A Two-Phase Algorithm for Blocking-based Entity-resolution with Multiple Keys

Debanjali Biswas[1] and Sasikumar Enugunta[2]

[1] Informatik, Universität Bonn, Germany
debanjali.biswas@uni-bonn.de
[2] Informatik, Universität Bonn, Germany
s6saenug@uni-bonn.de

Abstract

Here, we describe a Two-Phase Algorithm for solving the problem of Blocking-based Entity Resolution(ER) with Multiple Keys. Entity Resolution is to detect all the records referring to the same entity. Nowadays, the huge collection of data has lead to the need for a Blocking-based ER. Most of studies related to Blocking-based ER has considered only one blocking key for an entity. However, in reality an entity can have multiple blocking keys. Entities with multiple blocking keys help in solving the ER problem more efficiently due to the fact that a pair entities will be a similar if they share several common keys.we use the methodology described in [HLC14].

This report was a part of the Distributed Big Data Analytics Lab course held in the Winter Semester 2018/19 in University of Bonn.

1 Introduction

The task of Entity resolution is to identify all the manifestations referring to the same entity ([HS95]; [BT06]; [CGK06]; [ELO08]; [VCL10]; [ZZYH10]). ER is a crucial part of many applications such as price comparisons, citation matching [PMM+03] and in data augmentation.

A blocking-based ER accelerates the similarity computations. In the blocking-based ER, each entity has a blocking key which is an attribute or a specific value computed using hashing or other means.It represents specific features of entities. The blocking-based ER comprises of two steps: a blocking step and a matching step. The blocking step splits all the entities in the dataset into several groups or blocks according to their blocking keys. All entities in a particular block have the same blocking key. After this step is the matching step, which computes the similarity between all the entity pairs in each block. The outcome is a set of all the entity pairs with high similarity. The blocking key helps in reducing the total number of similarity computations needed in the blocking-based ER.

Most of the research on blocking-based ER consider only one blocking key for an entity. However, an entity, in reality, consists of several blocking keys. An entity for example may be a record in a database, or a web-page in a web-site, or an article in a research repository, or a article in the newspaper,etc. Consider a product as an entity, this product can be classified into various categories consisting of several blocking keys. For a news article, the blocking keys may be each word in the article title along with author names. A blocking based ER can be performed more efficiently if we have more blocking keys, because two entities will be similar only if they have several similar keys.

Traditionally, an entity was considered to have only one blocking key. This resulted in a time-consuming blocking step. Now that an entity is associated with more than one key, the possibility of sharing common keys with other entities, which also has multiple keys, has immensely increases. Resulting in the increase in the number of entity pairs for matching.

Multiple blocking keys help us, in the blocking step, to sort out the entities required to be matched, therefore accelerating the matching step.

With the increase in the volume of data collection, there is a need for an efficient ER process to handle such large capacity data. This report describes a `Two-Phase Algorithm` for Blocking-based ER that can manage large datasets with multiple keys.

The rest of the report is organized as follows: Section 2 defines our problem. Section 3 gives an overview the proposed model. Section 4 details the methodology used in details. The experimental results are given in Section 5. Section 6 explains our individual contributions. The last Section 7 concludes this report with some future works.

2 Problem Definition

A dataset $D = \{E_1, E_2, , E_n\}$ contains n entities, where an entity $E_i (1 \leq i \leq n)$ has $\|E_i\|$ blocking keys. The blocking keys of E_i is represented by $B_{E_i} = \{k_1, k_2, , k_{\|E_i\|}\}$ and $\|E_i\| > 1$. Let kc be the minimum number of blocking keys of the entities in D, $kc > 1$. The ER problem is to find out all the entity pairs which are similar in D for a given similarity measure and a similarity threshold θ. An entity pair (E_i, E_j) is similar if $sim(E_i, E_j) \geq \theta$, where $sim(E_i, E_j)$ is the similarity value between E_i and E_j calculated using the similarity measure. Particularly in our problem, the characteristics of blocking keys implies, that $sim(E_i, E_j) < \theta$ if $\|B_{E_i} \cap B_{E_j}\| < kc$, i.e, E_i and E_j is considered similar only if the number of common blocking keys between them is greater than equal to kc. Hence, the similarity computation $sim(E_i, E_j)$ between entities E_i and E_j can be eliminated in the ER process if $\|B_{E_i} \cap B_{E_j}\| < kc$.[HLC14]

