UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Enriching data analytics with incremental data cleaning and attribute domain management

Paulo Henrique de Oliveira

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Paulo Henrique de Oliveira

Enriching data analytics with incremental data cleaning and attribute domain management

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Caetano Traina Júnior

USP – São Carlos
June 2021

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

O48e

Oliveira, Paulo Henrique de
Enriching data analytics with incremental data cleaning and attribute domain management / Paulo Henrique de Oliveira; orientador Caetano Traina Junior. -- São Carlos, 2021.
155 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2021.

1. Data Analytics. 2. Data Quality. 3. Incremental Data Cleaning. 4. Attribute Domain. 5. Domain Index. I. Traina Junior, Caetano, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Paulo Henrique de Oliveira

Enriquecendo a análise de dados com limpeza incremental dos dados e gerenciamento dos domínios de atributos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Caetano Traina Júnior

USP – São Carlos
Junho de 2021

To my loved ones,
especially my family and my partner,
who have supported me on this journey.

ACKNOWLEDGEMENTS

O presente trabalho foi realizado com apoios da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) - Processos 2015/15392-7 e 2018/20360-5 e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

A Deus, pela vida, pelas inúmeras bênçãos e por todas as oportunidades concedidas.

Aos meus pais, Neiva e Paulo, e irmã, Aline, por nunca medirem esforços para investirem em minha educação, mesmo diante de tantas dificuldades pelas quais passamos, especialmente financeiras. Não poderia deixar de mencionar a ocasião em que venderam pertences para que pudessem pagar pelo meu vestibular. Cada detalhe contribuiu, de alguma maneira, para que eu chegasse até aqui. Obrigado por me ensinarem o valor da integridade e da perseverança.

Ao meu companheiro, Rafael, que tanto enriqueceu minha vida nos últimos anos. Sou grato por seu cuidado, pelo seu apoio incondicional e por me permitir aprender, a cada dia, a ser uma pessoa melhor e mais justa.

Ao meu orientador, Prof. Dr. Caetano Traina Junior, por ter me recebido para embarcar nessa jornada, por sua paciência e por todos os ricos ensinamentos. À Prof.ª Dr.ª Agma Juci Machado Traina, por seu apoio em tantos momentos, especialmente com o estágio BEPE.

Ao Prof. Dr. Ihab F. Ilyas, pela receptividade e pelos ensinamentos durante meu estágio na Universidade de Waterloo, no Canadá. Aos colegas do Data Systems Group (DSG). À Kayla, amiga tão especial que conheci no Canadá e ganhei para a vida.

Ao Prof. Dr. Daniel dos Santos Kaster, por confiar e investir em mim desde o mestrado, pela minha conexão com o Instituto de Ciências Matemáticas e de Computação (ICMC), por seu companheirismo no estágio no Canadá e, principalmente, por sua amizade.

Ao Grupo de Bases de Dados e Imagens (GBDI), por todas as colaborações e experiências. Por todas as amizades que fiz. Ao Lucas Scabora e à Mirela Cazzolato, por tantas conversas e por serem ombros amigos nos momentos de dificuldade.

Aos amigos e familiares que estiveram comigo em tantos momentos. Às amigas Mayara e Dayane, aos primos Priscyla e Matheus, ao Carlos, à tia Sônia, aos amigos da Universidade Estadual de Londrina (UEL), obrigado por terem me apoiado durante essa jornada.

Aos professores e funcionários do ICMC, pelo suporte ao longo desses anos. A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho e para a minha formação.

“The important work of moving the world forward
does not wait to be done by perfect [people].”
(George Eliot)

ABSTRACT

OLIVEIRA, P. H. **Enriching data analytics with incremental data cleaning and attribute domain management**. 2021. 155 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

In the present Big Data era, many businesses have become more data-driven, seeking to improve their decision-making processes based on solid Data Analytics practices. Several steps constitute the Data Analytics pipeline and all of them involve specific approaches and technologies, which are constantly evolving. In order to accommodate new needs and trends, there is always room for improvements in the steps of Data Analytics. In this context, this PhD research has focused on improving two of those steps: **(i)** data cleaning and **(ii)** data analysis. Regarding the first step, we addressed the problem of performing data cleaning *incrementally*, considering dynamic scenarios with incoming data batches, as well as *holistically*, that is, jointly taking into account multiple error detection criteria. As a result, we have developed an incremental data cleaning framework which significantly outperforms competitors, enabling higher efficiency while compromising little on repair quality, as well as addresses the problem in an innovative way, hence filling a gap in the literature. Regarding the second improved step, we addressed the problem of handling queries over an Attribute Domain, which consists of the set of stored values within a domain of attributes, usually across multiple relations. As a result, we have proposed three contributions: **(a)** the Domain Index, an access method for efficiently performing queries over Attribute Domains, which we refer to as Domain Queries; **(b)** a comprehensive case study of Domain Indexes applied to the medical domain, focusing on content-based Domain Queries for supporting physicians in decision-making; and **(c)** an approach for including support to Attribute Domains as first-class citizens in a Relational Database Management System (RDBMS). Together, those contributions target a distinct category of queries which, until the execution of this PhD research, had not been addressed in the literature elsewhere. Experimental results highlight the superior performance enabled by the Domain Index compared to existing techniques of modern RDBMSs, which not only are inefficient in several scenarios, but also are not always applicable. Ultimately, those contributions enrich data analyses down the road. Hence, this PhD research advances the state of the art in the field of Data Analytics, as well as opens several directions for future work.

Keywords: Data Analytics, Data Quality, Incremental Data Cleaning, Attribute Domain, Domain Index.

RESUMO

OLIVEIRA, P. H. **Enriquecendo a análise de dados com limpeza incremental dos dados e gerenciamento dos domínios de atributos**. 2021. 155 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2021.

Na presente era do *Big Data*, as organizações têm se tornado mais orientadas a dados, buscando melhorar seus processos de tomada de decisão com base em sólidas práticas de Análises de Dados. Diversos passos constituem o processo de Análises de Dados e todos envolvem abordagens e tecnologias específicas, que estão evoluindo constantemente. De maneira a acomodar as novas necessidades e tendências, há sempre espaço para melhorias nos passos de Análises de Dados. Nesse contexto, esta pesquisa de doutorado focou em melhorar dois desses passos: **(i)** limpeza de dados e **(ii)** análise de dados. Com relação ao primeiro, esta pesquisa lidou com o problema de realizar limpeza de dados *incrementalmente*, considerando cenários dinâmicos com novos lotes de dados, bem como *holisticamente*, isto é, juntamente levando em consideração múltiplos critérios para detecção de erros. Como resultado, desenvolveu-se um arcabouço para limpeza de dados incremental que supera significativamente os competidores, permitindo uma maior eficiência ao mesmo tempo em que se compromete pouco a qualidade de reparo, bem como trata o problema de forma inovadora, portanto preenchendo uma lacuna na literatura. Referente ao segundo passo, abordou-se o problema de manipular consultas sobre um Domínio de Atributos, que consiste no conjunto de valores que compõe um domínio de atributos, normalmente armazenados em múltiplas relações. Como resultado, propôs-se três contribuições: **(a)** o Índice de Domínio, um método de acesso voltado à execução eficiente de consultas sobre Domínios de Atributos, também chamadas de Consultas de Domínio; **(b)** um estudo de caso abrangente de Índices de Domínio aplicados sobre o domínio médico, focando em Consultas de Domínio baseadas em conteúdo para auxiliar profissionais da saúde no processo de tomada de decisão; e **(c)** uma abordagem para incluir suporte a Domínios de Atributos como cidadãos de primeira classe em um Sistema de Gerenciamento de Bancos de Dados Relacional (SGBDR). Juntas, essas contribuições focam em uma categoria distinta de consultas que, até a execução desta pesquisa de doutorado, não havia sido abordada na literatura. Resultados experimentais destacam o desempenho superior do Índice de Domínio comparado às técnicas existentes de SGBDRs modernos, que não somente são ineficientes sob diversos aspectos, como também não são aplicáveis a certos cenários. Portanto, essas contribuições também enriquecem análises de dados subsequentes. Assim, esta pesquisa de doutorado avança o estado da arte no campo de Análises de Dados, bem como abre diversas portas de trabalhos futuros.

Palavras-chave: Análise de Dados, Qualidade de Dados, Limpeza de Dados Incremental, Domínio de Atributos, Índice de Domínio.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

*k*NN	*k*-Nearest Neighbors
AI	Artificial Intelligence
API	Application Programming Interface
BI	Business Intelligence
BI-RADS	Breast Imaging-Reporting and Data System
CBIR	Content-Based Image Retrieval
CFD	Conditional Functional Dependency
DBA	Database Administrator
DBMS	Database Management System
DC	Denial Constraint
DDL	Data Definition Language
DDSM	Digital Database for Screening Mammography
DF	Data Fusion
DICOM	Digital Imaging and Communications in Medicine
ER	Entity Resolution
ERM	Empirical Risk Minimization
FD	Functional Dependency
GBDI	Database and Image Group
GIN	Generalized Inverted Index
GLM	Generalized Linear Model
IC	Integrity Constraint
ICMP	Internet Control Message Protocol
ID	Inclusion Dependency
iKL	individual KL
IP	Internet Protocol
IRMA	*Image Retrieval in Medical Applications*
IT	Information Technology
KL	Kullback-Leibler
LCC	Left CranioCaudal
LMLO	Left MedioLateral Oblique
MAC	Media Access Control

MAM	Metric Access Method
MAP	Maximum A Posteriori
MedFMI-SiR	*Medical user-defined Features, Metrics and Indexes for Similarity Retrieval*
MIAS	Mammographic Image Analysis Society
MIMIC-III	*Medical Information Mart for Intensive Care III*
ML	Machine Learning
NN	Neural Network
OD	Order Dependency
OHDSI	Observational Health Data Sciences and Informatics
OMOP	Observational Medical Outcomes Partnership
PACS	Picture Archiving and Communication System
PET-CT	Positron Emission Tomography and Computed Tomography
PGM	Probabilistic Graphical Model
RCC	Right CranioCaudal
RDBMS	Relational Database Management System
RMLO	Right MedioLateral Oblique
ROI	Region of Interest
RTT	Round-Trip Time
SF	Scale Factor
SGD	Stochastic Gradient Descent
SIMMC	Monitoring Integrated System of the Ministry of Communications
SPIRS	*Spine Pathology & Image Retrieval System*
SQL	Structured Query Language
SVM	Support-Vector Machine
TCP	Transmission Control Protocol
UCC	Unique Column Combination
UDP	User Datagram Protocol
VAE	Variational AutoEncoder
wKL	weighted KL

CONTENTS

CHAPTER

1

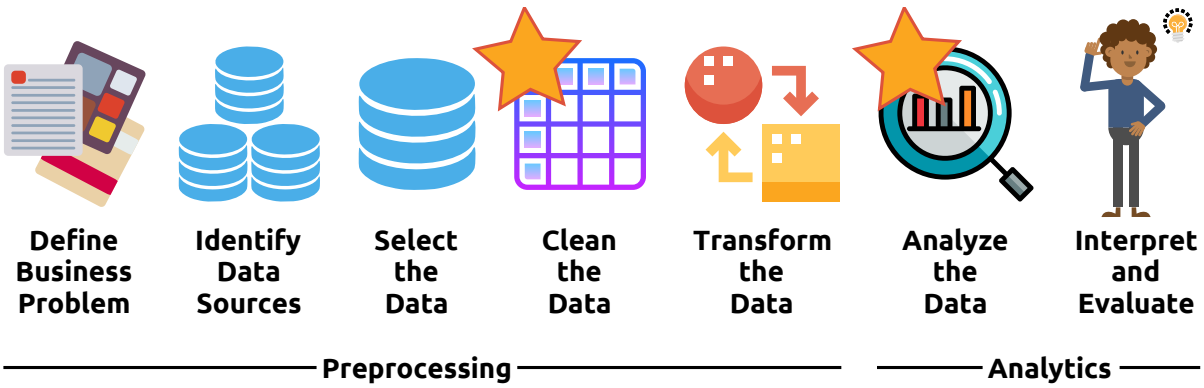
INTRODUCTION

Large data volumes have been generated at unprecedented rates from all sorts of sources (YAQOOB *et al.*, 2016; OUSSOUS *et al.*, 2018), such as healthcare facilities, social networks, governments, and companies in general. In this context, many businesses have become more data-driven, employing prominent technologies and systems such as the Database Management System (DBMS), Big Data technologies for parallel data processing, as well as Business Intelligence (BI) systems to perform Data Analytics and get insights from their data (TRIEU, 2017; VERBEKE; BAESENS; BRAVO, 2017). Nowadays, data ecosystems are vast and rich. Existing technologies are constantly evolving and newer ones are developed to accommodate new needs. Examples of data technologies being employed by enterprises include: **(i)** longstanding systems such as the Relational Database Management System (RDBMS) (MEIER; KAUFMANN, 2019), which has been around since the 70s with the proposal of the Relational Model (CODD, 1970); **(ii)** systems that are not limited to the well-established Structured Query Language (SQL) (MOLINARO; GRAAF, 2020) nor to conventional data types with inherent order relation, e.g. Content-Based Image Retrieval (CBIR) systems, which focus on more complex data types and handle them by similarity (JOHNSON; DOUZE; JÉGOU, 2019); and **(iii)** Artificial Intelligence (AI) systems that employ Machine Learning (ML) techniques, in order to automatically learn patterns from data and make predictions that support decision-making (BURKOV, 2019; RASCHKA; MIRJALILI, 2019; GÉRON, 2019). In such an evolving scenario, there is always room for new technologies and for improvements of the existing ones.

1.1 Motivation

Answering analytical questions based on data is fundamental to make principled decisions, both in research and in the market. This process, usually called *Data Analytics* or *Business Analytics* (VERBEKE; BAESENS; BRAVO, 2017), encompasses several steps in a pipeline, as shown in Figure 1. We describe such steps as follows.

Figure 1 – Data Analytics pipeline (VERBEKE; BAESENS; BRAVO, 2017). The star-shaped markers highlight the steps improved by the contributions of this PhD thesis.



Source: Elaborated by the author.

1. *Identify Business Problem.* A thorough definition of the problem to be addressed is needed. It is important that data scientists collaborate closely with business experts to accurately model the problem. For instance, a business problem could be crop harvest date prediction.

2. *Identify Data Sources.* All source data that can be of interest need to be identified. Initially, the more data, the better. So, the available data are gathered and stored, for instance, in a data lake or a data warehouse.

3. *Select the Data.* Upon close inspection of the gathered data, the data of interest are selected to be analyzed.

4. *Clean the Data.* This step is of major importance to this PhD thesis. In the context of Data Analytics, data cleaning is one of the most crucial tasks. After all, dirty data can be of little use when it comes to getting value from data to support decision-making. This step is also challenging, as data scientists reportedly spend most of their time cleaning and refining data before being able to answer analytical questions (REZIG *et al.*, 2019). Furthermore, according to a survey whose 81% of respondents work for companies which already use AI, data quality issues are among the main bottlenecks holding back further AI adoption within the organization (LORICA; NATHAN, 2019). Two data cleaning problems are particularly of interest to this thesis: **(i)** how to perform data cleaning considering multiple criteria at the same time for error detection, (e.g. missing data and business constraint violations), which is usually referred to as *holistic data cleaning* (REKATSINAS *et al.*, 2017; HEIDARI *et al.*, 2019); and **(ii)** how to perform data cleaning *incrementally*, considering that either the data or the business rules (or both) evolve over time (FAN *et al.*, 2014; GRUENHEID; DONG; SRIVASTAVA, 2014; VOLKOVS *et al.*, 2014; HE *et al.*, 2016; KRISHNAN *et al.*, 2016; SHAABANI; MEINEL, 2017; SHAABANI; MEINEL, 2019).

5. *Transform the Data.* Before analytics can take place, data might need to be transformed. For instance, data scaling might be needed so that certain ML models can be effectively

employed. Another possibility is the extraction of meaningful features from the data.

6. *Analyze the Data.* This is another step of interest to this PhD thesis. Data analysis can take place in numerous ways. We might be interested in posing SQL queries to answer important questions; analyzing a domain of data across distinct collections, such as data that play the same role in different contexts, stored in a data lake; data mining tasks such as clustering, in order to better understand our data; training an ML model so that it can make predictions over unseen data; and the list goes on. In this thesis, a problem of particular interest is the efficient handling of domains of attributes, which refer to sets of attributes playing the same (or similar) roles across multiple relations (or even multiple *collections* in the context of non-relational data). To the best of our knowledge, the work presented in this PhD thesis is the first to tackle such problem.

7. *Interpret and Evaluate.* Finally, stakeholders can interpret the data analysis results. Then, after a thorough evaluation, they can make principled decisions about their business.

This PhD research aimed at answering two main questions: **(i)** *"How can we improve the data cleaning step of Data Analytics, so that we can repair errors considering multiple criteria, as well as allowing it to be done incrementally over evolving databases?"*; and **(ii)** *"In the data analysis step of Data Analytics, how to effectively handle queries on domains of attributes across multiple relations?"*. The first question involved some challenging problems. One of them was to repair data errors holistically, i.e. considering a global view of multiple error detection criteria, in face of an evolving database. Another challenging problem was to efficiently keep the ML model trained as the database evolved, so that the model could predict high-quality repairs based on knowledge acquired from an increasing training set. The second research question, in turn, involved several challenges as well. One of them was to improve performance of queries over domains of attributes, compared to the commonly employed approaches of current RDBMSs. Another challenge was to turn domains of attributes into first-class citizens of RDBMSs, allowing a special category of queries to be performed natively in modern systems.

1.2 Problem Statement

As discussed in the previous section, the goal of this PhD research was to improve the overall Data Analytics pipeline by focusing on two of its steps: one in the preprocessing phase, regarding data cleaning, and another in the analytics phase, regarding data analysis. Accordingly, we elaborated methods targeting these improvements. As a result from this PhD research, we propose the following thesis:

Thesis. *The use of a holistic incremental approach for data cleaning, allied to the support for managing and querying domains of attributes in RDBMSs, improves performance in both the preprocessing and analytics phases of the Data Analytics pipeline.*

We support the stated thesis by proposing methods to solve the four following research problems, one of them targeting data cleaning and the three remaining ones focusing on managing and querying domains of attributes.

Research problem 1. *Performing data cleaning incrementally and holistically.* Driven by the diverse and voluminous information involved in modern analytics, large-scale data cleaning has been the key goal of several academic (BOHANNON *et al.*, 2007; HELLERSTEIN, 2008; ILYAS; CHU, 2015; CHU *et al.*, 2016; CHU; ILYAS, 2016; HEIDARI *et al.*, 2019) and industrial efforts, e.g. Trifacta Wrangler (KANDEL *et al.*, 2011) and Tamr (STONEBRAKER *et al.*, 2013). Existing approaches have been successful at cleaning the entire database at once, considering scenarios where the data to be cleaned are entirely available. Other approaches in the literature target evolving databases, but present limitations as well, e.g. requiring user interactions and dealing with limited error criteria. Based on this scenario, we targeted the problem of performing data cleaning in evolving scenarios, jointly considering multiple error detection criteria, as well as autonomously, i.e. without the need for user interactions.

Research problem 2. *Indexing and querying domains of attributes efficiently.* We refer to domains of attributes as the stored values in a set of attributes, usually across multiple relations. For the sake of simplicity, queries over these domains of attributes are called Domain Queries in this monograph. The approach commonly employed in these queries consists of searching multiple index structures separately and merging the local results into a final response. However, the need for using separate index structures is limiting. In this context, we focused on bypassing this limitation, aiming at improving performance of Domain Queries. More specifically, we targeted the problem of developing a novel category of index structures for Domain Queries, initially focusing on its concept and regardless of the underlying database system.

Research problem 3. *Performing Domain Queries in medical CBIR systems.* Domain Queries can also occur in the medical domain, where imaging data must be retrieved by similarity across multiple data repositories. This is an important use case, as oftentimes physicians need to rely on similar data from past cases to improve diagnoses and any associated decision-making. Moreover, medical data might be scattered across multiple data repositories, due to having been generated by distinct devices. As modern medical devices can generate high-resolution images of richer formats, query performance is a key point when searching data by similarity. In this scenario, we focused on the problem of efficiently indexing and querying domains of medical images, considering the specificities of the content-based retrieval field.

Research problem 4. *Managing domains of attributes in RDBMSs.* In the context of modern RDBMSs, Domain Queries can be executed by searching over a set of columns across multiple tables. Some technologies in current RDBMSs can perform Domain Queries, although with limitations. For instance, techniques such as *inheritance* and *partitioning* are only suitable to scenarios where the accessed tables have a similar schema. Furthermore, employing techniques such as the *indexed materialized view* incurs unnecessary overheads related to data redundancy.

In this context, we addressed the problem of measuring the potential performance of Domain Queries in RDBMSs, employing existing resources in these systems to simulate the execution of a novel solution specifically designed for Domain Queries, before the actual implementation of such a solution can take place.

1.3 Contributions

The main contributions of this PhD research are fourfold, each of them addressing one of the aforementioned research problems.

Contribution 1. *A batchwise probabilistic incremental data cleaning approach.* This is an ML-based framework that performs data cleaning in batches. Different error criteria can be encoded as SQL queries and the errors regarding all criteria are spotted in the database. Then, the framework calculates statistics from the data, which serve as the basis for ML features that are generated later on, and keeps them up-to-date as more data batches arrive. We developed some policies for training the ML models efficiently without having to consider all incoming batches, which allows computational cost savings. With the ML models trained, the framework infers repairs for the cells previously spotted as dirty. Compared to the baseline competitor, our approach enables superior performance while compromising little on repair quality. Furthermore, we advance the state of the art by providing a framework that is suitable to dynamic scenarios and can jointly consider multiple error detection criteria (OLIVEIRA *et al.*, 2020).

Contribution 2. *The Domain Index.* We proposed a category of index structures called Domain Index, whose purpose is to index a domain of attributes within a single index structure. The core idea is that Domain Queries can be executed by accessing a single structure, the Domain Index, which keeps track of the location (for instance, the corresponding table in an RDBMS) of each indexed record. The experiments showed the superior performance of Domain Indexes, both in conventional queries and in content-based retrieval, compared to the commonly employed competitor (OLIVEIRA *et al.*, 2017).

Contribution 3. *Case study of the Domain Index in the medical domain.* We accommodated the proposed Domain Index to the scenario where medical images, which are stored across multiple data repositories, need to be retrieved by similarity. The Domain Index approach was implemented using a well-known access method in the content-based retrieval literature. Extensive experiments showed its superior performance compared to the competitor in all evaluated scenarios, which involved two query types and many feature types extracted from the raw images (OLIVEIRA *et al.*, 2017a; OLIVEIRA *et al.*, 2019).

Contribution 4. *Approach for handling domains of attributes natively in RDBMSs.* This approach enables the creation of a domain of attributes using SQL in RDBMSs, which triggers the construction of a corresponding Domain Index over the specified domain. We also showed how Domain Queries are expressed in SQL and compared the Domain Index to existing techniques

of modern RDBMSs. The experiments, which comprehended both real-world and benchmark data, showcased the superior performance of the Domain Index with respect to several variables, such as query execution time, space requirements, as well as building time.

1.4 Outline

This chapter summarized the motivation, the tackled problems, and the corresponding contributions developed during this PhD research. The rest of this monograph is organized as follows. Chapter 2 contains the relevant background, necessary to follow the work. Chapter 3 presents our incremental data cleaning contribution. Chapter 4 describes the Domain Index and reports the experimental findings. Chapter 5 focuses on how to employ the Domain Index in the medical domain and presents a case study. Chapter 6 describes our approach for handling domains of attributes as first-class citizens in RDBMSs. Finally, Chapter 7 concludes the work, outlining the main contributions of this PhD research, summarizing the resulting publications, and commenting on directions for future research.

CHAPTER

2

BACKGROUND AND RELATED WORK

This chapter presents the background related to statistical concepts, similarity measures, content-based retrieval, data cleaning, as well as indexing techniques of modern RDBMSs, which are required to comprehending the contributions of this thesis.

2.1 Statistical Concepts

In this section, we present basic concepts from the Statistics discipline that were employed in this thesis.

Entropy and Conditional Entropy

We begin by presenting the concept of entropy, which lays the foundation for the concept of conditional entropy. Essentially, entropy is a measure of the *uncertainty* of a random variable (COVER; THOMAS, 2006). Let X be a discrete random variable with alphabet $\mathcal{X}$ and probability mass function $p(x) = \Pr\{X = x\},\ x \in \mathcal{X}$.

Definition. The entropy $H(X)$ of a discrete random variable X is defined by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x). \tag{2.1}$$

We also write $H(p)$ for this quantity, as it is a function of the distribution of X, obtained via the probability mass function $p(x)$. That is, the concept of entropy does not depend on the actual values taken by a random variable, but on their probabilities. The base of the logarithm in Equation 2.1 determines the unit in which entropy is measured (YEUNG, 2008). Specifically, when the base is 2, the unit for entropy is the *bit*, also called *shannon*. When the base is e, the unit for entropy is the *nat*, also known as the *natural unit of information* (due to the usage of the natural logarithm, *ln*). When the base is 10, the unit for entropy is the *dit*, short for *decimal digit*.

A more specific definition is that the entropy $H(X)$ of a random variable X measures the average amount of information in X or, equivalently, the average amount of *uncertainty* removed upon revealing the outcome of X (YEUNG, 2008). We can better understand the behavior of the function $H(X)$ with the following example, which resembles the experiment of a coin flip. For this example, entropy is measured in bits.

Example. Let

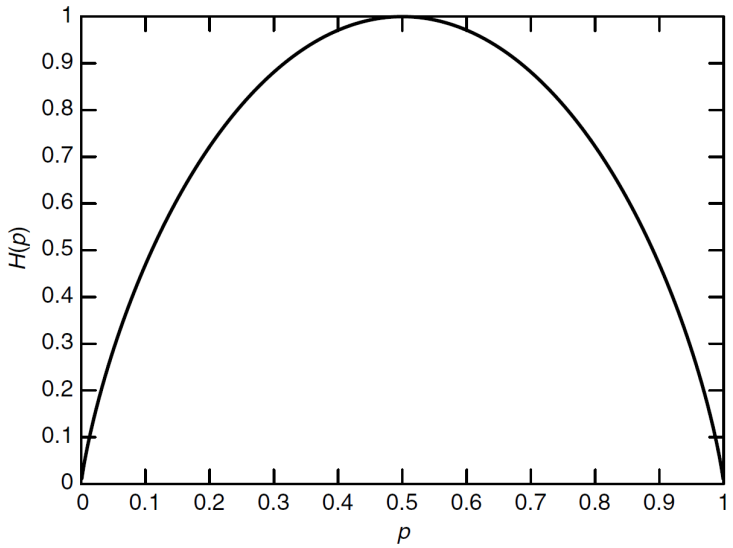
$$X = \begin{cases} 1 & \text{with probability } p, \\ 0 & \text{with probability } 1-p. \end{cases}$$

Then

$$H(X) \stackrel{\text{def}}{=\!=} H(p) = -p\log p - (1-p)\log(1-p)$$

Particularly, $p = 0.5 \rightarrow H(X) = 1$ bit. The graph of the function $H(p)$ is depicted in Figure 2. We can observe some basic properties of entropy: $H(p)$ is a concave function of the distribution and equals 0 when $p = 0$ or 1. This is intuitive, as the variable is not random when $p = 0$ or 1, which means that there is no uncertainty. Conversely, uncertainty is at its peak when $p = 0.5$, which also corresponds to the maximum entropy value.

Figure 2 – Relationship between p and $H(p)$.



Source: Cover and Thomas (2006).

Based on the fundamentals of entropy, we describe the *conditional entropy*, which is employed in the feature generation step of the incremental data cleaning framework proposed in this thesis (Section 3.2). The conditional entropy of a random variable given another one is defined as the expected value of the entropies of the conditional probabilities, averaged over the conditioning random variable (COVER; THOMAS, 2006).

Definition. If $(X,Y) \sim p(x,y)$, the conditional entropy $H(Y|X)$ is defined as

$$H(Y|X) = \sum_{x \in \mathcal{X}} p(x) H(Y|X = x) \tag{2.2}$$

$$= -\sum_{x\in\mathcal{X}} p(x) \sum_{y\in\mathcal{Y}} p(y|x) \log p(y|x) \tag{2.3}$$

$$= -\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y) \log p(y|x) \tag{2.4}$$

$$= -\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)}. \tag{2.5}$$

2.2 Content-Based Data Comparison

This section describes important concepts related to (dis)similarity measures, covering the case of data in metric spaces, as well as data whose (dis)similarity degree is usually measured by asymmetrical distance functions (e.g. probability distributions).

Metric Spaces

For certain data types in the present digital age, exact-match retrieval is neither feasible nor meaningful. This is due to the increasing data diversity in digital collections, which oftentimes store data that lack a precise structure. Consequently, what constitutes a match to a certain request might be different from that implied in more traditional, well-established query types (ZEZULA *et al.*, 2006). In this context, a very useful search paradigm consists of quantifying the *proximity*, *similarity* or *dissimilarity* of a query object compared to objects stored in a database. Essentially, objects that are *near* the query object compose the result set. A commonly used abstraction for nearness is provided by the mathematical concept of *metric space*, in which data objects can be compared based on the *distance* between them (SAMET, 2006; ZEZULA *et al.*, 2006).

Definition. A metric space M, denoted as $M = (S, \delta)$, is defined for a domain of objects S (either in their raw form or their extracted features) and a total (distance) function δ. In order for a metric space to exist, the function $\delta : S \times S \mapsto \mathbb{R}$ must satisfy a set of properties known as the *metric space postulates*:

- $\forall x, y \in \mathbb{S},\ \delta(x,y) \geq 0$ — non-negativity,
- $\forall x, y \in \mathbb{S},\ \delta(x,y) = \delta(y,x)$ — symmetry,
- $\forall x, y \in \mathbb{S},\ x = y \iff \delta(x,y) = 0$ — identity,
- $\forall x, y, z \in \mathbb{S},\ \delta(x,z) \leq \delta(x,y) + \delta(y,z)$ — triangular inequality.

For brevity, such a distance function is commonly referred to simply as *metric*.

The use of a metric space allows searching data with no fixed number of dimensions, that is, non-vector data such as character strings (e.g. natural language words and DNA sequences). In fact, metric spaces are ideal for addressing data variety because they can accommodate any data type, as long as their notion of similarity can be employed by a metric (CHEN *et al.*, 2020).

Distance Functions for Metric Spaces

A well-known family of metrics is formed by the *Minkowski distance functions* (WILSON; MARTINEZ, 1997; ZEZULA *et al.*, 2006). They are also referred to as the L_p metrics, since the individual functions depend on the numeric parameter p. These functions are defined on n-dimensional vectors of real numbers, hence being applicable to the special case of metric spaces where the data are points in a coordinated space. The Minkowski distance functions are defined as follows:

$$L_p((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}, \tag{2.6}$$

where the L_1 metric is known as the *Manhattan distance* (or *City-Block distance*), the L_2 metric denotes the *Euclidean distance*, and $L_\infty = \max_{i=1}^{n} |x_i - y_i|$ is the *maximum distance* (also called *chessboard distance* and *Chebyshev distance*).

The *Jaccard index*, also known as the *Jaccard similarity coefficient*, was first used in Ecology (JACCARD, 1912) and essentially denotes the similarity between two finite sample sets. The idea is to take the intersection size over the union size regarding both sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \tag{2.7}$$

By design, $0 \leq J(A,B) \leq 1$ and, if both sets are empty, $J(A,B) = 1$. A corresponding distance, called *Jaccard distance*, has been derived from this coefficient (LEVANDOWSKY; WINTER, 1971), defined as follows:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}. \tag{2.8}$$

The Jaccard distance has proven to be a metric (KOSUB, 2019), as it satisfies all metric space postulates. This metric can be seen as the basis for similarity measures over sets. In fact, it has given rise to other measures whose essence is the same, i.e. ratio of intersection over union, but applied to other fields and data types. An example is the *Tanimoto similarity* (TANIMOTO, 1958; ROGERS; TANIMOTO, 1960), which is equivalent to the Jaccard index. It is important to note, though, that its counterpart, referred to as the *Tanimoto distance*, is not equivalent to the Jaccard distance, as it does not always satisfy the triangular inequality property. The Tanimoto distance has been proven to work as a metric just under some specific circumstances with vector data, for which the triangular inequality holds (LIPKUS, 1999).

Regardless of being or not a metric, a distance function can be used to build a *distance matrix*, which consists of a square matrix containing the distances between each pair of elements in a set. If the set has N elements, the resulting matrix will have size $N \times N$. Distance matrices are usually employed in the Biology discipline, such as in Phylogeny studies (WEYENBERG; YOSHIDA, 2015), which focus on relationships among different groups of organisms and their

evolutionary development. Graph-theoretic applications benefit from distance matrices as well, whose values correspond to distances between graph vertices (AOUCHICHE; HANSEN, 2014).

There are many other distance functions in the literature, targeting different data types and serving numerous purposes. Another data type of interest to this thesis is the probability distribution. Thus, in the following, we present a fundamental similarity measure that is widely used with probability distributions, which we employ in the Chapter 3 of this thesis.

Kullback-Leibler Divergence

Let p and q be two probability distributions on a common alphabet $\mathcal{X}$. Suppose we are interested in measuring how much these probability distributions are different from each other. Such a measure should be always non-negative and take the zero value if and only if $p = q$. In this context, there is a well-known measure that serves such a purpose, called Kullback-Leibler (KL) divergence (KULLBACK; LEIBLER, 1951; CSISZÁR, 1967; AMARI, 2012), also known as *relative entropy* and *informational divergence* (COVER; THOMAS, 2006; YEUNG, 2008).

Definition. Given two probability distributions p and q on a common alphabet $\mathcal{X}$, the KL divergence from q to p is defined as

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_p \log \frac{p(x)}{q(x)}, \tag{2.9}$$

where E_p denotes the expected value with respect to p.

In the above definition, we use the convention that $0 \log \frac{0}{0} = 0$, as well as the convention that $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$. Therefore, if there is any symbol $x \in \mathcal{X}$ such that $p(x) > 0$ and $q(x) = 0$, then $D(p||q) = \infty$. We note that $D(p||q)$ is not symmetrical in p and q, hence not being a metric. Moreover, $D(\cdot||\cdot)$ does not satisfy the triangular inequality property, also required for a distance function to be a metric. Nevertheless, it is common in the literature to think of the KL divergence as a similarity measure between two probability distributions, in which the larger the measure value, the greater the distance between the distributions (COVER; THOMAS, 2006). It is possible to normalize the KL divergence by employing a different formula, as described by Schumacher, Strohmaier and Lemmerich (2021), bounding values between 0 (same distributions) and 1 (highest distance between distributions).

2.3 Content-Based Retrieval

We employ the term *complex data* when referring to data that are usually not directly compared, e.g. by employing relational operators ($<$, $\leq$, $>$, $\geq$). Instead, a function is employed, which defines the rule for comparing the data. As complex data can be compared in several ways, such a function can be chosen from many existing ones. Examples of complex data comprehend images, audio, time series, DNA sequences, geographical data, as well as large texts. Equality

operators ($=$, $\neq$) are not usually employed either, as they have little or no meaning for complex data. Therefore, complex data objects are better compared by *similarity*, based on their content (FALOUTSOS, 1996).

For comparison purposes, the representation of complex data objects employs *feature vectors*, which consist of features extracted from raw complex data by a given feature extractor. In the context of images, for instance, a widely employed color-based feature extractor is the normalized color histogram extractor. Other examples are texture-based and shape-based feature extractors (DESELAERS; KEYSERS; NEY, 2008).

The comparison of feature vectors occurs by employing distance functions. These functions receive a pair of complex objects, represented as feature vectors, and return a real, positive number that quantifies their dissimilarity. If a distance function meets the requirements to be considered a metric (as described in Section 2.2), the data can be indexed by index structures belonging to a category known as Metric Access Method (MAM) (ZEZULA *et al.*, 2006). The Slim-tree (TRAINA-JR. *et al.*, 2002) is an example of MAM, which we employed in several experiments over complex data presented in this thesis.

The basic types of content-based queries, which we interchangeably refer to as *similarity queries* in this thesis, are the k-Nearest Neighbors (kNN) query and the Range query (TRAINA *et al.*, 2011):

- ***k*NN query.** Represented as kNNq(s_q,k), where s_q is the feature vector of the query element and k is the number of elements to retrieve, it returns the k most similar elements to s_q. This query type is very useful in a clinical environment. Considering a scenario in which a physician needs to provide a diagnosis, he might want to analyze a number k of similar cases to increase their confidence in the decision-making process.
- **Range query.** Represented by Rq(s_q,ξ), where s_q is the feature vector of the query element and ξ is a radius value, it returns all elements within a distance ξ from s_q. For example: "return the mammograms that differ by at most 2 units from the one provided". This query type can be applied to specific scenarios, where a specialist knows a proper value of ξ for the given data domain, feature extractors and distance functions.

