possible to verify the presence of very frequent words (known as stop word [3]) by simplistic counting the number of occurrences of the terms in the inverted index. On this way, to avoid that such stop words produce a

[3]Information provided by Redis. Available at: <https://redis.io/>. Accessed on June 14, 2017.

bottleneck, we define a threshold to discard such frequent terms, i.e., if a term is shown in more than 50 records it is ignored by the inverted index.

3.5 Bolt - *Pair Generator*

The Pair Generator bolt (4) builds pairs (for example, a combination of two records with some chance to be a matching) based on the set of words presented in the dataset stored by the Bolt WordIndex Save. The idea is that all records sharing a common word must be compared to check if they represent a duplicate. The main functions of this bolt are to retrieve the entire record by accessing the database. The pairs produced here are sent the next step.

For example, if the term "João" appears in records 01, 02, and 03, each record will serve as a source for the Bolt Pair Generator to retrieve the record and send to the next bolt to the following pairs (1, 2) , (2,3), and (1,3) as shown on Table 6.

3.6 Bolt - *Pair Ranker*

The aim of this bolt is to compute the similarity of each pair received by the Bolt Pair Generator by measuring the degree of similarity of each attribute based on a similarity function. The similarity functions used are Jaccard and Levenshtein [3]. In the Jaccard method, the level of distance that two strings are from each other is found and measured. The Levenshtein distance between two strings is defined as the minimum number of formatting required to transform such string into the other string being represented by insertion, substitution, and deletion of one or more characters. The output of this bolt is pair id and the similarity degree of each attribute value.

3.7 Bolt - *Training Creator*

The goal of the bolt Training Creator is to build a training model using a decision tree algorithms. The model will be used later to identify matching and non-matching pairs. The training is set up from a training base containing a set of predefined size pairs labeled by the user. In this way, identifying such pairs and labeling them is a costly task that should be minimized as much as possible. We implement the sampling generator by using a random function to control the sample size. We experimentally define the sample size variable, as we show in the experimentation.

3.8 Bolt - *Decision Tree*

The Bolt Decision Tree (6) uses the classification model, previously produced, to identify pairs as duplicates or non-duplicates. This bolt receives as input the similarity values of each pair and outputs the classified instance as output. The classification algorithm used is J48 [3].

3.9 Bolt - *Counter Bolt*

The last bolt present in the topology is the Counter Bolt (7). While the topology is up and running this bolt will generate how many matching and non-matching pairs have been computed. This bolt will only have functionality in the presence of the groundtruth.

4 Related Works

A simple process of data deduplication can be accomplished by comparing all the records against all the others present in the database. However, such a method results in a quadratic cost of processing which would be non-practical in the case of a large dataset. In this context, blocking appears as an alternative to reduce the search space by only processing the records that have some indication of representing a duplicate [4].

The incremental database scenario which records are inserted over time is important for the deduplication method to be able to self-adjust to new standards and reduce the processing time of a new entry to meet online

demand. Thus, it is important for the blocking process that represents the largest processing slice [4], to be efficient enough to not result in processing delays.

One of the first works involving online deduplication was proposed by [7] who stated that a method is developed for queries involving matching data to be answered in real time. However, in this method, the dataset is previously processed to enable the real-time response. In the context of online blocking, [8] proposed a technique for the similarities between attributes that are previously calculated when an inverted index is created.

Most of the data is presented statically so the index is appropriately populated, different from the online context.[9] improves the index structure proposed by [10] for its dynamical operation by also previously calculating the similarities among the attributes. Redis produces the inverted index in real time, avoiding such preprocessing step.

To the best of our knowledge, we do not identify works that explore open source distributed platforms for online processing such as Spark or Apache Storm. Those platforms are capable of processing millions of inputs per second using a single computer [11]. The data stream in Apache Storm is defined using a topology structure to manage what must be processed at each instant.

5 Experimental Evaluation

This section we present experiments to evaluate our approach based on the effectiveness and efficient. First, we present an experiment evaluating the effectiveness of Redblock. Such experiment is important to identifying whether Redblock is able to correctly group the matching pairs correctly and subsequently construct the candidate pairs. In the following, we present the experiment illustrating the time efficiency.

5.1 Configuration

To carefully understanding of Redblock data flow, the use of the Storm UI feature was required. This is a graphic user interface that makes possible to observe the time flow of each bolt when the topology is being executed .

