Incremental Blocking for Entity Resolution over Web Streaming Data

Tiago Brasileiro Araújo
Tampere University
Tampere, Finland
Federal University of Campina Grande
Campina Grande, Brazil
tiago.brasileiroaraujo@tuni.fi

Kostas Stefanidis
Tampere University
Tampere, Finland
konstantinos.stefanidis@tuni.fi

Carlos Eduardo Santos Pires
Federal University of Campina Grande
Campina Grande, Brazil
cesp@dsc.ufcg.edu.br

Jyrki Nummenmaa
Tampere University
Tampere, Finland
jyrki.nummenmaa@tuni.fi

Thiago Pereira da Nóbrega
State University of Paraíba
Campina Grande, Brazil
thiagonobrega@uepb.edu.br

ABSTRACT

The widespread use of information systems has become a valuable source of semi-structured data. In this context, Entity Resolution (ER) emerges as a fundamental task to integrate multiple knowledge bases or identify similarities between data items (i.e., entities). Since ER is an inherently quadratic task, blocking techniques are often used to improve efficiency. Beyond the challenges related to the data volume and heterogeneity, blocking techniques also face two other challenges: streaming data and incremental processing. To address these challenges, we propose PRIME, a novel incremental schema-agnostic blocking technique that utilizes parallelism to enhance blocking efficiency. The proposed technique deals with streaming and incremental data using a distributed computational infrastructure. To improve efficiency, the technique avoids unnecessary comparisons and applies a time window strategy to prevent excessive memory consumption.

CCS CONCEPTS

• **Information systems → Entity resolution**; *Semi-structured data.*

KEYWORDS

entity resolution, heterogeneous data, incremental processing

1 INTRODUCTION

Nowadays, numerous information systems produce a large amount of data continuously, and have become a valuable source of heterogeneous data. Such data provided by different data sources may have overlapping knowledge. A fundamental step to integrate multiple knowledge bases or identify similarities between entities is Entity Resolution (ER) [2, 3, 10]. ER identifies records (the entities) from several data sources (the entity collections) that refer to the same real-world entity. In the context of Web and heterogeneous data, ER faces with two Vs: *volume*, as it handles a growing number of entities; and *variety*, since different formats and schemes are used to represent the entity profiles [3]. To deal with volume, blocking and parallel computing are applied [6, 9]. Blocking groups similar entities into blocks and perform comparisons within each block. ER in parallel aims to minimize the overall execution time of the task by distributing the computational cost (i.e., the comparisons between entities) among the resources of a computational infrastructure [1]. Regarding variety, schema-agnostic blocking techniques are applied [2, 3]. Schema-agnostic techniques ignore the schemas and perform the blocking based on the data provided by entities. Among the existing schema-agnostic blocking techniques, Metablocking emerges as the most promising approach [11]: blocks form a weighted graph and pruning criteria are applied to remove edges with weight below a threshold, aiming to discard comparisons between entities with few chances of being considered a match.

Beyond the two Vs, we can detach two additional problems tackled by the ER task: streaming data and incremental processing [4, 7, 8, 12]. Processing of streaming data is commonly related to dynamic data sources (e.g., from Web Systems, Social Media, sensors), which are continuously updated. Incremental ER introduces new challenges, such as i) *how to manage the entities already processed since they can be infinitely?* and ii) *how to execute efficiently the ER task considering the whole stream of entities?* [7]. To handle heterogeneous streaming data, we propose the PaRallel-based Incremental MEtablocking (PRIME) technique, a promising schema-agnostic blocking technique capable to incrementally process entity profiles. To our knowledge, there is a lack of blocking techniques for addressing all challenges emerged in this scenario.

2 STREAMING METABLOCKING

The state-of-the-art blocking techniques do not work properly in scenarios involving incremental and streaming data since they were not conceived to deal with these situations. In this sense, we developed three blocking techniques capable to deal with streaming and incremental data: Streaming Metablocking, PRIME, and PRIME-windowed. The Streaming Metablocking technique is based on the same state-of-the-art parallel workflow proposed in [5]. However, Streaming Metablocking was adapted to take into account challenges involving streaming and incremental data. Since the technique considers the incremental behavior, we need to update the blocks in order to consider the new entities that are coming. Using a brute force strategy, after the arrival of a new increment, Streaming Metablocking needs to rearrange all blocks, including blocks that did not suffer any update. Clearly, this strategy is costly in terms of efficiency since it performs a huge number of unnecessary comparisons that have already been performed. To avoid this kind of unnecessary comparisons, Streaming Metablocking applies a store structure provided by Spark Streaming that considers only the data being updated in the current increment. Therefore, Streaming Metablocking only considers the blocks that suffered at least one update for the current increment. The reduction on the number of comparisons helps to minimize the computational cost of generating the blocking graph and performing the pruning step since the technique evaluates fewer number of entity pairs.