Related Work on Medical CBIR Systems

Image analysis by similarity has proved to be a fundamental tool in the medical domain (KUMAR *et al.*, 2013). Diagnostic decision-making has traditionally involved using evidence from a patient's data, such as images from medical exams, coupled with the physician's prior experiences of similar cases (HOLT *et al.*, 2005). It has been seen (SEDGHI; SANDERSON; CLOUGH, 2012) that clinical staff tend to select such similar cases mainly according to visual properties, which highlights the importance of accessing relevant imaging data for more informed and effective treatment.

The need for medical image repositories in a clinical environment has led to the development of the Picture Archiving and Communication System (PACS) (HUANG, 2019). This kind of system enables the storage of medical images in large data repositories, accessed by clinical staff over a network. Then, it is possible for a physician to consider a patient's image history by retrieving the corresponding images from a PACS. Furthermore, large PACS repositories enable opportunities for interpatient image-based investigations (MULLER *et al.*, 2007).

The main limitation of a PACS is that its search capabilities are limited to text-based criteria, e.g. patient name and imaging equipment name. Since this requires dealing with manually assigned labels, in face of the massive volume of imaging data stored in modern clinical environments, the keyword-based image retrieval of a PACS is not viable (KUMAR *et al.*, 2013). An example of this problem is the daily image production at the Radiology Department of the Geneva University Hospitals (MULLER *et al.*, 2009), which was at a rate of about 120,000 daily images in 2009 and whose growth was exponential. There are publicly available medical datasets as well which showcase these massive volumes of data. For instance, the *Medical Information Mart for Intensive Care III* (MIMIC-III) dataset (JOHNSON *et al.*, 2016) is a large, publicly available database of anonymized health-related data associated with over 40,000 patients from 2001 to 2012, consisting of 40 tables of data that amount to about 700,000,000 rows. Another example is the *hospital-x* dataset employed in a study in the literature (DAI *et al.*, 2018), which consists of more than 800,000 diagnosis descriptions from a database at the National University Hospital of Singapore.

In addition to high volumes of data, modern hospitals acquire a diverse range of imaging data (KUMAR *et al.*, 2013). For example, high-resolution devices allow physicians to detect small lesions in exam images, and other devices produce multidimensional images with additional spatial or temporal information. Furthermore, it is common to use different imaging modalities to provide complementary information about a patient. An example of multimodality imaging technique used in clinical environments is the combined Positron Emission Tomography and Computed Tomography (PET-CT), which enables improved cancer treatment compared to the single-modality counterparts (BLODGETT; MELTZER; TOWNSEND, 2007). Motivated by such diversity in medical data, the Observational Medical Outcomes Partnership (OMOP) Common Data Model[1] was proposed by the Observational Health Data Sciences and Informatics (OHDSI) program (HRIPCSAK *et al.*, 2015). The goal for such data model is to enable data standards for healthcare datasets, so that research methods can be systematically applied, ultimately producing comparable and reproducible results.

In this context, image search using a PACS is unfeasible. Due to the massive amount of information encoded by modern medical images, manual annotation is impractical, in addition to being a subjective task. In face of those limitations, medical CBIR systems have gained an increasingly high adoption in modern hospitals. Concepts related to content-based retrieval are

[1] <https://www.ohdsi.org/data-standardization/the-common-data-model/>

at the core of these systems, thus enabling searches such as *k*NN and Range queries.

Few existing CBIR systems have the ability to integrate multiple medical data repositories and query them. An example is the Garlic system (CODY *et al.*, 1995), which is able to integrate repositories of different data types and enable content-based queries over multimedia content. There is a wrapper over each repository that translates queries into the corresponding protocol. When running queries over multiple repositories, each wrapper is responsible for retrieving the elements locally, which are joined by Garlic later on. Although this approach is useful in scenarios with multimedia data, the local results must be merged at a final step, which can slow down the whole retrieving process.

Many of the CBIR systems focus on specific data domains, such as lung (SONG *et al.*, 2010) and vertebra (XUE *et al.*, 2011), and usually target specific query predicates. Examples of these systems comprehend *Image Retrieval in Medical Applications* (IRMA) (LEHMANN *et al.*, 2004), *Spine Pathology & Image Retrieval System* (SPIRS) (HSU *et al.*, 2009), *Medical user-defined Features, Metrics and Indexes for Similarity Retrieval* (MedFMI-SiR) (KASTER *et al.*, 2011), and *MedInject* (CARVALHO *et al.*, 2014). IRMA and MedFMI-SiR are general-purpose systems for executing queries combining content-based and conventional predicates. However, IRMA deals with both predicate types in separate steps, whereas MedFMI-SiR can manage both simultaneously. SPIRS targets only spine images and deals with content-based and conventional predicates separately as well. MedInject is a general-purpose medical framework that focuses on queries with content-based predicates only. Its objective is to provide a base platform over which a developer can build CBIR systems. All of these systems target queries over a single medical data repository, which is a limitation in scenarios where data collections from multiple sources must be queried together.

A current trend in CBIR systems involves the use of ML techniques to enrich the feature extraction process (KUNDU; CHOWDHURY; DAS, 2017; DAS; NEELIMA, 2017; BRESSAN *et al.*, 2018; CHINO *et al.*, 2020; SUBRAMANIAM *et al.*, 2020). For instance, visual features can be generated along with semantic ones, so that the comparison between data objects occurs by aggregating values from distance functions, which focus on the visual features, and from ML techniques such as the Support-Vector Machine (SVM), which compute similarities based on semantic features (MA *et al.*, 2017).

2.4 Data Cleaning

This section covers a wide range of concepts related to data cleaning, such as important constraint types used to encode business rules, as well as the overall process of data cleaning and prominent systems.

Integrity Constraints

The research area of data profiling (ABEDJAN; GOLAB; NAUMANN, 2015) has made numerous advancements over the years, which resulted in many techniques for automatically summarizing datasets and discovering their structural properties. More specifically, some examples of data profiling tasks are: **(i)** statistics collection; **(ii)** metadata management; and **(iii)** Integrity Constraint (IC) discovery. The latter task can be applied to several IC types, each of them targeting certain business rules.

A widely used IC type is the Functional Dependency (FD) (LIU *et al.*, 2012; PAPENBROCK; NAUMANN, 2016), which captures relationships between two sets of attributes in a relation, in which the values in the first determine the values in the second. FDs are usually employed on schema normalization (BLEIFUSS; KRUSE; NAUMANN, 2017). Another IC type is denoted Unique Column Combination (UCC) (HEISE *et al.*, 2013). Essentially, a UCC is a set of attributes that have no duplicate values, hence being a primary key candidate. The Order Dependency (OD) (SZLICHTA *et al.*, 2017; CONSONNI *et al.*, 2019; SZLICHTA *et al.*, 2020) is another example of IC type. When two attributes A and B are constrained by an OD, the same order in values of A occurs in values of B, e.g. if the values of A are increasing, then so are the values of B. Hence, if we were to order the relation based on A, then the attribute B would end up being ordered too. ODs are useful in query optimization, as they allow the automatic rewriting of `ORDER BY` clauses when certain columns in the SQL command make the ordering redundant.

A more comprehensive IC type is the Denial Constraint (DC) (FAN; GEERTS, 2012; CHU; ILYAS; PAPOTTI, 2013; BLEIFUSS; KRUSE; NAUMANN, 2017), which generalizes many other ICs widely employed in databases, such as FDs, UCCs, and ODs. Consequently, DCs can serve as a unified reasoning framework for all of these ICs, expressing business rules that cannot be expressed by the more restrictive IC types. Essentially, a DC defines a set of predicates on a relation that should not occur in a relational database. Hence, a relational database *satisfies* that DC if, for any n distinct tuples, at least one predicate is not met. Conversely, a relational database *violates* that DC if all predicates are met. Given two tuples t and t', we define DCs as follows (BLEIFUSS; KRUSE; NAUMANN, 2017).

Definition. Let r be a relational database with schema $R(A_1,\ldots,A_n)$. A Denial Constraint ψ is a statement of the form

$$\forall t,t' : \neg\left(t^1[A^1]\ \phi_1\ u^1[B^1]\ \wedge\ \cdots\ \wedge\ t^k[A^k]\ \phi_k\ u^k[B^k]\right) \tag{2.10}$$

with $t^i,u^i \in \{t,t'\}, A^i,B^i \in \{A_1,\ldots,A_n\}$, and $\phi_i \in \{=,\neq,<,\leq,>,\geq\}$ The different clauses in the conjunction are referred to as *predicates*. The database r satisfies ψ if and only if ψ is satisfied for any two distinct tuples $t,t' \in r$.

In addition to ICs that target scalar data, there are ICs that consider multidimensional data and even data in metric spaces, such as the metric FDs (KOUDAS *et al.*, 2009). They are useful when minor variations occur in the data, such as changes in format, which normally would

trigger FD violations, but actually do not correspond to actual semantic violations. In this context, metric FDs allow small differences in the data, according to the criteria of a metric. Metric FDs have been evaluated and proven useful on various datasets lying in multidimensional spaces.

Data Cleaning Pipeline

We describe the steps that compose a full execution of a data cleaning system. There might be variations depending on the system, such as performing multiple iterations of the whole process, executing just part of the process, as well as including user validation at the end of each iteration. However, we focus on the basic steps that usually underlie data cleaning approaches. Throughout this thesis, a **cell** is referred to as **a single piece of data** in the dataset being cleaned. In the context of relational data, in which a relation is implemented as a table, a **cell** contains the value of an attribute (table column) for an entity (table row).

Step 1: Data Loading. The data cleaning system loads the raw dataset, which is kept in memory to be further processed and may be stored in an underlying Database Management System (DBMS) as well.

Step 2: Error Detection. The system runs error detectors to identify potentially dirty cells, which have values in the dataset spotted as erroneous based on the criteria employed by the detectors. An example of a straightforward error detector is a Null Detector, which looks for empty cells and marks them as errors. Another example is a Constraint Violation Detector, which identifies cells that violate a given set of constraints, such as FDs and DCs.

Step 3: Error Repairing. This step seeks to fix the newly-found dirty cells in a principled way. One of the existing operational principles for performing repairs is the *principle of minimality*, which consists of updating the minimum number of cells needed to bring the dataset to a clean state, i.e. free of all the spotted errors such as typos and constraint violations (KOLAHI; LAKSHMANAN, 2009). However, minimal repairs not necessarily lead to correct results, as the updated cells might still contain wrong values even if they are no longer spotted by the error detectors being used (ILYAS; CHU, 2019). Another operational principle aims for the *most probable value*. For instance, if a dirty cell belongs to a categorical attribute, this principle can be employed in a multi-class classification problem, in which the inferred correct value is the most probable value from the domain of such cell. Conversely, if a dirty cell belongs to a numerical attribute, this principle can be employed in a regression problem, in which the system learns a model (from the dataset itself and possibly external data) to estimate the correct value.

Holistic Data Cleaning

Significant efforts have been made to automate data cleaning tasks, and several surveys summarize these results (FAN; GEERTS, 2012; ILYAS; CHU, 2015). Regarding error detection, methods usually rely on IC violation (BOHANNON *et al.*, 2007; CHU; ILYAS; PAPOTTI, 2013)

or duplicate (KOUDAS; SARAWAGI; SRIVASTAVA, 2006; NAUMANN; HERSCHEL, 2010; GETOOR; MACHANAVAJJHALA, 2012) and outlier (DAS; SCHNEIDER, 2007; HELLERSTEIN, 2008) detection to identify errors. For data repairing, different input information can be considered to determine the criteria for repairs. In this context, input information of a specific type corresponds to a **signal**. In the literature, existing methods usually rely on one of several signals: (i) ICs (BESKALES *et al.*, 2013; PAPOTTI; CHU; ILYAS, 2013; GEERTS *et al.*, 2019); (ii) external information (FAN *et al.*, 2009; CHU *et al.*, 2015; HEIDARI *et al.*, 2020), such as dictionaries and knowledge bases; or (iii) quantitative statistics (MAYFIELD; NEVILLE; PRABHAKAR, 2010; YAKOUT; BERTI-ÉQUILLE; ELMAGARMID, 2013; WU *et al.*, 2020; CAPPUZZO; PAPOTTI; THIRUMURUGANATHAN, 2020).

Figure 3 – Motivation for performing data cleaning holistically.

(A) Input database – Chicago food inspections

	DBAName	AKAName	Address	City	State	Zip
t1	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	60608
t2	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	60609
t3	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	60609
t4	Johnnyo's	Johnnyo's	3465 S Morgan ST	Cícago	IL	60608

Conflicts between "60608" and "60609" due to c2

Does not obey data distribution

Conflict between "Chicago" and "Cícago" due to c2

(B) Functional Dependencies

c1: DBAName → Zip
c2: Zip → City, State
c3: City, State, Address → Zip

(C) Matching Dependencies

m1: Zip = Ext_Zip → City = Ext_City
m2: Zip = Ext_Zip → State = Ext_State
m3: (City = Ext_City) ^ (State = Ext_State) ^ (Address = Ext_Address) → Zip = Ext_Zip

(D) External information – Address listings

Ext_Address	Ext_City	Ext_State	Ext_Zip
3465 S Morgan ST	Chicago	IL	60608
1208 N Wells ST	Chicago	IL	60610
259 E Erie ST	Chicago	IL	60611
2806 W Cermak Rd	Chicago	IL	60623

(E) Repair using minimality regarding Functional Dependencies

	DBAName	AKAName	Address	City	State	Zip
t1	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60609**
t2	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60609**
t3	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60609**
t4	**Johnnyo's**	Johnnyo's	3465 S Morgan ST	**Cícago**	IL	60608

(F) Repair using Matching Dependencies

	DBAName	AKAName	Address	City	State	Zip
t1	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	60608
t2	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60608**
t3	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60608**
t4	**Johnnyo's**	Johnnyo's	3465 S Morgan ST	**Chicago**	IL	60608

(G) Repair that leverages Quantitative Statistics

	DBAName	AKAName	Address	City	State	Zip
t1	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	60608
t2	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60609**
t3	John Veliotis Sr.	Johnnyo's	3465 S Morgan ST	Chicago	IL	**60609**
t4	**John Veliotis Sr.**	Johnnyo's	3465 S Morgan ST	**Chicago**	IL	60608

Source: Adapted from Rekatsinas *et al.* (2017).

Methods that rely on ICs assume the majority of input data to be clean and perform repairs

based on the *principle of minimality* (ARENAS; BERTOSSI; CHOMICKI, 1999; CHOMICKI; MARCINKOWSKI, 2005; AFRATI; KOLAITIS, 2009).

Definition. Given an inconsistent database instance r and a consistent database instance r', the principle of minimality states that their symmetric difference $r \oplus r'$ is minimal with respect to set inclusion.

Essentially, the goal is to perform the minimal number of updates in the input dataset so that the associated ICs are no longer violated, leading the dataset to a consistent state. That is, given two candidate sets of repairs, the principle of minimality determines that the one with fewer changes with respect to the original data is preferable. However, minimal repairs do not necessarily lead to correct repairs. An example is shown in Figure 3(E), where the `Zip` code of tuple `t1` is updated so that all functional dependencies in Figure 3(B) are satisfied. This particular repair introduces an error, since the updated `Zip` code is wrong. Moreover, other errors remain (displayed in red) because altering such wrong cells would lead to a non-minimal repair.

To detect and repair errors in the input dataset, methods based on external data match records in the input dataset to records in external dictionaries or knowledge bases. The matching process is usually described by a collection of matching dependencies between the input dataset and the external information (Figure 3(C)). A repair using such methods is shown in Figure 3(F). This repair fixes most errors, but fails to repair the `DBAName` field of tuple `t4`, as no information for this field exists in the external data.

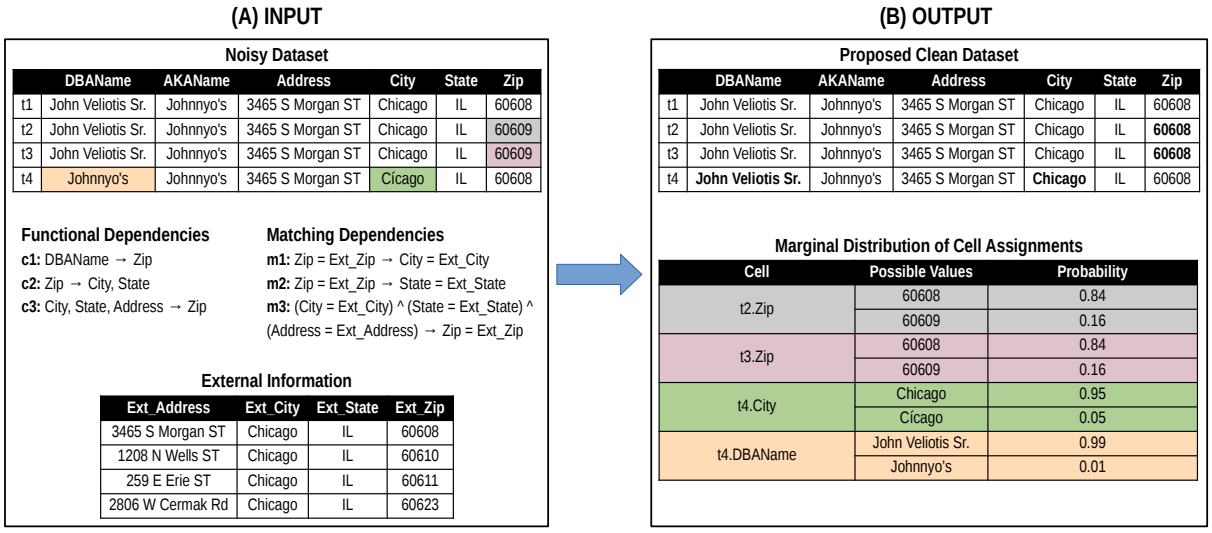
Finally, data repairing methods based on statistical analyses leverage quantitative statistics of the input dataset, such as co-occurrences of attribute values, and use those for cleaning. Such a repair is depicted in Figure 3(G), where the `DBAName` and `City` fields of tuple `t4` are updated, since their original values are outliers with respect to other tuples in the dataset. However, this repair does not have enough information to fix the `Zip` code of tuples `t2` and `t3`.

The HoloClean System

HoloClean is a state-of-the-art data cleaning system (ROH; HEO; WHANG, 2018) that combines multiple signals, such as quantitative statistics, quality rules, and reference data, to predict data repairs. The input and output data of the system are shown in Figure 4. In the figure, the four colored cells in the input table are erroneous cells, which were identified in the Error Detection step based on three sets of signals: *Functional Dependencies*, *Matching Dependencies*, and *External Information*. Then, in the output, we can see a *Proposed Clean Dataset*, whose cells in bold contain the new values assigned by the system. Specifically, for each erroneous cell, HoloClean picks the value whose marginal probability is the highest, as shown in the *Marginal Distribution of Cell Assignments* table.

The first published version of HoloClean (REKATSINAS *et al.*, 2017) builds a Probabilistic Graphical Model (PGM) (KOLLER; FRIEDMAN, 2009) whose random variables correspond

Figure 4 – Overview of the input and output data in HoloClean.



Source: Adapted from Rekatsinas *et al.* (2017).

to the cells in the input dataset. Specifically, given a dataset D, the system associates each cell $c \in D$ with a random variable T_c that takes values from a domain $dom(c)$ and compiles a PGM that describes the distribution of random variables T_c. Any set of error detectors can be used to identify erroneous cells in D, e.g. the aforementioned Null Detector and Constraint Violation Detector. Running error detectors splits the random variables T_c into sets T_c^e and T_c^c, where T_c^e is the set of random variables referring to erroneous cells and T_c^c is the set of random variables referring to clean cells. The published version of HoloClean (REKATSINAS *et al.*, 2017) uses DeepDive[2] (SHIN *et al.*, 2015) to declare the random variables and encode various signals (such as the ones shown in Figure 4) in the PGM.

Having the PGM created, prior to learning the model parameters, the system employs a method called *weak labelling* to enlarge the training set. Initially, only the set of cells identified as clean comprises the training set. Then, HoloClean employs an estimator such as Naive Bayes to predict the correct value of each dirty cell. If the estimator predicts a value with confidence greater than a given threshold, the predicted value is used as the correct label for that cell, adding it to the training set which originally had only examples from clean cells. Essentially, this is a self-training strategy, which falls under the semi-supervised learning umbrella (GRANDVALET; BENGIO, 2004; CHAPELLE; SCHLKOPF; ZIEN, 2010). After the weak-labelling step, HoloClean uses Empirical Risk Minimization (ERM) (SHALEV-SHWARTZ; BEN-DAVID, 2014) to estimate the model parameters. Finally, the system predicts the correct value of each cell in T_c^e by performing approximate inference via Gibbs sampling (ZHANG; RÉ, 2014), assigning those cells to the corresponding Maximum A Posteriori (MAP) estimates of variables T_c^e.

[2] <http://deepdive.stanford.edu/>

Following the published version of HoloClean, a new implementation[3] whose model is a Neural Network (NN) replaced the previous one based on PGMs. The new implementation, which was the basis for the data cleaning contributions presented in this thesis, is functionally similar to the PGM-based one. We describe the NN-based implementation in Chapter 3, when we highlight the modifications in the system that led to our contributions.

Related Work on Incremental Data Cleaning

We can analyze incremental data cleaning approaches along two dimensions: **(i)** the first corresponds to whether the dirty dataset is handled entirely or in batches, and **(ii)** the second corresponds to whether the cleaning process data occurs once or iteratively. This analysis leads to four main categories of data cleaning approaches, as shown in Table 1.

Table 1 – Data cleaning categories.

	Full dataset	Dataset in batches
One iteration	A	B
Multiple iterations	C	D

Source: Elaborated by the author.

Categories B and D, which correspond to handling datasets in batches, apply to evolving datasets and business rules, as well as when computational resources are limited. Categories C and D perform iterative cleaning, focusing on improving output quality over time, sometimes based on user feedback. Most data cleaning approaches are in category A, hence not being fit to incremental scenarios. There is a closely related work on continuous data cleaning (VOLKOVS *et al.*, 2014) in category C, and another one on online data cleaning (REZIG, 2018) in category B. Compared to approaches that work incrementally, data cleaning approaches in category A tend to be more efficient with respect to both repair quality and processing time, but are limited to scenarios in which the entire dataset is available and computational resources are enough. Our contribution described in Chapter 3 is in category B, but it also aims at performing as efficiently as possible compared to approaches in category A.

We further describe the related work with respect to the set of data cleaning tasks targeted.

Incremental constraint discovery and error detection. A new type of constraint called Conditional Functional Dependency (CFD) was proposed and extensively studied in data cleaning applications, along with incremental methods for checking CFDs in response to changes in the data (FAN *et al.*, 2008). Essentially, in contrast to traditional FDs that were developed mainly for schema design, CFDs seek to capture data consistency by incorporating bindings of semantically related values. For instance, if an attribute `country_code` equals 44 (United Kingdom), then `Zip` code determines `Street`. It can be seen as an FD *conditioned* on a value bound to an attribute.

[3] <https://github.com/HoloClean>

Such proposal was later extended to also deal with distributed data (FAN *et al.*, 2014). Another line of work deals with the problem of keeping Entity Resolution (ER) results up-to-date when the ER logic evolves frequently (WHANG; GARCIA-MOLINA, 2010), as well as when data updates occur (GRUENHEID; DONG; SRIVASTAVA, 2014). Another work proposed the Swan system, which targets UCC detection on dynamic data (ABEDJAN; QUIANÉ-RUIZ; NAUMANN, 2014), and other works target Inclusion Dependency (ID) discovery and maintenance (SHAABANI; MEINEL, 2017; SHAABANI; MEINEL, 2019) on evolving data. An approach for automatic large-scale data quality verification was developed (SCHELTER *et al.*, 2018b), which provides an Application Programming Interface (API) for combining quality constraints with user-defined validation code, enabling "unit tests" for data. Finally, DynFD is an algorithm for FD discovery over dynamic data (SCHIRMER *et al.*, 2019).

Incremental error repair. Continuous data cleaning (VOLKOVS *et al.*, 2014) seeks to repair FDs and data in a dynamic environment, with evolving business rules and data. Initially, a probabilistic classifier that infers the repair type (data, FD or both) is first trained using user-provided training data, i.e. repairs selected and validated by the user. As data and constraints change and violations arise, a set of statistics is maintained, e.g. the number of violating tuples. Then, based on those statistics, the trained classifier infers the repair types for addressing the violations. Specifically, it computes the probability of each repair type and generates predictions, which are passed to a repair algorithm that narrows down the search space of repairs. The repair algorithm recommends repairs to the user, who will decide which ones to apply. The applied repairs are used to retrain the classifier and the whole process is repeated.

ActiveClean (KRISHNAN *et al.*, 2016) is a model-specific approach which cleans data incrementally with the goal of improving ML results, targeting certain models trained based on gradient descent methods. The insight of ActiveClean is that those models, e.g. logistic regression, can perform training and cleaning simultaneously, based on the Mini-Batch Stochastic Gradient Descent (SGD) algorithm for learning model parameters. ActiveClean modifies the Mini-Batch SGD algorithm by computing the average gradient from cleaned data in the batch, rather than from the originally dirty data. ActiveClean runs the Mini-Batch SGD algorithm iteratively and is guaranteed to converge, since every record has a non-zero probability of being sampled to be cleaned in a batch. At the end of the process, the model accuracy is improved as a result of such convergence. While ActiveClean employs data cleaning to obtain a better model, our goal in this thesis is to improve the data cleaning process itself in dynamic environments.

Another work proposed the SampleClean framework (WANG *et al.*, 2014) for efficient and accurate query processing on dirty data. SampleClean only requires users to clean a sample of data, which the system uses when processing aggregate queries. The Falcon system (HE *et al.*, 2016) is an interactive, deterministic, and declarative data cleaning system, which uses SQL update queries to repair data. Bootstrapped by one user update, Falcon guesses a set of possible SQL update queries for repairing the data, aiming at finding a set of queries that is minimal in

size and at the same time fixes the largest number of errors in the data.

A thesis on online data cleaning (REZIG, 2018) presents two contributions. The first one consists of a framework for performing online ER and Data Fusion (DF) on Web databases. The framework processes queries issued to Web databases, then deduplicates and fuses query results, keeping them in a cache for future reference. The cache is iteratively updated with new query results, which allows performing ER and DF efficiently, as the existing data items are jointly cleaned with incoming query results. The second contribution targets IC violations, proposing a novel way to perform FD repairs that outperforms existing approaches.

PIClean (YU; CHU, 2019) is a probabilistic and interactive data cleaning system, which discovers column relationships in a dataset by means of a minimization problem called low-rank approximation. Then, based on those relationships, the system detects errors and produces repairs for the user to validate. PIClean constantly incorporates user feedbacks to improve cleaning.

2.5 Alternative Indexing Techniques in Modern RDBMSs

The indexing techniques described in this section are "alternative" because they solve specific problems, which may not be recurrent in day-to-day workloads. They are worth mentioning in order to highlight the fact that, until the proposal of the Domain Index in this thesis, the problem of searching Attribute Domains remained untouched. That is, none of these techniques are suitable to answer Domain Queries efficiently. Nonetheless, it is important to describe them to specify their differences from our contributions presented in Chapter 4.

The *Clustered Index* from Oracle Database (URBANO, 2016) is an indexing technique for clustered tables, which are two tables whose schemas have common columns that are often queried together. Rows of clustered tables with identical values for these columns are physically stored in the same data blocks. Therefore, queries over clustered tables (usually `JOIN` queries) have reduced I/O costs when accessing such common columns. After creating clustered tables and before inserting a new record, it is required to specify a *Clustered Index* over the common columns to physically guide the data organization. However, despite the benefits of clustered tables and *Clustered Indexes*, their two major problems are: **(i)** high costs of insertion and update operations; and **(ii)** when a query accesses only one of the clustered tables, it will have a worse performance, since the data blocks also contain columns from the other table.

Other approaches are the *Covering Index* and the *Index with Included Columns*, which physically embed additional columns in their data structures. Essentially, their goal is to allow queries to use only the information within the index to build the response, i.e. without having to fetch additional data blocks that store the records themselves. These indexes are available in the SQL Server RDBMS. While the *Covering Index* organizes the additional columns as part of its keys, i.e. using them for sorting the nodes in its structure, the *Index with Included Columns* does not do so for the additional columns. The *Index with Included Columns* must be employed in

queries whose predicates contain attributes in its key and, optionally, can contain the additional columns. Conversely, the use of a *Covering Index* depends on the order of all indexed columns in the predicate (LAHDENMAKI; LEACH, 2005; STRATE; FRITCHEY, 2015).

Another related approach is the use of a *Global Index* (KYTE; KUHN, 2014) defined on partitioned tables. Essentially, a *Global Index* forms a one-to-many relationship between an index partition and many table partitions. The Oracle RDBMS documentation specifies that a *Global Index* can be partitioned by range or hash methods, as well as can be defined on any type of partitioned/non-partitioned table. In the context of searching columns across multiple tables, a *Global Index* has the advantage of operating as a single access method over multiple partitions, whereas local indexes are each defined on one partition and queried separately. Local indexes are better suited for scenarios that require complex maintenance operations over the partitions. The problem of using partitioned tables in this context is that it imposes the constraint of having the same schema in all partitions, differently from the proposed Domain Index (Chapter 4).

CHAPTER

3

PROBABILISTIC INCREMENTAL DATA CLEANING

Enterprises have been collecting very large amounts of data from a variety of sources to power their applications, seeking richer and better-informed analytics (ILYAS, 2016). Data acquisition often introduces errors in data, such as missing values, typos, replicated entries for the same real-world entity, and violations of data quality constraints. Therefore, there is a need for creating more effective and efficient data cleaning solutions, which, not surprisingly, is rife with theoretical and engineering problems.

As described in Section 2.4, data cleaning consists of two main phases: **(i)** Error Detection, where data errors and constraint violations are identified and possibly validated by experts; and **(ii)** Error Repairing, where updates are made (or suggested) to bring the data to a cleaner state (ILYAS; CHU, 2019). New records may arrive constantly, and data quality constraints might also evolve over time. As new data are continuously collected and included to the existing and previously cleaned data, the process of cleaning each new snapshot of data is challenging and time-consuming, thus it is impractical to re-execute the whole data cleaning pipeline every time that data or constraints change. Therefore, performing data cleaning incrementally is invaluable in face of constantly changing data or constraints (ILYAS; CHU, 2015). Incremental data cleaning is even more important when the data do not fit into main memory, which tends to be the case for data profiling tasks (ABEDJAN; GOLAB; NAUMANN, 2015) such as constraint discovery and error detection.

With the increasing availability of resources for building large-scale Machine Learning solutions, employing ML techniques for data cleaning has become a promising direction. For instance, some works have been modelling the task of repairing categorical data as a multi-class classification problem (VOLKOVS *et al.*, 2014; REKATSINAS *et al.*, 2017). Employing ML models allows learning from data based on a **probabilistic** reasoning (i.e. using well-known tools of probability theory) (MURPHY, 2012), as well as enables a *holistic* treatment over a variety of

data errors, such as typos and constraint violations, rather than tackling each problem in isolation (ILYAS; CHU, 2019). Multiple types of features can be combined to feed ML models, such as features based on quantitative statistics (BESKALES *et al.*, 2013; PAPOTTI; CHU; ILYAS, 2013) and integrity constraints (MAYFIELD; NEVILLE; PRABHAKAR, 2010; YAKOUT; BERTI-ÉQUILLE; ELMAGARMID, 2013). The approach of treating various error types and data features holistically is a recent trend (PAPOTTI; CHU; ILYAS, 2013; ABEDJAN *et al.*, 2016; REKATSINAS *et al.*, 2017; HEIDARI *et al.*, 2019) in the data cleaning literature.

Some state-of-the-art techniques can perform data cleaning in a holistic fashion, but not incrementally (PAPOTTI; CHU; ILYAS, 2013; REKATSINAS *et al.*, 2017; HEIDARI *et al.*, 2019). Conversely, current incremental data cleaning approaches either **(i)** present limitations, such as dealing with a single error type and/or depending on user interactions (VOLKOVS *et al.*, 2014), or **(ii)** perform data cleaning targeting a better ML model just for the task and dataset at hand, rather than focusing on the cleaned dataset itself (KRISHNAN *et al.*, 2016). Hence, the literature lacks an ML-based data cleaning solution that is both holistic and incremental.

In ML-based data cleaning solutions, large evolving datasets are continuously repaired as new parts of the dataset arrive — throughout this chapter, we refer to such a new part of the dataset as a **batch** or **data batch** (not to be confused with data streaming). The ML model is eventually updated according to data changes. This series of tasks calls for an end-to-end ML management pipeline, which is a line of work that has been drawing increasing attention. Major players such as Google and Amazon have been working to offer comprehensive frameworks for deploying ML systems (BAYLOR *et al.*, 2017; SCHELTER *et al.*, 2018a). Such frameworks aim at supporting all the life cycle of an ML application, including different aspects of model management. These frameworks represent an essential first step towards managing ML applications, but they are general-purpose. To perform data cleaning incrementally using an end-to-end pipeline, a new solution is needed.

Technical Challenges

In an incremental scenario for holistic data cleaning, the following challenges arise.

- **How to select data cells to inspect for errors.** When looking for constraint violations, we are interested in identifying patterns that should not occur, analyzing cells from distinct tuples. It is possible that certain cells within existing data do not violate any constraint in the beginning, but reveal themselves as erroneous as new tuples arrive, and new patterns show up. In this context, we come across two possible measures: **(i)** take into account the whole data seen so far when spotting errors to be repaired, which can be increasingly costly as new batches arrive, or **(ii)** assume that few errors (if any) will occur within the existing data when more tuples arrive, thus looking for errors only across the incoming tuples, which is expected to be less costly than inspecting the whole data.

- **How to incrementally generate features.** Features are measurable properties of a phenomenon being observed. Considering an evolving dataset, it is natural to expect that the generated features change over time. Instead of computing features from scratch whenever a data batch arrives, an incremental data cleaning system should be able to generate features incrementally, which otherwise can be rather time-consuming. For instance, if the features are based on quantitative statistics, they should be updated whenever a new data batch arrives, preferably without recomputing the statistics from the very beginning.

- **Whether to retrain ML models as new batches arrive.** The ability to detect significant changes in the incoming data allows model training optimizations. For example, if an incoming data batch follows the same distribution of the training data, it is expected that the model will not improve significantly when retrained with examples from the new batch. Therefore, a repairing process should be able to automatically trigger the retraining of the ML models only when needed, which calls for proper metrics to guide the process.

Contributions

This chapter presents an end-to-end holistic incremental data cleaning framework. To the best of our knowledge, this is the first approach to perform incremental data cleaning holistically (combining multiple error detection criteria) and independently of user interactions. Specifically, our contributions are as follows:

1. We extend the framework underlying HoloClean[1], a state-of-the-art holistic data cleaning system (REKATSINAS *et al.*, 2017; ROH; HEO; WHANG, 2018), enhancing its existing modules and including a new one in order to enable incremental cleaning (Section 3.1).

2. Based on the existing implementation of HoloClean, we instantiate the proposed framework by developing a system for incrementally cleaning structured categorical data. Initially, we describe our system from the raw data loading to the feature generation steps. Specifically, we detail our approach to **(i)** identify errors, **(ii)** compute dataset statistics, and **(iii)** use error information and statistics to generate the data features that feed our ML models for training. Furthermore, we present our strategy to skip model retraining when a new batch does not present significant changes in data distribution. It allows determining that a model retraining can be skipped as early as before computing dataset statistics, which avoids generating features needlessly and enables inferring data repairs right away (Section 3.2).

3. We further describe the system focusing on the steps after feature generation. We detail our approach for setting up the ML models, which differs from the one used by HoloClean. Distinctly from the one-model setup in the original version of HoloClean, we employ one

[1] <http://www.holoclean.io/>

model per attribute in the dataset, which allows reducing memory requirements and avoids unnecessary model retraining more often, since the models are separate (Section 3.3).

4. We extensively evaluate our approach against competitors over publicly available categorical datasets with varying cardinalities, measuring repair quality based on how clean is the resulting dataset, as well as execution time and memory consumption (Section 3.4).

Problem Statement

Let D be a relation with the schema $(A_1, \ldots, A_N)$, where A_i is an attribute and N is the dimensionality of D. Furthermore, let $D\{t_1, \ldots, t_M\}$ be the set of tuples in D. For every tuple $t_j \in D$, each attribute stores a single value, and M is the cardinality of D. We refer to an attribute value in a tuple $t[A_i]$ as a cell. A cell c is dirty, i.e. it is an error, if its observed value v_c is different from its unknown true value v_c^*. All dirty cells in the relation D correspond to the erroneous subset D^E, whereas the clean subset $D^C = D \setminus D^E$ comprises the clean cells in D. A cell repair is an attribution of a value $\hat{v}_c$ to the cell, where $\hat{v}_c \neq v_c$, and the repair is correct if $\hat{v}_c = v_c^*$.