Traditional metrics were used to perform this experiment. The Precision(1) evaluates the pairs retrieved (The rate of pairs that were correctly identified). Recall(2) measures the rate of retrieved pairs compared to the total of duplicate pairs present in the database. F1(3) combines precision and recall in a single measurement. TP refers to True Positives which are pairs correctly labeled as positives by Redblock. False positives (FP) refer to negative pairs incorrectly labeled as positive. True negatives (TN) correspond to negatives correctly labeled as negative. Finally, false negatives (FN) refer to positive pairs incorrectly labeled as negative. The following metrics were applied:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

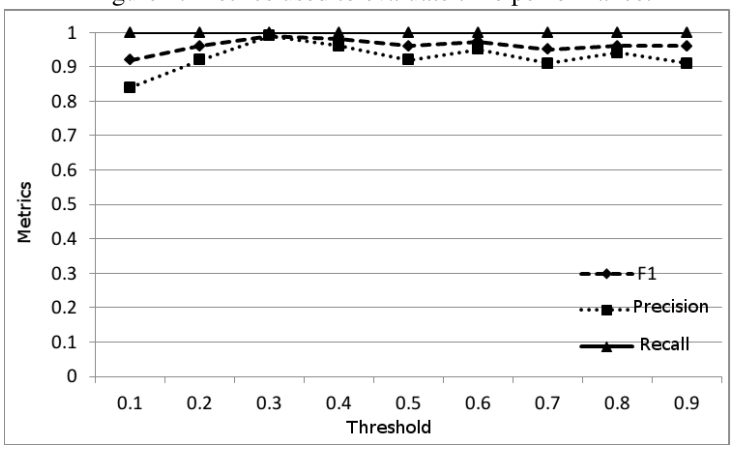
$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2(Precision * Recall)}{(Precision + Recall)} \tag{3}$$

A synthetic database containing 10.000 records was generated with the Febrl [12] tool. Such dataset is composed of 1.000 matching pairs. Synthetic bases are important because of its ability to control the number of duplicate pairs and the total of records that will be inserted into the database.

To evaluate Redblock's average time performance it was necessary to launch the experiment five times obtaining a higher accuracy of the data and a visible standard deviation.

Figure 4: Metrics used to evaluate time performance.



5.2 Effectiveness

Figure 4 shows the results of the metrics F1, precision, and recall obtained with Redblock. The X-axis defines the threshold that determines the size of the training set i.e., threshold 0.1 defines a training set with 10% of matching pairs (100 pairs) and the same number of negative pairs.

The threshold 0.1 resulted in an F1 value of 92%, increasing to 0.3 the F1 value is improved by 7%, reaching a maximum value. The thresholds above 0.3 resulted in a stable effectiveness in such dataset, that is, the increase in the training set did not result in a significant improvement of the results. In other words, the more labeled pairs do not improve the precision. It is important to realize that the recall remains at the maximum for all thresholds.

This initial experimentation illustrates that Redblock is able to promote a deduplication technique with high efficiency, finding all duplicate pairs as well as the classification method [4]

5.3 Efficiency

Table 7 states the efficiency of Redblock spouts and bolts regarding time consumed to be executed. More specifically, it shows the average time that a tuple spends to be executed by each step. The experiment was performed at least five times to calculate the average values for time performance and the standard deviation.

The full time that Redblock takes to perform its tasks is an average of 8.32 ms with a standard deviation of 1.97. As the table shows, the bolt Pair-Generator is the one that takes a longer processing time, around 60% processing time. This is due to the tuples that are processed by the bolt being a quadratic cost task (i.e., such bolt load the data from the database and perform a quadratic generation of pairs).

Accessing the database and recovering such values is a task which takes a longer processing time when compared to others bolts tasks. Line Saver Bolt and Split Sentence are also with a higher time than others bolts. In this case, it's recommended to increase the number of those bolts parallelism to obtain a better performance. The Line Saver bolt just saves the tuples in the database Redis for later retrieval in the Bolt Pair-generator. We believe that increasing the bolt parallelism we can reduce substantially the time-consuming.

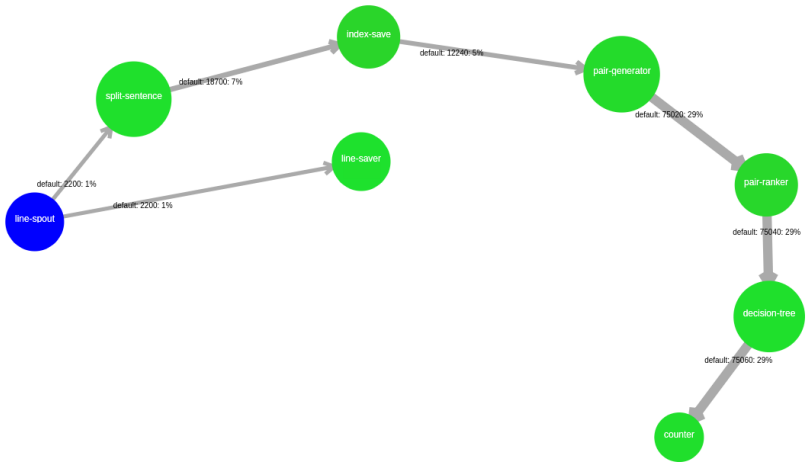
The data flow in Figure 5 is generated by Storm UI. It shows the processing time based on the node size and the tuple number of each bolt. We can observe that the bolt that produces more tuples (pair generator) are the one that is taking more time to perform its task. As stated, increasing the parallelism on this bolts may increase time performance.

