Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Tiago Brasileiro Araújo

Parallel Blocking for Entity Resolution in the Context of Semi-structured Data

Campina Grande – PB

2020

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Parallel Blocking for Entity Resolution in the Context of Semi-structured Data

Tiago Brasileiro Araújo

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Carlos Eduardo Santos Pires
(Orientador)

Campina Grande, Paraíba, Brasil

A663p Araújo, Tiago Brasileiro.
Parallel blocking for entity resolution in the context of semi-structured data / Tiago Brasileiro Araújo. – Campina Grande, 2020.
147 f. : il. color.

Tese (Doutorado em Ciências da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.
"Orientação: Prof. Dr. Carlos Eduardo Santos Pires".
Referências.

1. Banco de Dados. 2. Sistemas de Informação. 3. Resolução de Entidades. 4. Dados Semiestruturados. 5. Técnicas Agnósticas de Blocagem. 6. Computação Distribuída. 7. Streaming de Dados. 8. Processamento Incremental. I. Pires, Carlos Eduardo Santos. II. Título.

CDU 004.65 (043)
FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

PARALLEL BLOCKING FOR ENTITY RESOLUTION IN THE CONTEXT OF SEMI-STRUCTURED DATA

TIAGO BRASILEIRO ARAÚJO

TESE APROVADA EM 19/02/2020

CARLOS EDUARDO SANTOS PIRES, Dr., UFCG
Orientador(a)

CLÁUDIO DE SOUZA BAPTISTA, PhD., UFCG
Examinador(a)

CLÁUDIO ELÍZIO CALAZANS CAMPELO, PhD., UFCG
Examinador(a)

JOSÉ MARIA DA SILVA MONTEIRO FILHO, Dr., UFC
Examinador(a)

BERNADETTE FARIAS LÓSCIO, Dra, UFPE
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A tarefa de Resolução de Entidades (RE) surge como um passo fundamental para integrar múltiplas bases de conhecimento ou identificar semelhanças entre os dados (entidades). Para evitar o custo quadrático da tarefa de Resolução de Entidades, técnicas de blocagem (ou indexação) são amplamente aplicadas como uma etapa de pré-processamento. Neste contexto, dados semiestruturados e grandes fontes de dados (*Big Data*) emergem como os principais desafios enfrentados pelas técnicas de blocagem. No que diz respeito aos dados semiestruturados, o desafio está relacionado ao fato de que tais dados não compartilham um mesmo esquema, inviabilizando a aplicação de técnicas tradicionais de blocagem. Neste contexto, técnicas agnósticas de blocagem são aplicadas. Em relação às grandes fontes de dados (*Big Data)*, técnicas de blocagem e computação distribuída devem ser aplicadas com o intuito de melhorar a eficiência da tarefa de RE. Nesse sentido, este trabalho propõe um modelo de execução distribuída para blocagem de dados semiestruturados no contexto grandes fontes de dados, capaz de atender às diferentes necessidades dos perfis de aplicação enfrentados pela tarefa de RE. Esses perfis de aplicação estão relacionados às necessidades e características inerentes a cada aplicação, tais como a forma como os dados são recebidos (i.e., em lotes ou *streaming*), qualidade dos dados e priorização da eficácia/eficiência da blocagem. Além disso, o presente trabalho também propõe novas técnicas de blocagem que podem ser acopladas ao modelo proposto. Tais técnicas de blocagem endereçam desafios em aberto segundo a literatura, como técnicas agnósticas de blocagem em paralelo, processamento incremental e blocagem de dados em *streaming*. As técnicas de blocagem foram avaliadas experimentalmente com o objetivo de mensurar a eficiência e eficácia em relação às técnicas de blocagem do estado da arte, utilizando fontes de dados reais. Com base nos resultados experimentais, é possível destacar que as novas técnicas de blocagem apresentaram resultados mais promissores, podendo ser acopladas ao modelo de execução distribuída proposto, de maneira a atender as diferentes necessidades inerentes aos perfis de aplicação.

Palavras-chave: Resolução de Entidades, Dados Semiestruturados, Técnicas Agnósticas de Blocagem, Computação Distribuída, *Streaming* de Dados, Processamento Incremental.

Abstract

The Entity Resolution (ER) task emerges as a fundamental step to integrate multiple knowledge bases or identify similarities between data (entities). To avoid the quadratic cost of the Entity Resolution task, blocking (or indexing) techniques are widely applied as a preprocessing step. In this context, semistructured data and large data sources (Big Data) emerge as the major challenges faced by blocking techniques. Regarding semistructured data, the challenge is related to the fact that such data do not share the same scheme, difficulting the application of traditional blocking techniques. In this context, schema-agnostic blocking techniques are applied. For Big Data scenarios, blocking techniques and distributed computing should be applied to improve the efficiency of the RE task. In this sense, this work proposes a distributed execution model for blocking semistructured data in the context of large data sources, capable of dealing with different needs of application profiles faced by the ER task. These application profiles are related to the needs and characteristics inherent to each application, such as how the data are managed (i.e., batch or streaming), data quality and prioritization of effectiveness/efficiency. Furthermore, the present work also proposes new blocking techniques that can be integrated into the proposed model. Such blocking techniques address open challenges in the literature, such as parallel blocking techniques, incremental processing, and streaming data blocking. The blocking techniques proposed in this work were evaluated experimentally with the objective of measuring efficiency and effectiveness against the state-of-the-art ones, using real data sources. Based on the experimental results, it is possible to highlight that the novel blocking techniques presented better results when compared to the state-of-the-art blocking techniques. Therefore, the proposed techniques can be hosted to the proposed execution model, so that they can address different necessities inherent to the application profiles.

Keywords: Entity Resolution, Semistructured Data, Schema-agnostic Blocking Techniques, Distributed Computing, Streaming Data, Incremental Processing.

Agradecimentos

Quero agradecer, em primeiro lugar, a Deus por sua infinita misericórdia e fidelidade nos momentos de dificuldade. Agradeço ao Senhor por tantas bênçãos alcançadas e pelos dons do Espírito, que foram fundamentais até aqui. Abaixo, os demais agradecimentos:

- à família por sempre estar ao meu lado, especialmente minha mãe que em meio às dificuldades sempre prezou pela minha educação;
- a Polyanna, minha namorada, por todo cuidado e carinho nessa etapa da minha vida;
- ao meu orientador Professor Dr. Carlos Eduardo Pires pela imensa dedicação e presteza durante todas as etapas do doutorado;
- ao co-orientador Professor Dr. Kostas Stefanidis pela dedicação e por promover a experiência de vivenciar um doutorado sanduíche na Finlândia;
- aos meus amigos e membros do laboratório, que sempre vieram em meu auxílio nos momentos de comemoração e de dificuldades;
- aos professores e funcionários da COPIN;
- à Universidade Federal de Campina Grande e Tampere University.

Contents

List of Symbols

BLAST - *Blocking with Loosely-Aware Schema Techniques*

CEP - *Cardinality Edge Pruning*

CNP - *Cardinality Node Pruning*

CSV - *Comma-Separated Values*

DFS - *Distributed File System*

ER - *Entity Resolution*

GIS - *Geographic Information System*

GWNP - *Global Weighted Node Pruning*

JSON - *Java Script Object Notation*

LSH - *Locality Sensitive Hashing*

MR - *MapReduce*

NA-BLOCKER - *Noise-aware Agnostic BLOCKing for Entity Resolution*

PRIME - *PaRallel-based Incremental MEtablocking*

RDF - *Resource Description Framework*

SS-Metablocking - *Spark-based Streamlined Metablocking*

URI - *Uniform Resource Identifier*

WEP - *Weighted Edge Pruning*

WNP - *Weighted Node Pruning*

XML - *eXtensible Markup Language*

List of Figures

List of Tables

Chapter 1

Introduction

Currently, there is an increasing number of information systems producing a large amount of data continuously, such as Web systems (e.g., digital libraries and e-commerce), Social Media (e.g., Twitter and Facebook) and Internet of Things (e.g., mobiles, sensors and devices) [45; 50]. These applications have become a valuable source of heterogeneous data [64; 19]. Such kind of data presents a schema-free behaviour and can be represented in different formats (e.g., XML, RDF and JSON). Commonly, the data is provided by different data sources and may have overlapping knowledge. For instance, different social media will report the same event and generate mass similar data [29]. The widespread generation of data (resulting in large data sources, i.e., Big Data) and data that are constantly sent to several data sources (i.e., streaming data), academia and industry focus on searching innovative techniques that enable efficient data processing, analysis, management, and integration. In this sense, several data matching tasks have been proposed in order to integrate or identify similar data in multiple data sources. Among them, it is possible to highlight the task of Entity Resolution (ER), due to its importance and application in various areas of knowledge, such as health, safety, academic, smart cities, social media, Web, and spatial data [47].

A fundamental step to integrate multiple knowledge bases or identify similarities between entities is Entity Resolution (ER). ER (also known as entity reconciliation, entity matching, record linkage or deduplication) is a task that matches records (the entity profiles) from several data sources (the entity collections) which refer to the same real-world entity [21]. The ER task is widely used by the Web community because it is commonly necessary to integrate multiple knowledge bases, which store semi-structured data [71;

94]. Since ER is able to identify duplicate data or determine relationships between entities, this task has gained detach in supporting and providing useful information to various areas of knowledge, such as Data Quality, Data Integration and Data Mining [25]. More specifically, ER has been used in numerous applications: Web page deduplication, crime prevention (e.g., terrorist identification), plagiarism and fraud detection [21]. However, traditional ER has a quadratic behaviour since it requires that each entity in one data source be compared with all entities in the other data source (i.e., Cartesian product) [21]. When ER is guided by the Cartesian product, the execution time can achieve an asymptotic complexity of $O(n^2)$. This computational cost is inefficient when placed in the context of Big Data, where data sources usually contain thousands or millions of entities [21].

Regarding the different ways to process data, there are two main types of processing: batch or streaming. Batch processing is characterised by receiving and processing the entire input (i.e., set of entities) at once [69]. On the other hand, in streaming data processing, input data is received continuously at short intervals. This type of processing is commonly related to dynamic data sources that continually update. Here, we assume that not all data, from all data sources, are available at once [64]. Therefore, we have to match the entities as they appear, also considering the entities already matched previously. In this sense, the challenges are potentiated when the ER task is inserted into the context of Big Data and Streaming Data simultaneously, as the task has to deal with a huge amount of data and the continuous flow of data received in a short time (for example, seconds or minutes) [84].

In the Big Data context, the ER task faces mainly three "Vs": *volume*, as it deals with a large number of entities; *velocity*, related to the speed at which entities should be compared; and *variety*, since different formats describe entity profiles (heterogeneous data) [37]. In this sense, blocking techniques and distributed computing are applied in order to minimize problems related to the large and continuous volume of data to be processed [31]. Blocking techniques group similar entities into blocks and perform comparisons within each block, avoiding comparisons guided by the Cartesian product. Even though blocking techniques reduce the number of comparisons to be performed during the ER task, such techniques can still result in a high amount of comparisons [37]. It occurs due to the size of the input datasets and/or the distribution of entities within blocks. In some cases, the smallest block generated by a blocking technique can still result in a large number of comparisons.

When blocking techniques are not applied or their application is not sufficient to reduce the number of comparisons satisfactorily, the ER task tends to require hours or days to be performed. However, nowadays, it is not reasonable that the ER task runs for several hours or days [23]. Regarding the velocity of execution, the ER task can be performed in parallel to reduce the overall execution time by distributing entity comparisons among the various resources (for instance, computers or virtual machines) of a distributed infrastructure. In this context, the application of MapReduce (MR) based programmatic models (such as Spark, Flink and Hadoop) emerges as a possible solution for the execution of the ER task in parallel. MR models were developed in order to simplify the execution of parallel tasks over distributed infrastructures. Common characteristics of these frameworks, such as simplicity (in terms of usability), flexibility, fault tolerance, and the ability to scale high-cost computational processing, able these frameworks become excellent resources for performing data-intensive tasks efficiently (regarding time processing). In this work, Spark is applied for batch data scenarios while Flink is applied for streaming data scenarios since relevant works prove their efficiency (in terms of processing and scalability) for these scenarios [90; 64; 85].

Regarding variety, the heterogeneity of the data compromises the generation of entity blocks (by the blocking techniques) since the entity profiles hardly share the same schema. For this reason, in the context of heterogeneous data, it is not possible to define blocking keys (keys used to partition entities in blocks) based on the schema. Therefore, traditional blocking techniques do not present satisfactory effectiveness, for instance, Sorted Neighborhood [57] and Adaptive Window [69]). It occurs due to these techniques perform the blocking based on the entity profile schema [83; 80]. In turn, the variety challenge is addressed by schema-agnostic blocking techniques, which disregard attribute names and consider the values related to the entity attributes to perform blocking [21; 25]. Furthermore, it is possible to highlight two new problems tackled by the ER task: streaming data and incremental processing [30; 85]. Streaming data is related to dynamic data sources (e.g., from Web Systems, Social Media and sensors), which are continuously updated. When ER receives streaming data, we assume that not all data (from the data sources) are available at once. Therefore, ER needs to match the entities as they arrive, also considering the entities already matched previously. On the other hand, incremental

ER is related to receiving data continuously over time and re-processing only the portion of the matching results (i.e., similar entities) that were affected by the data increments. Thus, the challenges are strengthened when the ER task is considered in the context of Big Data, streaming data and incremental processing, simultaneously.

As previously discussed, the ER task in the context of Big Data presents challenges (related to the three "Vs", streaming data and incremental processing) which motivate the application of parallel computing. For instance, the application of frameworks such as Spark or Flink, which provide data streaming libraries and allow processing large amounts of data efficiently. Beyond these challenges, the ER task faces different scenarios associated with the domain (e.g., health, Web, academic, smart cities) in which the task is inserted, such as: how data is handled (batch or streaming); data quality (presence or absence of noise in the data, such as typos and misspelling); and prioritization of effectiveness or efficiency (according to application needs). Throughout this work, such scenarios (application associated) will be referred to as application profiles. In the execution model proposed in this work, the user (for example, a researcher or data specialist) will inform the execution model the application profile, according to the application characteristics (batch/streaming, data with/without noise, and focus on effectiveness/efficiency) identified by him/her.

1.1 Relevance

There is an increasing number of areas where ER application is needed, such as healthcare, national censuses, national security, business systems and, more recently, digital libraries, e-commerce and social networks. At the academy, ER has been applied in different areas of computing such as Data Mining, Machine Learning, Information Retrieval, Databases, and Data Quality [39]. The main challenge of ER is the lack of global identifiers (e.g., primary keys) of entities to be compared. For this reason, the ER task must explore attributes that contain partial identifying information for entities, such as name, description, address, or venue. However, the identifying information is often of poor quality. For instance, personal information (phone, marital status, and address) especially suffers from frequent variations and typographical errors [21].

Although the ER task has been studied over several decades and has achieved several advances (e.g., blocking techniques), the emergence of new challenges inherent in new ways of data manipulation and storage (such as Big Data, data streaming, and semi-structured data) made this task relevant until nowadays [25]. In this context, the search for approaches that enhance the efficiency of the ER task has been highlighted in recent years, since it is not interesting that this task requires a long time to complete [80; 83; 37].

In the context of information systems, there is a growing need to integrate different data sources in order to increase the amount of information available or supplement existing information in one of the data sources [25; 80; 21]. Therefore, as stated previously, the ER task can be applied to support data integration tasks. Since nowadays the data sources commonly store a large amount of data (i.e., Big Data) and the data is semi-structured, agnostic blocking techniques should be applied in order to improve the efficiency of ER. For this reason, this work aims to investigate challenges related to blocking techniques for ER in the context of large semi-structured data sources.

The relevance of this work is due to the proposition of a distributed execution model for agnostic blocking techniques, capable of addressing the needs of different application profiles. These application profiles are related to the characteristics of each application, such as how data is received, data quality and prioritization of effectiveness/efficiency. Based on the application profile informed by the user, the model will define the execution flow and the blocking technique to be applied according to the needs of the application profile in question. Furthermore, novel agnostic blocking techniques are proposed, which are coupled to the execution model. The proposed blocking techniques also enable the efficient execution of ER involving data sources containing thousands or millions of entities, by applying distributed computing. Moreover, the execution model can be extended and serve as a basis for proposing other models or blocking techniques in future scientific investigations.

1.2 Objetives

In this section, the main objective and specific objectives of this work are presented.

1.2.1 Main Objetive

Based on the challenges related to the context (semi-structured data, large data sources, and batch/streaming data) in which this work is inserted, the scope of the thesis is related to improving the efficiency of ER task without reducing the effectiveness of the results.

Therefore, the main objective of this work is to propose efficient blocking techniques in the context of semi-structured data, large data sources, and batch/streaming data. In this sense, parallel blocking techniques are investigated, proposed and evaluated in order to be integrated with the distributed execution model also proposed in this thesis, which aims to improve the efficiency of ER task in data integration scenarios involving semi-structured data.

Thus, the main hypothesis of the work is to evaluate if the application of the proposed blocking techniques is able to considerably improve the efficiency of the ER task without decreasing effectiveness, in scenarios where the ER task faces the needs associated with the application profiles (i.e., batch/streaming, data with/without noise, and focus on effectiveness/efficiency).

1.2.2 Specific Objectives

Considering the proposed main objective, this work has the following specific objectives:

1. Propose a distributed execution model, whose steps will be used to guide the workflow of the blocking techniques;

2. Propose and evaluate new efficient blocking techniques for parallel ER tasks, taking into account each challenge stated in the scope of this work (semi-structured data, large data sources and batch/streaming data);

3. Propose strategies, based on the execution model and application profile needs (informed by the user), to select the schema-agnostic blocking technique and the best execution workflow of the model (among the predefined workflows) related to the application profile in question;

4. Propose a distributed architecture to describe the execution model steps and the execution workflow for blocking;

5. Implement and evaluate blocking techniques based on the execution model so that they are capable of handling large semi-structured data sources in the context of batch/streaming data.

1.3 Methodology

In order to propose solutions for problems and challenges regarding ER task, in this work, we apply a methodology that divides the doctoral research into smaller research projects. We developed three research projects which resulted in three main blocking techniques proposed in this thesis, as described in Sections 5, 6 and 7. For each research project, a prior planning is performed to specify a detailed description and a schedule for the following activities: i) theoretical foundation and relevance justification of the problem or area to be investigated; ii) reading books, articles and technical reports related to the investigated problem; iii) proposing new approaches and/or methodologies to address the challenges of ER; iv) definition of hypotheses and planning of an experimental design to evaluate the research hypotheses investigated; v) formatting and promoting a discussion of the results achieved in previous activities; vi) listing of interpretations and conclusions as a result of the research conducted; and vii) definition of future works.

During the studies conducted for the first research project (see Section 5), we investigated several blocking techniques [25; 80; 83; 36; 37; 89]. In this sense, we noticed that these techniques commonly follow a set of steps. Thus, a distributed execution model for parallel blocking techniques emerges as a possible generic structure to guide the development of our techniques. In this thesis, the concept of model is considered as a set of operations (i.e. steps) to be used in a process [14]. Then, the proposed model defines a set of steps to perform the blocking task and possible workflows to perform these steps (which will be described in Section 4). Since aspects related to efficiency and the processing of large data sources are explored in this thesis, the proposed execution model is based on a distributed infrastructure to perform the blocking task in parallel. Thus, the three blocking techniques proposed in this thesis follow the steps and the workflows defined in the distributed execution model.

In this work, parallel schema-agnostic blocking techniques are proposed assuming the context addressed by this work. These blocking techniques use Spark and Flink as a programmatic model to parallelize the entity blocking. During the four years of the Ph.D., the research efforts were focused on four main objectives:

1. Define the problem to be addressed by our research;
2. Propose efficient schema-agnostic blocking techniques;
3. Develop parallel workflows to be followed by the proposed blocking techniques;
4. Explore the open problem related to handling streaming data (by proposing an incremental schema-agnostic blocking technique for streaming data).

More specifically, we accomplished the following objectives:

1. Investigate solutions for RE in the context of semi-structured data;
2. Propose an efficient parallel-based blocking technique in the context of semi-structured data;
3. Propose a noise-tolerant (able to tolerate typos and spelling errors on the data) blocking technique in semi-structured data;
4. Propose a parallel-based blocking technique able to handle streaming data;
5. Conduct a case study involving a real scenario of streaming data, where the proposed blocking techniques is applied.

Summary, we can highlight the three blocking technique proposed in this thesis. Since the application profile explores aspects related to how data is handled (batch or streaming); data quality (presence or absence of noise in the data) and prioritization of effectiveness or efficiency, the proposed techniques aim to address these aspects. Hence, the first technique (see Section 5) addresses batch data, without noise and prioritizes the efficiency of the blocking task. The second technique (see Section 6) aims to generate high-quality blocks (i.e., prioritizes the effectiveness) in the context of noise data in batch. The third technique (see Section 7) innovates in the sense of handle streaming data, with a focus on generating high-quality blocks efficiently. However, the third technique does not consider noise on the data. Although the proposed model can explore different combinations of application profiles (more specifically, eight combinations), we explore three combinations addressed by each proposed technique. However, it is important to highlight the generic behaviour of the execution model. Thus, other blocking techniques, proposed by us or other researchers, can be integrated into the execution model.

1.4 Main Results

Since the main purpose of this thesis is to propose a distributed execution model for parallel blocking techniques, the efforts are related to the proposition and implementation of efficient blocking techniques. Thus, the following results were achieved:

1. A blocking technique in the context of semi-structured data (i.e., schema-agnostic technique) in parallel, using the Spark programmatic model;

2. A noise-tolerant blocking technique (e.g., typos and spelling errors) in the context of semi-structured data. This technique seeks improvements in block generation, which may favor other blocking techniques, as proposed in item 1;

3. An incremental schema-agnostic blocking technique for streaming data. This technique is able to process a large amount of streaming data in an incremental way through a distributed computational infrastructure;

4. A case study involving a real scenario of streaming data. In this sense, data provided by Twitter was explored in order to validate the application of the proposed blocking techniques.

In addition, experimental evaluations were performed to evaluate the effectiveness and efficiency of the blocking techniques proposed in this work. Based on the achieved results, it is possible to detach that the novel blocking techniques present efficient solutions, regarding the execution time, and maintain (or outperform) the efficacy results achieved by state-of-the-art techniques.

1.5 Achieved Research Indicators

In summary, the following research indicators have been achieved:

1. Publication and presentation of the article *Spark-based Streamlined Metablocking* [8] in the *22nd IEEE Symposium on Computers and Communications* (2017) in Heraklion - Greece;

2. Publication and presentation of the article *Towards Reliable Data Analyzes for Smart Cities* [7] in the *21st International Database Engineering & Applications Symposium* (2017) in Bristol - England;

3. Publication of the article *Noisy-aware Blocking over Heterogeneous Data* [6] in the *17th International Semantic Web Conference* (2018) in Monterey - USA;

4. Publication and presentation of the article *A Noise Tolerant and Schema-agnostic Blocking Technique for Entity Resolution* [10] in the *34th ACM/SIGAPP Symposium on Applied Computing* (2019), in Limassol - Cyprus;

5. Presentation of the poster *A Parallel Approach for Data Matching over Streaming Web Data* in the *13rd Alberto Mendelzon International Workshop* (2019) in Asunción - Paraguay;

6. Publication of the article *Incremental Blocking for Entity Resolution over Web Streaming Data* [16] in the *18th IEEE/WIC/ACM International Conference on Web Intelligence* (2019) in Thessaloniki - Greece;

7. Publication and presentation of the article *A Parallel-based Map Matching Approach over Urban Place Records* in the *24th Brazilian Symposium on Databases* (2019) in Fortaleza - Brazil;

8. Publication and presentation of the article *Parallel Blocking for Entity Resolution over Heterogeneous Data* in the *Database Thesis and Dissertation Workshop hosted by 24th Brazilian Symposium on Databases* (2019) in Fortaleza - Brazil;

9. Acceptance of the article *Schema-agnostic Blocking for Streaming Data* to be published in *35th ACM/SIGAPP Symposium on Applied Computing* (2020), in Brno - Czech Republic.

10. The article *Incremental Blocking for Entity Resolution over Big Heterogeneous Streaming Data* will be submitted to a international journal.

1.6 Document Structure

The remainder of this document is organized as follows:

Chapter 2 - Theoretical Foundation: addresses the main concepts inherent to the understanding of this thesis. This chapter describes the main concepts related to the following topics: i) Semi-structured data; ii) Noise data; iii) Streaming data; iv) Entity Resolution; v) Schema-agnostic blocking techniques; vi) Quality metrics of blocking techniques for ER; vii) MapReduce; viii) MapReduce-based blocking techniques; ix) Locality Sensitive Hashing (LSH); and x) Entropy of attributes.

Chapter 3 - Related Work: this chapter describes the main works present in the literature, as well as the approaches and blocking techniques proposed by each of these works. For a better understanding, the related works are divided into three groups: i) blocking techniques in the context of semi-structured data, ii) parallel blocking techniques in the context of semi-structured data, and iii) incremental blocking techniques in the context of streaming data.

Chapter 4 - Distributed Execution Model for Blocking Semi-structured Data: presents the distributed execution model proposed in this thesis, capable of efficiently processing a large number of entities, without reducing the quality of the generated blocks, for the different application profiles.

Chapter 5 - Spark-based Metablocking Technique: in this chapter, we propose a schema-agnostic parallel blocking technique, called Spark-based Streamlined Metablocking (SS-Metablocking). In addition to the new blocking technique, this chapter also proposes a new pruning algorithm (in order to improve the efficiency of the generated blocks) and a new load balancing technique (in order to improve the efficiency of the blocking technique).

Chapter 6 - Noise Tolerant and Schema-agnostic Blocking Technique: the Noise-aware Agnostic BLOCKing for Entity Resolution (NA-BLOCKER) technique is proposed in this chapter. NA-BLOCKER is capable of tolerating noisy data to extract information regarding the schema from the data (i.e., group similar attributes based on the data) and enhance the quality of the generated blocks. To this end, the NA-BLOCKER applies hash algorithms in order to hash the attribute values of the entities 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.

Chapter 7 - Incremental Schema-agnostic Blocking Technique for Streaming Data: the objective of this chapter is to propose a schema-agnostic blocking technique able to handle streaming data, named PaRallel-based Incremental MEtablocking (PRIME). The blocking technique implements strategies to process efficiently streaming data, considering the incremental behaviour. To this end, strategies are developed to quickly receive and process entities (sent continuously) and prevent the blocking task from consuming resources (e.g., CPU and memory) excessively.

Chapter 8 - Conclusions and Future Works: in this chapter, the conclusions related to the proposed artefacts (i.e., execution model and blocking techniques) are presented. Furthermore, future directions for this research are highlighted with the objective of detaching open areas, identified during the development of this study.

Chapter 2

Theoretical Foundation

This chapter discusses the basic concepts inherent to the understanding of the topics addressed by this thesis. The main concepts presented are: i) Semi-structured data; ii) Noise data; iii) Streaming data; iv) Entity Resolution; v) Schema-agnostic blocking techniques; vi) Quality metrics of blocking techniques for ER; vii) MapReduce; viii) MapReduce-based blocking techniques; ix) Locality Sensitive Hashing (LSH); and x) Entropy of attributes.

2.1 Semi-structured Data

Currently, part of data provided by data sources consists of records stored in database tables, spreadsheets, or text files (with comma-separated values - CSV - or tab-separated values) [21; 59]. Each one of these records refers to an entity and contains the same attributes. There is no hierarchy between the attributes in these records, i.e., the data is stored in a structured manner. Relational database systems are usually standardized, so that, different components are stored in different tables that are linked by foreign keys. To identify matches between structured data, it is necessary to extract the data from the normalized databases through queries, which combine the attributes of different tables into a single table or view model [21].

However, the number of Web knowledge bases that do not store their data in database tables (i.e., structured) [102] is growing. Semi-structured data formats have gained popularity in recent times including: XML (eXtensible Markup Language), RDF (Resource Description Framework) and JSON (JavaScript Text Object Notation). In these formats, the data does not follow a rigid structure (as in relational tables). Moreover, semi-structured schemas can be represented by trees, where the attributes are composed of sub-attributes. Thus, for a specific entity, each attribute has an associated value that describes information related to the entity in question. Considering the example depicted in Figure 2.1, an entity *Person* contains the attributes *name* and *address*. The attribute *name* contains sub-attributes such as *first name*, *surname* and *title*, while the attribute *address* contains common sub-attributes related to addresses, such as *street*, *number*, *city*, and *country*.

In the context of semi-structured data, the attributes of an entity can provide partial, overlapping and sometimes contradictory information for the same entity (in the real world). Thus, since semi-structured data does not follow a rigid schema, to identify the similarity between two entities it is necessary to explore the similarities of the attribute values of each of them. In other words, to compare an entity pair, it is necessary to consider the similarity of their textual content (described by attribute values) [25]. In general, the high degree of semantic heterogeneity reflected in different schemas makes ER, in the context of semi-structured data, an intrinsically complex task [13].

2.2 Noisy Data

People are less careful about the lexical accuracy of content written in informal virtual environments (e.g., social networks) or when they are submitted to some form of pressure (e.g., business reporting) [3]. For this reason, real-world data often suffers from noise that can impair data interpretations, data manipulation tasks, and decision making [43]. In the context of semi-structured data sources there is no difference. Typically, these data sources (mainly from the Web) present noise in the data, such as pronunciation errors, typos, slang and/or abbreviations[1] [29].

In this context, blocking techniques face serious impacts on block construction, as truly similar entities can be inserted into distinct blocks due to the spelling difference of their attribute values. This scenario determines that these entities will not be compared in the ER task and, consequently, these corresponding entities will not be identified. In order to address this problem, noise-tolerant blocking techniques should be applied. These techniques eval-

[1]In this paper, we will consider the two most common types of noise: pronunciation errors and typos [3].

```
{
    "name": {
        "first_name": "John",
        "surname": "McLaren",
        "title": "Doctor",
    },
    "address": {
        "street": "St John's Wood",
        "number": "25A",
        "city": "London",
        "country": "England"
    }
}
```

Figure 2.1: Description of entity *Person* in JSON.

uate the entities seeking to establish an approximate similarity between the attribute values, taking into account the potential presence of noise in the data. Therefore, it is possible to decrease the chances of truly similar entities being inserted into distinct blocks. For this reason, noisy data commonly reduce the effectiveness of the ER task and emerge as a challenge to be faced [90]. However, the development of noise-tolerant (or noise-aware) techniques is considered an open research area for many authors [62].

2.3 Streaming Data

Streaming data is continuously generated by one or more data sources. This data is usually sent simultaneously by each data source [44; 33]. This kind of data includes a wide range of data from different domains, such as mobile devices or sensors, websites, e-commerce, social networks, financial trading, and geospatial applications.

Given the constant updating of data, streaming data must be processed continuously, sequentially and incrementally. Data processing can be performed by record (as each record/-data is received) or over a certain period of time (data is stored in small packets for processing). Information extracted from this data can provide a wide variety of analytical data (for example, correlations between data and sampling). Moreover, such data may provide crucial information for companies, government agencies, and research agencies in real-time [107]. For example, a company can evaluate (in real-time) the public perception of its products from streaming data provided by social networks.

Nowadays, streaming data processing is applied in all domains where data is dynamic and continuously produced. For this reason, streaming data is related to a variety of Big Data scenarios [107]. In the context of ER applications, streaming data empowers several challenges already faced by the ER task [21]. Among them, we can highlight the scalability of task processing, since applications that handle streaming data require response time in the order of seconds or minutes. Therefore, new blocking techniques should be investigated in order to provide the ER task with high runtime efficiency.

2.4 Entity Resolution

ER is the task of identifying records stored in one or more data sources that correspond to the same real-world entities [21]. Entities can represent the most diverse records, depending on the domain in which they are related, such as: people (e.g., patients, clients, contributors or travellers), publications, movies, or products. In this sense, the ER task has emerged as a support task for various applications related to data mining, information integration, and data cleansing [12; 58; 104]. Over the years, several application domains and research fields have developed their own solutions for the ER task, and as a result, this task is known by many different names: record linkage, entity matching, object identification, or deduplication. A particular case of ER arises when the goal is to find records that refer to the same entity within a single data source, this task is commonly known as duplicate detection (deduplication).

In general, the ER task considers two data sources [21]. Thus, given two entity collections $A \in S$ and $B \in R$ from the data sources S and R, the purpose of ER is to identify all matches between entities (e) of the data sources. In this sense, each entity $e \in A$ or $e \in B$ can be denoted as a set of key-value elements that model its attributes (a) and the values (v)

associated with each attribute: $e = [(a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n)]$. Given $sim(e_i, e_k)$ the function that determines the similarity between the entities e_i and e_k, and Φ the threshold ($\Phi \in \mathbb{Q}$) which determines whether the entities e_i and e_k are considered as a match (i.e., truly similar entities). Thus, the task of identifying similar entities (ER) can be denoted as $ER(S, R) = \{(e_i, e_k) \mid e_i \in S,\ e_k \in R \mid sim(e_i, e_k) \geq \Phi\}$.

In traditional ER, entity comparisons are guided by the Cartesian product. In other words, all entities from A are compared to all entities from B ($A \times B$). For this reason, the ER task is considered a data-intensive task. Therefore, it is possible to assume that the asymptotic complexity of the ER task is quadratic $O(n^2)$, which clearly indicates inefficiency problems when the collections A and B reach the order of millions of entities [37]. When ER deals with large collections (i.e., Big Data), the application of blocking techniques is critical since they reduce the search space and hence speed up the execution of the task. Considering the challenges and problems inherent in the ER task, there is a necessity of proposing effective new approaches to improve both quality results (detect similar entities) and execution time.

2.5 Schema-agnostic Blocking Techniques

In the context of large entity collections (order of tens of thousands or millions of entities), blocking techniques can be applied as a prior step to perform the ER task, in order to prevent comparisons guided by the Cartesian product ($A \times B$). Thus, blocking techniques group entities so that similar entities are inserted into the same block. Consequently, blocking techniques restrict the ER execution to a smaller subset of entity pairs that have more chances to result in matches. Over the years, a wide number of blocking techniques have been proposed in the context of structured data, such as *Standard Blocking* [22], *Sorted Neighborhood* [57], *Fuzzy Blocking* [20], *Canopy Clustering* [67], *Duplicate Count Strategy* (DCS) [34], *Adaptive Window* [69] and Machine Learning-based techniques [35; 72]. However, since in the context of semi-structured data entities rarely follow the same schema, the block generation (in blocking techniques) is compromised. Therefore, traditional blocking techniques (e.g., sorted neighborhood and adaptive window) do not present satisfactory effectiveness, since the blocking is based on the entity profile schema [83]. In this sense, the challenges inherent in semi-structured data are addressed by schema-agnostic blocking techniques, which ignore scheme-related information and consider the attribute values of each entity [25]. Following, the main schema-agnostic blocking techniques are presented. The entity profiles (p) described in Figure 2.2, which describe people and their jobs, will be used throughout this section.

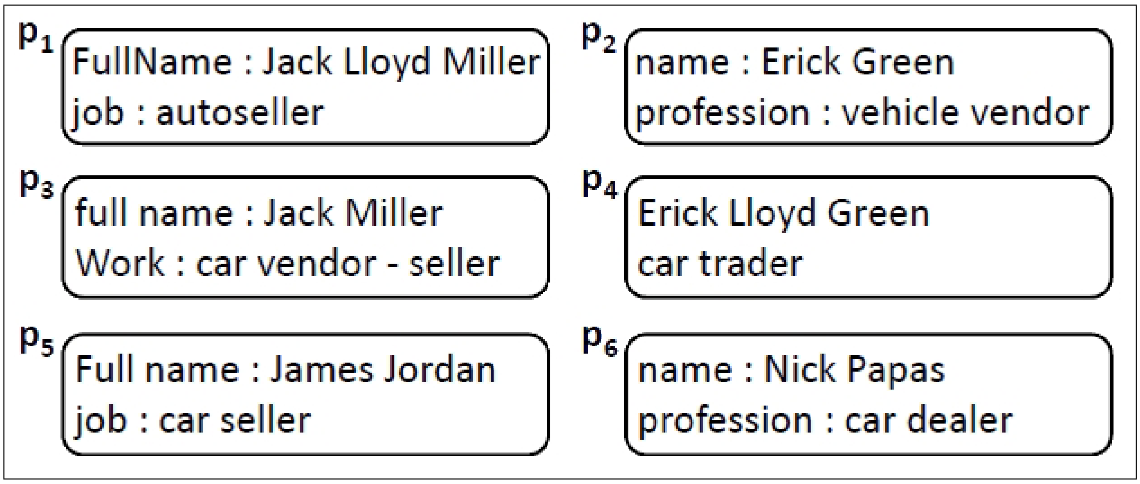

Figure 2.2: Example of semi-structured entities, extracted from [39].

Token-based Blocking

Token-based blocking [77] assumes that similar entities must have at least a value (or part of a value) of an attribute in common. In this way, the most repeating attribute values are selected so that each of these values assumes the role of a token. Tokens are used as a kind of identifier (key) for a block since each distinct token defines a new block. This blocking essentially builds an inverted index of entity descriptions: each token is a key and each block determines a list of entities associated with the key (i.e., token).

All entities that share the same token value (i.e., have some attribute value that contains the token) are grouped into the same block. Consequently, this kind of blocking overlaps entities between blocks. In other words, an entity may be contained in more than one block. Thus, by applying the token-based technique to the entities illustrated in Figure 2.2, it is possible to select the token *"car"*, present in several attribute values. In this sense, all entities that share the token *"car"* (i.e., have some attribute value that contains the token *"car"*) are inserted into this block. In this case, the entities p_3, p_4, p_5, and p_6 are inserted into the block associated with the token *"car"*. Figure 2.3 illustrates all blocks generated by the token-based blocking technique considering the entities illustrated in Figure 2.2. Besides the traditional token-based blocking, two other token-based blocking techniques have been proposed: [79] attribute blocking and prefix-infix(-suffix) [76; 78] blocking.