Suppose D is an evolving dataset with batches $D_1, D_2, \ldots, D_B$, such that $D = \bigcup_{k=1}^{B} D_k$, and at a certain point in time k only the batches $D_{1..k}$ are known. The cardinality $|D_k|$ of each batch can be distinct. The dirty cells in a batch D_k correspond to the erroneous subset D_k^E of that batch, and the clean subset $D_k^C = D_k \setminus D_k^E$ comprises the clean cells of such batch.

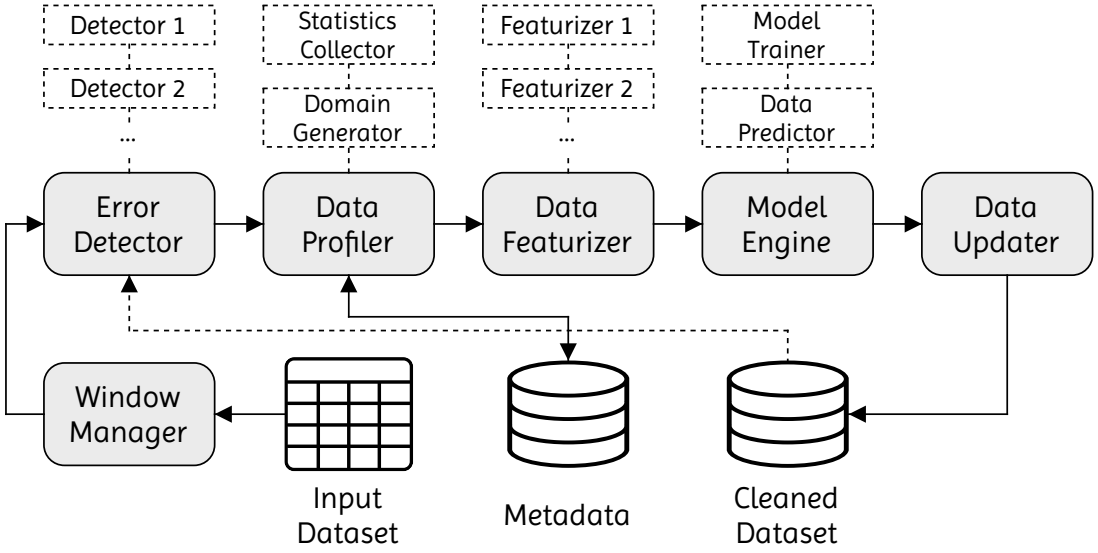
Finally, let $\Lambda = (\mathcal{C}_e, \mathcal{C}_f, \mathcal{C}_t)$ be an incremental data cleaning process, where $\mathcal{C}_e$ is the set of criteria for error detection, $\mathcal{C}_f$ is the set of criteria for feature vector generation, and $\mathcal{C}_t$ is the set of criteria for ML model training. Moreover, let $\hat{D}_{1..k}^E \subseteq D_{1..k}^E$ be the set of dirty cells identified according to $\mathcal{C}_e$. Given Λ and the set of batches $D_{1..k}$ of an evolving dataset D without ground-truth readily available, *the problem addressed in this chapter is to infer a repair for each dirty cell $c \in \hat{D}_{1..k}^E$, employing features generated based on $\mathcal{C}_f$ and models trained based on $\mathcal{C}_t$.*

3.1 Framework Overview

We describe the proposed framework following the architecture depicted in Figure 5, which extends the architecture on which the HoloClean system is based. The *Window Manager* is the only brand new module developed in this work, as the remaining modules are already part of the HoloClean system, but which have also been enhanced in order to work incrementally.

The *Window Manager* is responsible for handling the incoming *Input Dataset*, generating the data windows to be processed. Data windows can be defined according to memory resources or time limits, and the number of elements within a data window can vary over time. The window size affects **(i)** how much new cleaned data is produced by the data cleaning process, as well as **(ii)** the extent to which the existing database can be enriched upon receiving new cleaned data, which can improve ML model training.

Figure 5 – Architecture of the proposed framework.



Source: Elaborated by the author.

The *Error Detector* module receives each data window from the *Window Manager* and looks for potentially dirty cells. It applies one or more detectors, such as for identifying data cells whose values are missing or that violate integrity constraints. Errors can be detected either solely within the data window itself or involving cells both in the data window and in the clean subset (if any) of the dataset. The parts of the dataset to be inspected for errors are determined by the criteria $\mathcal{C}_e$ in Λ.

The *Data Profiler* first employs the Statistics Collector to gather statistics from the data. The statistics can account either for all the dataset seen so far or for a subset of it, as determined by the set of criteria $\mathcal{C}_f$. The *Data Profiler* executes the Domain Generator to compute $dom(c)$ for every cell c to be used either in model training or in data repairing, where $dom(c)$ consists of all values that can be used as evidence for repairing cell c, according to the collected statistics. The set of criteria $\mathcal{C}_f$ is also responsible for defining what statistics should be computed. The statistics are stored as part of the *Metadata* in an underlying DBMS and are incrementally updated as the dataset evolves.

The *Data Featurizer* module relies on one or more Featurizers to generate features from the cells to be used in model training and data repairing. Examples of features include attribute-value frequencies, co-occurrences of attribute-value pairs, and constraint violations. When multiple Featurizers are used, the *Data Featurizer* module concatenates the features into a composite feature vector for each cell, enabling the aforementioned holistic treatment of multiple signals in model training.

The *Model Manager* module consists of a Model Trainer and a Data Repairer. It can train one or more models, and the criteria $\mathcal{C}_t$ determine the model settings, such as the number of models and what portion of the training data each model is assigned to. After model training,

repairs are inferred by the Data Repairer. The Data Repairer forwards its inferred repairs to the *Data Updater* module, which persists the repaired data in the underlying DBMS by updating the Cleaned Dataset. When the next batch arrives, the Cleaned Dataset will be available to be examined (which can or cannot occur depending on the sets of criteria $\mathcal{C}_e$ and $\mathcal{C}_f$) by the *Error Detector* and *Data Profiler* modules.

3.2 Incremental Features

Our proposal performs incremental data cleaning holistically, which means that multiple signals can be jointly taken into account to infer data repairs, allowing us to generate distinct types of data features. We carried out a preliminary evaluation to support our proposal, analyzing the effect of several features on repair quality. The evaluation was in a non-incremental scenario, processing the whole dataset at once, in order to focus solely on the features' impact. We analyzed these features: **(i)** frequency count of each cell value in the entire dataset; **(ii)** co-occurrence count of each cell value with values from the other attributes in the relation; **(iii)** number of constraint violations in which a cell is involved; and **(iv)** embedding features (similar to the ones of word2vec models[2] (MIKOLOV *et al.*, 2013a; MIKOLOV *et al.*, 2013b)). The preliminary evaluation showed that the co-occurrence and embedding features enable the best results when inferring repairs, whereas the remaining feature types have a lower impact on repair quality. As our main interest is in categorical data types, we employ only co-occurrence features[3].

To generate the features, we first compute statistics from the data. Our approach considers all the data received so far, i.e. the existing data from previous batches plus the incoming data, to calculate them. The first set of statistics consists of:

- **single-attribute frequencies**, which maintain one dictionary per attribute in the dataset, where each key in the dictionary is a value from the current attribute domain, and such value is associated with its corresponding frequency count. Formally, we define the single-attribute frequencies as

$$single\_freq = \{A_i,\ \{(v_{ik},\ freq(v_{ik})) \mid 1 \leq i \leq N,\ 1 \leq k \leq |D[A_i]|\}\}, \tag{3.1}$$

  where A_i is the i-th attribute in relation D, N is the number of attributes in the schema of D, v_{ik} is the k-th value in the value set of A_i, $freq(v_{ik})$ is the frequency count of v_{ik}, and $|D[A_i]|$ is the number of distinct values in A_i.

- **pairwise frequencies**, which maintain one dictionary per attribute pair, where each dictionary keeps track of all co-occurrences between domain values in the corresponding pair. The dictionaries have the form `{first_value -> {second_value -> count}}`,

[2] <https://code.google.com/archive/p/word2vec/>

[3] Employing embedding features in an incremental scenario would require different strategies from the ones presented in this work, which considers only categorical attributes.

where `first_value` is a domain value from the first attribute in the pair, `second_value` is a domain value from the second attribute in the pair, and `count` is the number of times both values co-occur across all the data. The pairwise statistics are formally defined as

$$\begin{aligned} pair\_freq = \{A_i,\ \{A_{i'},\ \{v_{ik},\ \{(v_{i'k'},\ freq(v_{ik},\ v_{i'k'}))\ |\ & i \neq i', \\ & k \neq k', \\ & 1 \leq i, i' \leq N, \\ & 1 \leq k \leq |D[A_i]|, \\ & 1 \leq k' \leq |D[A_{i'}]|\}\}\}\}, \end{aligned} \tag{3.2}$$

where A_i and $A_{i'}$ are distinct attributes in relation D, v_{ik} is the k-th value in the value set of A_i, $v_{i'k'}$ is the k'-th value in the value set of $A_{i'}$, N is the number of attributes in the schema of D, $freq(v_{ik},\ v_{i'k'})$ is the frequency count of v_{ik} and $v_{i'k'}$, $|D[A_i]|$ is the number of distinct values in A_i, and $|D[A_{i'}]|$ is the number of distinct values in $A_{i'}$. The incremental maintenance of frequencies computes both single-attribute and pairwise frequencies from the current data batch, then merges them with the respective frequencies from the previous data batches. The computational cost of maintaining frequencies is proportional to the batch size.

Another set of statistics corresponds to **attribute correlations**, based on the normalized conditional entropy of attribute pairs. The normalized conditional entropy ranges from 0 to 1: it equals 0 when the attributes are strongly correlated, whereas it equals 1 when the attributes are independent. Thus, we reverse the normalized conditional entropy value to reflect the attribute correlation. We compute the correlation of attributes x and y, denoted by $corr(x,y)$, as follows:

$$corr(x,y) = 1 - norm\_cond\_ent(x,y), \tag{3.3}$$

where $norm\_cond\_ent(x,y)$ is the conditional entropy of x and y normalized by the domain size of attribute x. Such normalization occurs by using the number of distinct values in x (obtained from the previously computed single-attribute frequencies) as the log base in the conditional entropy formula (described in Section 2.4). Specific cases that obviate the normalized conditional entropy computation are: **(i)** when $x = y$, which assigns the correlation value to 1, and **(ii)** when the domain size of x is 1, which sets the correlation value to 0. As the conditional entropy is asymmetric, we need to perform pairwise computations involving all pairs of attributes. In the context of an evolving dataset, to the best of our knowledge, the literature lacks a principled approach for incrementally maintaining the normalized conditional entropy of an attribute pair. We have elaborated such an incremental approach, which we formally describe in Appendix A.

The feature vector of a cell has the co-occurrence statistics for all its *candidate values*, which are part of the domain of that cell. Specifically, the candidate values for a cell are those in its domain which co-occur with the attribute values within the tuple to which that cell belongs. The candidate values are determined this way based on the intuition that values within the same tuple share a relationship, and thus are likely to co-occur in other tuples as well. That is, given

a cell c of a tuple t, the only useful values to be considered as evidences for a repair are those appearing in other tuple(s) (any tuple $t' \neq t$) alongside attribute values from tuple t. To visualize this process in a concrete situation, suppose the following example depicting four tuples of a relation, focusing on attributes B and A.

```
B | A
-----
h   b   <- Current tuple, attribute A under analysis (cell storing "b").
h   c      Cell domain is (b, c, e).
i   d
h   e
```

Consider that we are generating the cell domain for the first tuple (indicated as *Current tuple* in the example) at attribute A, i.e. the cell with value b. We can observe that b co-occurs with h in that tuple, which suggests that value h in B might have a relationship with values from attribute A. Therefore, other values in A co-occurring with value h are candidate values for the cell currently storing b. Hence, based on this reasoning, we have (b, c, e) as the cell domain.

The domain of a cell is formally defined as follows. Let $A_i[t] = v_{it}$ be the value of attribute A_i in tuple t, assigned to a cell denoted by c. Moreover, let $A_{i'}[t] = v_{i't}$ be the value of an attribute $A_{i'} \neq A_i$ in tuple t, and let $A_i[t'] = v_{it'}$ be the value of attribute A_i in another tuple $t' \neq t$. Finally, let $corr(A', A'')$ be the correlation strength of a pair of attributes A' and A'', and let Ω be a given correlation threshold. Then, the domain of cell c, denoted by $dom(c)$, is the following set:

$$\begin{aligned} dom(c) = \{v_{it}\} \cup \{v_{it'} \mid & \forall A_{i'} \neq A_i,\ \forall t' \neq t, \\ & \{v_{i't},\ v_{it'}\} \subset t', \\ & corr(A_{i'},\ A_i) > \Omega\}, \end{aligned} \tag{3.4}$$

where $\{v_{i't}, v_{it'}\}$ denotes the co-occurrence of $v_{i't}$ with $v_{it'}$.

The full feature vector of a cell c is a 2D tensor. Its first dimension refers to all values in $dom(c)$, whereas its second dimension refers to all attributes in the dataset. Specifically, each feature is a number representing the co-occurrence ratio of a value in $dom(c)$ with a value $v_{i't}$ (of another attribute $A_{i'}$ in tuple t) in the whole dataset received so far. Algorithm 1 shows the feature generation procedure in more detail. Lines 2–5 initialize and assign variables, including a 2D tensor for the feature vector that is initialized completely with zeros. The first dimension in the tensor accommodates the maximum domain size of cells from the `cell_attr` attribute (which varies per attribute, hence our ability to save memory by employing attribute-specific ML models). The second dimension, in turn, accommodates the total number of attributes in the relation. Then, the algorithm iterates over all attributes (except when the analyzed attribute is the given `cell_attr` attribute itself), generating the co-occurrence features for each candidate value in `cell_domain`. Lines 10–16 obtain the frequency values from the previously-computed

statistics, then employ such values to compute the `ratio` assigned to the corresponding tensor position. Finally, Algorithm 1 returns the 2D feature vector tensor for the current data cell.

Algorithm 1 – Feature vector generation for data cell from *cell_attr* attribute

```
1:  procedure GENERATE_FEATURE_VECTOR(cell_attr, cell_domain, tuple)
2:      max_domain ← get_max_domain(cell_attr)        ▷ Maximum number of distinct values in cell_attr
3:      all_attrs ← get_attributes()                                  ▷ List of attributes in relation
4:      tensor ← zeros(1, max_domain, length(all_attrs))          ▷ Dimensions for tensor initialization
5:      single_stats, pair_stats ← get_computed_statistics()
6:      for attr in all_attrs do
7:          if attr = cell_attr then
8:              continue                                  ▷ Skip iteration when attributes are the same
9:          end if
10:         value ← tuple[attr]
11:         value_freq = single_stats[attr][value]      ▷ Frequency of value across all the data received so far
12:         co_list = pair_stats[attr][cell_attr][value]   ▷ Values in cell_attr co-occurring with value in attr
13:         for candidate in cell_domain do
14:             co_freq ← co_list.count(candidate)                ▷ Frequency of candidate with value
15:             ratio ← co_freq / value_freq
16:             tensor[0][get_domain_position(candidate)][get_attr_position(attr)] ← ratio
17:         end for
18:     end for
19:     return tensor
20: end procedure
```

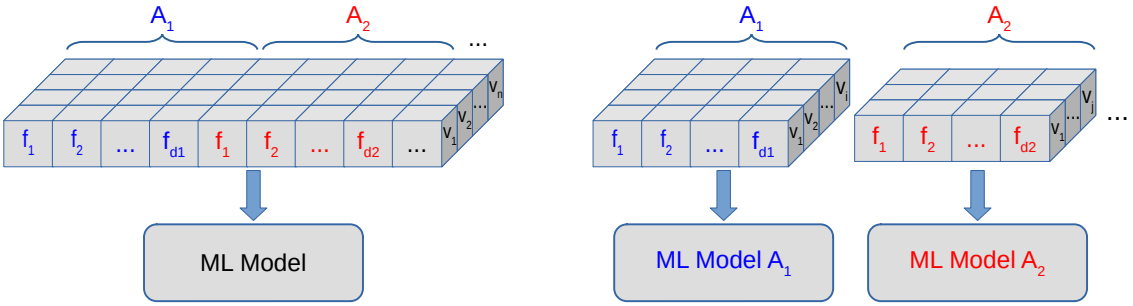
3.3 Model Training

The predictive model of HoloClean is a single-layer neural network. The model applies an exponential function over the product between the input feature vector and the learned weights, followed by applying the *softmax* function. The weights correspond to the importance of each feature to determine the most probable value of a cell to be repaired. As a single model is employed for all attributes, the feature vector has one feature set per attribute, and the weights for all attributes are jointly learned.

We extended the model design of HoloClean to employ multiple models, one per attribute, as shown in Figure 6. One of the benefits brought by this strategy is the size reduction of the feature vectors. The first dimension of the tensors is reduced because the maximum number of domain values now considers cells from a single attribute. Figure 6a shows that the single-model design requires reserving more positions in the tensors, in order to accommodate the largest domain size v_n across the whole dataset. Figure 6b shows that employing multiple models, on the other hand, allows reserving less positions in the tensors, corresponding to the largest domain size of each attribute, namely v_i for attribute A_1 and v_j for attribute A_2. In both designs, a mask is applied to disregard unused positions in the tensors for cells having smaller domain sizes, but nonetheless those positions are kept in memory. Reserving less positions in the first dimension is important to reduce memory consumption. Our strategy also reduces the second dimension by storing only one feature set (for the corresponding attribute), rather than one feature set per

attribute like in the previous model design, thus saving even more memory. In addition to the tensor size reduction, our model design brings the benefit of a fine-grained model maintenance. That is, it allows monitoring data variations in each attribute independently, so retraining can occur only when needed for specific models.

Figure 6 – Comparison of ML model designs.



(a) Model design of HoloClean. (b) Proposed model design.

Source: Elaborated by the author.

3.3.1 Training Skipping Strategy

We propose a strategy for monitoring joint attribute distributions, in order to identify expressive changes in data distributions that might require model retraining. Essentially, such changes correspond to the well-known notion of *concept drift*, which occurs when the patterns learned by the model(s) no longer hold. This strategy enables a sharp reduction of computational costs, as the decision of whether to retrain a model is based on the available space of attribute values, before starting the feature generation step.

Given a batch D_m, our strategy computes the joint distribution of each attribute pair A_i, A_j in the relation, $i \neq j$, considering their co-occurrence frequencies across batches $D_{1..m}$. Such a set of joint distributions is a simplification of the joint distribution over all attributes, and it is more compact and faster to compute. At the first batch D_1, our strategy trains all models. For the remaining batches, our strategy works as follows. It starts by fetching the training sets and joint distributions, for each attribute, from the last batch at which the model for A_i was trained. That is, considering that the model was trained at batch $D_h, 1 \leq h < m$, the training set and the joint distributions refer to batches $D_{1..h}$. Then, the decision of whether to train the model for attribute A_i is based on the KL divergence from each joint distribution (with the other attributes A_j, $i \neq j$) at the last batch that training occurred (over $D_{1..h}$), which we denote $P_h(A_i, A_j)$, to the corresponding joint distribution at the current batch (over $D_{1..m}$), which we denote $P_m(A_i, A_j)$. If a KL divergence $KL(P_m, P_h)$ is greater than a user-defined threshold ε_{kl}, the model should be retrained. The intuition is that a significant variation in the distributions was identified, and thus the data within batches $D_{(h+1)..m}$ contain new information that should be taken into account to

improve the model. Whenever a model is (re)trained, the corresponding joint distributions are saved for future reference.

We propose two variants for this strategy. The first one, denoted by *individual KL* (iKL), triggers model retraining for attribute A_i if it detects that any KL divergence value is greater than the threshold provided, i.e. $KL(P_m(A_i,A_j),P_h(A_i,A_j)) > \varepsilon_{kl}, i \neq j$. This variant is quite sensitive to distribution changes, as a single KL divergence greater than the given threshold, among all attributes A_j other than A_i, is considered to potentially impact repair accuracy. The second variant, called *weighted KL* (wKL), considers an aggregate value instead of the individual KL divergence. Specifically, each individual KL divergence is weighted by the correlation strength of the corresponding attribute pair, and the resulting aggregate value is compared to the user-defined threshold. In this variant, the model corresponding to attribute A_i is retrained if $wKL(A_i) > \varepsilon_{kl}$, and $wKL(A_i)$ is computed via the following equation:

$$wKL(A_i) = \frac{1}{N-1} \sum_{j \neq i} corr(A_i,A_j) \cdot KL(P_m(A_i,A_j),P_h(A_i,A_j)), \tag{3.5}$$

where N is the number of attributes in relation D, used as the normalization factor, and $corr(A_i,A_j)$ is the correlation strength of attributes A_i,A_j.

In the experimental evaluation, both variants of our strategy enable significant speed-ups in execution time, with a reduction in repair accuracy that can be tuned by the user-defined threshold ε_{kl}, as discussed in Section 3.4.

3.4 Experiments

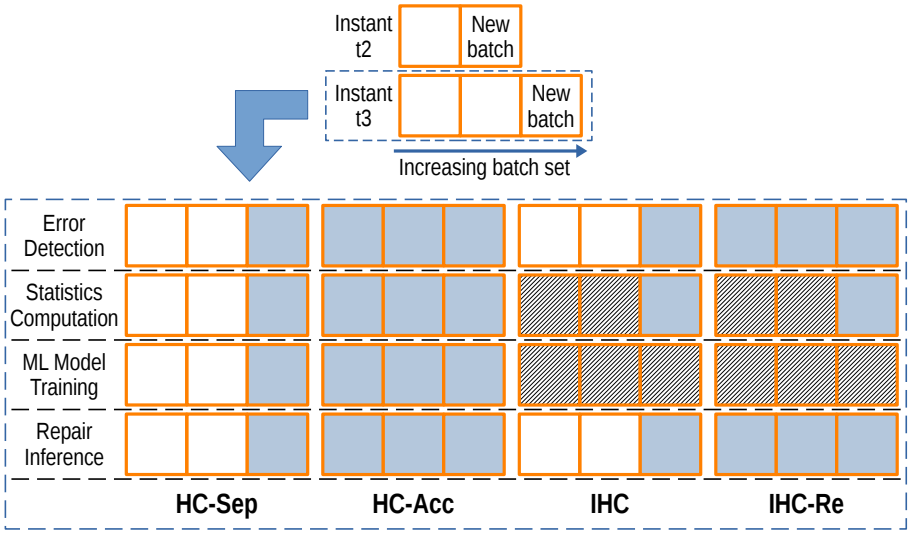
We evaluated four incremental data cleaning approaches: two competitors, based on the original HoloClean system, and two variants of our proposal, all of them employing the same fundamental components (i.e. error detectors, featurizers, and ML model type).

Evaluated Approaches

The approaches are shown in Figure 7 in terms of how they handle an incoming batch when performing error detection, statistics computation, model training, and repair inference.

HC-Sep. This approach executes the whole process of HoloClean once per data batch isolatedly. That is, each batch is handled as a brand new dataset, without the notion of a single dataset evolving. Therefore, feature generation and model training are performed from scratch for every batch, disregarding the knowledge obtained from previous batches. There is no information maintenance between batches, and this approach processes the data of a single batch at a time. Thus, considering an end-to-end execution of the data cleaning pipeline, this approach tends to be the fastest among the evaluated ones and the less demanding in terms of memory consumption,

Figure 7 – Overview of the evaluated approaches. The batches at instant $t3$ are shown as squares with the following notation: **(i)** blank squares are untouched batches; **(ii)** filled squares are processed batches; and **(iii)** hatched squares are batches that are more quickly processed/inspected.



Source: Elaborated by the author.

despite employing a single ML model for all attributes. Conversely, it tends to have the lowest repair quality, as it lacks an increasingly global view of the data.

HC-Acc. This approach employs the entire process of HoloClean as well, but accumulating incoming batches as an evolving dataset. Hence, instead of processing each data batch in isolation, HC-Acc appends every new batch to all the data seen so far and processes the entire dataset from scratch. Consequently, after appending the last incoming batch, this approach ends up processing the full dataset. Therefore, HC-Acc has the drawback of being the most time- and memory-consuming among all evaluated approaches. Furthermore, in a hypothetical scenario with limited computational resources, this approach might not be able to be employed. On the other hand, it tends to have the highest repair quality as it increasingly gets a global view of the dataset, being able to generate comprehensive data features to train its ML model, as well as to benefit from previous repairs while adjusting features and the model.

IHC. This approach is our basic proposal. It is an incremental version of HoloClean that cleans the incoming (m-th) batch with ML models trained using all batches seen so far, based on evolving features generated from statistics that are incrementally updated at each new data batch (hence the hatched squares in Figure 7 for "Statistics Computation"). This approach was evaluated for both variants of the proposed training skipping strategy (which explains the hatched squares in Figure 7 for "ML Model Training"), referred to as IHC-iKLε_{kl} and IHC-wKLε_{kl} accordingly, where $\varepsilon_{kl} \in \{0.1, 0.05, 0.01\}$. Such values for ε_{kl} were empirically obtained through a grid search over samples from the datasets.

IHC-Re. This approach is a variant of our basic proposal. Its main difference from IHC is that, when it receives the m-th batch, it seeks to repair errors spotted across batches $1..m$, i.e.

fixing errors spotted in cells from previous batches in addition to errors from the current batch. One of the intuitions behind IHC-Re is that certain repairs previously made in earlier batches might not have been successful. Furthermore, certain errors might not have been detected in previous batches because the available tuples at that moment were not enough to identify certain constraint violations. Therefore, unlike IHC, this variant does not assume that the previous repairs fully cleaned the data. Rather, it assumes that previous repairs turned the data into the cleanest state possible, given the available features and training data. Overall, IHC-Re presents higher repair quality than IHC does, but requires longer execution times and a greater memory consumption. Just like IHC, this approach also updates statistics incrementally and applies our training skipping strategy, thus performing feature generation and model training more quickly than HC-Acc (and hence the hatched squares in Figure 7 for IHC-Re as well).

We employed three error detectors: **(i)** Null Detector; **(ii)** Constraint Violation Detector, which spots Denial Constraints, hence being referred to as DC Detector in the experimental evaluation; and **(iii)** Perfect Detector, which emulates the ideal situation where the set of dirty cells detected is exactly the set of true errors in the dataset, obtained from the ground-truth data. Employing the Perfect Detector allows decoupling data repair from error detection, and this is useful for evaluating the approaches solely on data repair, which is the main focus of this work.

The implementation was based on PyTorch and PostgreSQL, and the experiments were carried out in a machine equipped with a 12-core Intel Xeon E5-2603 1.60GHz, 64GB 2133MHz RAM memory, 7200RPM SATA 6Gbps HDD, and Ubuntu 14.04.6 LTS operating system.

Datasets

We employed three datasets for the reported experiments, which are described in Table 2.

Table 2 – Datasets used in the experiments.

Dataset	# of tuples	# of attrs	Error rate	Error detector(s)
Hospital	1,000	17	∼3%	Null + DC
Food	5,000	13	∼0.5%	Null + DC
Soccer-12.5k01pct	12,500	10	0.1%	Perfect
Soccer-12.5k1pct	12,500	10	1%	Perfect
Soccer-12.5k10pct	12,500	10	10%	Perfect
Soccer-25k01pct	25,000	10	0.1%	Perfect
Soccer-25k1pct	25,000	10	1%	Perfect
Soccer-25k10pct	25,000	10	10%	Perfect
Soccer-50k01pct	50,000	10	0.1%	Perfect
Soccer-50k1pct	50,000	10	1%	Perfect
Soccer-50k10pct	50,000	10	10%	Perfect

Source: Elaborated by the author.

The Hospital dataset is commonly employed in the data cleaning literature and is based on hospital census information. The errors are artificially inserted typos. Food contains data about eating establishments in Chicago. The errors are inconsistencies of values between pairs of tuples, and the ground truth was manually generated by fixing a small subset of the existing errors. Soccer is a synthetic dataset about soccer players and their teams, with errors injected by the tool BART (AROCENA *et al.*, 2015). We employed a subset of the example available on BART's website[4]. For all experiments, the datasets were loaded in 100 sequential batches, each corresponding to 1% of the dataset size.

We present the DCs for the Hospital and Food datasets, where D refers to the corresponding relation of the dataset. The DCs for the Hospital dataset are the following.

$$dc_1 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.condition = t_\beta.condition \wedge t_\alpha.measure\_name = t_\beta.measure\_name \wedge t_\alpha.hospital\_type \neq t_\beta.hospital\_type)$$

$$dc_2 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.zip\_code \neq t_\beta.zip\_code)$$

$$dc_3 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.phone\_number \neq t_\beta.phone\_number)$$

$$dc_4 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.measure\_code = t_\beta.measure\_code \wedge t_\alpha.measure\_name \neq t_\beta.measure\_name)$$

$$dc_5 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.measure\_code = t_\beta.measure\_code \wedge t_\alpha.state\_avg \neq t_\beta.state\_avg)$$

$$dc_6 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.provider = t_\beta.provider \wedge t_\alpha.hospital\_name \neq t_\beta.hospital\_name)$$

$$dc_7 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.measure\_code = t_\beta.measure\_code \wedge t_\alpha.condition \neq t_\beta.condition)$$

$$dc_8 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.address1 \neq t_\beta.address1)$$

$$dc_9 : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.hospital\_owner \neq t_\beta.hospital\_owner)$$

$$dc_{10} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.provider \neq t_\beta.provider)$$

$$dc_{11} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.phone\_number = t_\beta.phone\_number \wedge t_\alpha.hospital\_owner = t_\beta.hospital\_owner \wedge t_\alpha.state \neq t_\beta.state)$$

$$dc_{12} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.city = t_\beta.city \wedge t_\alpha.county\_name \neq t_\beta.county\_name)$$

$$dc_{13} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.zip\_code = t_\beta.zip\_code \wedge t_\alpha.emergency \neq t_\beta.emergency)$$

$$dc_{14} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.hospital\_name = t_\beta.hospital\_name \wedge t_\alpha.city \neq t_\beta.city)$$

[4] http://db.unibas.it/projects/bart/

$$dc_{15} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.measure\_name = t_\beta.measure\_name \wedge t_\alpha.measure\_code \neq t_\beta.measure\_code)$$

The DCs for the Food dataset are the following.

$$dc_{16} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.dba\_name = t_\beta.dba\_name \wedge t_\alpha.address = t_\beta.address \wedge t_\alpha.facility\_type \neq t_\beta.facility\_type)$$

$$dc_{17} : \forall(t_\alpha, t_\beta) \in D,\ \neg(t_\alpha.zip\_code = t_\beta.zip\_code \wedge t_\alpha.city \neq t_\beta.city)$$

$$dc_{18} : \forall t_\alpha \in D, \neg(t_\alpha.facility\_type = \text{‘empty’})$$

$$dc_{19} : \forall t_\alpha \in D, \neg(t_\alpha.city = \text{‘empty’})$$

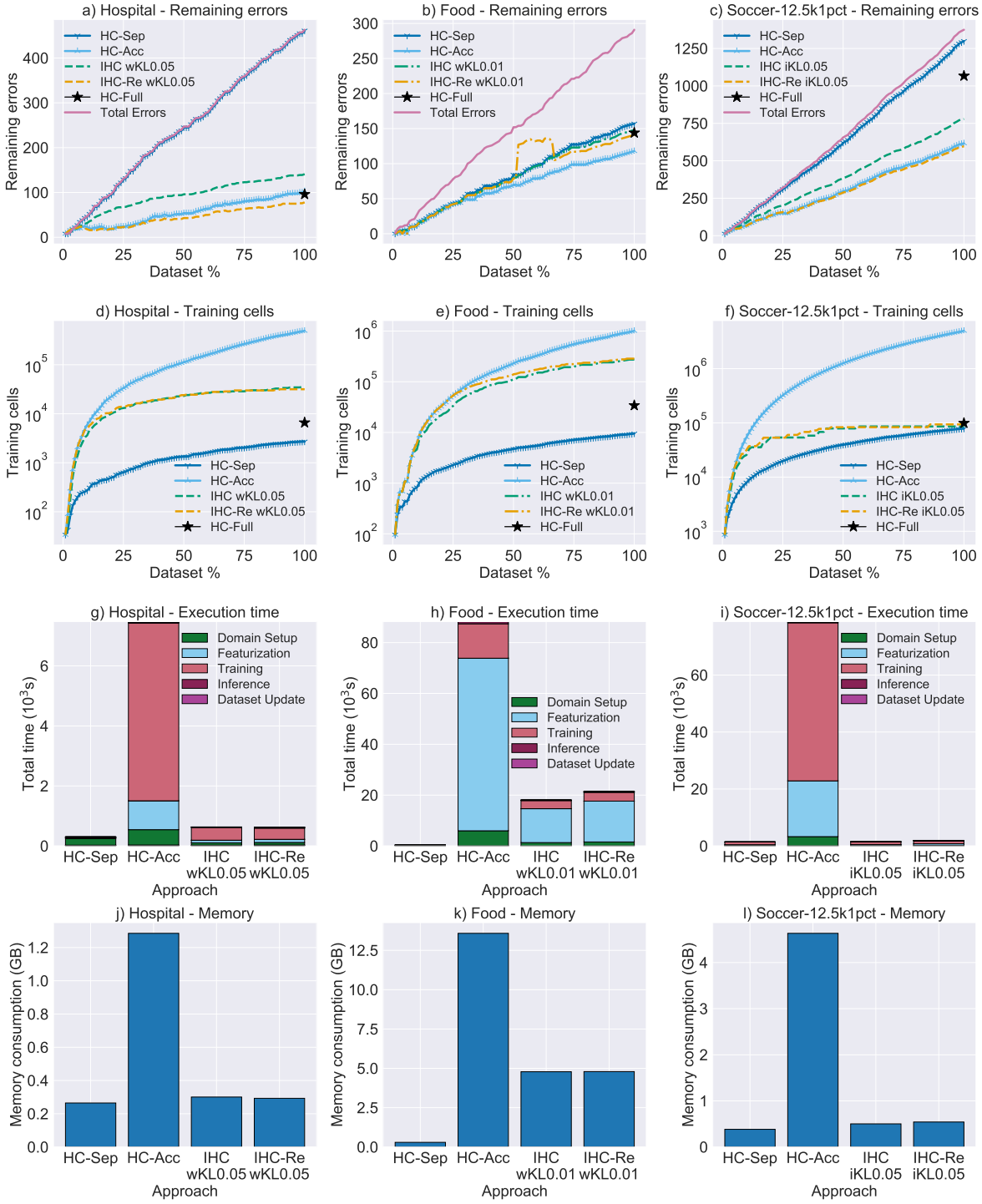
$$dc_{20} : \forall t_\alpha \in D, \neg(t_\alpha.zip\_code = \text{‘empty’})$$

3.4.1 Performance Overview

This section shows the experiments performed to evaluate the four approaches on repair quality and computational efficiency. We ran experiments with all three values of the ε_{kl} hyper-parameter and, after analyzing the results, we selected the best value for each dataset, which we report in this section. The results are shown in Figure 8 and discussed as follows.

With respect to **data cleaning effectiveness**, the charts at the first row show the number of accumulated errors at each incoming batch, as well as the total number of remaining errors after executing each approach. For reference, the figure depicts a star marker representing the number of non-repaired errors after executing HoloClean over the whole dataset (HC-Full). Figure 8a shows that the approach HC-Sep barely repaired the data, whereas the remaining approaches achieved cleaning rates greater than 70%. This behavior was expected, as the batch size is rather small for this dataset, and it also shows that updating features incrementally is important for the whole process. Compared to HC-Full, the HC-Acc approach was able to fix a similar number of errors. Our approach IHC-Re-wKL0.05 was superior, as it leverages the repairs performed in previous batches (which affect the features and models at posterior batches), in addition to employing attribute-based models. With respect to Food (Figure 8b), HC-Sep was quite effective at repairing, being slightly surpassed by IHC-wKL0.01 and IHC-Re-iKL0.01. IHC-Re-wKL0.01 enabled a superior result compared to HC-Full, although having presented a prediction problem between batches 51 and 67, but which was fixed afterwards. With respect to the Soccer12k01pct dataset (Figure 8c), HC-Sep performed fewer repairs once again. Interestingly, the number of repairs provided by the remaining approaches was much superior compared to the execution over the entire dataset, which shows that, for this dataset, maintaining the models and features incrementally was significantly more effective. Overall, we can observe that IHC-Re-iKL0.05 was the best approach, followed closely by HC-Acc and a bit farther away by IHC-iKL0.05.

Figure 8 – Overview of repair quality and computational efficiency of the proposed approaches IHC and IHC-Re against competitors HC-Sep and HC-Acc.



Source: Elaborated by the author.