3 PRIME TECHNIQUE

Figure 1 is used to illustrate the PRIME workflow, which is divided into three steps: token extraction, blocking generation, and pruning.
Token Extraction Step. In this step, tokens are extracted from data. Each token is used as a blocking key. Initially, for each increment, blocking receives a pair of entity collections E_{D_1} and E_{D_2} provided by the data sources D_1 and D_2. Tokens are extracted from the attribute values of each entity. For each entity e, all tokens Λ_e associated with e are extracted and stored. This set of tokens Λ_e will be used in the following step to determine the similarity between the entities. Each token in Λ_e will be used as a blocking key and included in the map of blocks B following the format $\langle t, \langle e, \Lambda_e, D \rangle \rangle$, such that t is the blocking key, e represents the entity, Λ_e the set of tokens (i.e., blocking keys) and D the data source that provided e.

In Figure 1, there are two sets of data that represent the entities provided by two distinct increments. For the first increment (top of the figure), D_1 provides entities e_1 and e_2, while D_2 provides entities e_3 and e_4. In the token extraction step, the tokens A, B and C are extracted from entity e_1. For example, in a real-world scenario, the entity e_1 can be represented by $e_1 = \{\langle name, Alan\ Turing \rangle \langle nationality, British \rangle\}$. Thus, the tokens A, B and C represent the attribute values "*Alan*", "*Turing*" and "*British*", respectively. From the extracted tokens, all entities sharing the same token are grouped in the same block. Thus, each token is used as a blocking key. For instance, block b_1 is related to token A and contains entities e_1 and e_4 since both entities share the token A. Moreover, in this step, the entities are arranged in the format $\langle e, B \rangle$ such that e represents a specific entity and B denotes the set of blocks that contain entity e.
Blocking Generation Step. In this step, the weight graph is generated to define the level of similarity between entities. Initially, the blocks B generated in the previous step are received as input. For each blocking key k in B, the entities stored in the same block are compared. Thus, the entities provided from different data sources are compared to define the similarity ρ between them. The similarity is defined based on the number of co-occurring blocks (i.e., similar blocking keys) between the entities. After defining the similarity between entities, the entity pairs are inserted into the graph G, such that the similarity ρ represents the weight of the edge that links the entity pair. The blocks generated in this step are stored in memory to maintain them available for the next increments. In this sense, new entity blocks will be included or merged with the entity blocks previously stored. The blocking generation step is the most costly (in terms of computational costs) in the workflow since the comparison between the entities is performed in this step.

For instance, in Figure 1, block b_1 contains entities e_1 and e_4. Therefore, these entities must be compared to determine the similarity between them. The similarity between them is 1 since they co-occur in all blocks in which each one is contained. On the other hand, in the second increment (bottom of the figure), block b_1 receives entities e_5 and e_7. Thus, in the second increment, block b_1 contains entities e_1, e_4, e_5 and e_7 since all of them share token A. For this reason, entities e_1, e_4, e_5 and e_7 must be compared[1] with each other to determine the similarity between them.
Pruning Step. After the comparison between entities, the pruning criterion is applied to discard entity pairs with low similarity values. In the pruning step, a pruning criterion is applied to generate the set of high-quality blocks B'. Regarding the pruning criterion, the works [5, 11] propose different pruning algorithms that can be applied in this step. Particularly, in this work, we apply the WNP-based pruning algorithm [11] since it has achieved better results than its competitors [5, 11]. The WNP algorithm applies the vertex-centric pruning algorithm with a local weight threshold that is given by the average edge weight of each neighborhood. Thus, for each vertex in G, the WNP algorithm calculates the sum of the edge weights and the average of the edge weights. The average of the edge weights is applied as the local pruning threshold. Therefore, the neighborhood entities whose edge weight is greater than the local threshold are inserted in B'. The other entities (i.e., edge weight is lower than the local threshold) are discarded.