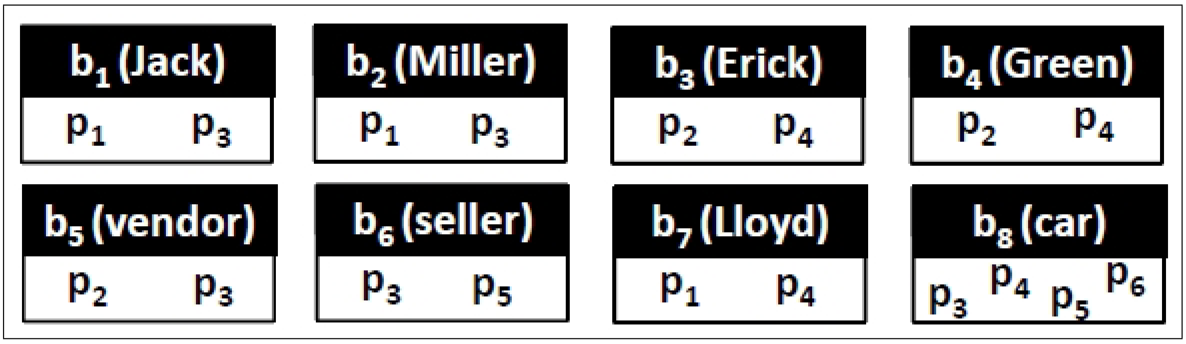

Figure 2.3: Blocks produced by the token-based blocking technique, extracted from [39].

Attribute-based blocking

The attribute-based blocking [79] exploits schematic information about the attributes of each entity in order to minimize the amount of false-positive matches. To this end, before the blocking step, an initial grouping of attributes is generated based on the similarities of their values. External sources such as thesaurus can be consulted to determine similar attributes. Thus, instead of searching for a common token, regardless of the attribute, only the tokens generated by similar attributes (i.e., attributes contained in the same group) are considered.

In the example illustrated in Figure 2.2, it is possible to group the attributes according to the semantic similarity. In this sense, to complement the example, the entity $p_7 : \{name : John\ Petter;\ job : Miller\ Beer\ seller\}$ will be considered. The attributes *"FullName", "full name", "Full name"* and *"name"* are grouped, so that only entities sharing tokens associated with attributes belonging to the same set are inserted into the same block. Afterwards, the blocking is performed considering the tokens associated with each block. In the example, although the token *"Miller"* is present in p_1, p_3, and p_7, the entity p_7 will not be included in the same block of the entities p_1 and p_3, since the attribute *"job"* (from p_7) is not in the same set of the attributes *"FullName"* and *"full name"* (from p_1 and p_3). For this reason, the entities p_1 and p_3 are grouped in the block *name.Miller* and the entity p_7 is inserted into the block *job.Miller*.

Prefix-infix(-suffix) blocking

Prefix-infix(-suffix) blocking [76; 78] is widely used in the Web context, especially in the Semantic Web, since it exploits information from URIs (*Uniform Resource Identifier*) associated with entity attributes. The key idea of this blocking technique is to analyze entity URIs in order to identify which entities have a common token in their literal values. The main disadvantage of prefix-infix(-suffix) blocking is the dependence of URIs on the description of entities.

This technique partitions the URI of each entity into prefix, infix, and suffix to identify the presence of common tokens. For example, the URI *"http://liris.cnrs.fr/Miller.company/seller: John Petter"* is partitioned into three tokens: *"http://liris.cnrs.fr"* (prefix), *"/ Miller.company/"* (infix), *"/seller:John Petter"* (suffix). Thus, this prefix-infix(-suffix) blocking technique uses URI-based tokens to generate the blocks.

Join-based Blocking

Join-based blocking [101] appears as an alternative blocking technique to other kinds of blocking since it applies linguistic similarity algorithms to determine the similarity between entities [25]. This blocking technique generates blocks based on linguistic similarity (*string-similarity join* [53]) between the attribute values of an entity. In other words, blocks are constructed by identifying all pairs of entities whose set of attributes have similarities above a certain threshold.

Unlike other techniques, join-based blocking selects keywords linguistically, from the number of occurrences of a given value (word) in the attribute values. For instance, the top-*"n"* values (words) that repeat among the entity attributes can be considered as keywords. After defining the keywords, the blocking technique generates an inverted index and compares the attribute values (of the entities) with the keywords, applying a linguistic similarity algorithm (e.g., Jaro-Winkler, Jaccard or Levenshtein [23]). If the similarity value between the attribute value and the keyword exceeds a predetermined threshold, the entity is inserted into the block.

Given the example illustrated in Figure 2.2, one possible keyword to consider is *"seller"* since it repeats twice among the attributes of the six entities in question. Then, the entity whose attribute value has a similarity above 0.4 (for instance, using the Jaccard algorithm) with the keyword *"seller"*, will be inserted into the block. In this sense, the block represented by the keyword *"seller"* will consist of the entities p_1, p_3, and p_5.

Frequency-based blocking

In this blocking technique, entities are grouped based on the occurrence of their attribute values [60]. Blocking techniques such as [61; 55] select the values (of attributes) that most repeat among the entities. Thus, after selecting the most frequent attribute values, the set of tokens (most frequent) are determined. Each set of tokens determines the formation of a block. Then, entities whose attribute values are contained in the set of tokens are inserted into the block in question. Assuming the entities described in Figure 2.2, the set of tokens $[Jack, Miller, seller]$ can be defined. Thus, the entities p_1 and p_3 are inserted into the block associated with the set ($[Jack, Miller, seller]$), since both entities share the $Jack, Miller$ and $seller$ attribute values.

Although this technique considerably reduces the amount of comparisons between entities with low chances of resulting in matches, the technique can also reduce the number of truly similar entity pairs identified, especially involving entities with a few number of attribute values in common.

Metablocking

The Metablocking technique initially applies a simpler schema-agnostic blocking (e.g., token-based) to identify possible pairs of entities to be compared. Token-based blocking is sensitive to the similarity of attribute values. Therefore, even entities that do not have a high similarity may belong to the same block. Moreover, this sensitivity to the similarity of attribute values tends to generate blocks with a large number of entities and, consequently, the amount of entity comparisons is not always reduced to a satisfactory quantity.

In order to minimize this problem, after the initial blocks are generated by the token-based blocking, the Metablocking technique creates a similarity graph based on the previously generated blocks, as shown in Figure 2.4 (a). Each node in the graph is represented by an entity and the edges represent similarity between nodes. The weight of edges between nodes (entities) is determined from the number of blocks in common between the pair of nodes (entity pair) linked by the edge. Then, as described in Figure 2.4 (b), the Metablocking technique prunes the graph based on edge weights, discarding edges that have a weight below a predetermined threshold. The resulting (pruned) graph is used to generate the new entity blocks, so that entities (nodes) that still have relationships (edges) are grouped together, and disconnected entities (no edges) are discarded, as shown in Figure 2.4 (c).

This technique, which was initially proposed in [77], has gained visibility among the

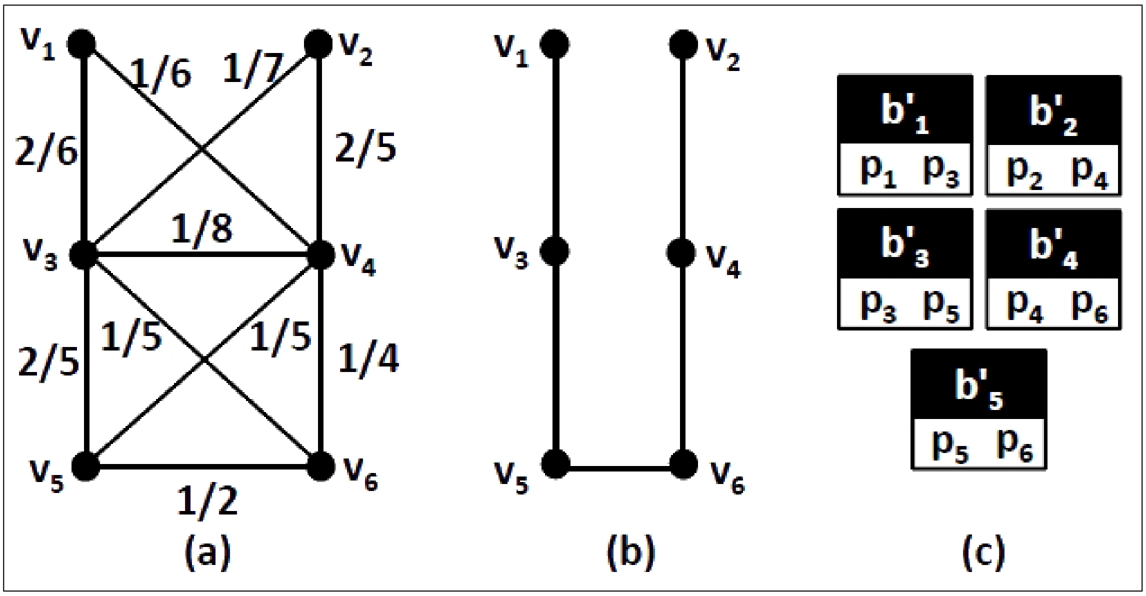

Figure 2.4: (a) Graph generated from the blocks in Figure 2.3, (b) Graph after the pruning step, and (c) New blocks after applying Metablocking. Figure adapted from [39].

schema-agnostic blocking techniques and is considered the state-of-the-art technique. For this reason, this technique has been improved both in terms of effectiveness [80; 83] and efficiency through the application of distributed computing [37; 8].

2.6 Quality Metrics for Entity Resolution and Blocking Techniques

The metrics Recall (R), Precision (P), and F-measure (F) are commonly used to measure the effectiveness of the ER task. These metrics use a gold-standard (containing truly similar entity pairs) to determine how close to the gold-standard the results given by an ER approach can achieve. These metrics are defined as follows.

- Recall (R): the number of correctly entity pairs classified as similar by the approach (C) divided by the total pairs of truly similar entities (G), defined by the gold-standard. This quality metric is calculated as follows: $R = \frac{C}{G}$.

- Precision (P): number of entity pairs correctly classified as similar by approach (C) divided by total entity pairs classified as similar by approach (T). This quality metric is calculated as follows: $P = \frac{C}{T}$.

- F-measure (F): the harmonic mean between precision and coverage. This quality metric is calculated as follows: $F = \frac{2 * R * P}{R + P}$.

More specifically, four metrics are commonly used to evaluate blocking techniques: Pair Completeness, Pair Quality, Reduction Ratio and F-score. Before describing each of the metrics, some variables must be defined: i) B : input block set, this set can consist of only two blocks, where one block contains all entities of a data source, and the other block contains all entities of the other data source; ii) $|B|$: total amount of comparisons between entities to be performed on the set of blocks; iii) B' : set of blocks generated after the execution of a particular blocking technique; iv) $D(B')$: amount of truly similar entity pairs that have been preserved by the blocking technique (i.e., which will still be compared in the ER task); v) G : amount of truly similar entity pairs contained in gold-standard. From the definition of these variables, the metrics are defined below.

- *Pair Completeness (PC)*: this metric indicates the proportion of truly similar entity pairs that have been preserved by the blocking technique in relation to the total of truly similar entities defined by the gold-standard. It is given by the formula: $PC = \frac{D(B')}{G}$.
- *Pair Quality (PQ):* measures the fraction between the number of truly similar entity pairs that have been preserved by the blocking technique and the total amount of entity comparisons determined by the blocking technique. It is calculated as follows: $PQ = \frac{D(B')}{|B'|}$.
- *Reduction Ratio (RR):* indicates the proportion of all possible pairs of entities that were eliminated by the blocking technique, i.e., indicates how much the search space has been reduced. It is determined by the formula: $RR = \frac{|B|}{|B'|}$.
- *F-score (Fs):* harmonic mean between the Pair Completeness and Pair Quality metrics. It is calculated as follows: $Fs = \frac{2 * PC * PQ}{PC + PQ}$.

The above quality metrics will be used to evaluate the quality of the blocking techniques proposed in this work.

2.7 MapReduce Model

MapReduce (MR) is a scalable distributed model for data processing whose data workflow is divided into the *map* and *reduce* phases [28]. In the *map* phase, *map* functions (tasks) transform input records into temporary records, which are not necessarily the same input data type. The *map* functions also map the entries with their respective keys, in order to form a set of pairs (key, value) of a temporary record. In the *reduce* phase, *reduce* functions are responsible for reducing a set of temporary values, which share the same key, in the final output of the process. The *reduce* phase has the following steps:

- *Sort*: this is the initial step of the *reduce* phase, which has as input the outputs of the *map* functions. The MR model groups the inputs of the *reduce* step by the keys, sorting them;
- *Shuffle*: in this step, the MR model analyzes the most relevant part of the output key of all *map* functions;
- *Reduce*: based on the pairs (key, value) generated by the *map* functions, the *reduce* phase groups the data (values) which will be processed in the same *reduce* function, grouping the data by the key.

In MapReduce, the data is initially partitioned between available infrastructure resources and stored in the Distributed File System (DFS). Data is represented as pairs (key, value) and calculations are expressed using two main functions:

- map : $(key_{input}; value_{input}) \rightarrow list(key_{tmp}; value_{tmp})$
- reduce : $(key_{tmp}; list(value_{tmp})) \rightarrow list(key_{output}; value_{output})$

All execution involving *map* and *reduce* phases in MapReduce is called a *job*. In each job, the map functions are applied in parallel on different partitions of the input data. The pairs (key, value) resulting from the *map* functions are partitioned by the key. For each reduce *task*, all incoming partitions are merged in a sorted order (*sort*), based on the keys. For each partition, the pairs are sorted by their key and then submitted to the *shuffle* step. All pair values that share the same key are sent to a single *reduce* function. The output of each *reduce* function is written to a distributed file (DFS).

In addition to the *map* and *reduce* functions, the MR framework also allows the user to provide a *combine* function, which runs as internal mappers after the conclusion of the *map* phase. *Combine* functions act as a local reducer, dealing with local pairs (key, value), which decrease the amount of data transmitted in the network. Finally, MapReduce also allows the user to configure (program) the *map* and *reduce* functions to address constraints and the workflow that will be parallelized. In this sense, the user can customize how the pairs (key, value) are partitioned and sorted [95].

Although there are several frameworks that implement the MapReduce programming model, such as Hadoop [99], Spark [103], Flink [18], and Samza [73]. Currently, Spark and Flink are the most efficient implementation of this paradigm in the open-source community. For this reason, the blocking techniques presented in this paper, as well as their evaluations, were implemented based on the Spark and Flink programmable model.

2.8 MapReduce-based Agnostic Blocking Techniques

In the context of structured data (i.e., where traditional blocking techniques are applied), a lot of parallel blocking techniques have already been proposed, including the application of MapReduce [68]. On the other hand, the main schema-agnostic blocking techniques have been proposed in recent years and few studies propose the application of distributed computing to improve the efficiency of these techniques. In this sense, recently published works highlight the lack of parallel schema-agnostic blocking techniques and also indicate the application of distributed computing in this context as an open research area [37].

Regarding the parallelization of blocking techniques, the approaches can be simple or complex. In the first case, the blocking technique is performed in a distributed manner, but without taking into account existing limitations in the MR model, such as load balancing and excessive resource consumption of the infrastructure (these limitations will be further discussed in the final part of this subsection). In the second case, the technique should be able to minimize the impact of MR model limitations. In this section, to provide a better understanding, we will use an example of a simple approach to the application of the MR model in a Token-based blocking technique.

In the simplistic approach, the *map* step determines the blocking keys (i.e., the tokens)

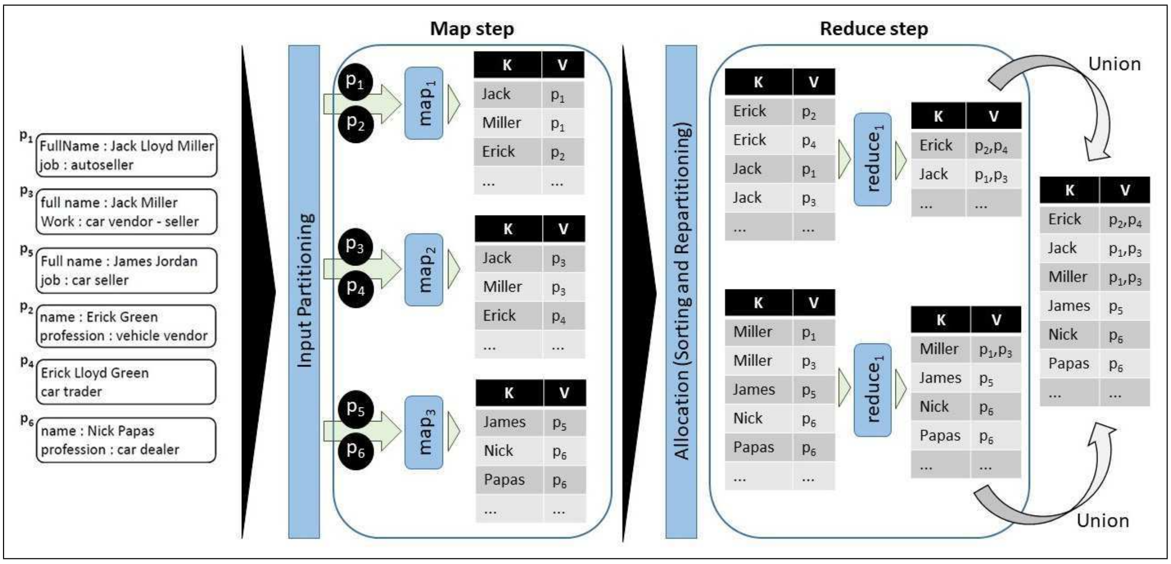

Figure 2.5: Example of a workflow for a Token-based blocking technique applying the MR model.

and returns a list of key-value pairs (token, entity). Later, in the allocation phase (*shuffle*), the blocking key is used to designate which key-value pairs will be sent for each available *reduce* task. In the *reduce* step, entities that share the same blocking key (token) are grouped into the same entity set. Thus, each of these entity sets represents a block.

In Figure 2.5, an example is illustrated where six entities from an input data source are processed by the Token-based blocking technique, applying the MR model. Regarding the MR model settings, three *map* tasks and two *reduce* tasks are used. Initially, in the partitioning step, input entities are distributed into subsets, so that each subset is sent to an available *map* task. In each *map* task, the data is read in parallel and the values of the blocking keys *K* (i.e., the tokens) for each input entity are determined. It is important to highlight that each entity can generate more than one token, and each token is associated with a blocking key. Therefore, the token *Jack* of entity p_1 assumes the role of key with the entity p_1 as the associated value. For a better visualization, in Figure 2.5 only a few *tokens* of each entity were considered. Finally, all key-value (token-entity) pairs are dynamically distributed by a partitioning function, so that all entities sharing the same key are sent to the same reduce *reduce* task. For this reason, both key-value pairs $\langle Jack,\ p_1\rangle$ and $\langle Jack,\ p_3\rangle$ are sent to the task *reduce$_1$*. In each *reduce* task, entities that share the same key are grouped into the same entity set to define the block associated with the *token* in question. In this sense, the entities p_1 and p_3 are inserted into the block defined by the token $Jack$. Finally, the outputs (i.e., generated blocks) from the *reduce* tasks are integrated and returned as the final result of the blocking technique.

Although the MR model facilitates the development and implementation of parallel approaches, it is important to highlight that there are three inherent limitations of the model that may degrade the efficiency (or make it impossible) of the execution of these approaches. Minimizing the impacts of these limitations tends to increase the logical and computational complexity of MR-based approaches. On the other hand, by addressing these limitations, it is possible to achieve better efficiency in terms of performance and resource consumption. For this reason, strategies that minimize the impacts of these limitations should be considered whenever novel approaches are proposed. The limitations are described below.

Disjoint data partitions. *Map* and *reduce* tasks in the MR paradigm are responsible for processing and sending data belonging to partitions specific to each of the tasks. For this reason, it is not possible to exchange data or information with other tasks (even those that have not yet been processed). This limitation directly impacts the entity mapping when there is a necessity to compare with entities or obtain information contained in other partitions. Therefore, it is necessary to ensure that all information necessary to perform the blocking technique, which will be processed by the corresponding task, is properly sent to the task in question, avoiding inconsistences and errors in the outputs of the blocking technique.

The frameworks Spark and Flink partially minimize this limitation by proposing two types of variables: accumulators and broadcast. Accumulators allow creating variables that accumulate values or information (e.g., counters). However, only the main program can read the value of an accumulator, i.e., distinct tasks (separate partitions) cannot read the value of an accumulator, only write to it. Broadcast variables allow data to be distributed across the resources of a computing infrastructure (computers or virtual machines) so that this data is stored in reading variables in the cache of each infrastructure resource. These variables can be used to provide nodes of a cluster with copies of large data sets. However, these variables cannot be dynamically edited by different tasks, only the main program can change the values stored in a broadcast variable.

Load Balancing. Load balancing refers to how workloads are distributed among two or more *map*/*reduce* tasks in an MR model [36]. The objective is a uniform distribution of workload among the task. Therefore, the tasks should receive similar loads (for example, entity comparisons) to be computed. Load-balanced distribution enables better resource utilization, maximizes approach performance, minimizes response time, and avoids overloading of the computing infrastructure. In the context of blocking techniques, load balancing for a MR-based technique seeks that each task performs approximately the same number of comparisons between entities. In this sense, the proposition and application of load balancing over MR-based techniques are generally fundamental to achieve uniformity in workload distribution [68].

The possibility of achieving a full load balancing (all tasks performing the same number of comparisons) depends on several factors pertinent to the problem in question (in this case, blocking techniques). For instance, it is possible to consider the data bias (difference in the size of entity blocks or number of attributes in each entity), the number of available map/reduce tasks, and the heterogeneity of the nodes available in clusters (configuration, processor, memory, operating system). Load unbalance directly impacts the total execution time of the approach as the tasks responsible for processing most comparisons will be time consuming, while other tasks will be idle due to the low workload allocated to them. Therefore, it is clear that each approach based on the MR model addresses in particular (proposing balancing strategies) the causes of load unbalance.

High Memory Consumption. As mentioned earlier, the entities to be compared must be sent to the same map/reduce task. However, in the MR model, all entities to be compared are loaded into memory before being processed in the task. Thus, after loading the entities in memory, the schedule will divide the entities among the tasks. The memory consumption is potentiated in scenarios where Token-based blocking techniques are applied, since the same entity may be tied to hundreds or thousands of tokens. For this reason, in some scenarios (especially in contexts involving Big Data) the memory available on each node is not sufficient to store a large set of entities, slowing or making the processing impossible. Notice that there is a strong relationship between high-memory consumption and load unbalance. It occurs due to a good load balancing implies directly balancing the memory consumption on each node, which avoids problems of high memory consumption by some computational infrastructure nodes.

In scenarios involving streaming data, data sources provide an infinite amount of data continuously. In this sense, incremental approaches that handle streaming data commonly suffer from memory issues since they need to maintain a large amount of data in memory [30; 85; 33]. It occurs due to incremental processing consumes information processed in previous increments. Considering these behaviours, it is necessary to develop strategies that manage the computational resources. In this sense, when blocking techniques face these scenarios, memory consumption tends to be the biggest challenge to be handled by incremental approaches [86; 46; 30].

2.9 Locality Sensitive Hashing (LSH)

Commonly used to assist the task of finding duplicate or similar documents (for instance, the ER task) from one or more data sources, LSH refers to a family of hash functions (i.e., functions able to map data of arbitrary size to fixed-size values) to hash records into a fixed-size set so that similar records are located in the same sets with high probability. On the other hand, low-similar records should be located in different sets. Hash values generated by hash functions are named hash-signatures. In general, LSH is used for approximating the near neighbour search in high-dimensional spaces [5]. It can be applied to reduce the dimensionality of a high-dimensional space, preserving the similarity distances and reducing significantly the number of the attribute values (or tokens) to be evaluated. Although LSH algorithms do not guarantee to generate exact answers (in terms of similarity), they will usually provide a good approximation with relevant results in terms of effectiveness.

Among the LSH-family of hash functions, it is possible to detach MinHash [87]. MinHash function was proposed by Broder [17] in order to estimate the Jaccard similarity (i.e., Jaccard similarity from MinHash - J_{MH}) between records (i.e., entities) of different data sources. The main idea of MinHash is to replace the set of original values with a unique representation. In other words, MinHash will use a single hash-signature to represent all values (from the attributes) related to the entity in question. To determine the similarity (between entities) from the attribute values of an entity ($V_a \in e$), MinHash applies k hash functions (MH_k) in all values that the attribute assumes (i.e., $\forall val \in V_a$). Finally, the algorithm selects the final representation (i.e., the hash-signature) of these values based on the k hash functions applied. Then, the hash-signatures (or MinHash signature - MS) is defined according to Equation 2.1:

$$MS_e = MH_k(val)\forall val \in V_a,\ where\, V_a \in e \tag{2.1}$$

To calculate the similarity between two hash-signatures (from two entities e_1 and e_2), the Jaccard similarity function is applied. Assume that the hash-signatures MS_{e_1} and MS_{e_2}, the Jaccard similarity between them is expressed by Equation 2.2 [17]:

$$J_{MH}(MS_{e_1}, MS_{e_2}) = \frac{MS_{e_1} \cap MS_{e_2}}{k} \tag{2.2}$$

with an error of $\theta = O(\frac{1}{\sqrt{k}})$, taking into account the approximation error inherent to hash functions.

2.10 Entropy of Attributes

To determine the relevance of a specific attribute of a data source regarding the other, it is possible to use a well-known concept in Information Theory, called entropy [26]. Entropy represents the information distribution of a random variable in an universe, in the context of this thesis the universe is all values assumed for an attribute. Therefore, the entropy of an attribute indicates how significant is the attribute, i.e., the higher the entropy of an attribute, the more significant is the observation of a particular value for that attribute [90]. For instance, the attribute *gender* (which assumes only two possible values - male and female) has a low relevance when compared with the attribute *name* (which assumes a wide number of possible values).

In this thesis, we specifically apply the Shannon entropy [63] to represent the information distribution of a random attribute. Thus, since X is a discrete random variable with the alphabet χ, and the probability distribution function $p(x) = Pr\{X = x\}, x \in \chi$, then, the *Shannon entropy* is defined as in Equation 2.3:

$$H(X) = -\sum_{x\in\chi} p(x)\log p(x) \tag{2.3}$$

In this thesis, entropy is applied in order to influence the similarity value between entities. Since the similarity between entities is based on the similarity between their attribute values, low-relevant attributes commonly do not contribute to determine the similarity between entities. In this sense, if the attribute assumes predictable values (for example, there are only two possible values), observing the same value in two distinct entities has a little relevance. On the other hand, if the attribute assumes on more unpredictable values (for example, there are 100 probable values for the attribute in question), observing two entities that have the same value for that attribute can be considered a significant indication that the entities are similar. For example, determining that two entities are similar just based on the similarity of the attribute *gender* may produce wrong matches, since this attribute is not so relevant when compared with similarity based on the attribute *name*. Therefore, the main idea is to apply entropy to emphasize similarities based on relevant attributes and minimize the impact of low-relevant attributes on the final similarity between entities.

2.11 Final Considerations

This chapter presented an overview of the context in which this thesis is inserted and described the main topics inherent to the domain of this research. First, the main concepts related to the nature of the data were presented and described, such as semi-structured data, noisy data and streaming data. Subsequently, the ER task was defined and described, as well as the main challenges faced by the task. The application of distributed computing (through MapReduce model) and blocking techniques were presented as possible solutions to minimize the problems related to the challenges faced by the ER task. Among the blocking techniques, we detach the schema-agnostic blocking techniques since they are widely applied in the context of semi-structured data. Then, the MapReduce model was detailed in order to highlight how blocking techniques can benefit, in terms of efficiency, from the application of this model. The limitations related to the MR model were stated and discussed in terms of the development of MR-based blocking techniques. Finally, LSH and entropy were described and defined. Following, we will present the related work chapter, which discusses the main works related to the ER task for semi-structured data.

Chapter 3

Related Work

In this chapter, we present the methodology to conduct the bibliography review. Moreover, we describe the main works present in the literature, as well as the approaches and blocking techniques proposed by each of these works. For a better understanding, the papers are divided into three groups: i) blocking techniques in the context of semi-structured data, ii) parallel blocking techniques in the context of semi-structured data, and iii) incremental blocking techniques in the context of streaming data.

3.1 Methodology

In the literature, there is a wide number of works involving ER and blocking techniques. The present work performed a formal systematic review to evaluate the main works related to schema-agnostic blocking techniques for ER. During all the development of this thesis, we performed a periodic searching using keywords in two platforms: GoogleScholar[1] and Mendeley[2]. Commonly, the keywords (or the combination of them) used for searching works were: Entity Resolution, schema-agnostic blocking techniques, noisy data, streaming data, and parallel blocking techniques. These searches resulted in a lot of works to be analyzed. To refine our search, we apply several criteria, such as title, summary, authors, year of publication, venue and number of citations. Title and summary assisted us to understand what the work proposes. From the authors, we can identify researchers that are working in the same

[1]https://scholar.google.com/

[2]https://www.mendeley.com/

area. The year of publication is useful to find the recently published works. Using venue and number of citations as a pruning criterion, we can identify the relevant works (i.e., with high quality).

In a subsequent step, the papers were read to identify the main contributions of the works. If the work is relevant and addresses the domain explored by this thesis, the work is included as a related work. More specifically, to consider a work as a related work we took into account the following criterion: i) the work addresses ER for semi-structured data, ii) schema-agnostic blocking techniques are proposed or evaluated, iii) the work addresses parallel blocking techniques, iv) the blocking technique proposed in the work handle noisy data, and v) the work proposes blocking techniques for streaming data. During the literature review, we found several research groups working with schema-agnostic blocking techniques, such as the groups guided by the Professors: George Papadakis, Giovanni Simonini, Kostas Stefanidis, and Vassilis Christophides. This information was useful in the sense to apply for the Ph.D. sandwich program, where we developed cooperative researches with the group guided by Prof. Kostas Stefanidis.

3.2 Blocking Techniques in the Context of Semi-structured Data

In the context of semi-structured data, the challenges inherent in blocking techniques (as discussed in Section 2.5) become even more complicated as the data does not follow a rigid and predefined schema [80]. For this reason, in recent years, a high number of works have been developed to propose blocking techniques capable of dealing with semi-structured data effectively and efficiently.

The work [77] emerges as one of the pioneers in proposing a blocking technique capable of effectively blocking semi-structured data. In this paper, the authors propose a technique capable of extracting tokens (i.e., keywords) from the attribute values of entities and grouping the entities that share tokens in common. Tokens are used as a block identifier, as each distinct token defines a new block. This blocking technique essentially builds an inverted index of entity descriptions: each token is a key, and each block stores a list of entities associated with a given token.

After the proposition of blocking techniques based on tokens, other works were developed in order to propose novel blocking techniques more effective and efficient, such as the works proposed in [55; 78; 79; 80; 89; 101]. In [75; 79], an attribute-based blocking technique is proposed, which also applies the strategy of using tokens as a blocking key. This technique aims to reduce the number of redundant entities present in the blocks (problem discussed in Chapter 2) and, consequently, to improve the performance of the work technique proposed in [77]. To this end, this technique initially groups similar attributes into the same group. To determine attribute similarity, *n-grams* are generated from attribute names and the linguistic similarity (for instance, by applying the Jaccard algorithm) between the attributes determines the degree of similarity between them. From the generation of attribute groups, entity blocks are constructed taking into account the attributes determined as similar (i.e., attributes contained in the same group). To generate entity blocks, tokens are extracted from attribute values. Thus, entities that share the same token are inserted into the same block, following the rule that the token is provided by attributes considered as similar. This strategy prevents linguistically identical tokens, but with distinct semantics, from determining the inclusion of entities in the same block. Therefore, this blocking technique further reduces the number of comparisons to be performed in the ER task (i.e., efficiency gains). On the other hand, attribute groups defined erroneously may prevent truly similar entities from being grouped into the same block, impairing effectiveness.

In [78], the authors present a blocking technique whose main idea is to reduce the number of redundant entity comparisons (i.e., entity pairs can be contained in more than one block) to be performed. The proposed blocking technique is applied in the context of Web data, which presents the Uniform Resource Identifier (URI) as one of their attributes. In this sense, the blocking technique extracts from the URI (i.e., prefix, infix, and suffix, as described in Chapter 2) information that will guide block generation. Such information is applied in order to insert into the same blocks only entities that have URI with high similarity. Therefore, this technique significantly reduces the number of comparisons to be performed in the ER task and aims to generate satisfactory blocks in terms of effectiveness.

The work [101] proposes a blocking technique based on the linguistic similarity of the attribute values. The proposed technique applies linguistic similarity algorithms (e.g., Jaccard, Dice or Levenshtein) to measure the degree of similarity between the attribute values and, consequently, to determine the similarity between entities. Thus, entities that have a similarity greater than a given threshold are inserted into the same block. The focus of this work is to generate high-quality blocks (i.e., capable of achieving good effectiveness results). On the other hand, due to the high number of comparisons between attribute values, the blocking technique does not present good results related to efficiency.

In the work [55], a blocking technique based on the frequency of occurrence of certain values is proposed. This work is an extension of the work which proposes blocking based on tokens [77]. However, the blocking technique proposed in [55] extracts sets of tokens based on the popularity of tokens among the entities. Then, each set of tokens is used as the blocking key. In other words, entities that share tokens contained in a given set (of tokens) are inserted into the block. The main purpose of this blocking technique is to improve the Pair Quality (i.e., PQ) metric of the generated blocks. However, the blocking technique has a limitation regarding the definition of token groups and the Pair Completeness (i.e., PC) metric. Thus, if the technique does not include a token in the set, entities truly similar are not contained in the same block, degrading the value of the PC metric.

In [80], the authors present the Metablocking technique, considered as one of the state-of-the-art techniques. This blocking technique aims to reduce the amount of redundant comparisons resulting from the blocks generated by the token-based techniques, and to prevent comparisons between entities with little chance of resulting in matches. Therefore, the technique in question seeks to improve results in terms of Pair Quality without degrading the Pair Completeness metric. For this reason, Metablocking is commonly applied as a post-processing step of the token-based blocking technique. Metablocking receives as input a set of blocks generated by the token-based blocking technique and converts them into a blocking graph, so that each node represents an entity and each edge (between a pair of nodes) denotes that the pair of nodes sharing at least one block in common. All edges are associated with a weight based on the number of blocks in common between the pair of nodes (entities). After, the blocking graph is pruned (using pruning algorithms) in order to generate a new set of blocks that involves significantly fewer comparisons (i.e., efficiency gains in the ER task) while maintaining the original level of effectiveness. This blocking technique is considered as one of the state-of-the-art techniques because the Metablocking achieves good results in terms of effectiveness and efficiency. However, the technique has limitations regarding efficiency when dealing with large data sources (Big Data). Such limitations led to the proposition of new blocking techniques, as described in Section 3.2.

Moreover, the paper [80] proposes four pruning algorithms, which emerge as a starting point for proposing new algorithms: i) Weighted Edge Pruning (WEP): individually evaluates each edge of the blocking graph with based on an overall threshold given by the average weight of all edges of the graph. All edges with a weight below the threshold are discarded; ii) Cardinality Edge Pruning (CEP): all edges of the graph are evaluated based on their weight and only the *top-k* highest edges are maintained; Weighted Node Pruning (WNP): evaluates each node in the graph individually by applying a local weight threshold (average weight of all edges connected to the node in question). Thus, the edges connected to the node, which have weight below the threshold, are discarded; and iv) Cardinality Node Pruning (CNP): edges connected to a specific node are evaluated based on their weight and only the *top-k* highest weight edges of the neighborhood of the node in question are not discarded. This process is performed for all nodes of the graph.

In [89], the authors propose a blocking technique called BLAST (Blocking with Loosely-Aware Schema Techniques). It aims to extract statistical information about the schema from the attribute values of the entities. Based on this statistical information, the attributes are partitioned (grouped) according to the similarity of their values, following the strategy proposed in [65]. After the attribute partition, BLAST applies a token-based blocking technique and considers the attribute groups to semantically differentiate the generated tokens. For this reason, entities are inserted into the same block only if they share the same token (linguistically similar) and the token is derived from attributes of the same group (of attributes). This strategy seeks to improve results related to the Pair Quality metric, since entities that have linguistically similar tokens but semantically distinct ones will not be inserted into the same block. After defining the blocks by the token-based blocking technique (respecting the constraint imposed by BLAST), Metablocking is applied in order to discard redundant comparisons and entity pairs with low similarity. In this sense, the BLAST blocking technique achieves effectiveness results superior to Metablocking work. However, in terms of efficiency, Metablocking gives better results compared to BLAST since a new step to extract schema information is added in BLAST.

The authors of [72; 106] propose the application of Machine Learning concepts over the

ER task in order to improve effectiveness. In this sense, in the work [72], a novel entity resolution framework (notice that this work addresses the ER task as a whole, not only the blocking step), which can effectively solve the heterogeneous data challenges. To this end, the proposed framework applies a deep align-compare-aggregate matching neural network over the ER task, which can effectively learn the token representations, capture the semantic relevance between tokens, and aggregate all matching evidence for accurate ER decisions in an end-to-end manner. Regarding [106], the authors propose a hands-off blocking framework for ER, based on similarity-preserving representation learning and nearest neighbor search. To this end, the framework applies neural networks to guide the blocking keys generation. However, this work does not directly address heterogeneous data challenges since it uses schema-based information to feed the neural network.