Focusing on **training effort**, the second row of charts in Figure 8 shows the accumulated number of instances used for training by each approach. Regarding Hospital (Figure 8d), the proposed approaches (IHC-wKL0.05 and IHC-Re-wKL0.05) used one order of magnitude more instances than HC-Sep, as well as one order of magnitude less instances than HC-Acc (note the

logarithmic scale on the charts). With respect to Food (Figure 8e), IHC-wKL0.01 and IHC-Re-wKL0.01 got closer to HC-Acc. Considering Soccer12.5k1pct, the proposed approaches used a similar number of instances compared to HC-Sep, which shows that our ML models converged more quickly. For all datasets, we can observe that HC-Sep used less instances than HC-Full due to differences both in the number of reported errors and in the number of labelled cells, generated by the *weak-labelling* step of HoloClean for augmenting the training set.

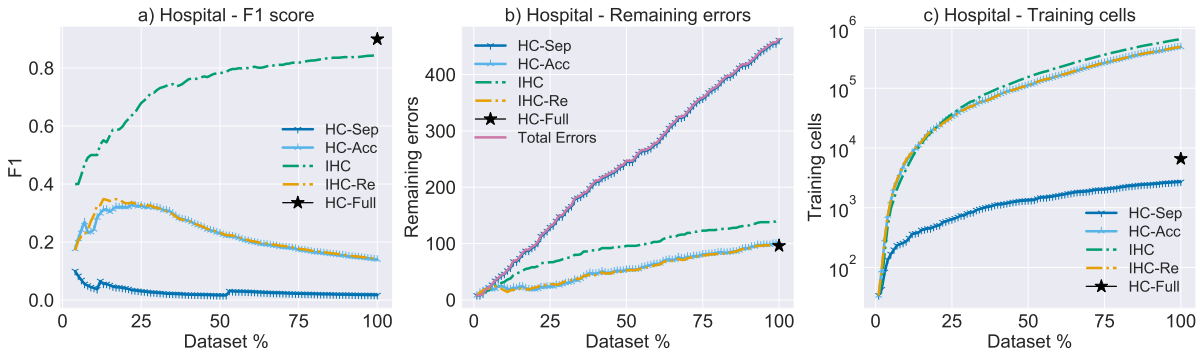
The third row of charts in Figure 8 shows the total **execution time** after processing all the batches. It is clear that the exhaustive execution performed by HC-Acc is too time-consuming and tends to be unsuitable for large datasets. Considering both proposed approaches, IHC enabled the best results in terms of execution time, taking just 2%–20% of the time spent by the HC-Acc competitor. A particular behavior regarding Food is the high cost for the feature generation step, which surpassed the execution time for model training. This occurred due to the attribute domains being too large. Overall, the HC-Sep approach was much faster than the other approaches, in detriment of repair quality.

Finally, with respect to **memory consumption**, the last row of charts in Figure 8 shows that our proposals reduced significantly the amount of memory required. Specifically, the IHC and IHC-Re approaches required 10%–35% the maximum amount of memory used by HC-Acc.

3.4.2 Analysis of Attribute-Based Models

A key point in our proposal is the adoption of attribute-based ML models. Both IHC and IHC-Re approaches had similar (and lower) feature generation and training costs compared to HC-Acc. Particularly, IHC-Re can be seen as a variant of HC-Acc that employs attribute-based models. Hence, our expectation was that attribute-based models would provide a similar repair quality compared to the original single-model design, although requiring less memory.

Figure 9 – Repair quality and number of training instances in attribute-based vs. single-model approaches.

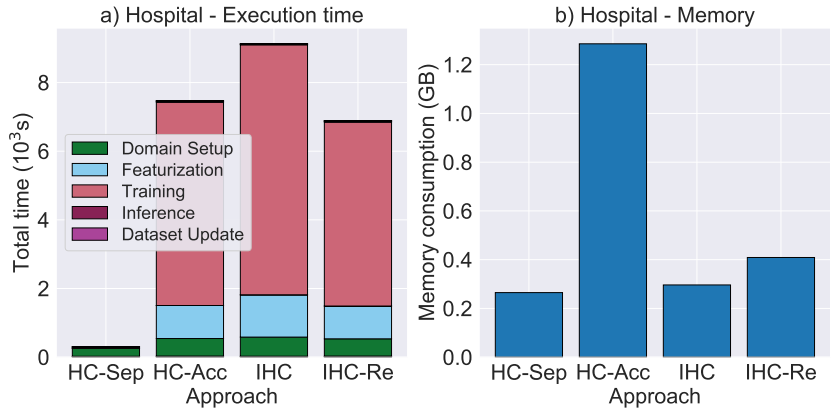


Source: Elaborated by the author.

Figure 9a presents the F1 score of each approach over Hospital. The figure shows that the repair quality of IHC-Re was similar to the one of HC-Acc, and IHC had a considerably

higher F1 score, close to the score of HC-Full. The attempt to re-repair errors from previous batches decreases the recall in IHC-Re and HC-Acc, as they repeatedly consider cells that could not be repaired previously. Hence, to effectively compare all the approaches in terms of repair quality, we show the accumulated number of non-repaired errors (Figure 9b). Furthermore, the accumulated number of training instances was similar for approaches HC-Acc, IHC, and IHC-Re (Figure 9c). Overall, the same behavior occurred for the remaining datasets.

Figure 10 – Execution time and memory consumption in attribute-based vs. single-model approaches.



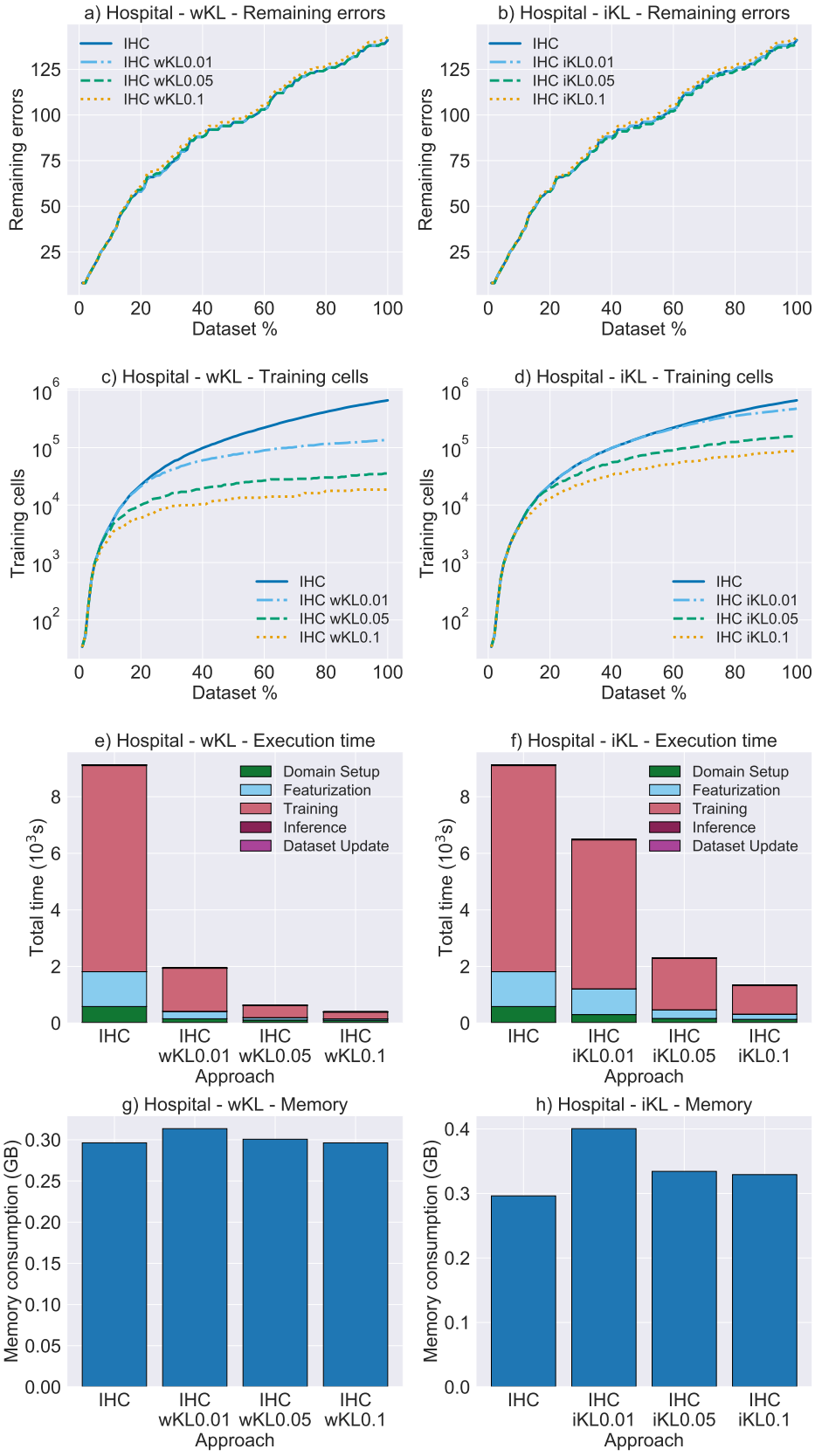
Source: Elaborated by the author.

With respect to execution time, Figure 10a shows that the total times of IHC and IHC-Re were relatively similar to the one of HC-Acc and quite longer compared to HC-Sep. For this specific case, the execution time for IHC-Re was shorter than for IHC. This happened because, while certain dirty cells from initial batches were cleaned when being revisited by IHC-Re, IHC left them untouched. Therefore, IHC allowed those cells to conflict with clean cells from the posterior batches, which caused such clean cells to be marked as potential errors. This fact made IHC spend more time by needlessly generating features for clean cells and attempting to repair them in posterior batches. Considering memory consumption, the proposed approaches were way better than HC-Acc, as shown in Figure 10b. Moreover, combining attribute-based models with our strategy for skipping model training enabled a sharp reduction in execution time, as we show in the next section.

3.4.3 Analysis of Training Skipping Strategy Variants

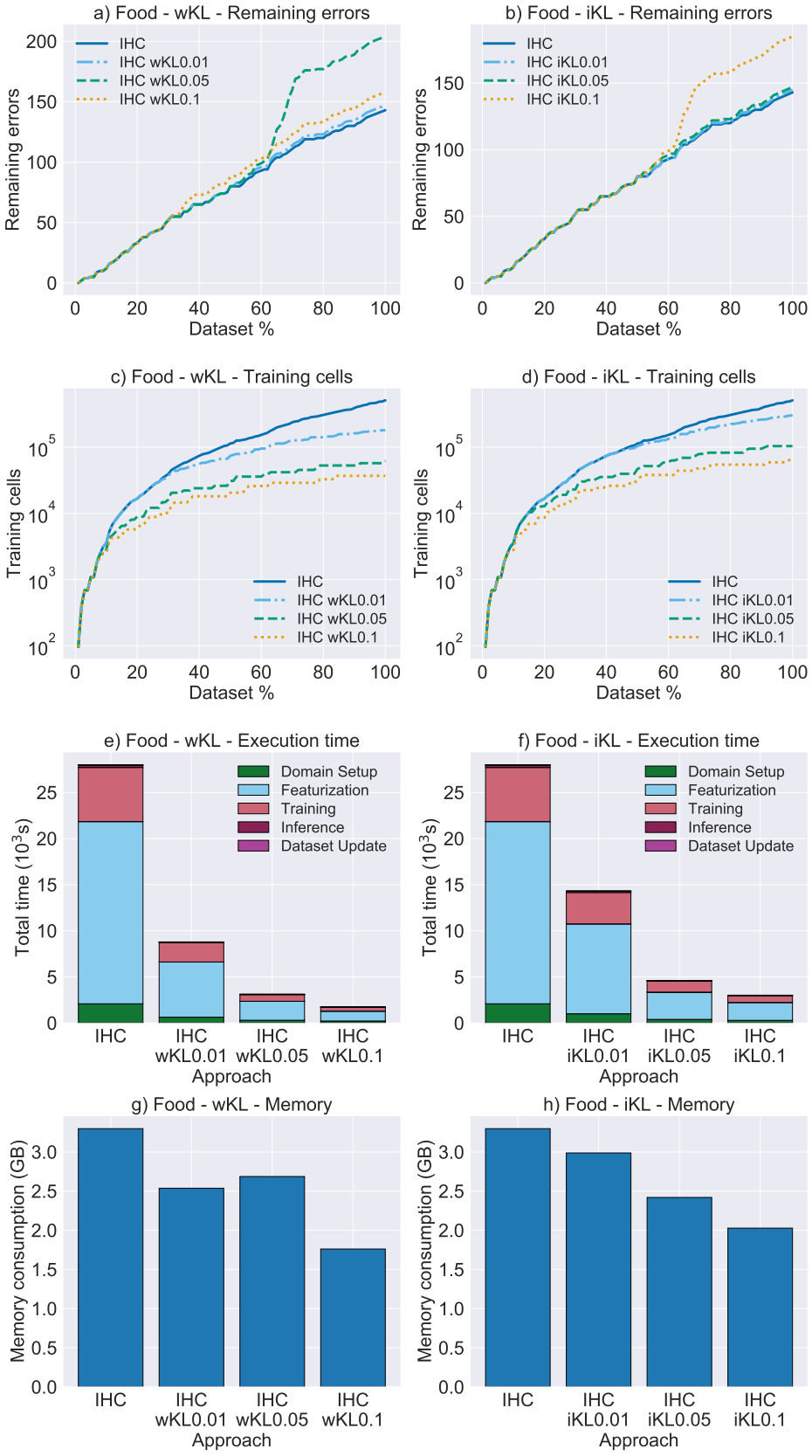
Domain and feature generation are among the most time-consuming tasks in our pipeline. Therefore, we focused on improving computational performance by attempting to skip these tasks whenever model retraining is not needed. Indeed, the proposed training skipping strategy led to significant performance improvements. The strategy reduces execution time according to a threshold parameter, in order to control the resulting impact on repair quality. This section presents the behavior of our data cleaning approaches when employing the iKL and wKL variants, taking as examples the datasets Hospital (Figure 11) and Food (Figure 12).

Figure 11 – Behavior of threshold variation in training skipping strategy for the Hospital dataset.



Source: Elaborated by the author.

Figure 12 – Behavior of threshold variation in training skipping strategy for the Food dataset.



Source: Elaborated by the author.

Regarding Hospital, Figure 11 shows that the highest threshold value ($\varepsilon_{kl} = 0.1$) enabled the fastest execution time, incurring just a slight reduction in the number of repairs compared to the other threshold values. Furthermore, IHC-iKL0.05 had the highest repair quality, which points out that more training not always leads to more repairs. We can also see that the number of training instances and the execution times decrease as the provided threshold increases. For instance, IHC-wKL0.1 spent about 2% the time taken by IHC, with a slight difference in the number of repairs. Regarding memory consumption, there is an overall reduction in the required memory amount as the threshold increases, although such reduction is less significant than the ones in the number of training instances and in execution time. Overall, we observed that the iKL strategy is more sensitive to changes than wKL, hence leading to model retraining more often.

A similar behavior could be seen for Food (Figure 12). However, compared to Hospital, distinct threshold values had a more significant effect on the resulting numbers of repairs, which were significantly more different from each other as the threshold varied. A particular situation occurred for approaches IHC-wKL0.05 and IHC-iKL0.1, which was a problem between batches 60 and 70 that led to unsatisfactory repairs. The models regained their performance around batch 70, which could have enabled a delayed repair of the affected cells if the re-repairing mechanism of IHC-Re were being employed.

3.4.4 Impact of Data Variations on the Evaluated Approaches

This section describes the behavior observed in the proposed approaches, compared to the competitors, as the data vary by size and error rate. This analysis was based on the Soccer dataset employing the Perfect error detector, in order to isolate the data repairing step and evaluate the parameters in a controlled environment.

We begin the analysis by inspecting the accumulated execution times, shown for each evaluated approach in Figure 13, as the Soccer12.5k1pct dataset grows larger. Both plots on the upper section of the figure depict the times for competitors HC-Acc and HC-Sep respectively. As expected, the execution time for HC-Acc increases exponentially as more batches are cleaned, owing to the processing of each batch becoming more comprehensive. In this experiment, the total execution for HC-Acc was around 56× longer than for HC-Sep. Right below in the figure, the accumulated execution times for HC-Sep are presented using a different scale, showing that the growth is linear, since the processing efforts at each new batch are constant for this approach. With respect to the proposed approaches IHC and IHC-Re making use of our training skipping strategy, the execution time evolution is sublinear. Compared to HC-Sep, we can observe higher amounts of effort in the initial batches. However, these amounts start to increase more slowly over time, as the ML models become stable and the need for retraining is reduced. The growth of execution time for IHC-Re is greater than for IHC, owing to the costs related to cell domain computation, feature generation, and dataset updates on cells from previous batches identified as erroneous. The (re)training costs in both approaches were almost the same, since the same

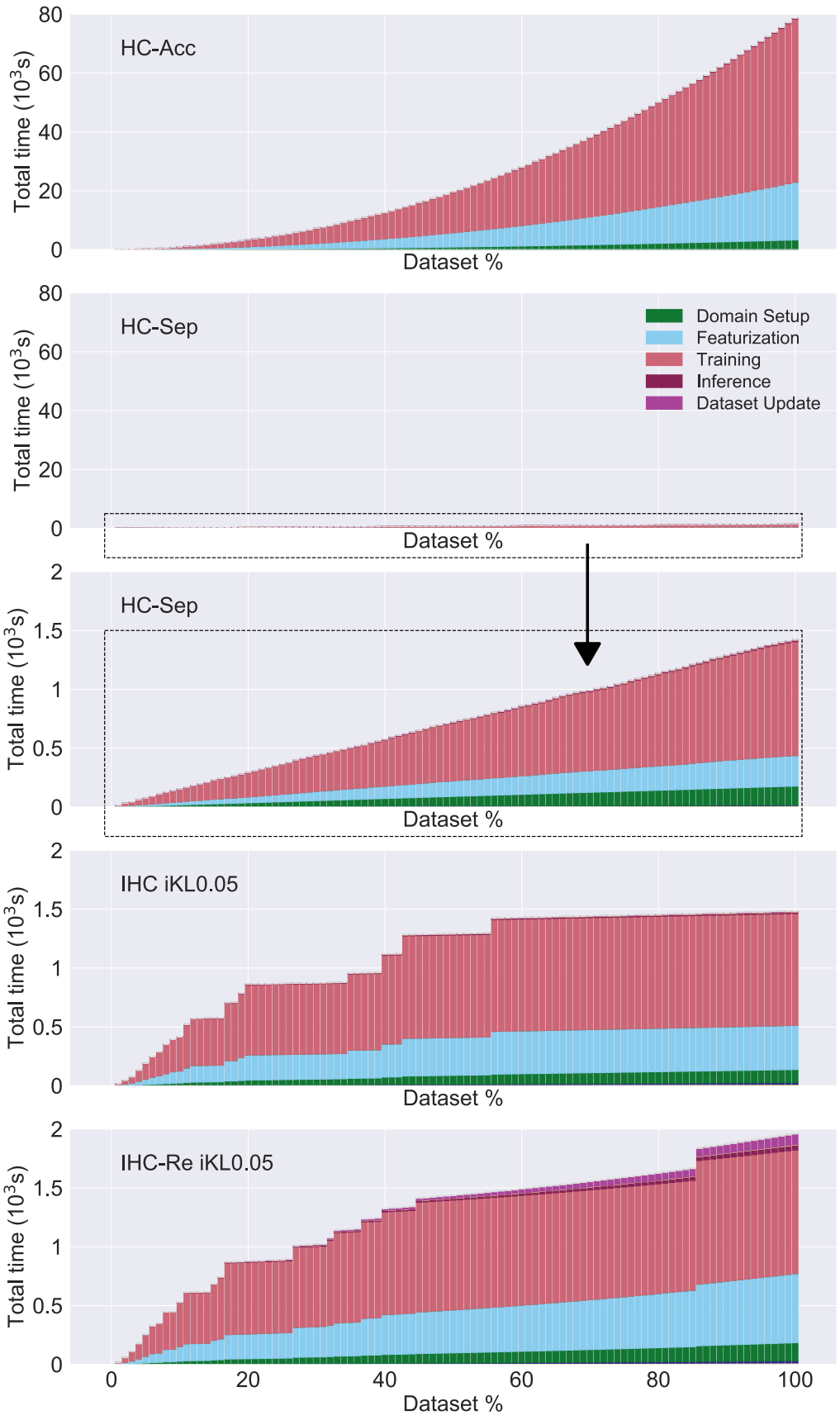
training skipping strategy and threshold value (iKL0.05) were employed. Based on this analysis, further results regarding the approach HC-Acc were not inspected for larger datasets.

Figure 14 shows our analysis on **varying dataset size**, considering results over the Soccer dataset having different amounts of tuples and a fixed error rate of 1%. The results for HC-Acc are presented just for the dataset with 12,500 tuples, as its costs become prohibitively high over larger datasets. As we can observe on the first row of plots in the figure, IHC-Re is the approach that overall enables the greatest number of repairs, followed by HC-Acc (when evaluated) and by IHC. For all cardinalities, these approaches were also better than executing HoloClean over the full dataset. The HC-Sep approach presented an increasingly higher repair quality as the dataset size grew larger, since the batches themselves also became larger. Yet, HC-Sep was still the worst approach. Considering the number of training instances, represented by the second row in the figure, we can notice that employing the proposed training skipping strategy allowed saving efforts as the dataset grew, since the models were able to stabilize with less processed batches. With respect to execution time, our approaches increasingly outperformed HC-Sep as the dataset grew larger. The plots in Figure 14h and Figure 14i, in turn, depict the overhead of IHC-Re in several tasks, due to its re-repairing mechanism. Finally, the last row of plots show the sublinear memory consumption growth as the dataset evolves.

Considering **varying error rate**, Figure 15 shows the behavior of each approach over Soccer with 25,000 tuples, using error rates of 0.1%, 1%, and 10%. In these experiments, the error rate refers to the fraction of *cells* (not *tuples*) that are dirty. We can observe that all approaches degrade as the error rate increases. This behavior was expected, as predicting the most likely repair relies on information extracted from the dataset itself and, typically, the greater the noise, the lower the accuracy. Regarding the number of training instances, our approaches IHC-iKL0.05 and IHC-Re-iKL0.05 ended up requiring less instances than HC-Sep for the error rates of 1% and 10%, which points out that our models stabilized more quickly. Considering execution time, all three approaches took longer to execute as the error rate increased, and IHC-Re-iKL0.05 was the one that took significantly longer, especially due to featurization. Overall, we see that memory consumption increased slightly with the error rate.

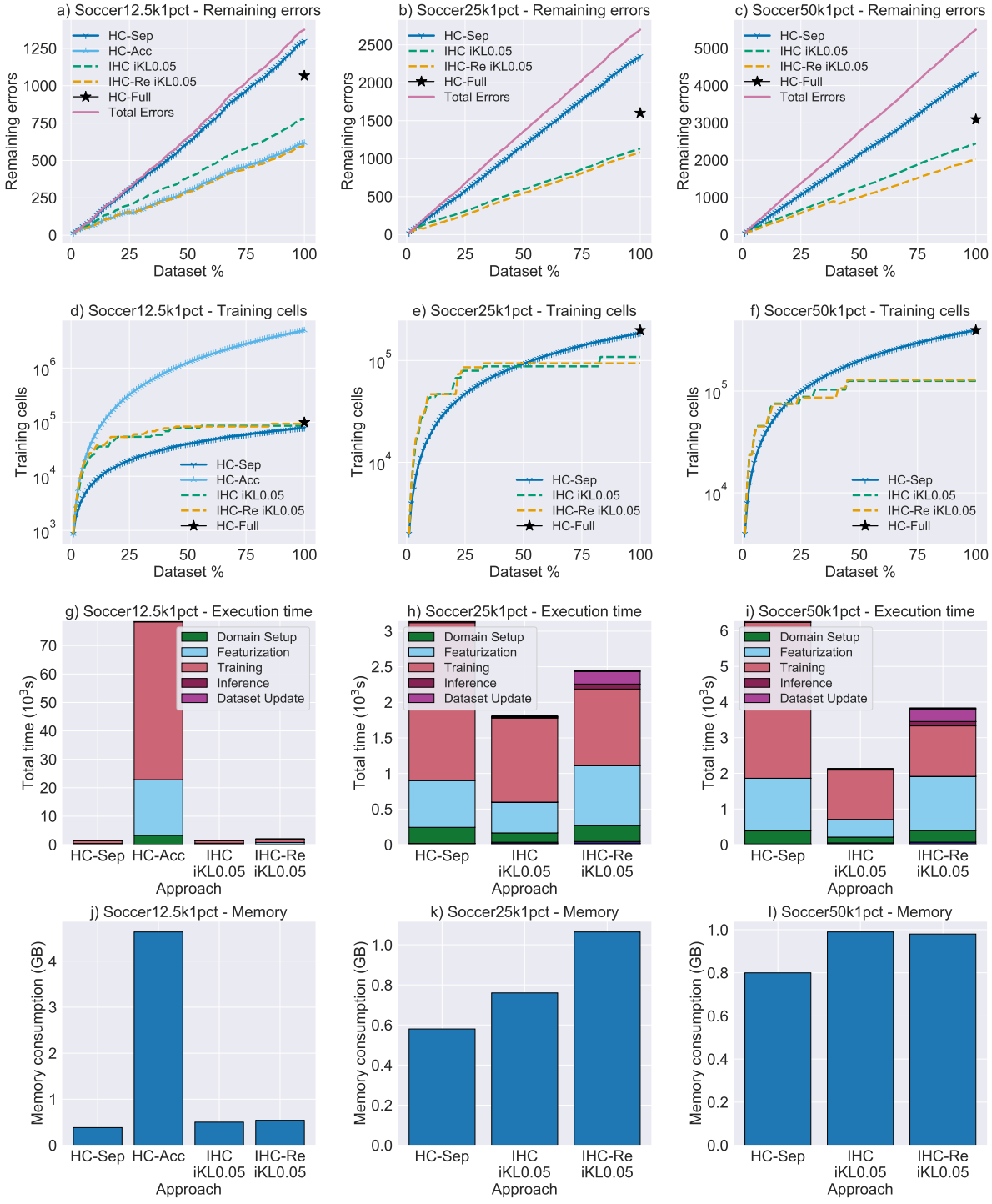
The last experiment analyzes the approaches with respect to the training skipping strategy variants, which were employed with multiple threshold values over the Soccer25k dataset with distinct error rates. Figure 16 and Figure 17 show the overall behavior of approaches IHC-Re and IHC, respectively, employing strategy variants iKL and wKL with threshold values 0.01, 0.05, and 0.1. For all evaluations, the number of repairs was not significantly impacted by the training skipping strategy variant nor by the threshold value — in both figures, all lines have almost the same shape within the graphs a), d), and g). In contrast, the threshold values were quite relevant when considering total execution time and, at a lowest degree, memory consumption. The total execution times in IHC-Re were incrementally affected by higher error rates (Figure 16b, Figure 16e, and Figure 16h), whereas for IHC such behavior was not seen (Figure 17b, Figure 17e, and

Figure 13 – Overview of accumulated execution times for the evaluated approaches over Soccer12.5k1pct.



Source: Elaborated by the author.

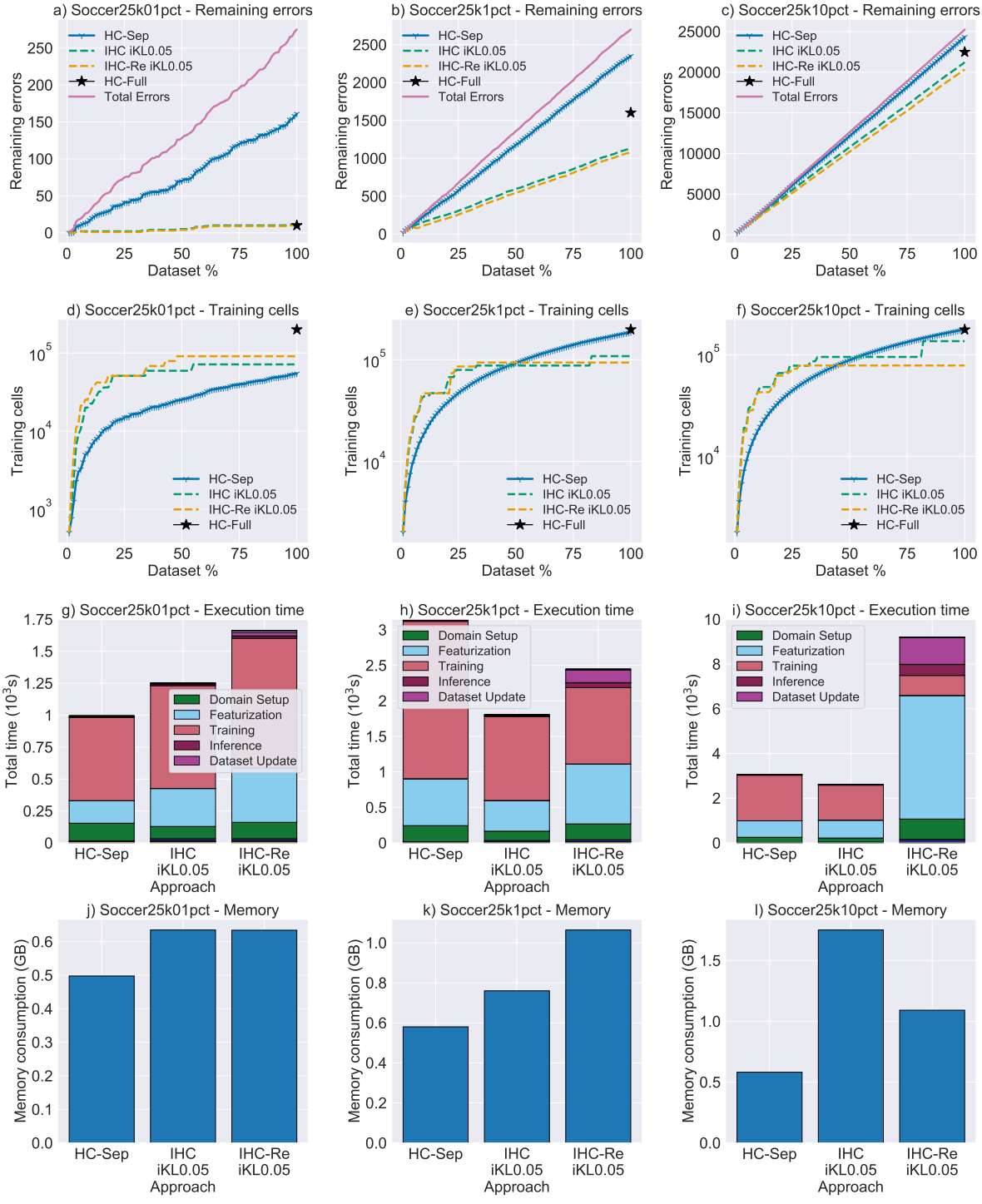
Figure 14 – Comparison of approaches with varying dataset sizes and a fixed error rate of 1%.



Source: Elaborated by the author.

Figure 17h).

Figure 15 – Comparison of approaches over the Soccer25k dataset with varying error rates.

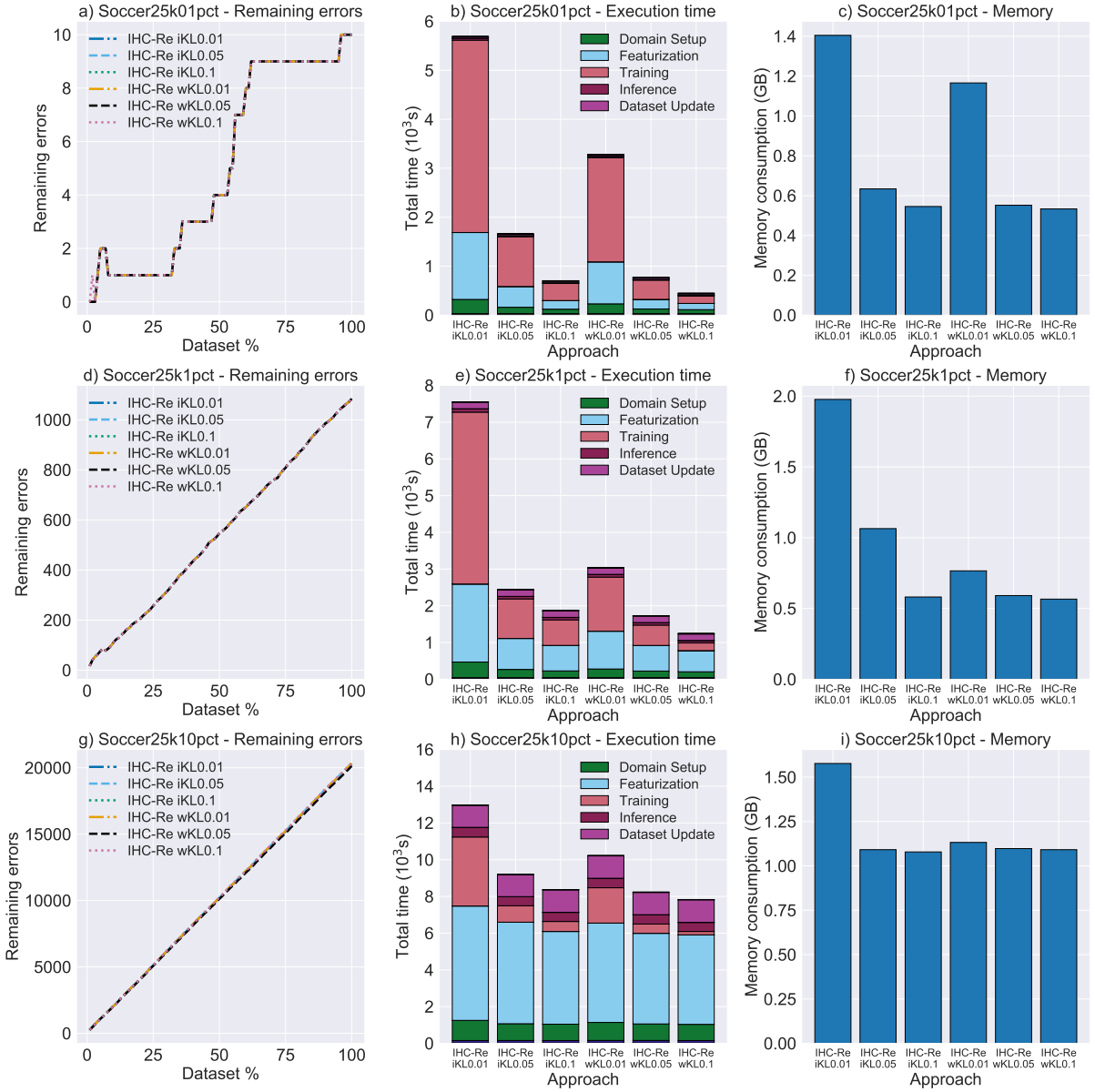


Source: Elaborated by the author.

3.5 Discussion

This chapter has presented our contributions for tackling the problem of holistic data cleaning incrementally. In addition to cleaning data in batches, our approach employs mechanisms that keep track of the acquired knowledge over time, embedding such knowledge in our ML

Figure 16 – Behavior of training skipping strategy variants in approach IHC-Re, considering threshold values 0.01, 0.05, and 0.1, over Soccer25k with varying error rates.