4 PRIME-WINDOWED TECHNIQUE

Considering the incremental challenges, PRIME faces limitations related to resource consumption (e.g., memory). Since PRIME stores the blocks previously generated to block the entities incrementally, the consumption of memory may increase infinitely as the increments are processed. This behavior directly results in memory-intensive consumption or problems related to memory overflow. The time window strategy is applied during the blocking generation step since in this step the generated blocks are stored in a data structure, to be used during the next increments. Therefore, the proposed strategy applies a time window to maintain the entities in the data structure for a certain time interval, preventing excessive memory consumption. However, it is worth mentioning that the application of a time window may affect negatively the effectiveness results since this strategy discards entities which exceed

[1]Following the restriction that only entities from different sources are compared.

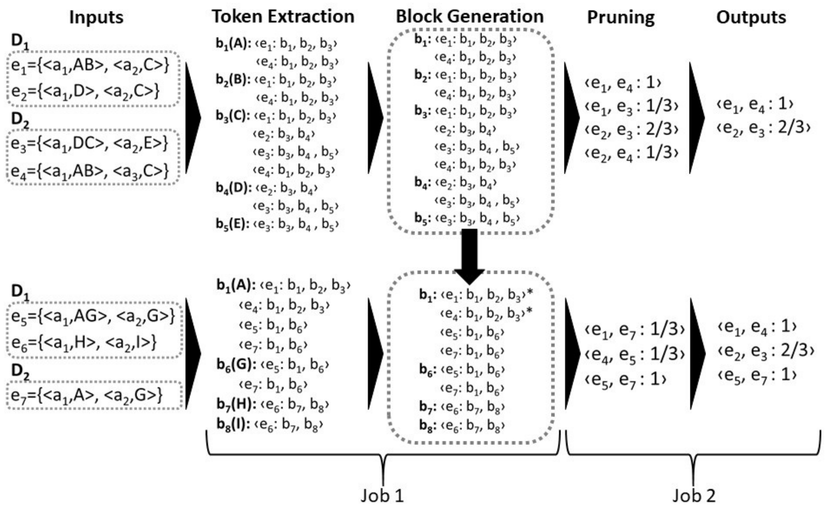

Figure 1: PRIME workflow.

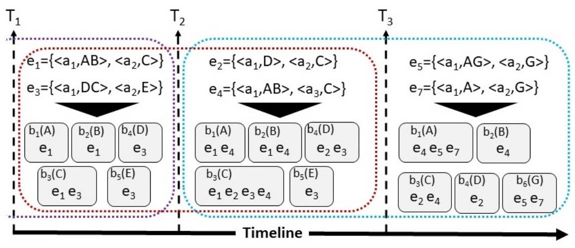

Figure 2: The time window strategy applied to PRIME.

the window time interval. Consequently, similar entities cannot be compared because they are not sent at the same time interval.

We will describe the PRIME-windowed technique by means of Figure 2. In this example, three increments are sent at three different times (i.e., T_1, T_2 and T_3). Moreover, the size of the time window (i.e., the time threshold) is given by the time interval of two increments. In the first increment (i.e., T_1), PRIME receives entities e_1 and e_3. After the blocking generation, blocks b_1, b_2, b_3, b_4 and b_5 are generated. For the second increment (i.e., T_2), PRIME receives entities e_2 and e_4. Considering the blocks already stored, the entity e_2 is added to blocks b_3 and b_4 and the entity e_4 is added to b_1, b_2 and b_3. Since the time threshold is two increments, the entities provided by the first increment should be discarded. Therefore, for the third increment (i.e., T_3), the entities e_1 and e_3 are removed from the blocks. Furthermore, it is important to detach that block b_5 is discarded since all entities contained in this block were removed. Finally, entities e_5 and e_7 are inserted into block b_1 and a new block b_6 is generated with the entities e_5 and e_7.

5 EXPERIMENTS

In this section, we evaluate PRIME[2] and Streaming Metablocking in terms of effectiveness and efficiency. We run our experiments on a cluster infrastructure with 13 nodes (one master and 12 slaves), each one with one core. Each node has an Intel(R) Xeon(R) 1.0GHz CPU, 6GB memory, runs the 64-bit Debian GNU/Linux OS with a 64-bit JVM and Apache Spark 2.0.1. We used the IMDB (27,615 entities, four attributes) vs. DBpedia (23,182 entities, seven attributes) datasets [13] with movies by imdb.com and dbpedia.org. To simulate the streaming behavior on the data, a data streaming sender was implemented. This data streaming sender reads the entities from the data sources and sends the entities to the Kafka producer. The Kafka producer provides the data, in a continuous way, to be consumed by PRIME for each τ time interval (i.e., increment).