[4]For continuous visualization of the enhancements in the code, it is available in Github. Link to access: <https://github.com/luanfelixpimentel/storminho>.

Table 7: Average timing values to perform a tuple.

Bolt	Time(ms)
Line-Spout	0.29+- (0.19)
Line-Saver	0.61+- (0.11)
WordIndex-Save	0.79+- (0.51)
Split Sentence	1.43+- (0.29)
Pair-Generator	5.02+- (1.52)
Pair-Ranker	0.28+- (0.18)
Decision Tree	0.04+- (0.01)
Full time	8.32+- (1.97)

Figure 5: Redblock Topology Graph.



6 Conclusion

This paper proposes a new tool for online deduplication focusing on the blocking process. The tool is entitled Redblock and combines a framework, Apache Storm, along with the Redis database to enable large scale data processing. In the initial experiment, it is possible to verify that Redblock maintained a high quality (high efficiency). The blocking steps and the classification were able to recover a high number of duplicate pairs without substantial losses of positive registers. Also, we illustrate the time consuming of each bolt to identify the bottlenecks.

The next steps involve performing maintenance on the tool in order to optimize the processing time of tuples with greater processing load and also test it with databases containing millions of records to analyze its efficiency in the massive processing of data. This strategy will turn possible to compare it with other tools in the state of the art present in the bibliography. Also, it will evaluate the performance of the tool when applied to a real database, such as Twitter which is an online tool and generates an indeterminate amount of data in real time.

7 Acknowledgment

Our special thanks are initially to the Federal University of Fronteira Sul, UFFS, Campus Chapecó, for the investment made in the project of this scientific and technological initiation. To the Regional Database School of 2017 (ERBD), for providing space, learning and recognition of the potential of the research. And, finally, the Applied Computing Brazilian Journal (RBCA), for the publication opportunity. We hope that the entire academic and industrial communities can gain knowledge through the study proposed with Redblock.

References

[1] CHRISTEN, P. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, IEEE, v. 24, n. 9, p. 1537–1555, 2012.

[2] MITRA, P. et al. Comparative study of name disambiguation problem using a scalable blocking-based framework. In: IEEE. *Digital Libraries, 2005. JCDL'05. Proceedings of the 5th ACM/IEEE-CS Joint Conference on*. [S.l.], 2005. p. 344–353.

[3] MANNING, C. D. et al. *Introduction to information retrieval*. [S.l.]: Cambridge university press Cambridge, 2008. v. 1.

[4] BIANCO, G. D. et al. A practical and effective sampling selection strategy for large scale deduplication. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 9, p. 2305–2319, 2015.

[5] ELMAGARMID, A. K.; IPEIROTIS, P. G.; VERYKIOS, V. S. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, IEEE, v. 19, n. 1, 2007.

[6] BAXTER, R. et al. A comparison of fast blocking methods for record linkage. In: CITESEER. *ACM SIGKDD*. [S.l.], 2003. v. 3, p. 25–27.

[7] BHATTACHARYA, I.; GETOOR, L. Query-time entity resolution. *Journal of Artificial Intelligence Research*, v. 30, p. 621–657, 2007.

[8] BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, p. 205–227, 2009.

[9] RAMADAN, B. et al. Dynamic similarity-aware inverted indexing for real-time entity resolution. In: SPRINGER. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. [S.l.], 2013. p. 47–58.

[10] WHANG, S. E. et al. Entity resolution with iterative blocking. In: ACM. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. [S.l.], 2009. p. 219–232.

[11] SRIKANTH, B.; REDDY, V. K. Efficiency of stream processing engines for processing bigdata streams. *Indian Journal of Science and Technology*, v. 9, n. 14, 2016.

[12] CHRISTEN, P. Febrl: a freely available record linkage system with a graphical user interface. In: *HDKM '08: Proceedings of the second Australasian workshop on Health data and knowledge management*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008. p. 17–25. ISBN 978-1-920682-61-3.