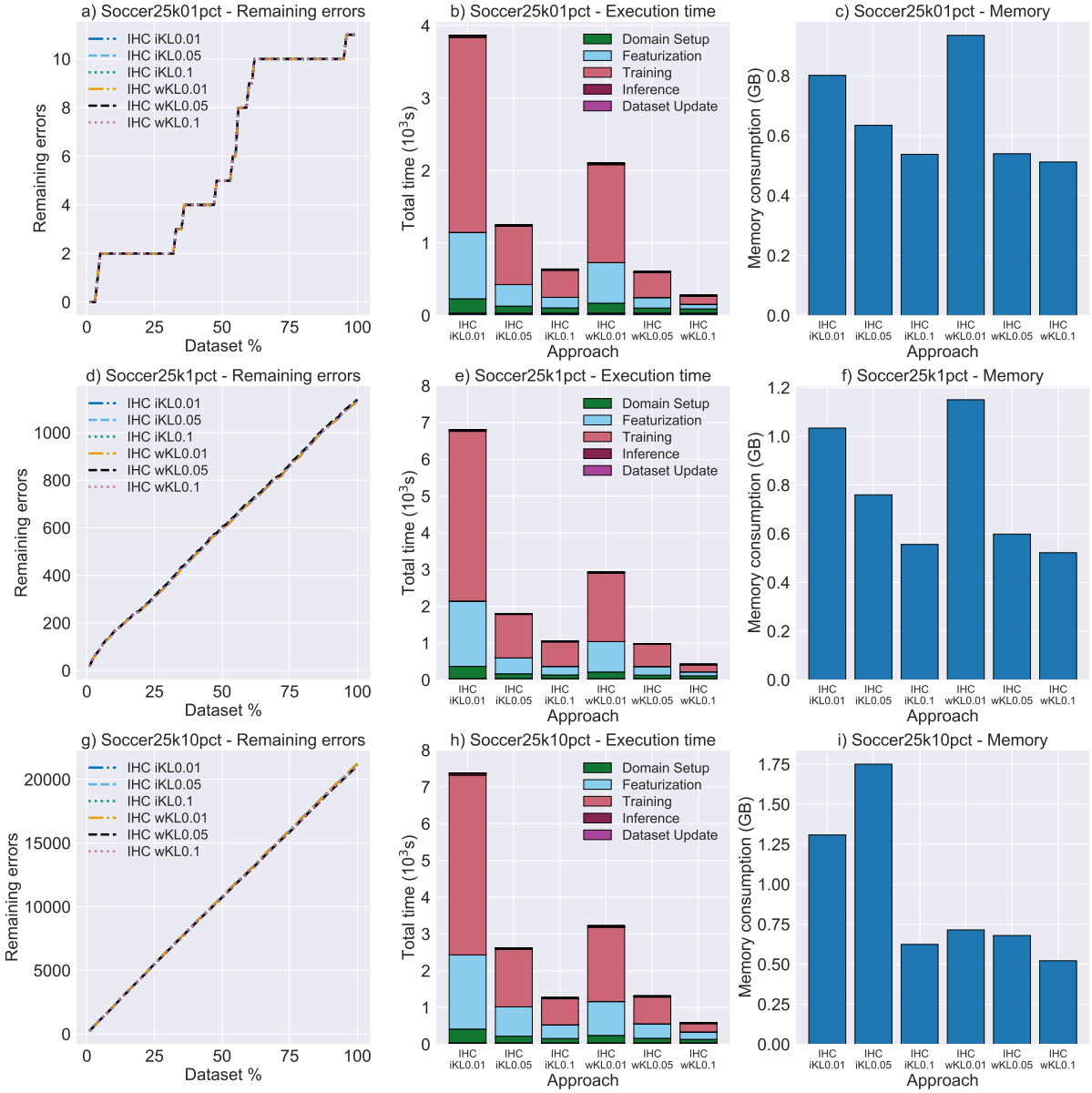


Source: Elaborated by the author.

models. To the best of our knowledge, our contributions have led to the first incremental data cleaning framework for performing repairs **(i)** independently of user interactions, **(ii)** without requiring prior knowledge about the incoming dataset such as the number of classes per attribute, and **(iii)** holistically, being able to simultaneously consider multiple signals for feature generation as well as performing repairs of multiple error types.

Our batchwise approach for data cleaning enabled significant gains in repair quality when compared to the evaluated competitors, occasionally outperforming even the HoloClean system, which was our non-incremental baseline approach. For instance, while HoloClean fixed 22% of Soccer-12.5k1pct dataset, the IHC-Re-iKL0.05 approach fixed 57%. Overall, IHC-Re was the

Figure 17 – Behavior of training skipping strategy variants in approach IHC, considering threshold values 0.01, 0.05, and 0.1, over Soccer25k with varying error rates.



Source: Elaborated by the author.

best approach with respect to repair quality, closely followed by the HC-Acc competitor and by the other variant of our proposal, IHC. We were able to use significantly less memory as well, as our proposed approaches required just 10%–35% of the maximum amount of memory used by the HC-Acc competitor (which ends up processing the whole dataset at the last batch). Furthermore, regarding execution time, we were overall between the HC-Sep and HC-Acc approaches (our lower- and upper-bound incremental competitors, respectively), but significantly outperforming HC-Acc while compromising little on repair quality. Specifically, the IHC approach employing the proposed training skipping mechanism managed to take just 2%–20% of the total execution time taken by the HC-Acc competitor.

The contributions presented in this chapter enable an important next step towards automatizing incremental data cleaning. This is an essential task taking place in the beginning of the data analytics pipeline, as it allows bringing more value to the data before any analytical task is carried out. In the next chapters, we present contributions for the later stages of this pipeline, focusing specifically on attribute domain management and querying.

CHAPTER

4

INDEXING AND QUERYING ATTRIBUTE DOMAINS

An RDBMS organize data into relations based on a schema. Such relations are commonly handled as tables, which in turn consist of a set of columns, each one representing an attribute whose type is defined in the schema. To optimize the execution of queries over tables, RDBMSs employ well-known data structures called indexes. This set of techniques grounded in the Relational Model (CODD, 1970) suits numerous applications and has been successful since its proposal in the 1970s.

Nevertheless, there is still room for improvement in relational techniques, such as index structures in RDBMSs, which are of particular interest in this thesis. Specifically, index structures are some of the optimization resources implemented by RDBMSs (ELMASRI; NAVATHE, 2015), and there are many types of indexes, each of them targeting certain classes of queries and data types. In this thesis, we particularly focus on queries that require searching through more than one index, defined on a single column type, across multiple tables. Such queries are referred to as *Domain Queries* throughout this thesis, as they essentially consist of searching through the *active domain* (CODD, 1970) of a subset of attributes that span multiple relations, but nevertheless play a related role, hence composing a single *Attribute Domain*. The usual approach for Domain Queries is to search each index individually and then combine the results. However, these queries could have a higher performance with a method for searching columns from multiple tables in a single index seek. To the best of our knowledge, despite the numerous literature on database indexes, no work has proposed such a method yet.

In this context, this chapter presents the *Domain Index*, a new category of indexes for RDBMSs conceived as part of this thesis. Although sharing the same name, the concept of our proposal does not correspond to the homonymous one existing in the Oracle Database[1]. Rather, our proposal consists of a category of indexes that allows searching columns of a same type,

[1] <https://docs.oracle.com/database/121/ADDCI/dom_idx.htm>

across multiple tables, within a single index structure. Our proposal differs from the indexes of modern RDBMSs with respect to the number of tables an index can be associated with. Modern RDBMSs share the characteristic of defining an index on a column or a concatenation of columns from a *single* table, except for the *Clustered Index* (URBANO, 2016), which although defined on two tables targets another purpose, as described in Chapter 2. A Domain Index as we define here, on the other hand, can be created on columns from *multiple* tables.

The hypothesis pursued in this chapter is that **Domain Indexes, compared to existing resources of modern RDBMSs, improve the performance of Domain Queries**. Some of the applications benefited from the proposed Domain Indexes include:

- **Network intrusion detection.** Consider a database design in which Internet Protocol (IP) flows are stored in separate tables according to their protocol type. To identify, for instance, the source IP addresses that generated the highest network traffic regardless of protocol types, a query could employ a Domain Index defined on the source IP address domain, instead of searching one separate index for each table of IP flows.
- **Content-based image retrieval.** Several domains deal with complex data such as images. In this context, users can pose content-based queries across multiple related datasets to carry out analyses. For instance, radiologists could benefit from a Domain Index when querying images by content from distinct sets of imaging exams, expressed by feature vectors describing the images' visual content.
- **Sales analyses.** Consider a database of retailers with online and physical stores, in which their activity histories are stored in separate tables, but with attributes that share a common role. Then, sales reports can be created by querying attributes with financial or temporal data across tables such as *web sales*, *physical sales*, *web returns*, and *physical returns*. In this hypothetical situation, we could have one Domain Index over the financial attributes and another Domain Index over the temporal attributes across the four tables. Then, the Domain Indexes could be employed to execute queries performing a single index seek, as opposed to employing four index structures, one per separate table, which would incur one index seek per separate index.

In this chapter, we formally define Domain Indexes and show that this novel category of index structures enables the higher performance claimed by our hypothesis. Specifically, we describe how Domain Indexes are employed both for conventional and content-based queries. Furthermore, to analyze the performance gain provided by Domain Indexes, we carried out experiments over real and synthetic data. The experiments consisted of running queries that search multiple tables/repositories, both through the Common approach (searching each index individually and combining the results) and the proposed Domain Index approach. The experiments show the higher performance enabled by our proposal, whose performance gains reached around 66% in conventional queries and 71% in content-based queries.

The remainder of the chapter is organized as follows. Section 4.1 describes the proposed Domain Index. Section 4.2 shows the results of our experiments. Finally, Section 4.3 wraps up the chapter discussing the benefits enabled by the Domain Index, as well as how it stands out among existing related work.

4.1 The Domain Index

The Relational Model does not impose constraints on how indexes should be implemented in RDBMSs. Therefore, the implementation of indexes may vary depending on the manufacturer, and we claim that a single database index could have the option to be associated with more than one table. Accordingly, our proposal aims at extending current relational technologies by enabling index creation in a more flexible manner, providing a new category of indexes for modern RDBMSs that improve the performance of Domain Queries.

Definition. *A Domain Index is an index created on columns of the same type, regardless of which table(s) these columns belong to. It allows searching efficiently the active domain in a set of attributes, even if they are from different relations.*

The definition of Domain Indexes is based on the concept of *active domain*. This concept, which was proposed along with the Relational Model, denotes "the set of values represented at some instant" (CODD, 1970). Such set of values may refer to parts of a database at any granularity level. For instance, we could use the term *active domain* while referring to the "active" values (the values stored in the database) within the scope of **(i)** a single attribute, **(ii)** a set of attributes within the same relation, as well as **(iii)** a set of attributes across multiple relations. In the context of Domain Indexes, the active domain consists of the values from a set of attributes usually across distinct relations, i.e. values from indexed columns spanning multiple tables.

Additionally, it is possible to create a Domain Index on multiple columns of the same table, given that the columns are of the same type. In such a hypothetical situation, in which a Domain Index would be created on more than one column from the same table, one should note that such a Domain Index would *not* be equivalent to an ordinary multi-column index. For example, consider a scenario in which the columns A_i and B_i from a table T_i are indexed. A multi-column index would be primarily sorted by the values of A_i and, for each repeated value of A_i, the index would be sorted by the values of B_i. Conversely, a Domain Index would keep track of the column storing each indexed value, and the values from A_i and B_i could be intertwined, organized as if they were from the same column.

4.1.1 Theoretical Foundation

Domain Indexes aim at speeding up Domain Queries. The rationale for such improved efficiency is that scanning one bigger index, created on the active domain of columns spanning distinct tables, is faster than scanning multiple indexes, each one created on a separate table.

Embedding more than one data collection, from the same domain, in an integrated search space allows performing specific optimizations. This is particularly true in content-based retrieval. For instance, typical kNN algorithms use a dynamic radius during query execution for narrowing down the search space. The search is guided by the current set of k-nearest neighbors, which is updated when the query finds elements closer to the query element than the current k-th one. By having a set of separate indexes over complex data, each index is employed to execute the kNN query and the (partial) results need to be merged into the final answer. On the other hand, employing a single Domain Index allows the query response to be built directly from a single index structure. This allows reducing the dynamic radius faster than by using several indexes, as it provides a global view of the active domain.

This rationale is also extensible to other data domains. Consider a database having tables which are usually queried together. Let N be the number of these tables, each of them containing r_1, r_2, ..., r_N rows. The total number of rows is R, defined by $r_1 + r_2 + \ldots + r_N$. Moreover, consider that each table T_i, $1 \leq i \leq N$, contains an index defined on one of its columns, which we represent as A_i, that is, the column A that belongs to table T_i. These columns are of the same type and the indexes defined on them are B-trees of order b. Then, if such an index searches for a value in column A_i, the corresponding query will have a complexity of $\log_b(r_i)$ (COMER, 1979).

In the situation in which there is one index per table, consider a query over tables T_1, T_2, ..., T_N that searches the indexed column of each table. Such query, according to the Common approach available in modern RDBMSs, has a complexity $\mathcal{C}_c$ defined as follows:

$$\mathcal{C}_c = \log_b(r_1) + \log_b(r_2) + \ldots + \log_b(r_N) \tag{4.1}$$

$$\mathcal{C}_c = \sum_{i=1}^{N} \log_b(r_i). \tag{4.2}$$

Based on the property which states that the sum of the logs is the log of a product ($\log_b(x) + \log_b(y) = \log_b(x \cdot y)$):

$$\mathcal{C}_c = \log_b(r_1 \cdot r_2 \cdot \ldots \cdot r_N) = \log_b\left(\prod_{i=1}^{N} r_i\right). \tag{4.3}$$

On the other hand, considering a situation in which there is a Domain Index defined on attributes A_1, A_2, ..., A_i, the same query now has the complexity $\mathcal{C}_{DI}$ defined as follows:

$$\mathcal{C}_{DI} = \log_b(r_1 + r_2 + \ldots + r_N) = \log_b\left(\sum_{i=1}^{N} r_i\right). \tag{4.4}$$

When comparing Equations 4.3 and 4.4, we can safely state:

$$\sum_{i=1}^{N} r_i \leq \prod_{i=1}^{N} r_i, \text{where } r_i \in \mathbb{N} \text{ and } r_i \geq 2. \tag{4.5}$$

Accordingly, $\mathcal{C}_{DI} \leq \mathcal{C}_c$, which points out that the Domain Index approach is potentially more efficient than the Common approach in Domain Queries.

In a practical scenario, the performance of a Domain Index is equivalent to the one of an ordinary individual index on the concatenation of the separate tables being accessed. However, the latter approach incurs data redundancy, whereas a Domain Index is designed to support the indexing of multiple tables in a single data structure.

4.1.2 Domain Indexes in Content-Based Queries

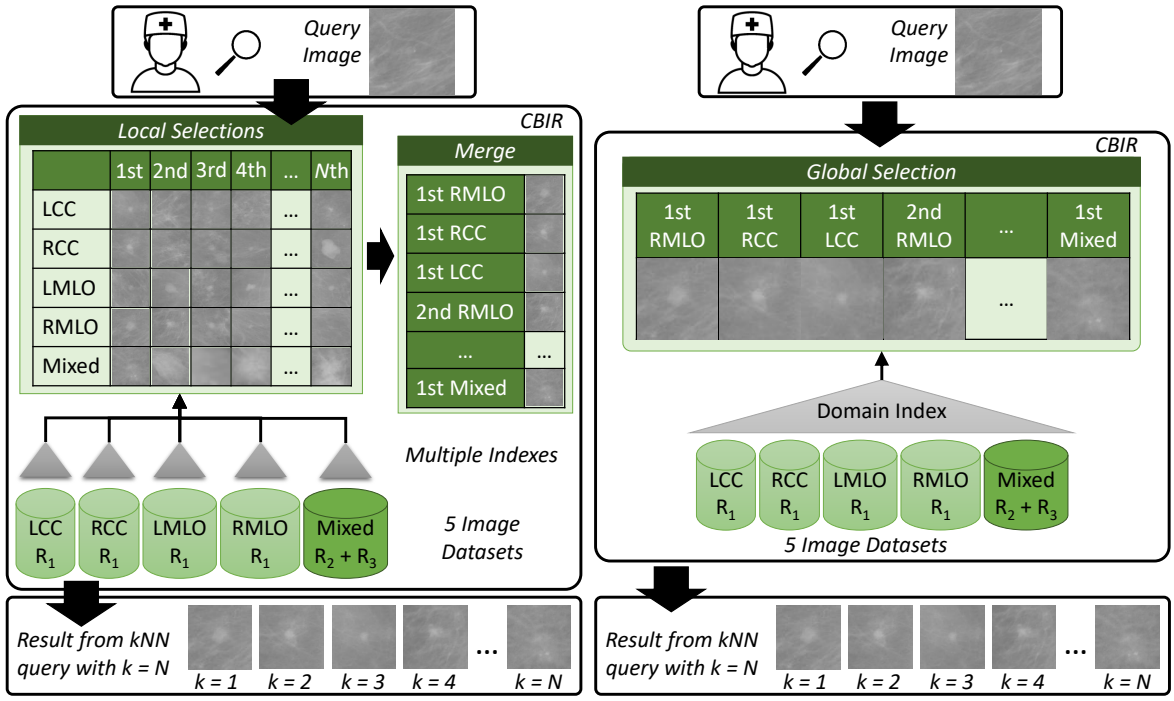
A Domain Index can also be employed as an indexing mechanism of CBIR systems. Even though data of CBIR systems are not always stored in an underlying DBMS, the idea of indexing an active domain across distinct repositories remains valid. Just like a table consists of a set of columns in an RDBMS, a complex dataset in a CBIR system contains a set of attributes, expressed according to the protocol employed by the CBIR system. Based on this notion, considering a scenario involving image datasets, a Domain Index is capable of indexing an active domain of images stored across multiple data repositories.

In order to properly work, the underlying index structure of a Domain Index must be compatible with the data type being handled. With respect to complex data, any content-based index structure can be employed, such as a MAM. However, every dataset to be indexed must have had the same set of features extracted, so that the distance function being employed by the Domain Index is able to compare any pair of elements.

To search multiple indexed complex datasets, the Common approach performs one index seek per index structure defined on its corresponding dataset. Then, a final step is performed to build the final response by merging the separate results obtained from the multiple index structures. Such final step is referred to as the *merge step*. On the other hand, to perform this task using the Domain Index approach, only one index seek must be performed, and there is no merge step: the answer is built directly within the data structure implementing the Domain Index. Both *k*NN and Range queries can be benefited from our proposal, as the Domain Index expands the search space of a Domain Query by providing a global view of the active domain, yet without compromising execution time. The benefit of employing a Domain Index is even greater for *k*NN queries. Such behavior is because the separate executions of a *k*NN query in the Common approach are more costly compared to the ones of a Range query, since a *k*NN query requires dealing with a higher amount of retrieved data.

As an example, Figure 18 illustrates the Common and Domain Index approaches in a medical scenario, showing how both approaches run a *k*NN query with $k = N$ across multiple medical image repositories. In the figure, five mammographic datasets are employed: **(i)** Left CranioCaudal (LCC), **(ii)** Right CranioCaudal (RCC), **(iii)** Left MedioLateral Oblique (LMLO), and **(iv)** Right MedioLateral Oblique (RMLO), originated from one data repository R_1; and **(v)** a dataset called Mixed, having data from two data repositories R_2 and R_3. The first four datasets have images categorized by mammographic view, and the fifth dataset has images with mixed mammographic views.

Figure 18 – Common and Domain Index approaches in a *k*NN query over medical image repositories. A Domain Index employs fewer steps to achieve the same result when querying multiple datasets (LCC, RCC, LMLO, RMLO, and Mixed) across repositories R_1, R_2, and R_3.



(a) Common approach. (b) Domain Index approach.

Source: Oliveira *et al.* (2019).

In Figure 18a, a physician uses a CBIR system to pose a *k*NN query employing multiple indexes, each of them defined on its respective dataset. After performing all index seeks, which correspond to each *k*NN query on its respective dataset, the CBIR system handles $5 \times N$ images, i.e. N from each dataset (*Local Selections* in the figure). Then, the CBIR system starts the merge step (*Merge* in the figure), which globally chooses the N most similar images within the set of $5 \times N$ locally selected ones. In the figure, the merge step chooses at least 2 images from RMLO, 1 from RCC, 1 from LCC, and 1 from Mixed. At the end of the query, after all indexes were searched, the N images are returned.

The Domain Index approach is presented in Figure 18b. In this case, the *k*NN query employs one Domain Index over the active domain of medical images, stored across the five image datasets. After performing a single index seek to run the *k*NN query, the CBIR system handles N images globally selected (*Global Selection* in the figure) rather than the $5 \times N$ images from Figure 18a. Then, the CBIR system returns the N most similar images.

Differently from *k*NN queries, which end up retrieving extra images via the Common approach (as shown in Figure 18a), Range queries do not incur any data overhead during query execution. That is, for both Common and Domain Index approaches, only the elements within the query radius are selected. Nevertheless, performing a single index seek over a bigger data

structure can be faster than sequentially seeking smaller indexes, one for each dataset. Hence, Range queries can still benefit from Domain Indexes as well.

It is important to mention that we do not take into account queries performed in parallel. Doing so would require another set of analyses, which are out of our scope in this thesis. It is not accurate to affirm that one bigger, serial index seek would be faster than multiple smaller index seeks performed in parallel. However, both Common and Domain Index approaches are parallelizable. Furthermore, it is worth noting that, even in the context of parallel *k*NN queries, the merge step would still be a bottleneck in the Common approach. Parallel Range queries, on the other hand, would face the same challenge as their serial counterpart, i.e. dealing with more (or less) data than needed due to potentially inaccurate radius values. Hence, regardless of running Range queries serially or in parallel, tuning is usually required to enable proper results. Such tuning is avoided in *k*NN queries, which can ease physicians' work.

Another important discussion is the impact of data bias on the performance of both Common and Domain Index approaches. If the elements are skewed, the indexes in the Common approach will have varied sizes on disk, also possibly having varied heights when considering hierarchical indexes, which impacts the number of disk accesses in queries. In such a scenario, some queries might need to access smaller parts of certain indexes, potentially leading to a superior performance than in the case of indexes with similar amounts of elements. Therefore, such evaluation is important when comparing the Domain Index to the Common approach, which is why we employ a real-world dataset with such characteristics in our experimental analysis.

4.2 Experiments

We evaluated the proposed Domain Index against the Common approach in conventional and content-based Domain Queries. The evaluation in conventional Domain Queries comprehended a real-world scenario of network intrusion detection based on a dataset of IP traffic flows, as well as a scalability evaluation over synthetic data. On the other hand, the evaluation in content-based Domain Queries comprehended a real-world scenario of CBIR, as well as a scalability evaluation over synthetic datasets.

We carried out most of the experiments on top of two modern RDBMSs. To employ the Common approach, considering attribute domains composed of multiple columns across distinct tables, we built an index on the corresponding column of each of those tables. The Domain Index approach, on the other hand, was emulated with a single index built on a table concatenating the originally separate tables. The index was created on the column storing the active domain to be queried, allowing to search the attribute domain of interest in one index seek. Such experimental setup enabled the comparison of both approaches in the same environment of a modern RDBMS.

The scalability experiments in content-based Domain Queries were carried out using standalone MAMs. We built a MAM on each separate dataset to employ the Common approach,

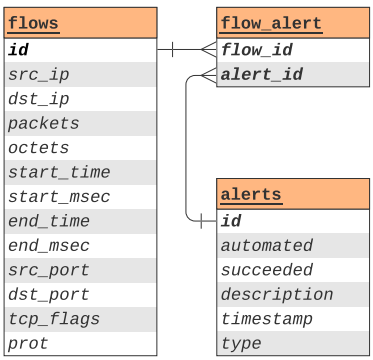
whereas a single MAM was built on the full dataset, which was the concatenation of all separate datasets into a single one, to employ the Domain Index approach. This setup allowed evaluating both approaches in an environment where there is no DBMS underlying the CBIR system.

We created typical queries for the considered use cases and executed them multiple times, both through the Common and Domain Index approaches. When employing an RDBMS, before each run, we restarted the RDBMS service and cleared the cache of the operating system. When employing standalone MAMs, we cleared the operating system cache as well. After executing the queries, we calculated a trimmed mean, removing 10% of the largest and 10% of the smallest values in order to get rid of outliers.

4.2.1 Real-World Use Case #1: Network Intrusion Detection

To evaluate the Common and Domain Index approaches over scalar data, we employed a flow-based intrusion detection dataset consisting of IP flows[2] data (SPEROTTO *et al.*, 2009). This dataset has IP flows of three protocol types: Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). The dataset was collected in six days, resulting in 14.2 millions of IP flows containing attributes such as protocol type, source IP address, source port, destination IP address, destination port, as well as start and end times. Also, there are 7.6 millions of alerts associated with most of the IP flows, describing security incidents and affected network services. Figure 19 shows the dataset schema.

Figure 19 – Original tables from the IP flow dataset.



Source: Adapted from Oliveira *et al.* (2017).

Assuming a real scenario such as the one in this set of experiments, it is reasonable to subdivide the *flows* table into three tables based on the protocol type (ICMP, TCP, and UDP). For instance, considering a hypothetical situation, TCP flows might contain an attribute representing the Round-Trip Time (RTT), which is a numeric attribute representing the time for a packet to travel from the client to the server and back to the client again (KUROSE; ROSS, 2016). Moreover, ICMP flows might include particular attributes as well, such as *type* and *code*. The

[2] An IP flow refers to all network traffic within a certain context: <https://tools.ietf.org/html/rfc7011>

values of such attributes could be used to further explain the message described by the IP flow (KUROSE; ROSS, 2016). Therefore, in this context, we subdivided the IP flows into three disjoint tables based on their protocol (the value in the attribute *prot*), namely *flows_tcp*, *flows_udp*, and *flows_icmp*, which have the same schema (as shown in Figure 19) and whose union is equivalent to the original *flows* table. This information is summarized in Table 3.

Table 3 – Tables used in the experiments over the IP flow dataset.

Table name	*flows*	*flows_tcp*	*flows_udp*	*flows_icmp*
Number of rows	14,170,132	14,151,511	583	18,038

Source: Oliveira *et al.* (2017).

For these experiments, we created three queries that are applicable to real scenarios, and the selection predicates of such queries are defined on different columns. We describe the queries as follows. Query 1 and Query 3 have predicates defined on the column *start_time*, which corresponds to the start time of an IP flow represented in Unix time, whereas the predicate of Query 2 is defined on the column *dst_ip*, which corresponds to the anonymized destination IP of an IP flow. We instantiated B-tree indexes over these columns both on the separate tables (for the Common approach) and on the full *flows* table (for the Domain Index approach). The predicates employed by Queries 1, 2, and 3 have the selectivity ratios of 0.005%, 0.117%, and 32.349% respectively, which are the ratios of retrieved rows over the total rows. Hence, in this work, a lower percentage means a higher selectivity, since less retrieved rows reflect a predicate that was able to filter out more rows. The queries are presented as follows.

Query 1. *Which source IPs and protocol types caused an unexpected high network traffic, namely more than 500 flows, in a period of 5 seconds (12:40:05 to 12:40:10 on 09/23/2008)?*

For the Domain Index approach, the SQL query is:

```
SELECT prot, src_ip, COUNT(*) AS flows_count
FROM flows
WHERE start_time BETWEEN 1222173605 AND 1222173610
GROUP BY prot, src_ip
HAVING COUNT(*) > 500;
```

Conversely, for the Common approach, the SQL query is:

```
SELECT * FROM (
  SELECT "TCP" AS prot, src_ip, COUNT(*) AS flows_count
  FROM flows_tcp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
  GROUP BY src_ip
  UNION ALL
  SELECT "UDP" AS prot, src_ip, COUNT(*) AS flows_count
  FROM flows_udp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
  GROUP BY src_ip
  UNION ALL
```

```
  SELECT "ICMP" AS prot, src_ip, COUNT(*) AS flows_count
  FROM flows_icmp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
  GROUP BY src_ip
) all_flows
WHERE flows_count > 500;
```

Query 2. *Which source IPs generated the 20 longest connections with the anonymized IP 2450649021, among all protocol types, and how many packets were sent?*

For the Domain Index approach, the SQL query is:

```
SELECT prot, src_ip, packets, (end_time - start_time) AS duration
FROM flows
WHERE dst_ip = 2450649021
ORDER BY duration DESC
LIMIT 20;
```

Conversely, for the Common approach, the SQL query is:

```
SELECT * FROM (
  SELECT "TCP" AS prot, src_ip, packets, (end_time - start_time) AS duration
  FROM flows_tcp
  WHERE dst_ip = 2450649021
  UNION ALL
  SELECT "UDP" AS prot, src_ip, packets, (end_time - start_time) AS duration
  FROM flows_udp
  WHERE dst_ip = 2450649021
  UNION ALL
  SELECT "ICMP" AS prot, src_ip, packets, (end_time - start_time) AS duration
  FROM flows_icmp
  WHERE dst_ip = 2450649021
) all_flows
ORDER BY duration DESC
LIMIT 20;
```

Query 3. *Which destination IPs were part of flows that generated the 10 greatest quantity of alerts, among all protocol types, from 03:00 to 15:00 on 09/24/2008?*

For the Domain Index approach, the SQL query is:

```
SELECT dst_ip, COUNT(alerts.id) AS total_alerts
FROM flows
JOIN flow_alert ON flows.id = flow_alert.flow_id
JOIN alerts ON flow_alert.alert_id = alerts.id
WHERE start_time BETWEEN 1222225201 AND 1222268400
GROUP BY dst_ip
ORDER BY total_alerts DESC
LIMIT 10;
```

Conversely, for the Common approach, the SQL query is:

```
SELECT dst_ip, COUNT(alerts.id) AS total_alerts
FROM (
  SELECT * FROM flows_tcp
```

```
  WHERE start_time BETWEEN 1222225201 AND 1222268400
  UNION ALL
  SELECT * FROM flows_udp
  WHERE start_time BETWEEN 1222225201 AND 1222268400
  UNION ALL
  SELECT * FROM flows_icmp
  WHERE start_time BETWEEN 1222225201 AND 1222268400
) all_flows
JOIN flow_alert ON all_flows.id = flow_alert.flow_id
JOIN alerts ON flow_alert.alert_id = alerts.id
GROUP BY dst_ip
ORDER BY total_alerts DESC
LIMIT 10;
```

The experiments were run on the RDBMS PostgreSQL 9.5.4, executed on a machine equipped with an Intel® Core™ i7-2600 @ 3.40GHz processor, 8GB of DDR3 1333MHz RAM memory, SATA 6Gb/s 7200RPM hard disk and Linux operating system Fedora 24.

Table 4 – Average time results of the Common and Domain Index approaches over the IP flow dataset.

Query	Common time (s)	Domain Index time (s)	Gain
1	0.447	0.307	∼31%
2	1.660	1.316	∼21%
3	30.707	29.320	∼5%

Source: Adapted from Oliveira *et al.* (2017).

Table 4 shows that our approach enabled gains ranging from 5% to 31%. The highest gain was in Query 1, whereas the lowest one was in Query 3. Query 1 applies the most selective predicate, which enables the Domain Index approach to perform an index seek quite faster than the index seeks of the Common approach. Moreover, Query 1 has little overhead from operations other than the index seeks, as opposed to Query 3, whose execution time also involves costly `JOIN` operations with tables *flow_alerts* and *alerts* (see Figure 19).

4.2.2 Scalability Evaluation in Conventional Domain Queries

This section describes experiments carried out to evaluate the scalability behavior of Domain Indexes over scalar data. We created three variations of Query 1 and Query 2, which are the queries that enabled the best performance results in the previous section. They are defined as follows: **(i)** Q1A and Q2A are equal to Query 1 and Query 2; **(ii)** Q1B and Q2B maintain only `COUNT(*)` in the `SELECT` clause, as well as remove the other clauses such as `GROUP BY`; **(iii)** Q1C and Q2C are similar to Q1B and Q2B, however they project all columns in the `SELECT` clause rather than counting the returned rows. These queries were created to allow analyzing the impact of different amounts of overhead from operations besides the index seeks.

SQL Commands for Q1B, Q1C, Q2B, and Q2C

Q1B — Domain Index approach:

```
SELECT COUNT(*) AS flows_count FROM flows
WHERE start_time BETWEEN 1222173605 AND 1222173610;
```

Q1B — Common approach:

```
SELECT COUNT(*) AS flows_count
FROM (
  SELECT * FROM flows_tcp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
  UNION ALL
  SELECT * FROM flows_udp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
  UNION ALL
  SELECT * FROM flows_icmp
  WHERE start_time BETWEEN 1222173605 AND 1222173610
) all_flows;
```

Q1C — Domain Index approach:

```
SELECT * FROM flows
WHERE start_time BETWEEN 1222173605 AND 1222173610;
```

Q1C — Common approach:

```
SELECT * FROM flows_tcp
WHERE start_time BETWEEN 1222173605 AND 1222173610
UNION ALL
SELECT * FROM flows_udp
WHERE start_time BETWEEN 1222173605 AND 1222173610
UNION ALL
SELECT * FROM flows_icmp
WHERE start_time BETWEEN 1222173605 AND 1222173610;
```

Q2B — Domain Index approach:

```
SELECT COUNT(*) AS flows_count FROM flows WHERE dst_ip = 2450649021;
```

Q2B — Common approach:

```
SELECT COUNT(*) AS flows_count
FROM (
  SELECT * FROM flows_tcp WHERE dst_ip = 2450649021
  UNION ALL
  SELECT * FROM flows_udp WHERE dst_ip = 2450649021
  UNION ALL
  SELECT * FROM flows_icmp WHERE dst_ip = 2450649021
) all_flows;
```

Q2C — Domain Index approach:

```
SELECT * FROM flows WHERE dst_ip = 2450649021;
```

Q2C — Common approach:

```
SELECT * FROM flows_tcp WHERE dst_ip = 2450649021
UNION ALL
SELECT * FROM flows_udp WHERE dst_ip = 2450649021
UNION ALL
SELECT * FROM flows_icmp WHERE dst_ip = 2450649021;
```

Results

The queries Q1A–C and Q2A–C were employed in two experiments: **(i)** one varying the number of tables accessed by the Common approach, while fixing the total number of accessed rows; and **(ii)** one varying the cardinality of tables, while fixing their number. In both experiments, the sum of rows from all separate tables equals the sum of rows in the table accessed by the Domain Index approach.

In the experiments varying the number of tables, the first step was to replicate the original *flows* table twice, which generated a table with triple the original size (42,510,396 rows) and the same data distribution, called *augmented_flows* table. Then, the tables shown in Table 5 were created by equally dividing the rows from *augmented_flows* into 3, 4, 6, and 12 separate tables.

Table 5 – Tables of the experiments over the IP flow dataset varying the number of tables. The complete *augmented_flows* table has 42,510,396 rows.

Tables	3	4	6	12
Rows per table	14,170,132	10,627,599	7,085,066	3,542,533

Source: Oliveira *et al.* (2017).

Conversely, in the experiments varying the cardinality, the number of separate tables was fixed to three. Employing a Scale Factor of 1, each of the three tables contained the rows from the original *flows* table. Then, employing a Scale Factor of 2, each table contained twice the rows from the original *flows* table, and so on. These numbers are summarized in Table 6.

Table 6 – Tables of the experiments over the IP flow dataset varying the cardinality of tables, considering three separate tables for the Common approach.

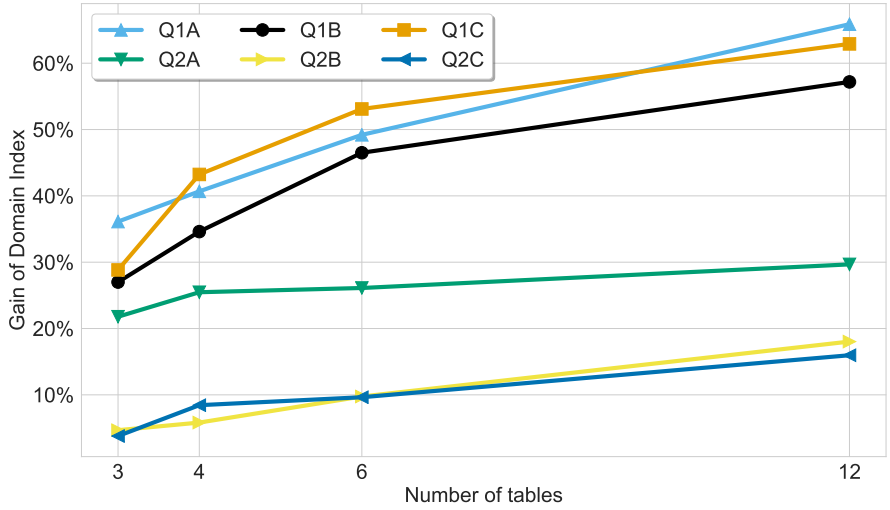
Scale Factor	**Rows per separate table**	**Total rows**
1	14,170,132	42,510,396
2	28,340,264	85,020,792
3	42,510,396	127,531,188
4	56,680,528	170,041,584

Source: Oliveira *et al.* (2017).

Figures 20 and 21 present the results from the experiments varying the number of tables. From Figure 20, we can see the gain increasing as the number of tables grows. The gain was

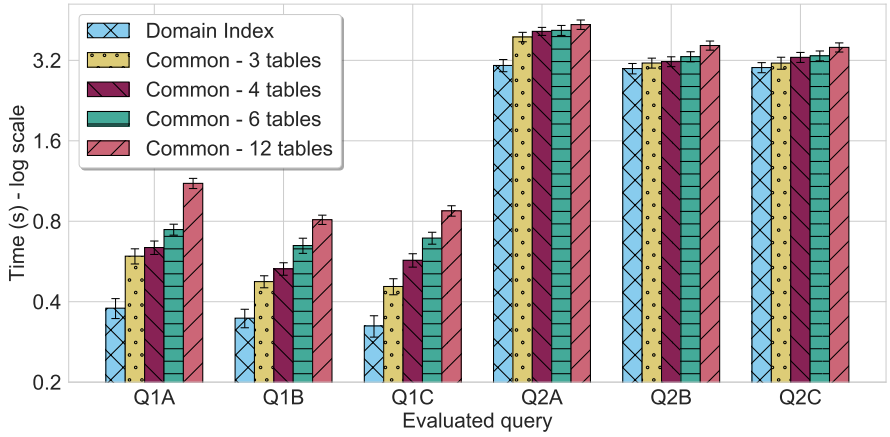
higher for Q1 than for Q2, like in the previous experiments. Moreover, the gain was higher for the original queries (Q1A and Q2A) than for the variations B and C. This is because the execution times tend to be smaller for the variations B and C in both approaches, but mainly in the Common one, due to the lower overhead of operations besides index seeks. Such behavior is better visualized in Figure 21, which presents the absolute execution times. In these experiments, the gains ranged from 27% to 66% for Q1 queries and from 4% to 30% for Q2 queries.

