information

Article

Incremental Entity Blocking over Heterogeneous Streaming Data

Tiago Brasileiro Araújo [1,2,3,*], Kostas Stefanidis [3], Carlos Eduardo Santos Pires [1], Jyrki Nummenmaa [3] and Thiago Pereira da Nóbrega [1]

1 Academic Unit of Systems and Computing, Federal University of Campina Grande, Campina Grande 58429-900, Brazil
2 Federal Institute of Paraíba, Monteiro 58500-000, Brazil
3 Faculty of Information Technology and Communication Sciences, Tampere University, 33100 Tampere, Finland
* Correspondence: tiagobrasileiro@copin.ufcg.edu.br

Abstract: Web systems have become a valuable source of semi-structured and streaming data. In this sense, Entity Resolution (ER) has become a key solution for integrating multiple data sources or identifying similarities between data items, namely entities. To avoid the quadratic costs of the ER task and improve efficiency, blocking techniques are usually applied. Beyond the traditional challenges faced by ER and, consequently, by the blocking techniques, there are also challenges related to streaming data, incremental processing, and noisy data. To address them, we propose a schema-agnostic blocking technique capable of handling noisy and streaming data incrementally through a distributed computational infrastructure. To the best of our knowledge, there is a lack of blocking techniques that address these challenges simultaneously. This work proposes two strategies (attribute selection and top-*n* neighborhood entities) to minimize resource consumption and improve blocking efficiency. Moreover, this work presents a noise-tolerant algorithm, which minimizes the impact of noisy data (e.g., typos and misspellings) on blocking effectiveness. In our experimental evaluation, we use real-world pairs of data sources, including a case study that involves data from Twitter and Google News. The proposed technique achieves better results regarding effectiveness and efficiency compared to the state-of-the-art technique (metablocking). More precisely, the application of the two strategies over the proposed technique alone improves efficiency by 56%, on average.

Keywords: entity resolution; incremental processing; parallel computing; schema-agnostic blocking techniques; streaming data

check for updates

Citation: Araújo, T.B.; Stefanidis, K.; Pires, C.E.S.; Nummenmaa, J.; da Nóbrega, T.P. Incremental Entity Blocking over Heterogeneous Streaming Data. *Information* **2022**, *13*, 568. https://doi.org/10.3390/info13120568

Academic Editor: Annalisa Appice

Received: 7 October 2022
Accepted: 2 December 2022
Published: 5 December 2022

1. Introduction

Currently, there is an increasing number of information systems that produce large amounts of data continuously, such as Web systems (e.g., digital libraries, knowledge graphs, and e-commerce), social media (e.g., Twitter and Facebook), and the Internet of Things (e.g., mobiles, sensors, and devices) [1]. These applications have become a valuable source of heterogeneous data [2,3]. These kinds of data present a schema-free behavior and can be represented in different formats (e.g., XML, RDF, and JSON). Commonly, data are provided by different data sources and may have overlapping knowledge. For instance, different types of social media will report the same event and generate similar mass data. In this sense, Entity Resolution (ER) emerges as a fundamental step to support the integration of multiple knowledgebases or identify similarities between entities. The ER task aims to identify records (i.e., entity profiles) from several data sources (i.e., entity collections) that refer to the same real-world entity (i.e., similar/correspondent entities) [4–6].

The ER task commonly includes four steps: blocking, comparison, classification, and evaluation [7]. In the former, to avoid the quadratic cost of the ER task (i.e., comparisons guided by the Cartesian product), blocking techniques are applied to group similar entities into blocks and perform comparisons within each block. In the comparison step, the actual entity pair comparison occurs, i.e., the entities of each block are pairwise compared using a variety of comparison functions to determine the similarity between them. In the classification step, based on the similarity level, the entity pairs are classified into matches, non-matches, and potential matches (depending on the decision model used). In the evaluation step, the effectiveness of the ER results and the efficiency of the process are evaluated. This work focuses on the blocking step.

The heterogeneity of the data compromises the block generation (by the blocking techniques) since the entity profiles hardly share the same schema. Therefore, traditional blocking techniques (e.g., sorted neighborhood and adaptive window) do not possess satisfactory effectiveness because blocking is based on a fixed entity profile schema [8]. In turn, the heterogeneous data challenge is addressed by schema-agnostic blocking techniques, which disregard attribute names and consider the values related to the entity attributes to perform blocking [5]. Furthermore, we tackle three other challenges related to the ER task and, consequently, the blocking techniques: streaming data, incremental processing, and noisy data [9–11].

Streaming data are related to dynamic data sources (e.g., from Web systems, social media, and sensors), which are continuously updated. When blocking techniques receive streaming data, we assume that not all data (from the data sources) are available at once. Therefore, blocking techniques need to group the entities as they arrive, also considering the entities already blocked previously.

Incremental blocking is related to receiving data continuously over time and re-processing only the portion of the generated blocks (i.e., that store similar entities) that were affected by the data increments. For this reason, it commonly suffers from resource consumption issues (e.g., memory and CPU) since ER approaches need to maintain a large amount of data in memory [9,10,12]. This occurs due to the fact that incremental processing consumes information processed in previous increments. Considering this behavior, it is necessary to develop strategies that manage computational resources. In this sense, when blocking techniques face these scenarios, memory consumption tends to be the biggest challenge that is handled by incremental techniques [13,14]. Considering that streaming data are frequently processed in an incremental way [2], the challenges are strengthened when the ER task deals with heterogeneous data, streaming data, and incremental processing simultaneously. For this reason, the development of efficient incremental blocking techniques able to handle streaming data appears as an open problem [13,15]. To improve efficiency and provide resources for incremental blocking techniques, parallel processing can be applied [16]. Parallel processing distributes the computational costs (i.e., to block entities) among the various resources (e.g., computers or virtual machines) of a computational infrastructure to reduce the overall execution time of blocking techniques.

Concerning noisy data, in practical scenarios, people are less careful with the lexical accuracy of content written in informal virtual environments (e.g., social networks) or when they are under some kind of pressure (e.g., business reports) [17]. For these reasons, real-world data often present noise that can impair data interpretations, data manipulation tasks, and decision making [18]. As stated, in the ER context, noisy data enhance the challenge of determining the similarities between entities. This commonly occurs in scenarios where the similarity between entities is based on the lexical aspect of their attribute values, which is the case in a vast number of blocking techniques [19–24]. In this sense, the two most common types of noise in the data are considered in this work: typos and misspelling errors [17].

Considering the challenges described above, this research proposes a parallel-based blocking technique able to process streaming data incrementally. The proposed blocking technique is also able to process noisy data without a considerable impact on the effectiveness of the blocking results. Moreover, we propose two strategies to be applied in the blocking technique: attribute selection and top-*n* neighborhood. Both strategies intend to enhance the efficiency of the blocking technique by quickly processing entities (sent continuously) and preventing the blocking from consuming resources excessively. Therefore, the general hypothesis of our work is to evaluate whether (or not) the application of the proposed blocking technique is able to improve the efficiency of the ER task without decreasing the effectiveness in streaming data scenarios.

Although the proposed blocking technique follows the idea behind token-based techniques such as those in [21,22], the latter are neither able to handle streaming data nor perform blocking incrementally. Since they were originally developed for batch data, they do not take into account the challenges related to incremental processing and streaming data. To the best of our knowledge, there is a lack of blocking techniques that address all the challenges faced in this work. Among the recent works, we can highlight the work in [19], which proposes a schema-agnostic blocking technique to handle streaming data. However, this work presents several limitations related to blocking efficiency and excessive resource consumption. Therefore, as part of our research, we propose an enhancement of the technique proposed in [19], by offering an efficient schema-agnostic blocking technique able to incrementally process streaming data. Overall, the main contributions of our work are:

1. An incremental and parallel schema-agnostic blocking technique able to deal with streaming and incremental data, as well as minimize the challenges related to both scenarios;
2. An attribute selection algorithm, which discards the superfluous attributes of the entities, to enhance efficiency and minimize resources consumption;
3. A top-n neighborhood strategy, which maintains only the n most similar entities (i.e., neighbor entities) of each entity, improving the efficiency of the proposed technique;
4. A noise-tolerant algorithm, which generates hash values from the entity attributes and allows the generation of high-quality blocks even in the presence of noisy data;
5. An experimental evaluation applying real-world data sources to analyze the proposed technique in terms of efficiency and effectiveness;
6. A real-world case study involving data from Twitter and Google News to evaluate the application of the attribute selection and top-n neighborhood strategies over the proposed technique in terms of effectiveness.

The rest of the paper is structured as follows: Section 2 formalizes the problem statement related to the understanding of this work. In Section 3, we present the most relevant works available in the literature related to the field addressed in this paper. Section 4 presents the parallel-based blocking technique and describes the workflow to process streaming data. Section 5 describes the time-window strategy, which is applied in the proposed technique to avoid the excessive consumption of computational resources. Section 6 presents the attribute selection strategy, which discards superfluous attributes in order to improve efficiency. In Section 7, we discuss the experimental evaluation and in Section 8, the results of the case study are addressed. Finally, Section 9 concludes the paper along with directions for future works.

2. Problem Statement

Given two entity collections $E_1 \in D_1$ and $E_2 \in D_2$, where D_1 and D_2 represent two data sources, the purpose of ER is to identify all matches among the entities (e) of the data sources [5,7]. Each entity $e \in E_1$ or $e \in E_2$ can be denoted as a set of key-value elements that models its attributes (a) and the values (v) associated with each attribute $e = [(a_1, v_1), (a_2, v_2), \dots, (a_n, v_n)]$. Given that $sim(e_i, e_k)$ is the function that determines the similarity between the entities e_i and e_k, and μ is the threshold ($\mu \in \mathbb{Q}$) that determines whether the entities e_i and e_k are considered a match (i.e., truly similar entities), the task of identifying similar entities (ER) can be denoted as $ER(D_1, D_2) = \{(e_i, e_k) \mid e_i \in D_1,\ e_k \in D_2 \mid sim(e_i, e_k) \geq \mu\}$.

In traditional ER, entity comparisons are guided by the Cartesian product (i.e., $D_1 \times D_2$). For this reason, blocking techniques are applied to avoid the quadratic $O(n^2)$ asymptotic complexity of ER. Note that the blocking output is a set of blocks containing entities $B = \{b_1, b_2, \ldots, b_{|B|}\}$ such that, $\bigcup_{b \in B}(b) = D_1 \cup D_2$. Thus, each block b can be denoted as $b = \{(e_i, e_k) \mid e_i \in D_1,\ e_k \in D_2 \mid sim(e_i, e_k) \geq \kappa\}$, where κ is the threshold of the blocking criterion.

Since this work considers that the data are sent incrementally, the blocking is divided into a set of increments $I = \{i_1, i_2, i_3, \ldots, i_{|I|}\}$, where $|I|$ is the number of increments and each $i \in I$ contains entities from D_1 and D_2. Given that the sources provide data in streams, each increment is associated with a time interval τ. Thus, the time interval between two increments is τ, i.e., $time(i_k) - time(i_{k-1}) = \tau$, where $time$ returns the timestamp that the increment was sent and $k \in [2, |I|]$ is the index of the increment $i \in I$. For instance, in a scenario where data sources provide data every 30 s, we can assume that τ = 30 s.

For each $i \in I$, each data source sends an entity collection $E_i = \{e_1, e_2, e_3, \ldots, e_{|E_i|}\}$. Thus, $E_i = E_{iD_1} \cup E_{iD_2}$, where E_{iD_1} contains the entities of D_1 and E_{iD_2} contains the entities of D_2 for the specific increment i. Since the entities can follow different loose schemes, each $e \in E_i$ has a specific attribute set and value associated with each attribute denoted by $A_e = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \langle a_3, v_3\rangle, \ldots, \langle a_{|A_e|}, v_{|A_e|}\rangle\}$ such that $|A_e|$ is the number of attributes associated with e. When an entity e does not present a value for a specific attribute, it implies the absence of this attribute. For this reason, $|A_e|$ can be different for a distinct e. As stated in Section 1, an attribute selection algorithm, which discards the superfluous attributes from the entities, is proposed in this work. Hence, when the attribute selection algorithm is applied, the attributes in A_e may be removed, as discussed later in Section 6. In this sense, the set of attributes after the application of the attribute selection is given by $A_{e\Upsilon} = \bigcup(\langle a, v\rangle \mid \Upsilon(a) = false\ \wedge\ \langle a, v\rangle \in A_e)$, where Υ represents the attribute selection algorithm that returns true if the attribute (as well as its value) should be removed and *false* otherwise.

In order to generate the entity blocks, tokens (e.g., keywords) are extracted from the attribute values. Note that the blocking techniques aim to group similar entities (i.e., $e \in E$) to avoid the quadratic cost of the ER task, which compares all entities of D_1 with all entities of D_2. To group similar entities, blocking techniques such as token-based techniques can determine the similarity between entities based on common tokens extracted from the attribute values. For token extraction, all tokens associated with an entity e are grouped into a set Λ_e, i.e., $\Lambda_e = \bigcup(\Gamma(v) \mid \langle a, v\rangle \in A_e)$, such that $\Gamma(v)$ is a function to extract the tokens from the attribute value v. For instance, $\Gamma(v)$ can split the text of v using whitespace and remove stop words (e.g., "and", "is", "are", and "the"). The set of the remaining words is considered the set of tokens. Stop words are not useful as tokens because they are common words and hardly determine a similarity between entities. In scenarios involving noisy data, we apply LSH algorithms to generate a hash value for each token. Therefore, a hash function H should be applied to $\Gamma(v)$, i.e., $H(\Gamma(v))$, to generate the set of hash values Λ_{e_h} instead of the set of tokens Λ_e. In summary, when noisy data are considered, Λ_e should be replaced with $\Lambda_{e_h} = \bigcup(H(\Gamma(v)) \mid \langle a, v\rangle \in A_e)$.

To generate the entity blocks, a similarity graph $G(X, L)$ is created in which each $e \in E_i$ is mapped to a vertex $x \in X$ and a non-directional edge $l \in L$ is added. Each l is represented by a triple $\langle x_1, x_2, \rho\rangle$ such that x_1 and x_2 are the vertices of G and ρ is the similarity value between the vertices. Thus, the similarity value between two vertices (i.e., entities) x_1 and x_2 is denoted by $\rho = \Phi(x_1, x_2)$. In this work, the similarity value between x_1 and x_2 is given by the average of the common tokens between them $\Phi(x_1, x_2) = \frac{|\Lambda_{x_1} \cap \Lambda_{x_2}|}{max(|\Lambda_{x_1}|, |\Lambda_{x_2}|)}$. This similarity function is based on the functions proposed in [11,21]. In scenarios involving noisy data, the similarity value between x_1 and x_2 is given by the average of the common hash values between them. Since the idea is not to compare (or link) two entities from the same data source, x_1 and x_2 must be from different data sources. Therefore, x_1 is a vertex generated from an entity $e \in E_{iD_1}$ and x_2 is a vertex generated from another entity $e \in E_{iD_2}$.

The attribute selection algorithm aims to discard superfluous attributes in the sense of providing tokens that will hardly assist the blocking task to find similar entities. For instance, consider two general entities $e_1 \in E_{iD_1}$ (i.e., $e_1 \in D_1$) and $e_2 \in E_{iD_2}$ (i.e., $e_2 \in D_2$). Let $A_{e_1} = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \langle a_3, v_3\rangle\}$ and $A_{e_2} = \{\langle a_4, v_4\rangle, \langle a_5, v_5\rangle\}$; if only the attributes a_1 and a_5 have the same meaning (i.e., contain attribute values regarding the same content), the tokens generated from a_2, a_3, and a_4 will hardly result in blocking keys that truly represent the similarity between the entities. To this end, the attribute selection strategy is applied.

The idea behind attribute selection is to determine the similarity among attributes based on their values. After that, the attributes (and, consequently, their values) that do not have similarities with others are discarded. Formally, all attributes associated with the entities belonging to data sources D_1 and D_2 are extracted. Moreover, all values associated with the same attribute (a_i) are grouped into a set $V_{a_i} = \bigcup_{e \in D}(v \mid \langle a_i, v\rangle \in A_e)$. In turn, the pair $\langle a_i, V_{a_i}\rangle$ represents the set of values associated with a specific attribute a_i. Let R be the list of discarded attributes, which contains the attributes that do not have similarities with others. Formally, given two attributes $a_i \subset D_1$ and $a_j \subset D_2$, $a_i \in R \iff \nexists a_j : sim(V_{a_i}, V_{a_j}) \geq \beta$, where $sim(V_{a_i}, V_{a_j})$ calculates the similarity between the sets V_{a_i} and V_{a_j} and β is a given threshold. In this work, we compute the value for β based on the similarity value of all attribute pairs provided by D_1 and D_2. From the mean of the similarity between the attributes, β assumes the value of the first quartile. Then, all attribute pairs whose similarities are in the first quartile (i.e., comprise 25% of the attribute pairs with the lowest similarities) are discarded. The choice of the β value was based on previous experiments, which indicates a negative influence on the effectiveness results when the β value is given after the first quartile. For this reason, β assumes the value of the first quartile to avoid significantly impacting the effectiveness results. Then, the token extraction considering attribute selection is given by $\Lambda_e = \bigcup(\Gamma(v) \mid \langle a, v\rangle \in A_e \land a \notin R)$. Similarly, when noisy data are considered, the token extraction considering attribute selection is given by $\Lambda_{e_h} = \bigcup(H(\Gamma(v)) \mid \langle a, v\rangle \in A_e \land a \notin R)$.