Concerning the blocking techniques stated above, none of them address problems related to streaming data or noisy data. For this reason, this thesis proposal treats such problems as possible open areas of research. In addition, it is important to highlight that all blocking techniques stated in this subsection detach limitations related to efficiency, especially for large data sources (i.e., sources that store hundreds of thousands or millions of entities). These challenges motivated the proposition of an execution model for blocking techniques, capable of addressing different scenarios faced by blocking techniques in the context of semi-structured data and large data sources.

3.3 Parallel Blocking Techniques in the Context of Semi-structured Data

The proposition of various agnostic blocking techniques, such as those described in Section 3.1, indicated several limitations commonly faced by these techniques. Among these limitations, it is possible to highlight the low efficiency achieved by blocking techniques in the context of large data sources (Big Data). For example, the work [83] states that Metablocking took more than 186 hours (8 days) to process a data source with approximately seven million entities. Such scenarios motivate the development of parallel schema-agnostic blocking techniques [36; 37; 90; 38], by applying programmable models such as MR.

Considering the proposition of schema-agnostic blocking techniques in parallel, these two works [36; 37] are the pioneers. Since Metablocking can face efficiency issues when dealing with large data sources, the works [36; 37] propose Metablocking in parallel from the application of the Hadoop framework. This blocking technique is divided into three steps, which served as the basis for the development of other works [90; 38]. Each step consists of a MapReduce job:

- *Block Filtering*: filters the input blocks to reduce the graph size used by Metablocking to perform the blocking. This step calculates the number of entity comparisons to be performed on each of the blocks and identifies which blocks each entity is contained. Finally, for each entity, blocks are sorted (by cardinality) in descending order and only the *top-k* (where "k" is a predetermined value) blocks are selected; the remaining blocks are discarded.

- *Preprocessing*: organizes the input data in the appropriate format for processing by the pruning algorithm (next step). The data provided by the block filtering step is formatted in key-value pairs, such that the key is the block identifier and the value is a set of entities (each entity has information about which blocks it is contained);

- *Metablocking*: in this step, the weighted blocking graph is generated and a pruning algorithm is applied to discard the low weight edges. It is important to remember that each node in the graph is represented by an entity, the edges represent the similarity between the nodes and the weight of the edges between the nodes (entities) is determined from the number of blocks in common between the pair of nodes (entities) linked by the edge. For each key-value pair provided by the preprocessing step, entities that share the same key are compared to generate the graph and define the weight of the edges. Finally, the pruning algorithm is executed and the low weight edges are discarded. Thus, the pruned graph determines which entities should be compared in ER task.

Regarding the load balancing technique (limitation discussed in Section 2.8) proposed in [37], the technique aims to evenly distribute the entity pairs among the partitions, so that each partition has a similar amount of comparisons to be performed. However, generating partitions with a similar amount of entity comparisons is not a guarantee of good load balancing, since allocating partitions to nodes can also cause load unbalance [69]. In this sense, the work proposed in [37] has limitations regarding the efficiency of the use of resources provided by the distributed infrastructure to be applied.

As an evolution of the work [89], the authors of [90] propose the parallelization of BLAST technique. In this sense, the technique maintains the strategy to automatically extract the loose schema information by adopting an LSH-based step. Compared with the steps proposed in [37], [90] added an extra step responsible to extract the loose schema information. Thus, all steps of the blocking technique proposed in [90] are performed in parallel for efficiently handling volume and schema heterogeneity of the data. To this end, the work employed Spark as the MR programmable model. The work [90] also appears as an evolution of the NA-BLOCKER technique (to be seen in Chapter 6) since it addresses a future work stated by us: the necessity of parallel techniques able to handle noisy data.

Regarding the work [38], it explores quality issues faced by ER task: incompleteness (i.e., partial data), redundancy (i.e., overlapping data), inconsistency (i.e., conflicting data), and incorrectness (i.e., data errors). To address these issues, the work proposes the MinoanER framework. This framework takes into account not only the similarities between the attribute values of the entity, but it explores semantic relations (i.e., neighborhoods) between entities. Thus, as the similarity value of entity pairs may still be weak (considering only attribute values), due to a highly heterogeneous description content, it is necessary to consider additional sources of matching evidence; for instance, the similarity of neighboring entities, which are interlinked through various semantic relations. To provide efficiency to the ER task, all the process is performed in parallel through the application of Spark as MR programmable model. Notice that, although [38] proposes an ER approach for semi-structured data, it explores different challenges related to complementary data, linking discovery, and semantic enrichment. It is important to highlight that exploring semantic characteristics is not in the scope of this thesis.

3.4 Incremental Blocking Techniques in the Context of Streaming Data

As stated in the last two subsections, a wide number of blocking techniques in stand-alone [79; 83; 102; 89; 27] and parallel [8; 37; 90; 42] modes have been proposed to deal with different types of data [84; 24]. In terms of incremental blocking techniques for relational data sources, the authors of [30; 46; 70] propose approaches capable of blocking entities in an incremental way. Thus, these works propose an incremental workflow to the blocking task, considering the evolutionary behavior of data sources to perform the blocking. The main idea of these papers 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 speed up, even more, a solution to incremental record linkage. However, these stated works do not deal with semi-structured and streaming data.

Related to other incremental tasks that propose useful strategies for ER, we can detach the works [54; 86] that propose incremental models to address the tasks of Name Disambiguation and Dynamic Graph Processing, respectively. More specifically in ER context, the work [74] proposes an incremental approach to perform ER on Social Media data sources. Although such sources commonly provide semi-structured data, this work generates an intermediate schema so that the extracted data from the sources follow such schema. Therefore, although the data is originally semi-structured, it is converted to a structured format before being sent to ER task. For this reason, [74] differs from our technique since it does not consider the semi-structured data challenges and does not apply/propose blocking techniques to support the ER task.

Regarding ER approaches that deal with streaming data, we can highlight the works [64; 85; 56]. 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 deals simultaneously with the three challenges (i.e., semi-structured data, streaming data and incremental processing) addressed by our work.

Based on the related work cited in this section (summarized in Table 3.1) and the information provided by several works [25; 31; 44; 84; 24], it is possible to detach the lack of works, in different areas, that address the challenges related to streaming data and incremental processing simultaneously. The same occurs in the context of ER approaches for semi-structured data. There is a lack of ER works that address challenges related to streaming data and incremental processing simultaneously, as stated in recent works [25; 84; 24; 90]. In this sense, our work addresses an open research area, which can emerge as a useful schema-agnostic blocking technique for supporting ER approaches in both scenarios.

3.5 Final considerations

Table 3.1 presents a comparative analysis of schema-agnostic blocking techniques in order to highlight the new contributions proposed in this work in relation to already reported works in the literature. Notice that, the last three works (in bold) of Table 3.1 are the three blocking techniques proposed in this thesis. Blocking techniques are compared according to the following criteria:

- *Classification:* the type of blocking strategy applied in the proposed technique, based on the classification described in Section 2.5;
- *Technique Complexity:* the level of complexity of the applied blocking strategy. For this criterion, it is necessary to evaluate the consumption of computational resources (memory and processing) when the blocking technique is executed and the asymptotic complexity of the blocking algorithm. The techniques are classified into three categories: simple, medium and complex. Simple determines that the technique applies simplistic blocking strategies, which do not consume many computational resources and the blocking algorithm does not present high asymptotic complexity. Medium means that the technique tends to consume a lot of computational resources, but the blocking algorithm does not present high asymptotic complexity. Complex techniques are related to a high resource consumption and high asymptotic complexity of the blocking algorithm;
- *Input Data:* the way (batch or streaming) how the blocking technique handle the input data;

- *Schema Information Extraction:* determines whether (or not) the blocking technique in question applies a strategy to extract schema-related information;
- *Pruning of Comparisons:* determines whether (or not) the blocking technique applies a strategy related to comparisons pruning;
- *Parallel:* determines whether (or not) the blocking technique is performed in parallel through a distributed computing infrastructure;
- *Framework:* name of the framework used by the work to parallelize the blocking technique;
- *Load Balancing:* the load balancing technique (only for parallel blocking techniques) proposed by the paper;
- *Incremental:* determines whether (or not) the technique performs the blocking incrementally (i.e., incremental processing).

Considering Table 3.1, it is possible to detach the works [75; 79; 89; 90], which propose blocking techniques that benefit from information related to entity attributes. To this end, the approaches exploit the loose schema information (i.e., statistics collected directly from the data) to enhance the quality of blocks. Based on [89], we propose the work [10], which outperforms the baseline in terms of effectiveness (as highlighted in Chapter 6). However, another evolution of [89] is also proposed in [90], which provides effectiveness improvements and proposes the parallelization of the blocking technique [89]. Although these works propose strategies to extract information from the attributes, they do not use this information to discard useless attributes. This idea is addressed in Chapter 8, which proposes a strategy to extract information and perform a kind of attribute clusterization to guide the block generation. In Chapter 8, we will discuss the use of schema information to discard useless attributes and provide efficiency for blocking techniques in the context of streaming data.

Another point to be highlighted is related to the fact that the present thesis proposes incremental blocking techniques to deal with streaming data, considered as an open area since no work reported in the literature addresses both challenges simultaneously. Finally, it is important to highlight the robustness and innovative character of the schema-agnostic blocking techniques proposed in this paper, which outperform the results (in terms of effectiveness and efficiency) achieved by state-of-the-art blocking techniques, as will be discussed in the following chapters.

Table 3.1: Related work comparison.

Work	Classification	Technique Complexity	Input Data	Schema Information Extraction	Pruning of Comparisons	Parallel	Framework	Load Balancing	Incremental
[77]	Token	Simple	Batch	No	No	No	Not applicable	Not applicable	No
[79; 75]	Attribute	Médio	Batch	Yes	No	No	Not applicable	Not applicable	No
[78]	URI	Simple	Batch	No	No	No	Not applicable	Not applicable	No
[101]	Join	Simple	Batch	No	No	No	Not applicable	Not applicable	No
[55]	Frequency	Simple	Batch	No	No	No	Not applicable	Not applicable	No
[80]	Metablocking	Complex	Batch	No	Yes	No	Not applicable	Not applicable	No
[89]	Metablocking	Complex	Batch	Yes	Yes	No	Not applicable	Not applicable	No
[36; 37]	Metablocking	Complex	Batch	No	Yes	Yes	Hadoop	MaxBlock	No
[90]	Metablocking	Complex	Batch	Yes	Yes	Yes	Spark	Greedy	No
[38]	Metablocking	Complex	Batch	No	Yes	Yes	Spark	Greedy	No
[30; 46; 70; 74]	Relational Schemas	Médio	Batch	No	No	No	Not applicable	Not applicable	Yes
[72; 106]	Machine Learning-based	Complex	Batch	No	No	No	Not applicable	Not applicable	No
[64; 56]	Not applicable	Complex	Streaming	No	No	Yes	Hadoop	Greedy	No
[85]	Not applicable	Complex	Streaming	No	No	Yes	Spark	Greedy	No
SS-Metablocking [8]	Metablocking	Complex	Batch	No	Yes	Yes	Spark	Cardinality-based	No
NA-BLOCKER [10]	Metablocking	Complex	Batch	Yes	Yes	No	Not applicable	Not applicable	No
PRIME	Metablocking	Complex	Streaming	Yes	Yes	Yes	Flink	Greedy	Yes

Chapter 4

A Distributed Execution Model for Blocking Semi-structured Data

Currently, the most part of the information systems have a growing necessity of integrating different data sources to provide complete and consistent information for their users [25; 21]. Data sources related to these systems commonly have a large number of entities (Big Data) and store semi-structured data. Moreover, such data typically has different characteristics associated with the application profile in question (e.g., health, Web, academic, and smart cities), such as: how the data is handled (batch or streaming); the quality of the data (presence, or absence, of noise in data); and prioritization of effectiveness/efficiency of the blocking (according to the necessity of the application).

As mentioned in previous chapters, ER is considered a complex task, not only because of its quadratic behavior, but also due to the nature of the data involved. Thus, large semi-structured databases and streaming data enhance the challenges inherent to the ER task, especially regarding efficiency. For this reason, blocking and distributed computing techniques can be used as solutions to improve efficiency. However, some studies [25; 37] report the lack of blocking techniques proposed in the literature capable of dealing with semi-structured data, especially in the context of streaming and Big Data. Therefore, the proposition of novel schema-agnostic blocking techniques is considered by many authors as an open research opportunity [25; 89]. These factors motivated the proposition of a distributed execution model for schema-agnostic blocking techniques. As stated, in this thesis, model is defined as a set of steps to perform the blocking task [14]. This execution model emerges as a robust, efficient and extensible solution to minimize the problems inherent to the ER task. Each sequence of step execution is defined as a workflow. In this thesis, we define four workflows, as described in 4.3. The workflows guide the proposition of the blocking techniques, which determines how (i.e., the strategies) to perform each step. For instance, the three proposed blocking techniques follow different workflows and contribute with novel strategies to be applied in the steps of the blocking task.

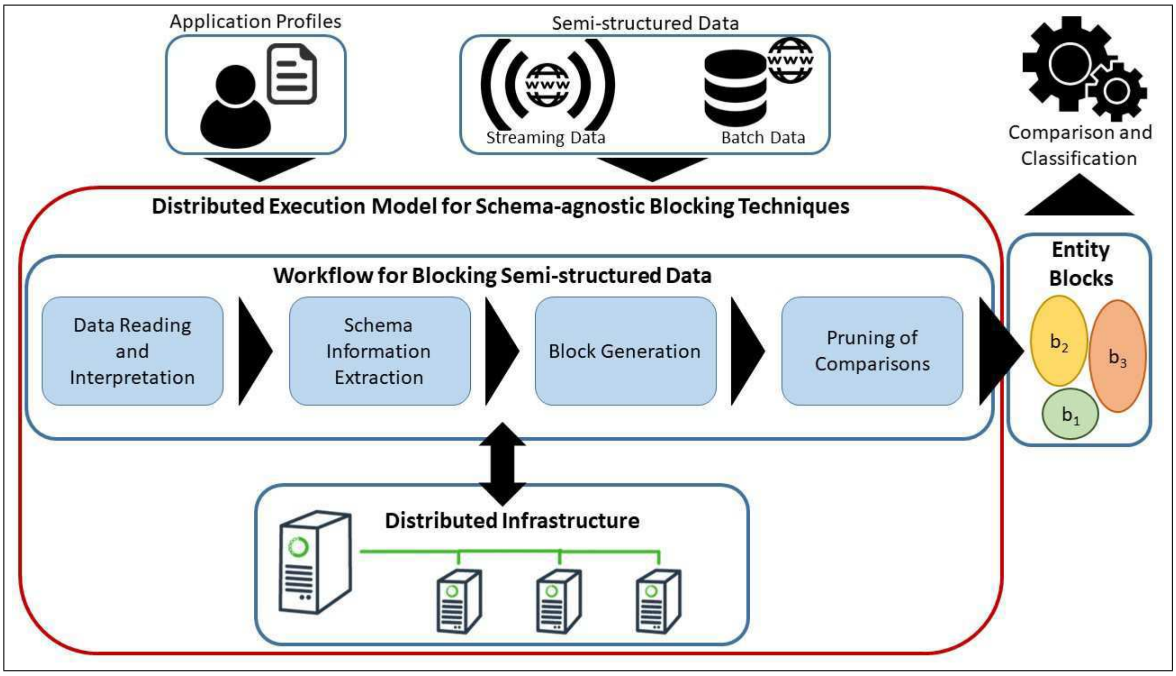

Figure 4.1: Distributed execution model for blocking semi-structured data.

The distributed execution model proposed in this work aims to enhance the efficiency of the application of schema-agnostic blocking techniques. Thus, the model allows the techniques to efficiently process a large number of entities without reducing the quality of the generated blocks. The other steps of the ER task (i.e., comparison and classification of entity pairs) are not covered by the model since the scope of this paper is limited to the entity blocking.

As illustrated in Figure 4.1, the distributed execution model receives as input the application profile provided by the user (e.g., a researcher or data expert) and the data sources. In other words, the user will select (e.g., from the use of a form) the options based on the characteristics defined by the application profiles: a) data provided in batch or streaming;

b) whether the data present noise; and c) the blocking task should prioritize efficiency or effectiveness. From the characteristics of the application profile (i.e., batch or streaming, presence/absence of noise in data, and prioritization of the effectiveness/efficiency of the blocking technique), we manually determine the workflow and the blocking technique that best addresses the profile in question is defined. The workflows are predefined, as described in Section 4.1 and illustrated in Figure 4.3. The workflows are related to the prioritization of efficiency or effectiveness. Focus on efficiency implies on select workflows with less number of steps. On the other hand, to improve effectiveness, other steps (such as, schema information extraction and pruning) should be considered in the blocking task. The blocking techniques address the different challenges related to data nature while the workflow will guide which steps of the execution model (i.e., *data reading and interpretation, schema information extraction, block generation* and *pruning of comparisons*) will be performed, as well as the order to execute each step. For instance, if a application profile involves batch data, noisy data and prioritization of the effectiveness, in this case, a noise-tolerant blocking technique able to handle batch data should be applied. Regarding the workflow, since effectiveness is prioritized, all the steps illustrated in Figure 4.1 should be applied to generate as high-quality blocks as possible.

At the distributed execution model, proposed by the present work, all entity blocking steps are executed in parallel. Therefore, with respect to the distributed computing application, the proposed execution model will use the MR programming model (more specifically the frameworks Spark and Flink) to parallelize blocking techniques. Regarding data manipulation, the proposed execution model will allow the application of blocking techniques for both batch and streaming data. The purpose of the execution model is, based on the characteristics of the informed application profile, to determine the workflow (from the set of predefined flows, see Figure 4.3) and the most appropriate blocking technique. The output of the execution model is composed of entity blocks generated from the blocking technique applied. These entity blocks are further processed by approaches that address the following steps of the ER task (i.e., comparison and classification).

4.1 Overview

From the distributed execution model, it is possible to propose novel blocking techniques or apply existing techniques, which follow the model workflow, in order to support the ER task (in the context of semi-structured data) in terms of efficiency. In the component diagram illustrated in Figure 4.2, it is easier to understand the different workflows of the proposed model. The component diagram is used to describe the blocking steps (*data reading and interpretation, schema information extraction, block generation* and *pruning of comparisons*) and the information transfer between each step, denoted by the connections between the components. The application profile selected by the user influences the choice of the workflow (from the predefined set of workflows), since the workflow that best fits the application profile in question will be selected. Moreover, to maximize the efficiency of blocking techniques, the execution model steps will use the MR programming model as a solution for executing tasks in parallel. More specifically, in this thesis, we applied the distributed frameworks Spark and Flink and the HDFS distributed file system (Hadoop Distributed File System), described in Section 2.7.

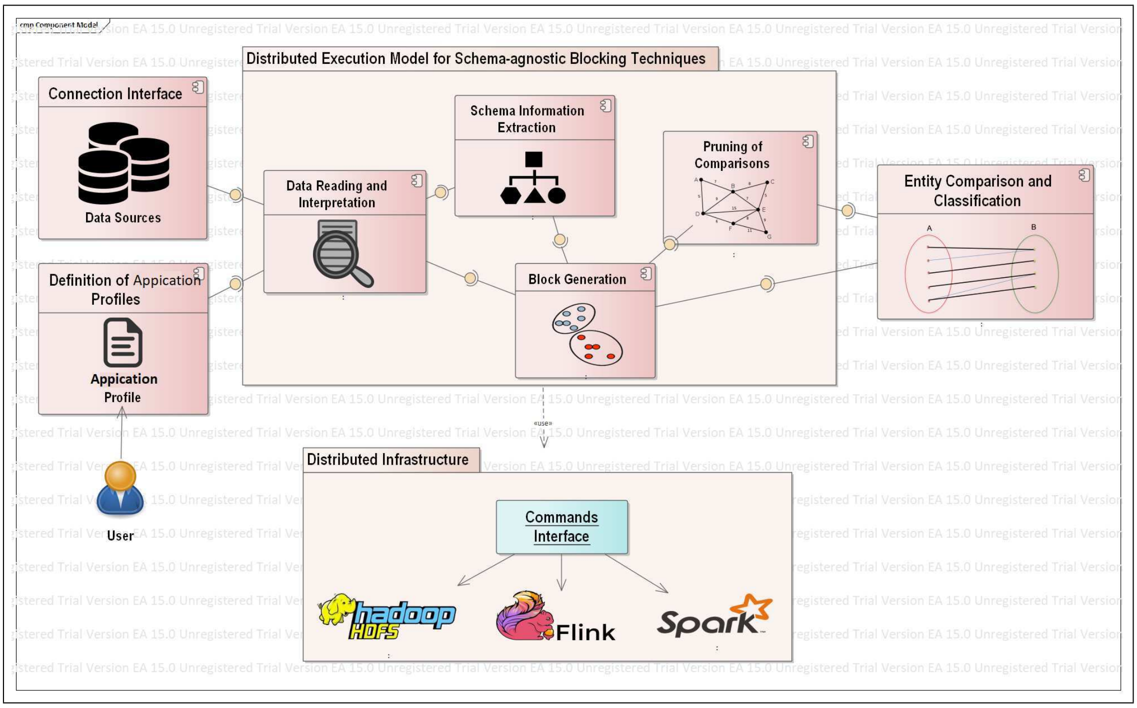

Figure 4.2: Component diagram of the distributed execution model for blocking semi-structured data.

Regarding the specific characteristics of each application, it is possible to highlight three profiles, related to how the data is handled (batch or streaming), data quality (noise level in data) and prioritization of the effectiveness/efficiency of the blocking technique. Following, we will describe the characteristics of each of these profiles.

- **How is data handled?** In general, data can be provided to the *data reading* step in two ways: batch or streaming. Data in batch is consumed, manipulated and processed at once. In other words, all data from data sources is sent at once to the *data reading* step. Another characteristic is related to data processing, which occurs only once, without the need of executing the execution model continuously. On the other hand, streaming data is received continuously. Data sources simultaneously send data packages to the *data reading* step at certain time intervals. In this sense, streaming data requires improvements in the execution model steps in order to address the efficiency restrictions of the data processing (where the data is received continuously). The characteristics (i.e., batch or streaming data) of this profile are fundamental for the model in order to select the appropriate blocking technique, which manipulates and processes the input data. In this sense, for batch data, the model should select blocking techniques which receive and process the data at once (for instance, the blocking technique proposed in Chapter 5). On the other hand, for streaming data, the model should select blocking techniques able to continuously process the data, such as the technique proposed in Chapter 7.

- **Data Quality**. This profile is related to the level and type of noise present in the data. For instance, data from social networks often present various noises related to the spelling of the words [62]. As described in Section 2.2, the presence of noise in data may reduce the effectiveness of blocking techniques. For this reason, the model allows the possibility of including blocking strategies that mitigate noise interference in the block generation. Thus, data that has a high level of noise should be blocked so that such noise minimally interferes with the quality of the generated blocks. Therefore, in applications where noise is present in the data, the execution model will select noise-tolerant blocking techniques (such as the technique proposed in Chapter 6). Otherwise (i.e., there is no noise in the data or low levels of noise), blocking techniques such as the technique proposed in Chapter 5 can be applied.

- **Effectiveness/efficiency**. In the context of blocking techniques, effectiveness and efficiency are antagonistic metrics [25]. Commonly, to achieve good results related to effectiveness metrics, it is necessary to apply sophisticated blocking strategies, which require a longer execution time (i.e., degradation of efficiency metrics). On the other hand, good results related to efficiency metrics can be achieved by applying simpler blocking strategies. However, more simplistic strategies tend to generate low-quality blocks and, consequently, low effectiveness values. In this sense, the necessities related to each application may influence the prioritization of the efficiency or effectiveness of blocking techniques. For this reason, the execution model provides the definition of alternative workflows, so that the efficiency or effectiveness of blocking techniques can be prioritized. Figure 4.3 illustrates the four possible workflows to be executed according to the effectiveness/efficiency prioritization. In applications whose focus is on the effectiveness of the generated blocks, the workflow 1 should be executed, in which the *schema information extraction* and *pruning of comparison* steps are applied in order to achieve better results related to the effectiveness of the generated blocks. In applications that focus on efficiency, the workflow 4 should be executed as the number of steps is reduced to minimize execution time. When the goal is to achieve harmonious results between effectiveness and efficiency, the workflows 2 and 3 can be executed.

In the next section, we will detail all the steps present in the execution model, illustrated in Figure 4.2. The application profiles detailed in this section will be covered in the description of each step in order to specify how each step will address the necessities inherent in each of the profiles.

4.2 Architecture of the Distributed Execution Model

Figure 4.4 illustrates the architecture of the distributed execution model for blocking semi-structured data, as well as the workflow of the model. All workflows are performed on a distributed infrastructure which applies Spark/Flink and HDFS to process and store data in

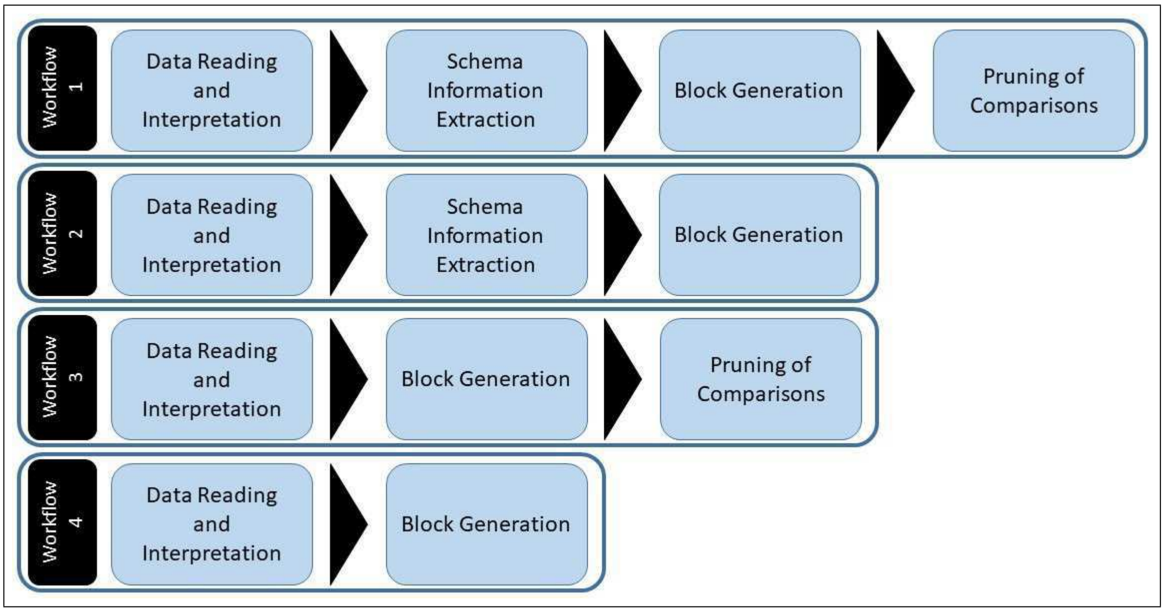

Figure 4.3: Possible workflows of the distributed execution model steps, according to the prioritization of the blocking technique effectiveness/efficiency.

parallel, respectively. Next, the execution of each step of the execution model are detailed.

4.2.1 Data Reading and Interpretation

In this step, the input data provided by the data sources is read. From the interpretation of these data, the entities of each data source are extracted, as well as the attribute names and attribute values of each entity. In the example depicted in Figure 4.4, three entities from each data source (D_1 and D_2) are identified. The entities e_1 : $\{$*name: Jon Snow; house: Stark*$\}$, e_2 : $\{$*name: Arya; house: Stark; adversary: Lennister*$\}$ and e_3 : $\{$*name: Tywin; house: Lennister*$\}$ belong to data source D_1 while the entities e_4 : $\{$*Full name: Jon Snow; family: Stark*$\}$, e_5 : $\{$*Full name: Aria; family: Stak; enemy: Lenister*$\}$ and e_6 : $\{$*Full name: Tywin; family: Lennister*$\}$ belong to data source D_2.

Notice that, in this step, the application profile informed by the user is interpreted to identify its characteristics. From these characteristics, the workflow (given the set of predefined workflows available in the model) and the schema-agnostic blocking technique that best fit the application requirements are determined. Afterwards, the entities are sent to the next step (i.e., schema information extraction), according to the selected workflow. Furthermore,

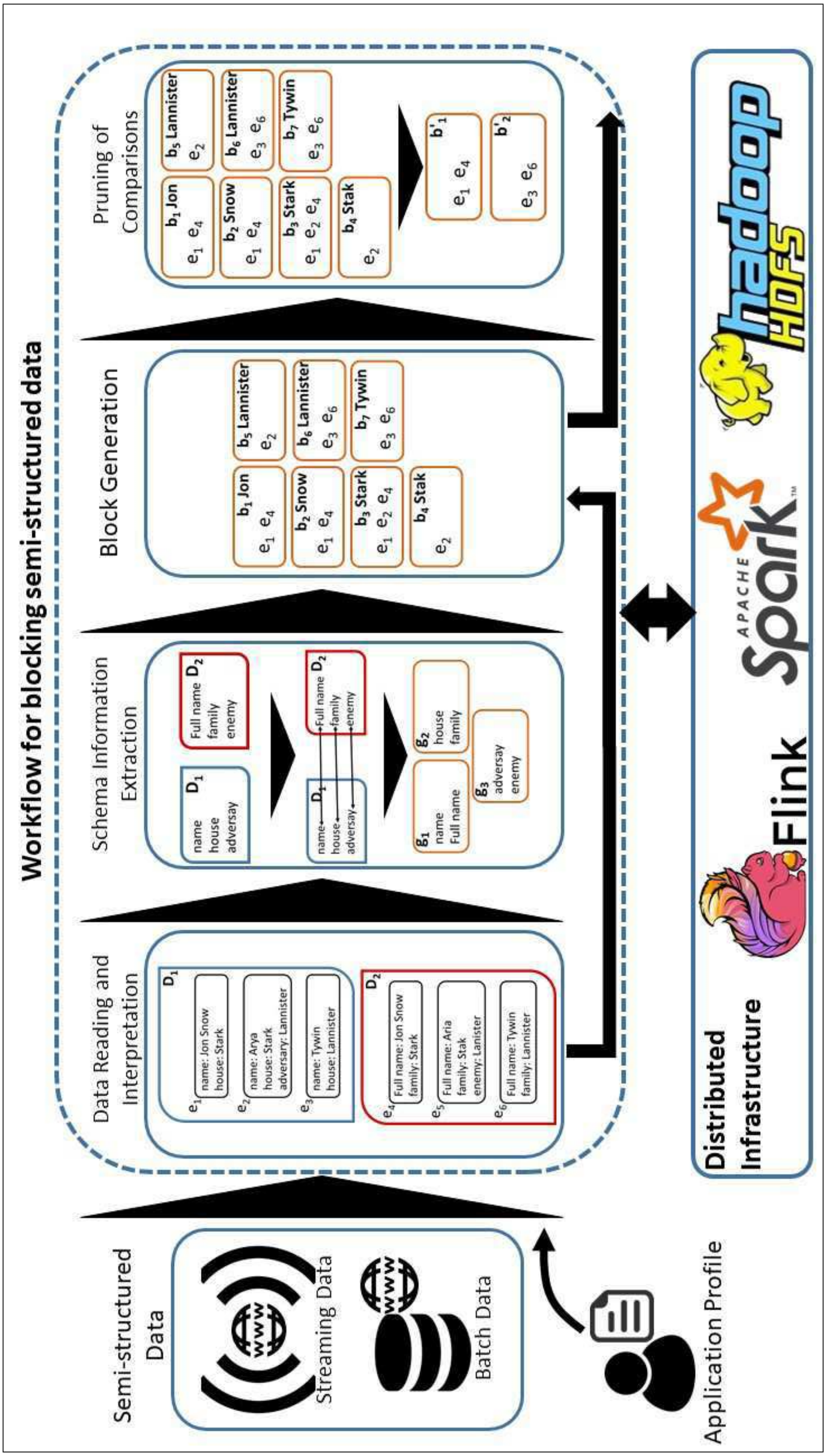

Figure 4.4: Architecture of the distributed execution model for blocking semi-structured data.

depending on how data is handled (batch or streaming) and the presence/absence of noise in data, the appropriate blocking technique is selected for the scenario in question.

4.2.2 Schema Information Extraction

This step receives as input the entities extracted from the two data sources D_1 and D_2 in the previous step. Initially, for each entity, the attribute values are analyzed in order to extract only the relevant data (tokens) associated with each attribute. In other words, punctuation, special characters and irrelevant words (e.g., articles and prepositions) are removed from attribute values. Thus, a set of values (keywords) V, extracted from the attribute value in question, is associated with each attribute ($\langle a, V\rangle$) of the entity. This transformation of the attribute values is important to remove characters or words that may degrade the computation of similarity between attributes.

In this step, techniques such as the one proposed by the authors of [89], which identify similarities between attributes based on values (V), can be applied. Such techniques are applied in order to group similar attributes. Therefore, from the similarity of attribute values, attribute groups are determined. In Figure 4.4, the attributes are evaluated and organized into three distinct groups: $g_1 = \{name, Full\ name\}, g_2 = \{family, house\}$ and $g_3 = \{adversay, enemy\}$. These groups will be used in next steps to avoid that attribute values derived from semantically distinct attributes determine the presence of similarity between two entities. In other words, attribute are considered similar only if their values are linguistically similar and the attributes associated with the values in question are contained in the same group. At the end of this step, the generated attribute groups are sent to the *block generation* step.

Depending on the application profile provided by the user, the *schema information extraction* step may be suppressed. This step directly influences the effectiveness of the blocking technique as it provides additional information related to the schema so that this information improves the quality (in terms of effectiveness) of the generated blocks. However, this step requires more computational resources (processing and memory) and time to be executed. Therefore, the *schema information extraction* step tends to negatively influence the efficiency of the model as a whole. In this sense, profiles that prioritize efficiency should follow an alternate workflow (workflows 3 or 4 illustrated in Figure 4.3) in which this step is not performed, as described in Figure 4.4. Alternatively, this step should be considered for profiles where noise is present in the data and/or profiles which prioritize the effectiveness of the blocks to be generated.

4.2.3 Block Generation

In this step, the inputs are the entities from the data sources D_1 and D_2 (extracted in the *data reading and interpretation* step) and the attribute groups generated in the *schema information extraction* step (if this step was performed). Initially, attribute values from the entities are evaluated in order to start the blocking of entities, so that similar entities are grouped into the same block. To this end, different schema-agnostic blocking techniques (such as those discussed in Section 2.5) can be applied. The extensible behavior of this step allows novel schema-agnostic blocking techniques to be added and, consequently, coupled to the model proposed in this work.

In the example illustrated in Figure 4.4, the blocking technique applied is based on tokens. Thus, tokens (keywords) are extracted from the attribute values of each entity. Then, entities sharing the same token are grouped into the same block. Therefore, entities e_1, e_2, and e_4 are grouped into the block b_3 since these entities share the token "*Stark*". It is important to highlight that the entity e_5 was not inserted into the block b_3 due to the misspelling of the attribute value *"family"*, originating the *token* "*Stak*". For this reason, another block (i.e., block b_4) is generated to store the entities that share the *token* "*Stak*" (in the example, only entity e_5). This scenario exemplifies how noisy data can directly influence the construction of the blocks and, consequently, impact on the quality of the generated blocks (in terms of effectiveness). This type of problem denotes the importance of applying noise-tolerant blocking techniques to noisy data, such as the technique proposed in Chapter 6. Such techniques are able to tolerate the noise in data in order to prevent negative impacts on the effectiveness of the results (i.e., in the generated blocks).

The attribute groups determined in the previous step are applied to this step only if the *schema information extraction* step has been performed. The absence of the attribute groups does not invalidate the execution of the *block generation* step since attribute groups are only used to improve the efficiency of the generated blocks, especially with respect to the *pair quality* metric. To this end, the attribute groups guide the block generation to include in the same block only entities that share tokens from attributes contained in the same attribute group. It is important to highlight that the application of attribute groups prevents entities that have linguistically similar tokens, but with distinct semantic meanings, from being included in the same block. For instance, in Figure 4.4, two blocks were generated from the key based on the token *"Lannister"*, one block based on the token *"Lannister"* derived from the attribute *"house/family"* ($\in g_2$) while the other block is based on the token *"Lannister"* derived from the attribute*"adversary"* ($\in g_3$). Therefore, entity blocking guided by attribute groups prevents that tokens with distinct semantic meanings (for instance, derived from the attributes *"house/family"* vs. *"adversary"*) group entities into the same block.

Considering the different application profiles, this step should be performed regardless of the profile provided by the user, as illustrated in the workflow of Figure 4.4. The *block generation* step is indispensable to the model (which hosts schema-agnostic blocking techniques) since this step is responsible for generating blocks and, consequently, determining the pairs of entities to be compared. In application profiles where noise is present in the data, noise-tolerant blocking techniques should be applied (e.g., the blocking technique proposed in Chapter 6). Finally, the output of the *block generation* step is formed by entity blocks, which may (or not) be refined in the *pruning of comparisons* step.

4.2.4 Pruning of Comparisons

The main objective of this step is to discard redundant comparisons and comparisons involving entity pairs that rarely will result in matches. To this end, techniques based on *Metablocking* [36; 80; 89] can be applied. Such techniques receive as input the blocks generated in the previous step and restructure the blocks into new blocks that involve significantly fewer comparisons, maintaining the original level of effectiveness [80]. To determine which pairs of entities have more chances to result in a match, *Metablocking* techniques are based on the premise that the larger the number of blocks the pair of entities in question co-occur, the greater the chances of this pair of entities being considered as a match.

As described in Section 2.5, the Metablocking technique generates a weighted graph to restructure the input blocks. Thus, each node in the graph represents an entity and each edge (between a pair of nodes) denotes that the pair of entities share at least one block in common. In addition, all edges have an associated weight. The weight is determined based on the number of blocks in common between the pair of nodes (entities) connected by the edge. After the graph generation, pruning algorithms are applied to the graph in order to discard edges with a weight below a given threshold (usually determined by the pruning technique). Finally, only the nodes (entities) that remained linked by edges will be structured in new blocks. Such blocks are the output of the *pruning of comparisons* step.