To measure the effectiveness of blocking, three quality metrics have been applied: i) Pair Completeness (PC) - similar to recall - estimates the portion of matches that were identified, denoted by $PC = \frac{|M(B')|}{|M(D_1, D_2)|}$, where $|M(B')|$ is the amount of duplicate entities in the set of pruned blocks B' and $|M(D_1, D_2)|$ is the amount of duplicate entities between the data sources D_1 and D_2; ii) Pair Quality (PQ) - similar to precision - estimates the portion of executed comparisons that result in matches, denoted by $PQ = \frac{|M(B')|}{||B'||}$, where $||B'||$ is the amount of comparisons to be performed in the pruned blocks; iii) Reduction Ratio (RR) - estimates the portion of comparisons that are avoided in B' (i.e., $||B'||$) with respect to the comparisons guided by Cartesian product (i.e., $|D_1| \cdot |D_2|$) - is defined by $RR = 1 - \frac{||B'||}{|D_1| \cdot |D_2|}$. PC, PQ and RR take values in the interval [0, 1], with higher values indicating a better result. In terms of efficiency, we measure the whole execution time (i.e., including all steps) of PRIME considering the execution of all increments. In addition, we evaluate the memory consumption of the distributed infrastructure. Thus, we calculate the average of memory consumed (in percentage) by the nodes that compose the cluster.

To compare PRIME against Streaming Metablocking, we evaluate both techniques in a scenario where the increment size is the same for all increments. To this end, we set the number of entities per increment as 10% of the whole data source. Thus, for each data source, there are 10 increments containing 10% of entities from the

[2]https://github.com/brasileiroaraujo/Streaming

Table 1: Effectiveness results of PRIME, PRIME-windowed and Metablocking techniques.

	PRIME-windowed												PRIME/Metablocking		
	$2 \cdot \tau$			$4 \cdot \tau$			$6 \cdot \tau$			$8 \cdot \tau$			-		
Data Sources	**PC**	**PQ**	**RR**	**PC**	**PQ**	**RR**	**PC**	**PQ**	**RR**	**PC**	**PQ**	**RR**	**PC**	**PQ**	**RR**
IMDB-DBpedia	0.27	$3 \cdot 10^{-4}$	0.96	0.58	$3 \cdot 10^{-4}$	0.93	0.85	$3 \cdot 10^{-4}$	0.92	0.95	$3 \cdot 10^{-4}$	0.91	0.98	$3 \cdot 10^{-4}$	0,89

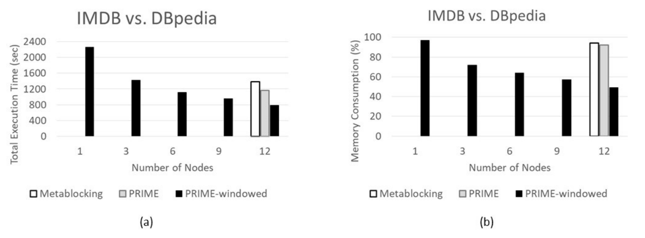

Figure 3: (a) Execution time of PRIME, PRIME-windowed and Metablocking techniques. (b) Memory consumption of PRIME, PRIME-windowed and Metablocking techniques for IMDB-DBpedia data sources.

data source. For PRIME-windowed, we vary the size of the time window in order to evaluate the impact of the window size on the PRIME technique. Thus, we apply the notation $\alpha \cdot \tau$ to determine the window size, such that α determines the number of time intervals τ (i.e., increments) covered by the window.

Efficiency. In this experiment, we evaluate the efficiency of the PRIME, PRIME-windowed and Streaming Metablocking techniques. The execution times are given by the average of five executions of each blocking technique. The Figure 3 (a) illustrates the results of the comparative analysis between the PRIME, PRIME-windowed and Metablocking techniques. We evaluate the execution time (in seconds) varying the number of nodes (one up to 12 nodes) in the distributed infrastructure. It is possible to detach that both PRIME techniques (with and without the time window strategy) outperformed the Metablocking. By comparing the PRIME-windowed technique against the PRIME one, it is possible to notice a small decreasing in the execution time, since the former applies the time window strategy (described in Section 5) and, therefore, takes into account a fewer number of entities to be compared.

In this experiment, PRIME (without time window) and Metablocking are not able to be executed when less than 12 nodes are used by the distributed infrastructure. It occurs since these techniques consider all the entities sent in all increments. Therefore, the data structure that stores the blocks previously generated requires a large amount of memory of the distributed infrastructure. For this reason, PRIME and Metablocking have enough memory to be executed only when 12 nodes are used. This limitation leads us to propose the PRIME-windowed technique which handles a higher amount of entities efficiently. The maximum window size applied was $4 \cdot \tau$ since the application of bigger window sizes exceeds the memory consumption for one node.