In ER, a blocking technique aims to group the vertices of G into a set of blocks denoted by $B_G = \{b_1, b_2, \ldots, b_{|B_G|}\}$. In turn, a pruning criterion $\Theta(G)$ is applied to remove entity pairs with low similarities, resulting in a pruned graph G'. For pruning, a criterion is applied to G (i.e., $\Theta(G)$) so that the triples $\langle x_1, x_2, \rho\rangle$ whose $\rho < \theta$ are removed, generating the pruned graph G'. Thus, the vertices of G' are grouped into a set of blocks denoted by $B'_{G'} = \{b'_1, b'_2, \ldots, b'_{|B'_{G'}|}\}$, such that $\forall b' \in B'_{G'}(b' = \{x_1, x_2, \ldots, x_{|b'|}\})$, $\forall\langle x_1, x_2\rangle \in b' : \exists\langle x_1, x_2, \rho\rangle \in L$ and $\rho \geq \theta$, where θ is a similarity threshold defined by a pruning criterion $\Theta(G)$. In Section 4.2.3, how a pruning criterion $\Theta(G)$ determines the value for θ in practice is presented.

However, when blocking techniques deal with streaming and incremental challenges, other theoretical aspects need to be considered. Intuitively, each data increment, denoted by ΔE_i, also affects G. Thus, we denote the increments over G by ΔG_i. Let $\{\Delta G_1, \Delta G_2, \ldots, \Delta G_{|I|}\}$ be a set of $|I|$ data increments on G. Each ΔG_i is directly associated with an entity collection E_i, which represents the entities in the increment $i \in I$. The computation of B_G for each ΔG_i is performed on a parallel distributed computing infrastructure composed of multiple nodes (e.g., computers or virtual machines). Note that to compute B_G for a specific ΔG_i, previous ΔG are considered, following the incremental behavior of G. In this context, it is assumed that $N = \{n_1, n_2, \ldots, n_{|N|}\}$ is the set of nodes used to compute B_G. The execution time using a single node $n \in N$ is denoted by $T^n_{\Delta G_i}(B_G)$. On the other hand, the execution time using the whole computing infrastructure N is denoted by $T^N_{\Delta G_i}(B_G)$.
Execution time for parallel blocking. Since blocking is performed in parallel over the infrastructure N, the whole execution time is given by the execution time of the node that demanded the longest time to execute the task for a specific increment ΔG_i: $T^N_{\Delta G_i}(B_G) = max(T^n_{\Delta G_i}(B_G)), n \in N$.

Time restriction for streaming blocking. Considering the streaming behavior, where each increment arrives in each τ time interval, it is necessary to determine a restriction on the execution time to process each increment, given by $T^N_{\Delta G_i}(B_G) \leq \tau$.

This restriction aims to prevent the blocking execution time from overcoming the time interval of each data increment. Note that the time restriction is related to streaming behavior, where data are produced continuously. To achieve this restriction, blocking must be performed as quickly as possible. As stated previously, one possible solution to minimize the execution time of the blocking step is to execute it in parallel over a distributed infrastructure.

Blocking efficiency in distributed environments. Given the inputs E_i, N, B_G, ΔG_i, Γ, Φ, and Θ, the blocking task aims to generate $B'_{G'}$ over a distributed infrastructure N in an efficient way. To enhance the efficiency, the blocking task needs to minimize the value of $\Delta T = |N| - \frac{T^n_{\Delta G_i}(B_G)}{T^N_{\Delta G_i}(B_G)}$. For instance, considering a distributed infrastructure with N = 10 nodes and 10 s as the required time to perform the blocking task using only one node (i.e., $T^n_{\Delta G_i}(B_G)$), the ideal execution time using all nodes (i.e., $T^N_{\Delta G_i}(B_G)$) should be one second. Thus, ideally, the best minimization is $\Delta T = 0$. Due to the overhead required to distribute the data in each node and the delay in the communication between the nodes, $\Delta T = 0$ is practically unreachable. However, the minimization of ΔT is important to provide efficiency to the blocking task.

Table 1 highlights the goals regarding the stated problems, which are exploited throughout this paper. Furthermore, Table 2 summarizes the symbols used throughout the paper.

Table 1. Summary of the problem statement.

Input:	$\langle D_1, D_2, I, \Delta G_i, N, \tau, \Gamma, \Phi, \Theta \rangle$
Generate:	B_G, *using* Γ *and* Φ *over* N
Output:	$B'_{G'}$, *using* Θ *over* N
Minimize:	$max(T^n_{\Delta G_i}(B_G))$
Following:	$T^N_{\Delta G_i}(B_G) \leq \tau$
Minimize:	$|N| - \frac{T^n_{\Delta G_i}(B_G)}{T^N_{\Delta G_i}(B_G)}$
Maximize:	$Effectiveness(B'_{G'})$

Table 2. Summary of symbols.

Symbol	**Description**
D_1, D_2	Input data sources
I	Set of increments
i	Increment for a specific moment
τ	Time interval between increments
E_i	Entity collection for the specific increment i
A_e	Set of attributes with their associated values for the entity e
Υ	Attribute selection function
Λ_e	Tokens associated with an entity e
Γ	Token extraction function
H	Hash function
G	Similarity graph

Table 2. *Cont.*

Symbol	Description
ΔG_i	Increments over G; each ΔG_i is associated with an entity collection E_i
Φ	Similarity function applied to vertices of G
B_G	Blocks from the similarity graph G
Θ	Pruning criterion function
G'	Pruned graph
$B'_{G'}$	Blocks from the pruned graph G'
N	Set of nodes in a distributed computing infrastructure

3. Related Works

According to the taxonomy proposed in [5], blocking techniques employed in the ER task are classified into two broad categories: *schema-aware* (e.g., canopy clustering [25,26], sorted neighborhood [27,28], and adaptive window [29]) and *schema-agnostic* (e.g., token blocking [30,31], metablocking [21,22], attribute clustering blocking [8], and attribute-match induction [11,20,32]). Concerning the execution model, several schema-agnostic blocking techniques have been proposed to deal with heterogeneous data in standalone [20,31,33,34] and parallel [11,16,22,35] modes. In this sense, the research focus of this work is to explore open topics related to *schema-agnostic* blocking techniques.

3.1. Blocking for Heterogeneous Data

In the context of heterogeneous data, entities rarely follow the same schema. For this reason, the block generation (in blocking techniques) is compromised. Therefore, traditional blocking techniques (e.g., sorted neighborhood [27] and adaptive window [29]) do not present satisfactory effectiveness since blocking is based on the entity profile schema [33]. In this sense, the challenges inherent in heterogeneous data are addressed by schema-agnostic blocking techniques, which ignore scheme-related information and consider the attribute values of each entity [5].

Among the existing schema-agnostic blocking techniques, metablocking has emerged as one of the most promising regarding efficiency and effectiveness [33]. Metablocking aims to identify the closest pairs of entity profiles by restructuring a given set of blocks into a new one that involves significantly fewer comparisons. To this end, this technique's blocks form a weighted graph and pruning criteria are applied to remove edges with weights below a threshold, aiming to discard comparisons between entities with few chances of being considered a match. Originally, the authors in [21] named the pruning step metablocking, which receives a redundancy-positive set of blocks and generates a new set of pruned blocks. However, to avoid misunderstanding, in this work, we call metablocking the blocking technique as a whole and *pruning* the step responsible for pruning the blocking graph. Recently, some works [16,22] have proposed parallel-based blocking techniques to increase the efficiency of metablocking. However, these state-of-the-art techniques (either standalone or parallel-based) do not work properly in scenarios involving incremental and streaming data since they were developed to work with batch data [6].

3.2. Blocking in the Noisy-Data Context

To address the problem of noisy data, three strategies are commonly applied: n-gram algorithms, Natural Language Processing (NLP), and Locality-Sensitive Hashing (LSH) [36,37]. In the context of blocking techniques based on tokens, the application of *n-gram* strategies negatively affects efficiency since n-gram algorithms increase the number of tokens and consequently the number of blocks managed by the blocking technique. Regarding NLP applications, word vectors and dictionaries can be applied in the sense of fixing misspelled words and recovering the correct tokens. However, NLP also negatively affects the efficiency of the blocking task as a whole since it is necessary to consult the word vector for each token. Thus, among the possible strategies to handle noisy data, LSH is the most promising in terms of results [20,32,38].

Recently, the BLAST technique [20] has been applied to LSH to determine the linkages between the attributes of two large data sources in order to address efficiency issues. However, the BLAST technique does not explore noisy data as a contribution but introduces the application of LSH in blocking techniques as a factor for improving effectiveness. More specifically, this technique reduces the dimensionality of data through the application of LSH and guides the blocking task to enhance the effectiveness results. Following the BLAST idea, the work in [32] applies LSH in order to hash the attribute values and enable the generation of high-quality blocks (i.e., blocks that contain a significant number of entities with high chances of being considered similar/matches), even with the presence of noise in the attribute values. In [38], the Locality-Sensitive Blocking (LSB) strategy is proposed. LSB applies LSH to standard blocking techniques in order to group similar entities without requiring the selection of blocking keys. To this end, LSH works in the sense of generating hash values and guiding the blocking, which increases the robustness toward blocking parameters and data noise.

In contrast to the previously mentioned works, our work not only focuses on noisy data but also allows the proposed incremental blocking technique to handle streaming and noisy data simultaneously. Although the works in [20,32,38] do not explore aspects such as incremental processing or streaming data, the idea behind the application of LSH to minimize the negative effects of noisy data on the blocking techniques can also be applied to the proposed technique to expand its applications. Therefore, this work adapts the application of LSH (in blocking techniques) to the contexts of distributed computing, incremental processing, and streaming data.

3.3. Blocking Benefiting from Schema Information

Schema information can benefit matching tasks by enhancing efficiency without compromising effectiveness. Works such as [39,40] suggest the application of strategies able to identify the functional dependencies (FDs) among the attributes of a schema to support data quality and data integration tasks. FDs are constraints that determine the relationship between one attribute and another in a schema. Therefore, these relations can guide matching tasks in order to compare values considering only the values from similar attributes, which are determined by the FDs. Regarding ER, works such as [41,42] propose extracting information from relational schemas (i.e., rigid schemas) based on the matching dependencies, which are extensions of FDs. In this sense, the idea is to select matching dependencies as rules that determine the functional dependencies among attributes, where attribute values do not need to be exactly equal but similar. Thus, the idea behind matching dependencies is to efficiently select the best subset of rules (i.e., dependencies among attributes) to support the ER task. However, this strategy does not consider data heterogeneity, which does not follow a rigid schema, or the streaming context, which demands processing entities following a time budget. Particularly, the approach proposed in [42] can take several minutes to complete the process, even for data sources with a few thousand entities.

For contexts involving blocking and heterogeneous data, we can identify the works in [11,20,31,32], which propose blocking techniques that benefit from information related to entity attributes. In the attribute-based blocking in [31], before the blocking step, there is an initial grouping of attributes that is generated based on the similarities of their values. External sources such as a thesaurus can be consulted to determine similar attributes. Similarly, the idea behind [11,20,32] is to exploit the loose schema information (i.e., statistics collected directly from the data) to enhance the quality of blocks. Then, rather than looking for a common token regardless of the attribute to which it belongs, entity descriptions are compared only based on the values of similar attributes. Hence, comparisons of descriptions that do not share a common token in a similar attribute are discarded. To summarize, the main idea is to exploit the schema information of the entity descriptions in order to minimize the number of false matches.

On the other hand, the attribute selection algorithm (proposed in the present work) before the token blocking step identifies and discards superfluous attributes, which will provide useless tokens that unnecessarily consume memory. Note that attribute selection aims to minimize the number of false-positive matches and improve resource consumption. Therefore, even though the related works extract information from the attributes, they do not use this information to discard superfluous attributes; the idea is to perform a kind of attribute clustering to guide block generation. Furthermore, the stated works deal with neither streaming nor incremental data, whose challenges are addressed in our work.

3.4. Incremental Blocking

In terms of incremental blocking techniques for relational data sources, the authors of [9,43–45] propose approaches that are capable of blocking entities incrementally. These works propose an incremental workflow for the blocking task, considering the evolutionary behavior of data sources to perform the blocking. The main idea is to avoid (re)processing the entire dataset during the incremental ER to update the deduplication results. For doing so, different classification techniques can be employed to identify duplicate entities. Therefore, these works propose new metrics for incremental record linkage using collective classification and new heuristics (which combine clustering, coverage component filters, and a greedy approach) to further speed up incremental record linkage. However, the stated works do not deal with heterogeneous and streaming data.

In terms of other incremental tasks that propose useful strategies for ER, we can identify the works in [14,46,47], which present parallel and incremental models to address the tasks of name disambiguation, event processing, and dynamic graph processing, respectively. More specifically, in the ER context, the work in [48] proposes an incremental approach to perform ER on social media data sources. Although such sources commonly provide heterogeneous data, the work in [48] generates an intermediate schema so that the extracted data from the sources follow this schema. Therefore, although the data are originally semi-structured, they are converted to a structured format before being sent to the ER task. For this reason, the approach in [48] differs from our technique since it does not consider the heterogeneous data challenges and does not apply/propose blocking techniques to support ER.

3.5. Blocking in the Streaming Data context

Streaming data commonly add several challenges, not only in the context of ER. The works in [49,50], which address clustering and itemset frequencies, highlight the necessity of developing strategies to continuously process data sent in short intervals of time. Even if the works in [49,50] do not consider other scenarios, such as heterogeneous data and incremental processes, both suggest the need to develop an appropriate architecture to deal with streaming data, which is discussed in Section 4.

Regarding ER approaches that deal with streaming data, we can highlight the works in [2,10,51]. These works propose workflows to receive data from streaming sources and perform the ER task. In these workflows, the authors suggest the application of time windows to discard old data and, consequently, avoid the growing consumption of resources (e.g., memory and CPU). On the other hand, it is important to highlight that these works apply time windows only in the sense of determining the entities to be processed together. Hence, they do not consider incremental processing and, therefore, discard the previously processed data. Thus, none of them deal simultaneously with the three challenges (i.e., heterogeneous data, streaming data, and incremental processing) addressed by our work.

In the ER context, we can highlight only the recent work proposed in [19], which addresses challenges related to streaming data and incremental processing. The authors of [19] propose a Spark-based blocking technique to handle heterogeneous streaming data. As previously stated, the present research is an evolution of the work in [19]. Overall, it is possible to highlight the following improvements: (i) an efficient workflow able to address the memory consumption problems present in [19], which decrease the technique's efficiency, as well as its ability, to process large amounts of data; (ii) an attribute selection algorithm, which discards superfluous attributes to enhance efficiency and minimize memory consumption; (iii) a top-n neighborhood strategy, which maintains only the "n" most similar neighbor entities of each entity; (iv) a noise-tolerant algorithm, which allows the proposed technique to generate high-quality blocks, even in the presence of noisy data; and (v) a parallel architecture for blocking streaming data, which divides all the blocking processes among two components (sender and blocking task) to enhance efficiency.

Based on the related works cited in this section (summarized in Table 3) and information provided by several other works [5,52,53], it is possible to identify a lack of works in different areas that address the challenges related to streaming data and incremental processing efficiently. The same applies to ER approaches to heterogeneous data [19]. In this sense, our work addresses an open research area, and it can be a useful schema-agnostic blocking technique for supporting ER approaches in scenarios involving not only streaming data and incremental processing but also noisy data.

Table 3. Related work comparison. Note, "X" means that the works address the research topic, "-" otherwise.

Work (s)	ER	Heterogeneous	Noisy Data	Streaming	Parallel	Incremental
[14,46]	-	-	-	-	X	X
[47]	-	-	-	X	X	X
[41,42]	X	-	-	-	-	-
[9,43]	X	-	-	-	-	X
[44,45]	X	-	-	-	X	X
[8,20,31–34]	X	X	-	-	-	-
[38]	X	-	X	-	-	-
[20,32]	X	X	X	-	-	-
[11,16,22,35]	X	X	-	-	X	-
[2,10,48,51]	X	-	-	X	X	-
Proposed work	X	X	X	X	X	X

4. Incremental Blocking over Streaming Data

Figure 1 illustrates the traditional steps involved in an ER process. However, it is necessary to redesign this workflow to adapt it to the needs of the context addressed in this work. The focus of this work is the blocking step; therefore, information about the data processing step is limited throughout this section. When streaming data are considered, the traditional ER workflow should include a pre-step responsible for organizing the micro-batches to be processed. For this reason, the sender component was inserted into the architecture to block streaming data, as depicted in Figure 2. Note that since streaming data are constantly sent, the ER steps should be performed according to a time budget (represented by τ in this work, as stated in Section 2). This behavior creates a challenge for the workflow, which needs to be performed as fast as possible. To address this challenge, the top-n neighborhood and attribute selection strategies were proposed to enhance blocking efficiency without having a negative impact on effectiveness.

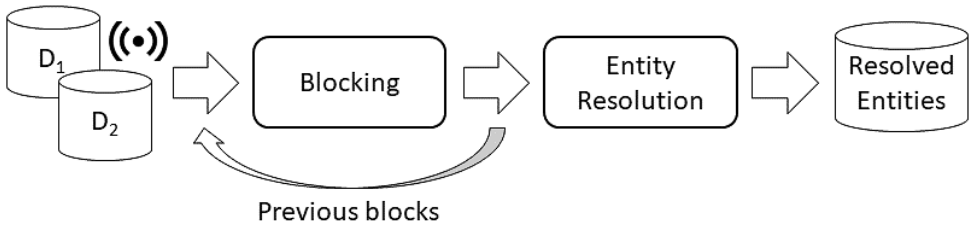


Figure 1. ER workflow considering the streaming data and incremental context.

Regarding incremental blocking, the idea is to consider the coming streaming data and generate incremental blocks during a time window. Therefore, the traditional ER workflow should be updated to receive as input not only the data from the sources but also the blocks previously generated in the previous increments. For the first increment, the received entities are blocked similarly to the traditional ER workflow. However, from the second increment, the received entities are blocked, merging/updating with the blocks generated previously. To handle this behavior, we propose an incremental architecture able to store the generated blocks and update them as the new entities arrive.