One of the similarity measure between entities E_i and E_j is the Jaccard similarity coefficient, which is defined as the size of their intersection divided by the size of their union.

$$sim(E_i, E_j) = \frac{\|B_{E_i} \cap B_{E_j}\|}{\|B_{E_i} \cup B_{E_j}\|}$$

3 Approach

3.1 Overview of the Model

The proposed model is a `Two-Phase Algorithm`: The first phase is **Combination-Based Blocking** and the second phase is **Load-Balanced Matching**. [Figure:1]

The `Combination-Based Blocking` is the entity distribution procedure. An entity generally has more than *kc Blocking Keys*. This phase create *kc*-combinations of keys, these combinations are used for key comparisons. After the blocking stage, the main idea is to balance the workload for the similarity computations, hence, we evenly partition the computation. This step is called the `Load-Balanced Matching`.

The Algorithm is using `Apache Spark RDD`[1] data-structure.

3.2 Spark Apache RDD

The collection of data is increasing continuously with an astonishing velocity. The huge volume of data, nowadays, effect the ER process performed on a single machine as the total number of similarity computations between entity pairs increase drastically. The problem aggregates if

[1] https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

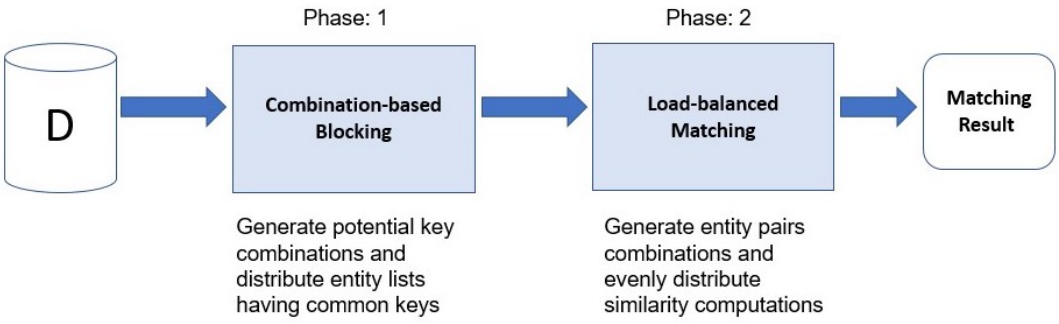

Figure 1: Overview of the Proposed Model

the data size assembles up to tetrabytes or petrabytes. Thus, making it a necessity to use parallel operations on a cluster. For this purpose, Apache Spark provides `Resilient Distributed Dataset (RDD)`, which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

4 Implementation

4.1 Dataset and Data Preprocessing

The dataset used is `Cora Information Extraction` dataset from `CiteSeerX`[2]. The dataset consists of research paper headers and citations, with labeled segments for authors, pubnum, title, institutions, venue, date, page numbers and several other fields. We extracted the pubnum and title of records from the dataset. The pubnum is used as the entity id. The records with no pubnum were removed. The title is split into list of words. Stop words (The list of stop words can be found here[3].) and punctuation were removed from this list to form the list of blocking keys.

We used Spark Dataframes[4] to parse the XML file and to extract the inputs. Then inputs where converted into Spark RDD.

4.2 Combination-Based Blocking

Each input, from the dataset, to the blocking phase consists of an entity id and its corresponding list of blocking keys. The `Combination-Based Blocking` generates all the kc-combinations from the set of blocking keys. Each combination is outputted with the id of the entity. For example, when kc = 2, $\langle(a, c), 2\rangle$, $\langle(a, e), 2\rangle$, and $\langle(c, e), 2\rangle$ are outputted for entity E_2 with $B_{E_2} = \{a, c, e\}$; when kc = 3 $\langle(a, c, e), 2\rangle$ is outputted for E_2 and $\langle(a, c, d), 1\rangle$, $\langle(a, c, e), 1\rangle$, $\langle(a, d, e), 1\rangle$, and $\langle(c, d, e), 1\rangle$ are outputted for E_1 with $B_{E_2} = \{a, c, d, e\}$. After, this the entities which have the same key pair are grouped together in a list.