Figure 20 – Gains of Domain Index for scalability experiments in conventional Domain Queries, fixing the cardinality and varying the number of tables.



Source: Adapted from Oliveira *et al.* (2017).

Figure 21 – Query execution times for scalability experiments in conventional Domain Queries, fixing the cardinality and varying the number of tables.

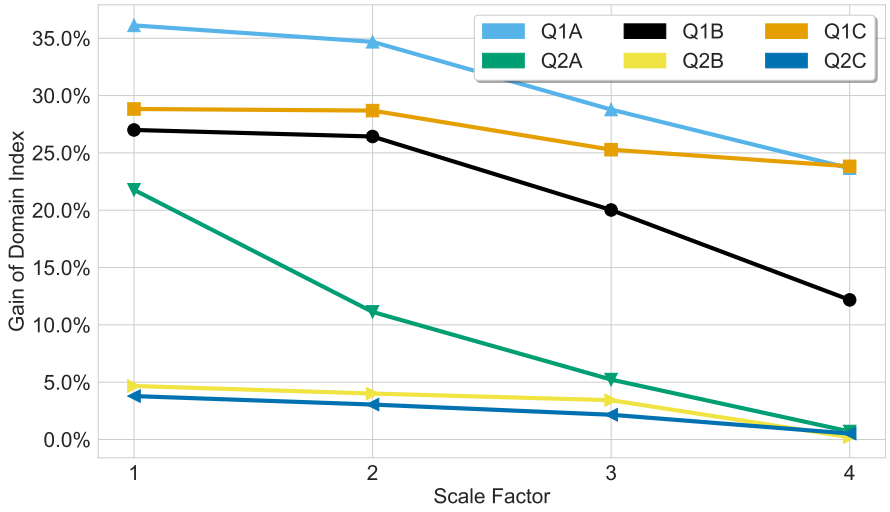


Source: Adapted from Oliveira *et al.* (2017).

Figures 22 and 23 present the results from the experiments varying the cardinality of tables. Figure 22 shows that the gain decreases as cardinality grows, which can be explained by the fact that higher cardinalities tend to degrade indexes in general. In the case of B-trees, for

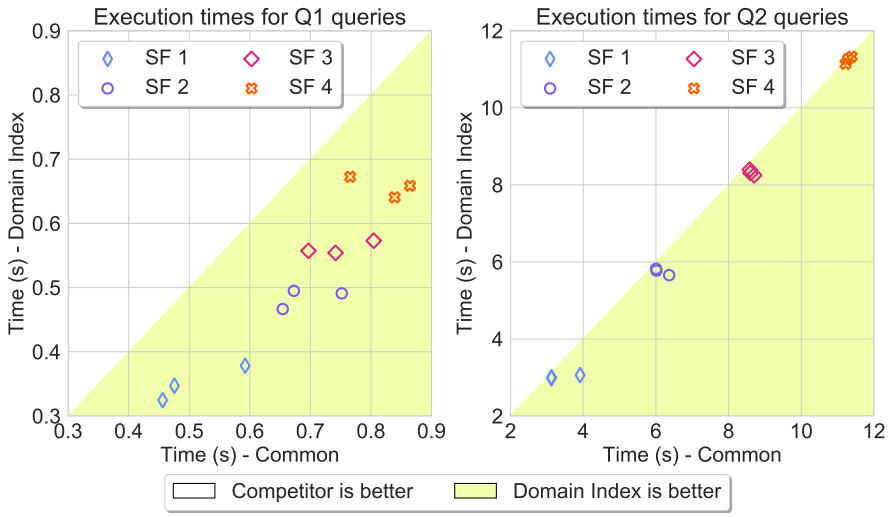
example, having more indexed elements leads to tree structures with more levels. Nevertheless, the gain is still positive and significant for all cardinalities, especially when considering Q1 queries, whose predicate has the highest selectivity. Figure 23 shows that all executions were faster with the Domain Index approach, as represented by all markers within the highlighted triangle, which delimits the area where our approach is better. In these experiments with varying cardinalities, the gains ranged from 12% to 36% for Q1 queries and up to 22% for Q2 queries.

Figure 22 – Gains of Domain Index for scalability experiments in conventional Domain Queries, fixing the number of tables and varying their cardinality.



Source: Adapted from Oliveira *et al.* (2017).

Figure 23 – Query execution times for scalability experiments in conventional Domain Queries, fixing the number of tables and varying their cardinality.



Source: Adapted from Oliveira *et al.* (2017).

4.2.3 Real-World Use Case #2: Content-Based Image Retrieval

To evaluate the performance of the Common and Domain Index approaches in content-based Domain Queries, we employed a dataset having 13,497 medical images in the Digital Imaging and Communications in Medicine (DICOM) format[3], collected from the Clinics Hospital of the University of São Paulo in Ribeirão Preto. Despite its low cardinality, this dataset occupies around 3.5GB on disk due to the storage of DICOM data. This dataset was loaded in varying quantities of complementary tables, each storing a part of the dataset. Each table row has an image itself and its 256-bin color histogram, both represented as byte arrays, along with associated metadata. Slim-trees were created on the columns storing the histograms.

We employed both evaluated approaches in *k*NN queries that were executed multiple times, each time with a random query element. We used the MedFMI-SiR module (KASTER *et al.*, 2011), designed for running content-based queries with user-defined index structures on the RDBMS Oracle, to employ the MAM Slim-tree. We employed the Manhattan Distance function and executed the *k*NN query using the following values of k: 1, 5, 10, 50, 100, and 200. Finally, we carried out this whole process for 2, 3, and 4 separate tables for Common.

For illustration purposes, the SQL command for a 5NN query in the MedFMI-SiR module is expressed as follows:

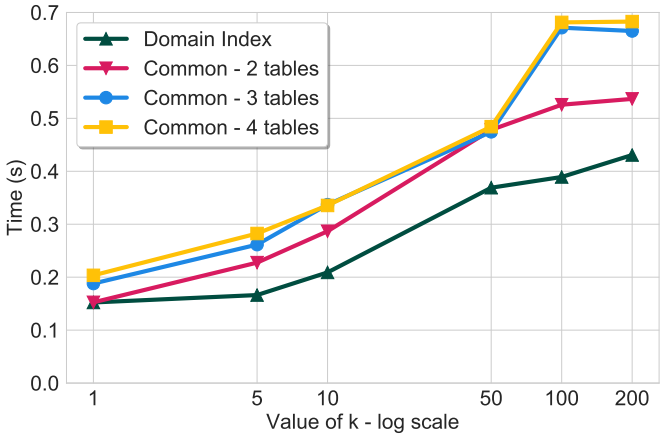
```
SELECT image_id FROM medical_images
WHERE manhattan_knn(image_column, query_element) <= 5;
```

These experiments were run on the RDBMS Oracle 11g Release 2. We used a machine equipped with an Intel® Core™ i5-2400 @ 3.10GHz processor, 4GB of DDR3 1333MHz RAM memory, SATA 6Gb/s 7200RPM hard disk, and Linux operating system Ubuntu 12.04.5.

Figure 24 depicts the results for this set of experiments. From the figure, we can see that the performance of the Domain Index approach is higher compared to the Common approach for 2, 3, and 4 separate tables. Furthermore, such higher performance can be seen for all values of k. Considering $k = 1$, which is a point query in the context of content-based retrieval, the gain ranged from 0.2% to 25%. The tie regarding the gain of 0.2% occurred between the Domain Index and the Common approach using 2 tables, and the combination of two main reasons led to such result. The first reason is that the 2-table variant of the Common approach is faster than the other variants, due to performing less index seeks in queries. The second reason is that keeping track of only the top candidate in the result set of a 1NN query, compared to a greater number of candidates for the other values of k, leads to a smaller overhead in queries, not to mention that a 1NN query does not require sorting the result set. Such lack of overhead further reduces the execution time of the Common approach with 2 tables in 1NN queries, hence the tie with the Domain Index. For the other values of k, our proposal enabled gains from 20% to 43%. Hence, the overall gain of the Domain Index is positive and significant for content-based queries.

[3] <http://dicom.nema.org/>

Figure 24 – *k*NN query execution times for the Common approach using distinct setups and for the Domain Index approach, varying the value of *k*.



Source: Oliveira *et al.* (2017).

4.2.4 Scalability Evaluation in Content-Based Domain Queries

We evaluated the scalability of our proposal over synthetic data, created by employing a data generator from the literature (CIACCIA; PATELLA; ZEZULA, 1997). The data elements consisted of 64-dimension feature vectors. For the Common approach, we generated multiple repositories of data. The number of elements in each repository was the same, but the elements were randomly generated for each repository by providing the data generator with distinct seeds, and the data of each repository followed a Gaussian distribution.

Both approaches were implemented on top of the Arboretum C++ library[4], developed and maintained by the Databases and Images Group at the University of São Paulo. This library has built-in feature extractors, distance functions, and MAMs, as well as provides an interface to implement new MAMs. In this set of experiments, we employed the Slim-tree as the MAM to implement both approaches. For the Common approach, we built multiple Slim-trees, one per repository. For the Domain Index approach, one Slim-tree indexed the entire active domain across the repositories. The distance function employed was the Canberra Distance.

We executed both *k*NN and Range queries. The following values of *k* were used in the *k*NN queries: 5, 10, 25, 50, 100, and 200. Moreover, we tuned the query radius parameter in the Range queries to retrieve the same number of elements as each *k*NN query. We also analyzed the average building time and disk space usage of the indexes. We carried out two evaluations:

1. **Varying cardinality.** We set the number of repositories to 5 and varied the cardinality of the repositories; and

2. **Varying number of repositories.** The cardinality was fixed to 1,000,000 elements and we varied the number of repositories. We employed the data generator for each separate

[4] <https://bitbucket.org/gbdi/arboretum>

repository, passing the same parameters, so that the generated data followed a Gaussian distribution with the same mean and standard deviation. Each separate repository had the same number of elements, and summing all of them always totalled 1,000,000.

We used a computer equipped with an Intel® Xeon® CPU X5650, DDR3 1333MHz RAM of 32GB, and Fedora 23.

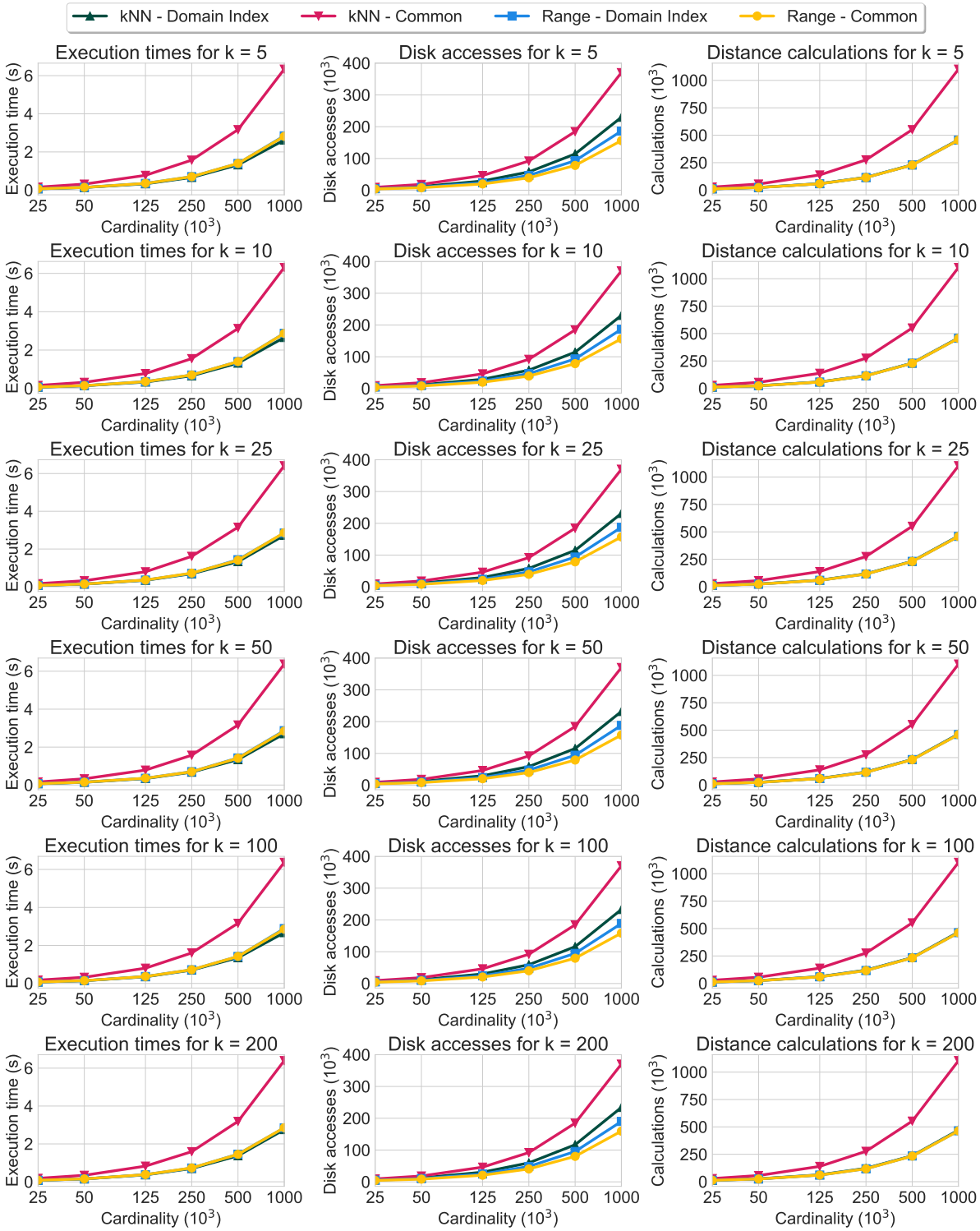
Results — Varying Cardinality

Figure 25 shows the results with respect to execution time, number of disk accesses, and number of distance calculations. Considering *k*NN queries, our gains in query execution time ranged from 56% to 59% for all values of k. Interestingly, by analyzing the curves in the plots across different values of k, we can see that the overall number of disk accesses required by the Common approach was high and stable for *k*NN queries, and this can be explained by two main reasons: **(i)** the high overlap rate in the MAMs, which leads the queries to access more index blocks, and **(ii)** the multiple dynamic radii (one per separate index), which adapt according to the current set of candidates as these are continuously updated in the query. This is a drawback of the Common approach in *k*NN queries, as this approach lacks a global view of the active domain and therefore takes longer to adjust the query radii. Still inspecting the results from *k*NN queries, the same stable behavior can be seen for the number of distance calculations in the Common approach. This indicates that all index blocks ended up being accessed by this approach, which in turn led to a stable number of distance calculations.

Considering Range queries, on the other hand, both evaluated approaches presented an overall similar behavior. The Domain Index approach incurred a slightly greater number of disk accesses compared to the Common approach. This behavior can be explained by the potentially higher overlap rate between index blocks in the MAM implementing the Domain Index approach, due to a single Slim-tree being employed to index the whole active domain. Nevertheless, with respect to distance calculations and execution times, both evaluated approaches tied, with the Domain Index enabling around 2% of gain in execution time. As we see that the curves grow similarly for execution time and number of distance calculations, it becomes evident that the overall number of distance calculations had a major impact on the resulting execution times, which explains the tie between both approaches in Range queries despite the minor difference in the number of disk accesses.

Figure 26 presents the average building time of index structures for the Common and Domain Index approaches. The Common approach took less time to build all separate indexes in most of the cases, being 7%–12% faster. The exception was for the cardinality value of 125,000, for which the Domain Index was around 5% faster. The Common approach was overall faster in the construction phase because, in the context of hierarchical index structures, a Domain Index is likely to have at least one more level, which requires at least one more disk access when traversing the index from the root to the leaf node into which the element will be inserted.

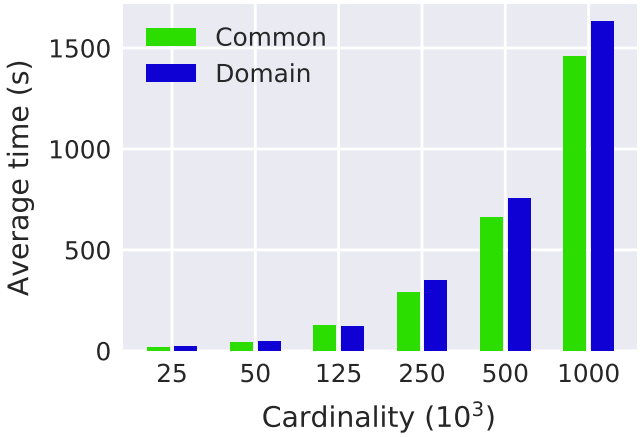
Figure 25 – Results of scalability experiments in content-based Domain Queries, varying the cardinality of repositories and fixing the number of repositories to 5.



Source: Elaborated by the author.

As presented in Table 7, the Domain Index was always one level taller than the indexes employed by the Common approach. Therefore, descending the tree to insert each element required one more disk access, which in turn made the whole process a bit slower. However, as

Figure 26 – Index building times for scalability experiments in content-based Domain Queries, varying the cardinality of repositories and fixing their number.



Source: Oliveira *et al.* (2019).

Table 7 – Disk space usage of index structures in both approaches as cardinality scales.

Cardinality	Method	Blocks	Height	Size (MB)
25,000	Common	9,093	6 (each index)	35.54
	Domain Index	**8,900**	**7**	**34.77**
50,000	Common	18,387	7 (each index)	71.84
	Domain Index	**17,785**	**8**	**69.48**
125,000	Common	46,220	7 (each index)	180.57
	Domain Index	**44,585**	**8**	**174.16**
250,000	Common	92,028	8 (each index)	359.50
	Domain Index	**90,708**	**9**	**354.33**
500,000	Common	184,591	8 (each index)	721.08
	Domain Index	**181,517**	**9**	**709.05**
1,000,000	Common	370,255	9 (each index)	1446.33
	Domain Index	**365,120**	**10**	**1426.25**

Source: Oliveira *et al.* (2019).

we can see in the previous results with respect to query time, such minor overhead in the building time makes up for an overall higher performance in queries. Furthermore, index construction tends to happen less often (usually once, if the index is never rebuilt) than queries, which on the other hand are likely to happen more often in practice. Also in Table 7, we can see the index sizes considering the number of disk blocks and the overall disk space being used. The table shows that our method used less disk space than the Common approach. This is explained by the smaller number of blocks allocated to index the whole dataset via the Domain Index approach, which in turn means that the blocks ended up having a higher occupation ratio. Conversely, the Common approach employed indexes having more blocks with a lower occupation ratio.

Results — Varying Number of Repositories

Figure 27 presents the results with respect to execution time, disk accesses and distance calculations. In *k*NN queries, the Domain Index enabled gains in execution time from 36% to 71%, also surpassing the performance of both approaches in Range queries. Considering each number of repositories, the gains were similar among all values of k. Furthermore, as the number of repositories grew, our gains increased as well. The gains are detailed in Table 8.

Table 8 – Detailed gains in *k*NN query execution time for each number of repositories.

Repositories	**Minimum Gain**	**Maximum Gain**
2	36%	37%
4	53%	55%
8	59%	62%
16	68%	69%
32	70%	71%

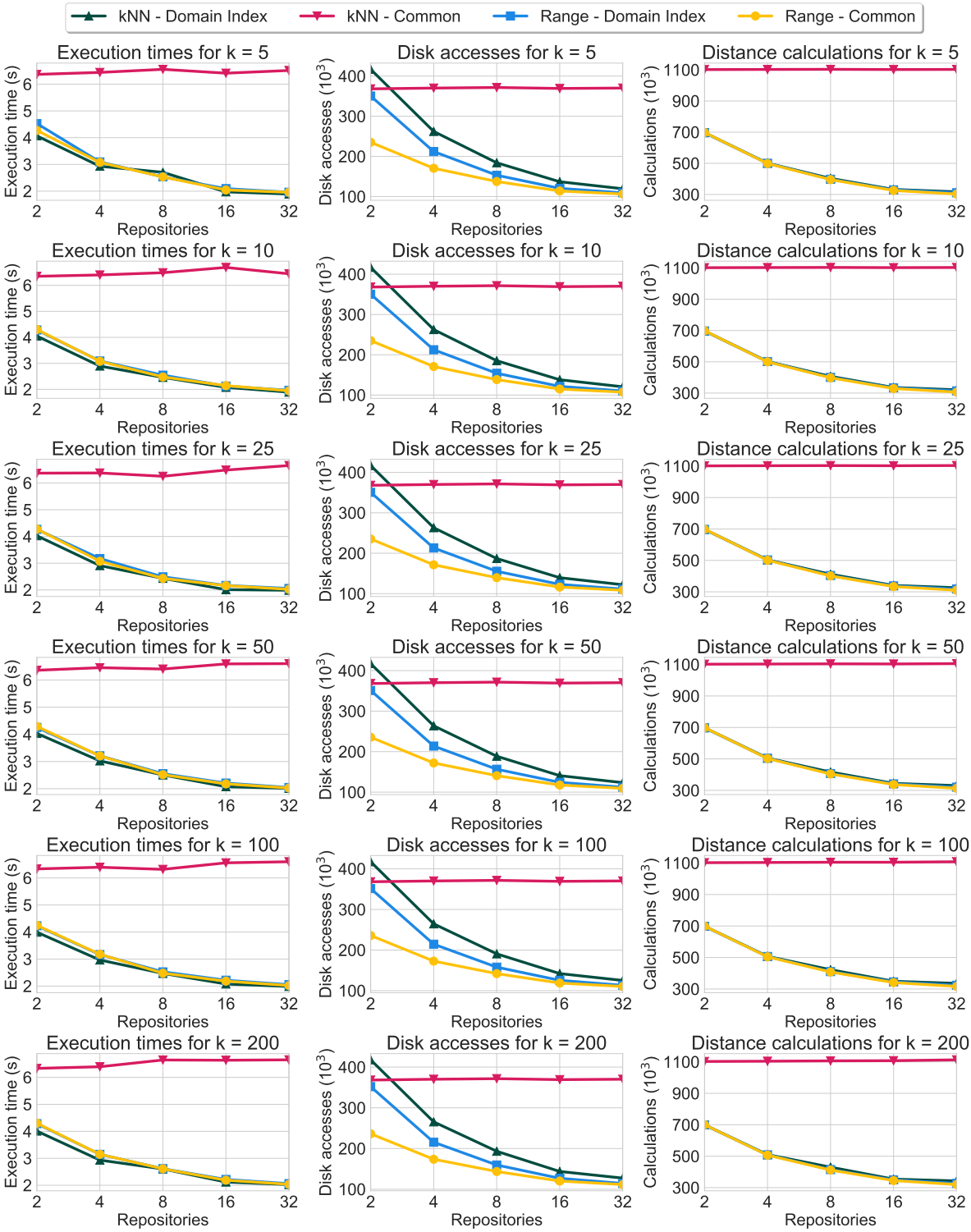
Source: Oliveira *et al.* (2019).

We can see in Figure 27 that the *k*NN queries in the Common approach took the longest time to run. This can be explained by the overhead of data in those queries, as previously shown in Figure 18, in addition to the lack of a global view of the indexed active domain, which tends to slow down the reduction of the dynamic query radii. With respect to the other combinations of evaluated approaches and queries, we see that the execution times are significantly shorter. As there is no data overhead in such queries, query time is mostly impacted by index searches.

Since the total number of elements was fixed, a bigger number of repositories implies a smaller quantity of elements within each repository. Moreover, an important detail to be noted is how the synthetic data were generated: each repository had elements following a Gaussian distribution, with a distinct seed having been provided to the data generator. Therefore, elements within the same repository are likely to be closer to each other in the metric space as the number of repositories increases. This is reflected in the resulting index structures, which end up having subtrees with more condensed element groups. In this context, a content-based Domain Query tends to retrieve fewer elements from each repository as the number of repositories increases, accessing smaller portions of the MAMs. This is why the average query time decreases along the x-axes in Figure 27 for all methods other than *k*NN queries in the Common approach.

Figure 27 also shows that, compared to execution time and number of distance calculations, there were more significant differences between the approaches with respect to disk accesses. This can be explained by the different overlap ratios in the index structures, which tend to be higher for the Domain Index approach, and by how the elements are spread across the metric space. These factors have a direct impact on what portions of the MAMs need to be accessed during queries, as previously discussed in the scalability experiments varying the cardinality. Still in Figure 27, we can see that the curves corresponding to distance calculations

Figure 27 – Results of scalability experiments in content-based Domain Queries, varying the number of repositories and fixing the total cardinality to 1,000,000.



Source: Elaborated by the author.

are similar to the ones corresponding to query time, which indicates that the amount of distance calculations had a major impact on the resulting execution time. Moreover, building the query result set directly in the index structure implementing a Domain Index also contributed to the

overall performance, leading to similar execution times among the Domain Index in *k*NN queries and both approaches in Range queries, despite the Domain Index having incurred more disk accesses in Range queries when compared to some setups of the Common approach.

Table 9 – Disk space usage of MAMs for the Common and Domain Index approaches in scalability experiments varying the number of repositories.

Repositories	Method	Blocks	Height	Size (MB)
2	Common	368,168	9 (each index)	1,438.16
	Domain Index	**364,353**	**10**	**1,423.26**
4	Common	370,129	9 (each index)	1,445.83
	Domain Index	**365,944**	**10**	**1,429.47**
8	Common	371,570	8–9 (each index)	1,451.4
	Domain Index	**363,522**	**10**	**1,420.01**
16	Common	369,183	8 (each index)	1,442.18
	Domain Index	**359,317**	**10**	**1,403.59**
32	Common	370,178	7–8 (each index)	1,446.13
	Domain Index	**353,327**	**10**	**1,380.19**

Source: Oliveira *et al.* (2019).

As presented in Table 9, the Domain Index always had the same height and required less disk blocks to be stored, as opposed to the Common approach which had smaller height values and occupied more disk blocks. Therefore, the Domain Index approach enabled slightly smaller index structures, having less blocks with a higher occupation ratio.

4.3 Discussion

We have proposed a novel category of indexes for improving performance in a class of queries that search attribute domains. The proposal aims at enhancing the current relational technologies used to build RDBMSs, and we also showed that the concept of Domain Index can be applied to scenarios employing standalone index structures in content-based retrieval. To the best of our knowledge, the Domain Index is the first approach towards defining an index on the active domain of columns spanning multiple tables — and analogously on the active domain of complex datasets spanning multiple repositories.

In addition to presenting the theoretical foundation for the Domain Index, we evaluated it with a comprehensive set of experiments, involving both real-world and synthetic datasets. The performance gains provided by Domain Indexes, which reached 66% in conventional Domain Queries and 71% in content-based Domain Queries, confirmed our hypothesis that “Domain Indexes, compared to the existing resources from modern RDBMSs, improve the performance of Domain Queries”. Furthermore, Domain Indexes are particularly useful in the context of CBIR.

CHAPTER

5

CONTENT-BASED DOMAIN QUERIES OVER MEDICAL DATA

The volume of imaging data in healthcare institutions has been increasing at a very fast pace. Such data are usually represented in diverse forms due to constant advances both in medical imaging software and equipment. The complexity of this modern scenario requires proper tools to manage the data.

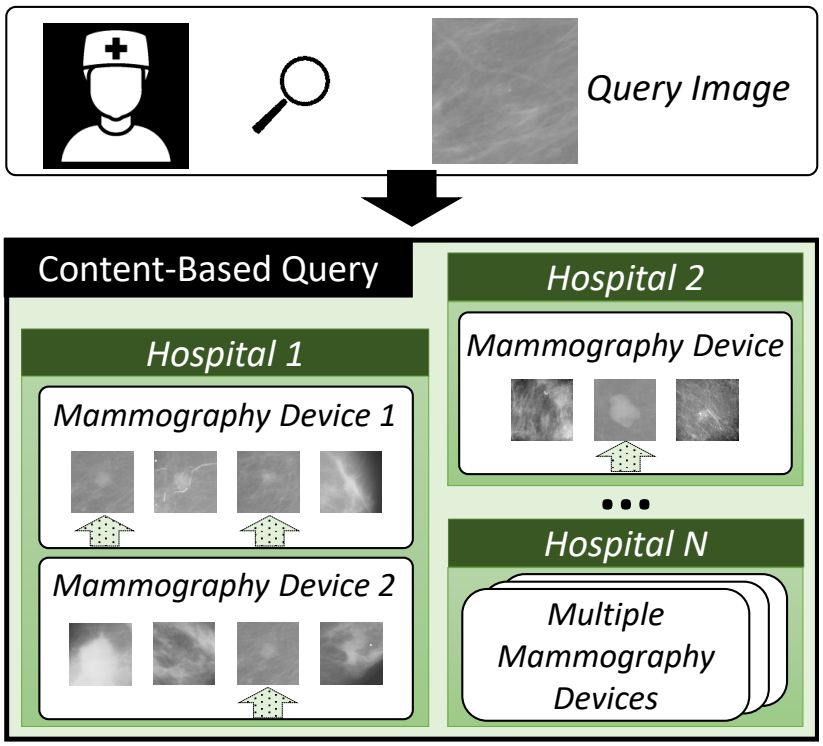
The consolidated PACS is a well-known class of medical imaging software. A PACS manages data acquired from patients, such as exam images, in order to organize image distribution in a healthcare environment. The system operates as an interface between the screening equipment and the workstations in which the data are analyzed. However, a PACS is not able to effectively support physicians in decision-making processes with content-based queries (LIU *et al.*, 2007).

Another class of medical imaging software is the CBIR system, which is able to handle large image collections as well as to aid in biomedical studies. A typical usage of CBIR systems in a clinical environment is for retrieving exam images according to similarity measures. Two capabilities of these systems allow them to be used in such a scenario: **(i)** the indexing of feature vectors extracted from images; and **(ii)** the retrieval of images by similarity based on the extracted features (LONG *et al.*, 2009).

CBIR systems can be employed in scenarios with the need for managing multiple medical data repositories (OLIVEIRA *et al.*, 2017). For example, considering mammograms, the images may have been generated by distinct devices. They may also have been obtained from distinct data repositories, hence potentially being expressed in diverse formats, as well as may present varied resolutions and heterogeneous metadata (MOREIRA; BACELAR-SILVA; RODRIGUES, 2012). Figure 28 shows such a scenario, in which images from mammograms have been gathered from multiple hospitals, each image being a Region of Interest (ROI) of the corresponding mammogram. In this example, the image features have been locally extracted at each site and finally integrated. In this context, a physician may want to search for similar past cases from such

data and use them to support the decision-making process. More specifically, the physician may provide a ROI to a content-based query and retrieve the most similar ROIs. This is illustrated in the figure, in which dotted arrows (in the boxes regarding each mammography device) point to the retrieved ROIs.

Figure 28 – Content-based query over ROIs of mammograms generated by distinct mammography devices from multiple hospitals.



Source: Oliveira *et al.* (2019).

In spite of the continuous progress in the development of medical CBIR systems, they lack a proper approach for performing Domain Queries over images across multiple repositories. Existing CBIR systems would need to employ the Common approach when performing Domain Queries, independently indexing each repository and searching them individually, then merging local results to compose the final query response (OLIVEIRA *et al.*, 2017; OLIVEIRA *et al.*, 2017a). As we discussed in the previous chapter, this approach can be time-consuming, especially when querying many repositories by similarity. Accordingly, we proposed a novel category of index structures, the Domain Index, which can index an active domain of complex data with a single structure, even if the indexed data came from separate repositories.

In this chapter, we explore an application of the Domain Index to a medical scenario, targeting the problem of querying an active domain of medical images across multiple repositories. Experimental results over real-world medical data show that the Domain Index approach enables a higher performance in this context. Furthermore, empowering a CBIR system with *Domain Indexes* can allow physicians to seamlessly retrieve medical images from distinct repositories, which in turn can help them in decision-making.

This chapter is organized as follows. Section 5.1 specifies the experimental setup. Section 5.2 describes the real-world data used in the experiments. Section 5.3 presents the results. Finally, Section 5.4 concludes the chapter.

5.1 Experimental Setup

Considering a real-world medical scenario, like the one in our experimental evaluation, in order for the Domain Index to be seamlessly employed, the data must have been previously gathered from distinct medical devices (and possibly distinct hospitals). Then, the data can be stored in a single local machine, either in the form of multiple datasets or directly in the leaf nodes of the MAM implementing the Domain Index.

We compared the performance of both Common and Domain Index approaches, which were implemented on top of the Arboretum library. In general, MAMs have been widely used for data retrieval in medical scenarios, so any MAM could be employed. As the Slim-tree has already been employed in the literature (DOHNAL; GENNARO; ZEZULA, 2008; OLIVEIRA *et al.*, 2016; NESSO-JR. *et al.*, 2018) and is implemented in the Arboretum library, this was the MAM used in the experiments. Specifically, we measured the average query time for *k*NN and Range queries, along with the average building time and disk space usage of the index structures, in the context of Domain Queries across multiple repositories. We used a computer equipped with an Intel® Xeon® CPU X5650, DDR3 1333MHz RAM of 32GB, and Fedora 23.

5.2 MAMMOSET

To evaluate our proposal in a real-world scenario, we employed three public mammogram repositories available in the literature, briefly detailed as follows.

1. The first repository, known as Digital Database for Screening Mammography (DDSM)[1] (HEATH *et al.*, 2001), consists of 2,892 annotated ROIs, cropped from more than three thousand original images. These ROIs are categorized into four datasets, according to the four mammographic views LCC, RCC, LMLO, and RMLO, as in the example depicted in Figure 18.

2. The VIENNA repository was created by the University of Vienna (TRAINA *et al.*, 2011) with images collected from the Breast Imaging-Reporting and Data System (BI-RADS) Tutorium, which was held at the same university. The data originated from this repository comprise 447 annotated ROIs of mammograms, which are not organized by mammographic view, but include other useful metadata.

[1] <http://marathon.csee.usf.edu/Mammography/Database.html>

3. The third repository, consisting of data from a mini mammographic repository provided by the Mammographic Image Analysis Society (MIAS), is called MINI-MIAS[2] (SUCKLING *et al.*, 1994) and contains 118 ROIs of mammograms. Similarly to the VIENNA repository, the ROIs are not separated by mammographic view, but include additional metadata.

The data from these repositories have been made available with the MAMMOSET dataset (OLIVEIRA *et al.*, 2017b). It standardizes the ROIs as thumbnails of 256×256 pixels, organizes the distinct metadata from the three repositories, as well as provides feature vectors generated by nine feature extractors for all ROIs. We were interested in evaluating our proposal against the Common approach in varied real-world scenarios of medical CBIR, and using MAMMOSET in our experiments made it possible, as this dataset contains distinct image features with varied dimensionalities and data collections of different cardinalities.

The data from the first repository, DDSM, consist of four datasets, one for each mammographic view, whereas the data from the other two repositories, VIENNA and MINI-MIAS, are integrated into a single dataset of mixed mammographic views. Hence, as there is a total of five datasets, five indexes were employed for the Common approach, one over each dataset, whereas one index was employed for the Domain Index approach. This organization allows the physicians to ask for different combinations of datasets to be evaluated, such as retrieving images from a subset of mammographic views (e.g. LCC + RCC) or images depicting the same side with mixed mammographic views (e.g. LCC + LMLO). However, in this thesis, we only report results from queries searching all the datasets.

Table 10 – Features extracted from the ROIs of MAMMOSET, their dimensionalities and the corresponding page sizes used to index them.

Feature Extractor	**Dimensions**	**Page Size**
Texture Spectrum	8	2KB
Daubechies	16	2KB
Haar	16	2KB
Haralick	24	2KB
Zernike	36	2KB
Rotation-Invariant LBP	108	8KB
Edge Histogram	150	12KB
Normalized Histogram	256	16KB
BIC Histogram	502	32KB

Source: Oliveira *et al.* (2019).

Table 10 presents the setup of the experiments over the MAMMOSET data. The table contains information about each feature extractor used for generating the feature vectors for the ROIs, as well as the page sizes employed by the Slim-trees to index each kind of feature vector.