Since the data can present noise, which commonly minimizes blocking effectiveness, we also propose a noise-tolerant algorithm to be applied to the ER workflow. The noise-tolerant algorithm benefits from the idea behind LSH to generate high-quality blocks, even in the presence of noisy data. In this sense, the sender component (see Figure 2) applies this algorithm to the data in order to generate hash values instead of tokens. Thus, the blocking step generates blocks using hash values as the blocking keys, as explained later in this section.

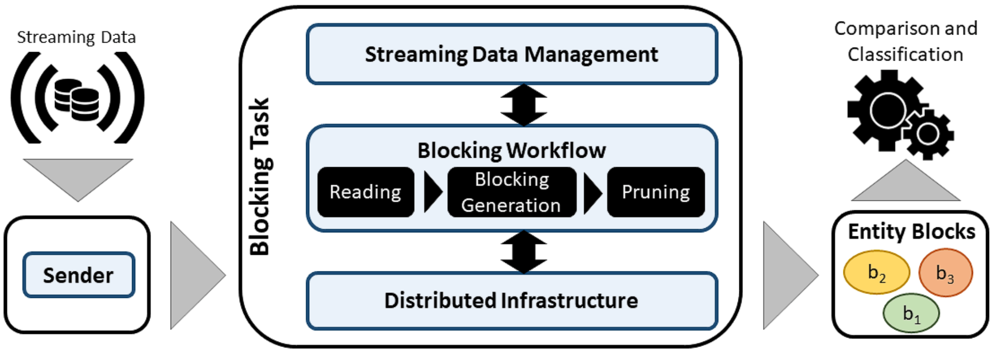


Figure 2. Parallel architecture for blocking streaming data.

Throughout this section, we propose a parallel-based blocking technique able to incrementally process streaming data. Furthermore, we describe a parallel architecture, which hosts the components necessary to perform the blocking of streaming data, as well as clarify how the proposed technique is coupled to the architecture.

4.1. *Parallel Architecture for Blocking Streaming Data*

The architecture is divided into two components: the sender and blocking task, as depicted in Figure 2. The sender component, which is executed in standalone mode, consumes the data provided by the data sources in a streaming way, buffers it in micro-batches, and sends it to the blocking task component. Note that buffering streaming data in micro-batches is a common strategy to process streaming data [47,54,55], even in critical scenarios where data arrive continuously. Thus, it is possible to follow the τ time interval and respect the time restriction for streaming blocking, as defined in Section 2. In the blocking task component, blocking is performed over a parallel infrastructure. To connect both components, a streaming processing platform, namely Apache Kafka, is applied.

The blocking task component is divided into three layers: streaming data management, the blocking workflow, and the distributed infrastructure. The first layer is responsible for continuously receiving and manipulating the streaming data. Thus, the data provided by the sender are collected and sent to the blocking workflow layer where blocking is performed. Therefore, the data (i.e., entities) received in the first layer are processed (i.e., blocked) in the blocking layer. In this layer, the proposed blocking technique is coupled to the architecture. Note that different parallel-based blocking techniques can be applied to the blocking workflow layer. The blocking layer is directly connected to the distributed infrastructure, which provides all the distributed resources (such as the MapReduce engines and virtual machines) needed to execute the proposed streaming blocking technique. Finally, after blocking, the generated blocks are sent to the next steps in the ER task (i.e., comparison and classification of entity pairs). Note that the scope of this paper is limited to the blocking step.

When **noisy data** are considered, the noise-tolerant algorithm is applied. The algorithm is hosted by the sender component. Thus, as the data arrive in a streaming way, the noise-tolerant algorithm is applied to convert the tokens present in each entity (from its attribute values) into hash values. To this end, the noise-tolerant algorithm applies Locality-Sensitive Hashing (LSH) to the tokens from the entities in order to avoid the issues generated by the noisy data [36]. In general, LSH is used for approximating the near-neighbor search in high-dimensional spaces [56]. It can be applied to reduce the dimensionality of a high-dimensional space, preserving the similarity distances of the tokens to be evaluated. Thus, for each attribute value, a hash function (e.g., MinHash [56]) converts the attribute value into a probability vector called a signature (Minhash signature). Then, the arriving entities are buffed in micro-batches and sent to the blocking task component. Note that the execution of the noise-tolerant algorithm occurs in the order of milliseconds per entity, which does not negatively impact the efficiency results of blocking as a whole, as addressed in Sections 7 and 8.

Since the hash function preserves the similarity of the attribute values, it is possible to take advantage of this behavior and obtain a similar hash signature for similar attribute values, even in the presence of noise [56]. Then, the hash function can generate similar signatures that will be used to guide the blocking task. To clarify how the noise-tolerant algorithm works over the proposed technique, consider the following running example. Given two entities from two different datasets $e_1 = \{\langle name, Linus\ Torvalds\rangle\}$ and $e_2 = \{\langle name, Lynus\ Tordalds\rangle\}$, where e_1 and e_2 represent the same entities in the real world but there are typos in e_2. Without the application of the noise-tolerant algorithm, the tokens considered to block the entities will be "*Linus*" and "*Torvalds*" from e_1, and, "*Lynus*" and "*Torvalds*" from e_2. Note that the token-based blocking will not group the entities e_1 and e_2 at the same block since they do not share common tokens and, consequently, they will not be considered a match. On the other hand, with the application of the noise-tolerant algorithm, the attribute values of e_1 and e_2 will be converted into a probability vector with the same size as their token sets. Then, e_1 could be converted to $e_1 = \{\langle name, 8709973\ 6431654\rangle\}$ and e_2 could be converted to $e_2 = \{\langle name, 8709973\ 7338779\rangle\}$. After the execution, the entities e_1 and e_2 (with their hash signatures) are sent to the blocking task component. Since the hash function preserves the similarity of the attribute values, it is important to highlight that it is common to obtain the same hash value, even in the presence of noise on the data (as occurs with the tokens "*Linus*" and "*Lynus*"). In this sense, instead of considering tokens during the blocking task, the noise-tolerant algorithm allows the proposed technique to take into account the hash signatures and applies them to the block. Therefore, entities that share a common hash value (in the hash signatures) should be grouped at the same block. For this reason, e_1 and e_2 will be grouped at the same block due to the hash value "8709973".

4.2. Incremental Blocking Technique for Streaming Data

To address the challenges previously stated, the proposed blocking technique is based on a MapReduce workflow. The workflow is composed of two MapReduce jobs (Figure 3), i.e., one fewer job than the method available in [22] (further details can be found in the token extraction step below). In addition, the proposed workflow does not negatively affect the effectiveness results since the block generation is not modified. The workflow is divided into three steps as depicted in Figure 3: token extraction, blocking generation, and pruning. It is important to highlight that for each increment, the three blocking steps are performed following the workflow depicted in Figure 3.

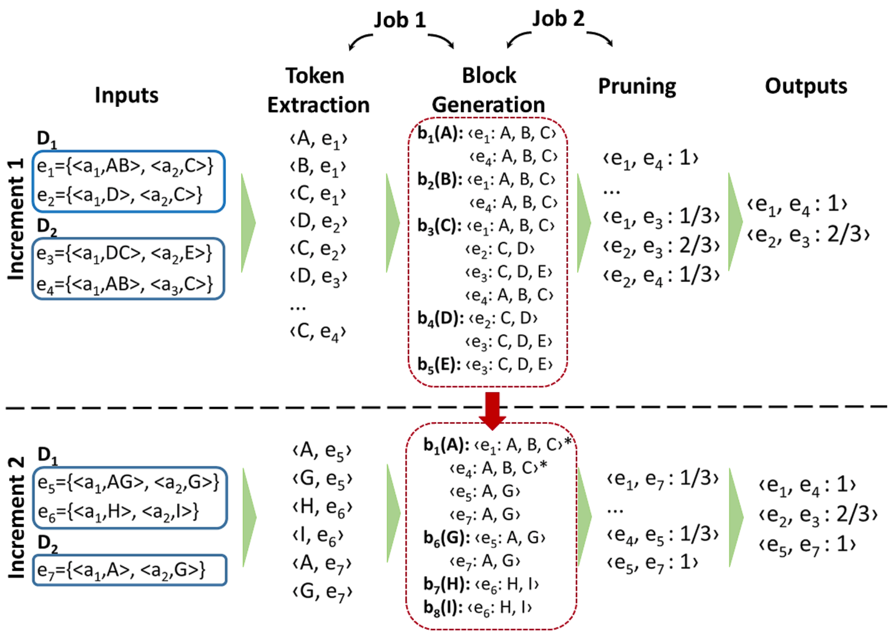


Figure 3. Workflow for the streaming blocking technique.

4.2.1. Token Extraction Step

This step is responsible for extracting the tokens from the attribute values (of the input entities), where each token will be used as a blocking key, as illustrated in Algorithm 1. Technically, the token step can be considered a map function through the application of the Flink function called *flatmap*. Thus, each entity will generate a set of key–value pairs where the key is represented by the token (blocking key) and the value is the entity in question.

Algorithm 1: Token Extraction

```
Input: $E_{iD_1}$ and $E_{iD_2}$: increments provided by input data sources
Output: $B$: blocks based on tokens
1  $B \leftarrow \emptyset$;
2  $tokenBlocking(E_{iD_1})$;
3  $tokenBlocking(E_{iD_2})$;
4  return $B$
5  Function $tokenBlocking(E_{iD})$ do
6      foreach $e$ in $E_{iD}$ do
7          foreach $t$ in $\Lambda_e$ do
8              $B.put(t, \langle e, \Lambda_e, D \rangle)$;
9          end
10     end
11 end
```

For each increment i, the blocking receives a pair of entity collections E_{iD_1} and E_{iD_2} provided by data sources D_1 and D_2 (lines 2 and 3). For each entity $e \in E_{iD_1} \cup E_{iD_2}$ (lines 6 to 10), all tokens Λ_e associated with e are extracted and stored. Each token in Λ_e will be a blocking key that determines a specific block $b \in B$, and the set Λ_e will be applied to determine the similarity between the entities in the next step. Similarly, when noisy data are present, Λ_e is replaced with Λ_{e_h}. Note that, as stated, $|\Lambda_e| = |\Lambda_{e_h}|$. Then, each hash value in Λ_{e_h} will be a blocking key that determines a specific block $b \in B$, and the set Λ_{e_h} will be applied to determine the similarity between the entities in the next step. From now on, to facilitate the understanding, we will consider only Λ_e. Therefore, note that the hash values in Λ_{e_h} work in a similar way to the tokens in Λ_e.

It is important to highlight that from this step, every entity e contains information regarding its tokens (blocking keys) Λ_e and the data source D that it comes from (lines 7 and 9). From the Λ_e stored in each entity during the token extraction step, it is possible to avoid one MapReduce job (compared with the workflow proposed in [22]). In [22], the workflow applies an extra job to process the entities in each block and determine all the blocks that contain each entity. In other words, an extra job is used to compose the Λ_e. On the other hand, our blocking technique determines Λ_e in this step and spreads this information to the next steps, avoiding the necessity of another MapReduce job. Note that although the work in [22] presents various strategies for parallelizing each pruning algorithm of the metablocking, the avoided MapReduce job is related to the token extraction step (i.e., before the pruning step). Thus, the proposed workflow tends to enhance the efficiency of blocking as a whole.

The time complexity of the token extraction step is directly related to the generation of the blocking keys. This step evaluates all tokens in Λ_e of each entity $e \in E_i$ (i.e., $E_i = E_{iD_1} \cup E_{iD_2}$) to generate the blocking keys. Therefore, the time complexity of this step is $\mathcal{O}(||\Lambda_{E_{iD_1}}|| + ||\Lambda_{E_{iD_2}}||)$, where $||\Lambda_{E_{iD_1}}||$ is given by $\sum_{e \in E_{iD_1}} |\Lambda_e|$ and $||\Lambda_{E_{iD_2}}||$ is given by $\sum_{e \in E_{iD_2}} |\Lambda_e|$. We guarantee that the produced blocking keys are the same as the ones produced by metablocking using the following lemma.

Lemma 1. *Considering the same entity collection $E_i = E_{iD_1} \cup E_{iD_2}$ as input, if the proposed technique and metablocking apply the same function $\Gamma(v)$ to extract tokens from attribute values, then both techniques will produce the same blocking keys Λ_e.*

Proof. In the token extraction step, the proposed technique receives as input the entity collection E_i. During this step, the entities provided by E_i are processed to extract tokens, applying a function Γ. For each entity $e \in E_i$, a set of blocking keys Λ_e is produced from the application of $\Gamma(v)$ to the attribute values v of e. If metablocking receives the same E_i and applies the same $\Gamma(v)$ to the attribute values v of $e \in E_i$, the technique will produce the same tokens and, consequently, the same blocking keys Λ_e, even if the parallel workflows differ. □

In Figure 3, there are two different increments (i.e., Increment 1 and Increment 2) containing the entities that will be processed by the blocking technique. For the first increment (top of the figure), D_1 provides entities e_1 and e_2, and D_2 provides entities e_3 and e_4. From entity e_1, the tokens A, B, and C are extracted. To clarify our example, e_1 can be represented as $e_1 = \{\langle name, Linus\ Torvalds\rangle\ \langle creator, Linux\rangle\}$. Thus, the tokens A, B, and C represent the attribute values "*Linus*", "*Torvalds*", and "*Linux*", respectively. Since the generated tokens work as blocking keys, the entities are arranged in the format $\langle k, e\rangle$ such that k represents a specific blocking key and e represents an entity linked to the key k.

4.2.2. Blocking Generation Step

In this step, the blocks are created and a weighted graph is generated from the blocks in order to define the level of similarity between entities, as described in Algorithm 2. From the key–value pairs generated at the token extraction step, the values (i.e., entities) are grouped by key (i.e., blocking key) to generate the entity blocks. The entity blocks are generated through a *groupby* function, provided by Flink. Note that the first MapReduce job, which started in the token extraction step, concludes with the *groupby* function. Moreover, during the blocking generation step, another MapReduce job is started. To compute the similarity between the entities, a *flatmap* function generates pairs in the format $\langle e_{D_1}, \langle e_{D_2}, \rho\rangle\rangle$, which will be processed in the pruning step.

Algorithm 2: Blocking Generation

```
Input: $B$: blocks generated in the token extraction step
Output: $G$: blocking graph
1  $G \leftarrow \emptyset$;
2  foreach $k$ in $B.keys$ do
3      $entities \leftarrow B.get(t)$;
4      while $entities$ is not $\emptyset$ do
5          $e_1 \leftarrow entities.pop()$;
6          foreach $e_2$ in $entities$ do
7              if $e_1.D$ is not $e_2.D$ then
8                  $\rho \leftarrow \Phi(e_1.\Lambda_{e_1}, e_2.\Lambda_{e_2})$;
9                  $G.put(e_1, \langle e_2, \rho \rangle)$;
10                 $G.put(e_2, \langle e_1, \rho \rangle)$;
11             end
12         end
13     end
14 end
15 return $G$
16 Function $\Phi(\Lambda_{e_1}, \Lambda_{e_2})$ do
17     $\rho \leftarrow \frac{|\Lambda_{e_1} \cap \Lambda_{e_2}|}{max(|\Lambda_{e_1}|, |\Lambda_{e_2}|)}$;
18     return $\rho$
19 end
```

Initially, the entities are arranged in the format $\langle e, \Lambda_e \rangle$; implicitly, Λ_e denotes the blocks that contain entity e. In scenarios involving noisy data, the entities assume the format $\langle e, \Lambda_{e_h} \rangle$, where Λ_{e_h} also represents the blocks that contain entity e. Based on the blocking keys (i.e., tokens or hash signatures), all entities sharing the same key are grouped in the same block. Then, a set of blocks B is generated so that a block $b \in B$ is linked with a key k and all $\langle e, \Lambda_e \rangle \in b$ share (at least) the key k (lines 2 to 14). For instance, in Figure 3, b_1 is related to the key A and contains entities e_1 and e_4 since both entities share the key A. The set of blocks B generated in this step is stored in memory to be used for the next increments. In this sense, new blocks will be included or merged with the blocks previously stored.

Afterward, entities stored in the same block are compared (lines 3 to 13). Thus, entities provided by different data sources (line 7) are compared to define the similarity ρ between them (line 8). The similarity is defined based on the number of co-occurring blocks between the entities. Note that co-occurring blocks are a well-known strategy for defining similarity between entities in scenarios involving token-based blocking. The works in [21,57,58] formalize and describe the effectiveness of defining entity similarity based on the number and frequency of common blocks. After defining the similarity between the entities, the entity pairs are inserted into the graph G (lines 9 and 10) such that the weight of an edge linking two entities is given by the similarity ρ (computed in lines 16 to 18) between them. The blocking generation step is the most computationally costly in the workflow since the comparison of the entities is performed in this step. The time complexity of this step is given by the sum of the cost to compare the entity pairs in each block $b \in B$. Therefore, the time complexity of the blocking generation step is $\mathcal{O}(||B||)$ such that $||B||$ is given by $\sum_{b \in B} ||b||$ and $||b||$ is the number of comparisons to be performed in b. Furthermore, to guarantee that the graphs generated by our technique and metablocking are the same, the following lemma states the conditions formally.