The procedure further determines those entities having at least kc common keys to form a potential list for the matching step. Only entity lists having size more than kc are sent for

[2]http://csxstatic.ist.psu.edu/downloads/data
[3]https://kb.yoast.com/kb/list-stop-words/
[4]https://spark.apache.org/docs/latest/sql-programming-guide.html

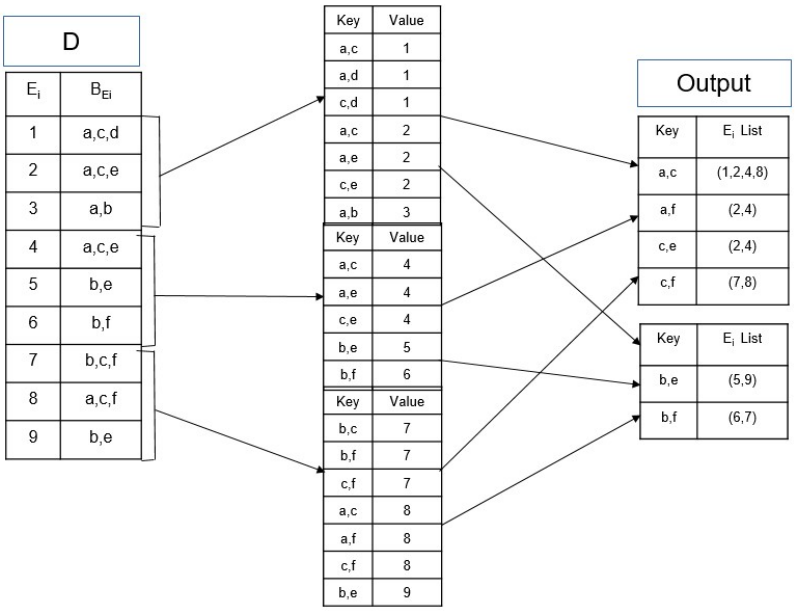

Figure 2: An Example for Combination-based Blocking

similarity computation in the matching step. Hence, we count the number of entities in the *entity_list* (*entity_list.count*) while adding entities of the same key pair K_b to it's associated *entity_list*.

An example of the Combination-Based Blocking phase, with 9 entities and kc=2, is shown in Figure:2.

4.3 Load-Balanced Matching

The Load-Balanced matching step computes similarities between entity pairs of all the 2-combinations of each entity list generated in the blocking step. In order to average the workload for the similarity computations, we use partitioning which evenly distributes the total number of comparisons. This is done according to a partition number (i.e. *partno*) that counts and marks each entity pair. Hence, the matching step first generates a (*partno*,*entitypair*), as seen in Figure:3. Then does a designated partitioning. Finally, similarity computations are done on each partition separately for all the entity pairs.

The similarity measure used here is Jaccard similarity. The matching steps outputs the entity pairs whose Jaccard similarity index is not less than the threshold θ. In Figure:3, we see an example of the `Load-Balanced Matching` phase where the input is the output from the blocking stage seen in Figure:3, here θ is considered to be 0.5.

5 Evaluation

We use the `Cora Information Extraction` dataset from `CiteSeerX`[5] for evaluation. We set the value of kc to be equal to 2 and the value of threshold θ to be equal to 0.2. The implemen-