In Figure 4.4, the entities e_1 and e_4 are contained in the blocks b_1, b_2, and b_3. Therefore, these entities must be compared three times (i.e., they represent redundant comparisons), one comparison in each block. Thus, pruning techniques are applied in order to prevent such pairs of entities from being compared more than once. Furthermore, based on the number of blocks in common between entity pairs, only entity pairs with a high number of common blocks (i.e., with a high-weight edge) will be maintained by the pruning algorithm; the others will be discarded. In this sense, the entity pair e_1 and e_4 (contained in three blocks - b_1, b_2 and b_3) generates the block b'_1 and the entity pair e_3 and e_6 (contained in two blocks - b_6 and b_7) generates the block b'_2. Therefore, only two blocks (b'_1 and b'_2) will be generated as a result of the *pruning of comparisons* step.

This step is critical for application profiles that aim to generate high-quality blocks since redundant comparisons and entity pairs with little chance of matching are discarded. However, there is a cost (i.e., increased execution time and resource consumption) associated with this step regarding efficiency. For this reason, this step can be suppressed for application profiles that prioritize efficiency, as described in Figure 4.4.

4.3 Final Considerations

In this chapter, a distributed execution model for schema-agnostic blocking techniques is proposed, as well as the definition of the different workflows of the execution model and the architecture which describes the steps and components of the proposed model. The execution model is divided into four steps: data reading and interpretation, schema information extraction, block generation, and pruning of comparisons. The steps of the execution model served as a basis for implementing blocking techniques able to handle the different application profiles described in this chapter. In the next chapters, schema-agnostic blocking techniques proposed during the development of the thesis will be presented. The proposed blocking techniques were based on the execution model steps, implementing the particular functionalities of each step of the model (as depicted in Figure 4.4). Depending of the context addressed by the proposed blocking techniques, they can apply all or part of the steps (see Figure 4.3).

Chapter 5

Spark-based Metablocking Technique

In the context of large data sources, the ER task faces different challenges related to data volume and variety since blocking techniques need to process a large number of entities, whose entity profiles follow different formats (i.e. heterogeneous data) [36]. For this reason, blocking techniques and distributed computing are applied in order to minimize the problems related to Big Data [68]. However, in Big Data scenarios, blocking techniques also present efficiency-related limitations [36]. In this sense, the parallelization of blocking techniques aims to reduce the overall execution time of entity blocking by distributing the process of blocking entities among the various resources (e.g., computers or virtual machines) of a computational infrastructure.

As stated in Chapter 2, there are several schema-agnostic blocking techniques, such as Token-based blocking, Attribute-based blocking, Metablocking, and Join-based blocking [25]. Among the schema-agnostic blocking techniques, Metablocking (described in Section 2.5) emerges as one of the most promising techniques in terms of efficiency and effectiveness [83]. To enhance the efficiency of Metablocking, the work [37] proposes the parallel-based metablocking technique by applying the Hadoop framework. In particular, parallel techniques often face problems related to load imbalance (described in Section 2.8) [68]. For this reason, the work [37] proposes the balancing technique called MaxBlock, which seeks to evenly distribute the number of entity pairs to be compared in each partition. In this context, this chapter aims to propose effective and efficient strategies to be applied in the *block generation* and *pruning of comparisons* steps of the distributed execution model proposed in Chapter 4. The main contributions described in this chapter are:

- We propose an evolution of the work [37], called Spark-based Streamlined Metablocking (SS-Metablocking). Although the novel technique is based on the workflow proposed in [37], it applies parallel resources (e.g., accumulators and broadcast variables) provided by Spark to assist the execution of the blocking task;

- We propose a *Cardinality-based* load balancing technique which is applied to SS-Metablocking. Based on the amount of comparisons (cardinality) to be performed in each block, the load balancing technique evenly rearranges the pairs of entities into partitions that will be sent to the nodes (i.e., computers or virtual machines), such that each node performs a similar number of comparisons;

- We propose the *Global Weighted Node Pruning* (GWNP), a novel pruning algorithm that evaluates globally and locally the graph in order to improve the effectiveness of the technique, without compromising the efficiency.

5.1 Spark-based Streamlined Metablocking

In this section, we describe the Spark-based Streamlined Metablocking[1] technique with load balancing. The GWNP pruning algorithm and the *Cardinality-based* load balancing technique are applied to SS-Metablocking in order to minimize the challenges reported in Section 2.8. The novel technique is divided into three steps: block filtering, pre-processing, and metablocking, as depicted in Figure 5.1.

The SS-Metablocking receives as input the blocks generated by a schema-agnostic blocking technique (for instance, token blocking). For each block, the amount of comparisons between entities is calculated in order to assist the *Cardinality-based* load balancing technique. Blocks that contain a large number of entities tend to have low relevance, since they were generally generated from tokens commonly found in entities. For this reason, in the block filtering step, blocks that have a high cardinality are discarded. The Pre-processing step is responsible for organizing the entities so that each entity contains the information of which blocks have the entity in question, enabling the definition of similarity between the entities (based on the number of blocks in common). In the Metablocking step, the blocking graph

[1] https://bitbucket.org/account/signin/?next=/tbrasileiro/ss-metablocking

is generated and the GWNP pruning algorithm is applied. In order to improve the efficiency of this step, the *Cardinality-based* load balancing evenly distributes comparisons among the nodes. As a result of the blocking process, a pruned graph is generated containing the entities to be compared in the next steps of the ER task. Following, we describe the execution flow of each step and the execution of the proposed load balancing technique.

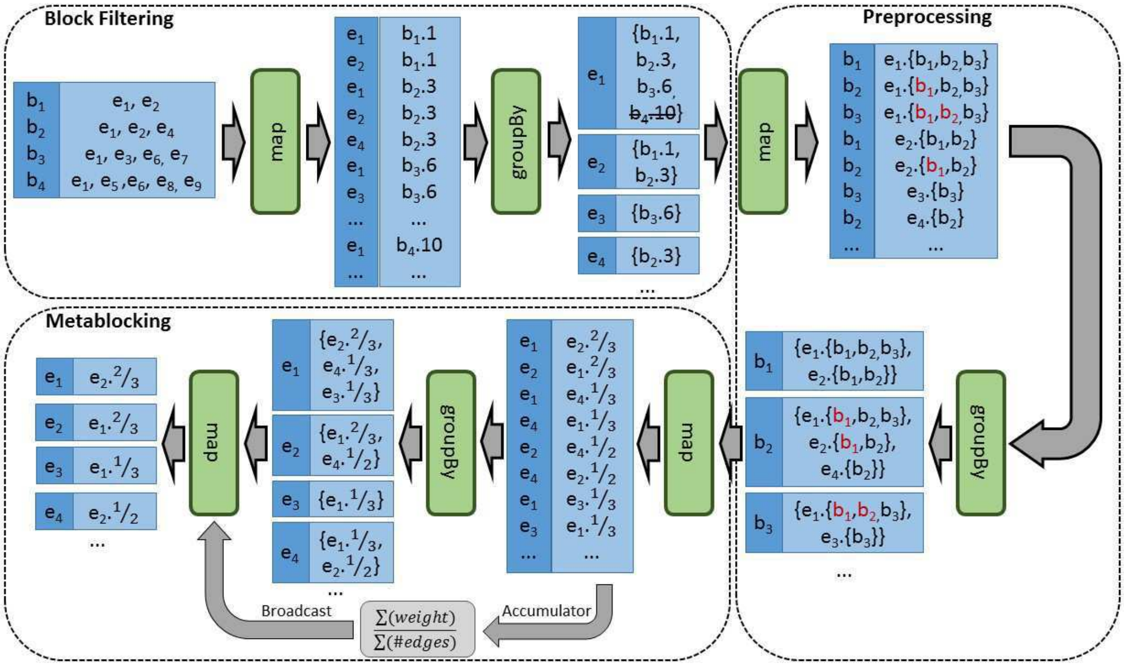

Figure 5.1: The workflow of the blocking technique *Spark-based Streamlined Metablocking*.

5.1.1 Step 1: Block Filtering

This step receives as input a set of agnostic blocks. Each agnostic block is formatted as $\langle k_b, E \rangle$, where k_b is the block identifier and E is a set of entities (represented by its identifiers) contained in the block. In the *map* task, for each entity of a block k_b a key-value pair $\langle e, k_b.||b|| \rangle$ is generated, where e is the entity identifier and $||b||$ is the cardinality of the block. Thereafter, at the *groupBy* task, the values $(k_b.||b||)$ are grouped by the key (e) and sorted by the block cardinality in descending order. In turn, only the top-k blocks are considered in the next step.

In the first *map* task depicted in Figure 5.1, the entities e_1 and e_2 of block b_1 receive $b_1.1$ as value, since b_1 contains only one entity pair to be compared. In the *groupBy* task, the values are grouped by the key e_1 to generate the set $B = \{b_1.1, b_2.3, b_3.6, b_4.10\}$. Considering only the top-3 blocks, the block b_4 is discarded from the blocks of e_1.

5.1.2 Step 2: Pre-processing

This step organizes the entities so that each entity contains the information of which blocks have the entity in question, enabling the definition of similarity between the entities (based on the number of blocks in common) in the Metablocking step.

In this step, the key-value $\langle e, B\rangle$ pairs, generated in the previous step, are transformed into a format to favor the execution of the pruning algorithm, at the Metablocking step. As illustrated in Figure 5.1, for each pair $b \in B$, a new key-value pair is generated in the format $\langle b, e.B\rangle$. Thus, in the *map* task, the key-value pairs $\langle b_1, e_1.\{b_1, b_2, b_3\}\rangle$, $\langle b_2, e_1.\{b_1, b_2, b_3\}\rangle$ and $\langle b_3, e_1.\{b_1, b_2, b_3\}\rangle$ are generated from the pair $\langle e_1, \{b_1, b_2, b_3\}\rangle$. Moreover, for each pair $\langle b, e.B\rangle$, the blocks contained in B with an identifier lower than b are added in the set of marked blocks (MB). Then, for the pair $\langle b_3, e_1.\{b_1, b_2, b_3\}\rangle$, the blocks b_1 and b_2 ($\in B$) are added to MB since their identifiers are lower than b_3 (the key). This information will be consumed by the Marked Common Block Index (MaCoBI) condition in order to avoid redundant comparisons. The MaCoBI condition is satisfied when there are no marked blocks in common between the entities e_i and e_j, i.e., $MB_i \cap MB_j = \emptyset$.

In the *groupBy* task, the values that share the same key (b) are grouped. As output, the *groupBy* task emits key-value pairs in the format $\langle b, S\rangle$, where S is a set of values $e.B$ that share the same block b. Furthermore, together with each entity e, the set B is also emitted with all the blocks which contain the entity e. This information is necessary to calculate the weight of edges between entities. Thus, since the values $e_1.\{b_1, b_2, b_3\}$ and $e_1.\{b_1, b_2\}$ share the key b_1, the pair $\langle b_1, \{e_1.\{b_1, b_2, b_3\}, e_1.\{b_1, b_2\}\}\rangle$ is generated and sent to the Metablocking step.

5.1.3 Step 3: Metablocking

In the Metablocking step, the weight of edges are calculated and the GWNP pruning algorithm is applied in order to generate the pruned graph. The edges (between two entities) of the graph correspond to pairs of entities that should be compared in the ER task.

Based on the workflow of the Hadoop-based Metablocking, we can highlight some weaknesses regarding efficiency and effectiveness of the blocking technique. The use of local pruning algorithm can include entity pairs with edges of low weight into the pruned graph since the weights of edges for a certain neighborhood can be low. Consequently, the value of the pruning threshold will be low. For instance, in Figure 5.2, the weight of the edges linked to e_1 can be considered low (the edge weight varies between 0 and 1). Assuming that the pruning threshold for entity e_1 is 0.13 (given by the average of weights), the edges that link e_1 to e_3 and e_1 to e_4 are not discarded since their weight is higher than the pruning threshold. Although the entity pairs $\langle e_1, e_3 \rangle$ and $\langle e_1, e_4 \rangle$ have edges with low weight, they will be compared in the ER task. It is important to highlight that entity pairs whose edge have low weight should be discarded since they have few chances of being considered a match. This scenario directly affects the number of entity pairs that must be compared in the ER task. Although WNP identifies a large number of matches, this pruning algorithm performs more comparisons than the other algorithms (e.g., WEP, CEP and CNP), as evidenced by experiments conducted in [37]. To discard the entity pairs with edges of low weight, a global threshold can be applied, which considers the weight of all edges of the graph. However, in the context of Hadoop-based Metablocking, the combination of global and local thresholds requires increasing the number of MapReduce jobs and, consequently, enhancing the overall execution time. This scenario motivated the development of the Global Weighted Node Pruning (GWNP) pruning algorithm. GWNP is an algorithm that evaluates the weight of edges globally and locally without the necessity of additional MR jobs.

The GWNP emerges as a possible pruning algorithm to be applied in the Metablocking step in order to achieve satisfactory effectiveness results without significant impacts on the efficiency of the blocking technique. The Metablocking step applies two *map* tasks and one *groupBy* task, as shown in Figure 5.1. The first *map* task receives the key-value pairs provided by the Pre-processing step. For each block (key), all entities contained in the set of entities (value) are compared following the Cartesian product. The comparison between entities applies weighting schemes (e.g., $Jaccard(e_i, e_j, B) = \frac{\|B_{ij}\|}{\|B_i\|+\|B_j\|-\|B_{ij}\|}$) to define the weight of the edge that links the entities. The weighting scheme receives as input the pair of entities $\langle e_i, e_j \rangle$ and the block collection B to estimate the corresponding weight. Thus, considering the example, the weight of edge for the entity pair $\langle e_1, e_2 \rangle$ is $\frac{2}{3}$, since

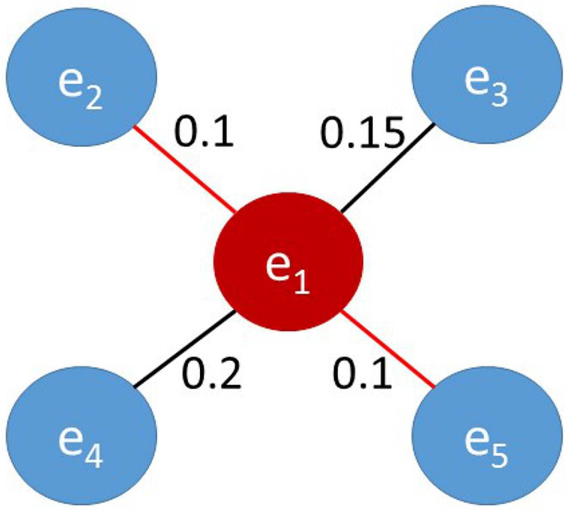

Figure 5.2: WNP pruning algorithm in context of low weight edges.

e_1 is contained in three blocks ($\{b_1, b_2, b_3\}$), e_2 is contained in two blocks ($\{b_1, b_2\}$), and both entities have two blocks in common ($\{b_1, b_2\}$). Furthermore, for each entity pair to be compared, the *map* task emits two key-value pairs $\langle e_i, e_j.w_{ij}\rangle$ and $\langle e_j, e_i.w_{ij}\rangle$, where w_{ij} is the weight of the edge.

To avoid redundant comparisons, only entity pairs that satisfy the MaCoBI condition are compared. Therefore, the entity pair $\langle e_1, e_2\rangle$ will not be compared in block b_2, since the marked block in common is b_1 ($MB_1 \cap MB_2 \neq \emptyset$). Moreover, the entity pair $\langle e_1, e_2\rangle$ is already compared in the block b_1. During the comparisons between entities, the amount of edges and the sum of edge weights are stored at two accumulator variables. This information will be used to determine the global threshold. The global threshold is given by the average of edges weight, which is stored in the accumulator variables. The value of the global threshold is sent to the second *map* task through a broadcast variable.

In the *groupBy* task, all key-value pairs that share the same key e are grouped. Therefore, all entities linked to e (key) are grouped in the same set of values. As output, the *groupBy* task emits key-value pairs in the following format: $\langle e, E\rangle$, where E is the set of neighbouring entities (of e) with their respective edge weights. Thus, in Figure 5.1, the pairs $\langle e_1, e_2.\frac{2}{3}\rangle$, $\langle e_1, e_3.\frac{1}{3}\rangle$ and $\langle e_1, e_4.\frac{1}{3}\rangle$ are grouped to generate the pair $\langle e_1, \{e_2.\frac{2}{3}, e_3.\frac{1}{3}, e_4.\frac{1}{3}\}\rangle$.

The second *map* task prunes the edges according to a weight threshold. To define the threshold, the GWNP pruning algorithm takes into account the weight of edges locally and globally. The global threshold (GT), which was generated in the first *map* task, is provided by the broadcast variable. To determine the local threshold (LT), the *map* task evaluates the key-value pairs, provided by the *groupBy* task. For each key-value pair $\langle e, E\rangle$, the local threshold for the entity e is given by the average of the edge weights of all neighbouring entities (contained in E). In order to define the weight threshold (WT), we combine the global and local threshold according to the following formula: $WT = (\alpha \cdot LT) + ((1 - \alpha) \cdot GT)$, where $\alpha \in [0, 1]$ defines the influence factor of each threshold. Thus, for the pair $\langle e_1, \{e_2.\frac{2}{3}, e_3.\frac{1}{3}, e_4.\frac{1}{3}\}\rangle$, the local threshold is $\frac{4}{9}$, i.e., $LT = 0.44$. Assuming that the $GT = 0.46$ and $\alpha = 0.7$, the weight threshold is given by $WT = (0.7 \cdot 0.44) + (0.3 \cdot 0.46) = 0.45$. Therefore, only the pair $\langle e_1, e_2.\frac{2}{3}\rangle$ is emitted, since the edge weight (0.67) is higher than WT (0.45). In this sense, only the edges of the graph that have a weight above the weight threshold will not be discarded. Finally, only the pairs of entities that remain connected after pruning the graph will be compared in the ER task.

5.1.4 Cardinality-based Load Balancing Technique

Particularly, parallel-based techniques usually tackle load imbalancing problems [68], as described in Section 2.8. This problem directly influences the efficiency of the whole parallel-based process, since the slowest node dominates the overall execution time [9]. Therefore, to minimize the load imbalancing problem, we propose the *Cardinality-based* load balance technique.

Since the load imbalancing occurs in the first *map* task of the Metablocking step [37], the *Cardinality-based* load balance technique is applied before this step in order to provide balanced inputs. To guide the even distribution of comparisons (between entities) among the nodes, the *Cardinality-based* technique takes into account the amount of comparisons (cardinality) to be performed in each block. For this purpose, the *Cardinality-based* technique reads each pair $\langle k_b, E\rangle$ (Step 1) and calculates the amount of comparisons ($||E||$) for the block k_b. After calculating the amount of comparisons to be performed in each block k_b, the blocks are sorted by the amount of comparisons in descending order. Then, a greedy algorithm evaluates the amount of available nodes and defines to which node the entity pairs (in E) of the block k_b will be emitted.

First, n partitions are defined, where n is the number of nodes available in the computing infrastructure. Then, the greedy algorithm removes the first block (with the highest amount of comparisons) and allocates the block to the emptiest partition. This step continues until all blocks have been allocated. An allocation table T is generated as output, where each line follows the format $\langle k_b, idPartition\rangle$. The allocation table is shared among the nodes of the cluster through a broadcast variable. Finally, the output of the Pre-processing step ($\langle b, S\rangle$) is partitioned based on the allocation table. Thus, each $\langle b, S\rangle$ will be emitted to the Spark partition according to the $idPartition$, where $k_b = b$ in the allocation table. This allocation evenly distributes the amount of comparisons among the available nodes of the cluster.

For instance, considering the example illustrated in Figure 5.1, the blocks b_1, b_2, b_3 and b_4 perform 1, 3, 12 and 20 comparisons, respectively. Sorting the blocks by their cardinalities (amount of comparisons), the allocation table is generated: $T = \{b_4 = 20, b_3 = 12, b_2 = 3, b_1 = 1\}$. Supposing that there are two available nodes, the blocks are allocated to two partitions. After applying the greedy algorithm, the table T is modified: $T = \{b_4 = 1, b_3 = 2, b_2 = 2, b_1 = 2\}$. Thus, the entity pairs belonging to block b_4 are allocated to the first partition whilst the other entity pairs are allocated to the second partition. After the partitioning step, each partition is sent to its respective node and the two nodes will perform 20 and 16 comparisons, respectively.

5.2 Experimental Evaluation

In this section, we evaluate the proposed spark-based technique and the application of the *Cardinality-based* load balancing technique, using a cluster infrastructure. The experiments address effectiveness and efficiency issues. We run our experiments on a cluster infrastructure with 16 nodes, each one with one core. Each node has an Intel(R) Xeon(R) 1.0GHz CPU, 2GB memory, runs the 64-bit Debian GNU/Linux OS with a 64-bit JVM and Apache Spark 2.0. Since the cluster has a limited amount of memory, to avoid the burst of memory, we employed an excerpt of the token blocking output of the *DBpedia* ($D_{dbpedia}$) data source [37]. This excerpt contains 1,189,424 blocks, 59,431,686 entity pairs to be compared and it is possible to identify at most 749,773 correspondences. In the experiment regarding the load balancing, we simulated the execution of the load balancing techniques applying all the token blocking output of the $D_{dbpedia}$ data source. The output contains 1,239,424 blocks, 42,310,902,031 entity pairs to be compared and 892,579 correspondences.

5.2.1 Efficiency

In this experiment, we evaluated the efficiency of the SS-Metablocking technique applying the GWNP pruning algorithm and the *Cardinality-based* load balancing technique against: i) the same SS-Metablocking technique applying the WNP pruning algorithm (with and without the *Cardinality-based* technique); and ii) the Hadoop-based technique [37] applying the WNP pruning algorithm. The execution time results are given by the average of three executions of each blocking technique. To execute the Hadoop-based technique, we configured the cluster infrastructure (i.e., the same node configurations) with Hadoop 2.6.0, with one *map* and one *reduce* task per node since each node has one core.

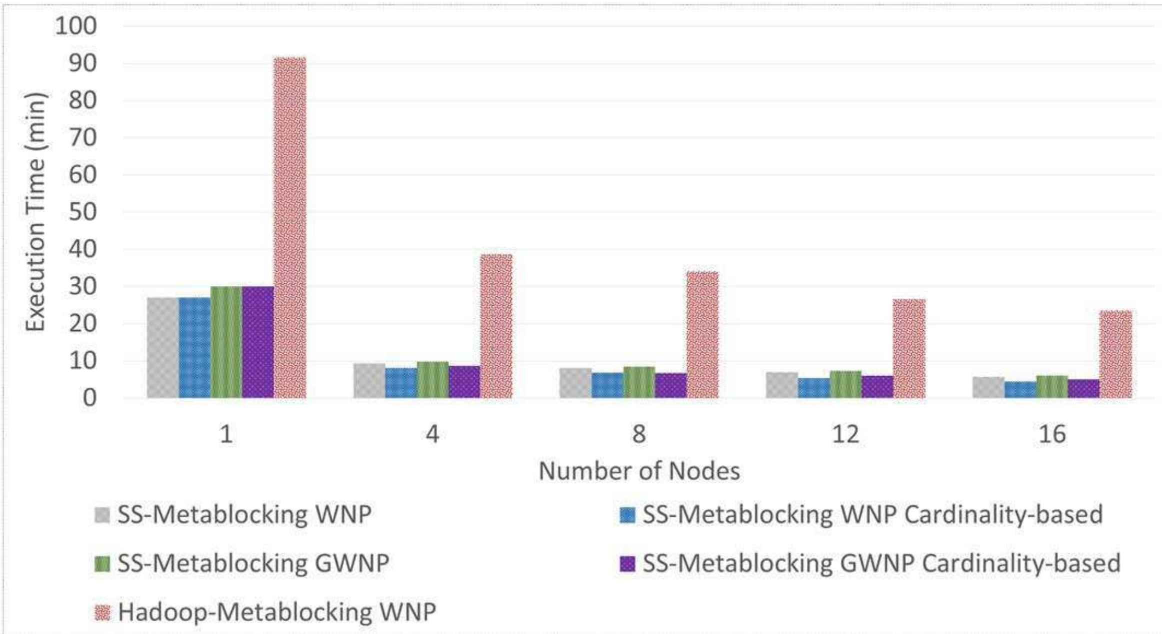

Figure 5.3: Execution time of SS-Metablocking and Hadoop-based Metablocking techniques.

Figures 5.3 and 5.4 illustrate the results of the comparative efficiency analysis. Regarding execution time (Figure 5.3), the SS-Metablocking technique with the *Cardinality-based* technique outperformed the SS-Metablocking (without the *Cardinality-based* technique) and Hadoop-based Metablocking techniques for all variations of the number of nodes. Although the SS-Metablocking with GWNP algorithm presents greater execution times than

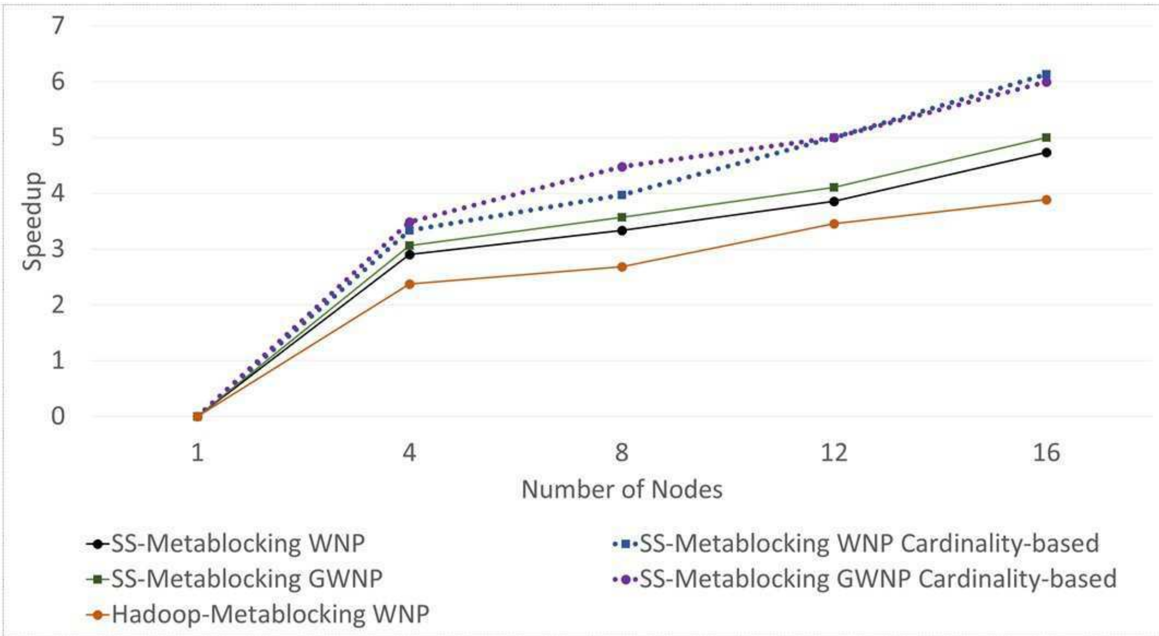

Figure 5.4: Speedup of SS-Metablocking and Hadoop-based Metablocking techniques.

SS-Metablocking with WNP algorithm, SS-Metablocking with GWNP achieves better effectiveness results, as shows the effectiveness experiments.

To measure the efficiency of the blocking technique, the speedup metric was used, in addition to the technique execution time. Speedup measures how fast a process runs in parallel when compared to the time of the same process in sequential mode. In other words, speedup is given by the fraction between the time required to execute a process on a single node and the time required to execute the same process on n nodes, where n is the number of nodes. When the value of speedup is equal to the number of nodes (n), the speedup is considered ideal. However, the ideal speedup rarely occurs in real scenarios since to parallelize a process there are execution time additions. It occurs due to the time required to configure nodes and distribute (entity) comparisons between them.

Concerning speedup (Figure 5.4), the SS-Metablocking with the *Cardinality-based* technique has also obtained better results than the other two techniques. Based on these results, we can highlight that applying more than four nodes does not present significant improvements regarding the speedup results. Thus, four is the ideal number of nodes for this experiment configuration (i.e., data source size and nodes configuration). Based on the experimental results and the *T-Student* test (with 95% of confidence), we concluded that the SS-

Metablocking technique (with and without the *Cardinality-based* technique) has achieved a better efficiency when compared to the state-of-the-art parallel blocking technique (i.e., Hadoop-based technique).

5.2.2 Effectiveness

This experiment evaluated the effectiveness of the SS-Metablocking approach when the GWNP or WNP pruning algorithms are applied. In this sense, three effectiveness metrics described in Chapter 2 were applied: *Pairs Completeness* (*PC*), *Pairs Quality* (*PQ*) and *Reduction Ratio (RR)*.

As described in Section 5.1, the GWNP algorithm applies a global threshold to minimize the amount of comparisons between entities to be performed in the ER task. Therefore, the pruned graph generated by GWNP results in fewer comparisons than the pruned graph of WNP algorithm. In this sense, we can highlight the better results presented by the PQ and RR metrics without decreasing the value of PC, as illustrated in Table 5.1. The most rigorous pruning proposed in GWNP provides a reduction of more than 4,000,000 comparisons when compared with the WNP algorithm. Thus, GWNP achieved better results regarding the PQ and RR metrics due to the both metrics are directly influenced by the amount of comparisons resulted by the pruned graph. Although the value of PC has not been changed, GWNP can impact negatively in this metric since it can discard entity pairs which are considered correspondences. Based on the results, we can infer that the GWNP algorithm improves the efficiency of the ER task since it minimizes the amount of comparisons to be performed.

Table 5.1: Effectiveness results of the pruning algorithms.

Pruning Algorithm	**PC**	**PQ**	**RR**
WNP	0.99	0.03	0.59
GWNP	0.99	0.04	0.66

5.2.3 Load Balancing

This experiment evaluated the load balancing, regarding the amount of comparisons performed at the nodes, for *Cardinality-based* and *MaxBlock* techniques. To this end, we executed both techniques for two inputs: i) the excerpt of the token blocking output from $D_{dbpedia}$ and ii) all the token blocking output from $D_{dbpedia}$. To evaluate the load imbalancing for the input ii), we performed only the allocation of the comparisons to the nodes, without comparing the entity pairs (to avoid the burst of memory). We calculated the standard deviation of the amount of comparisons performed in each node to measure the load imbalancing among the nodes of the distributed infrastructure.

The results of the comparative analysis are shown in Figure 5.5. To smooth the curves and facilitate the understanding, the standard deviation values are plotted in log scale (base 10). Although *MaxBlock* generates several partitions with a similar amount of comparisons, it does not consider the amount of available nodes. Consequently, the distribution of the partitions among the nodes can cause imbalance. On the other hand, the *Cardinality-based* technique evaluates the cardinality of the blocks and evenly distributes the entity pairs among the nodes of the distributed infrastructure.

The standard deviation for the *Cardinality-based* technique is less than one for both outputs of $D_{dbpedia}$ (excerpt and full) and all amounts of nodes. In more detail, the *Cardinality-based* technique provides a load balancing mechanism in which the amount of comparisons to be performed in each node differs by one comparison in the worst case. However, the *MaxBlock* technique presents high values of standard deviation since the partitions were not distributed evenly among the nodes. Therefore, we can infer that the *Cardinality-based* technique provides a better load balancing than the *MaxBlock* technique.

5.3 Final Considerations

In this chapter, a blocking technique based on Metablocking in parallel was proposed from the Spark application. The proposed blocking technique (i.e., SS-Metablocking) applies the GWNP pruning algorithm in order to provide efficiency to the ER task since this pruning algorithm reduces the amount of comparisons to be performed on the ER task. Furthermore, the research described in this chapter proposes a load balancing technique based on the cardinality of the blocks. This load balancing technique aims to improve the efficiency of SS-Metablocking by providing a better utilization of the computational infrastructure resources while avoiding the load imbalance problem. The contributions proposed in this

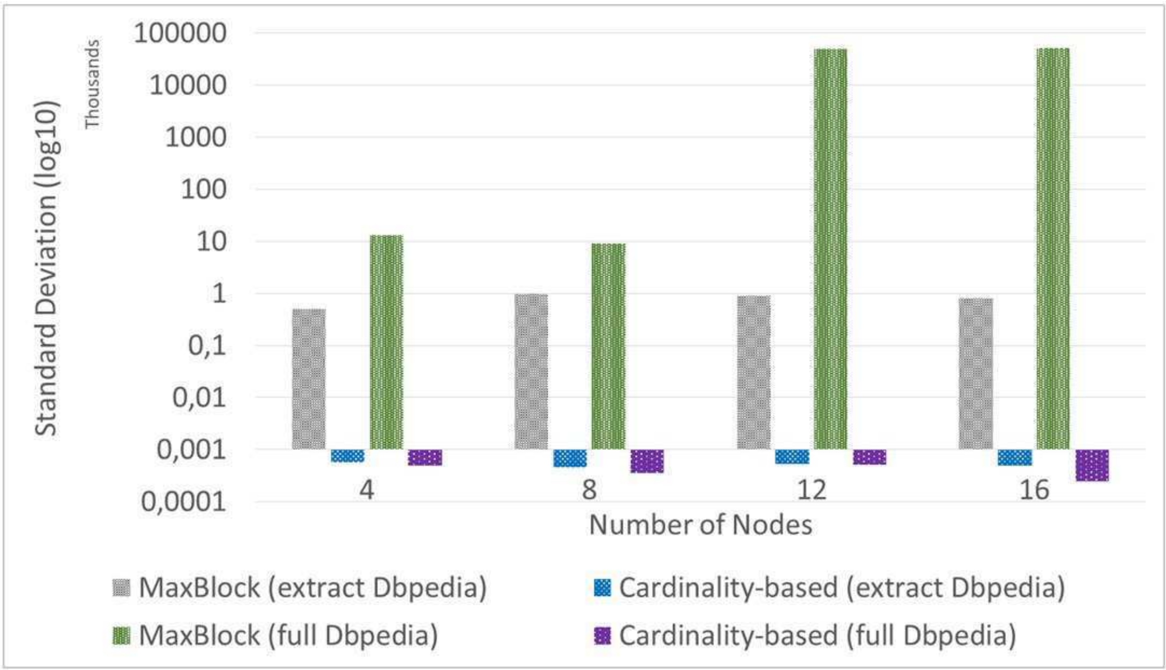

Figure 5.5: Standard deviation of the load balancing techniques.

research are directly related to the *block generation* and *pruning of comparisons* steps of the execution model proposed in Chapter 4. Based on the results obtained from the experimental evaluation, it is possible to highlight that the SS-Metablocking technique achieved better results in terms of efficiency and effectiveness than the state-of-art blocking technique (i.e., Hadoop-based Metablocking). In the next chapter, another Metablocking-based blocking technique will be proposed, called NA-BLOCKER. NA-BLOCKER is considered a noise-tolerant technique since it is able to handle noisy data.

Chapter 6

A Noise Tolerant and Schema-agnostic Blocking Technique

As described in previous chapters, a fundamental step to integrate multiple knowledge bases or identifying similar entities is the application of the ER task. In the context of semi-structured data, the ER task addresses challenges related to data variety since different formats are used to represent entity profiles (heterogeneous data) [36]. Furthermore, regarding challenges inherent in the ER task (discussed in Chapter 2), it is possible to highlight another problem faced by ER task: noisy data (described in Section 2.2), such as pronunciation errors, spelling errors, or typos in the attribute values of the entities [62]. In practice, noise can impair data interpretations, data manipulation tasks and decision-making processes [43; 88]. In the context of ER task, noisy data has a direct impact on the identification of similar entities, since the difference in the spelling of their attributes may determine that two truly similar entities in the real world are not considered similar. In this chapter, we will consider the two most common types of noise in the data: typos and misspellings [3].

Considering scenarios involving semi-structured data, schema-agnostic blocking techniques should be applied. Among the schema-agnostic blocking techniques, those capable of exploring possible schema information appear as the most promising in terms of effectiveness [89]. Such techniques extract information related to the schema based on data (i.e., statistics collected directly from the data) in order to improve the quality of the generated blocks. Among them, the BLAST blocking technique [89] emerges as the state-of-the-art technique. It is important to highlight that the information related to the schema can be applied to Metablocking-based techniques in order to further improve the quality of the generated blocks. However, the presence of noise in attribute values (noisy data) directly compromises the effectiveness of blocking techniques, since these values are used to extract schema information and to generate blocks.

In this chapter, the Noise-aware Agnostic BLOCKing for Entity Resolution (NA-BLOCKER[1]) technique is proposed. Although the NA-BLOCKER technique is not performed in parallel, it is possible to highlight that this blocking technique covers the steps of *schema information extraction*, *block generation* and *pruning of comparisons* of the execution model described in Chapter 4. Notice that NA-BLOCKER addresses these steps because strategies for extracting schema information are proposed, in addition to applying a Metablocking-based block.

The contribution of this chapter refers to propose a novel schema-agnostic blocking technique capable of tolerating noisy data to extract information regarding the schema from the data (i.e., group similar attributes based on the data) and enhance the quality of the generated blocks. To this end, the NA-BLOCKER applies algorithms in order to hash the attribute values of the entities 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.

Regarding the hash algorithm, NA-BLOCKER applies the Locality-Sensitive Hashing (LSH) algorithm. As described in Section 2.9, such algorithm can be applied to reduce the dimensionality (i.e., the amount of attribute values) of the objects, preserving the similarity between objects and significantly reducing the number of attribute values (or tokens) to be evaluated. Therefore, for each attribute of an entity, a hash function (for example, MinHash [66]) converts the value of the attribute into a probability vector called signature (MinHash signature). In this sense, the hash function is applied to avoid the quadratic complexity of comparing all tokens derived from attribute values.