Lemma 2. *Assuming that the proposed technique and metablocking apply the same similarity function Φ to the same set of blocks B (derived from the blocking keys Λ_e), both techniques will produce the same set of vertices X and edges L. Therefore, since the blocking graph G is composed of vertices (i.e., X) and edges (i.e., L), the techniques will generate the same graph G.*

Proof. During the blocking generation step, the set of blocks B (based on the blocking keys Λ_e) is given as input. The goal of this step is to generate the graph G(X, L). To this end, each entity $e \in b$ (such that $b \in B$) is mapped to a vertex $x \in X$. Thus, if the metablocking technique receives the same set of blocks B, the technique will generate the same set of vertices X. To generate the edges $l \in L$ (represented by $\langle x_1, x_2, \rho \rangle$) between the vertices, the similarity function $\Phi(x_1, x_2)$ (such as x_1 and $x_2 \in X$) is applied to determine the similarity value ρ. Therefore, assuming that the metablocking technique applies the same Φ to the vertices of X, the set of edges L produced by the proposed technique will be the same. Since both techniques produce the same X and L sets, the generated blocking graph G will be identical, even though they apply different workflows. □

In Figure 3, b_1 contains entities e_1 and e_4. Therefore, these entities must be compared to determine their similarity. Their similarity is one 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), b_1 receives entities e_5 and e_7. Thus, in the second increment, 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 with each other to determine their similarity. However, since e_1 and e_4 were already compared in the first increment, they should not be compared twice. This would be considered an unnecessary comparison.

Technically, the strategies used to avoid redundant comparisons are conditional restrictions implemented at the *flatmap* function of the blocking generation step. During incremental blocking, three types of unnecessary comparisons should be avoided. First, due to the block overlapping, an entity pair can be compared in several blocks (i.e., more than once). Since this type of comparison is commonly related to metablocking-based techniques, we apply the Marked Common Block Index (MaCoBI) [16,22] condition to avoid this type of unnecessary comparison. The MaCoBI condition aims to guarantee that an entity pair will be compared in only one block. To this end, it uses the intersection of blocking keys (stored in both entities) to select the block in which this entity pair should be compared. Thus, based on the intersection of blocking keys, the comparison is performed only in the first block in which both entities co-occur, preventing the comparison from being performed more than once (on the other blocks).

The second type of comparison is related to incremental blocking. For a specific increment, it is possible that some blocks may not suffer updates. This occurs due to the fact that the entities provided by this specific increment may not be related to any of the preexisting blocks and, consequently, do not update them. Therefore, the entities contained in these blocks that did not suffer updates must not be compared again. To solve this issue, the proposed workflow applies an update-oriented structure to store the blocks previously generated. This structure, commonly provided and managed by MapReduce frameworks (e.g., Flink (https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/stream/state/state.html, accessed on 28 September 2022)), loads only blocks that have been updated for the current increment. Note that the blocks that did not suffer updates are not removed; they just are not loaded into memory for the current increment. The application of update-oriented structures assists the technique as a whole to improve the consumption of computational resources. Moreover, the application of an update-oriented structure (e.g., stateful data streams (https://flink.apache.org/stateful-functions.html, accessed on 28 September 2022) provided by Flink) allows the technique to maintain all the blocks during a time window. Thus, it is possible to add new blocks and update the pre-existing blocks (i.e., add entities from the current increment) over the structure, which satisfies the incremental behavior of the proposed blocking technique. In Section 5, we discuss the time-window strategy, which discards entities from the blocks based on their arrival time.

The third type of comparison is also related to incremental blocking. Since the blocking technique can receive a high (or infinite) number of increments and the blocks can still exist for a huge number of increments, it is necessary to take into account the comparisons that have already been performed in previous iterations (increments). Note that even updated blocks contain entity pairs that have been compared in previous iterations, which should not be compared again. For this reason, our blocking technique marks the entity pairs that have already been compared. This mark works like a flag to decide if the entity pair was compared in previous iterations. Thus, an entity pair must only be compared if at least one of the entities is not marked as already compared.

To illustrate how our technique deals with the three types of unnecessary comparisons, we consider the second iteration (bottom of Figure 3). The entity pair $\langle e_1, e_4 \rangle$ should be compared in blocks b_1, b_2, and b_3, which represents the first type of unnecessary comparison. However, the MaCoBi condition is applied to guarantee that $\langle e_1, e_4 \rangle$ is compared only once. To this end, the MaCoBi condition evaluated the intersection of blocks (i.e., b_1, b_2, or b_3) between both entities and determined that the comparison should be performed only in the first block of the intersection list (i.e., b_1). Considering the second type of unnecessary comparison, the blocks b_1, b_2, b_3, b_4, and b_5 are considered pre-existing blocks since they were generated in the first iteration. In the second iteration, only block b_1 is updated. In addition, blocks b_6, b_7, and b_8 are created. To avoid the blocks that did not suffer updates (i.e., b_2, b_3, b_4, and b_5) from being considered in the second iteration, the update-oriented structure only takes into account blocks b_1, b_6, b_7, and b_8. Therefore, this strategy avoids a huge number of blocks from being loaded into memory unnecessarily. To avoid the third type of unnecessary comparison, during the first iteration (top of Figure 3), entities e_1 and e_4 are marked (with the symbol *) at block b_1 to indicate that they have already been compared. Therefore, since an entity pair is only compared if at least one of the entities is not marked, the pair $\langle e_1, e_4 \rangle$ will not be compared again in the second iteration.

Top-*n* neighborhood strategy. Incremental approaches commonly suffer from resource issues [9,10], for instance, the work in [19] highlights problems related to memory consumption such as a lack of memory to process large data sources. To avoid excessive memory consumption, we propose a top-n neighborhood strategy to be applied during the creation of the graph G (i.e., in the blocking generation step). The main idea of this strategy is to create a directed graph, where each node (i.e., entity) maintains only the n most similar nodes (neighbor entities). To this end, each node will store only a sorted list of its neighbor entities (in descending order of similarity).

Note that the number of entities and connections between the entities in G is directly related to the resource consumption (e.g., processing and memory) since G needs to be processed and stored (in memory) during the blocking task. Considering incremental processing, this resource consumption needs to be treated since G also needs to maintain the entities according to incremental behavior (i.e., in an accumulative way). For this reason, a limitation on the number of entity neighbors is useful in the sense that it helps to reduce the number of connections (i.e., edges) between the entities. Furthermore, it is necessary to remember that entities (nodes) without any connections (edges) should be discarded from G (following the metablocking restrictions). Therefore, the application of the top-n neighborhood strategy saves memory and processing by minimizing the number of entities and connections in G. On the other hand, it decreases the effectiveness of the blocking technique since truly similar entities (in the neighborhood) can be discarded. However, as discussed later in Section 7, it is possible to increase efficiency by applying the top-n neighborhood strategy without significant decreases in effectiveness.

For instance, consider the first increment in Figure 3 e_1 has $\{\langle e_4, 1 \rangle, \langle e_2, 1/3 \rangle, \langle e_3, 1/3 \rangle\}$ as the set of neighbor entities. The neighbor entities of e_1 are sorted in descending order of similarity ρ. Thus, for each increment, the top-n neighbor entities of e_1 can be updated according to the order of similarity. To maintain only the top-n neighbor entities, the less similar neighbor entities are removed. If we hypothetically apply a top-1 neighborhood, entities e_2 and e_3 will be removed from the set of neighbor entities. Considering large graphs generated from large data sources or as a result of incremental processing, reducing the information stored in each node will result in a significant decrease in memory consumption as a whole. Therefore, the application of the top-n neighborhood strategy can optimize the memory consumption of our blocking technique, which enables the processing of large data sources. In Section 7, we discuss the gains in terms of resource savings of the impact on the effectiveness results when different values of n are applied.

4.2.3. Pruning Step

The pruning step is responsible for discarding entity pairs with low similarity values, as described in Algorithm 3. To generate the output (i.e., pruned blocks), the pruning step is composed of a *groupby* function, which receives the pairs $\langle e_{D_1}, \langle e_{D_2}, \rho\rangle\rangle$ from the blocking generation step and groups the entities by e_{D_1}. Thus, the neighborhood of e_{D_1} is defined and the pruning is performed. Note that the second MapReduce job, which started in the blocking generation step, concludes with the *groupby* function, whose outputs are the pruned blocks.

Algorithm 3: Pruning

```
   Input: $G$: graph generated in block generation step
   Output: $B'$: pruned block
1  $B' \leftarrow \emptyset$;
2  $B' \leftarrow WNP(G)$;
3  return $B'$
4  Function WNP(G) do
5      foreach $e$ in $G.keys$ do
6          $neighbors \leftarrow G.get(e)$;
7          $sum \leftarrow 0$;
8          foreach $pair$ in $neighbors$ do
9              $sum \leftarrow sum + pair.\rho$;
10         end
11         $\theta \leftarrow \frac{sum}{|neighbors|}$;
12         foreach $pair$ in $neighbors$ do
13             if $pair.\rho \geq \theta$ then
14                 $B'.put(e, pair)$;
15             end
16         end
17     end
18 end
```

After generating the graph G, a pruning criterion is applied to generate the set of high-quality blocks B' (lines 1 to 3). As a pruning criterion, we apply the WNP-based pruning algorithm [33] since it has achieved better results than its competitors [22,33]. The WNP is a vertex-centric pruning algorithm, which evaluates each node (of G) locally (lines 5 to 17), considering a local weight threshold (based on the average edge weight of the neighborhood—lines 6 to 11). Therefore, the neighbor entities whose edge weights are greater than the local threshold are inserted into B' (lines 12 to 16). Otherwise, the entities (i.e., whose edge weights are lower than the local threshold) are discarded. Note that the application of the top-n neighborhood strategy (in the previous step) does not make the pruning step useless. On the contrary, the top-n neighborhood can provide a refined graph in terms of the high-quality entity pairs and smaller neighborhood to be pruned in this step.

The time complexity of this step is given by the cost of the WNP pruning algorithm, which is $\mathcal{O}(|X| \cdot |L|)$ [33], where $|X|$ is the number of vertices and $|L|$ is the number of edges in the graph G. Therefore, the time complexity of the pruning step is $\mathcal{O}(|X| \cdot |L|)$. Finally, the following lemma and theorem state the conditions to guarantee that the proposed technique and metablocking produce the same output (i.e., B') and, therefore, present the same effectiveness results.

Lemma 3. *Considering that the proposed technique and metablocking apply the same pruning criterion Θ for the same input blocking graph G, both techniques will produce the same pruned graph G'. Since the output blocks B' are directly derived from G', both techniques will produce the same output blocks B'.*

Proof. In the pruning step, the proposed technique receives as input the blocking graph $G(X, L)$. This step aims to generate the pruned graph G' according to a pruning criterion $\Theta(G)$. To this end, the edges $l \in L$ (represented by $\langle x_1, x_2, \rho \rangle$), whose value of $\rho < \theta$ are removed, and the vertices $x \in X$ that have no associated edges are discarded. Note that θ is given by a pruning criterion $\Theta(G)$. Therefore, if the metablocking technique receives the same blocking graph $G(X, L)$ as input and applies the same pruning criterion $\Theta(G)$, both techniques will generate the same pruned graph G' and, consequently, the same output blocks B'. □

Theorem 1. *For the same entity collection $E_i = E_{iD_1} \cup E_{iD_2}$, if the proposed technique and metablocking apply the same function $\Gamma(v)$ to extract tokens, the same similarity function $\Phi(x_1, x_2)$, and the same pruning criterion $\Theta(G)$, both techniques will generate the same output blocks B'.*

Proof. Based on Lemma 1, with the same E_i and $\Gamma(v)$, the proposed technique and metablocking will generate the same blocking keys Λ_e and, consequently, the same set of blocks B. Based on Lemma 2, if both techniques receive the same set of blocks B and apply the same $\Phi(x_1, x_2)$, they will generate the same blocking graph G. Based on Lemma 3, when the proposed technique and metablocking prune the same graph G using the same $\Theta(G)$, both techniques generate the same pruned graph G' and, consequently, the same output blocks B'. □

5. Window-Based Incremental Blocking

In scenarios involving streaming data, data sources provide an infinite amount of data continuously. In addition, incremental processing consumes information processed in previous increments. Therefore, the computational resources of a distributed infrastructure may not be enough for the accumulative sum of entities to be processed in each increment. Considering these scenarios, memory consumption is stated as being one of the biggest challenges faced by incremental blocking techniques [14,19,43]. Time windows are at the heart of processing infinite streams since they split the stream into buckets of finite size, reducing the memory consumption (since entities that exceeded the time threshold are discarded) of the proposed technique [59]. For this reason, a time window is used in the blocking generation step of our technique, where blocks are incrementally stored in the update-oriented data structure (see Section 4.2). In addition to being one of the most commonly used strategies in incremental approaches [9,14], time windows emerged as a viable solution for the experiments and case study developed in this work since time is a reasonable criterion for discarding.

For instance, in a scenario involving incremental and streaming data, the entities arrive continually over time (e.g., over hours or days). Therefore, as time goes by, the number of entities increases due to incremental behavior. In other words, the number of blocks and the number of entities arriving in the blocks increase. On the other hand, commonly, the amount of computational resources (especially memory and CPU) is limited, which hampers the techniques to consider all entities during blocking. For this reason, a criterion such as a time window for removing entities becomes necessary. The application of a time window is a trade-off between the effectiveness results (since it discards entities) and rational use of computational resources (since a large number of entities can overload the available resources). Note that the idea behind time windows is to discard old entities based on the principle that entities sent in a similar period of time have a higher chance of resulting in matches [9,14,59].

In this work, we apply a sliding (time)-window strategy, which defines the time interval for which the entities should be maintained in the update-oriented data structure, preventing the structure from excessively consuming memory. For instance, consider the example in Figure 4, where the entities are divided into three increments and sent at three different time points T_1, T_2, and T_3. The size of the sliding window (i.e., the time threshold) is given by the time interval of two increments. Therefore, entities that exceeded the time equivalent to the time of two increments will be discarded. In other words, these entities will no longer be considered by the blocking technique. In the first increment, i.e., T_1, e_1 and e_3 are blocked, generating blocks b_1, b_2, and b_3. In T_2, e_2 and e_4 are blocked. Considering the blocks previously created, e_2 and e_4 are added to b_1 and b_3. Since the size of the window is equivalent to two increments, for the third increment, the window slides for the next increment (i.e., T_3). For this reason, entities e_1 and e_3 (sent in the first increment) are removed from the blocks that contain them. Block b_2 is discarded since all entities contained in it were removed. Regarding the entities of the third increment, entity e_5 is inserted into b_1 and b_3, and e_6 is also inserted into b_1, triggering the creation of a new block b_4.

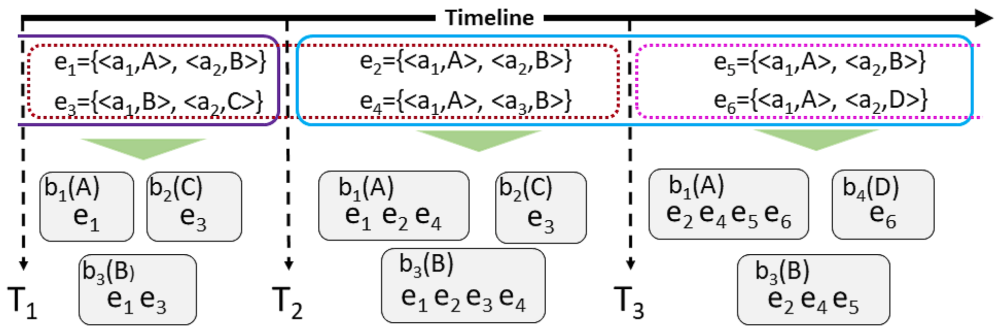


Figure 4. Time-window strategy for incremental blocking.

6. Attribute Selection

Real-world data sources can present superfluous attributes, which can be ignored by the blocking task [5]. Superfluous attributes are attributes whose tokens will generate low-quality blocks (i.e., with entities with low chances of matching) and, consequently, increase the number of false matches. To prevent the blocking task from handling unnecessary data, which consumes computational resources (processing and memory), we propose the attribute selection algorithm to remove the superfluous attributes.

Although some works [11,20,31,32] exploit attribute information to enhance the quality of blocks, none of them proposes the disposal of attributes. Therefore, in these works, the tokens from superfluous attributes are also considered in the blocking task and, consequently, they consume computational resources. On the other hand, the discarding of attributes may negatively affect the identification of matches since tokens from discarded attributes can be decisive in the detection of these matches. Therefore, the attribute selection should be performed accurately, as discussed in Section 7. To handle the superfluous attributes, we developed an attribute selection strategy, which is applied to the sender component (see Figure 2). Note that, the sender component is executed in a standalone way.

For each increment received by the sender component, the entity attributes are evaluated and the superfluous ones are discarded. Thus, the entities sent by the sender are modified by removing the superfluous attributes (and their values). As stated, discarding attributes may negatively impact effectiveness, since the discarded attributes for a specific increment can become relevant in the next increment. To avoid this problem, the attribute selection strategy takes the attributes into account globally, i.e., the attributes are evaluated based on the current increment and the previous ones. Moreover, the applied criterion to discard attributes focuses on attributes whose tokens have a high chance of generating low-quality blocks, which is discussed throughout this section. Based on these two aspects, the discarded attributes tend to converge as the increments are processed and do not significantly impact the effectiveness results, as highlighted in Section 7.2. Note that even if a relevant attribute is discarded for a specific increment it can return (not be discarded) for the next increments.

In this work, we explore two types of superfluous attributes: attributes of low representativeness and unmatched attributes between the data sources. The attributes of low representativeness are related to attributes whose contents do not contribute to determining the similarity between entities. For example, the attribute *gender* has a low impact on the similarity between entities (such as two male users from different data sources) since its values are commonly shared by a huge number of entities. Note that the tokens extracted from this attribute will generate huge blocks that consume memory unnecessarily since these tokens are shared by a large number of entities. Hence, this type of attribute is discarded by our blocking technique to optimize memory consumption without a significant impact on the effectiveness of the generated blocks.