[2] <http://peipa.essex.ac.uk/info/mias.html>

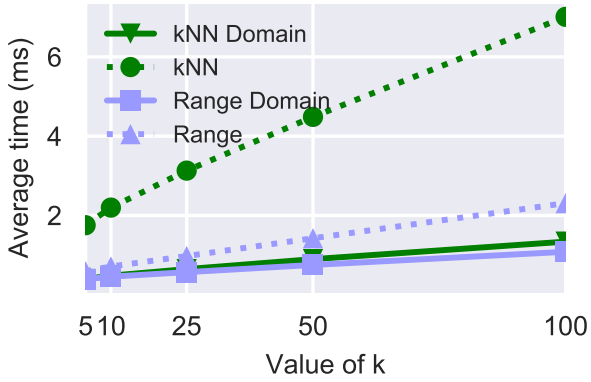
Moreover, we employed the Canberra Distance for all query executions. Such choice was based on a study on mammography image retrieval (BUGATTI *et al.*, 2008), which showed that the Canberra Distance usually enables more accurate query results in the context of mammograms.

To run the content-based queries, all images from MAMMOSET were used as query images, and both *k*NN and Range queries were executed once for each query image. The queries accessed the five individual Slim-trees in the Common approach and the unified Slim-tree in the Domain Index approach. We used the following values of *k*: 5, 10, 25, 50, and 100. Such values were used in the *k*NN queries, and the distance value of each *k*-th element retrieved was used as the query radius in the respective Range queries for retrieving the same number of elements. Then, the average time of all executions was computed both for *k*NN and Range queries. In the Common approach, both the time for searching the indexes and the time for performing the merging step were taken into account.

5.3 Results

After running the experiments for the nine feature extractors, we were able to visualize the difference between *k*NN and Range queries in terms of performance gains. For every feature extractor, regardless of their different dimensionalities, the *k*NN queries achieved greater gain using the Domain Index approach. Figure 29 presents results over Haralick feature vectors.

Figure 29 – Average execution times of *k*NN and Range queries, for both Common and Domain Index approaches, over Haralick feature vectors in MAMMOSET.



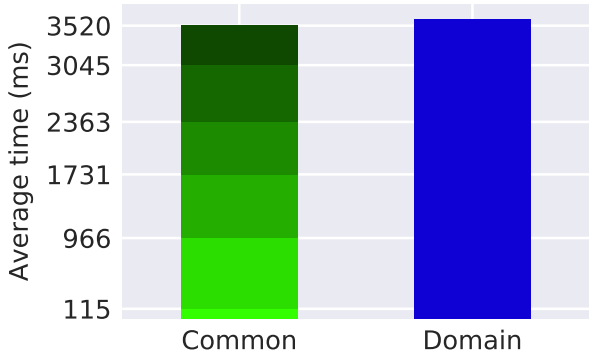
Source: Oliveira *et al.* (2019).

In the figure, the line depicting *kNN Domain* is lower than both *kNN* and *Range* lines, which shows the significant gains in execution time provided by our method (between 77% and 81%) when running *k*NN queries. On the other hand, the performance gains of our proposal in Range queries varied from 34% to 53%. As the figure shows, the gains in both query types become higher as *k* grows. Such behavior was similar for the other eight feature extractors as well, so we picked the Haralick results just for the sake of illustration. Since the Domain Index

enabled higher performance gains in *k*NN queries, we focus our discussion on *k*NN queries throughout the rest of this section.

Figure 30 shows the average time required to build the index structures in both approaches over the Haralick feature vectors. The *Common* bar represents the sum of building times for the separate index structures, one over each dataset. Each tick on the y-axis corresponds to the accumulated elapsed time in the Common approach during index construction over each dataset. Table 11 presents the detailed building times for both approaches.

Figure 30 – Average building times of index structures, for Common and Domain Index approaches, over Haralick feature vectors from MAMMOSET.



Source: Oliveira *et al.* (2019).

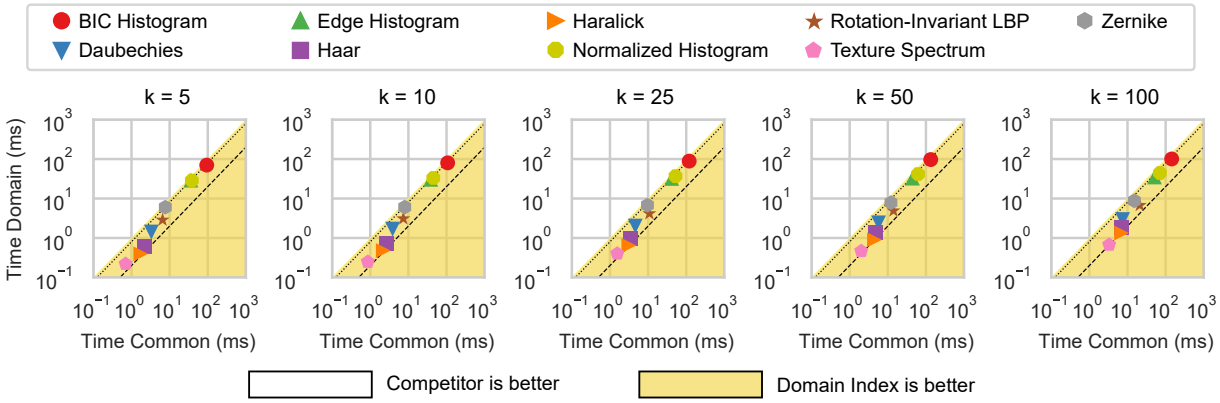
Table 11 – Detailed values of the average building time of index structures considering the Haralick feature vectors of MAMMOSET.

Dataset	**Elements**	**Average Time (ms)**
MINI-MIAS	118	115.3
DDSM - LCC	761	851.6
DDSM - LMLO	780	764.4
DDSM - RCC	656	632.1
DDSM - RMLO	695	682.5
VIENNA	446	474.7
Common (Total)	3456	**3520.3**
Domain	3456	**3601.7**

Source: Oliveira *et al.* (2019).

These building times consist of the mean calculated from 1000 executions, from which 10% of the best and 10% of the worst collected times were left out in order to remove potential outliers. We can see that the building times are similar: the Common approach was only 0.02% faster. By dividing both average times by the number of elements in the dataset, the verified time for inserting a single element was 1.02ms in the Common approach and 1.04ms in the Domain Index. The time for inserting one element comprises the time required to descend the Slim-tree from the root to the leaf node plus the time to store the element in the node.

Figure 31 – Scatter plots of the average *k*NN query times, for both Common and Domain Index approaches, over all feature types from MAMMOSET. The highlighted area in each graph, delimited by the lower right triangle, is where our method wins. In all query executions, our method enables from 19% to 81% of performance gain, the former being for the Edge Histogram feature vectors and the latter being for the Haralick feature vectors. We highlight this fact through the dotted and dashed lines in the graphs, which mostly encompass all nine markers in the five graphs and represent, respectively, 20% and 80% of gain for the Domain Index.



Source: Oliveira *et al.* (2019).

Figure 31 shows the results from *k*NN queries for all feature extractors employed. The figure presents the query execution times of the Common and Domain Index approaches for each value of *k*. It shows the markers, each of which representing the average query time execution for a feature extractor, mostly between the dotted and dashed lines, which represent, respectively, 20% and 80% of performance gain for our method.

Table 12 – Detailed gains of the Domain Index for each feature extractor in *k*NN queries.

Feature Extractor	**Dimensions**	**Minimum Gain**	**Maximum Gain**
Texture Spectrum	8	68%	79%
Daubechies	16	56%	61%
Haar	16	72%	73%
Haralick	24	77%	81%
Zernike	36	22%	42%
Rotation-Invariant LBP	108	56%	68%
Edge Histogram	150	19%	31%
Normalized Histogram	256	25%	37%
BIC Histogram	502	25%	30%

Source: Oliveira *et al.* (2019).

Furthermore, Table 12 provides additional details from the same experiments in terms of performance gains. More specifically, the table shows the minimum and maximum percentages of gain, considering the performance of our method compared to the Common approach, for each feature extractor employed in the experiments. The table shows that the feature extractors having low and moderate dimensionalities (from 8 to 108) allowed greater percentages of performance

gain. Nonetheless, the feature extractors with higher dimensionalities (from 150 to 502) provided significant gains as well, from 19% to 37%.

5.3.1 Statistical Analyses on Query Execution Times

In Figure 29, we can see a linear behavior in the evaluated methods. We conducted statistical analyses and they showed that the Gamma Generalized Linear Model (GLM) (DOBSON; BARNETT, 2018) correctly describes the results, which confirms that the evaluated methods are linear indeed. Table 13 shows the estimates of the model. Range queries employing the Domain Index approach (*Range Domain* in Figure 29) had the best performance. Therefore, in Table 13, they were the basis against which the other methods were compared.

The interpretation of Table 13 is as follows. Row 1 shows information about the estimated initial point of query time for *Range Domain*, whereas Rows 2–4 represent the gain (or loss) of the other query-method combinations when compared to *Range Domain*. Row 5 presents the effect of the variable k on query time, considering *Range Domain* as the basis. Rows 6–8, in turn, present the effect of the other query-method combinations on query time compared to *Range Domain*. The meaning of Rows 5–8 can be observed in the slopes of the lines in Figure 29.

Table 13 – GLM for query times over Haralick feature vectors of MAMMOSET.

Row		Estimate	Standard Error	p-value
1	**(Intercept)**	**0.3747**	**0.0011**	**0%**
2	Range	0.1466	0.0019	0%
3	kNN Domain	-0.0067	0.0016	0%
4	kNN	1.1013	0.0050	0%
5	**k**	**0.0071**	**0.0000**	**0%**
6	k × Range	0.0003	0.0001	0%
7	k × kNN Domain	0.0029	0.0001	0%
8	k × kNN	0.0348	0.0002	0%

Source: Oliveira *et al.* (2019).

Focusing on the *Estimate* column of Table 13, it is possible to estimate the query time as a function of k. Take as an example the kNN query in the Common approach ("kNN" both in Figure 29 and in Table 13). The initial point is obtained by $0.3747 + 1.1013$, i.e. adding the *Estimate* values of Rows 1 and 4. Then, if we consider $k = 10$, we can obtain the estimate by adding $10 \times (0.0071 + 0.0348)$ to the previous sum. By doing so, we are considering the impact of k (*Estimate* value of Row 5) when it equals 10 plus the impact of employing the Common approach in kNN queries (*Estimate* value of Row 8). The final sum is 1.895, which represents the estimated query time in this example.

Analyzing Table 13, we can observe that the coefficients in the column *Estimate* of Rows 5–8 are all positive and distinct. Moreover, as the p-values are 0%, we are able to state that the evaluated methods are statistically different from each other and that *Range Domain* has the best overall performance. Also, as the *Estimate* value of Row 3 is negative, *kNN Domain* has a lower initial point when compared to *Range Domain*, hence being superior when retrieving few elements from the dataset (considering small values of k). However, as k grows, *Range Domain* surpasses *kNN Domain* in terms of query performance.

5.4 Discussion

CBIR systems are commonly used in clinical environments for supporting physicians in the decision-making process. Such systems are especially important in scenarios in which physicians have to deal with data from multiple repositories, which is often the case in large-sized hospitals due to the diversity of medical equipment. Dealing with such variety in medical scenarios can be cumbersome, since it usually implies the analyzing data from separate repositories by querying them with separate index structures.

In this context, we explored an application of the Domain Index in queries commonly performed by medical CBIR systems. From the experimental results, we can see that the Domain Index provided significant gains, both in kNN and Range queries, considering data features of varied dimensionalities. Hence, our proposal enables the development of more usable and robust tools, which can ease the work of repository developers/maintainers and healthcare Information Technology (IT) support in scenarios involving data domains across multiple repositories.

CHAPTER

6

ENABLING AN RDBMS WITH ATTRIBUTE DOMAINS AS FIRST-CLASS CITIZENS

The RDBMS is one of the prevailing technologies for data querying, and recent studies have enhanced the core capabilities of these systems, e.g. the inclusion of native support for semi-structured data (CHASSEUR; LI; PATEL, 2013; TAHARA; DIAMOND; ABADI, 2014; LIU; HAMMERSCHMIDT; MCMAHON, 2014; LIU *et al.*, 2016), as well as the ability of querying data from heterogeneous sources, including formats other than database tables (ALAGIANNIS *et al.*, 2012; CHAMANARA; KÖNIG-RIES; JAGADISH, 2017). As previously discussed in this thesis, there is still room for improvement in RDBMSs, and we have targeted the problem of efficiently searching attribute domains across multiple tables, which led to the proposal of the Domain Index.

Some techniques of modern RDBMSs can be somehow employed to perform Domain Queries. We analyze three of them in this chapter: *inheritance*, *partitioning*, and *indexed materialized view* (ELMASRI; NAVATHE, 2015). However, these solutions may not be suitable for performing Domain Queries depending on the scenario. The main reasons are: **(i)** some of them are rather limited in terms of applicability; **(ii)** they may lack efficiency, both for handling attribute domains and for performing Domain Queries; and **(iii)** one of them (indexed materialized view) leads to data redundancy. Such drawbacks indicate the need for a wider solution for attribute domain management in RDBMSs, in order to enable efficient Domain Query execution while avoiding costly data maintenance operations.

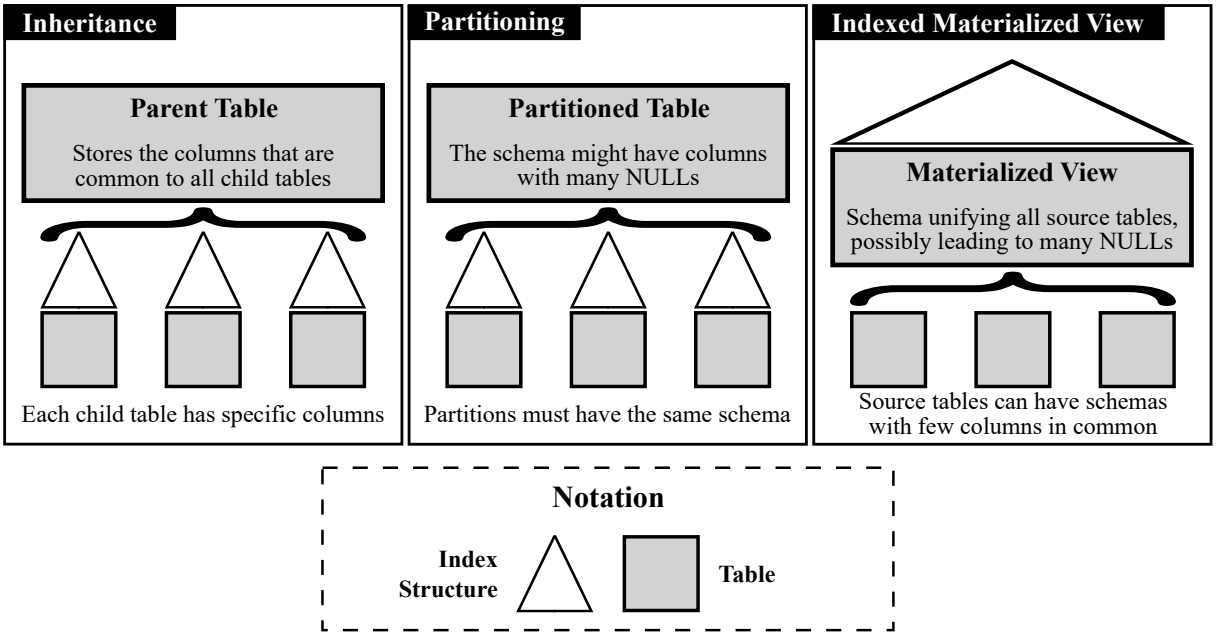
This chapter presents our approach for enhancing RDBMSs with the concept of *attribute domains*, enabling their definition in RDBMSs with the automatic creation of their corresponding Domain Indexes. We also detail the existing (and limited) support for Domain Queries in modern RDBMSs. Moreover, we report an empirical assessment of the alternative techniques, available in a modern row-store RDBMS, compared to the Domain Index. Results over real-world and synthetic data show that our proposal outperforms the competitors in all evaluated queries.

The remainder of this chapter is organized as follows. Section 6.1 describes the evaluated techniques of modern RDBMSs in the context of Domain Queries. Section 6.2 presents our approach for enabling attribute domains in RDBMSs. Section 6.3 discusses the experimental evaluation and the results. Finally, Section 6.4 concludes the chapter.

6.1 Existing Techniques Applied to Domain Queries

Overall, the techniques analyzed in this chapter are available in major open-source and commercial RDBMSs. Some details may vary in the implementations depending on the RDBMS, such as the available criteria for assigning data to the correct partition in a partitioning scenario. With respect to SQL commands, each technique has its own way to express queries. Regardless of such details, we describe the techniques in their essence. To express Domain Queries using SQL for each technique, we employ the syntax used by the RDBMS PostgreSQL v10 onwards. The available techniques are presented in Figure 32 and are further detailed as follows.

Figure 32 – Existing techniques of modern RDBMSs applied to Domain Queries.



Source: Elaborated by the author.

Inheritance. This technique is employed in scenarios involving specialization. There is a parent table, whose schema is composed of columns representing *common* properties, and child tables, whose schema is composed of columns representing *specific* properties. If desired, the parent table can be left empty, for instance when implementing partitioning via table inheritance. In the context of Domain Queries, employing the inheritance technique works when the queries search across a parent table and its child tables. This is possible because, even though each child table has its own specific columns, the child tables may have columns of a same data type playing related roles, i.e. columns from the same attribute domain. The inheritance technique has the

advantage of allowing a specific schema to each table involved in the specialization, without the need for a unified schema to accommodate the data. However, when executing Domain Queries, this technique must resort to the Common approach previously described in this thesis, since each table must have its separate index structure. To express a Domain Query in SQL using the inheritance technique, the following structure can be employed:

```
SELECT <ColumnList> FROM <ParentTable> WHERE <Criteria>;
```

This query accesses the parent table and all child tables, even if some table is empty, individually scanning the index structures. In Figure 32, the setup employed by the inheritance technique keeps the parent table empty, hence the depiction of index structures only on the child tables.

Partitioning. This technique is employed when the data of a same entity can be divided into distinct tables according to some criterion. The Database Administrator (DBA) creates the parent table and specifies the criterion that must be employed to partition the table. The criterion may be, for instance, a range or list of values that can be stored in a column of the partitioned table. Such column acts as a router, which we refer to as the *routing column* of the table, since it is used for directing inserted data to the correct partition. Then, the DBA creates partitions based on the respective (e.g. range or list of) values in the routing column for each partition. Differently from the inheritance technique, the partitioning technique requires all partitions to have the same schema. Therefore, in the context of Domain Queries, the DBA must define a unified schema for the partitioned table to accommodate the data. However, such unified schema is likely to have nullable columns, since some columns may be originally present in only part of the data. Like in the inheritance technique, Domain Queries must be executed through the Common approach. As the structure of a partitioned table is only logical, the data are actually stored in the partitions, thus each partition must have its own index structures. To express a Domain Query in SQL using the partitioning technique, the following structure can be employed:

```
SELECT <ColumnList> FROM <PartitionedTable> WHERE <Criteria>;
```

Although a SQL query with this structure has the partitioned table in the `FROM` clause, it searches only the partitions. When `Criteria` includes any filter on the routing column, the query accesses only the satisfying partitions. Otherwise, it searches all partitions.

Indexed Materialized View. In the context of Domain Queries, materialized views can be created with a unified schema, in order to accommodate the data from all tables sharing the attribute domain. The index over the materialized view, in turn, can be defined on the column storing the active domain. An indexed materialized view has the advantage of precomputing certain queries by centralizing data from distinct tables. However, the use of a materialized view requires additional time related to data reorganization, as well as increased disk usage due to data redundancy. To express a Domain Query in SQL over a materialized view, the following structure is employed:

```
SELECT <ColumnList> FROM <IndexedMaterializedView> WHERE <Criteria>;
```

Differently from the inheritance and partitioning techniques, the `Criteria` on the (now unified) active domain can be satisfied with a single index scan.

6.2 Enabling Attribute Domains in RDBMSs

Domain Queries can be seen as a counterpart of the relational queries currently available in RDBMSs. A relational query applies relational operators to the set of values that constitute a relation. A Domain Query, in turn, applies relational operators to the set of values that constitute the active domain within a set of attributes. Such active domain is also a relation, resulting from the union of values from each individual attribute. Therefore, Domain Queries are relational queries as well, and as such they can be implemented in RDBMSs based on relational grounds.

6.2.1 Specifying Attribute Domains

The first step to address the problem of attribute-domain querying in RDBMSs is to enable the specification of attribute domains.

Definition. An attribute domain is the active domain of a set of attributes from multiple relations. To specify an attribute domain in an RDBMS, the following parameters are required:

- a set of tables $T = \{\text{Table}_1, \ldots, \text{Table}_N\}$; and
- a set of column lists $CL = \{\text{CList}_1, \ldots, \text{CList}_N\}$, one per table, where CList_i represents the column list of Table_i, $1 \leq i \leq N$, $N \in \mathbb{Z}^+$, $N \geq 2$.

Each column list in CL can have a distinct number of *elements*. The *elements*, however, must present the same structure in the entire attribute domain. Such structure can either be:

- a single column in the respective table; or
- a fixed-length sequence of columns in the respective table.

Additionally, the elements must have compatible types. In the case of single-column elements, all columns must store values of the same type. Considering fixed-length sequences of columns, on the other hand, the columns may be of different types within a sequence, but all sequences must have the same structure.

Example #1

Suppose a `Date/Time` attribute domain from data of financial activities, such as sales and returns both on the Web and in-store, corresponding to the active domain of attributes storing values of date and time. Consider such attributes are stored in columns from tables `wSales`,

sSales, wReturns, and sReturns. To specify the Date/Time attribute domain, the required parameters would be defined as follows.

```
 T = {sSales, sReturns, wSales, wReturns}
CL = {sSalesList, sReturnsList, wSalesList, wReturnsList}

  sSalesList = {(SaleDate, SaleTime)}
sReturnsList = {(ReturnDate, ReturnTime)}
  wSalesList = {(SaleDate, SaleTime), (ShipDate, ShipTime)}
wReturnsList = {(ReturnDate, ReturnTime)}
```

In this example, the elements in all column lists are sequences of length 2 (i.e. multi-column elements, whose two columns store values of Date and Time types respectively).

Example #2

Now, consider a Customer attribute domain in the same context of financial activities. To specify this attribute domain over the same set of tables, the required parameters would be:

```
 T = {sSales, sReturns, wSales, wReturns}
CL = {sSalesList, sReturnsList, wSalesList, wReturnsList}

  sSalesList = {CustomerID}
sReturnsList = {CustomerID}
  wSalesList = {SaleCustomerID, ShipCustomerID}
wReturnsList = {CustomerID}
```

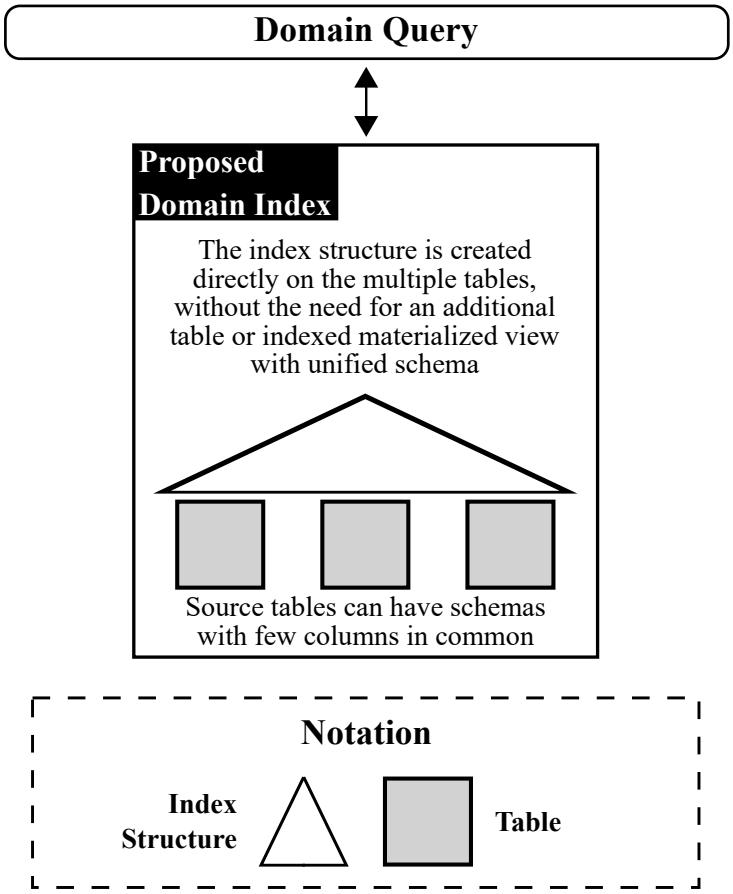
In this example, the elements in all column lists are single columns of the same type, i.e. all of them store numeric identifiers.

6.2.2 Technical Aspects of Domain Indexes

Analogously to the analyzed techniques of modern RDBMSs in Section 6.1, we show in Figure 33 the proposed Domain Index being employed by an RDBMS.

Compared to the existing approaches of modern RDBMSs in the context of Domain Queries, Domain Indexes do not impose any data reorganization nor data redundancy. When employing the inheritance technique, for instance, there could be some overhead when arranging the original data into a specialization scheme, if the original data were not organized in such a way beforehand. The partitioning technique might lead to a similar overhead, with the difference that the data would be subdivided into partitions rather than being arranged according to a specialization hierarchy. Regarding the indexed materialized view technique, there is always the

Figure 33 – Overview of Domain Index employed by an RDBMS in a Domain Query.



Source: Elaborated by the author.

need for initially building the materialized view and keeping it synchronized with the source data. Furthermore, additional disk usage is required to store the materialized view, as well as any index created on it. In addition to not imposing any data overhead, the Domain Index is able to be employed in any scenario, in contrast to the inheritance and partitioning techniques, which in turn have limited applicability.

The Domain Index is a category of access methods that can be implemented using any underlying index structure. Regardless of which index structure is used in the implementation, two steps are necessary to embed a Domain Index in a modern RDBMS: **(i)** to define the metadata storage about the Domain Index; and **(ii)** to modify the structure of row identifiers (row IDs), so that the Domain Index is able to correctly fetch the data during queries. Regarding the metadata storage, RDBMS developers can define their own criteria on how the metadata must be stored in the database catalog. With respect to the row IDs, their structure must somehow include information about what is the source table in which the indexed value is stored. This is necessary because the Domain Index is an access method that spans multiple tables. Hence, it must be aware of the table where each value is located. Such information can be physically embedded in the row ID itself, although this approach could lead to an overall swollen row ID structure. That

is, instead of just storing the disk block identifier (within the table) and the offset for the data entry in that block, such approach would add a third piece of information, corresponding to the table ID. Therefore, to avoid swelling the row ID structure of the index structure that implements the Domain Index, it is possible to employ a hashing scheme, which can work as follows. Instead of storing three pieces of information, the row ID stores a single identifier. Then, the RDBMS can employ a mechanism that hashes the single identifier into the triple `(table_id, block_id, offset)`, for instance by keeping track of the identifier-to-triple correspondences via metadata information in the catalog.

In this thesis, we have not addressed parallelism in Domain Queries. The parallel query engines of major systems, such as the Oracle Database, are able to identify when an index is partitioned. Then, they can fire simultaneous queries to scan partitioned indexes. In such scenarios, our proposal may be less efficient when compared to parallel index scans. Nevertheless, the proposed Domain Index can be made parallelizable as well. Furthermore, the Domain Index has the advantage of being applicable to scenarios where the tables have different schemas, differently than partitioned indexes.

6.2.3 Extended SQL

We extend some Data Definition Language (DDL) commands for attribute domains, namely the `CREATE`, `ALTER`, and `DROP` operations, and we also show how to express Domain Queries using SQL.

Maintenance Syntax

The `CREATE` command can specify an attribute domain as follows:

```
CREATE ATTRIBUTE DOMAIN <Name> ON
  <Table_1> (<CList_1>),
  ...,
  <Table_N> (<CList_N>)
  [ USING <AccessMethod> ];
```

Each column list (`CList_i` in the command) is composed of elements which can present one of the two structures described in Section 6.2.1. If the elements present the first structure, which corresponds to single-column elements, they are expressed as comma-separated column names. If they present the second structure, which corresponds to multi-column elements, each element must be a comma-separated sequence enclosed by the characters `{` and `}`.

In addition to storing the metadata about the attribute domain in the RDBMS catalog, the `CREATE` command triggers the creation of a Domain Index. It is possible (but not mandatory) to provide the `AccessMethod` to be employed by the Domain Index. If the SQL command is written without specifying the access method, a standard structure (such as B-tree) can be used

by default. Moreover, other options can be provided according to the available implementations in the RDBMS, such as Hash and Generalized Inverted Index (GIN).

Considering the same examples presented in Section 6.2.1, to express the specification of `Date/Time` and `Customer` attribute domains in SQL, the following commands could be used:

```
CREATE ATTRIBUTE DOMAIN Date_Time ON
    sSales ({SaleDate, SaleTime}),
    wSales ({SaleDate, SaleTime}, {ShipDate, ShipTime}),
  sReturns ({ReturnDate, ReturnTime}),
  wReturns ({ReturnDate, ReturnTime})
  USING btree;
```

```
CREATE ATTRIBUTE DOMAIN Customer ON
    sSales (CustomerID),
    wSales (SaleCustomerID, ShipCustomerID),
  sReturns (CustomerID),
  wReturns (CustomerID)
  USING btree;
```

The `ALTER` and `DROP` commands can be used, respectively, to rename and remove existing attribute domains, along with their corresponding Domain Indexes. The commands are:

```
ALTER ATTRIBUTE DOMAIN <Name> RENAME TO <NewName>;
```

```
DROP ATTRIBUTE DOMAIN <Name>;
```

Query Syntax

The overall structure of SQL commands for Domain Queries is as follows:

```
SELECT <ColumnList_1> FROM <Table_1> WHERE <Criteria>
UNION ALL
SELECT <ColumnList_2> FROM <Table_2> WHERE <Criteria>
UNION ALL
...
UNION ALL
SELECT <ColumnList_N> FROM <Table_N> WHERE <Criteria>;
```

Each table accessed by the query, expressed in the `FROM` clause, belongs to the same attribute domain. Moreover, the column lists projected in each `SELECT` clause can contain either columns from the tables involved in the attribute domain or columns generated at runtime, such as for displaying values of aggregate functions. Furthermore, when retrieving columns that are specific to a certain table, the respective position in the column list for the other tables should be `NULL`, in order to enable a compatible schema for the `UNION ALL` operator.

As shown in the command, there is no specific reserved word to trigger the execution of a Domain Index. The idea is to make a Domain Query execution transparent, without requiring the RDBMS user to determine when the query should employ a Domain Index behind the curtains. Instead, this decision is left to the RDBMS, which takes into account the `Criteria` expressed in the `WHERE` clause to make such a decision. More specifically, this process works as follows.

- If the `Criteria` are specified solely on columns that constitute the attribute domain, then the respective Domain Index can be employed.
- If only part of the columns in the attribute domain appears in the `WHERE` clause, then the query optimizer must decide whether to employ a Domain Index or ordinary indexes. For instance, if only one of the multiple columns (consequently, only one table) of an attribute domain is specified as part of the `Criteria`, then it might be more beneficial to search an ordinary index structure defined on the specified column. Therefore, the query optimizer must evaluate the database statistics, such as histograms for selectivity estimation, in order to make the best choices regarding indexes.
- If the `Criteria` involve columns that do not belong to an attribute domain, then the query optimizer will rely on the Common approach, at least for part of the query execution.

Finally, other clauses such as `GROUP BY` and `ORDER BY` can be employed as needed, just like any query not involving attribute domains.

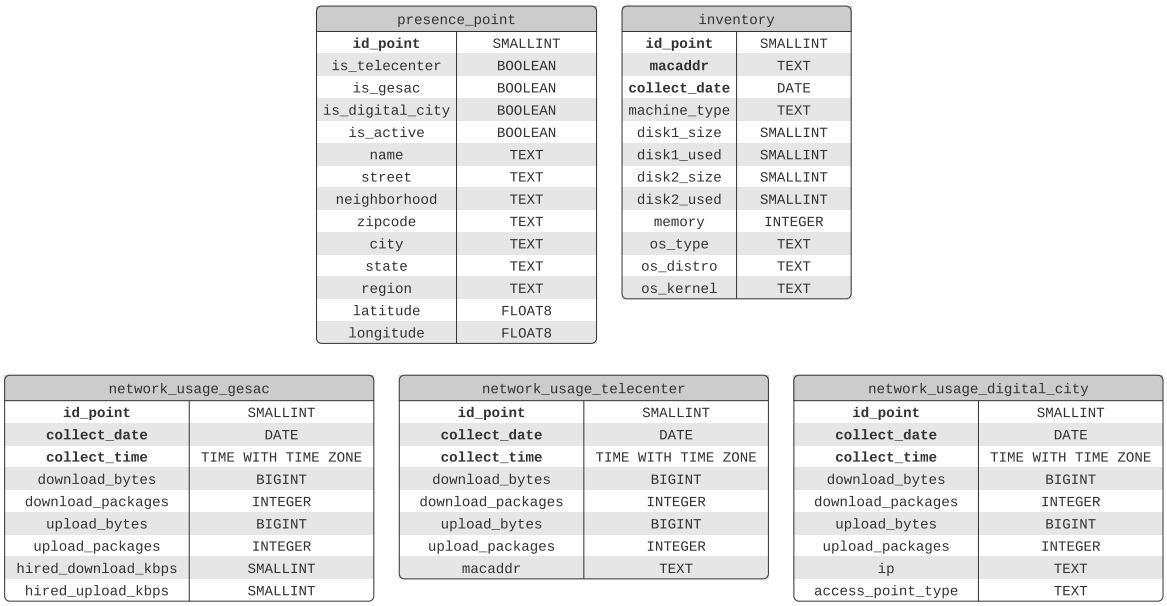
6.3 Experiments

We evaluated the techniques described in Section 6.1, comparing them to the proposed Domain Index, both on real-world and synthetic datasets. For simplicity, the plots refer to the techniques using shorter words: inheritance → INH, partitioning → PAR, indexed materialized view → VIEW, Domain Index → DOM. The experiments were run on PostgreSQL v10, using a computer with an Intel® Core™ i5-2400 @ 3.10GHz processor, 4GB of DDR3 1333MHz RAM memory, SATA 6Gb/s 7200RPM hard disk, and Linux operating system Fedora 27.

6.3.1 Real-World Dataset: SIMMC

The real dataset employed is called Monitoring Integrated System of the Ministry of Communications (SIMMC) (PASQUALIN *et al.*, 2017), which consists of data collected through a monitoring system used by the Ministry of Science, Technology, Innovation and Communications of Brazil for tracking digital inclusion projects. The dataset is still active at the moment this thesis was written, storing more than one million of records per day, which comes to billions of records since 2014 (its release date). The data are divided into three projects: Telecenter, Gesac, and Digital Cities. For each project, there is information about software, hardware inventory, and network usage (acquired from routers). There are common attributes among the three projects, such as the network traffic collection date and time, as well as download and upload bandwidths. Conversely, there are exclusive attributes for each project, e.g. the Media Access Control (MAC) address (in the Telecenter project), the hired download and upload bandwidths (in the Gesac project), as well as the computer IP address and access point type (in the Digital Cities project). For this set of experiments, we used a part of the dataset referring to the year 2017.

Figure 34 – Original tables from the SIMMC dataset.



Source: Elaborated by the author.

As seen in Figure 34, the three "network usage" tables have most of their columns in common. Therefore, they are suitable for being modelled in inheritance and partitioning schemes. The tables and materialized views, along with their respective sizes and cardinalities, for all evaluated techniques are shown in Table 14.

Table 14 – Tables and materialized views for all evaluated techniques over SIMMC.

Name	Type	Size	Cardinality
inh_network_usage	Table	0 B	0
inh_network_usage_d	Table	78 GB	764,248,128
inh_network_usage_g	Table	45 GB	521,986,760
inh_network_usage_t	Table	483 MB	4,636,343
par_network_usage	Table	0 B	0
par_network_usage_d	Table	87 GB	764,248,128
par_network_usage_g	Table	53 GB	521,986,760
par_network_usage_t	Table	518 MB	4,636,343
view_network_usage	View	131 GB	1,290,871,231
view_telecenter	View	267 MB	4,644,788
inventory	Table	1048 kB	8,445
presence_point	Table	267 MB	17,269

Source: Elaborated by the author.