Since the hash function preserves the similarity of the attribute values, it is possible to apply distance functions (e.g., Jaccard) to determine the similarity between attribute values of two distinct entities [5]. In the context of schema-agnostic blocking, the hash function can generate similarity vectors that would guide the block generation. Therefore, entities with

[1] https://bitbucket.org/tbrasileiro/matchingunstructureddata

similar vectors will be inserted into the same block. To improve efficiency, signatures generated by the hash function can replace the tokens used by token-based blocking techniques. This strategy will be further discussed throughout this chapter.

As previously discussed, the first contributions were focused on the impact of LSH on the effectiveness of the generated blocks. In this sense, although the NA-BLOCKER technique was not performed in parallel, the parallelization of the technique was planned for future researches. Thus, NA-BLOCKER in parallel would appear as an efficient version (i.e., addressing the efficiency problems) of the technique proposed in this chapter. However, the authors of [90] proposed the parallelization of the BLAST technique [89]. Moreover, the work [90] proposed improvements in terms of effectiveness. For this reason, we decided to focus on researches related to challenges involving streaming data and incremental processing (to be discussed in Chapter 7) since it appears as an open area to explore [81]. In turn, as described in Chapter 4, the distributed execution model proposed in our research can host different blocking techniques since these techniques follow at least one of the proposed workflows. For this reason, it is important to highlight that the blocking technique proposed in [90] is compatible with the proposed execution model. Therefore, BLAST in parallel can be included in our model in order to address scenarios involving noisy data (in batch) with focus on efficiency (e.g., large data sources).

6.1 NA-BLOCKER: Noise-aware Schema-agnostic Blocking for Entity Resolution

In this section, the NA-BLOCKER blocking technique is described in detail as well as its workflow. The main objectives of NA-BLOCKER are: efficiently extract schema information from attribute values present in entities without user interference (e.g., review from a specialist) or the application of thesaurus (lack of thesaurus for a wide number of domains [21]); and generate high-quality blocks even in the presence of noisy data. Overall, NA-BLOCKER is divided into three steps: i) schema information extraction, ii) block generation, and iii) pruning, as depicted in Figure 6.1.

The proposed technique receives as input two data sources D_1 and D_2. Each data source is an entity collection $D = \{e_1, e_2, e_3, ..., e_n\}$, such that n is the number of entities in D.

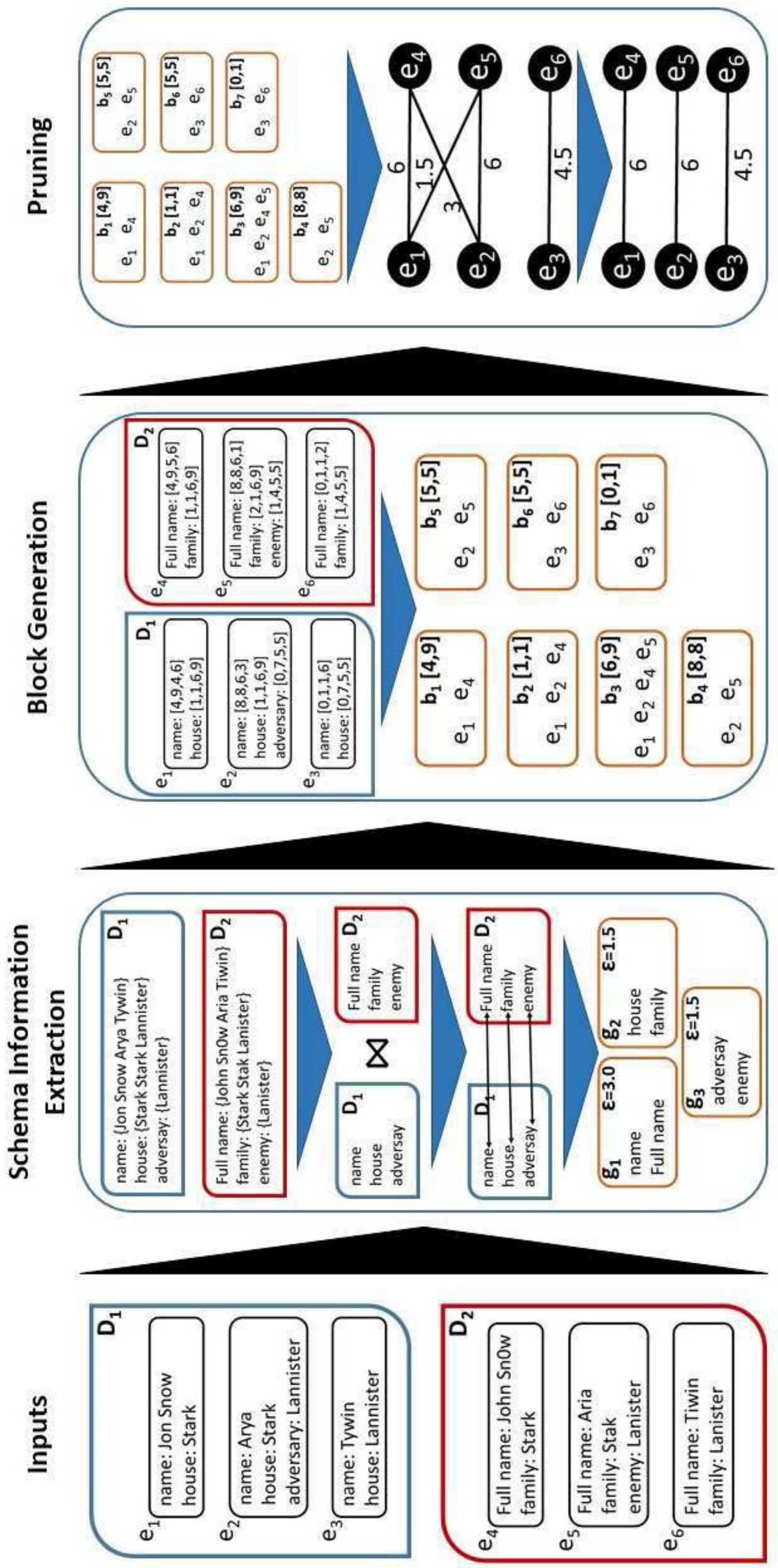

Figure 6.1: An instance of the NA-BLOCKER blocking technique workflow.

The attributes contained in each data source are denoted by $A(D) = \{a_1, a_2, \cdots, a_k\}$, such that k is the number of attributes in D. Notice that the value of k may vary for each data source. Since the entities can follow different loose schemas, each entity $e \in D$ has a specific attribute set and a value associated to each attribute, denoted by $A_e = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \langle a_3, v_3\rangle, ..., \langle a_k, v_k\rangle\}$, such that k is the maximum amount of attributes associated with e.

In the schema information extraction step, all attributes associated to the schemes of the entities belonging to D_1 and D_2 are extracted. Moreover, all values associated to the same attribute (a_i) are grouped into a set V_{a_i}, i.e., $V_{a_i} = \bigcup_{e \in D}(v \mid \langle a_i, v\rangle \in A_e)$. In turn, the pair $\langle a_i, V_{a_i}\rangle$ represents the set of values associated to a specific attribute a_i. The attributes present in D_1 and D_2 are grouped based on the similarity between their attribute values, denoted by Equation 6.1. The sets V_{a_i} and V_{a_j} are considered similar if $sim(V_{a_i}, V_{a_j}) \geq \Phi$, where $sim(V_{a_i}, V_{a_j})$ calculates the similarity between the sets V_{a_i} and V_{a_j} and Φ is a given threshold[2].

$$\begin{aligned} G(D_1, D_2) = \{g_1, g_2, g_3, ..., g_m\} \mid \forall g \in G(D_1, D_2) :& \\ g \subseteq (A(D_1) \times A(D_2))\, and\, \forall g \in G(D_1, D_2)& \\ \forall \langle a_i, a_j\rangle \in g : V_{a_i} \simeq V_{a_j}& \end{aligned} \tag{6.1}$$

In the block generation step, each set of attribute values V (associated with an attribute a) is converted into a hash-signature S (provided by LSH), given by $hash(\langle a, V\rangle) = \langle a, S\rangle$. To compute the similarity of all possible pairs of attributes, the process takes an overall time complexity of $\mathcal{O}(|U_{D_1}| \cdot |U_{D_2}|)$, such that $U_{D_1} = \bigcup_{a_i \in A(D_1)}(V_{a_i} \mid V_{a_i} \in \langle a_i, V_{a_i}\rangle)$ and $U_{D_2} = \bigcup_{a_j \in A(D_2)}(V_{a_j} \mid V_{a_j} \in \langle a_j, V_{a_j}\rangle)$. However, this time complexity is impractical for semi-structured data that appear on the Web, since data sources can commonly have hundreds of attributes and millions of attribute values [89]. For this reason, the LSH technique, which has a linear cost in relation to the set size [97], is applied to reduce the dimensionality of these sets, i.e., U_{D_1} and U_{D_2}, aiming to minimize the time complexity to a linear cost.

The set of LSH-signatures S (from $\langle a, S\rangle$) guides the block generation, since entities with a similar LSH-signature are grouped into the same block. The loose-schema information (i.e., $G(D_1, D_2)$) is applied to the block generation step in order to avoid that similar LSH-signatures originated from attributes with different semantics (due to the fact that the

[2]In this work, we apply a threshold value equal to 0.35 ($\Phi = 0.35$), since the experiments in [89] suggested this threshold value as the best one.

attributes are not in the same g) being inserted into the same block by the blocking technique. The output of the block generation step is a collection of blocks B, as denoted by Equation 6.2.

$$
\begin{aligned}
& B = \{b_1, b_2, b_3, ..., b_x\} \mid \\
& \quad \forall b \in B : (e_1 \in b \wedge e_2 \in b) \Leftrightarrow \\
& \quad (\exists \langle a_1, a_2 \rangle \in (A(D_1) \times A(D_2)) : \\
& \quad \langle a_1, a_2 \rangle \in \bigcup_{g \in G(D_1, D_2)} g \ \wedge \ hash(\langle a_1, v_1 \rangle \in A_{e_1}) \sim \\
& \quad hash(\langle a_2, v_2 \rangle \in A_{e_2}))
\end{aligned}
\tag{6.2}
$$

Finally, in the pruning step, Metablocking is applied in order to discard comparisons between entities with low-weight edge (i.e., low similarity). In this sense, the collection B provided by the block generation step is restructured relying on the intuition that the more blocks two entities share, the more likely they result in a correspondence. Then, the output of the pruning step is a restructured collection of blocks B' with significantly fewer comparisons. Next, we describe in details each step of the NA-BLOCKER technique.

6.1.1 Step 1: Schema Information Extraction

This step receives as input two data sources D_1 and D_2. As described in Algorithm 1, for each entity, the attribute values are read in order to extract the tokens (only the relevant words) associated with each attribute (illustrated by the function *extractSignatures*, lines 13 to 29). In other words, punctuation, special characters (e.g., @, $, *, and &) and stop words (e.g., *the, for, to, at, which*, and *on*) are removed from the attribute values. Thus, a set of values (keywords) (V), extracted from the attribute values, is associated to each attribute ($\langle a, V \rangle$) of an entity (lines 15 to 24). The transformation of the attribute values is important in order to remove characters or words that can degrade the similarity between attributes. To perform the attribute-match induction (i.e., group the similar attributes), the sets of keywords are clustered (merged) according to the attributes (lines 19 and 21). For instance, in Figure 6.1, all keywords of the attribute *name* are merged into the same set of values. The sets of values are used to measure the similarity between the attributes. In this sense, given two attributes and their respective sets of keywords, the Jaccard[3] function evaluates the similarity between the sets of values and, consequently, determines the similarity values between the attributes.

To enhance the efficiency of the attribute-match induction (i.e., avoid the quadratic complexity of comparing all possible attribute values in this step), the LSH algorithm is applied [5]. As a result, the hash function (MinHash, in our case) generates a signature (*MinHash* signature) for each set of values $\langle a, V \rangle \rightarrow \langle a, S \rangle$, where S denotes the *MinHash* signature (lines 25 to 28). Since the hash function preserves the similarity between the sets of values, it is possible to apply a similarity function (e.g., Jaccard) to the signatures in order to determine the similarity between the attributes (line 6). It is important to highlight that computing the similarity (e.g., applying Jaccard) between attributes based on the signatures is faster than computing it based on the sets of keywords, since the quadratic cost is avoided. This fact motivates the application of LSH in the first two steps of NA-BLOCKER.

In order to perform the attribute-match induction, the similarities between the attributes are evaluated. Therefore, attributes with high similarity are inserted into the same group g (lines 4 to 11). For example, based on the similarities of the sets of keywords, the attribute groups $g_1 = \{name, Full\ name\}, g_2 = \{family, house\}$ and $g_3 = \{adversary, enemy\}$ are built, as illustrated in Figure 6.1.

Moreover, this step calculates the entropy of each attribute (line 7). As described in Section 2.10, the entropy of an attribute indicates how significant is the attribute, i.e., the higher the entropy of an attribute, the more significant is the observation of a particular value for that attribute [89]. We specifically apply here the Shannon entropy [63] to represent the information distribution of a random attribute.

Thereafter, the aggregated entropy ε is generated from the entropy of each attribute contained in a particular attribute group (line 8). Thus, in Figure 6.1, the entropies ε associated with each attribute group are: $\langle g_1, 3.0\rangle, \langle g_2, 1.5\rangle$, and $\langle g_3, 1.5\rangle$. The entropy of an attribute group influences the weighting of the blocking graph, as will be discussed in the next steps. Finally, since in this step we need to evaluate all possible attributes contained in our data sources, namely, $A(D_1)$ and $A(D_2)$, and the cost to compare the attributes (given by $\langle a, S\rangle$) is linear in relation to the size of S, the time complexity of this step is expressed

[3] $Jaccard(a_1, a_2) = \frac{|V_1 \cap V_2|}{|V_1| + |V_2| - |V_1 \cap V_2|}$.

as $\mathcal{O}(|A(D_1)| \cdot |A(D_2)| \cdot |S|)$.

6.1.2 Step 2: Block Generation

In this step, the inputs are the entities (provided by D_1 and D_2) and the attribute groups generated by the previous step (with their respective entropies), as illustrated in Algorithm 2. Initially, the entities are read and, for each attribute of a particular entity, the LSH algorithm is applied to generate a signature for the attribute value (lines 2 to 5 and 15 to 18). Then, an entity *e* is denoted as follows: $e = \{\langle a_1, S_1\rangle, \langle a_2, S_2\rangle, \langle a_3, S_3\rangle, ..., \langle a_k, S_k\rangle\}$. For instance, to generate the attribute signatures of entity e_1, in Figure 6.1, the attribute values $\langle a_{name},\ Jon\ Snow\rangle$ and $\langle a_{house},\ Stark\rangle$ are converted into the signatures $\langle a_{name}, [4, 9, 4, 6]\rangle$ and $\langle a_{house},\ [1, 1, 6, 9]\rangle$, respectively. Notice that the hash function always generates signatures with the same size, in terms of elements.

Once the signatures are defined, the blocks are generated based on each of the signatures. To this end, the signatures are split into α equal parts, termed subsignatures s. The subsignatures are used as the key for each block to be generated (lines 5 and 18). Thereafter, the entities that share a particular subsignature originated from attributes contained in the same attribute group (provided by the previous step) are inserted into the same block (lines 6 to 12 and 19 to 26). Furthermore, since each group of attributes has an associated entropy, the blocks generated from attributes contained in a particular group will assume the same entropy value of the attribute group.

Regarding the time complexity, in this step, it is necessary to evaluate all $\langle a, S\rangle$ (derived from A_e, defined in Section 3) of each entity $e \in D$ to generate the blocking keys. Furthermore, to evaluate each $\langle a, S\rangle$, it is necessary to take into account the number of subsignatures s derived from S, denoted by α. Therefore, the time complexity of this step is $\mathcal{O}((||A_{D_1}|| + ||A_{D_2}||) \cdot \alpha)$, where $||A_{D_1}||$ is given by $\sum_{e \in D_1} |A_e|$ and $||A_{D_2}||$ is given by $\sum_{e \in D_2} |A_e|$.

In Figure 6.1, the NA-BLOCKER technique generates two blocks (b_5 and b_6) with the key *[5, 5]* since one subsignature is provided by the attributes $house/\ family$ ($\in g_2$) and the other one by the attributes $adversary/enemy$ ($\in g_3$). In this sense, the entities e_2 and e_5 are inserted into b_5 since both entities contain the subsignature *[5, 5]* and the attributes $adversary$ (in e_2) and $enemy$ (in e_5) are contained in the same attribute group (g_3). Simi-

Algorithm 1: Schema Information Extraction step

Data: D_1, D_2 and Φ: the input data sources and the similarity threshold

Result: G: attribute groups

```
attribSignaturesD1 ← extractSignatures(D1);
attribSignaturesD2 ← extractSignatures(D2);
attribGroup ← ∅;
foreach a1 in attribSignaturesD1 do
    foreach a2 in attribSignaturesD2 do
        if sim(a1.S, a2.S) > Φ then
            ε ← shanonEntropy(a1, a2);
            attribGroup.append(⟨a2, a1, ε⟩);
        end
    end
end
return attribGroup
Function extractSignatures(D) : map⟨a, S⟩ do
    attribSignatures ← ∅;
    foreach e in D do
        foreach a in A_e do
            V ← keyWords(a_value);
            if attribSignatures.contains(a) then
                attribSignatures.get(a).union(V);
            else
                attribSignatures.put(a, V);
            end
        end
    end
    foreach ⟨a, V⟩ in attribSignatures do
        S ← minHash(V);
        attribSignatures.replace(a, S);
    end
    return attribSignatures
end
```

larly, the entities e_3 and e_6 are inserted into b_6 because both entities contain the subsignature *[5, 5]* and the attributes $house$ (in e_3) and $family$ (in e_6) are contained in g_2. It is important to highlight that, even though there exists noisy data in data source D_2 (e.g., "John Sn0w", "Stak", "Aria", "Lanister", and "Tiwin") the entities with similar attributes have been inserted into the same blocks. This occurs due to the fact that the NA-BLOCKER technique benefits from the application of LSH, which generates signatures of the attributes. The signatures maintain the degree of similarity between attribute values even though the set of keywords is transformed into an array of integers. Thereby, entities whose attribute values are similar, even with the presence of noisy data, present high chances to share several subsignatures and, consequently, will be inserted into the same blocks. On the other hand, if a token blocking technique is applied, entity pairs, such as $\langle e_2, e_5 \rangle$ and $\langle e_3, e_6 \rangle$, will not be inserted into the same block since they do not share the same tokens.

6.1.3 Step 3: Pruning

The goal of this step is to discard redundant comparisons between entities, as well as comparisons with few chances of resulting in correspondences, as illustrated in Algorithm 3. To this end, Metablocking-based techniques [80; 89; 36] are applied. These techniques receive as input the blocks generated at the previous step. In Figure 6.1, notice that the entities e_1 and e_4 are contained in blocks b_1, b_2 and b_3. Therefore, these entities should be compared three times (i.e., redundant comparisons). To avoid the redundant comparisons, the Metablocking technique restructures the input blocks into new ones that involve significantly fewer comparisons, while maintaining the original level of effectiveness [80]. Then, the input blocks are converted into a blocking graph (lines 2 to 15), such that each node represents an entity and each edge (between a pair of nodes) denotes that the pair of nodes sharing at least one block in common (lines 4 to 14). Every edge is associated with a weight, based on the number of blocks in common between the pair of nodes (entities) linked by the edge and the entropy value associated with the blocks in common (line 7).

Regarding the influence of entropy in the weight of edges, the entropy value determines the relevance of the blocks, since not all the blocks have the same importance. Therefore, the edge weight is given by the sum of entropies ε associated with each block in common between the pair of nodes linked by the edge (lines 9 and 11). For instance, in Figure 6.1, the

Algorithm 2: Block Generation step

Data: D_1, D_2, G: the input data sources and the attribute groups

Result: B: blocks of entities

```
mapOfBlocks ← ∅;
foreach e1 in D1 do
    foreach a1 in A_e1 do
        S ← LSH(a1.value);
        blockKeys ← splitSignature(S);
        foreach blockKey in blockKeys do
            if mapOfBlocks.contains(blockKey.a1) then
                mapOfBlocks.get(blockKey.a1).append(e1);
            else
                mapOfBlocks.put(⟨blockKey.a1, [e1]⟩);
            end
        end
    end
end
foreach e2 in D2 do
    foreach a2 in A_e2 do
        S ← LSH(a2.value);
        blockKeys ← splitSignature(S);
        attribOfD1InSameGroup ← attribGroup.get(a2);
        foreach blockKey in blockKeys do
            foreach a1 in attribInSameGroup do
                if mapOfBlocks.contains(blockKey.a1) then
                    mapOfBlocks.get(blockKey.a1)  .append(e2);
                end
            end
        end
    end
end
return mapOfBlocks
```

edge that links nodes e_1 and e_4 assumes the weight of 6, since the pair of entities shares three blocks: b_1 (from group $\langle g_1, 3.0\rangle$), b_2 (from group $\langle g_2, 1.5\rangle$), and b_3 (from group $\langle g_2, 1.5\rangle$). Thus, the edge weight, which links e_1 and e_4, is given by $3.0 + 1.5 + 1.5 = 6$.

Once the graph is built, it is pruned according to a pruning criterion, which eliminates low-weighted edges to skip part of the redundant comparisons. Regarding the pruning criteria, the works [80; 89] propose different pruning algorithms that can be applied in this step. Particularly, in this work, we apply the WNP-based pruning algorithm [80] since it has achieved better results than other competitors [36]. The WNP algorithm applies the node-centric pruning algorithm with a local weight threshold that is given by the average edge weight of each neighborhood.

Concerning the time complexity, in this step, the complexity is given by the sum of the cost to evaluate the entity pairs in each block $b \in B$ (i.e., the cardinality of B) and the cost of the WNP pruning algorithm. The time complexity of the WNP algorithm is $\mathcal{O}(|N_B| \cdot |E_B|)$ [80], where $|N_B|$ is the number of nodes and $|E_B|$ is the number of edges in the graph generated from B. Therefore, the time complexity of the pruning step is $\mathcal{O}(||B||)$ + $\mathcal{O}(|N_B| \cdot |E_B|)$.

In Figure 6.1, the pruning criteria evaluates locally the weight of the edges and discards the low-weighted comparisons (i.e., comparisons with few chances to result in correspondences). Hence, the resulting graph (pruned graph) infers only three comparisons: $\langle e_1, e_4\rangle$, $\langle e_2, e_5\rangle$, and $\langle e_3, e_6\rangle$. The significant decrease in the number of comparisons occurs due to the enhancing in the quality of blocks built in the Block Generation step (Step 2), which includes in each block only truly similar entities. Therefore, such step becomes fundamental in order to provide high-quality blocks, in terms of effectiveness, especially in the presence of noisy data, since the blocks built in the Block Generation step directly influence the edge weight of the graph (in Pruning step). The overview of the NA-BLOCKER technique, with the application of each step described previously, is summarized in Algorithm 4.

Algorithm 3: Pruning step

Data: G, B: the attribute groups and the blocks of entities

Result: B': pruned blocks

```
mapEntities ← ∅;
foreach block in mapOfBlocks do
    entities ← block.values;
    while entities.size > 1 do
        e_current ← entities.pop();
        foreach e in entities do
            ε ← attribGroup.getEntropy(block.key);
            if mapEntities.contains(e_current.e) then
                mapEntities.get(e_current.e).sumWeight(ε);
            else
                mapEntities.put(e_current.e, ε);
            end
        end
    end
end
B' ← WNP(mapEntities);
return B'
```

6.2 Experimental Evaluation

In this section, we evaluate the NA-BLOCKER[4] technique against BLAST[5] [89], the state-of-the-art method. Although the focus of the contributions proposed in this work is to improve the effectiveness of the generated blocks, the efficiency of the NA-BLOCKER technique is also evaluated. We run our experiments on a Windows 7 computer with 16GB of memory and Intel Core I7-4790 3.60 GHz. In our experimental evaluation, five real-world pairs of datasets[6] (provided by [89]) were used, as described in Table 7.1: i) Abt-Buy: prod-

[4] https://bitbucket.org/tbrasileiro/matchingunstructureddata

[5] http://stravanni.github.io/blast/

[6] Available at the project repository.

Algorithm 4: NA-BLOCKER

Data: D_1, D_2: input data sources

Result: B': set of pruned blocks

```
1 G ← SchemaInformationExtraction(D1, D2);
2 B ← BlockGeneration(D1, D2, G);
3 B′ ← Pruning(G, B);
4 return B'
```

Table 6.1: Datasets characteristics.

Pairs of datasets	$\|D_1\|$	$\|D_2\|$	Duplicates	$\|A_1\|$	$\|A_2\|$
Abt-Buy	1,076	1,076	1,076	3	3
Amazon-GP	1,354	3,039	1,104	4	4
DBLP-ACM	2,616	2,294	2,224	4	4
DBLP-Scholar	2,516	61,353	2,308	4	4
IMDB-DBpedia	27,615	23,182	22,863	4	7

uct profiles provided by abt.com and buy.com; ii) Amazon-GP: product profiles provided by amazon.com and google.com; iii) DBLP-ACM: scientific article profiles provided by dblp.org and dl.acm.org; iv) DBLP-Scholar: scientific article profiles provided by dblp.org and scholar.google.com; and v) IMDB-DBpedia: movie profiles provided by imdb.com and dbpedia.org. Table 7.1 shows the amount of entities (D) and attributes (A) contained in each dataset as well as the number of duplicates (i.e., matches) present in each pair of datasets.

To measure the effectiveness of the techniques, three quality metrics (described in Chapter 2) were applied: i) *Pair Completeness* (PC), ii) *Pair Quality (PQ)*; and iii) *F-score* (Fs). To evaluate the effectiveness results of the techniques in different scenarios of noisy data, we synthetically insert typos and misspellings (i.e., noise) into the attribute values of the entities contained in one dataset of each pair of datasets. In order to simulate the occurrence of typos/misspellings [62], for all attributes of an entity, one character of each token (i.e., relevant words) present in the attribute values is randomly exchanged by other characters, or additional characters are inserted into the tokens[7]. In this sense, we vary the level of noise in the dataset. The noise level ranges between 0 (i.e., no noise is inserted into the attribute values

[7] All data sources (with/without noise) are available at the project repository.

of any entity) to 1 (i.e., noise is inserted into the attribute values of all entities). For instance, the noise level of 0.4 indicates that 40% of the entities (contained in the first dataset) had their attribute values modified (i.e., noise was inserted). For these experiments, the execution time results are given by the average of three executions (of the blocking techniques) for each dataset pair.

Figures 6.2, 6.3, 6.4, 6.5 and 6.6 illustrate the results of the effectiveness comparative analysis for each pair of datasets. Regarding the effectiveness metrics (i.e., pair completeness, pair quality and F-Measure), NA-BLOCKER outperforms BLAST for all variations of noise level. It is important to highlight that, as the noise level increases, the effectiveness metrics decrease for both techniques. This decrease occurs due to the fact that the noise on the data negatively interferes on the block generation, as discussed in Section 6.1. However, the decrease in effectiveness metrics for BLAST occurs abruptly when compared to NA-BLOCKER. Since the latter applies strategies to tolerate noisy data, the effectiveness decrease is amortised. Even for a high level of noise (i.e., a noise level of 1.0), NA-BLOCKER has achieved a pair completeness greater than 60% for the first three pairs of datasets, as depicted in Figures 6.2, 6.3 and 6.4.

Regarding F-score (Fs), NA-BLOCKER reaches an average, with respect to all pairs of datasets, of 48% in the proportional decrease[8], whereas BLAST reaches 90%. The most significant result achieved by NA-BLOCKER was in terms of pair quality. In this case, it achieved an average pair quality (considering all pairs of datasets) two times better than the BLAST technique, in scenarios without noisy data. The main reason for that is the generation of multiple tokens per entity attribute (based on a particular attribute value) as blocking keys, in the BLAST technique. Since non-matching entities eventually share multiple tokens, they are included in the same block erroneously. On the other hand, the proposed technique generates a single hash value based on a particular attribute value. Thus, non-matching entities sharing the same hash value are harder to occur than non-matching entities sharing tokens in common. For this reason, NA-BLOCKER enhances the pair quality metric. Finally, based on the experimental results and the pair-wise (considering the level of noise) distribution *T-Student* test (with 95% of confidence), we concluded that our technique has achieved a

[8] $Proportional\ decrease = 1 - \frac{FM(noise=1.0)}{FM(noise=0.0)}$

better[9] effectiveness than the BLAST technique.

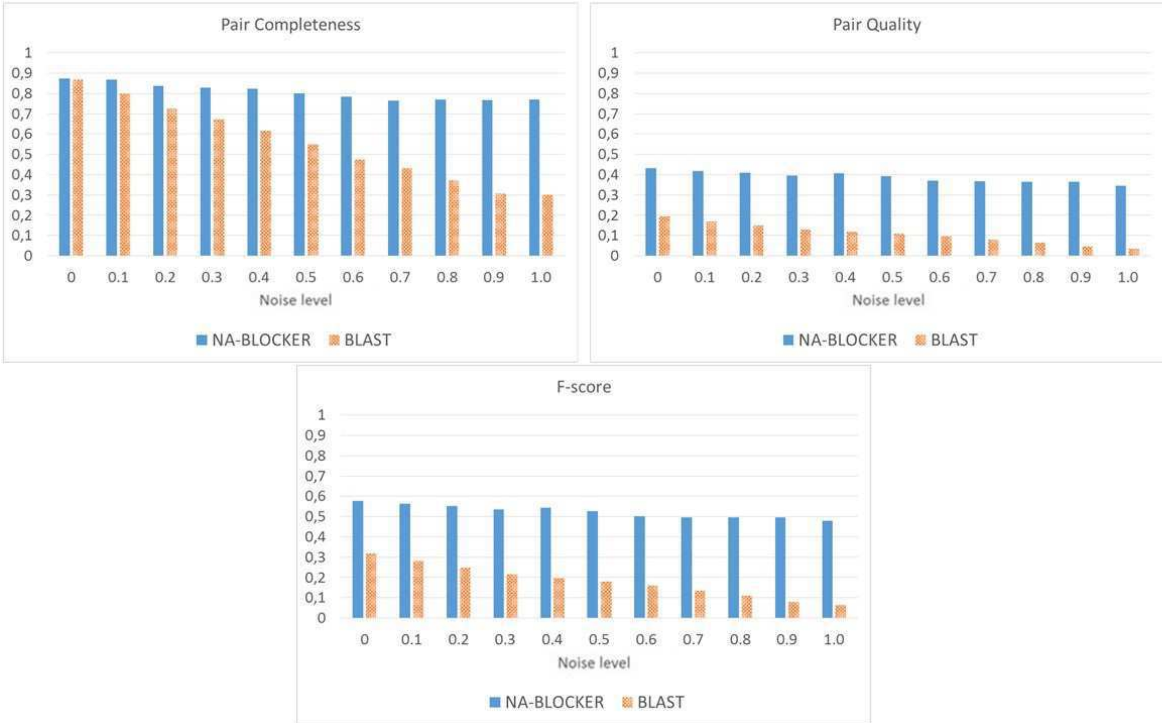

Figure 6.2: Effectiveness results for data sources Abt vs. Buy.

Regarding efficiency, the execution time (in seconds) of the NA-BLOCKER and BLAST techniques are evaluated for each pair of datasets, as depicted in Figure 6.7. BLAST achieves better results than NA-BLOCKER for all pairs of datasets. On average, the NA-BLOCKER increased the execution time around 30% (or seven seconds). These results are already expected, since the proposed technique requires more time to block the entities. This is due to the fact that our technique needs more time to generate the LSH-signatures and determine the similarity of their attribute values based on the approximate similarity.

On the other hand, it is important to highlight that NA-BLOCKER achieved better results compared to BLAST regarding the *aggregate cardinality* (i.e., $||B'||$) for all dataset pairs, as depicted in Figure 6.8. In other words, NA-BLOCKER requires less comparisons to be executed in the ER task. On average, the blocks generated by NA-BLOCKER indicate a total number of comparisons 34% (or more than seven thousand comparisons) less when compared to the blocks generated by BLAST. Thus, the efficiency results achieved by NA-BLOCKER may be compensated by efficiency gains generated by the execution of

[9]In the best case, NA-BLOCKER achieved an effectiveness 60% higher than BLAST.

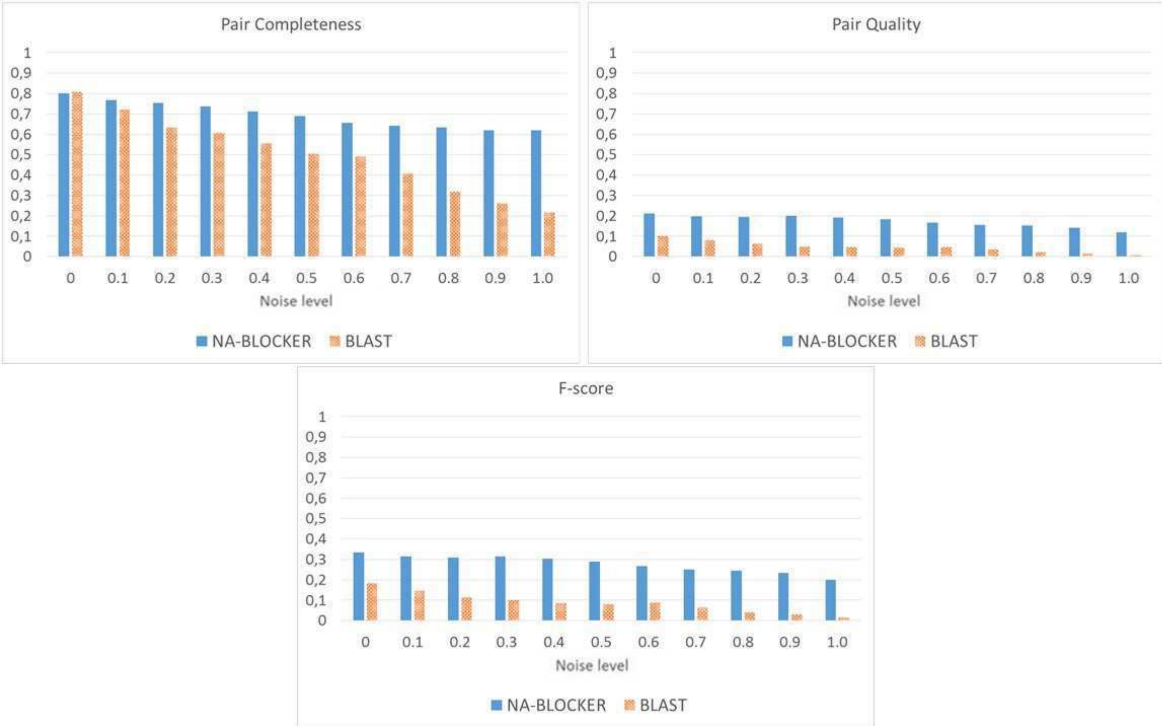

Figure 6.3: Effectiveness results for data sources Amazon vs. Google Product.

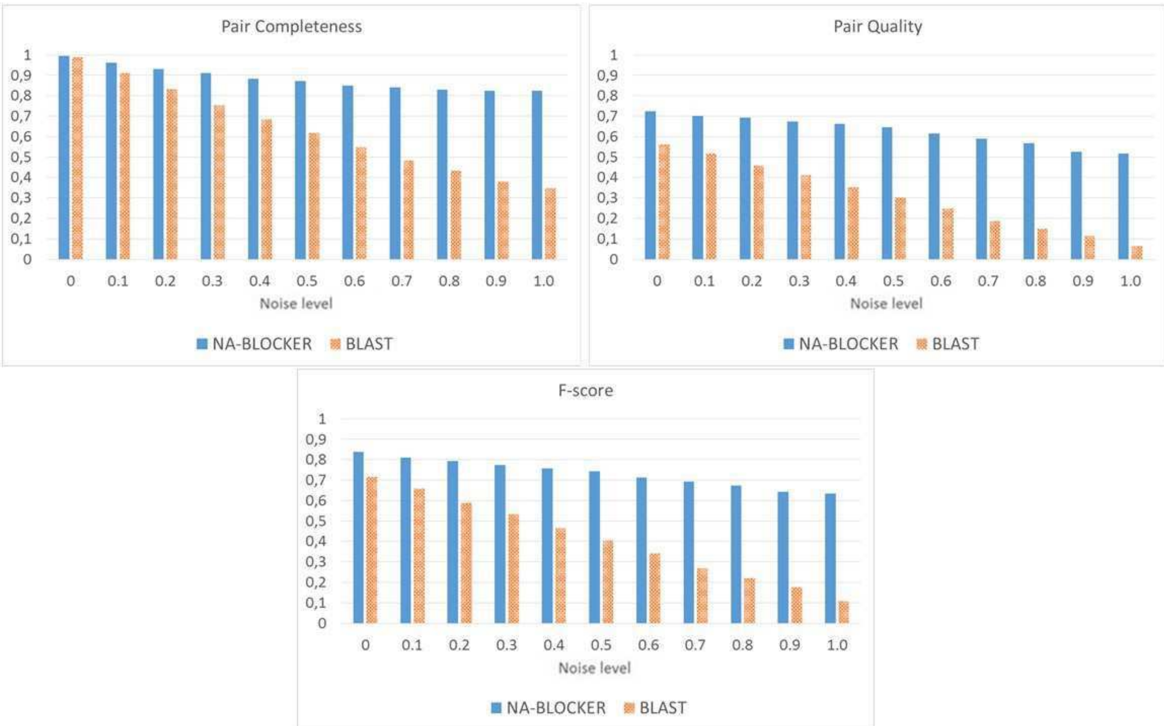

Figure 6.4: Effectiveness results for data sources DBLP vs. ACM.