To identify the attributes of low representativeness, the attribute selection strategy measures the attribute entropy. The entropy of an attribute indicates the significance of the attribute, i.e., the higher the entropy of an attribute, the more significant the observation of a particular value for that attribute [11]. We apply the Shannon entropy [60] to represent the information distribution of a random attribute. Assume a random attribute X with alphabet χ and the probability distribution function $p(x) = Pr\{X = x\}, x \in \chi$. The *Shannon entropy* is defined as $S(X) = -\sum_{x \in \chi} p(x) \log p(x)$. The attributes with low entropy values (i.e., low representativeness) are discarded. In this work, we removed the attributes whose entropy values were in the first percentile (i.e., 25%) of the lowest values. As stated in Section 2, the use of this condition was based on previous experiments.

Unmatched attributes between the data sources are attributes that do not have a corresponding attribute (i.e., with similar content) in the other data source. For instance, assume two data sources D_1 and D_2, where D_1 contains the attributes *full name* and *age*, and D_2 the attributes *name* and *surname*. Since *full name*, *name*, and *surname* address the same content, it is possible to find several similar values provided by these attributes. However, D_2 does not have any attributes related to *age*. That is, *age* will generate blocks from its values but these blocks will hardly be relevant to determining the similarities between entities. It is important to highlight that even though some metablocking-based techniques remove blocks with only one entity, the blocks are generated and consume resources until the blocking filter removes them. For this reason, this type of attribute is discarded by our blocking technique before the block generation.

Regarding the unmatched attributes, attribute selection extracts and stores the attribute values. Hence, for each attribute, a set of attribute values is generated. Based on the similarity between the sets, a similarity matrix is created denoting the similarity between the attributes of D_1 and D_2. To avoid expensive computational costs for calculating the similarity between attributes, we apply Locality-Sensitive Hashing (LSH) [61]. As previously discussed, LSH is commonly used to approximate near-neighbor searches in high-dimensional spaces, preserving the similarity distances and significantly reducing the number of attribute values (or tokens) to be evaluated. In this sense, for each attribute, a hash function converts all attribute values into a probability vector (i.e., hash signature).

Since the hash function preserves the similarity of the attribute values, it is possible to apply distance functions (e.g., Jaccard) to efficiently determine the similarity between the sets of attribute values [56]. Then, the hash function can generate similarity vectors that will feed the similarity matrix and guide the attribute selection algorithm. This similarity matrix is constantly updated as the increments arrive. Therefore, the similarity between the attributes is given by the mean of the similarity per increment. The matrix is evaluated and the attributes with no similarity to attributes from the other data source are discarded. After removing the superfluous attributes (and their values), the entities are sent to the blocking task. Note that instead of only generating hash signatures as in the noise-tolerant algorithm, LSH is applied in the attribute selection to reduce the high-dimensional space of the attribute values and determine the similarity between attributes based on a comparison of their hash signatures.

7. Experiments

In this section, we evaluate the proposed blocking technique in terms of efficiency and effectiveness, as well as the application of the attribute selection and top-n neighborhood strategies to the proposed technique. To conduct a comparative experiment, we also evaluate the proposed technique against a baseline one, which is described in Section 7.1. Moreover, we present the configuration of the computational cluster, the experimental design, and the achieved results.

The experiments address the following research questions:

- **RQ1**: In terms of effectiveness, is the proposed blocking technique equivalent to the state-of-the-art technique?
- **RQ2**: Does the noise-tolerant algorithm improve the effectiveness of the proposed blocking technique in scenarios of noisy data?
- **RQ3**: Regarding efficiency, does the proposed blocking technique (without the attribute selection and top-n neighborhood strategies) outperform the baseline technique?
- **RQ4**: Does the attribute selection strategy improve the efficiency of the proposed blocking technique?
- **RQ5**: Does the top-n neighborhood strategy improve the efficiency of the proposed blocking technique?

7.1. Baseline Technique

As stated in previous sections, state-of-the-art blocking techniques do not work properly in scenarios involving incremental and streaming data. For this reason, a comparison of our technique with these blocking techniques is unfair and mostly unfeasible. Thus, in this work, to define a baseline to compare with the proposed blocking technique, we implemented a metablocking technique capable of dealing with streaming and incremental data called streaming metablocking. This section provides an overview of streaming metablocking in terms of implementation and how it applies to the context of this work (i.e., streaming and incremental data).

Streaming metablocking is based on the parallel workflow for metablocking proposed in [22]. However, it was adapted to provide a fair comparison with our technique in terms of effectiveness and efficiency. The first adaptation was related to addressing incremental behavior, which needs to update the blocks to consider the arriving entities (i.e., new entities). Using a brute-force strategy, after the arrival of a new increment, streaming metablocking needs to rearrange all blocks including the blocks that were not updated (e.g., insertion of new entities). 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, streaming metablocking only considers the blocks that were updated for the current increment. Note that the original workflow proposed in [22] did not consider incremental behavior; therefore, the idea to (re)process the blocks generated previously was considered in this work.

The second adaptation of metablocking was related to streaming behavior. Streaming metablocking was implemented following the MapReduce workflow proposed in [22]. However, note that this parallel-based workflow was developed to handle batch data at once. Therefore, the workflow was redesigned to consider incremental behavior and allow the processing of streaming data. To this end, streaming metablocking was implemented using the Flink framework (i.e., MapReduce), which natively supports streaming behavior, as described in Section 4. In this sense, streaming metablocking applies a data structure to store the blocks previously generated and also avoids comparisons between entities previously compared (i.e., unnecessary comparisons).

On the other hand, it is necessary to highlight that the blocking technique proposed in this paper applies a different workflow, which reduces the number of MapReduce jobs compared to the streaming metablocking workflow described in Section 4. Moreover, the proposed strategies (i.e., attribute selection and top-*n* neighborhood) are not applied in streaming metablocking.

7.2. Configuration and Experimental Design

We evaluated our approach (https://github.com/brasileiroaraujo/ER_Streaming, accessed on 4 December 2022) in terms of effectiveness and efficiency. We ran our experiments in a cluster infrastructure with 21 nodes (one master and 20 slaves), each with one core. Each node had an Intel(R) Xeon(R) 2.10GHz CPU, 4GB memory, running on the 64-bit Ubuntu/Linux 16.04.5 OS with a 64-bit JVM, Apache Kafka (https://kafka.apache.org/, accessed on 4 December 2022) 2.1.0, and Apache Flink (https://flink.apache.org/, accessed on 4 December 2022) 1.7.1. Note that both the proposed technique and streaming metablocking applied Flink [62] as the MapReduce framework. The application of Flink was motivated by its better efficiency results compared to other frameworks such as Apache Spark in streaming scenarios [63,64]. Regarding the streaming processing platform to connect the sender and blocking task components (in the architecture), Apache Kafka [55] was applied. We used three real-world pairs of data sources, which are available in the project's repository, as described in Table 4: (i) Amazon-GP, which includes product profiles provided by amazon.com and google.com; (ii) IMDB-DBpedia, which includes movie profiles provided by imdb.com and dbpedia.org; and (iii) DBPedia$_1$-DBPedia$_2$, which consists of two different versions of the DBPedia Infobox dataset (https://wiki.dbpedia.org/Datasets, accessed on 4 December 2022, snapshots from October 2007 and 2009, respectively). Table 4 also shows the number of entities (D) and attributes (A) contained in each dataset, as well as the number of duplicates (i.e., matches—M) present in each pair of data sources.

Table 4. Data source characteristics.

Pairs of Datasets	$\|\mathbf{D_1}\|$	$\|\mathbf{D_2}\|$	$\|\mathbf{M}\|$	$\|\mathbf{A_1}\|$	$\|\mathbf{A_2}\|$
Amazon-GP	1354	3039	1104	4	4
IMDB-DBpedia	27,615	23,182	22,863	4	7
DBpedia$_1$-DBpedia$_2$	100,000	200,000	76,424	13,492	23,284

Blocking large data sources requires a huge amount of resources (e.g., memory and CPU) [19,29]. Moreover, it is necessary to highlight that a high number of attributes commonly tends to increase the amount of information and, consequently, the number of tokens related to each entity, which also increases resource consumption [5,13]. To avoid problems involving a lack of resources in our cluster, DBPedia$_1$-DBPedia$_2$ is a sample from the whole data source containing millions of entities. In addition, since the cluster used in this experiment was able to process at most tens of thousands of entities per increment, data sources containing millions of entities would significantly increase the number of increments (by several hundred) unnecessarily. Then, we randomly selected a sample of 100,000 entities from DBPedia$_1$ and 200,000 entities from DBPedia$_2$. It is important to highlight that the samples maintain proportionality in terms of the number of duplicates and the difference in the number of entities between the data sources.

To simulate streaming data behavior, a data-streaming sender was implemented. This sender reads entities from the data sources and sends entities in each τ time interval (i.e., increment). In this work, $\tau = 1$ min (i.e., one increment per minute) is used for all experiments. To measure the effectiveness of the blocking, we use the (i) pair completeness ($PC = \frac{|M(B')|}{|M(D_1,D_2)|}$), which estimates the portion of matches identified, where $|M(B')|$ is the number of duplicate entities in the set of pruned blocks B' and $|M(D_1, D_2)|$ is the number of duplicate entities between D_1 and D_2; (ii) pair quality ($PQ = \frac{|M(B')|}{||B'||}$), which estimates the executed comparisons that result in matches, where $||B'||$ is the number of comparisons to be performed in the pruned blocks; and (iii) reduction ratio ($RR = 1 - \frac{||B'||}{||B'_{Metablocking}||}$), which estimates the comparisons avoided in B' (i.e., $||B'||$) with respect to the comparisons produced by the baseline technique (i.e., metablocking). PC, PQ, and RR take values in [0, 1], with higher values indicating a better result. It is important to highlight that schema-agnostic blocking techniques (even state-of-the-art techniques) yield high recall (i.e., PC) but at the expense of precision (i.e., PQ) [5,11]. For schema-agnostic blocking techniques, the low PQ is due to the unnecessary comparisons defined by the generated blocks: redundant comparisons entail the comparison of entities more than once and superfluous comparisons entail the comparison of non-matching entities [11].

For efficiency, we comparatively evaluate the maximum number of entities processed per increment for the proposed technique and streaming metablocking. To measure the maximum number of entities processed per increment, we monitored the back pressure (https://ci.apache.org/projects/flink/flink-docs-stable/monitoring/back_pressure.html, accessed on 28 September 2022) metric. This metric takes values in the interval [0, 1], where a high back pressure (i.e., higher than 0.5) means that the task is producing data faster than the downstream operators (in the cluster) can consume. In other words, a high back-pressure value denotes that the cluster does not have enough resources to process the amount of data sent by the sender component per increment. Thus, from the back pressure, it is possible to define the computational limit (in terms of the number of entities per increment) of the cluster for each blocking technique.

To evaluate the impact of the number of neighborhood entities in the top-n neighborhood strategy, we conducted experiments varying the value n. The value n starts with a small number of neighborhood entities (i.e., 25 entities) and increases to a large number of neighborhood entities (i.e., 6400 entities). Thus, it is possible to compare the effectiveness and efficiency results for different values of n. We did not increase the value of n to over 6,400 entities because the achieved results did not present significant gains in terms of effectiveness. These results are discussed in Section 7.3. Following, we use the acronyms PT for the proposed technique, SM for streaming metablocking, AS for attribute selection, top-n for the top-n neighborhood strategy, and NT for the noise-tolerant algorithm.

7.3. Effectiveness Results

In Figure 5, the evaluation of the effectiveness of the PT and SM can be seen. The effectiveness results of the PT (without AS and top-n) are similar to the results achieved by the technique proposed in [19]. This occurred due to both techniques following the same blocking workflow and applying the same pruning algorithm (i.e., WNP). Since the window size is a limitation of streaming processing and interferes with effectiveness (true matches will never be compared if they are not covered by the window size), we implemented the sender to always send true matches (according to the ground truth) in a time interval covered by the window size. Note that the effectiveness of SM and the PT are exactly the same (Theorem 1) when the PT does not apply the top-n neighborhood and/or attribute selection strategies. Therefore, part of the research question **RQ1** (i.e., *Regarding effectiveness, is the proposed blocking technique with/**without** the proposed strategies comparable to the existing state-of-the-art technique?*) is answered since the PT and SM present the same effectiveness.

For the PQ and RR, the application of the top-n neighborhood and attribute selection strategies in the PT outperformed the results of SM for all data sources. This occurred due to the fact that the application of these strategies decreased the number of entity pairs to be compared within the blocks. Note that the top-n neighborhood aims to reduce the number of neighbor entities in the similarity graph. For this reason, the size of the blocks (in the number of entities) tends to decrease and, consequently, the number of entities to be compared after the blocking step. The attribute selection strategy decreases the number of blocks since it discards some tokens that would generate new blocks. Therefore, it is important to highlight that the attribute selection strategy works as a controller of the block amount, which is one of the challenges faced by token-based blocking techniques [65]. Then, the application of these strategies improves the PQ and RR, which are directly related to the number of comparisons generated from the blocks (i.e., $||B'||$). We can also analyze the RR as an efficiency metric since it estimates the number of comparisons to be executed in the ER task. In this sense, we can identify the efficiency gains (for the ER task as a whole) provided by the application of both strategies, which produces blocks with fewer comparisons.

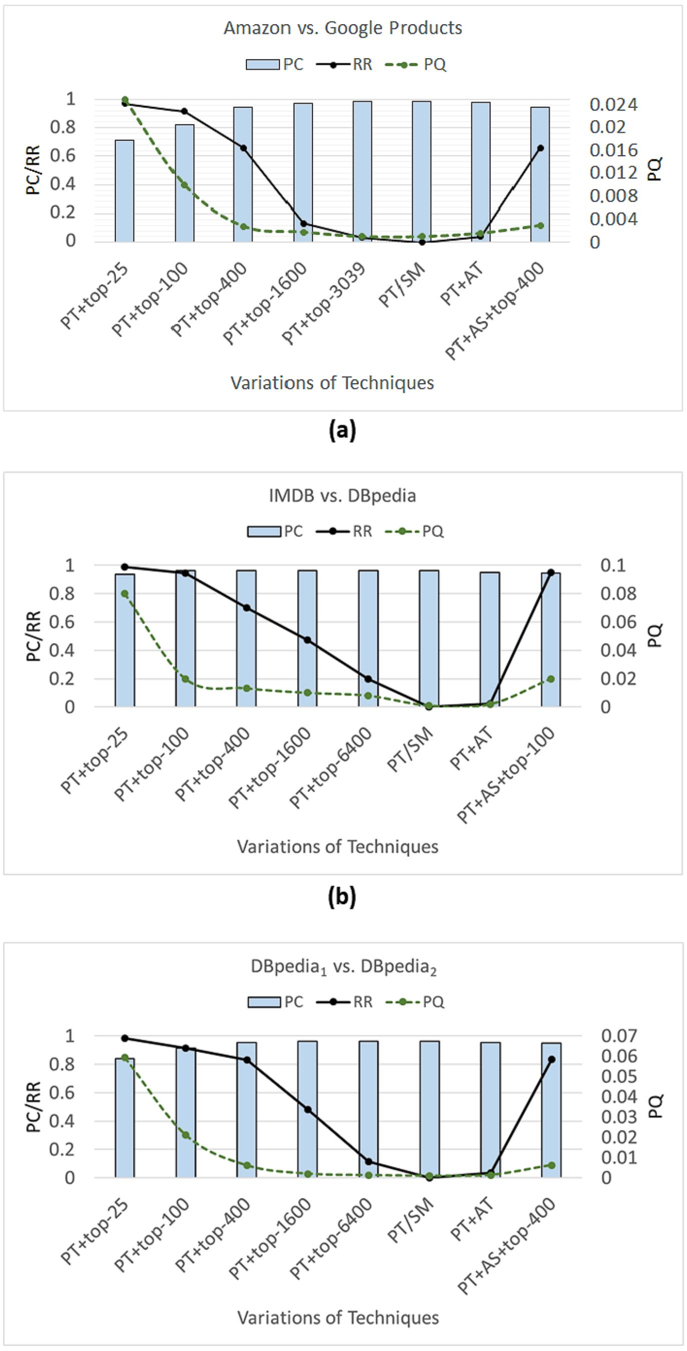


Figure 5. Effectiveness results of the techniques for the data sources: (**a**) Amazon vs. Google Product, (**b**) IMDB vs. DBpedia, and (**c**) DBpedia$_1$ vs. DBpedia$_2$.

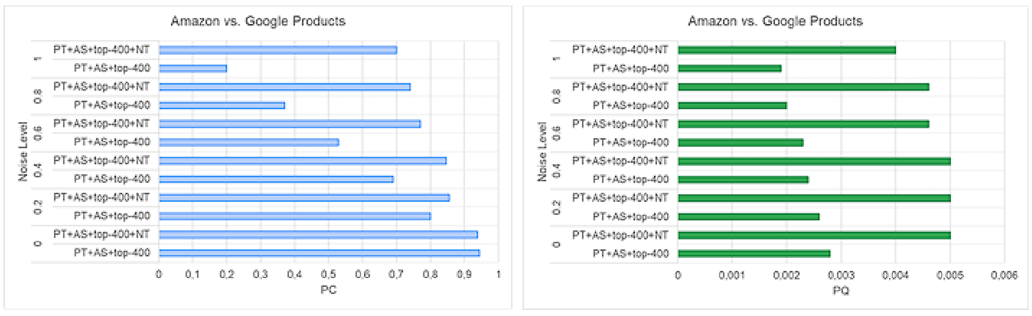
When analyzing the PQ and RR, notice that as the value of n in the top-n neighborhood strategy decreased, these metrics achieved better results. This behavior occurred due to the reduction in the number of neighbor entities, which decreased the number of entities to be compared in ER task. For this reason, low values of n implied good results for the PQ and RR. However, the PC decreased when the top-n neighborhood was applied; in the worst case, for Amazon-GP, the PC achieved was 0.72 for $n = 25$. Thus, the top-n neighborhood significantly decreased the PC to very low n values (e.g., $n = 25$). On the other hand, we identified that, even for large data sources (e.g., DBPedia$_1$-DBPedia$_2$), n values between 100 and 400 presented reasonable PC values (PC higher than 0.95) for all data sources. For instance, the top-n neighborhood, with $n = 400$, decreased the PC by only 1.6 percentage points, on average, for all data sources. In this experiment, we selected the best top-n value to be applied together with the attribute selection (i.e., PT + AS + top-n). In this sense, the application of both strategies maintained efficiency, enhancing the RR and PQ with a small decrease in the PC compared to the baseline technique.