[5]http://csxstatic.ist.psu.edu/downloads/data

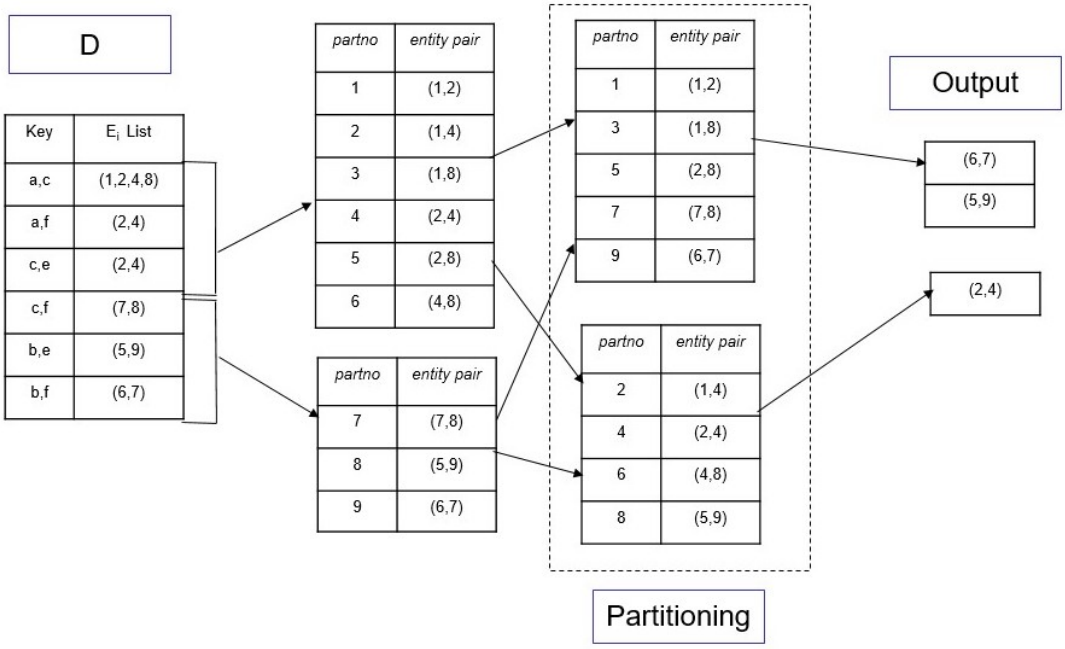

Figure 3: An Example for Load-balanced Matching

tation is done using Apache Spark on `Scala`[6].(It can be found here[7]).

The execution time for the `Combination-Based Blocking` phase take only a few milliseconds and for the `Load-Balanced Matching` phase takes only a second.

6 Project Timeline

This project report is a part of the Distributed Big Data Analytics Lab course held in the Winter Semester 2018/19 in University of Bonn. The project was scheduled from the month November to February. We were a team of two and both of us contributed equally in the project implementation.

7 Conclusion

In this report, we describe an algorithm to solve the entity resolution problem for big data analytics, using the Apache Spark's RDD. The characteristic of multiple attributes of a particular entity is utilized for an effective entity distribution to improve the entity resolution process. The model consists of the combination-based blocking phase and load-balanced matching phase. The combination-based blocking constructs potential key combinations and distributes entity lists having common keys. The load-balanced matching produces entity-pairs combinations and maintains the workload balance of similarity computations. The experimental results show that

[6]https://www.scala-lang.org/api/2.11.12/#package
[7]https://github.com/SmartDataAnalytics/MA-INF-4223-DBDA-Lab/tree/master/projects

the proposed algorithm may efficiently solve the entity resolution problem. Future works may include implementing this model for larger datasets.

8 Acknowledgement

We would like to thank Dr. Hajira Jabeen and Gezim Sejdiu for guiding us throughout the project. We would also like to thank Prof. Dr. Jens Lehmann for providing us with such an opportunity.

References

[BT06] David Guy Brizan and Abdullah Uz Tansel. A survey of entity resolution and record linkage methodologies. 2006.

[CGK06] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, pages 5–, Washington, DC, USA, 2006. IEEE Computer Society.

[ELO08] Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 265–268, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[HLC14] Sue-Chen Hsueh, Ming-Yen Lin, and Yi-Chun Chiu. A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. In *Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing - Volume 152*, AusPDC '14, pages 3–9, Darlinghurst, Australia, Australia, 2014. Australian Computer Society, Inc.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 127–138, New York, NY, USA, 1995. ACM.

[PMM+03] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart J Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1425–1432. MIT Press, 2003.

[VCL10] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 495–506, New York, NY, USA, 2010. ACM.

[ZZYH10] Qi Zhang, Yue Zhang, Haomin Yu, and Xuanjing Huang. Efficient partial-duplicate detection based on sequence matching. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 675–682, New York, NY, USA, 2010. ACM.