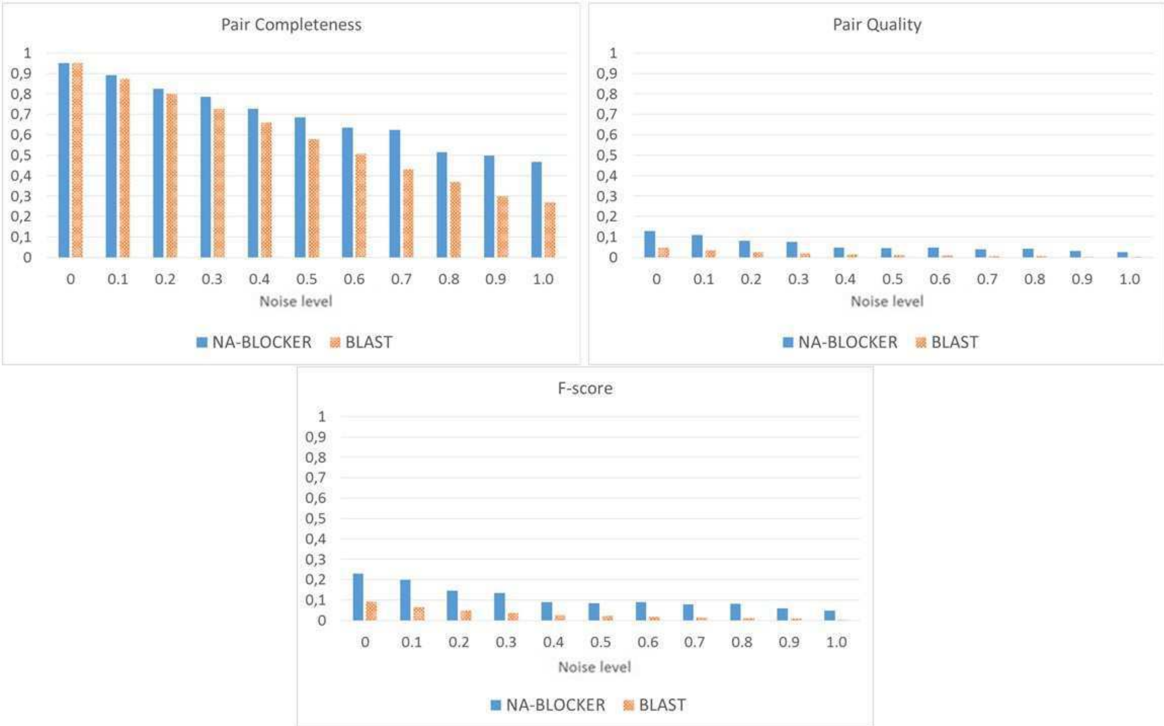

Figure 6.5: Effectiveness results for data sources DBLP vs. Google Scholar.

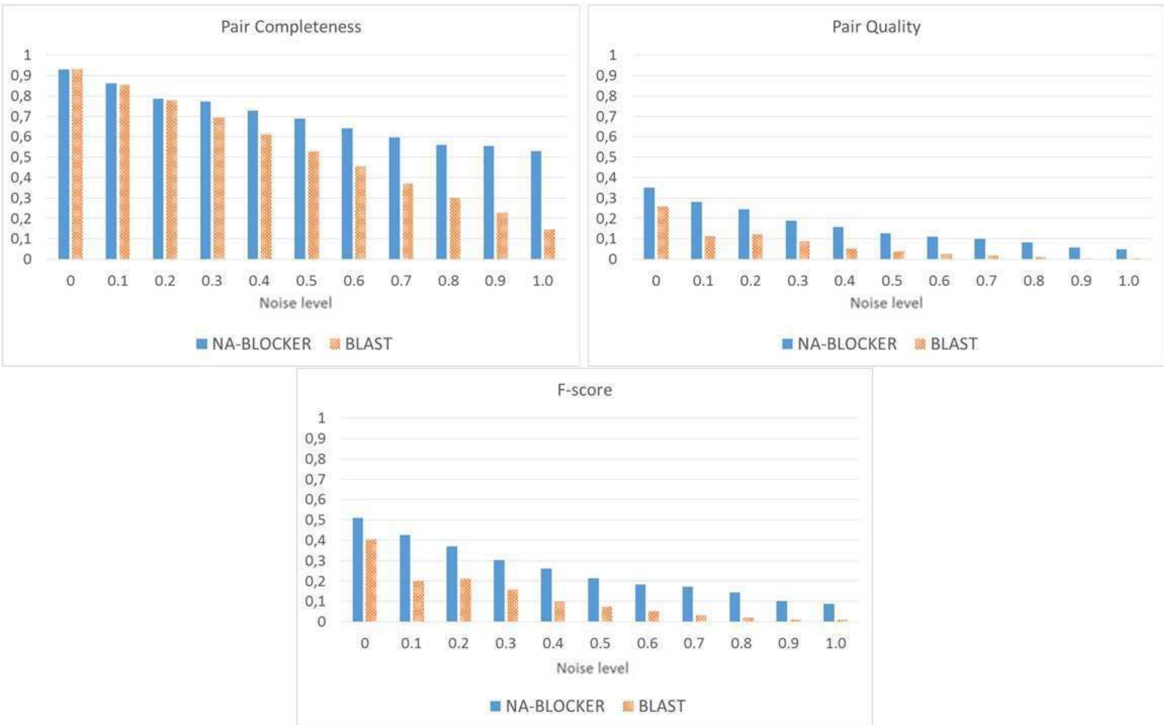

Figure 6.6: Effectiveness results for data sources IMDB vs. DBpedia.

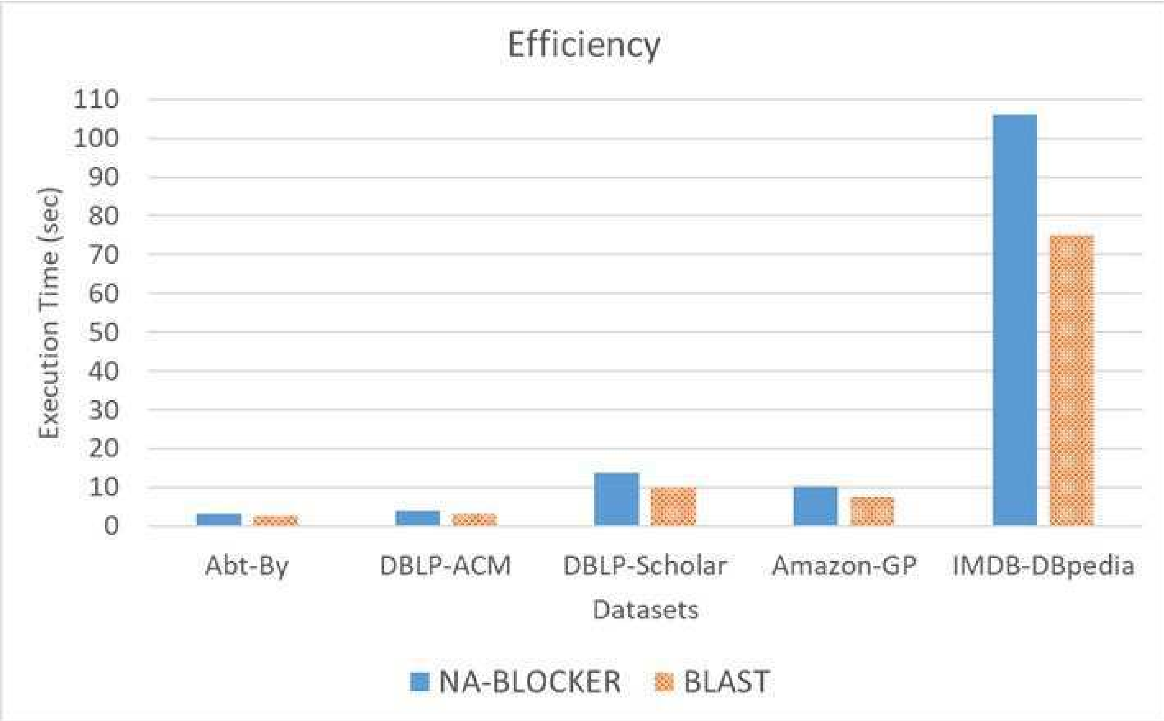

Figure 6.7: Execution time of NA-BLOCKER and BLAST techniques.

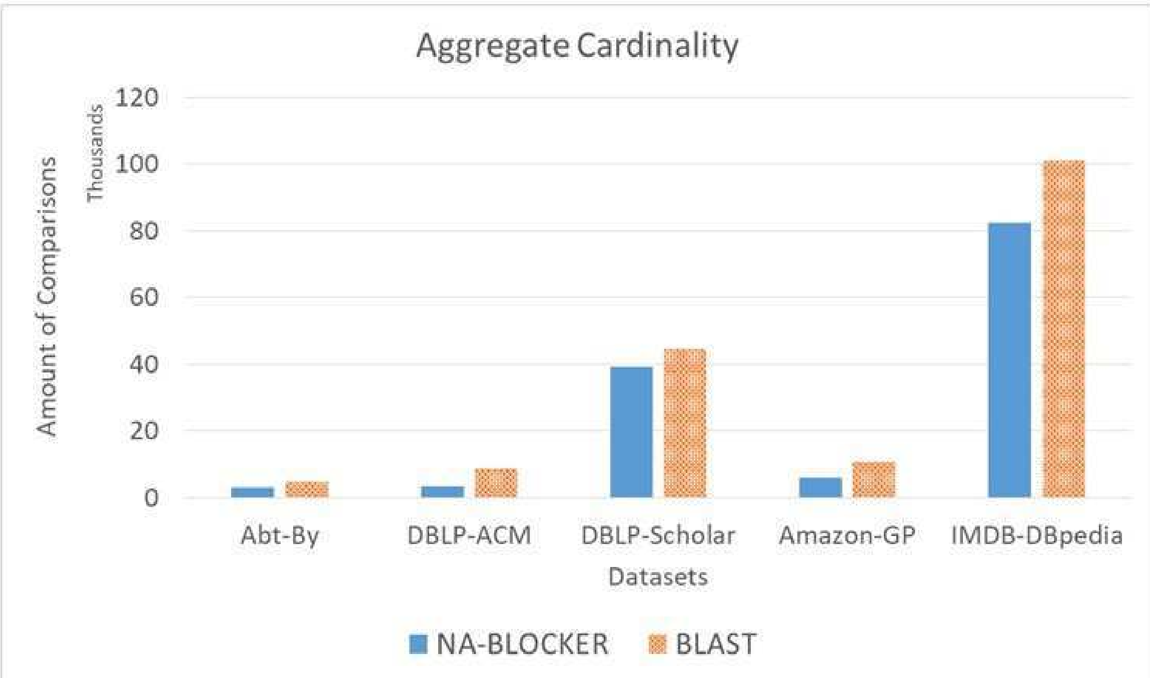

Figure 6.8: Aggregate Cardinality of NA-BLOCKER and BLAST techniques.

fewer comparisons between entities to be performed in the following steps of the ER task, particularly the Comparison step. For instance, considering that the comparison of an entity pair in the ER task requires one unit of time, then NA-BLOCKER will provide a reduction of 34% in the execution time of the Comparison step. Therefore, NA-BLOCKER does not significantly affect the efficiency of the ER task as a whole, since the Comparison step is the most costly step (in terms of execution time) of the ER task [21].

6.3 Final Considerations

In this chapter, a noise-aware blocking technique was proposed, which is able to extract schema information from the data in order to improve the quality (effectiveness) of the generated blocks. The NA-BLOCKER blocking technique applies LSH in order to generate signatures (from a hash function) that can guide the block generation. The application of LSH benefits the NA-BLOCKER technique since the technique generates blocks capable of achieving good effectiveness results even in the presence of noisy data. Based on the results obtained by the NA-BLOCKER technique in the experimental evaluation, it is possible to highlight that such technique achieved better results in terms of effectiveness compared to the state-of-the-art blocking technique (i.e., BLAST), also considering scenarios of noisy data. The contributions proposed in this chapter are part of the development of blocking techniques that can be coupled with the execution model proposed in Chapter 4. In particular, the NA-BLOCKER technique is directly related to the *schema information extraction* steps, *block generation* and *pruning of comparisons* of the proposed execution model.

As stated in this chapter, during the development of the NA-BLOCKER technique, the first contributions were focused on the impact of LSH on the effectiveness of the generated blocks, whose results were disclosed in [6] and [10]. Although the NA-BLOCKER technique was planned to be parallelized, the publication of the article [90] forced us to change our original plans. Considering that [90] proposed the parallelization of the BLAST technique with improvements in terms of effectiveness, the focus of the thesis was updated to face the challenges involving streaming data and incremental processing (see Chapter 7). However, it is important to highlight that the blocking technique proposed in [90] can also be included in the proposed model (see Chapter 4) to address scenarios involving noisy data (in batch) with focus on efficiency (e.g., large data sources). In the next chapter, the novel blocking technique (called PRIME) able to handle streaming data incrementally will be proposed. PRIME is considered an innovative technique since in the best of our knowledge none of the proposed blocking techniques faces the challenges involving Big Data, streaming data and incremental processing, simultaneously.

Chapter 7

An Incremental Schema-agnostic Blocking Technique for Streaming Data

Considering the main aspects highlighted in Chapter 2 faced by the ER task, we detach four in this Chapter: semi-structured data, large volume of data, streaming data and incremental process. Among them, this chapter focuses on streaming data and incremental process, since they are considered as an open area for the ER task [81; 90]. Streaming data is related to dynamic data sources (e.g., from Web systems, Social media and Sensors), which are continuously updated. When the ER task receives streaming data, we assume that not all data (from the data sources) are available at once. Therefore, ER needs to match the entities as they arrive, also considering the entities already matched previously. On the other hand, incremental ER is related to receiving data continuously over time and re-processing only the portion of the matching results (i.e., similar entities) that was affected by the data increments. Regarding incremental ER, new challenges need to be considered, such as i) *how to manage the entities already processed since they can be infinite?* and ii) *how to execute efficiently the ER task considering the whole stream of entities?* [46]. The challenges are strengthened when the ER task is considered in the context of Big Data, streaming data and incremental processing, simultaneously.

Streaming data is related to our daily life in different contexts: 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) [85]. Particularly for social data, in a single minute, 456,000 tweets are posting, 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 while 4,146,600 videos are watched [51]. Therefore, streaming data is generated every second, and such 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 are found on the Web. For this reason, the development of efficient blocking techniques able to handle streaming data appears as an open problem.

Notice that the challenges are strengthened when the ER task is placed in the context of heterogeneous data, streaming data and incremental processing, simultaneously. In this sense, the objective of this chapter is to propose a schema-agnostic blocking technique to be applied in the proposed execution model (described in Chapter 4), named PaRallel-based Incremental MEtablocking (PRIME). The blocking technique implements strategies to process efficiently streaming data, considering the incremental behaviour. To this end, strategies are developed to quickly receive and process entities (sent continuously) and prevent the blocking task from consuming resources (e.g., CPU and memory) excessively. Therefore, the general hypothesis of our work is to evaluate whether (or not) the application of PRIME is able to improve the efficiency of the ER task without decreasing the effectiveness for incremental processing and streaming data scenarios.

Overall, the contributions of this chapter are the following: i) an incremental and parallel-based schema-agnostic blocking technique (i.e., PRIME). PRIME is able to deal with streaming and incremental data, as well as to minimize the challenges related to both scenarios; ii) a parallel-based workflow for entity blocking, which reduces the number of parallel steps and the memory consumption without affecting the effectiveness of generated blocks; iii) a time-window strategy to discard entities already processed based on the time they were received, avoiding problems related to excessive memory consumption; iv) an attribute selection algorithm, which discards useless attributes from the entities, enhancing efficiency and minimizing memory consumption; and v) a top-n neighborhood strategy, which maintains only the "n" most similar entities (i.e., neighbor entities) of each entity, enabling PRIME to deal with large data sources.

7.1 Theoretical Foundations for Incremental Blocking in Streaming Data Scenarios

A blocking technique receives two data sources D_1 and D_2 as input. Since data is sent incrementally, the ER task processing is divided into a set of increments $I = \{i_1, i_2, i_3, ..., i_{|I|}\}$, where $|I|$ is the number of increments. 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., $time(i_{|I|}) - time(i_{|I|-1}) = \tau$, where $time$ returns the timestamp the increment was sent. For instance, in a scenario where data sources provide data every 30 seconds, we can assume that $\tau = 30\ seconds$. For each $i \in I$, each data source sends an entity collection $E_i = \{e_1, e_2, e_3, ..., e_{|E_i|}\}$. Since the entities can follow different loose schemes, each $e \in E_i$ has a specific attribute set and a value associated to each attribute, denoted by $A_e = \{\langle a_1, v_1\rangle, \langle a_2, v_2\rangle, \langle a_3, v_3\rangle, ..., \langle a_{|A_e|}, v_{|A_e|}\rangle\}$, such that $|A_e|$ is the amount of attributes associated with e ($|A_e|$ can be different for distinct e). Moreover, in order to generate the entity blocks, tokens (e.g., keywords) are extracted from the attribute values.

Definition 1 (Token extraction) *All tokens associated to 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, given two entities from distinct data sources D_1 and D_2, $e_1 = \{\langle name, Linus\ \ Torvalds\rangle,\ \langle creator, Linux\rangle,\ \langle nationality, Finland\rangle\}$ and $e_2 = \{\langle developer, Torvalds\rangle,\ \langle system, Linux\ \ OS\rangle,\ \langle age, 49\rangle\}$. Then, after the token extraction, the sets $\Lambda_{e_1} = \{Linus,\ Torvalds,\ Linux,\ Finland\}$ and $\Lambda_{e_2} = \{Torvalds,\ Linux,\ OS,\ 49\}$ are generated.

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 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 $\rho = \Phi(x_1, x_2)$[1]. The attribute selection algorithm aims to discard

[1]In this work, the similarity value between x_1 and x_2 is given by the average of common tokens between them, $\Phi(x_1, x_2) = \frac{|\Lambda_{x_1} \cap \Lambda_{x_2}|}{max(|\Lambda_{x_1}|, |\Lambda_{x_2}|)}$.

useless attributes in the sense of providing tokens that will hardly assist the blocking task to find similar entities. For instance, consider two general entities $e_1 \in E_i$ and $e_2 \in E_i$, where $e_1 \in D_1$ and $e_2 \in D_2$. 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 significative blocking keys (in terms of representing similarity between the entities). To this end, the attribute selection algorithm determines the similarity between attributes based on their values. Then, the algorithm discards attributes (and consequently their values) that do not have similarity between each other.

Definition 2 (Attribute selection) *All attributes associated to the entities belonging to data sources D_1 and D_2 are extracted. Moreover, all values associated to 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 to a specific attribute a_i. Let R be the list of discarded attributes and two general 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. Therefore, the token extraction considering attribute selection is given by $\Lambda_e = \bigcup(\Gamma(v) \mid \langle a, v\rangle \in A_e \wedge a \notin R)$.*

For example, considering the attributes *name, creator* and *nationality* of data source D_1 and the attributes *developer, system* and *age* of data source D_2, it is possible to determine the similarity between them based on their attribute values. Based on the running example with the entities $e_1 = \{\langle name, Linus\ Torvalds\rangle,\ \langle creator, Linux\rangle,\ \langle nationality, Finland\rangle\}$ and $e_2 = \{\langle developer, Torvalds\rangle,\ \langle system, Linux\ OS\rangle,\ \langle age, 49\rangle\}$, we can add into the set R the attributes *nationality* and *age* since both do not present similar values with the other attributes. For this reason, the set of tokens should be updated to remove the values derived from the attributes contained in R. Therefore, the new token sets are: $\Lambda_{e_1} = \{Linus,\ Torvalds,\ Linux\}$ and $\Lambda_{e_2} = \{Torvalds,\ Linux,\ OS\}$.

In the ER task, a blocking technique aims to group the vertices of G into a set of blocks denoted by $B_G = \{b_1, b_2, ..., b_{|B_G|}\}$. In turn, a pruning criterion $\Theta(G)$ is applied to remove redundant comparisons, resulting in a pruned graph G' [80].

Definition 3 (Pruning) *The vertices of G' are grouped into a set of blocks denoted by*

$B'_{G'} = \{b'_1, b'_2, ..., b'_{|B'_{G'}|}\}$, *such that* $\forall b' \in B'_{G'}(b' = \{x_1, x_2, ..., x_{|b'|}\})$, $\forall \langle x_1, x_2 \rangle \in b'$: $\exists \langle x_1, x_2, \rho \rangle \in L$ *and* $\rho \geq \theta$*, where* θ *is a similarity threshold defined by a pruning criterion* $\Theta(G)$.

For instance, considering the running example, the blocks $B_G = \{b_{Linus} = \{e_1\}, b_{Torvalds} = \{e_1, e_2\}, b_{Linux} = \{e_1, e_2\}, b_{OS} = \{e_2\}\}$ are generated based on the tokens. Then, a pruning criterion $\Theta(G)$ is applied to remove redundant comparisons, which results in a set of blocks $B'_{G'} = \{b'_1 = \{e_1\}, b'_2 = \{e_1, e_2\}, b'_3 = \{e_2\}\}$.

When blocking techniques deal with streaming and incremental behavior, 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 $\{\Delta G_1, \Delta G_2, ..., \Delta G_{|I|}\}$ 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 by multiple nodes (e.g., computers or virtual machines). In this context, it is assumed that $N = \{n_1, n_2, ..., n_{|N|}\}$ 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 $T^N_{\Delta G_i}(B_G)$.

Definition 4 (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 highest time to execute the task for a specific increment* ΔG_i*:* $T^N_{\Delta G_i}(B_G) = max(T^n_{\Delta G_i}(B_G)), n \in N$.

Definition 5 (Time restriction for streaming blocking) *Considering the streaming behaviour, where each increment arrives in each* τ *time interval, it is necessary to determine a restriction of execution time to process each increment, given by* $T^N_{\Delta G_i}(B_G) \leq \tau$.

The restriction denoted in Definition 5 aims to prevent the blocking execution time from overcoming the time interval of each data increment. 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. To this end, the proposed blocking technique needs to be efficient, as denoted in Definition 6.

Definition 6 (Blocking efficiency in distributed environments) *Given as input: 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 $|N| - \frac{T^n_{\Delta G_i}(B_G)}{T^N_{\Delta G_i}(B_G)}$.*

7.2 Streaming Metablocking

The state-of-the-art blocking techniques (either stand-alone or parallel-based) do not work properly in scenarios involving incremental and streaming data, since they were developed to work with batch data [11]. In this chapter, to define a baseline which will be compared against PRIME, we also implemented a Metablocking technique capable to deal with streaming and incremental data, called Streaming Metablocking. Streaming Metablocking is based on the parallel workflow for Metablocking proposed in [37]. However, it was adapted to take into account challenges involving streaming and incremental data (e.g., time restrictions, increasing amount of data and resource consumption). Since the Streaming Metablocking technique (as well as PRIME) considers the incremental behavior, we need to update the blocks in order 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 blocks that did not suffer any update (e.g., insertion of new entities). Clearly, this strategy is costly in terms of efficiency since it performs a huge number of unnecessary comparisons that have already been performed. To avoid this kind of unnecessary comparisons, Streaming Metablocking only considers the blocks that suffered at least one update for the current increment. Notice that the original workflow proposed in [37] does not consider the incremental behavior. Therefore, the idea of (re)processing the blocks generated previously is discussed in this chapter. Regarding the streaming behavior, Streaming Metablocking was implemented following the MapReduce workflow proposed in [37]. However, a specific MapReduce framework (i.e., Flink) is able to handle streaming data was applied.

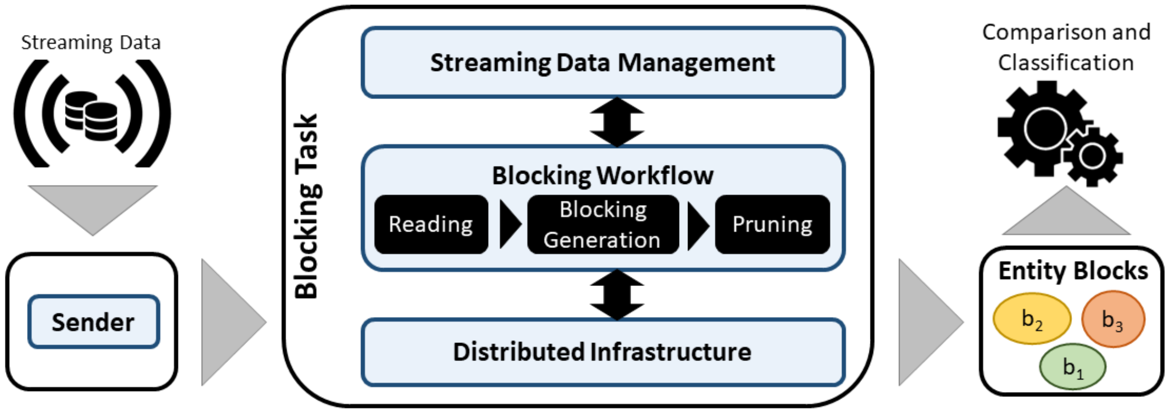

Figure 7.1: A model for incremental blocking over streaming data.

7.3 Incremental Blocking Technique for Streaming Data

Considering the distributed execution model proposed in Chapter 4, it is important to note the necessity of a Sender component to connect the streaming data source (e.g., social media API, sensor network, news feed) to the Blocking task, as depicted in Figure 7.1. The Blocking component is related to the steps of the distributed execution model responsible for blocking semi-structured data. Regarding the Sender component, it consumes the data provided by the data sources in a streaming way, buffers it in micro-batches and sends to the Blocking component. In this work, the streaming processing platform Apache Kafka[2] is applied to connect the Sender and the Blocking components.

Regarding the blocking step, unlike previous parallel Metablocking approaches [37], the proposed blocking technique (i.e., PRIME) applies a different workflow to reduce the number of MapReduce jobs and improve efficiency. In turn, this workflow is composed of two MapReduce jobs (Figure 7.2), i.e., one job less than the method available in [37] (will be discussed in Token Extraction step). Besides, the proposed workflow does not affect negatively on the effectiveness results since the block generation is not modified. The workflow is divided into three steps (as illustrated in Figure 7.2): token extraction, blocking generation and pruning. Considering the steps of the execution model (i.e., data reading and interpretation, schema information extraction, block generation and pruning of comparisons), it is possible to detach that the PRIME steps (proposed in this chapter) are compatible with the execution model ones. Notice that, the token extraction step (in PRIME workflow) covers

[2]https://projects.apache.org/projects.html

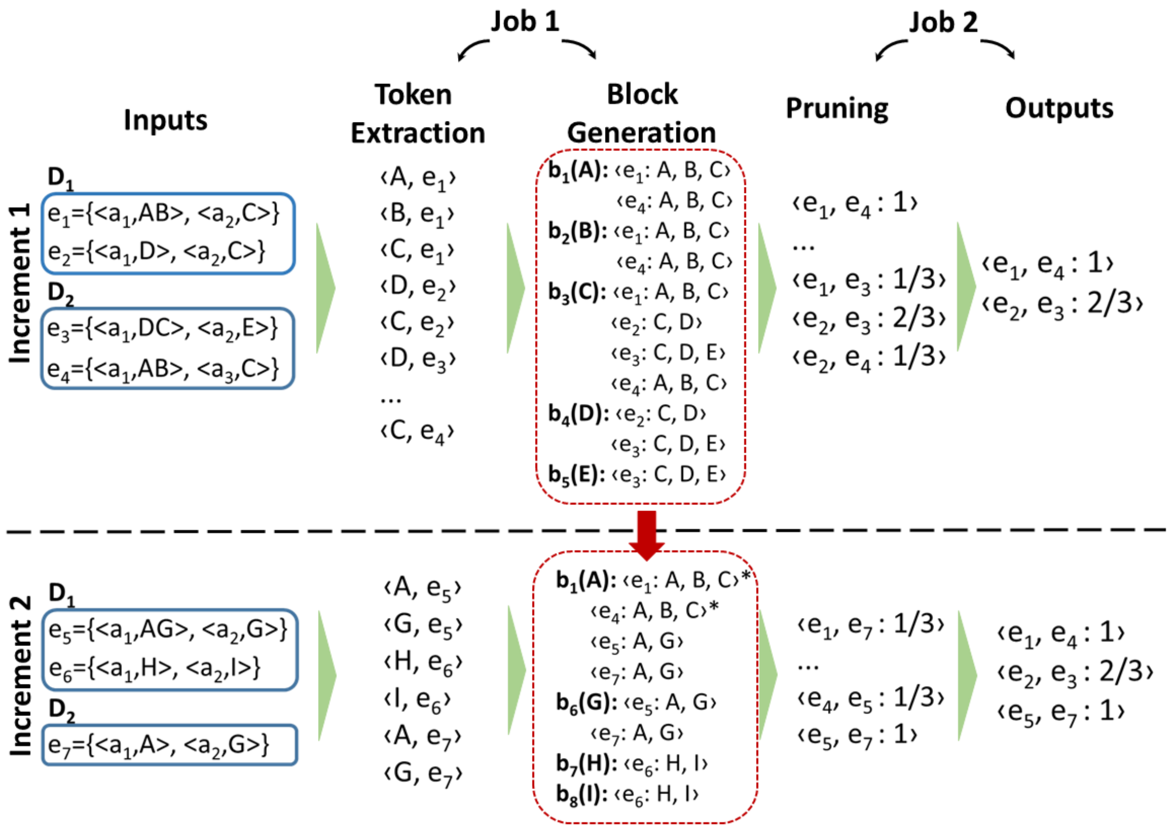

Figure 7.2: Workflow of the streaming blocking technique.

the data reading and interpretation step (in the execution model workflow). Therefore, due to efficiency reasons in the streaming data context, the schema information extraction step (in the execution model workflow) is not considered in this chapter.

7.3.1 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 5. For each increment, the blocking step receives a pair of entity collections E_{D_1} and E_{D_2} provided by data sources D_1 and D_2 (lines 2 and 3). For each entity $e \in E_{D_1} \cup E_{D_2}$ (lines 6 to 10), all tokens Λ_e associated with e are extracted and stored in memory.

Each token in Λ_e will be considered 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. It is important to highlight that, from this step, every entity e contains information

Algorithm 5: Token Extraction

Data: E_{D_1} and E_{D_2}: increments provided by input data sources

Result: B: blocks based on tokens

1 $B \leftarrow \emptyset$;
2 $tokenBlocking(E_{D_1})$;
3 $tokenBlocking(E_{D_2})$;
4 **return** B
5 **Function** *tokenBlocking(*E_D*)* **do**
6   **foreach** e ***in*** E_D **do**
7   **foreach** t ***in*** Λ_e **do**
8             $B.put(t, \langle e, \Lambda_e, D \rangle)$;
9   **end**
10   **end**
11 **end**

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 discard one MapReduce job (compared with the workflow proposed in [37]). In [37], the workflow applies an extra job to process the entities in each block and determine all blocks which 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.

The time complexity of token extraction step is directly related to the generation of blocking keys. This step evaluates all tokens in Λ_e of each entity $e \in E_D$ to generate the blocking keys. Therefore, the time complexity of this step is $\mathcal{O}(||\Lambda_{E_{D_1}}|| + ||\Lambda_{E_{D_2}}||)$, where $||\Lambda_{E_{D_1}}||$ is given by $\sum_{e \in E_{D_1}} |\Lambda_e|$ and $||\Lambda_{E_{D_2}}||$ is given by $\sum_{e \in E_{D_2}} |\Lambda_e|$. We guarantee that the produced blocking keys are the same as the ones produced by Metablocking, with the following lemma.

Lemma 1 *Considering the same entity collection E_i as input, if PRIME and Metablocking techniques apply the same function $\Gamma(v)$ to extract tokens from attribute values, then both techniques will produce the same blocking keys Λ_e.*

Proof 1 *In the token extraction step, PRIME 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 $\Gamma(v)$ over the attribute values v of e. If Metablocking receives the same E_i and applies the same $\Gamma(v)$ over 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 7.2, 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, e_2 while D_2 provides entities e_3, 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.

7.3.2 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 6. Initially, the entities are arranged in the format $\langle e, \Lambda_e \rangle$, implicitly Λ_e denotes the blocks that contain entity e. Based on the blocking keys (i.e., tokens), 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 7.2, 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.

Afterwards, 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 determined based on the number of co-occurring blocks between the entities. After defining the similarity between the entities, the entity pairs are

Algorithm 6: Blocking Generation

Data: B: blocks generated in the token extraction step

Result: G: blocking graph

```
1  G ← ∅;
2  foreach k in B.keys do
3      entities ← B.get(t);
4      while entities is not ∅ do
5          e1 ← entities.pop();
6          foreach e2 in entities do
7              if e1.D is not e2.D then
8                  ρ ← Φ(e1.Λ_e1, e2.Λ_e2);
9                  G.put(e1, ⟨e2, ρ⟩);
10                 G.put(e2, ⟨e1, ρ⟩);
11             end
12         end
13     end
14 end
15 return G
16 Function Φ(Λ_e1, Λ_e2) do
17     ρ ← |Λ_e1 ∩ Λ_e2| / max(|Λ_e1|, |Λ_e2|);
18     return ρ
19 end
```

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 between 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 amount 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 PRIME and Metablocking techniques apply the same similarity function Φ over 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 by vertices (i.e., X) and edges (i.e., L), the techniques will generate the same graph G.*

Proof 2 *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 over the vertices of X, the set of edges L will be the same produced by PRIME. 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 7.2, block b_1 contains entities e_1, e_4. Therefore, these entities must be compared to determine the similarity between them. The 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.

Incremental approaches that handle streaming data commonly suffer from memory issues since they need to maintain a large amount of data in memory [30; 85]. To avoid excessive consumption, we propose a top-n neighborhood strategy to be applied during the creation of graph G. 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). For example, for the first increment of Figure 7.2, 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. If we hypothetically apply a top-1 neighborhood, entities e_2 and e_3 will be removed from the neighbor entities. Clearly, considering large graphs (generated from by Big Data sources), the reduction of information stored in each node will result in a significant decreasing of 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.

During the 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). To avoid this, we apply the Marked Common Block Index (MaCoBI) [8; 37] condition. 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 determine the block in which the entity pair should be compared. Second, some blocks may not suffer updates for a specific increment, during the incremental process. Therefore, entities contained in such blocks 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, provided and managed by Flink, loads only blocks that have been updated for the current increment. Third, updated blocks also contain entity pairs that have been compared in previous iterations, which should not be compared again. The solution for this third type is to mark the entity pairs that have already been compared. 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 will consider the second iteration (bottom of Figure 7.2). The entity pair $\langle e_1, e_4 \rangle$ should be compared in blocks b_1, b_2 and b_3 (i.e., first type). However, the MaCoBi condition is applied to guarantee that $\langle e_1, e_4 \rangle$ is compared in only one of the possible blocks (b_1, b_2 or b_3). Regarding the second type, notice that only the updated block b_1 and the new blocks b_6, b_7 and b_8 are considered for the second increment. To avoid the third type, during the first increment, entities e_1 and e_4 are marked (with the symbol *) at block b_1. Therefore, for the second increment, the pair $\langle e_1, e_4 \rangle$ will not be compared again.

7.3.3 Pruning step

After generating the graph G, the pruning step is responsible to discard entity pairs with low similarity values, as described in Algorithm 7. To this end, a pruning criterion is applied over G to generate the set of high-quality blocks B' (lines 1 to 3). As a pruning criterion, we apply the WNP-based pruning algorithm [83] since it has achieved better results than its competitors [83; 37]. 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 average edge weight of the neighborhood - lines 6 to 11). Therefore, the neighbor entities whose edge weight is greater than the local threshold are inserted into B' (lines 12 to 16). Otherwise, the entities (i.e., whose edge weight is lower than the local threshold) are discarded. Notice that the application of the top-n neighborhood strategy (in the previous step) does not turn the pruning step useless. On the contrary, the top-n neighborhood strategy can provide a refined graph, in terms of 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 $\mathcal{O}(|X| \cdot |L|)$ [83], 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 PRIME and Metablocking produce the same output (i.e., B') and, therefore, present the same effectiveness results.

Lemma 3 *Considering that PRIME 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 3 *In the pruning step, PRIME 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 $\rho < \theta$ 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*

Algorithm 7: Pruning

```
Data: G: graph generated in block generation step
Result: B′: pruned block
B′ ← ∅;
B′ ← WNP(G);
return B'
Function WNP(G) do
    foreach e in G.keys do
        neighbors ← G.get(e);
        sum ← 0;
        foreach pair in neighbors do
            sum ← sum + pair.ρ;
        end
        θ ← sum / |neighbors|;
        foreach pair in neighbors do
            if pair.ρ ≥ θ then
                B′.put(e, pair);
            end
        end
    end
end
```

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, if PRIME 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 4 *Based on Lemma 1, with the same E_i and $\Gamma(v)$, PRIME 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 PRIME and*

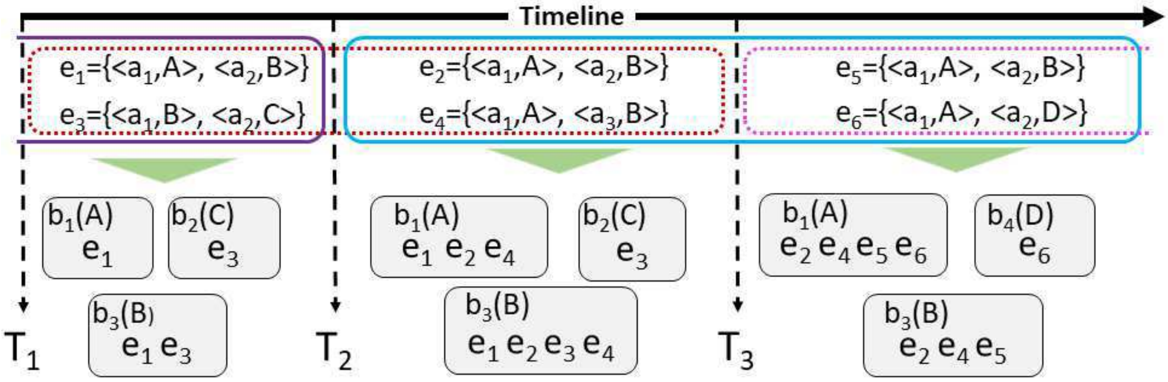

Figure 7.3: Time-window strategy for incremental blocking.