When attribute selection was applied, the PC decreased, the worst being for IMDB-DBpedia, by 1.5 percentage points. Thus, it is possible to infer that AS discards only those attributes that do not significantly interfere with the PC results (i.e., superfluous attributes). Regarding the PQ and RR, the application of attribute selection showed small improvements in these metrics since this strategy reduced the number of generated blocks and, consequently, the number of comparisons to be performed in the ER task. The application of the top-n neighborhood and attribute selection together achieved reasonable results in terms of effectiveness for all data sources since the PQ and RR increased compared to the PC, which suffered a decrease of only one percentage point, on average. Therefore, based on the results highlighted in this subsection, it is possible to affirm that the proposed blocking technique with the application of the proposed strategies is comparable to SM in terms of effectiveness, answering the question in **RQ1**.

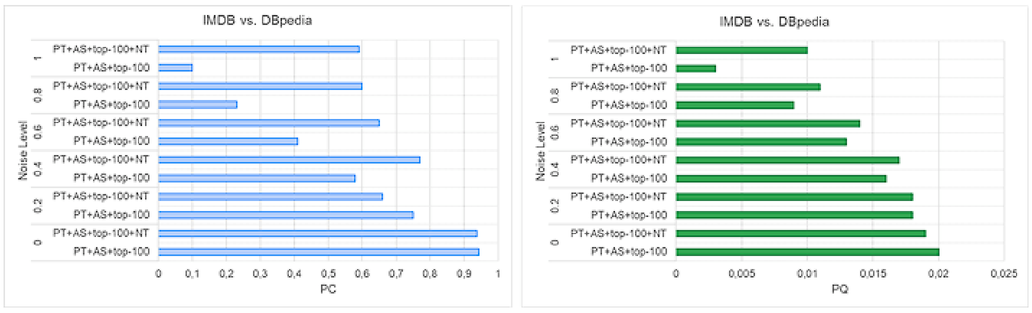
Noisy data scenario. To evaluate the effectiveness results of the proposed techniques with the application of the noise-tolerant algorithm, we inserted synthetic typos and misspellings (i.e., noise) into the attribute values of the entities to simulate scenarios of noisy data. For each dataset pair described in this section, the noise was inserted in the first dataset (the datasets with noise are available in the project repository). To simulate the occurrence of typos/misspellings [36], for all attribute values (from an entity), one character of each value (i.e., token) was randomly exchanged for another or additional characters were inserted into the attribute value. The noise level in the dataset varied between 0 to 1, where 0 means no noise was inserted into the dataset and 1 means noise was inserted into the attribute values of all entities in the dataset. Hence, a noise level of 0.6 indicates that 60% of the entities (contained in the first dataset) had noise inserted into their attribute values.

Figure 6 illustrates the effectiveness results of the proposed technique (with/without the noise-tolerant algorithm) for each pair of datasets. In this sense, a comparative analysis was conducted to evaluate the application of the noise-tolerant algorithm to the proposed technique. To this end, a combination of the proposed technique with the attribute selection and top-n neighborhood strategies, which achieved the best effectiveness results in the experiment addressed in this section, was considered. The following scenarios were selected throughout this experiment: PT+AS+top -400 for Amazon-GP, PT+AS+top-100 for IMDB-DBpedia, and PT+AS+top-400 for DBPedia$_1$-DBPedia$_2$.

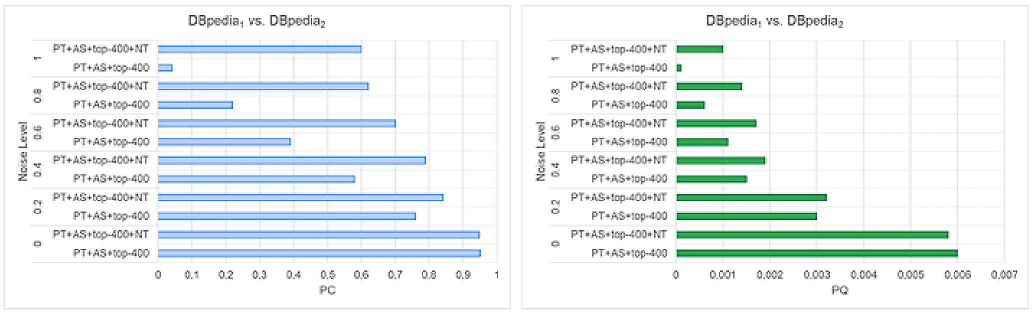
Regarding the PC metric, the application of the NT increased the PC in all scenarios with the presence of noise. Even for different scenarios of noise levels, the application of the noise-tolerant algorithm to the proposed technique presented better results for all pairs of datasets. It is important to highlight that as the noise level increased, the effectiveness metrics (i.e., PC and PQ) decreased in both scenarios (with/without NT). This occurred due to the fact that the noise on the data negatively impacted the block generation, as already discussed in Sections 3 and 4. However, the effectiveness metrics decreased rapidly when the NT was not applied. Thus, the application of the NT amortized the decrease in effectiveness. Even for the highest level of noise (i.e., a noise level of 1.0), the application of the noise-tolerant algorithm to the proposed technique achieved a pair completeness greater than 60% for all pairs of datasets.



(a)



(b)



(c)

Figure 6. Effectiveness results in the context of noisy data for the data sources: (**a**) Amazon vs. Google Product, (**b**) IMDB vs. DBpedia, and (**c**) DBpedia$_1$ vs. DBpedia$_2$.

In terms of the PQ metric, in scenarios with the presence of noise, the application of the NT almost doubled the PQ results, on average. The main reason for this was the generation of multiple tokens per entity attribute (based on a particular attribute value) as blocking keys when the NT was not applied. Since non-matching entities eventually share multiple tokens, they were included in the same block erroneously. On the other hand, the NT algorithm generated a hash value based on a particular attribute value. Thus, non-matching entities sharing the same hash value were harder to find than non-matching entities sharing tokens in common. For this reason, the NT enhanced the PQ metric. Therefore, based on the results highlighted in this experiment, it is possible to affirm that the application of the noise-tolerant algorithm to the proposed technique enhances the effectiveness in scenarios involving noisy data, answering the research question **RQ2** (i.e., *Does the noise-tolerant algorithm improve the effectiveness of the proposed blocking technique in scenarios of noisy data?*).

7.4. Efficiency Results

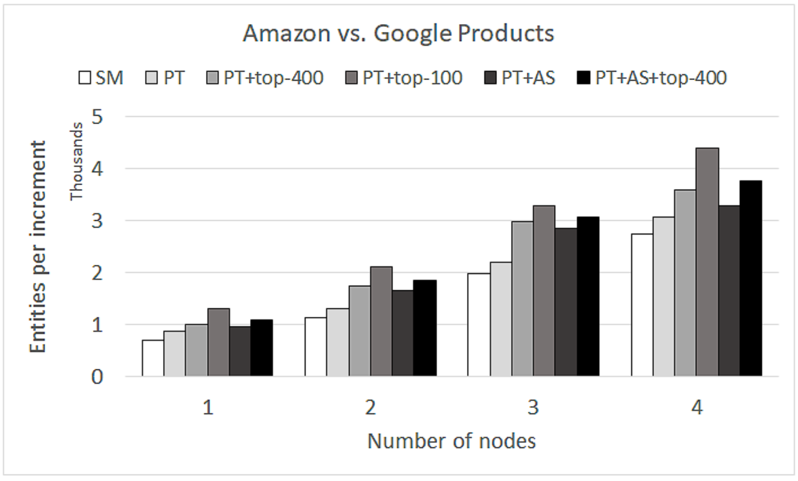
Since the PT is an evolution of the technique proposed in [19], it is possible to affirm that the PT overcomes the technique in terms of efficiency. As stated in [19], due to excessive resource consumption, the technique was not able to process the IMDB-DBpedia pair with fewer than 12 nodes, each with 6GB of memory. The nodes applied in [19] also had an Intel(R) Xeon(R) 2.10GHz CPU and ran on a 64-bit Debian GNU/Linux OS with a 64-bit JVM. On the other hand, our technique was able to process the same data source pair with one node with 4GB of memory. Moreover, the PT was also able to process larger data sources such as DBpedia$_1$-DBpedia$_2$. This occurred due to the application of the AS algorithm, top-*n* strategy, and Flink (instead of Spark), which enhanced efficiency. For this reason, the focus of our experiments was to evaluate the impact of the AS and top-*n* on the PT.

To evaluate the proposed technique against SM (Figure 7), we applied a sliding window with a size of four minutes to all data sources since this size achieved reasonable effectiveness results even if the entities were sent in the order stored in the data sources. To evaluate efficiency, we measured the maximum number of entities processed by the techniques per increment, varying the number of nodes between 1 and 20. Note that the maximum number of processed entities denotes the efficiency of the techniques in terms of resource management (e.g., CPU and memory). Since the pair Amazon-GP was composed of small data sources, we varied the number of nodes up to four. After that, the number of nodes did not interfere with the results and all data from the data source were consumed in only one increment.

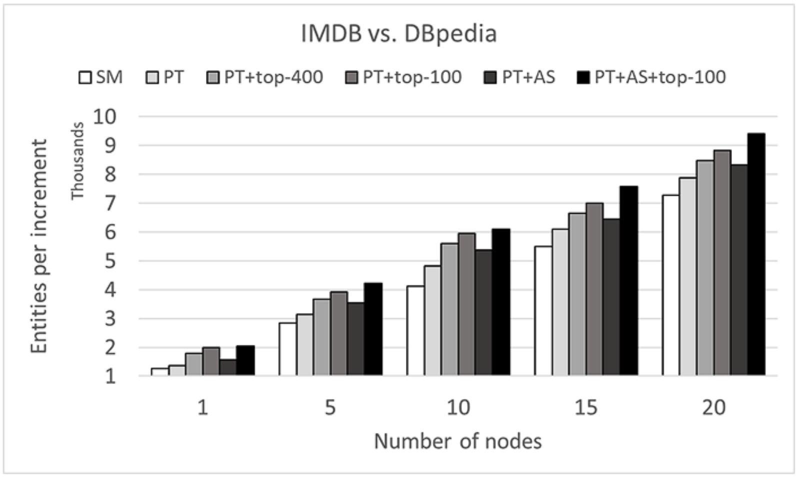
Based on the results shown in Figure 7, the PT outperformed SM for all data sources, even without the application of the top-*n* neighborhood or attribute selection strategies due to the reduction in MapReduce jobs (see Section 4.2), which saved computational resources and improved efficiency. On average, the PT increased the number of processed entities by 12% compared to SM. Thus, these results answer the research question **RQ3** (i.e., *Regarding efficiency, does the proposed blocking technique (without attribute selection and top-n neighborhood strategies) outperform the baseline technique?*).

Since the attribute selection strategy aims to reduce the number of tokens and, consequently, the number of generated blocks, we evaluated the application of attribute selection regarding the reduction in the number of tokens. Therefore, evaluating the PT with attribute selection (i.e., PT + AS), the removal of attributes resulted in a reduction of 10% for Amazon-GP, 34% for IMDB-DBpedia, and 2% for DBpedia$_1$-DBpedia$_2$ in terms of the number of generated tokens compared to when attribute selection was not applied. It is important to highlight that the 2% reduction for DBpedia$_1$-DBpedia$_2$ resulted in 18,000 fewer tokens. In other words, attribute selection avoided the creation of more than 18,000 blocks during the blocking phase. Specifically, attribute selection discarded the following number of attributes for each data source: one from Amazon, one from GP, one from IMDB, four from DBpedia, 3,375 from DBpedia$_1$, and 5,826 from DBpedia$_1$. These reductions in the number of blocks directly affected the number of entities processed per increment since it reduced resource consumption. Therefore, based on the results presented in Figure 7, it can be stated that the PT + AS outperformed the PT for all data sources. On average, the PT + AS increased the number of entities processed per increment by 20% compared to the PT, which answers the research question **RQ4** (i.e., *Does the attribute selection strategy improve the efficiency of the proposed blocking technique?*).

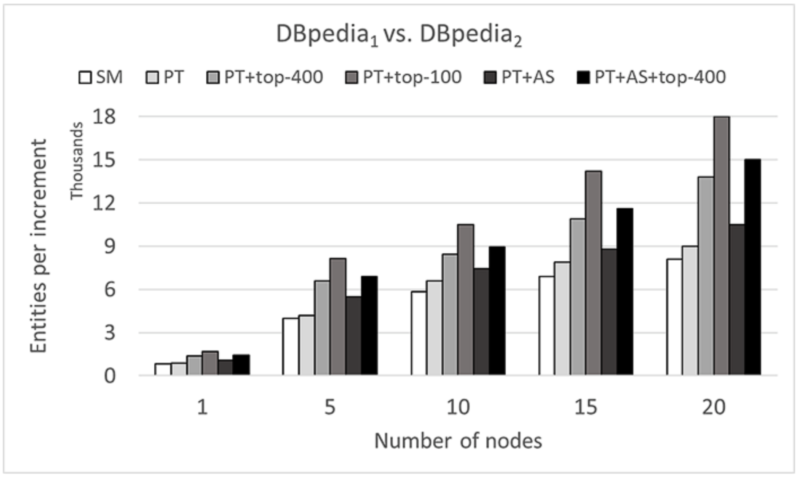
Regarding the top-*n* neighborhood strategy, we evaluated the PT with the top-100 and top-400, the two most promising values in terms of effectiveness. Low values of "*n*" (in top-*n*) negatively impacted the PC since the small number of neighbor entities reduced the chance to find true matches. On the other hand, high "*n*" values did not decrease the PQ metric since high "*n*" values achieved the original number of neighbor entities for each entity. Thus, the application of high "*n*" values tended to achieve similar results (in terms of effectiveness) to the PT without the top-*n* neighborhood strategy. Regarding efficiency, the lower the "*n*" values (in top-*n*), the greater the number of entities processed per increment. This occurred due to the reduction in the neighbor entities linked to each node (i.e., entity) in the similarity graph and, consequently, the reduction in the consumed computational resources. For this reason, in DBpedia$_1$-DBpedia$_2$, the PT applying the top-100 neighborhood increased the number of entities processed per increment by 85%, on average, compared to the PT, whereas the application of the top-400 increased the number of processed entities by 47%. Thus, it is possible to answer research question **RQ5** (i.e., *Does the top-n neighborhood strategy improve the efficiency of the proposed blocking* *technique?*) since the achieved results show that the top-*n* neighborhood strategy improved the PT's efficiency.



(a)



(b)



(c)

Figure 7. Efficiency results of the techniques for the data sources: (**a**) Amazon vs. Google Product, (**b**) IMDB vs. DBpedia, and (**c**) DBpedia$_1$ vs. DBpedia$_2$.

The application of the attribute selection and top-n neighborhood strategies to the PT enabled us to achieve better results in terms of efficiency for all data sources. For instance, in DBpedia_1-DBpedia_2, the combination of attribute selection and top-400 neighborhood provided the PT with an increase of 56%, on average, in terms of the number of entities processed per increment. On the other hand, the PT + AS increased the number of processed entities by 20%, whereas with the PT + top-400, these were increased by 47%. Therefore, based on the results presented in Figure 7, we highlight the efficiency improvements promoted by both strategies for all data sources. The efficiency gains are due to the reduced number of blocks and the generation of smaller blocks, which reduce resource consumption and increase the capacity to process more entities per increment.

Regarding scenarios where the noise-tolerant algorithm was applied to the proposed technique, there was no impact in terms of efficiency. The noise-tolerant algorithm was performed in the sender component (as described in Section 4) and produced the same amount of hash values as the proposed technique produced tokens (as stated in Section 2). In this sense, during all the experiments conducted in this Section, the execution time of the noise-tolerant algorithm did not exceed the time interval τ, which represents the time between two consecutive increments sent by the sender component (as described in Section 4). Moreover, since the computational costs to compare hash values were similar to comparing tokens during the blocking task and the number of generated hash values (when the noise-tolerant algorithm was applied) was the same as the number of generated tokens (when the noise-tolerant algorithm was not applied), the execution time of the proposed technique did not suffer significant interference to the noise-tolerant algorithm in terms of efficiency.

8. A Real-World Case Study: Twitter and News

Streaming data are present in our daily lives in different contexts including personal devices (mobile phones and wearable devices such as smartwatches), vehicles (intelligent transportation systems, navigation, theft prevention, and remote vehicle control), day-to-day living (tracking systems, various meters, boilers, and household appliances), and sensors (weather data temperature, humidity, air pressure, and measured data) [10]. Particularly, for social data, in a single minute, 456,000 tweets are posted, 2,460,000 pieces of content are shared on Facebook, Google conducts 3,607,080 searches, the weather channel receives 18,055,555 forecast requests, Uber riders take 45,787.54 trips, and 72 h of new videos are uploaded to YouTube and 4,146,600 videos are watched [66]. Streaming data can be utilized in various areas of study as they are linked with a large amount of information (oil price, exchange rate, social data, music, videos, and articles) that is found on the Web. Such data may also provide crucial information about product prices, consumer profiles, criminality, diseases, and disasters for companies, government agencies, and research agencies in real time [54]. For example, a company can evaluate (in real time) the public perception of its products through streaming data provided by social networks. In this context, ER becomes a fundamental task to integrate and provide useful information to companies and government agencies [4]. Particularly, blocking techniques should be used to provide efficiency to the ER task since it is necessary to handle a large amount of data in a short period of time.