We also evaluate the memory consumption. We vary the number of nodes used by the distributed infrastructure, as depicted in Figure 3 (b). The memory consumption for PRIME and Metablocking achieve (on average) around 95% of all available memory in the nodes, when 12 nodes are applied. On the other hand, PRIME-windowed (with a window size of $4 \cdot \tau$) consumes around 47% of all available memory in the nodes. However, as the number of nodes decreases (consequently, the amount of total memory available decreases), the average memory consumption increases.

Effectiveness. Regarding effectiveness results, Table 1 illustrates the PC, PQ and RR metrics for PRIME, PRIME-windowed and Streaming Metablocking techniques. For the PRIME-windowed technique, the effectiveness results are shown for different window sizes, $2 \cdot \tau$ to $8 \cdot \tau$. If PRIME and Metablocking techniques receive the same input and apply the same pruning criterion, they will generate the same output. For this reason, notice that the effectiveness results are also the same for both techniques.

For PC, PRIME and Metablocking present promising results for both incremental size scenarios, achieving more than 96%. However, since PRIME-windowed considers only the entities sent between a time interval according to the window size, PC is directly affected. Intuitively, the larger the window size, the better the PC value. Therefore, PC tends to be low for small window sizes and higher for large window sizes. Since PQ is different from Precision, commonly used to evaluate the results of ER. PQ evaluates the accuracy of generated blocks. Thus, PQ values achieved by the PRIME and Metablocking techniques, as depicted in Table 1, are satisfactory results for blocking [1, 5, 11].

RR estimates the relative decrease in the number of comparisons conveyed by blocking techniques. RR is fundamental for measuring the efficiency gains of ER since it directly estimates the percentage of comparisons that are avoided after blocking. PRIME presents promising results in terms of RR, achieving RR values higher than 0.89. Therefore, PRIME is able to enhance the efficiency of ER since it reduces up to 90% the number of comparisons to be executed in the ER task.

6 SUMMARY

In this work, we propose PRIME, a novel incremental schema-agnostic blocking technique that utilizes parallelism to enhance blocking efficiency. PRIME is able to deal with streaming and incremental scenarios as well as minimize the challenges related to both scenarios. Based on the experimental results, we can highlight that PRIME presents better results regarding efficiency than the state-of-the-art technique (i.e., Streaming Metablocking) without negative impacts on the effectiveness.

ACKNOWLEDGMENT

This work has been supported by CAPES-Brazil (grant 88881.187788/2018-01) and Virpa D project funded by Business Finland.

REFERENCES

[1] Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, and Thiago Pereira da Nóbrega. 2017. Spark-based Streamlined Metablocking. In *ISCC*.

[2] Peter Christen. 2012. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer Science & Business Media.

[3] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. Entity Resolution in the Web of Data. *Synthesis Lectures on the Semantic Web* (2015).

[4] Dimas Cassimiro do Nascimento, Carlos Eduardo Santos Pires, and Demetrio Gomes Mestre. 2018. Heuristic-based approaches for speeding up incremental record linkage. *Journal of Systems and Software* 137 (2018), 335–354.

[5] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems* 65 (2017), 137–157.

[6] V. Efthymiou, G. Papadakis, K. Stefanidis, and V. Christophides. 2019. MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities. In *EDBT*.

[7] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. 2014. Incremental record linkage. *Proceedings of the VLDB Endowment* 7, 9 (2014), 697–708.

[8] Kun Ma and Bo Yang. 2017. Stream-based live entity resolution approach with adaptive duplicate count strategy. *International Journal of Web and Grid Services* 13, 3 (2017), 351–373.

[9] Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, Dimas Cassimiro Nascimento, Andreza Raquel Monteiro de Queiroz, Veruska Borges Santos, and Tiago Brasileiro Araujo. 2017. An efficient spark-based adaptive windowing for entity matching. *Journal of Systems and Software* 128 (2017), 1–10.

[10] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection.* Morgan & Claypool Publishers.

[11] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. 2016. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. EDBT.

[12] Xiangnan Ren and Olivier Curé. 2017. Strider: A hybrid adaptive distributed RDF stream processing engine. In *International Semantic Web Conference*.

[13] Giovanni Simonini, Sonia Bergamaschi, and HV Jagadish. 2016. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB* 9, 12 (2016), 1173–1184.