Metablocking techniques 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'.

7.4 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. Considering these behaviours, it is necessary to develop strategies that manage the computational resources. In this sense, when blocking techniques face these scenarios, memory consumption tends to be the biggest challenge to be handled by incremental approaches [86; 46; 30]. For this reason, a time window is used in the blocking generation step of PRIME, where blocks are incrementally stored in the update-oriented data structure (see Section 7.3). Time windows are at the heart of processing infinite streams since they split the stream into buckets of finite size. The memory consumption of PRIME is reduced by discarding entities that exceeded the time threshold.

In this work, we apply a sliding (time) window strategy, which defines a time interval that the entities should be maintained in the update-oriented data structure, preventing the structure from consuming memory excessively. For instance, consider the example of Figure 7.3, 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 the first increment (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 (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, while e_6 is also inserted into b_1 and triggers the creation of a new block b_4.

7.5 Attribute Selection

Real-world data sources can present useless attributes, which can be ignored by the blocking task [25]. We denote useless attributes those attributes whose tokens will generate low-quality blocks (i.e., with entities with low chances of matching). In this work, we explore two types of useless attributes: attributes of low representativeness and unmatched attributes between the data sources. The former are related to attributes whose content does not contribute to determine 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. Moreover, 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. Therefore, this type of attribute is discarded by our blocking technique to optimize memory consumption without significant impact on the effectiveness of generated blocks.

Unmatched attributes between data sources are the 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, probably it is possible to find several similar values provided by these attributes. However, D_2 does not have any attribute related to *age*. That is, *age* will generate blocks from its values, but these blocks will hardly be relevant to determine similarities between entities. For this reason, this type of attribute is also discarded by our blocking technique.

To handle useless attributes, we developed an attribute selection strategy applied in the

Sender component (see Figure 7.1). For each increment received by the Sender, the entity attributes are evaluated and the useless ones are discarded. Thus, the entities sent by the Sender component are modified by removing the useless attributes (and their values). To identify the attributes of low representativeness, the attribute selection strategy measures the attribute entropy. Notice that, the entropy of an attribute indicates how significant is the attribute (see Section 2.10). The attributes with low entropy value[3] (i.e., low representativeness) are discarded. For the unmatched attributes, attribute selection extracts and stores the attributes 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. Then, the matrix is evaluated and the attributes without corresponding attributes from the other data source are discarded. After removing these attributes (and their values), the entities are sent to the Blocking task.

In an overview, the attribute selection algorithm can be useful to other blocking techniques, including the techniques proposed in this thesis, which claim for efficiency. However, it is necessary to adapt to handle batch data. In this sense, the attribute selection algorithm should be applied at the *schema information extraction step*, similar to the blocking technique proposed in Section 6 (i.e., NA-BLOCKER).

7.6 Experimental Evaluation

Streaming data is continuously generated by one or more data sources in a specific time interval. This behavior inserts the idea of data frequency, which estimates the frequency (i.e., time interval) at which the data source sends data. Low-frequency data sources are related to sources that send data with a low frequency (i.e., large time intervals). In contrast, high-frequency data sources are related to sources that send data with a high frequency (i.e., short time intervals).

Considering incremental ER for streaming data (which receives data from two data sources), we can highlight three main scenarios regarding the data frequency: two low-frequency data sources, one low-frequency data source vs. one high-frequency data source

[3]In this work, we remove the attributes whose entropy values are in the first percentile (i.e., 25%) of the lowest values.

and two high-frequency data sources. In the first, both data sources provide the data in large time intervals (for instance, in hours or at the end of the day). In this sense, the ER can be performed without challenging time constraints, which potentiates all the other challenges discussed in this chapter. For these reasons, the first scenario is the easiest one in terms of computational challenges.

The second scenario is more challenging since one of the sources sends data in a short time interval. It means that the incremental ER task needs to match data from the data source respecting the time constraint given by the high-frequency data source. The second scenario will be addressed in Section 7.7.

The most chaotic scenario is the last one since both data sources send data in a short time interval. In this sense, the third scenario tends to increase the volume of data to be processed due to both data sources send data simultaneously. Therefore, ER tends to handle challenges involving high-volume of data and high-frequency data sources. Following, we will conduct experimental analyses addressing the third scenario to evaluate PRIME[4] in terms of effectiveness and efficiency.

We run our experiments in a cluster infrastructure with 21 nodes (one master and 20 slaves), each one with one core. Each node has an Intel(R) Xeon(R) 2.10GHz CPU, 4GB memory, runs the 64-bit Ubuntu/Linux 16.04.5 OS with a 64-bit JVM, Apache Kafka 2.1.0 and Apache Flink 1.7.1. Notice that PRIME and Streaming Metablocking apply Flink as a MR framework. We use three real-world pairs of data sources[4] as described in Table 7.1: i) Amazon-GP: includes product profiles provided by amazon.com and google.com; ii) IMDB-DBpedia: includes movie profiles provided by imdb.com and dbpedia.org; and iii) DBPedia$_1$-DBPedia$_2$: consists of two different versions of the DBPedia Infobox dataset[5] (snapshots from October 2007 and 2009, respectively). Table 7.1 also shows the amount 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.

Since working with large data sources requires a huge amount of resources (e.g., memory and CPU), DBPedia$_1$-DBPedia$_2$ is a sample from the whole data source containing millions of entities. The cluster used in this experiment is able to process at most tens of thousands

[4]https://github.com/brasileiroaraujo/PRIME-F

[5]https://wiki.dbpedia.org/Datasets

Table 7.1: Data sources characteristics.

Pairs of datasets	$\|D_1\|$	$\|D_2\|$	$\|M\|$	$\|A_1\|$	$\|A_2\|$
Amazon-GP	1,354	3,039	1,104	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

of entities per increment. Therefore, data sources containing millions of entities will significantly increase the number of increments (several hundreds) unnecessarily. To avoid problems involving lack of resources in our cluster, we randomly select 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 the proportionality in terms of the number of duplicates and the difference in the amount of entities between the data sources.

To simulate the 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[6] (i.e., increment). Notice that the data streaming sender assumes the role of the data provider, such as feeds of news, movie/product reviews, weather information or twitters. Therefore, the Sender component (see Figure 7.1) is able to receive data from any feed providing data in streaming. To measure the effectiveness of blocking, we use the following quality metrics: Pair Completeness (PC), Pair Quality (PQ) and Reduction Ratio (RR), which take values in [0, 1], with higher values indicating a better result. It is important to highlight that schema-agnostic blocking techniques (even the state-of-the-art ones) yield high recall (i.e., PC), but at the expense of precision (i.e., PQ) [25; 90]. 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 [90].

For efficiency, we comparatively evaluate the maximum number of entities processed per increment for the proposed technique (PRIME) and Streaming Metablocking. To measure the maximum number of entities processed per increment, we monitored the Back Pressure metric. This metric takes values in the interval [0, 1], where a high Back Pressure (e.g., higher than 0.5) means that the task is producing data faster than the downstream operators (in the cluster) can consume. Thus, from the Back Pressure, it is possible to define the

[6]In this work, $\tau = 1$ *min* (i.e., one increment per minute) is used for all experiments.

computational limit (in number of entities per increment) of the cluster for each blocking technique. Next, we use the acronyms PRIME for the proposed technique, SM for Streaming Metablocking, AS for Attribute Selection and Top-n for the Top-n neighborhood strategy.

7.6.1 Effectiveness

In Figures 7.4, 7.5 and 7.6, we evaluate the effectiveness of the PRIME and SM techniques. Since the window size is a limitation of streaming processing and affects the effectiveness (truly matches will never be compared if they are not covered by the window size), we implement the Sender so as to always send truly matches (according to the ground-truth) in a time interval covered by the window size. Note that the effectiveness of SM and PRIME are exactly the same (see Theorem 1), when PRIME does not apply the top-n neighborhood and/or attribute selection strategies.

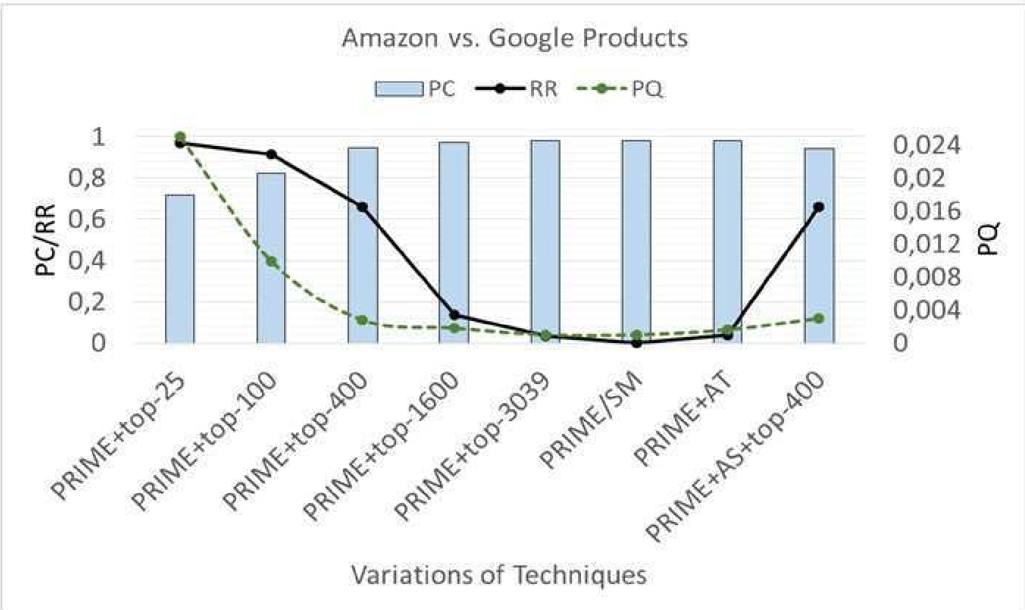

Figure 7.4: Effectiveness results of the techniques for the data sources Amazon vs. Google Product.

For PQ and RR, the application of the top-n neighborhood and attribute selection strategies in PRIME outperforms the results of SM for all data sources. It occurs due to the fact that the application of these strategies decreases the number of entity pairs to be compared within the blocks. Note that top-n neighborhood aims to reduce the number of neighbor

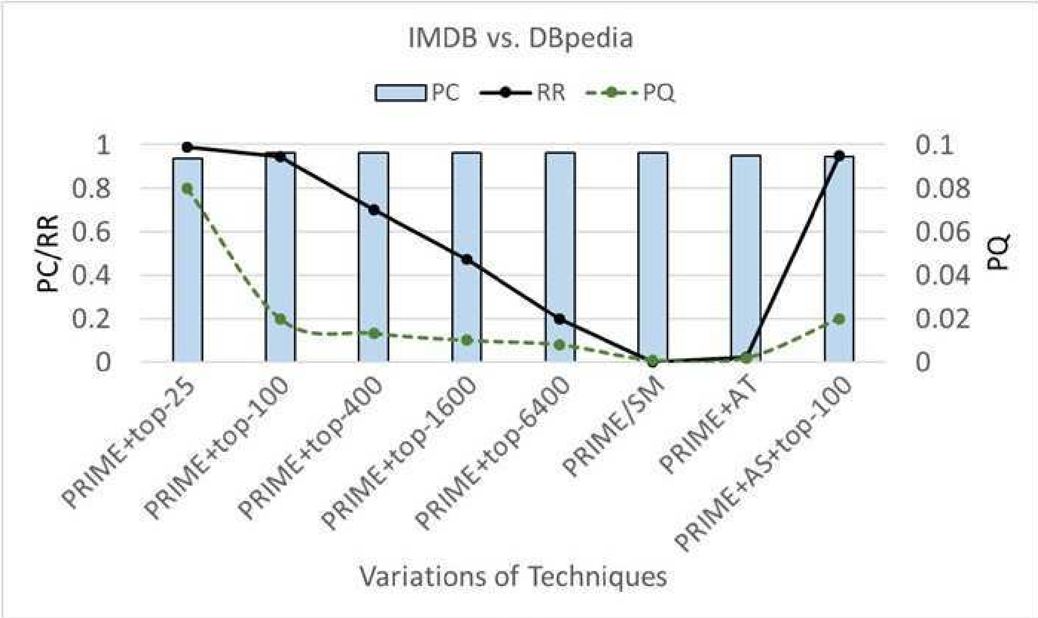

Figure 7.5: Effectiveness results of the techniques for the data sources IMDB vs. DBpedia.

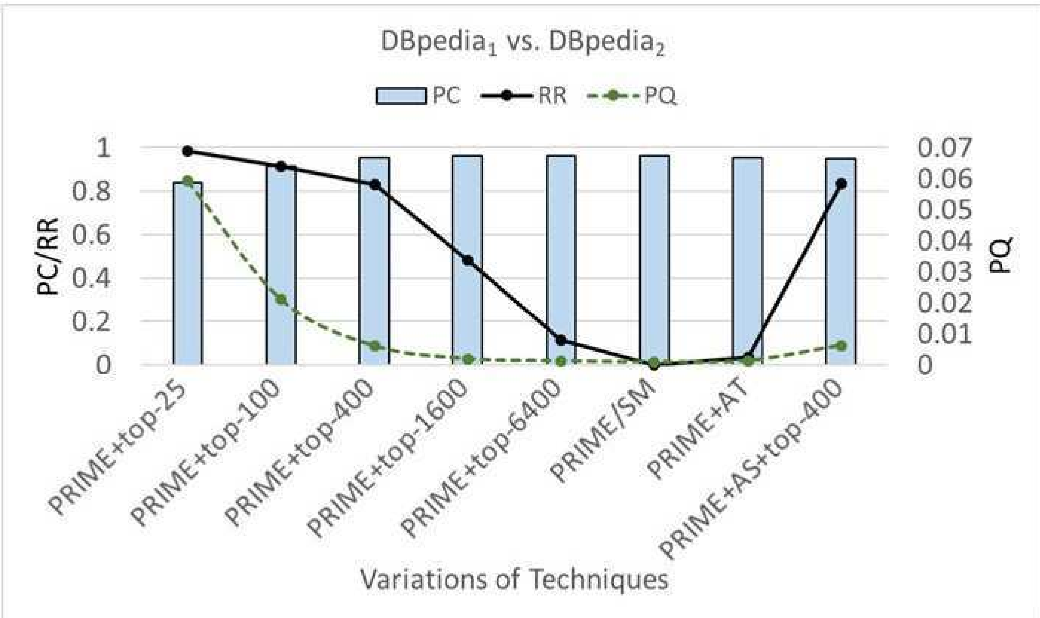

Figure 7.6: Effectiveness results of the techniques for the data sources DBpedia$_1$ vs. DBpedia$_2$.

entities in the similarity graph. For this reason, the size of the blocks (in 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, the application of these strategies improves PQ and RR, which are directly related to the number of comparisons generated from the blocks (i.e., $||B'||$). We can also analyze RR as an efficiency metric since it estimates the number of comparisons to be executed in the ER task. In this sense, we can detach the efficiency gains (for the ER task as a whole) provided by the application of both strategies, which produces blocks with fewer comparisons.

Regarding PQ and RR, notice that as the value of n in top-n neighborhood strategy decreases, these metrics achieve better results. This behavior occurs due to the reduction in the number of neighbor entities, which decreases the number of entities to be compared in ER task. For this reason, low values of n imply good results for PQ and RR. However, PC decreases when top-n neighborhood is applied, in the worst case for Amazon-GP, PC achieved 0.72 for $n = 25$. Thus, top-n neighborhood may decrease PC significantly to very low n values (e.g., $n = 25$). On the other hand, we can detach that, even for large data sources (e.g., DBPedia_1-DBPedia_2), n values between 100 and 400 present reasonable PC values (PC higher than 0.95) for all data sources. For instance, top-n neighborhood, with $n = 400$, decreases PC only 1.6%, on average, for all data sources. In this experiment, we select the best top-n value to be applied together with the attribute selection (i.e., PRIME + AS + top-n). In this sense, the application of both strategies maintains efficiency, enhancing RR and PQ with a small decrease in PC, when compared with the baseline technique.

When attribute selection is applied, PC decreases, in the worst case for IMDB-DBpedia, only 1.5%. Thus, it is possible to infer that AS discards only attributes that do not significantly affect PC results (i.e., in fact, useless attributes). Regarding PQ and RR, the application of attribute selection implies in small improvements in these metrics since this strategy reduces the number of generated blocks and, consequently, the number of comparisons to be performed in ER task. The application of top-n neighborhood and attribute selection together achieved reasonable results in terms of effectiveness for all data sources, since PQ and RR increased comparing with PC, which suffered only a decreasing of 1%, in average.

7.6.2 Efficiency

To evaluate the PRIME against SM (see Figures 7.7, 7.8 and 7.9), we apply a sliding window with a size of four minutes for all data sources, since this size achieves reasonable effectiveness results even if the entities are sent in the order stored in the data sources. To evaluate the efficiency, we measure the maximum number of entities processed by the techniques per increment, varying the number of nodes between 1 and 20. Notice that the maximum number of processed entities denotes the efficiency of the techniques in terms of resource management (e.g., CPU and memory), as well as the variation in the number of nodes demonstrates the scalability of techniques. Since the pair Amazon-GP is composed of small data sources, we vary the number of nodes up to four. After that, the number of nodes does not interfere in the results and all data from the data source is consumed in only one increment.

Based on the results shown in Figures 7.7, 7.8 and 7.9, PRIME outperforms SM for all data sources, even without the application of top-n neighborhood or attribute selection strategies, due to the reduction of MapReduce jobs (see Section 7.3), which saves computational resources and improves efficiency. On average, PRIME increases the number of processed entities in 12% when compared with SM.

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 PRIME with attribute selection (i.e., PRIME + 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 the number of generated tokens comparing when attribute selection is not applied. It is important to highlight that 2% of reduction for DBpedia_1-DBpedia_2 results in more than 18,000 tokens less. In other words, attribute selection avoids 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. This reduction in the number of blocks affects directly the number of entities processed per increments since it reduces resource consumption. Therefore, based on the results present in Figures 7.7, 7.8 and 7.9, notice that PRIME + AS outperform PRIME for all data sources. On average, PRIME + AS increase the number of entities processed per increment in 20% compared with PRIME.

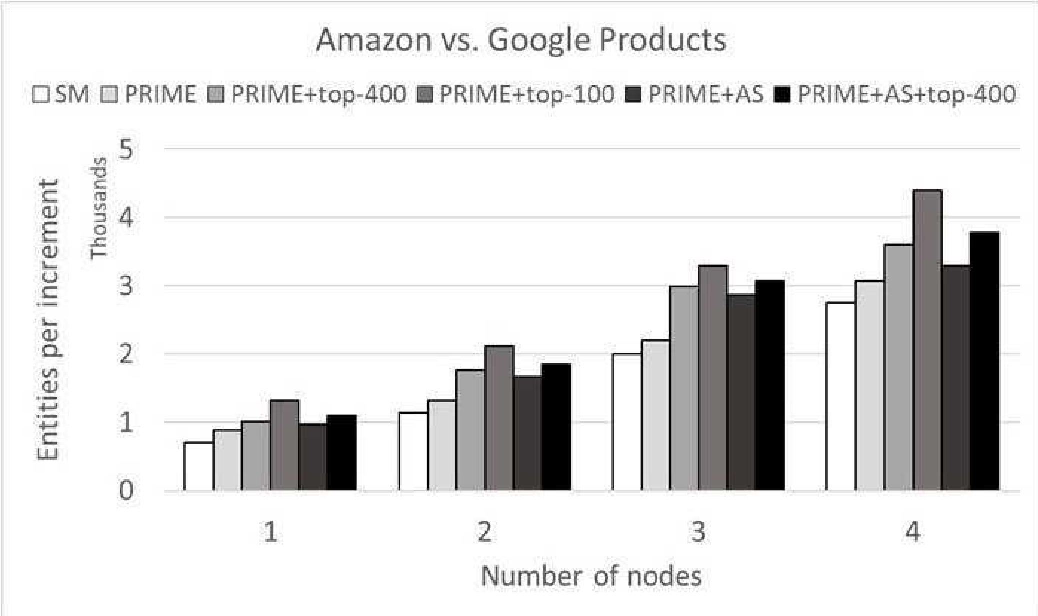

Figure 7.7: Efficiency results of the techniques for the data sources Amazon vs. Google Product.

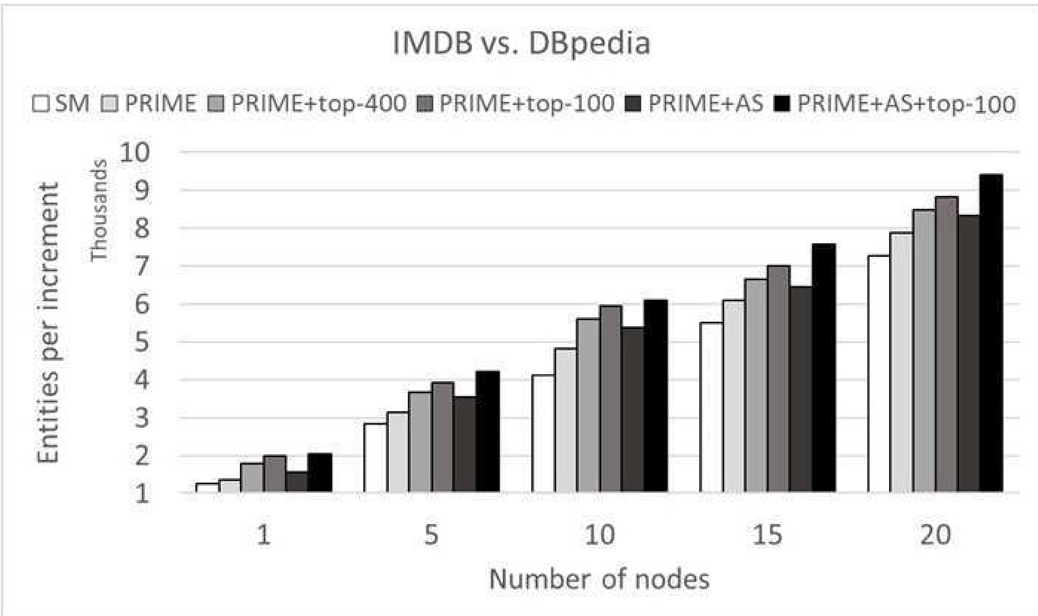

Figure 7.8: Efficiency results of the techniques for the data sources IMDB vs. DBpedia.

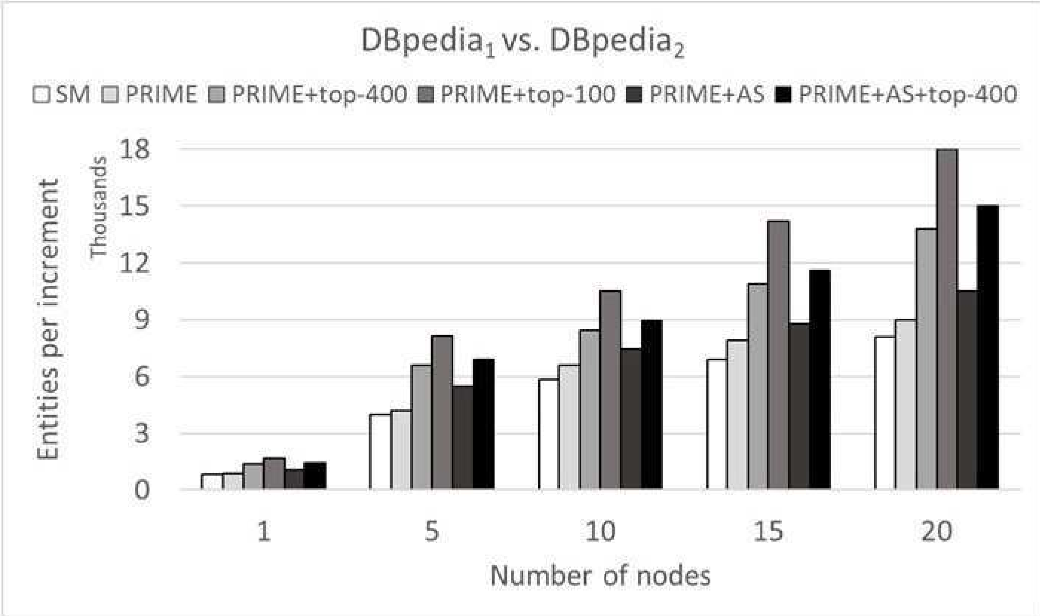

Figure 7.9: Efficiency results of the techniques for the data sources DBpedia$_1$ vs. DBpedia$_2$.

Regarding the top-n neighborhood strategy, we evaluate PRIME with top-100 and top-400, the two most promising values in terms of effectiveness. Low "n" values (in top-n) negatively impact PC since a small number of neighboring entities reduces the chances to find true matches. On the other hand, high "n" values do not decrease the PQ metric, since high "n" values probably will achieve the original number of neighboring entities for each entity. Thus, the application of high "n" values tend to achieve similar results (in terms of effectiveness) to PRIME 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. It occurs due to the reduction of neighboring entities linked to each node (i.e., entity) in the similarity graph and, consequently, the reduction of consumed computational resources. For this reason, in DBpedia$_1$-DBpedia$_2$, PRIME applying top-100 neighborhood increased the number of entities processed per increment in 85%, on average, when compared with PRIME while the application of top-400 increased 47% the number of processed entities.

The application of the attribute selection and top-n neighborhood strategies over PRIME enables 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 provides to PRIME an increasing of 56%, on average, in the number of entities processed per increment. On the other hand, PRIME + AS increased 20% the number of processed entities while PRIME + top-400 increased 47%. Therefore, based on the results presented in Figures 7.7, 7.8 and 7.9, 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.

7.7 A Real-world Case Study

As stated in Section 7.6, we explore two scenarios involving streaming data. In the first experiment (see Section 7.6), involving two high-frequency data sources, we evaluate the PRIME technique against Streaming Metablocking (the baseline) in terms of effectiveness and efficiency. Here, to explore other research opportunities and scenarios, we apply PRIME over a real-world scenario with one low-frequency data source vs. one high-frequency data source. Moreover, we use a (time) window size of five min over PRIME since it achieved the best results in terms of effectiveness when compared with the application of window sizes equals to two and ten min. Overall, the main objectives of this experimental evaluation are: i) validate the application of PRIME in real-world scenarios in terms of effectiveness, and ii) evaluate the influence of different sets of attributes (which are considered during the blocking step) over the effectiveness results of PRIME. More specifically, in order to evaluate the application of PRIME in real-world scenarios, which consider streaming data, we conducted an experimental analysis involving events. The idea is to apply the proposed technique in order to define similarity relations between records (i.e., events) produced by two (or more) data sources. In other words, given a set of event records provided (in a streaming way) by two data sources, PRIME should identify and group into the same block records related to the same event in the real world. It is important to highlight that blocks generated by PRIME will represent the connected events provided by data sources for a specific time window, since this case study addresses streaming data incrementally (see the time window strategy discussed in Section 7.4). More specifically, during this case study, we explore football games of the Brazilian Championship 2019 as the events to be evaluated. To this end, we collected real-world data: tweets posted at Brazil (high-frequency data source) and information regarding football games from a sports monitoring API (low-frequency data source). Then, PRIME technique is applied as a clustering strategy to define which tweets (among all the tweets posted during the football game) are related to a specific football game.

7.7.1 Contextualization

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 to distribute news and engage with their audiences [40]. According to the survey [1], more than half of journalists stated 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 or sports events). The information provided by these services commonly address personal point of views (e.g., opinions, emotions, pointless babbles) about newsworthy events, as well as updates and discussions on these events [48]. For instance, journalists use social media frequently for sourcing news stories since the data provided by social media are faster access to elites, to the voice of the people, and to regions of the world that are otherwise difficult to access [96]. Regarding consumers, according to the survey [2], social networks outperformed print media (e.g., magazines, newspapers, flyers and newsletters) as their primary source of newsgathering.

Among social networking, we can detach Twitter. Twitter is a popular microblogging service with around 310 million monthly-active users, where users can post and read short text messages (known as tweets), as well as publish Web site links or share photos. For this reason, Twitter emerges as an important data source producing content from all over the world continually and as an essential host of sensors for events as they happen. An event, in the context of social media, can be understood as an occurrence of interest in the real world which instigates a discussion associated topic at a specific time [49]. The occurrence of an event is characterised by topic and time, and often associated with entities such as people and location. Commonly, the occurrence of an event causes a change in the volume of data workflow [32].

Events can be related to different contexts such as sports, concerts, festivals, natural disasters, terrorist attacks, and elections. In this sense, official data sources (e.g., governmental sources, official event website, or event monitoring companies) can be supplemented with information provided by social networks. For instance, given a final match of a championship whose information is provided by a sports website, it is possible to get real-time information from Twitter data related to this match. From this data, different analysis can be explored: the opinion provided by fans can influence odds of bookmakers, sentiment analysis regarding the match, identification of riots and accidents, and micro-events related to the match (e.g., goals, red cards, penalties, and the winner). Therefore, social networks are fundamental in the context of enriching event information provided by official data sources. Assuming the semi-structured characteristics of the data provided by Web sources and Social Media [25; 92], schema-agnostic blocking techniques can be used to cluster (i.e., as a clustering strategy) records from different data sources (for instance, the official sources and Twitter) that refer to the same real-world event [41; 105].

7.7.2 Results

As stated, in these experiments, we use two real-world pairs of data sources: a low-frequency source (i.e., information about a game from the monitoring sports API) and a high-frequency source (i.e., tweets from Twitter), as described in Table 7.2. Concerning the data provided by the monitoring sports API, we collected 20 records (i.e., games) referred to two rounds (20^{th} and 23^{rd}) of the Brazilian football championship 2019 (i.e., 10 games per round). The games provided by the sports API have seven attributes: stadium, name of the home team, name of the away team, name or nickname of the football players of the home team, name or nickname of the football players of the away team, nickname of the home team, and nickname of the away team. Regarding the tweets, we collected 591,900 tweets posted during the games of the two rounds of the Brazilian championship (the same rounds collected from the sports API). For each tweet, we consider three attributes (the others - e.g., id, source device, URLs and images links - are not useful for blocking): user name, full text of the tweet, and hashtags. For a first view, three attributes may seem to represent a small amount of data. However, notice that the full text of a tweet (with 280 characters) can provide a significant number of tokens (i.e., keywords) for each tweet. The amount of data is further increased due to the high-frequency behavior of Twitter, which provides 406,8 tweets per minute, on average. To explore the influence of attributes in the PRIME technique, we divided the attributes of the sports API into three attribute sets: $Set1 = \{stadium, name\_home\_team, name\_away\_team\}$, $Set2 = Set1 \cup \{name\_or\_nickname\_players\_home\_team, name\_or\_nickname\_players\_away\_team\}$, and $Set3 = Set1 \cup Set2 \cup \{nickname\_home\_team, nickname\_away\_team\}$. Regarding the computational infrastructure, we run our experiments in a cluster with six nodes (one master and five slaves), each one with one core. Each node has an Intel(R) Xeon(R) 2.10GHz CPU, 3GB memory, runs the 64-bit Windows 10 OS with a 64-bit JVM, Apache Kafka 2.1.0 and Apache Flink 1.7.1.

To evaluate PRIME in terms of effectiveness, we apply two metrics: Pair Quality (PQ) and Pair Coverage (PCov). PQ measures the accuracy/precision of a blocking technique, as described in Section 2.6. To calculate PQ, it is necessary to know the number of matches correctly identified by the blocking technique. For this reason, in this experiment, the entity blocks generated by PRIME were submitted to a human review in order to check which tweets were correctly linked to each specific game. Therefore, it is possible to determine the number of matches correctly identified among all matches determined by PRIME and, consequently, calculate PQ. Regarding PCov, it is a quality metric based on Pair Completeness (see Section 2.6) to measure the recall of the blocking technique. Notice that we just calculate the number of matches correctly identified by the blocking technique since computing the number of true matches (for the real-world) among almost 600,000 tweets is an extremely costly and error-prone work. Thus, this scenario does not provide a gold-standard, making Pair Completeness impossible to calculate. For this reason, we apply the Pair Coverage metric. PCov estimates the level of recall (of a blocking technique) considering the maximum number of matches correctly identified among all matches determined by a blocking technique. Thus, in this experiment, we consider as a gold-standard the maximum number of matches correctly identified among the results generated by PRIME taking into account the three sets of attributes. For instance, if PRIME with $Set1$ achieved 25 matches correctly identified, PRIME with $Set2$ achieved 50 matches correctly identified and PRIME with $Set3$ achieved 100 matches correctly identified, applying PCov is possible to determine the proportional level of recall among the attribute sets used in PRIME. Therefore, $PCov_{PRIME+Set1} = 0.25$, $PCov_{PRIME+Set2} = 0.5$ and $PCov_{PRIME+Set3} = 1$.

In Figures 7.10 (a) and (b), it is possible to note the average of PQ and PCov for the three attribute sets in each round of the championship. Notice that, in terms of PCov, PRIME+$Set3$

Table 7.2: Data sources characteristics.

Round	|Twitter|	|Sport API|	|Identified by PRIME|	|Correctly identified|
20^{th} round	294,039	10	4,174	3,391
23^{rd} round	297,861	10	4,414	3,011

achieved the best results when compared with PRIME+$Set1$ and PRIME+$Set2$ for both rounds. It occurs due to $Set3$ considers other semantical information (i.e., the nicknames of the teams) beyond the information already considered by $Set1$ and $Set2$. However, evaluating the results of PRIME+$Set1$ and PRIME+$Set2$, PRIME+$Set1$ achieved better results for both rounds, especially for the 23^{rd} round (Figure 7.10 (b)). The main reason for this behavior is related to the semantic context of this scenario. The attribute $Set2$ adds the information regarding the name of the football players involved in the game. On the other hand, people name is a piece of common information in tweets. For instance, a tweet may mention the name "*Ronaldo*", but this "*Ronaldo*" is just a friend of the user that posted the tweet, not the famous football player. Thus, the application of the football players' names can insert non-similar tweets into the cluster (related to a specific game) by considering similar names and, consequently, decrease the effectiveness of PRIME.

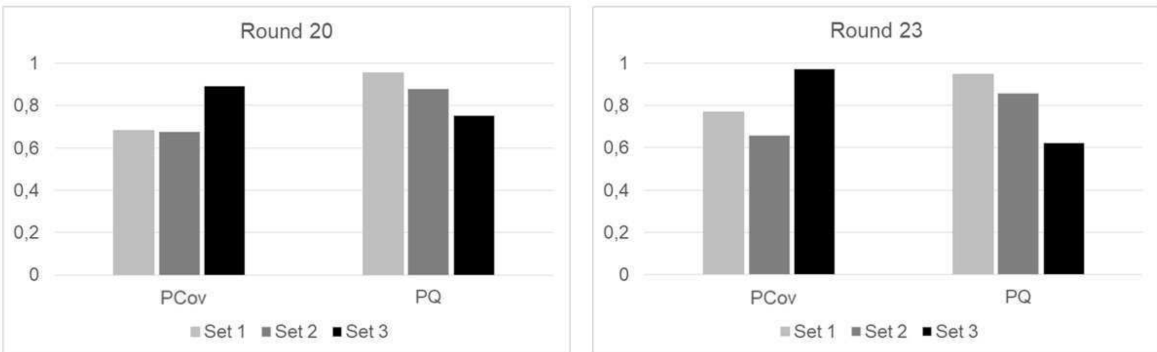

Figure 7.10: PCov and PQ results of PRIME for each attribute set. The results are given by the average of all games per round.

Regarding PQ, the results maintain a standard for both rounds, as illustrated in Figures 7.10 (a) and (b). PRIME+$Set1$ achieved better results when compared with PRIME+$Set2$ which in turn achieved better results than PRIME+$Set3$. This behavior can be explained if we consider the amount of semantic information contained in each attribute set. Since PRIME+$Set3$ takes into account more attributes (and semantic information) than PRIME+$Set2$, PRIME+$Set3$ tends to insert more tweets into the clusters and some of them can be wrongly inserted (decreasing the PQ). The same occurs for PRIME+$Set2$ and PRIME+$Set1$. Analyzing PCov and PQ simultaneously, we can highlight that considering a small number of attributes (e.g., PRIME+$Set1$), PRIME tends to achieve high values for PQ, but low values for PCov. On the other hand, taking into account a higher number of attribute values (e.g., PRIME+$Set3$), the blocking technique tends to achieve low values for PQ, but high values for PCov. In this sense, it is necessary to determine strategies to select (or consider) attributes in order to positively impact the effectiveness results. For instance, based on the results, the application of the football players' names (in $Set2$) decreases the PCov instead of increasing. Therefore, the development of strategies and algorithms able to select relevant attributes to benefit the blocking techniques emerges as an open problem.