News can be easily found on websites or through RSS feeds. For instance, RSS (Really Simple Syndication) feeds have emerged as an important news source since this technology provides real-time content distribution using easily consumable data formats such as XML and JSON. In this sense, social media (e.g., Twitter, Facebook, and Instagram) have become a valuable data source of news for journalists, publishers, stakeholders, and consumers. For journalists, social networks are an important channel for distributing news and engaging with their audiences [67]. According to the survey in [68], more than half of all journalists stated that social media was their preferred mode of communication with the public.

Over social networking services, millions of users share information on different aspects of everyday life (e.g., traffic jams, weather, and sporting events). The information provided by these services commonly addresses personal points of view (e.g., opinions, emotions, and pointless babble) about newsworthy events, as well as updates and discussions of these events [69]. For instance, journalists use social media frequently for sourcing news stories since the data provided by social media are easy to access by elites, regular people, and people in regions of the world that are otherwise isolated [70]. Since semantic standards enable the web to evolve from information uploaded online, this kind of data can benefit Web systems by aggregating additional information, which empowers semantic knowledge and the relationships between entities [71].

Among the social networking sites, we can highlight Twitter. Twitter is a popular micro-blogging service with around 310 million monthly active users, where users can post and read short text messages (known as tweets), as well as publish website links or share photos. For this reason, Twitter has emerged as an important data source, which produces content from all over the world continually, and an essential host for sensors for specific events. An event, in the context of social media, can be understood as an occurrence of interest in the real world that instigates a discussion-associated topic at a specific time [72]. The occurrence of an event is characterized by a topic and time and is often associated with entities such as news. Hot-topic events are grouped by Twitter and classified as trending topics, which are the most commented topics on Twitter. Twitter provides the trending topics list on its website and API, where it is possible to filter the trending topics by location (for instance, a specific city or country).

The main objective of this experimental evaluation was to validate (in terms of effectiveness) the application of the top-n neighborhood strategy, attribute selection strategy, and noise-tolerant algorithm to the proposed blocking technique in real-world streaming scenarios. More specifically, the real-world scenarios considered in this study were related to tweets and news. In this sense, it is important to note that the idea of this case study was to apply the proposed technique in order to group tweets that presented similarities regarding news published by media sites. In other words, given a set of news records provided (in a streaming way) by a news data source, the proposed technique should identify and group into the same block the tweets that address the same topic as a news record. In this aspect, our technique can be useful for journalists and publishers who are searching for instant information provided by tweets related to a specific event. Moreover, stakeholders, digital influencers, and consumers can benefit from the application of the proposed technique in the sense of analyzing tweets (for instance, applying a sentiment analysis approach) related to a specific topic described in a news article such as a novel product, the reputation of a brand, or the impact of media content.

8.1. Configuration and Experimental Design

Concerning the news data, we collected data provided by the Google News RSS feed (https://news.google.com/rss/, accessed on 28 September 2022), whereas the tweets were collected from the Twitter Developer API (https://developer.twitter.com/, accessed on 28 September 2022). For both data sources (a copy of the collected data is available in the project repository.), the data were filtered for the USA location (i.e., tweets and news in English) to avoid cross-language interference in the experiment. Considering that tweets and news can provide a huge number of tokens and since the number of tokens is directly related to the efficiency of the blocking technique (as discussed in Section 7), the framework Yake! [73] was applied as a keyword extractor. Yake! is a feature-based system for multilingual keyword extraction, which presents high-quality results in terms of efficiency and effectiveness compared to the competitors [73]. The application of Yake! is also useful in the sense of removing useless words (e.g., stop words) and extracting relevant words (i.e., keywords) from long texts such as a news item. Note that the words extracted from tweets and news items become tokens during the blocking task.

One of the challenges of this real-world experiment was computing the effectiveness results since the generation of the ground truth was practically unfeasible. In this scenario, for each news item, the ground truth consisted of a set containing all tweets that were truly related to the news item in question. To produce the ground truth regarding tweets and news, it was necessary to use a group of human reviewers to label the hundreds of thousands of tweets for the hundreds of news items collected during the experiment, which represents a high-cost and error-prone demand task in terms of time and human resources.

To minimize the stated challenge, we applied the strategy used in [74,75], which generates the ground truth based on the hashtags contained in the tweets. In this sense, we collected the trending topic hashtags from the Twitter Developer API that referred to the most commented topics on Twitter at a specific time. From the trending topic hashtags, a search was performed on the Google News RSS feed using each hashtag as the search key. In turn, Google News returned a list of news related to the hashtag topic. To avoid old versions of the news items, we considered news published on the same day the experiment was conducted. In this way, it was possible to define a link between the news and the hashtags (topics). When the tweets were collected, the hashtags were extracted in the sense of correlating the tweet with one of the trending topic hashtags. Hence, based on the trending topic hashtags as a connection point, it was possible to link tweets and news that addressed the same topic. For instance, from the hashtag "#BlackLivesMatter", it was possible to link tweets that contained this hashtag to news related to this topic provided by the Google News search using the hashtag as the search key. By doing this, we generated a ground truth that was able to support the effectiveness evaluation of the proposed blocking technique.

Regarding the computational infrastructure, we ran our experiments in a cluster with six nodes (one master and five slaves), each with one core. Each node had an Intel(R) Xeon(R) 2.10 GHz CPU, 3 GB memory, running on a 64-bit Windows 10 OS with a 64-bit JVM, Apache Kafka 2.1.0, and Apache Flink 1.7.1. This experiment was conducted between May 12 and May 16 2021 and was divided into 15 different moments at 3 different times each day. During this experiment, 483,811 tweets and 776 news items were collected and processed using the proposed technique. We applied a time window of 20 min with two minutes of sliding. These parameters were chosen to maximize the effectiveness results without back pressure problems (as discussed in Section 7.2).

To evaluate the top-*n* neighborhood, attribute selection, and noise-tolerant algorithm, we conducted experiments combining the application (or not) of these strategies to the proposed technique. Regarding the top-*n* neighborhood, we applied the top-400 and top-1600, since these parameters achieved better effectiveness results without back pressure problems. Concerning the attribute selection strategy, we performed experiments with and without its application. In terms of the noise-tolerant algorithm, we applied it to the scenario where the top-1600 and attribute selection were applied since this combination achieved the best effectiveness results. Note that from the Twitter data, the attributes were *text*, *user name*, *user screen name*, *hashtags*, *cited users*, and *created at*. The attributes from Google News included *title*, *text*, *published at*, and *list of links*. When the attribute selection was ignored, all the stated attributes were considered during the blocking. On the other hand, when attribute selection was applied, only the attributes *text* and *hashtags* from tweets and *title* and *text* from news items were considered. In all scenarios in the study case, the results of the attribute selection converged in the sense of achieving the same discarded attributes.

8.2. Results and Discussion

Assuming the configuration set previously described, we evaluated the effectiveness of the proposed technique (PT) when the top-*n* neighborhood strategy, attribute selection (AS) strategy, and noise-tolerant algorithm (NT) were applied, resulting in seven different combinations: PT, PT + AS, PT + Top-400, PT + AS + Top-400, PT + Top-1600, PT + AS + Top-1600, and PT + AS + Top-1600 + NT. Note that to evaluate the PQ metric, it was necessary to conduct a human evaluation. This occurred due to the fact that, even when applying the strategy to generate a ground truth based on the hashtags, the proposed technique can correlate tweets and news that truly address the same topic. However, these correlations may not have been included in the ground truth because the tweets did not present hashtags. Therefore, due to the challenges of generating the ground truth, only the PC metric was applied during the experiment.

Overall, based on the results illustrated in Figure 8, the PT combination achieved an average of 84% for the PC (considering all the moments). Moreover, the PC value was enhanced when the AS was applied. Specifically, as depicted in Figure 9, the PT + AS presented a PC value of 92% and the PT + AS + Top-1600 achieved a PC value of 91%, on average. In Figure 8, it is possible to see that the application of the AS tended to increase the PC results of the PT in all evaluated moments. These results can be explained due to the discarding of superfluous attributes, which may have disturbed the effectiveness results, as discussed in Section 6 and evaluated in Section 7.2. Considering the collected attributes from tweets (i.e., *text*, *user name*, *user screen name*, *hashtags*, *cited users*, and *created at*) and news items (i.e., *title*, *text*, *published at*, and *list of links*), when the AS was applied, only *text* and *hashtags* from tweets and *title* and *text* from news items were considered. Regarding the discarded attributes, we observed that *created at* and *published at* were removed by the entropy rule since all the values presented a small variation considering that the tweets and news items were collected at similar time intervals. Regarding *user name*, *user screen name*, and *cited users* from tweets and *list of links* from news items, we observed that they were discarded based on the unmatched attribute rule since none of these attributes presented a relation among them (or with any other attribute). Thus, the application of the AS discarded the superfluous attributes and, consequently, tended to generate high-quality blocks. It also improved effectiveness, as discussed in Section 6.

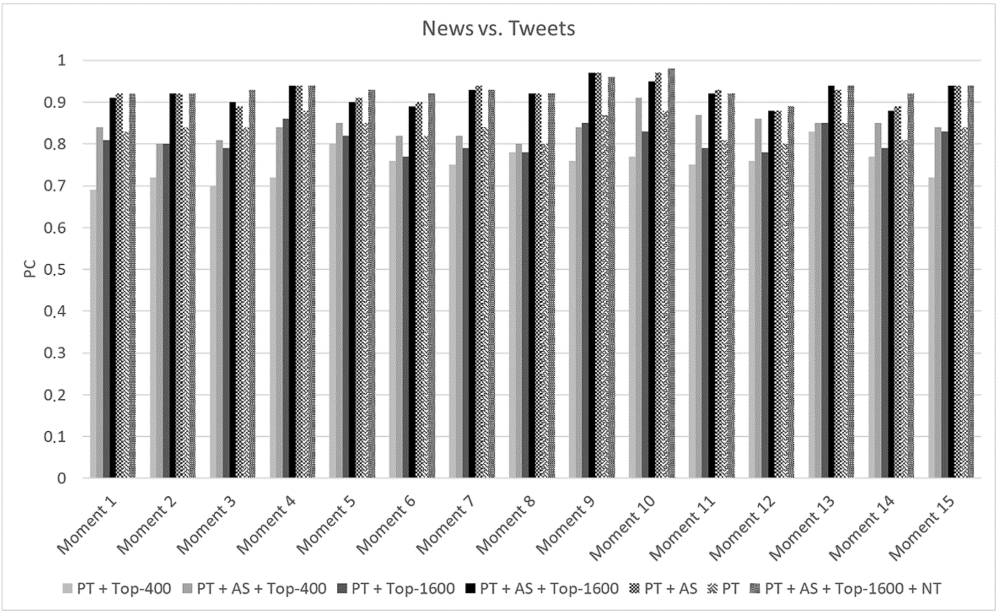


Figure 8. Effectiveness results of the proposed technique for the 15 different moments involving news and Twitter data.

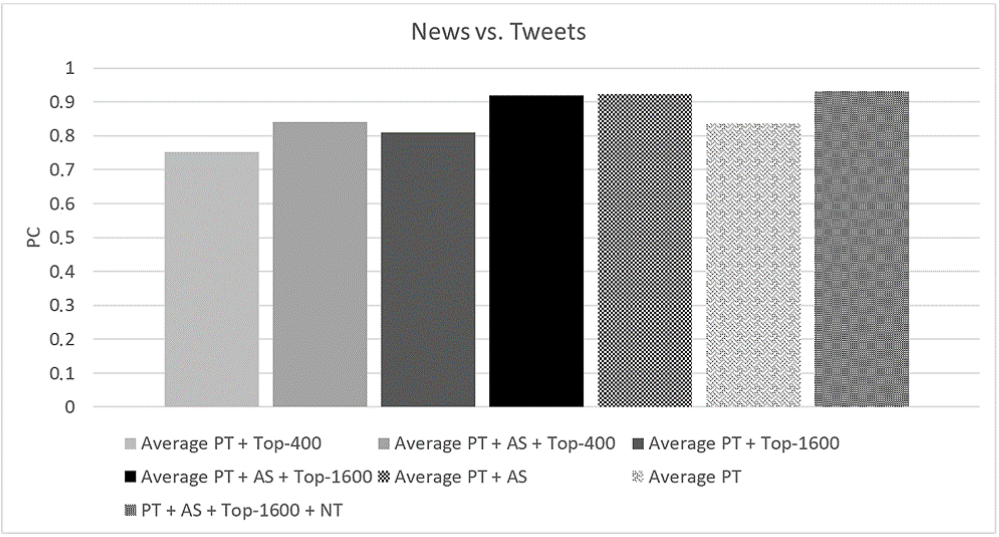


Figure 9. Average of effectiveness results of the proposed technique for news and Twitter data.

Based on the results depicted in Figure 8, it is possible to identify a pattern regarding the application of the top-*n* neighborhood and attribute selection. Notice that the application of the top-*n* neighborhood tended to decrease the PC metric compared to the PT (without top-*n*) results. More specifically, the application of the top-1600 decreased the PC metric by 3 percentage points on average and the top-400 decreased the PC metric by 8 percentage points on average, as depicted in Figure 9. These results were expected since they were also obtained in the experiments presented in Section 7.3. The top-*n* neighborhood was developed to enhance efficiency without significant interference in the effectiveness results, which was achieved when the top-1600 was applied (i.e., with only a 3% decrease). In contrast, when the top-*n* neighborhood and attribute selection were applied together, the PC results tended to increase and achieved similar results when only attribute selection was applied. For instance, the PT + AS achieved a PC value of only one percentage point higher than the PT + AS + Top-1600 on average, as illustrated in Figure 9.

Regarding the application of the noise-tolerant algorithm, the combination of the PT + AS + Top-1600 + NT achieved the best results among the other combinations in 12 of the 15 moments, as described in Figure 8. Only in three moments did the combinations PT + AS + Top-1600 or PT + AS outperform the PT + AS + Top-1600 + NT. However, in these three moments, the difference between the PC results was only one percentage point. Specifically, as depicted in Figure 9, the PT + AS + Top-1600 + NT presented a PC value of 93% on average, which was one percentage point higher than the PC average for the combination of the PT + AS. Based on these results, it is important to highlight that the application of the noise-tolerant algorithm tended to enhance the effectiveness of the proposed technique. This occurred due to the fact that the noise-tolerant algorithm allowed the proposed technique to handle data in the presence of noise, which is common in real-world scenarios. In other words, the noise-tolerant algorithm enabled the proposed technique to find matches that were not identified due to the noisy data.

It is important to highlight the effectiveness results achieved by the proposed technique during the case study. Moreover, although efficiency was not the main evaluation objective, with the application of the configuration set stated in Section 8.1, the proposed technique did not present back pressure problems. Taking into account the real-world scenario involved in this case study, the impact of superfluous attributes is clearer. As exposed by the works in [11,31], tokens provided by superfluous attributes tend to generate low-quality blocks. Furthermore, when data are provided by social networks, this challenge is strengthened [72]. For instance, a tweet may be linked to a news item that addresses a topic about "Black Lives Matter" only because the tweet presents "Lives" as a value in the attribute user name (since both share the token "Lives"). This link based only on the token "Lives", has a high chance of being a false-positive result. Although the experiments described in Section 7 also considered the application of attribute selection, an important result of this case study is the application of attribute selection in the sense of improving the effectiveness of real-world data sources. In this sense, the case study scenario presented several superfluous attributes (i.e., *created at*, *user name*, *user screen name*, *cited users*, *published at*, and *list of links*), which negatively interfered with the quality of the generated blocks. For this reason, based on the achieved results in this case study and the comparative experiments in Section 7, the attribute selection strategy emerges as an important ally for blocking techniques in the sense of enhancing not only effectiveness but also efficiency.

9. Conclusions and Further Work

Blocking techniques are largely applied as a pre-processing step in ER approaches in order to avoid the quadratic costs of the ER task. In this paper, we address the challenges involving heterogeneous data, parallel computing, noisy data, incremental processing, and streaming data. To the best of our knowledge, there is a lack of blocking techniques that address all these challenges simultaneously. In this sense, we propose a novel schema-agnostic blocking technique capable of incrementally processing streaming data in parallel. In order to enhance the efficiency of the proposed technique, we also propose the attribute selection (which discards superfluous attributes from the entities) and top-*n* neighborhood (which maintains only the top "*n*" neighbor entities for each entity) strategies. Furthermore, the noise-tolerant algorithm was proposed in order to enhance the effectiveness results of the proposed technique in scenarios involving noisy data. Based on the experimental results, we can confirm that our technique presents better efficiency results than the state-of-the-art technique (i.e., streaming metablocking) without a significant impact on effectiveness. This main result was achieved due to the reduction in the number of MapReduce jobs in the proposed workflow. Moreover, as addressed in Sections 7 and 8, the application of the proposed strategies improved the results achieved by our blocking technique in terms of efficiency and effectiveness in scenarios involving real-world data sources.

In future work, we intend to study different kinds of window strategies (i.e., not only time-window strategies) that can be applied to the proposed technique, for instance, maintaining (and prioritizing) blocks constantly updated during increments. Moreover, we would like to highlight the possibility of adapting other existing blocking techniques to our parallel workflow such as progressive blocking techniques [65,76] and blocking based on machine learning concepts [12,77]. Thus, through the proposed workflow, it is possible that such techniques can handle streaming data.

One aspect that can be explored in future works is related to the lack of balance in data increments. In this context, the number of entities contained in each micro-batch to be processed by blocking techniques varies over time. This behavior commonly occurs in streaming sources that present load peaks. For instance, traffic sensors commonly send a high amount of data during rush hour. On the other hand, the amount of produced data is reduced during low-traffic hours. In this sense, parallel-based blocking techniques should be able to consume resources dynamically. Dynamic schedulers that provide an on-demand allocation of resources in distributed infrastructures could be applied [78,79]. These schedulers control the number of resources (e.g., number of nodes) according to the amount of data received by the blocking technique in question. Since the lack of balance in data increments is a problem faced by incremental techniques, the development of dynamic schedulers for blocking techniques has emerged as an open area to be explored.

Author Contributions: Conceptualization, T.B.A., K.S., C.E.S.P. and J.N.; methodology, T.B.A., K.S., C.E.S.P. and J.N.; software, T.B.A. and T.P.d.N.; validation, T.B.A. and T.P.d.N.; formal analysis, T.B.A., K.S. and C.E.S.P.; investigation, T.B.A., K.S., C.E.S.P. and T.P.d.N.; resources, K.S., J.N. and T.P.d.N.; data curation, T.B.A. and T.P.d.N.; writing—original draft preparation, T.B.A.; writing—review and editing, T.B.A., K.S., C.E.S.P., J.N. and T.P.d.N.; visualization, T.B.A.; supervision, K.S. and C.E.S.P.; project administration, K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used for this work is available at the project's repository.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gentile, A.L.; Ristoski, P.; Eckel, S.; Ritze, D.; Paulheim, H. Entity Matching on Web Tables: A Table Embeddings approach for Blocking. In Proceedings of the 20th International Conference on Extending Database Technology, EDBT, Venice, Italy, 21–24 March 2017; pp. 510–513.
2. Ma, K.; Yang, B. Stream-based live entity resolution approach with adaptive duplicate count strategy. *Int. J. Web Grid Serv.* **2017**, *13*, 351–373. [CrossRef]
3. Chen, L.; Gu, W.; Tian, X.; Chen, G. AHAB: Aligning heterogeneous knowledge bases via iterative blocking. *Inf. Process. Manag.* **2019**, *56*, 1–13. [CrossRef]
4. Liu, X.L.; Wang, H.Z.; Li, J.Z.; Gao, H. EntityManager: Managing dirty data based on entity resolution. *J. Comput. Sci. Technol.* **2017**, *32*, 644–662. [CrossRef]
5. Christophides, V.; Efthymiou, V.; Stefanidis, K. Entity Resolution in the Web of Data. *Synth. Lect. Semant. Web* **2015**, *5*, 1–122.
6. Ayat, N.; Akbarinia, R.; Afsarmanesh, H.; Valduriez, P. Entity resolution for probabilistic data. *Inf. Sci.* **2014**, *277*, 492–511. [CrossRef]
7. Christen, P. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*; Springer Science & Business Media: Berlin, Germany, 2012.
8. Papadakis, G.; Alexiou, G.; Papastefanatos, G.; Koutrika, G. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proc. VLDB Endow.* **2015**, *9*, 312–323. [CrossRef]
9. do Nascimento, D.C.; Pires, C.E.S.; Mestre, D.G. Heuristic-based approaches for speeding up incremental record linkage. *J. Syst. Softw.* **2018**, *137*, 335–354. [CrossRef]
10. Ren, X.; Curé, O. Strider: A hybrid adaptive distributed RDF stream processing engine. In *Proceedings of the International Semantic Web Conference*; Springer: Cham, Switzerland, 2017; pp. 559–576.
11. Simonini, G.; Gagliardelli, L.; Bergamaschi, S.; Jagadish, H. Scaling entity resolution: A loosely schema-aware approach. *Inf. Syst.* **2019**, *83*, 145–165. [CrossRef]
12. Dragoni, M.; Federici, M.; Rexha, A. An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Inf. Process. Manag.* **2019**, *56*, 1103–1118. [CrossRef]
13. Li, B.H.; Liu, Y.; Zhang, A.M.; Wang, W.H.; Wan, S. A Survey on Blocking Technology of Entity Resolution. *J. Comput. Sci. Technol.* **2020**, *35*, 769–793. [CrossRef]
14. Santana, A.F.; Gonçalves, M.A.; Laender, A.H.; Ferreira, A.A. Incremental author name disambiguation by exploiting domain-specific heuristics. *J. Assoc. Inf. Sci. Technol.* **2017**, *68*, 931–945. [CrossRef]
15. Christophides, V.; Efthymiou, V.; Palpanas, T.; Papadakis, G.; Stefanidis, K. End-to-End Entity Resolution for Big Data: A Survey. *arXiv* **2019**, arXiv:1905.06397.
16. Araújo, T.B.; Pires, C.E.S.; da Nóbrega, T.P. Spark-based Streamlined Metablocking. In Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, Greece, 3–6 July 2017.
17. Agarwal, S.; Godbole, S.; Punjani, D.; Roy, S. How much noise is too much: A study in automatic text classification. In Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM 2007), Omaha, NE, USA, 28–31 October 2007; pp. 3–12.
18. García, S.; Luengo, J.; Herrera, F. *Data Preprocessing in Data Mining*; Springer: Boston, MA, USA, 2015.
19. Araújo, T.B.; Stefanidis, K.; Santos Pires, C.E.; Nummenmaa, J.; da Nóbrega, T.P. Schema-Agnostic Blocking for Streaming Data. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, SAC '20, Brno, Czech Republic, 30 March–3 April 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 412–419. [CrossRef]
20. Simonini, G.; Bergamaschi, S.; Jagadish, H. BLAST: A loosely schema-aware meta-blocking approach for entity resolution. *Proc. VLDB Endow.* **2016**, *9*, 1173–1184. [CrossRef]
21. Papadakis, G.; Koutrika, G.; Palpanas, T.; Nejdl, W. Meta-blocking: Taking entity resolutionto the next level. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 1946–1960. [CrossRef]
22. Efthymiou, V.; Papadakis, G.; Papastefanatos, G.; Stefanidis, K.; Palpanas, T. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Inf. Syst.* **2017**, *65*, 137–157. [CrossRef]
23. Efthymiou, V.; Stefanidis, K.; Christophides, V. Benchmarking Blocking Algorithms for Web Entities. *IEEE Trans. Big Data* **2020**, *6*, 382–395. [CrossRef]
24. Efthymiou, V.; Stefanidis, K.; Christophides, V. Big data entity resolution: From highly to somehow similar entity descriptions in the Web. In Proceedings of the 2015 IEEE International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, 29 October 29–1 November 2015; pp. 401–410.
25. Burys, J.; Awan, A.J.; Heinis, T. Large-Scale Clustering Using MPI-Based Canopy. In Proceedings of the 2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC), Dallas, TX, USA, 12 November 2018; pp. 77–84.
26. Dai, W.; Yu, C.; Jiang, Z. An improved hybrid Canopy-Fuzzy C-means clustering algorithm based on MapReduce model. *J. Comput. Sci. Eng.* **2016**, *10*, 1–8. [CrossRef]

27. Kolb, L.; Thor, A.; Rahm, E. Multi-pass sorted neighborhood blocking with MapReduce. *Comput.-Sci.-Res. Dev.* **2012**, *27*, 45–63. [CrossRef]
28. Ramadan, B.; Christen, P.; Liang, H.; Gayler, R.W. Dynamic sorted neighborhood indexing for real-time entity resolution. *J. Data Inf. Qual. (JDIQ)* **2015**, *6*, 15. [CrossRef]
29. Mestre, D.G.; Pires, C.E.S.; Nascimento, D.C.; de Queiroz, A.R.M.; Santos, V.B.; Araujo, T.B. An efficient spark-based adaptive windowing for entity matching. *J. Syst. Softw.* **2017**, *128*, 1–10. [CrossRef]
30. Ma, Y.; Tran, T. Typimatch: Type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, Rome, Italy, 4–8 February 2013; pp. 325–334.
31. Papadakis, G.; Ioannou, E.; Palpanas, T.; Niederee, C.; Nejdl, W. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 2665–2682. [CrossRef]
32. Araújo, T.B.; Pires, C.E.S.; Mestre, D.G.; Nóbrega, T.P.d.; Nascimento, D.C.d.; Stefanidis, K. A noise tolerant and schema-agnostic blocking technique for entity resolution. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, 8–12 April 2019; pp. 422–430.
33. Papadakis, G.; Papastefanatos, G.; Palpanas, T.; Koubarakis, M. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In Proceedings of the 19th International Conference on Extending Database Technology (EDBT), Bordeaux, France, 15–18 March 2016.
34. Yang, Y.; Sun, Y.; Tang, J.; Ma, B.; Li, J. Entity matching across heterogeneous sources. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, 10–13 August 2015; pp. 1395–1404.
35. Gagliardelli, L.; Simonini, G.; Beneventano, D.; Bergamaschi, S. SparkER: Scaling Entity Resolution in Spark. In Proceedings of the Advances in Database Technology—22nd International Conference on Extending Database Technology, EDBT, Lisbon, Portugal, 26–29 March 2019; pp. 602–605.
36. Liang, H.; Wang, Y.; Christen, P.; Gayler, R. Noise-tolerant approximate blocking for dynamic real-time entity resolution. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Tainan, Taiwan, 13–16 May 2014; pp. 449–460.
37. Lai, K.H.; Topaz, M.; Goss, F.R.; Zhou, L. Automated misspelling detection and correction in clinical free-text records. *J. Biomed. Inform.* **2015**, *55*, 188–195. [CrossRef]
38. Sohail, A.; Qounain, W.u. Locality sensitive blocking (LSB): A robust blocking technique for data deduplication. *J. Inf. Sci.* **2022**. [CrossRef]
39. Caruccio, L.; Deufemia, V.; Naumann, F.; Polese, G. Discovering relaxed functional dependencies based on multi-attribute dominance. *IEEE Trans. Knowl. Data Eng.* **2020**, *33*, 3212–3228. [CrossRef]
40. Caruccio, L.; Cirillo, S. Incremental discovery of imprecise functional dependencies. *J. Data Inf. Qual. (JDIQ)* **2020**, *12*, 1–25. [CrossRef]
41. Singh, R.; Meduri, V.V.; Elmagarmid, A.; Madden, S.; Papotti, P.; Quiané-Ruiz, J.A.; Solar-Lezama, A.; Tang, N. Synthesizing entity matching rules by examples. *Proc. VLDB Endow.* **2017**, *11*, 189–202. [CrossRef]
42. Koumarelas, I.; Papenbrock, T.; Naumann, F. MDedup: Duplicate detection with matching dependencies. *Proc. VLDB Endow.* **2020**, *13*, 712–725. [CrossRef]
43. Gruenheid, A.; Dong, X.L.; Srivastava, D. Incremental record linkage. *Proc. VLDB Endow.* **2014**, *7*, 697–708. [CrossRef]
44. Nentwig, M.; Rahm, E. Incremental clustering on linked data. In Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW), Singapore, 17–20 November 2018; pp. 531–538.
45. Saeedi, A.; Peukert, E.; Rahm, E. Incremental Multi-source Entity Resolution for Knowledge Graph Completion. In *Proceedings of the European Semantic Web Conference*; Springer: Cham, Switzerland, 2020; pp. 393–408.
46. Ju, W.; Li, J.; Yu, W.; Zhang, R. iGraph: An incremental data processing system for dynamic graph. *Front. Comput. Sci.* **2016**, *10*, 462–476. [CrossRef]
47. Körber, M.; Glombiewski, N.; Morgen, A.; Seeger, B. TPStream: Low-latency and high-throughput temporal pattern matching on event streams. *Distrib. Parallel Databases* **2019**, *39*, 361–412. [CrossRef]
48. Opitz, B.; Sztyler, T.; Jess, M.; Knip, F.; Bikar, C.; Pfister, B.; Scherp, A. An Approach for Incremental Entity Resolution at the Example of Social Media Data. In Proceedings of the AI Mashup Challenge 2014 co-located with 11th Extended Semantic Web Conference (ESWC 2014), Crete, Greece, 27 May 2014.
49. Ao, F.; Yan, Y.; Huang, J.; Huang, K. Mining maximal frequent itemsets in data streams based on fp-tree. In Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition, Leipzig, Germany, 18–20 July 2007; pp. 479–489.
50. Kumar, A.; Singh, A.; Singh, R. An efficient hybrid-clustream algorithm for stream mining. In Proceedings of the 2017 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), Jaipur, India, 4–7 December 2017; pp. 430–437.
51. Kim, T.; Hwang, M.N.; Kim, Y.M.; Jeong, D.H. Entity Resolution Approach of Data Stream Management Systems. *Wirel. Pers. Commun.* **2016**, *91*, 1621–1634. [CrossRef]
52. Dong, X.L.; Srivastava, D. Big data integration. *Synth. Lect. Data Manag.* **2015**, *7*, 1–198.

53. Garofalakis, M.; Gehrke, J.; Rastogi, R. *Data Stream Management: Processing High-Speed Data Streams*; Springer: Cham, Switzerland, 2016.
54. Zikopoulos, P.; Eaton, C.; Zikopoulos, P. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*; McGraw-Hill Osborne Media: New York , NY, USA, 2011.
55. Kreps, J.; Narkhede, N.; Rao, J. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB, Athens, Greece, 12 June 2011; Volume 11, pp. 1–7.
56. Andoni, A.; Indyk, P.; Nguyen, H.L.; Razenshteyn, I. Beyond locality-sensitive hashing. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Portland, OR, USA, 5–7 January 2014; pp. 1018–1028.
57. Efthymiou, V.; Papadakis, G.; Stefanidis, K.; Christophides, V. MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities. In Proceedings of the Advances in Database Technology—22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, 26–29 March 2019; pp. 373–384. [CrossRef]
58. Efthymiou, V.; Stefanidis, K.; Christophides, V. Minoan ER: Progressive Entity Resolution in the Web of Data. In Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, 15–16 March 2016; pp. 670–671.
59. Golab, L.; Özsu, M.T. Processing sliding window multi-joins in continuous queries over data streams. In Proceedings of the Proceedings 2003 VLDB Conference, Berlin, Germany, 9–12 September 2003; pp. 500–511.
60. Singh, P.K.; Cherukuri, A.K.; Li, J. Concepts reduction in formal concept analysis with fuzzy setting using Shannon entropy. *Int. J. Mach. Learn. Cybern.* **2017**, *8*, 179–189. [CrossRef]
61. Dasgupta, A.; Kumar, R.; Sarlós, T. Fast locality-sensitive hashing. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 21–24 August 2011; pp. 1073–1081.
62. Carbone, P.; Katsifodimos, A.; Ewen, S.; Markl, V.; Haridi, S.; Tzoumas, K. Apache flink: Stream and batch processing in a single engine. *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.* **2015**, *36*, 28–38.
63. Lopez, M.A.; Lobato, A.G.P.; Duarte, O.C.M. A performance comparison of open-source stream processing platforms. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; pp. 1–6.
64. Veiga, J.; Expósito, R.R.; Pardo, X.C.; Taboada, G.L.; Tourifio, J. Performance evaluation of big data frameworks for large-scale data analytics. In Proceedings of the 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 5–8 December 2016; pp. 424–431.
65. Stefanidis, K.; Christophides, V.; Efthymiou, V. Web-scale blocking, iterative and progressive entity resolution. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 1459–1462.
66. Hassani, M. Overview of Efficient Clustering Methods for High-Dimensional Big Data Streams. In *Clustering Methods for Big Data Analytics*; Springer: Cham, Switzerland, 2019; pp. 25–42.
67. Fedoryszak, M.; Frederick, B.; Rajaram, V.; Zhong, C. Real-time Event Detection on Social Data Streams. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2774–2782.
68. Global Social Journalism Study. Available online: https://www.cision.com/us/resources/white-papers/2019-sotm/?sf=false (accessed on 7 January 2017).
69. Hasan, M.; Orgun, M.A.; Schwitter, R. TwitterNews: Real time event detection from the Twitter data stream. *PeerJ Prepr.* **2016**, *4*, e2297v1.
70. Von Nordheim, G.; Boczek, K.; Koppers, L. Sourcing the Sources: An analysis of the use of Twitter and Facebook as a journalistic source over 10 years in The New York Times, The Guardian, and Süddeutsche Zeitung. *Digit. J.* **2018**, *6*, 807–828. [CrossRef]
71. Idrissou, A.; Van Harmelen, F.; Van Den Besselaar, P. Network metrics for assessing the quality of entity resolution between multiple datasets. *Semant. Web* **2021**, *12*, 21–40. [CrossRef]
72. Hasan, M.; Orgun, M.A.; Schwitter, R. A survey on real-time event detection from the twitter data stream. *J. Inf. Sci.* **2018**, *44*, 443–463. [CrossRef]
73. Campos, R.; Mangaravite, V.; Pasquali, A.; Jorge, A.; Nunes, C.; Jatowt, A. YAKE! Keyword extraction from single documents using multiple local features. *Inf. Sci.* **2020**, *509*, 257–289. [CrossRef]
74. Lee, P.; Lakshmanan, L.V.; Milios, E.E. Incremental cluster evolution tracking from highly dynamic network data. In Proceedings of the 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, 31 March 31–4 April 2014; pp. 3–14.
75. Sharma, S. Dynamic hashtag interactions and recommendations: An implementation using apache spark streaming and GraphX. In *Data Management, Analytics and Innovation*; Springer: Cham, Switzerland, 2020; pp. 723–738.
76. Altowim, Y.; Mehrotra, S. Parallel progressive approach to entity resolution using mapreduce. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 909–920.
77. Huang, C.; Zhu, J.; Huang, X.; Yang, M.; Fung, G.; Hu, Q. A novel approach for entity resolution in scientific documents using context graphs. *Inf. Sci.* **2018**, *432*, 431–441. [CrossRef]
78. Singh, S.; Chana, I. A survey on resource scheduling in cloud computing: Issues and challenges. *J. Grid Comput.* **2016**, *14*, 217–264.
79. Tsai, J.T.; Fang, J.C.; Chou, J.H. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. *Comput. Oper. Res.* **2013**, *40*, 3045–3055. [CrossRef]