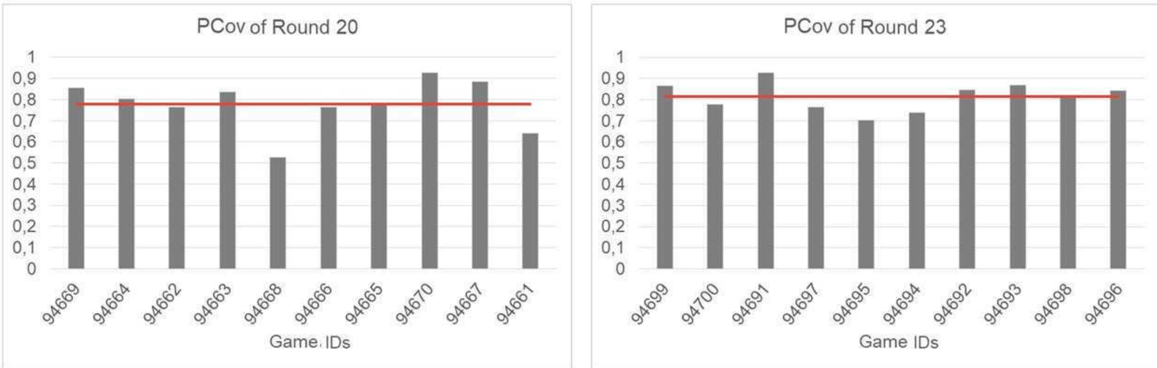

Figure 7.11: Average of PCov results of PRIME for each game.

In Figures 7.11 (a) and (b), it is possible to analyze in more details the PCov results for each game. The columns represent the average (per game) of PCov values for the three attribute sets and the red line denotes the average of PCov for the round in question. Regarding round 20 (see Figure 7.11 (a)), only the games 94668 and 94661 present results significantly lower than average (i.e., PCov between 0.5 and 0.63). It means that, on average, PRIME achieved only 50-63% of the best result (i.e., applying an attribute set able to reach the best PCov value). Clearly, these results indicate a high disparity between the results reached by PRIME applying the different attribute sets. For a better understanding, the results illustrated in Figure 7.12 (a) demonstrate the PCov values achieved by PRIME for each attribute set and the lines denote the average of PCov values for each attribute set in the round. Evaluating the games 94668 and 94661 again, notice the disparity between the results reached by PRIME applying the three attribute sets. For instance, PRIME+$Set1$ identified less than 20% of the tweets related to the game 94668 when compared with PRIME+$Set3$ (the best result). Regarding round 23 (see Figure 7.11 (b)), we can highlight the game 94695, which presents PCov results significantly lower than average. Analyzing the results of the game 94695 in Figure 7.12 (b), the reason for the low PCov value (on average) is motivated by the low PCov achieved by PRIME+$Set2$, which identified less than 40% of the tweets related to the game 94695 when compared with PRIME+$Set3$ (the best result). Regarding PCov, we can detach the different levels of PCov depending on the attributes considered by the blocking technique. Therefore, the selection (or not) of an attribute (even that the attribute is apparently relevant, such as the names of the players) to be considered during the blocking can be decisive to achieve good effectiveness results.

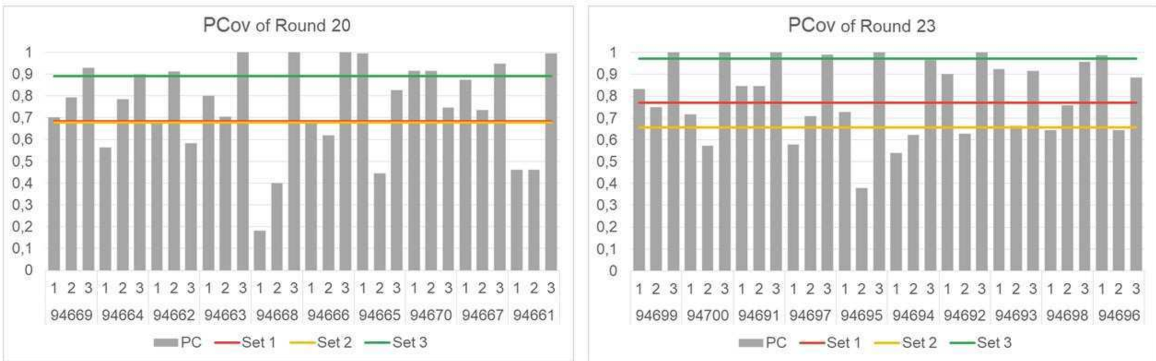

Figure 7.12: PCov results of PRIME for each attribute set per game.

In Figures 7.13 (a) and (b), the columns represent the average (per game) of PQ values for the three attribute sets and the red line denotes the average of PQ for the round in question. For the rounds 20 (see Figure 7.13 (a)) and 23 (see Figure 7.13 (b)), it is possible to highlight the games 94665 and 94693, which achieved low PQ values. PRIME reached a PQ value of 0.51 for the game 94665 and 0.45 for the game 94693. To understand better the reason for these low values for PQ it is necessary to take into account two aspects: the attribute sets applied to PRIME (see Figures 7.14 (a) and (b)) and the number of tweets related to these games. In Figure 7.14 (a), for the game 94665, PRIME+$Set1$ achieved a PQ value near 1 since in $Set1$ only the name of the teams and the name of the stadium are considered. Notice that the attributes of $Set1$ provide specific information regarding the game. For this reason, the precision of the blocking technique tends to be better when compared, for instance, with the PRIME+$Set3$. For PRIME+$Set3$, which achieved a PQ value of 0.23, the attributes of $Set3$ provide information related to the name or nickname of the players and nicknames of the teams. As stated, names and nicknames commonly provide generic information, not only related to specific content of the game. The same occurs for game 94693, as illustrated in Figure 7.14 (b). For instance, one of the nicknames of the team Ceará (in the games 94665 and 94693) is "*Vovô*" (in English, grandfather). Thus, any tweet that mentions "*Vovô*", even if it is being used to designate grandfather, can be inserted into the cluster related to the games 94665 and 94693.

On the other hand, the pruning algorithms are applied in order to discard low similar entities. Thus, the scenario where a tweet mentions "*Vovô*" to designate grandfather will hardly have other similar tokens with the game 94665. Therefore, pruning algorithms should discard this tweet. However, considering that PRIME applies a WNP-based pruning algorithm, the pruning strategy discards entities based on the average similarity (i.e., the threshold) of neighboring entities. For this reason, in scenarios with a low number of tweets with high similarity with a game, the pruning strategy tends to maintain some tweets with low similarity with the game due to the low value for the threshold. This fact clearly affects games 94665 and 94693, whose number of tweets truly similar related to the games are 80 and 58 (from an average of 341), respectively. Considering the real-world aspect, this scenario can be explained by the size (in number of fans) of these teams in Brazil. The games 94665 and 94693 involve the teams Ceará, Goiás and CSA, which summing the fans represent a little more 1% of the Brazilian population, according to a survey conducted by Datafolha[7]. Thus, considering the low number of fans of these teams, the number of tweets related to these games tends to be low and, consequently, cause the problem of few similar tweets related to a game (impacting the effectiveness of pruning algorithms). Summarizing, we would like to highlight the scenario involving a few number of similar entities as a challenge to be addressed by novel pruning algorithms and blocking techniques.

[7] http://media.folha.uol.com.br/datafolha/2019/09/17/77975ecbd43522f8fe59b29b8f93d09atdp.pdf (accessed in Nov, 18th 2019)

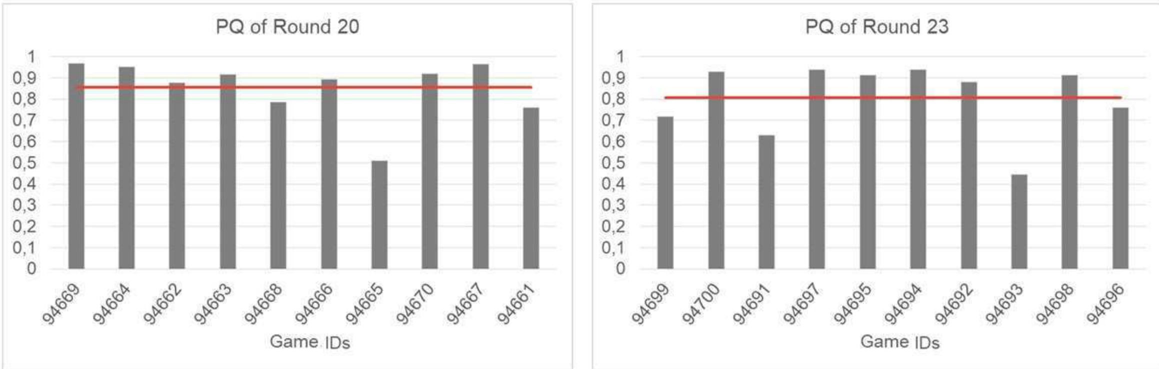

Figure 7.13: Average of PQ results of PRIME for each game.

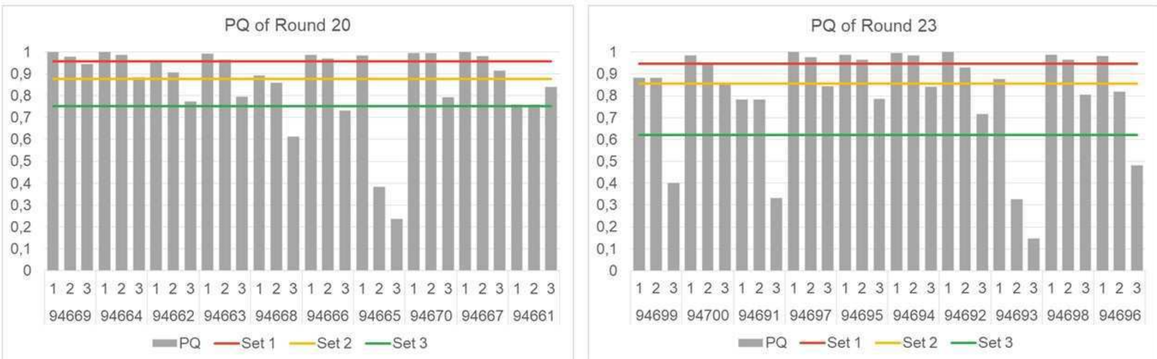

Figure 7.14: PQ results of PRIME for each attribute set per game.

7.8 Final Considerations

In this chapter, we address the challenges involving heterogeneous data, parallel computing, incremental processing, and streaming data. To the best of our knowledge, there is a lack of blocking techniques proposed to address all these challenges simultaneously. In this sense, we propose a novel schema-agnostic blocking technique capable to incrementally process streaming data in parallel. In order to enhance the efficiency of the proposed technique, we also propose the strategies: attribute selection (which discards useless attributes from the entities) and top-n neighborhood (which maintains only the top "n" neighbor entities for each entity). Based on the experimental results, we can highlight that our technique presents better efficiency results than the state-of-the-art technique (i.e., Streaming Metablocking) without significant impacts on the effectiveness. This main result was achieved due to the reduction in the number of MapReduce jobs in the proposed workflow. Moreover, the application of the proposed strategies improved the results achieved by our blocking technique in terms of efficiency and effectiveness. The contributions proposed in this chapter are part of the development of blocking techniques that can be coupled with the execution model proposed in Chapter 4. In particular, the PRIME technique is directly related to the *data reading and interpretation*, *block generation* and *pruning of comparisons* steps (of the proposed execution model) in the context of streaming data and incremental processing. To this end, in this chapter, we propose novel strategies to allow blocking techniques to face both challenges. In the next chapter, final considerations and future works are discussed.

Chapter 8

Conclusions and Future Works

In this chapter, the conclusions related to the proposed artefacts (i.e., execution model and blocking techniques) are presented. Furthermore, the weaknesses and topics do not address by the proposed artefacts are highlighted as future directions of this research, with the objective of detaching open areas, identified during the development of this study.

8.1 Conclusions

Taking into account the challenges described during the previous chapters, all of them are considered as opportunities to explore and develop researches. In this sense, the scope of this thesis is related to trade-off of how to provide efficiency to the ER task without reducing the effectiveness of the results. Thus, the main objective of the present work is to propose a distributed execution model for schema-agnostic blocking techniques in scenarios involving noisy/clean data, large data sources, and batch/streaming data. In this context, the proposed model assists the ER task in data integration scenarios involving Web data. Furthermore, parallel blocking techniques are investigated, proposed and evaluated in order to be coupled to the proposed model, aiming to improve the efficiency of ER task. Such blocking techniques seek to achieve better results in terms of effectiveness and efficiency, when compared with other state-of-the-art techniques. In this sense, the present work achieved the following results:

- A distributed execution model for schema-agnostic blocking techniques capable of addressing the necessities inherent to the application profile selected by the user (e.g., a data specialist);

- An architecture, with different workflows (see Figure 4.3), to execute parallel schema-agnostic blocking techniques, based on the proposed execution model;
- SS-Metablocking: a Metablocking-based technique in parallel, which applies the framework Spark as a programmable model. The parallelization of the technique aims to improve efficiency, and reduce the execution time as a whole;
- GWNP: a pruning algorithm capable of evaluating the weight of blocking graph edges locally and globally, in order to improve the effectiveness of the generated blocks (i.e., high-quality blocks);
- A load balancing technique that takes into account the cardinality (i.e., number of comparisons) of the blocks to guide the distribution of comparisons between the nodes of a distributed infrastructure. The application of the load balancing technique aims to enhance the efficiency of the blocking technique and optimize the resource consumption of the distributed infrastructure;
- NA-BLOCKER: a schema-agnostic blocking technique capable of tolerating noisy data and extracting schema information from the data to improve the effectiveness of the generated blocks;
- PRIME: an incremental and parallel-based schema-agnostic blocking technique able to deal with streaming and incremental data, as well as to minimize the challenges related to both scenarios;
- Top-n strategy: a top-n neighborhood strategy, which maintains only the "n" most similar entities (i.e., neighbor entities) of each entity, enabling the proposed technique to deal with large data sources;
- Attribute Selection: an attribute selection algorithm, which discards useless attributes from the entities to enhance efficiency and minimize memory consumption.

In addition, experimental evaluations were conducted to validate the efficacy and efficiency of the proposed blocking techniques. Based on the results obtained from the experiments, the novel blocking techniques present solutions that significantly decrease the execution time of the ER task and achieved effectiveness results equivalent (or higher) to those achieved by the competing techniques. Therefore, the evaluation results have given support to the general hypothesis of the work.

To deal with the scenario involving semi-structured data in batch, without noise (in the data) and large data sources, we propose the SS-Metablocking technique. This novel technique, an evolution of the work [37], applies parallel resources (e.g., accumulators and broadcast variables) provided by Spark to enhance efficiency and effectiveness of the blocking task. Also, we exploit these resources to propose the GWNP pruning algorithm and the *Cardinality-based* load balancing technique. The GWNP emerges as a possible pruning algorithm to be applied in the Metablocking in order to improve effectiveness results without significant impacts on the efficiency of the blocking technique. Regarding the load balancing, to guide the even distribution of comparisons (between entities) among the nodes, the *Cardinality-based* technique takes into account the amount of comparisons (cardinality) to be performed in each block. Based on the block cardinality, a greedy algorithm evaluates the amount of available nodes and defines to which node the entity pairs will be emitted. Based on the experimental results, we can highlight that the SS-Metablocking presents better results regarding efficiency and effectiveness than the state-of-the-art approach [37].

Regarding scenarios involving semi-structured data (in batch) and noisy data, we propose the NA-BLOCKER technique. This novel schema-agnostic blocking technique was developed to block entities that present noise in its attribute values. To this end, the NA-BLOCKER applies LSH in order to hash the attribute values of the entities and enable the generation of high-quality blocks, even with the presence of noise in the attribute values. Moreover, NA-BLOCKER applies attribute clustering strategies, by extracting information regarding the schema from the data (i.e., group similar attributes based on the data) and enhance the quality of the generated blocks. Based on the experimental results, we can highlight that NA-BLOCKER presents better results regarding effectiveness and aggregate cardinality (i.e., the number of comparisons to be performed after blocking) than the state-of-the-art technique. Considering that the proposed distributed execution model can host other blocking techniques, notice that the work [90] can be applied to address scenarios involving large data sources and noisy data.

To address scenarios involving semi-structured streaming data, large data sources and incremental processing, we propose the PRIME technique. PRIME is a schema-agnostic blocking technique capable to incrementally process entities received in streaming way. To improve efficiency, PRIME applies a novel parallel workflow, which reduces the number of MR jobs when compared with state-of-the-art techniques. Moreover, we propose an attribute selection algorithm and a top-n neighborhood strategy to better manage the computational resources and improve efficiency to the blocking technique. Considering the results obtained by the experimental results, it is possible to detach that PRIME presents better efficiency results than the state-of-the-art technique without significant impacts on the effectiveness. These results were achieved by the reduction in the number of MapReduce jobs in the proposed workflow and the application of the attribute selection algorithm and the top-n neighborhood strategy, which enhance efficiency and effectiveness of the PRIME technique.

On the other hand, it is important that the research also provides the weaknesses of the contributions. Thus, in this thesis, we can highlight the following weaknesses:

- In the execution model, the selection of the workflow and the blocking technique is done manually. Therefore, it is inefficient and susceptible to errors since each scenario requires different workflows and blocking techniques to meet their needs.

- The proposed techniques were evaluated using clusters with a small number of nodes. For instance, we did not conduct experiments involving hundreds or thousands of nodes. With a large number of nodes, it is possible to better evaluate the behavior of the proposed techniques and the results in terms of efficiency.

- The SS-Metablocking, described in Section 5, proposes the application of GWNP algorithm to prune the similarity graph. However, this algorithm depends on broadcast variables and accumulators, resources provided by some parallel frameworks (e.g., Spark), but do not provide by others (e.g., Hadoop). Thus, this contribution is not generic and completely depends on the parallel framework applied in the cluster.

- The NA-BLOCKER, proposed in Section 6, is executed in a standalone mode and present efficiency issues to achieve better results for effectiveness. Thus, efficiency aspects did not explore in this research. Moreover, only typos inserted synthetically in the data sources were evaluated, which can be considered as a threat to validation.

- The PRIME, illustrated in Section 7, is a pioneer blocking technique for handle streaming data. In this sense, several aspects did not consider, such as different frequency of sending data, dynamic resource allocation and a huge mass of data being processed in each increment.

- A toolkit, based on the proposed execution model and able to integrate different blocking techniques, has not been implemented. This toolkit would have helped in the validation and integration of the contributions of this thesis.

8.2 Future Works

Considering the weaknesses stated in Section 8.1, in this section, we highlight the main open areas, research opportunities, and future work in the context of the developed work.

Load balancing techniques. As future work in terms of load balancing, it is possible to highlight improvements of the Cardinality-based technique proposed in this thesis. In this sense, the technique should consider internal comparisons, i.e., the comparisons performed by similarity functions (e.g., Jaccard) to estimate the weight edge in the Metablocking step. The load balancing technique could provide a fine-grained level of distribution since it takes into account the lowest level of comparison (i.e., the comparison of attribute values in similarity functions). Moreover, another possible future work is related to propose a load balancing technique for scenarios involving streaming data. Notice that, in the streaming data context, the blocking task should be executed quickly since the data is received in short time intervals. On the other hand, load balancing techniques tend to increase the time execution since they need time to compute the number of comparisons and distribute the comparisons among the nodes. Therefore, load balancing techniques in the context of streaming data should take into account the harmonic relation between a fair distribution of comparisons and execution time.

Token extraction. Considering that the tokens guide the block generation (in token-based techniques), the extraction of tokens from the attribute values directly impacts on blocking step. In this sense, the development of novel strategies of token extraction is an open area to be explored. For instance, for long-text descriptions (in attribute values), such as tweets[1] and product descriptions, content extraction algorithms [98; 100; 15] can be applied. These algorithms aim to extract only the main content (i.e., relevant words) of a text and discard irrelevant information. Then, content extraction algorithms can be applied to consider only relevant keywords (i.e., tokens) from the long-text descriptions and, consequently, minimize the number of tokens considered during the blocking step. Hence, the application of content extraction algorithms tends to minimize resource consumption and improve efficiency.

Pruning algorithms. Since Metablocking emerged as a promising technique in the semi-structured data context, a wide number of Metablocking-based techniques were proposed. In this sense, the pruning step (in Metablocking) is fundamental in terms of improving the effectiveness of blocking techniques. On the other hand, complex pruning algorithms, which tend to improve the effectiveness, commonly handle more information or apply costly pruning strategies and, therefore, increase the execution time of the blocking technique as a whole. Then, the development of efficient pruning algorithms is considered as an open area to be explored. These new pruning algorithms should aim to improve the effectiveness results without meaningly increasing the execution time.

Different types of noisy data. The NA-BLOCKER technique proposed in this work (see Chapter 6) deals with noisy data. However, this technique considers two types of noise in the data: typos and misspellings. Clearly, there are other types of noise present on the data, such as pronunciation errors, slang, and abbreviations. Hence, the development of noise-aware blocking techniques able to handle different types of noise is a possible future work. Another point to be explored in the context of noisy data is related to propose efficient techniques. As stated in Chapter 6, noise-aware blocking techniques need more time to be executed since they apply additional steps to manage the data and generate high-quality blocks even in the presence of noise. Therefore, the application of parallel computing emerges as a possible solution to propose new efficient noise-aware blocking techniques, as discussed in [90].

Incremental blocking techniques over streaming data. Considering that there is a lack of blocking techniques dealing with streaming data in an incremental way, this context clearly appears as a promising area to be explored. Although PRIME (see Chapter 7) addresses the context of streaming data and incremental processing, several issues are

[1]https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/entities-object

still uncovered and can be explored, such as the application of different kinds of windows (e.g., based on the amount of data), management of noisy data, exploration of schema information, and application of machine learning techniques to assist the blocking technique. Regarding the different kinds of windows, it is necessary to remember the importance of discarding some parts of the information (entities in our case). The application of windows aims to avoid the high consumption of computational resources since entities are received infinitely and processed incrementally. However, different kinds of window strategies (i.e., not only time window, such that applied in PRIME) can be proposed and applied to the blocking techniques (e.g., PRIME). For instance, one possible window strategy is to maintain (and prioritize) blocks constantly updated during the increments. Another future work is related to noisy data. In this sense, the strategies developed in this work (as well as the strategies proposed by other works) to handle noisy data can be combined with PRIME in the sense of addressing streaming and noisy data simultaneously. Blocking techniques that address streaming and noisy data are directly related to the application profiles described during this thesis. For this reason, such techniques can be applied to the execution model proposed in Chapter 4. Regarding explore schema information, although the PRIME technique applies an attribute selection algorithm, which explores information related to the schema to discard useless attributes from the entities, the application of attribute grouping (see Chapter 6) is still open. Therefore, efficient strategies for attribute grouping can be proposed and applied in PRIME in order to improve effectiveness. Finally, we would like to highlight the possibility of adapting other existing blocking techniques in the proposed execution model (see Chapter 4), such as progressive blocking techniques [4; 92] and blocking based on machine learning concepts [52; 82].

Unbalance in data increments. One aspect does not addressed by this thesis is related to the unbalance in data increments. In this context, the number of entities present in each micro-batch to be processed by blocking techniques varies over time. This behavior commonly occurs in streaming sources which present load peaks. For instance, traffic sensors commonly send a high number of data during the rush hours. On the other hand, the number of produced data is reduced during low-traffic hours. In this sense, parallel-based blocking techniques should be able to consume resources dynamically. Therefore, dynamic schedulers that provide an on-demand allocation of resources in distributed infrastructures should be applied [91; 93]. 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 unbalance in data increments is a problem faced by incremental techniques, the development of dynamic schedulers for blocking techniques emerges as an open area to be explored.

Real-world applications. To validate the PRIME technique in real-world scenarios, we apply PRIME to clustering events using data from Twitter and a football game API. In this sense, it is possible to highlight the opportunity to apply schema-agnostic blocking techniques over other real-world streaming sources, such as news feeds, social networks, and sensors [47]. More specifically, we would like to explore scenarios involving two streaming sources with high latency of data (i.e., a large amount of data is provided by the source in a small period of time). Such scenarios potentiate the challenges involving streaming data and large data sources, as discussed in Chapter 7.

Development of a toolkit for schema-agnostic blocking techniques. Based on the execution model and the parallel architecture proposed in Chapter 4, it is possible to develop a toolkit for parallel schema-agnostic blocking techniques. In this sense, the toolkit can receive from the user (e.g., a data specialist) the application profile. Considering the application profile, the best workflow (see Figure 4.3) and blocking technique are selected and, posteriorly, executed according to the application characteristics. This toolkit should host different schema-agnostic blocking techniques (such as the three techniques proposed in this thesis) in order to address different scenarios faced by blocking techniques (for instance, noisy data, streaming data, focus on efficiency/effectiveness). Finally, the toolkit can benefit from the parallel architecture and be implemented over a distributed infrastructure, providing efficiency to the blocking techniques hosted by the toolkit.

Bibliography

[1] Global social journalism study. https://www.cision.com/us/resources/white-papers/2019-sotm/?sf=false, 2017. Accessed: 2019-09-12.

[2] Social networks finally bypassed print newspapers as a primary source of news. https://www.adweek.com/digital/social-networks-finally-bypassed-print-newspapers-as-a-primary-source-of-news/, 2018. Accessed: 2019-09-12.

[3] Sumeet Agarwal, Shantanu Godbole, Diwakar Punjani, and Shourya Roy. How much noise is too much: A study in automatic text classification. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 3–12. IEEE, 2007.

[4] Yasser Altowim and Sharad Mehrotra. Parallel progressive approach to entity resolution using mapreduce. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 909–920. IEEE, 2017.

[5] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. SIAM, 2014.

[6] Tiago Araújo, Carlos Eduardo S Pires, and Kostas Stefanidis. Noisy-aware blocking over heterogeneous data. In *International Semantic Web Conference (P&D/Industry/BlueSky)*, 2018.

[7] Tiago Brasileiro Araújo, Cinzia Cappiello, Nadia Puchalski Kozievitch, Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, and Monica Vitali. Towards reliable data analyses for smart cities. In *Proceedings of the 21st International Database Engineering & Applications Symposium*, pages 304–308. ACM, 2017.

[8] Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, and Thiago Pereira da Nóbrega. Spark-based streamlined metablocking. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pages 844–850. IEEE, 2017.

[9] Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, Thiago Pereira da Nóbrega, and Dimas C Nascimento. A fine-grained load balancing technique for improving partition-parallel-based ontology matching approaches. *Knowledge-Based Systems*, 111:17–26, 2016.

[10] Tiago Brasileiro Araújo, Carlos Eduardo Santos Pires, Demetrio Gomes Mestre, Thiago Pereira da Nóbrega, Dimas Cassimiro do Nascimento, and Kostas Stefanidis. A noise tolerant and schema-agnostic blocking technique for entity resolution. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 422–430. ACM, 2019.

[11] Naser Ayat, Reza Akbarinia, Hamideh Afsarmanesh, and Patrick Valduriez. Entity resolution for probabilistic data. *Information Sciences*, 277:492–511, 2014.

[12] Carlo Batini and Monica Scannapieco. *Data and Information Quality: Dimensions, Principles and Techniques*. Springer, 2016.

[13] Zohra Bellahsène, Angela Bonifati, and Erhard Rahm. *Schema matching and mapping*. Springer, 2011.

[14] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information systems*, 35(6):615–636, 2010.

[15] Saroj Kr Biswas, Monali Bordoloi, and Jacob Shreya. A graph based keyword extraction model using collective node weight. *Expert Systems with Applications*, 97:51–59, 2018.

[16] Tiago Brasileiro Araújo, Kostas Stefanidis, Carlos Eduardo Santos Pires, Jyrki Nummenmaa, and Thiago Pereira da Nóbrega. Incremental blocking for entity resolution over web streaming data. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 332–336. ACM, 2019.

[17] A.Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29. IEEE Comput. Soc, 1997.

[18] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[19] Ling Chen, Weidong Gu, Xiaoxue Tian, and Gencai Chen. Ahab: Aligning heterogeneous knowledge bases via iterative blocking. *Information Processing & Management*, 56(1):1–13, 2019.

[20] Peter Christen. Febrl–an open source data cleaning, deduplication and record linkage system with a graphical user interface (demonstration session. In *In ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'08*. Citeseer, 2008.

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

[22] Peter Christen et al. Towards parameter-free blocking for scalable record linkage. 2007.

[23] Peter Christen, Ross Gayler, and David Hawking. Similarity-aware indexing for real-time entity resolution. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1565–1568. ACM, 2009.

[24] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. End-to-end entity resolution for big data: A survey. *arXiv preprint arXiv:1905.06397*, 2019.

[25] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015.

[26] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2006.

[27] Gustavo de Assis Costa and José Maria Parente de Oliveira. A blocking scheme for entity resolution in the semantic web. In *2016 IEEE 30th international conference on Advanced Information networking and applications (AINA)*, pages 1138–1145. IEEE, 2016.

[28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[29] Leon Derczynski, Alan Ritter, Sam Clark, and Kalina Bontcheva. Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *RANLP*, pages 198–206, 2013.

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

[31] Xin Luna Dong and Divesh Srivastava. Big data integration. *Synthesis Lectures on Data Management*, 7(1):1–198, 2015.

[32] Wenwen Dou, Xiaoyu Wang, William Ribarsky, and Michelle Zhou. Event detection in social media data. In *IEEE VisWeek Workshop on Interactive Visual Text Analytics-Task Driven Analytics of Social Media Content*, pages 971–980, 2012.

[33] Mauro Dragoni, Marco Federici, and Andi Rexha. An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Information processing & management*, 56(3):1103–1118, 2019.

[34] Uwe Draisbach, Felix Naumann, Sascha Szott, and Oliver Wonneberg. Adaptive windows for duplicate detection. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1073–1083. IEEE, 2012.

[35] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.

[36] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking: Realizing scalable entity resolution over large, heterogeneous data. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 411–420. IEEE, 2015.

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

[38] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. Minoaner: Schema-agnostic, non-iterative, massively parallel resolution of web entities. *arXiv preprint arXiv:1905.06170*, 2019.

[39] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007.

[40] Mateusz Fedoryszak, Brent Frederick, Vijay Rajaram, and Changtao Zhong. Real-time event detection on social data streams. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2774–2782. ACM, 2019.

[41] Wei Feng, Chao Zhang, Wei Zhang, Jiawei Han, Jianyong Wang, Charu Aggarwal, and Jianbin Huang. Streamcube: hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1561–1572. IEEE, 2015.

[42] Luca Gagliardelli, Giovanni Simonini, Domenico Beneventano, and Sonia Bergamaschi. Sparker: Scaling entity resolution in spark. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT*, pages 602–605, 2019.

[43] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[44] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2016.

[45] Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. Entity matching on web tables: a table embeddings approach for blocking. In *EDBT*, pages 510–513, 2017.

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

[47] Francesco Guerra, Paolo Sottovia, Matteo Paganelli, and Maurizio Vincini. Big data integration of heterogeneous data sources: The re-search alps case study. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 106–110. IEEE, 2019.

[48] Mahmud Hasan, Mehmet A Orgun, and Rolf Schwitter. Twitternews: real time event detection from the twitter data stream. *PeerJ PrePrints*, 4:e2297v1, 2016.

[49] Mahmud Hasan, Mehmet A Orgun, and Rolf Schwitter. A survey on real-time event detection from the twitter data stream. *Journal of Information Science*, 44(4):443–463, 2018.

[50] Mahmud Hasan, Mehmet A Orgun, and Rolf Schwitter. Real-time event detection from the twitter data stream using the twitternews+ framework. *Information Processing & Management*, 56(3):1146–1165, 2019.

[51] Marwan Hassani. Overview of efficient clustering methods for high-dimensional big data streams. In *Clustering Methods for Big Data Analytics*, pages 25–42. Springer, 2019.

[52] Changqin Huang, Jia Zhu, Xiaodi Huang, Min Yang, Gabriel Fung, and Qintai Hu. A novel approach for entity resolution in scientific documents using context graphs. *Information Sciences*, 432:431–441, 2018.

[53] Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. String similarity joins: An experimental evaluation. *Proceedings of the VLDB Endowment*, 7(8):625–636, 2014.

[54] Wuyang Ju, Jianxin Li, Weiren Yu, and Richong Zhang. igraph: an incremental data processing system for dynamic graph. *Frontiers of Computer Science*, 10(3):462–476, 2016.

[55] Batya Kenig and Avigdor Gal. Mfiblocks: An effective blocking algorithm for entity resolution. *Information Systems*, 38(6):908–926, 2013.

[56] Taehong Kim, Mi-Nyeong Hwang, Young-Min Kim, and Do-Heon Jeong. Entity resolution approach of data stream management systems. *Wireless Personal Communications*, 91(4):1621–1634, 2016.

[57] Lars Kolb, Andreas Thor, and Erhard Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science-Research and Development*, 27(1):45–63, 2012.

[58] Lars Kolb, Andreas Thor, and Erhard Rahm. Don't match twice: redundancy-free similarity computation with mapreduce. In *Proceedings of the Second Workshop on Data Analytics in the Cloud*, pages 1–5. ACM, 2013.

[59] Chao Kong, Ming Gao, Chen Xu, Weining Qian, and Aoying Zhou. Entity matching across multiple heterogeneous data sources. In *International Conference on Database Systems for Advanced Applications*, pages 133–146. Springer, 2016.

[60] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[61] Lingli Li, Jianzhong Li, and Hong Gao. Rule-based method for entity resolution. *IEEE Transactions On Knowledge And Data Engineering*, 27(1):250–263, 2015.

[62] Huizhi Liang, Yanzhe Wang, Peter Christen, and Ross Gayler. Noise-tolerant approximate blocking for dynamic real-time entity resolution. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 449–460. Springer, 2014.

[63] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

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

[65] Yongtao Ma and Thanh Tran. 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*, pages 325–334. ACM, 2013.

[66] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[67] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. Citeseer, 2000.

[68] Demetrio Gomes Mestre and Carlos Eduardo Santos Pires. Improving load balancing for mapreduce-based entity matching. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000618–000624. IEEE, 2013.

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

[70] Markus Nentwig and Erhard Rahm. Incremental clustering on linked data. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 531–538. IEEE, 2018.

[71] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, 2011.

[72] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. Deep sequence-to-sequence entity matching for heterogeneous entity resolu-

tion. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 629–638. ACM, 2019.

[73] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, 2017.

[74] Bernd Opitz, Timo Sztyler, Michael Jess, Florian Knip, Christian Bikar, Bernd Pfister, and Ansgar Scherp. An approach for incremental entity resolution at the example of social media data. In *AIMashup@ ESWC*. Citeseer, 2014.

[75] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323, 2015.

[76] George Papadakis, Gianluca Demartini, Peter Fankhauser, and Philipp Kärger. The missing links: Discovering hidden same-as links among a billion of triples. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, pages 453–460. ACM, 2010.

[77] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544. ACM, 2011.

[78] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 53–62. ACM, 2012.

[79] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederee, and Wolfgang Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2665–2682, 2013.

[80] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. Meta-blocking: Taking entity resolutionto the next level. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1946–1960, 2014.

[81] George Papadakis and Themis Palpanas. Blocking for large-scale entity resolution: Challenges, algorithms, and practical examples. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, volume 00, pages 1436–1439, 2016.

[82] George Papadakis, George Papastefanatos, and Georgia Koutrika. Supervised meta-blocking. *Proceedings of the VLDB Endowment*, 7(14):1929–1940, 2014.

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

[84] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. A survey of blocking and filtering techniques for entity resolution. *CoRR*, abs/1905.06167, 2019.

[85] Xiangnan Ren and Olivier Curé. Strider: A hybrid adaptive distributed rdf stream processing engine. In *International Semantic Web Conference*, pages 559–576. Springer, 2017.

[86] Alan Filipe Santana, Marcos André Gonçalves, Alberto HF Laender, and Anderson A Ferreira. Incremental author name disambiguation by exploiting domain-specific heuristics. *Journal of the Association for Information Science and Technology*, 68(4):931–945, 2017.

[87] Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *Artificial Intelligence and Statistics*, pages 886–894, 2014.

[88] Jorge Silva and Fernando Silva. Feature extraction for the author name disambiguation problem in a bibliographic database. In *Proceedings of the Symposium on Applied Computing*, pages 783–789. ACM, 2017.

[89] Giovanni Simonini, Sonia Bergamaschi, and HV Jagadish. Blast: a loosely schema-aware meta-blocking approach for entity resolution. *Proceedings of the VLDB Endowment*, 9(12):1173–1184, 2016.

[90] Giovanni Simonini, Luca Gagliardelli, Sonia Bergamaschi, and HV Jagadish. Scaling entity resolution: A loosely schema-aware approach. *Information Systems*, 83:145–165, 2019.

[91] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing*, 14(2):217–264, 2016.

[92] Kostas Stefanidis, Vassilis Christophides, and Vasilis Efthymiou. Web-scale blocking, iterative and progressive entity resolution. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1459–1462. IEEE, 2017.

[93] Jinn-Tsong Tsai, Jia-Cen Fang, and Jyh-Horng Chou. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. *Computers & Operations Research*, 40(12):3045–3055, 2013.

[94] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. Linked open vocabularies (lov): a gateway to reusable semantic vocabularies on the web. *Semantic Web*, 8(3):437–452, 2017.

[95] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[96] Gerret Von Nordheim, Karin Boczek, and Lars Koppers. 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. *Digital Journalism*, 6(7):807–828, 2018.

[97] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[98] Yu Wang, David Fink, and Eugene Agichtein. Seeft: Planned social event discovery and attribute extraction by fusing twitter and web content. In *Ninth International AAAI Conference on Web and Social Media*, 2015.

[99] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[100] Clarissa Castellã Xavier and Marlo Souza. Extraction and classification of semantic data from twitter. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, pages 15–18. ACM, 2018.

[101] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)*, 36(3):15, 2011.

[102] Yang Yang, Yizhou Sun, Jie Tang, Bo Ma, and Juanzi Li. Entity matching across heterogeneous sources. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1395–1404. ACM, 2015.

[103] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[104] Mohammed J Zaki, Wagner Meira Jr, and Wagner Meira. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[105] Junsheng Zhang, Yunchuan Sun, and Changqing Yao. Semantically linking events for massive scientific literature research. *The Electronic Library*, 35(4):724–744, 2017.

[106] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. Autoblock: A hands-off blocking framework for entity matching. In *Proceedings of the Thirteenth ACM International Conference on Web Search and Data Mining*. ACM, 2020.

[107] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.