In Table 14, tables with prefixes "inh" and "par" refer, respectively, to the inheritance and partitioning tables, whereas the suffixes "d", "g", and "t" refer, respectively, to the Digital

Cities, Gesac, and Telecenter inclusion projects. Specifically, the partitioned tables were a bit larger than the inheritance ones, due to an extra column used for routing the inserted rows to the correct partitions. Moreover, the `view_network_usage` materialized view was built from concatenating all rows from the partitioned tables, including all their columns (even the routing column, in order to allow identifying the source table) except for the `id` serial column, which had been created for both inheritance and partitioning techniques as an artificial primary key. The `view_telecenter` materialized view was created from the `par_network_usage_t` and `inventory` tables, and consisted of columns `id_point`, `collect_date`, `macaddr`, and `source`, the latter being a column for denoting the source table.

The following attribute domains were considered for the experiments: **(i)** `download_bytes` across network tables; **(ii)** `upload_bytes` across network tables; **(iii)** `(collect_date, collect_time)` across network tables; and **(iv)** `collect_date` across `par_network_usage_t` and `inventory` tables. These attribute domains were each accessed by a corresponding query. The queries executed in this set of experiments are as follows.

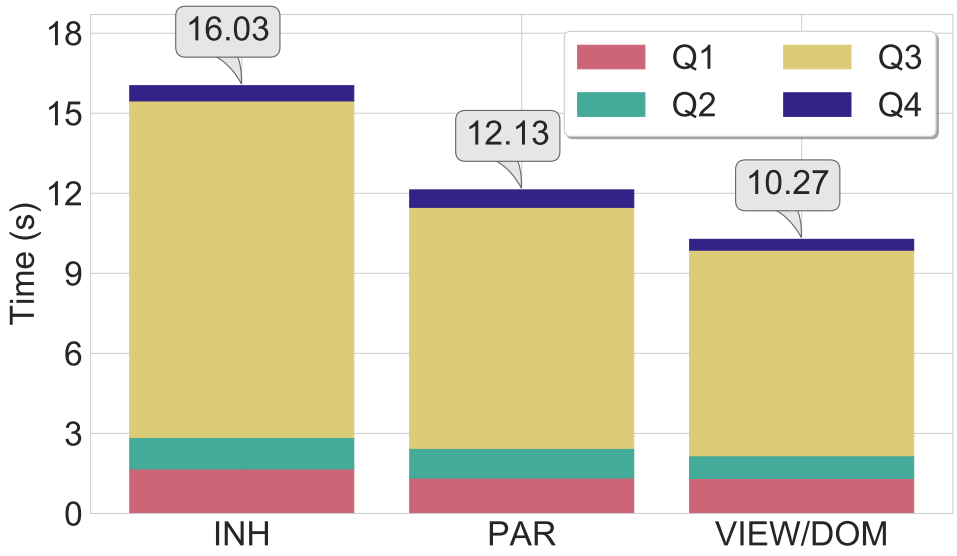
Q1. Return the presence points that had the 50 highest measures of network usage, considering all inclusion projects, in terms of download bytes.

Q2. Report the inclusion projects involved in alerts of more than 3 GB of upload, also informing the corresponding date and time.

Q3. Retrieve the total amount of download (in TB), measured from 20 December 2017 to 31 December 2017, which occurred from 2 PM to 6 PM for every project.

Q4. Considering the Telecenter project, retrieve the presence points which had any network or inventory activity on 1st July 2017.

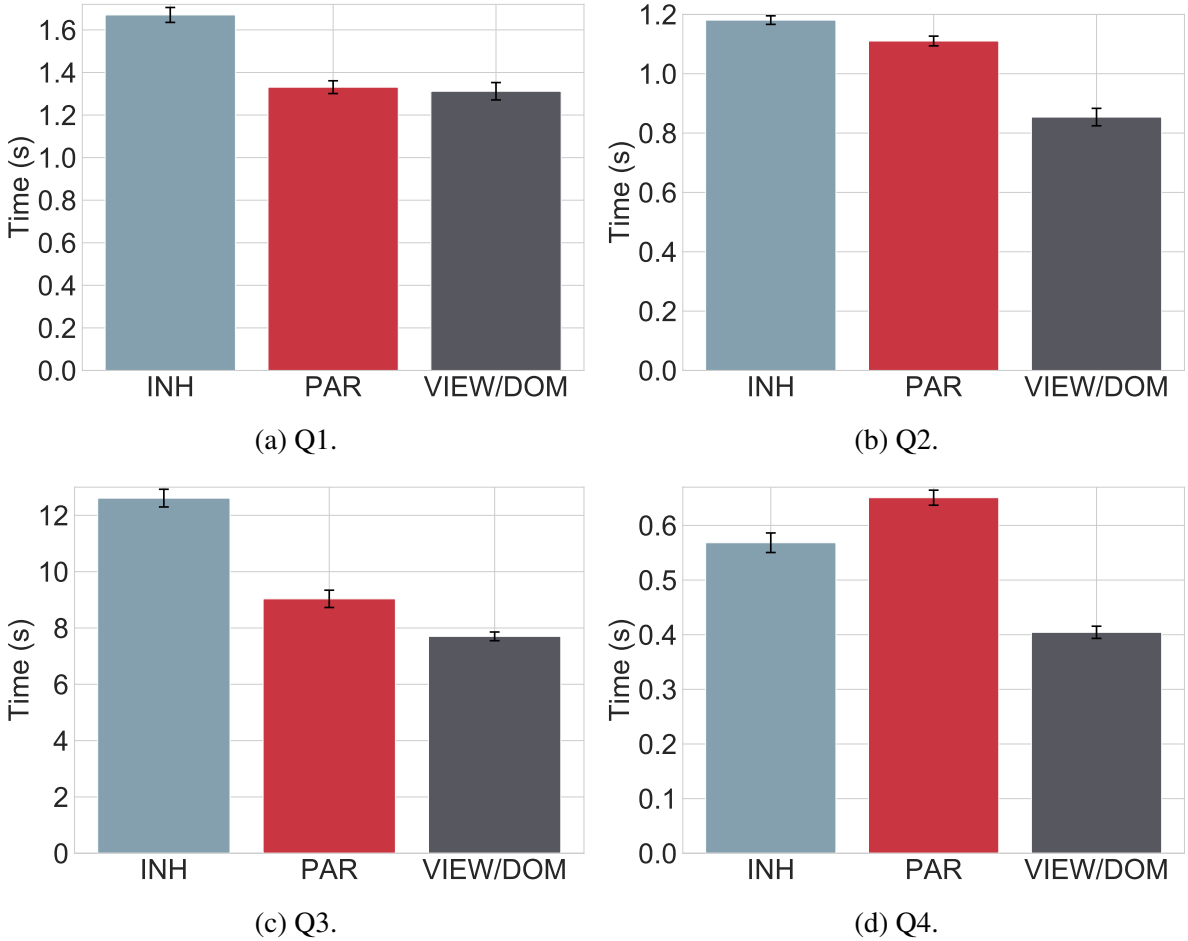
Figure 35 – Stacked execution times over SIMMC.



Source: Elaborated by the author.

Figure 35 shows for each evaluated technique the total execution time comprehending all four queries. We can see that indexed materialized view approach provided the best performance, which is equivalent to the one enabled by Domain Indexes due to employing a single index structure, hence the "Domain Index" term in the figure.

Figure 36 – Individual execution times over SIMMC.



(a) Q1. (b) Q2.

(c) Q3. (d) Q4.

Source: Elaborated by the author.

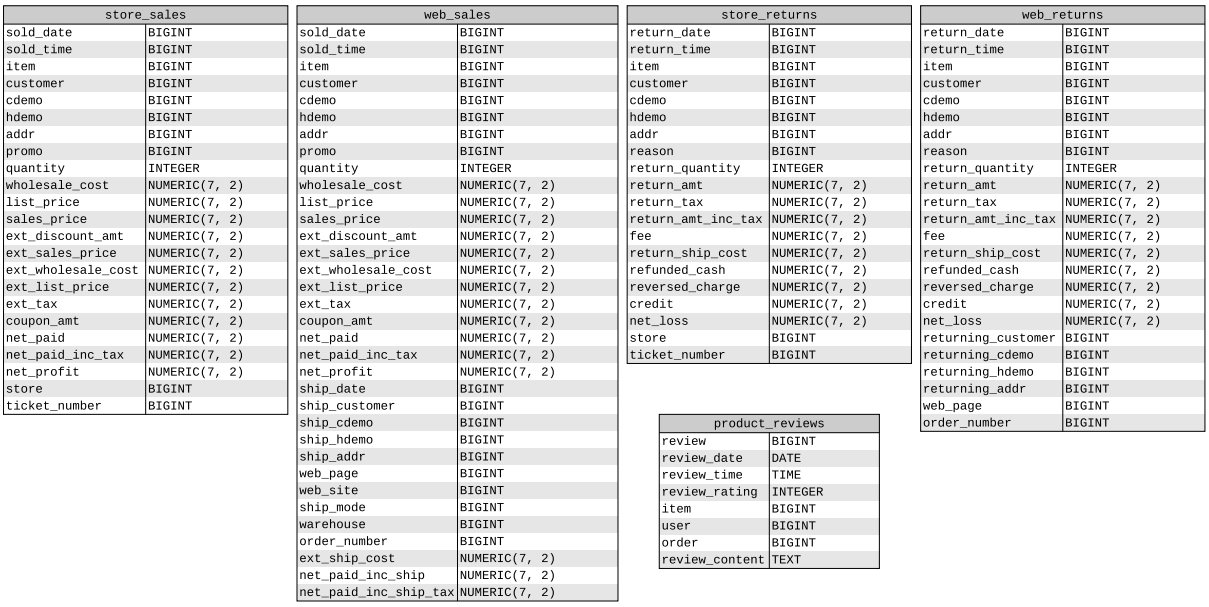
Figure 36 shows separate plots, one per query, enabling to analyze the techniques for each query individually. We can see that the indexed materialized view (and consequently the Domain Index) enabled the best performance for all queries, which was an expected behavior. Conversely, the inheritance and partitioning exhibited variations in their behaviors. For instance, the results from Q4 (Figure 36d) comprise the only situation in which inheritance outperformed partitioning, albeit slightly. Considering Q2 (Figure 36b), inheritance and partitioning almost tied. We noticed that the RDBMS produced distinct query plans for inheritance and partitioning: although both accessed indexes for all predicates, the algorithms employed for accessing the indexes varied between the two techniques. With respect to the indexed materialized view technique, only at Q1 (Figure 36a) it tied with the partitioning technique. We can observe that this is the only query that does not involve filtering out rows by means of a selective predicate. Rather, the indexes are only

inspected backwards due to the descending ordering of `download_bytes`, which led to a quite minimal performance gain enabled by the indexed materialized view. Nevertheless, considering all the four queries, we see that the Domain Index would enable the best performance.

6.3.2 Synthetic Dataset: TPCx-BB Benchmark

The synthetic data were generated through the TPCx-BB benchmark[1] (CAO *et al.*, 2016). TPCx-BB uses a retail model of activities on distinct sales channels, such as in-store and online. In this set of experiments, we employed part of the tables in TPCx-BB, corresponding to both structured and unstructured data. The structured data refer to sales and returns occurred in-store and online, whereas the unstructured data are composed of generated textual product reviews. Figure 37 shows the tables, along with their schemas, used in the experiments.

Figure 37 – Original tables from TPCx-BB used in the experiments.



Source: Elaborated by the author.

Table 15 – Table cardinalities for each SF considered in the TPCx-BB dataset.

Name	SF 1 — Cardinality	SF 3 — Cardinality	SF 10 — Cardinality
`store_sales`	667,579	2,061,140	8,228,343
`web_sales`	668,052	2,063,659	8,223,189
`store_returns`	37,902	117,448	467,208
`web_returns`	38,487	117,484	465,812
`product_reviews`	89,991	157,076	292,731

Source: Elaborated by the author.

[1] <http://www.tpc.org/tpcx-bb/>

In TPCx-BB, the resulting table sizes are based on a parameter named Scale Factor (SF), which allowed us to conduct a scalability evaluation. For the experiments, we used SF values of 1, 3, and 10. We describe the table cardinalities for each SF in Table 15. Analogously to how the tables were arranged for the experiments over the SIMMC dataset, the tables of Table 15 were arranged as follows. For inheritance, both "sales" tables constitute a specialization scheme, and the same applies to the "returns" tables. For the `product_reviews` table, we added a column of `tsvector`[2] type generated from the `review_content` column, in order to enable indexed text searches. Then, the parent table had the same original columns, and we created six child tables, all of them having the additional column of `tsvector` type. Each child table of `product_reviews` stored reviews within a 1-year period. For partitioning, the table arrangement was analogous, but unifying distinct schemas into a single one. With respect to sales and returns data, an additional column was added to be the partition key based on the sales channel (in-store or online). Such additional column was not necessary for the inheritance technique, as PostgreSQL provides a way to natively obtain the table source when using inheritance. Considering product reviews, the partitioning occurred by range, based on the same 1-year periods used for inheritance. Finally, the materialized views were created from the partitioned tables: one view encompassing sales and returns of both sales channels, as well as one view of product reviews.

The following attribute domains were considered in these experiments: **(i)** `customer` across sales and returns; **(ii)** `review_content` across product reviews; and **(iii)** `date` across sales (including sale dates and shipment dates) and returns. Each attribute domain was accessed by a corresponding query, designed according to a potential day-to-day scenario.

Q5. Show the activity history of customers 10896 and 20000 for in-store and online purchases.

Q6. Identify product reviews that used the word "bad", retrieving the review content and the associated Web company.

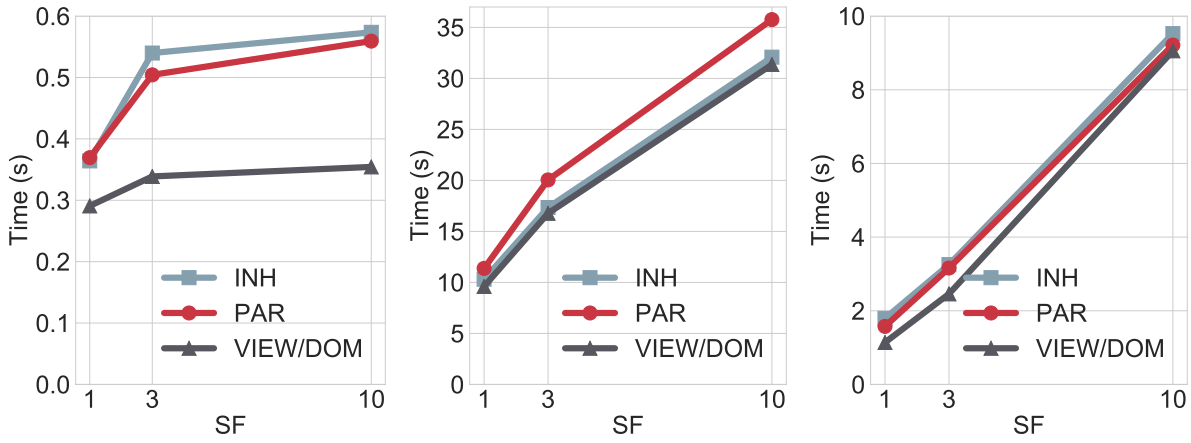
Q7. Display the in-store and online activities from 1st April 2001 to 1st April 2002.

Figure 38 shows the execution times of all evaluated techniques for Q5, Q6, and Q7. We can observe that VIEW/DOM had the best performance. For Q5, VIEW/DOM achieved from 22% to 39% of performance gain compared to the most time-consuming approach (inheritance or partitioning depending on the query-SF combination), as well as up to 16% for Q6 and up to 37% for Q7. Specifically, VIEW/DOM enabled a significantly better performance compared to the other techniques for Q5, which was the query involving the most selective predicate. As the SF values grew, INH and PAR had a sharper increase in time compared to VIEW/DOM, as we can see by analyzing the curve shapes on the plot. For Q6, VIEW/DOM had an overall similar performance compared to INH, although slightly better. This query had a less selective predicate, in addition to performing text search, which added some overhead on top of the index accesses. Q7 employed a less selective predicate as well, filtering out data according to a considerably

[2] <https://www.postgresql.org/docs/current/datatype-textsearch.html>

large time span. In this context, it was expected that the overall performance of indexes would start to degrade for larger SF values, which could be seen for SF 10. Nonetheless, for SF values 1 and 3, VIEW/DOM enabled more significant performance gains compared to INH and PAR techniques.

Figure 38 – Execution times over TPCx-BB.



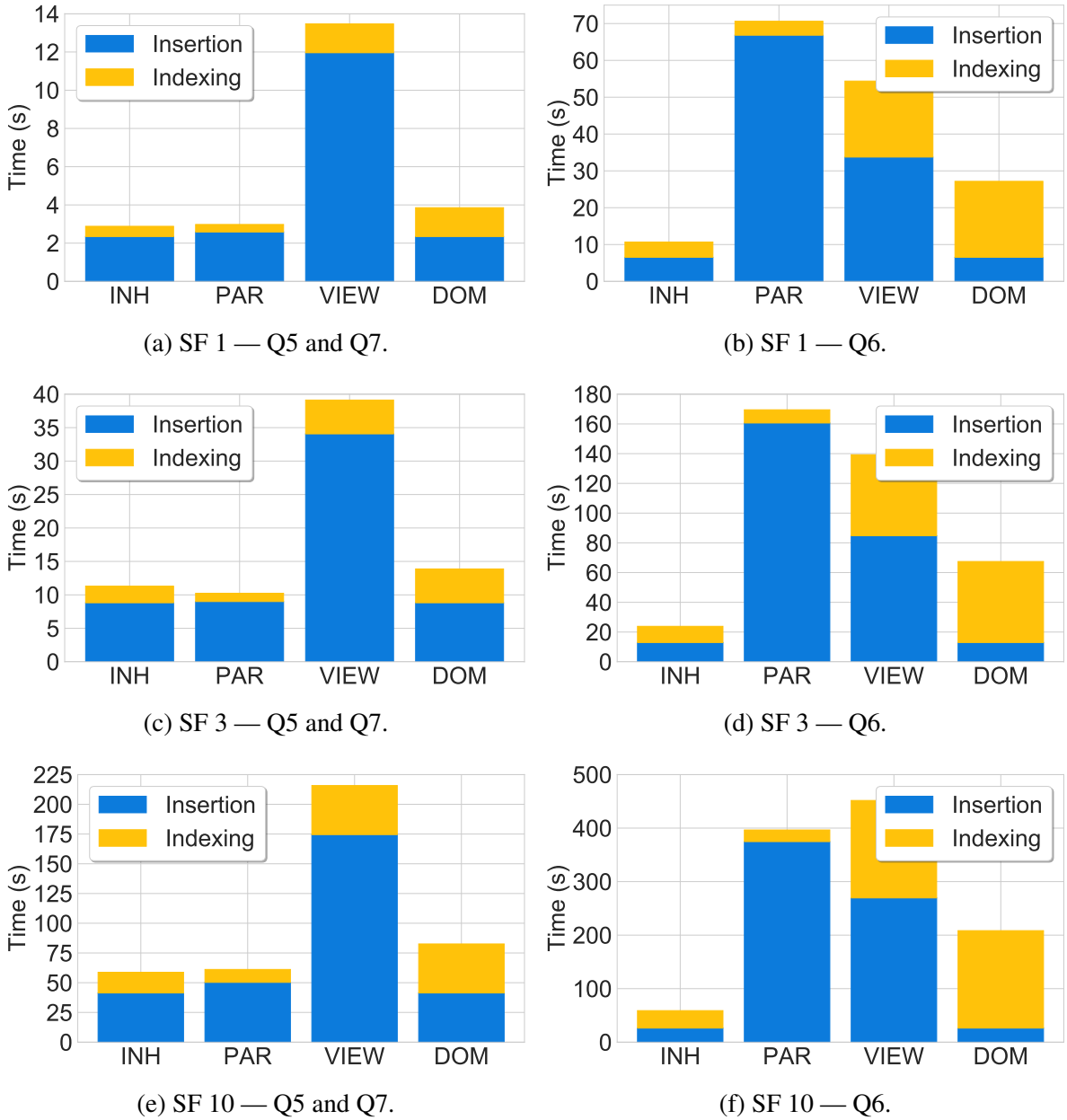
(a) Q5. (b) Q6. (c) Q7.

Source: Elaborated by the author.

Figure 39 shows the loading times spent by the evaluated techniques, for all SF values, to build the tables/views and the respective indexes accessed by the queries. As Q5 and Q7 accessed the same set of tables, both queries are mentioned together in the plots (Figure 39a, Figure 39c, and Figure 39e). For Q5 and Q7, we can observe that INH and PAR had an overall similar behavior, as both techniques deal with multiple tables, each table storing the same number of elements when comparing INH and PAR to each other, as well as multiple indexes, one per table. VIEW took the longest time on the insertion step, as this step involved duplicating data from the separate tables into a single materialized view. The index in VIEW took a bit longer to build as well, which was expected, since a bigger hierarchic index usually has more levels compared to multiple indexes, each one indexing part of the same data. With respect to the times shown for DOM, they correspond to the insertion time spent by INH plus the indexing time spent by VIEW. This is because the Domain Index approach is expected to present the behavior of INH for the insertion step, since the original dataset tables will be built like any ordinary table in an RDBMS, and is expected to present the behavior of VIEW for the indexing step, as a single index structure will be employed to index the data. As for Q6 (Figure 39b, Figure 39d, and Figure 39f), we can observe a similar behavior for all techniques, except for PAR. That occurred because, differently from INH, the data rows were not straightforwardly inserted into the (partitioned) tables. Rather, there was an additional processing for the RDBMS to route each row to the correct partition. Although such processing had a minor impact on the insertion time for Q5 and Q7, it incurred a quite long time for Q6 due to the overhead associated with comparing date values at each new row. With respect to VIEW and DOM, their indexing time was longer than the ones of INH

and PAR as well, but such a difference was more significant for Q6 than for Q5 and Q7. That occurred because the indexes employed for Q6 were GINs, and the indexing step took especially longer for VIEW and DOM due to the overhead of comparing textual keys on top of the (already longer) time to traverse the index structure. Nevertheless, analyzing the times for DOM, we can see that they are between the times of the other techniques, but especially closer to the times of INH, which were the lowest in all scenarios.

Figure 39 – Loading times over TPCx-BB.



(a) SF 1 — Q5 and Q7.

(b) SF 1 — Q6.

(c) SF 3 — Q5 and Q7.

(d) SF 3 — Q6.

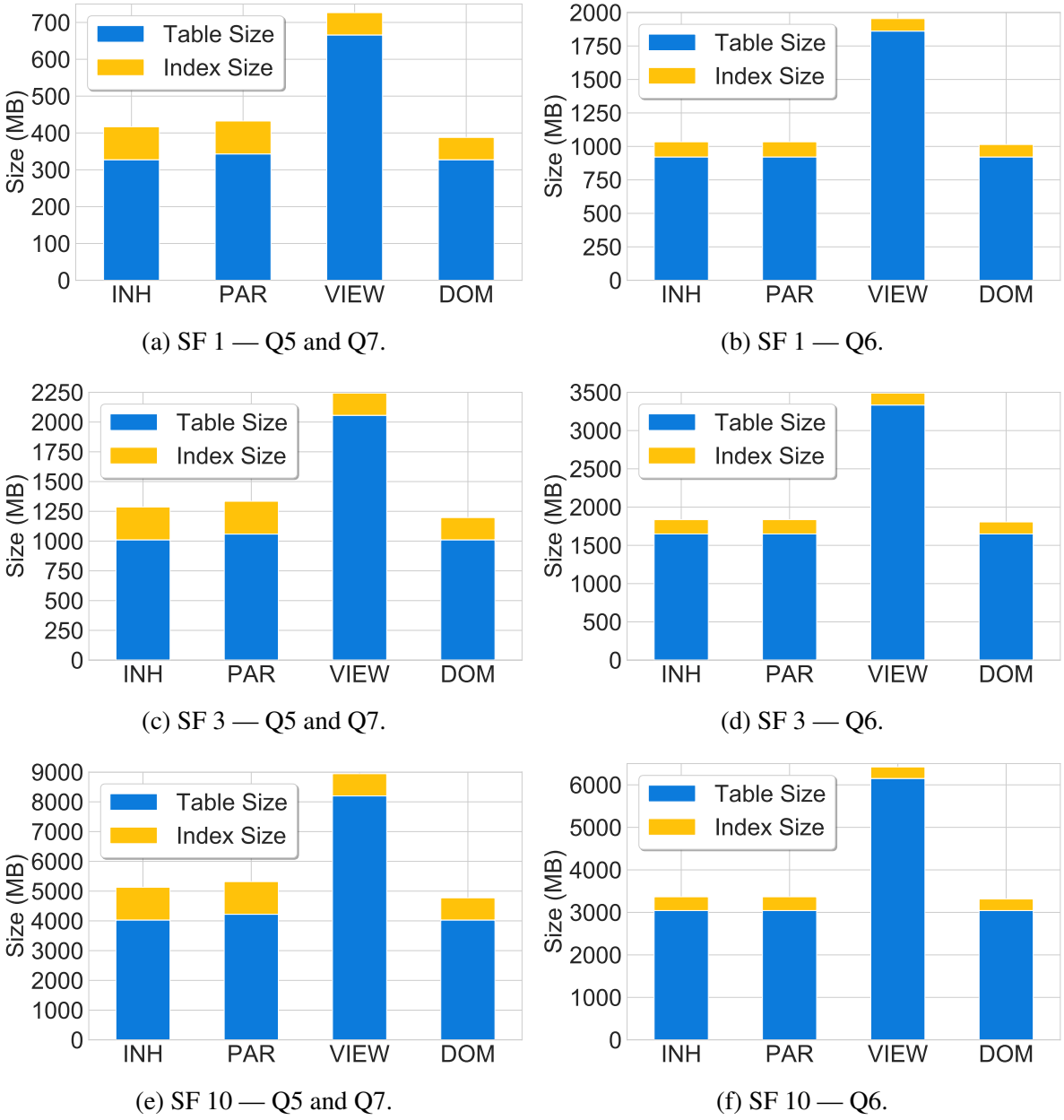
(e) SF 10 — Q5 and Q7.

(f) SF 10 — Q6.

Source: Elaborated by the author.

Figure 40 shows the table and index sizes in MB, for all SF values, considering the same tables-per-query arrangement that we just analyzed. We observe that INH and PAR had a similar behavior, with PAR occupying slightly more space due to the additional column used

Figure 40 – Table and index sizes over TPCx-BB.



(a) SF 1 — Q5 and Q7.

(b) SF 1 — Q6.

(c) SF 3 — Q5 and Q7.

(d) SF 3 — Q6.

(e) SF 10 — Q5 and Q7.

(f) SF 10 — Q6.

Source: Elaborated by the author.

as partition key. VIEW, on the other hand, accounts for the original data in the separate tables plus the redundant data in the materialized view. However, the index size of VIEW is smaller compared to the ones of INH and PAR. This is because the index of VIEW was stored using less blocks in disk, and such blocks had a higher occupancy rate, as opposed to the indexes of INH and PAR that used more blocks with a lower occupancy rate. We can see this behavior on all the plots in the figure. The sizes of DOM, in turn, consist of the table size of INH plus the index size of VIEW, analogously to the loading times in Figure 39. In this case, however, DOM was the one with the lowest occupied space in disk.

6.4 Discussion

In this chapter, we presented a novel approach that allows enhancing RDBMSs with attribute domains. We showed the elements that must be provided when defining an attribute domain, as well as how to express maintenance commands for attribute domains using SQL. We also described how (and when) the existing techniques of modern RDBMSs can be employed in the context of Domain Queries, as well as how a Domain Index would operate in the environment of an RDBMS. Our experimental evaluation shows the potential of the Domain Index, which outperforms the existing techniques in many scenarios with respect to query time, data loading time, as well as disk space requirements. Furthermore, we highlight the fact that the Domain Index is a proper technique for dealing with Domain Queries, as opposed to the existing techniques of modern RDBMSs that are quite limited in terms of applicability.

CHAPTER

7

CONCLUSIONS

As presented in this monograph, this PhD research addressed problems related to data quality and data analysis. We discuss our contributions next, together with the research problems they solve and the thesis proposed in this PhD research.

7.1 Contributions

We have advanced the state of the art in the area of Databases with four contributions, which bring improvements to both the preprocessing and analytics phases of the Data Analytics pipeline. We summarize our contributions as follows.

- The first contribution, which focused on the preprocessing phase of the Data Analytics pipeline, consists of an incremental data cleaning framework which performs repairs of categorical data holistically, i.e. jointly considering multiple error detection criteria.
- From the second contribution onwards, we focused on the analytics phase of Data Analytics. Specifically, our second contribution consists of an access method, called Domain Index, for speeding up queries over Attribute Domains. Our experimental evaluation showed that the proposed method outperforms the approach commonly employed in Domain Queries.
- The third contribution focused on carrying out a comprehensive case study in the context of medical data, employing the proposed Domain Index. This contribution allows facilitating and speeding up the access to domains of medical data across multiple repositories, both in *k*NN and Range queries, ultimately aiding physicians in decision-making.
- The fourth contribution focused on specifying the support to Attribute Domains as first-class citizens in RDBMSs. We also simulated the Domain Index against existing technologies of modern RDBMSs, which proved to be unsuitable to performing Domain Queries, both in terms of applicability and performance.

Given the invaluable task of preparing data prior to analyzing them, our first contribution has the potential to improve data quality, thus leading to better data analysis results down the road. Furthermore, our three other contributions approached a new category of queries, denoted Domain Queries, which had not been explored in the literature until the execution of this PhD research, but are increasingly useful to process data collected in several related repositories, e.g. as part of data lakes. Our latter contributions also enabled the validation of our hypothesis that Domain Indexes lead to a superior performance of Domain Queries. Therefore, by combining our four contributions, it is possible to improve the handling of Attribute Domains from their collection, enabling better data quality upon integrating data from distinct repositories, to the analysis phase, speeding up the access in queries.

We recall the thesis proposed in this PhD research:

Thesis. *The use of a holistic incremental approach for data cleaning, allied to the support for managing and querying domains of attributes in RDBMSs, improves performance in both the preprocessing and analytics phases of the Data Analytics pipeline.*

Indeed, according to the proposals and results presented in this monograph, we are able to state that our contributions have the potential to improve Data Analytics as a whole. Moreover, they open possibilities to further developments in research.

7.2 Future Work

Based on the studies and developments carried out in this PhD research, we envision some lines of work for extending our contributions, which we present as follows.

1. With respect to incremental data cleaning, a future work includes enabling the proposed framework to support other data types, such as numerical data. This can introduce different challenges not addressed in this PhD research, such as incrementally updating statistics other than co-occurrences, considering the usage of different feature types that do not rely on counting. One example is to employ embedding features, which are low-dimensional, learned continuous vector representations of raw data. In this context, it can be challenging to update the learned embedding features incrementally. Strategies using an autoencoder (GOODFELLOW; BENGIO; COURVILLE, 2016) or a Variational AutoEncoder (VAE) (KINGMA; WELLING, 2019) can be inspected.

2. Regarding our contributions on Domain Queries, it is possible to explore parallelism in the Domain Index implementation. Considering hierarchical index structures, for instance, a possibility is to analyze the performance of Domain Indexes at accessing multiple subtrees concurrently in a Domain Query. Such analysis could also consider the parallel access to multiple separate indexes, as in the commonly employed approach of Domain Queries.

3. As a further development to the previous line of work, it is possible to deepen the case study carried out for CBIR systems in the medical domain. Specifically, one can explore Domain Indexes over medical repositories via a network. This line of work involves some challenging aspects, such as dealing with fault tolerance, as it implies accessing distinct parts of the index structure through the network. For instance, with respect to the structure employed by the Domain Index to handle row IDs, a possible investigation in the context of a network could be to employ a quadruple, including an element for keeping track of the host machine storing the indexed element, e.g. `(host_ip, table_id, block_id, offset)`. Furthermore, it is possible to access distinct parts of the index in parallel and then synchronize the results.

4. Another future work consists of actually implementing the support to Attribute Domains as a new feature of RDBMSs, directly in the source code of an open-source RDBMS, for instance. This line of work, given its more applied nature, can eventually allow vendors to embed such feature in their products, thus increasing the availability and reach of the proposed contributions.

7.3 List of Publications

The following publications have resulted directly from this PhD research.

- **SAC 2017** (OLIVEIRA *et al.*, 2017)
  One Index to Dominate Them All:
  Domain Indexes for Improving Queries across Multiple Tables
  Link: <https://dl.acm.org/doi/10.1145/3019612.3019678>

- **CBMS 2017** (OLIVEIRA *et al.*, 2017a)
  Efficiently Indexing Multiple Repositories of Medical Image Databases
  Link: <https://ieeexplore.ieee.org/document/8104204>

- **DSW 2017 (Dataset Showcase Workshop at SBBD 2017)** (OLIVEIRA *et al.*, 2017b)
  MAMMOSET: An Enhanced Dataset of Mammograms
  Highlight: Best Paper Award of DSW 2017.
  Link: <https://repositorio.usp.br/item/002854721>

- **Journal of Biomedical and Health Informatics** (OLIVEIRA *et al.*, 2019)
  Employing Domain Indexes to Efficiently Query Medical Data from Multiple Repositories
  Link: <https://pubmed.ncbi.nlm.nih.gov/30452381/>

- **arXiv – Computing Research Repository (CoRR)** (OLIVEIRA *et al.*, 2020)
  Batchwise Probabilistic Incremental Data Cleaning
  Link: <https://arxiv.org/abs/2011.04730>

The PhD candidate also worked on additional publications. Two of them were carried out primarily by the PhD candidate: **(i)** a research assignment conducted as part of a course in the PhD program (OLIVEIRA *et al.*, 2016); and **(ii)** the final publication of his MSc research (OLIVEIRA; TRAINA-JR.; KASTER, 2017), which had contributions from his current supervisor. The other publications resulted from the PhD candidate's collaborations with members of the Database and Image Group (GBDI). These collaborations were invaluable experiences that contributed to the PhD candidate's formation. The additional publications are listed as follows.

- **ICEIS 2016** (OLIVEIRA *et al.*, 2016)
  On the Support of a Similarity-Enabled RDBMS in Civilian Crisis Situations
  Link: <https://dl.acm.org/doi/10.5220/0005816701190126>

- **Information Systems** (OLIVEIRA; TRAINA-JR.; KASTER, 2017)
  CLAP, ACIR and SCOOP:
  Novel Techniques for Improving the Performance of Dynamic Metric Access Methods
  Link: <http://dx.doi.org/10.1016/j.is.2017.10.003>

- **ITNG 2017** (SPADON *et al.*, 2017a)
  Complex Network Tools to Understand the Behavior of Criminality in Urban Areas
  Link: <https://arxiv.org/abs/1612.06115>

- **SBBD 2017** (SCABORA *et al.*, 2017)
  Relational Graph Data Management on the Edge:
  Grouping Vertices' Neighborhood with Edge-k
  Highlight: Best Paper Award of SBBD 2017.
  Link: <http://sbbd.org.br/2017/wp-content/uploads/sites/3/2018/02/p124-135.pdf>

- **ICCS 2017** (SPADON *et al.*, 2017b)
  Behavioral Characterization of Criminality Spread in Cities
  Link: <https://doi.org/10.1016/j.procs.2017.05.118>

- **CBMS 2018** (NESSO-JR. *et al.*, 2018)
  RAFIKI: Retrieval-Based Application for Imaging and Knowledge Investigation
  Link: <https://doi.org/10.1109/CBMS.2018.00020>

- **Journal of Information and Data Management** (SCABORA *et al.*, 2018)
  Cutting-Edge Relational Graph Data Management with Edge-k:
  From One to Multiple Edges in the Same Row
  Link: <https://periodicos.ufmg.br/index.php/jidm/article/view/413/9779>

- **SAC 2020** (SCABORA *et al.*, 2020)
  Enhancing Recursive Graph Querying on RDBMS with Data Clustering Approaches
  Link: <https://doi.org/10.1145/3341105.3375770>

BIBLIOGRAPHY

ABEDJAN, Z.; CHU, X.; DENG, D.; FERNANDEZ, R. C.; ILYAS, I. F.; OUZZANI, M.; PAPOTTI, P.; STONEBRAKER, M.; TANG, N. Detecting Data Errors: Where Are We and What Needs to Be Done? **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 12, p. 993–1004, Aug. 2016. ISSN 2150-8097. Available: <https://doi.org/10.14778/2994509.2994518>. Citation on page 50.

ABEDJAN, Z.; GOLAB, L.; NAUMANN, F. Profiling Relational Data: A Survey. **The VLDB Journal**, Springer-Verlag, Berlin, Heidelberg, v. 24, n. 4, p. 557–581, Aug. 2015. ISSN 1066-8888. Available: <https://doi.org/10.1007/s00778-015-0389-y>. Citations on pages 39 and 49.

ABEDJAN, Z.; QUIANÉ-RUIZ, J.-A.; NAUMANN, F. Detecting Unique Column Combinations on Dynamic Data. In: **Proceedings of the 30th International Conference on Data Engineering**. Los Alamitos, CA, USA: IEEE Computer Society, 2014. (ICDE '14), p. 1036–1047. Citation on page 45.

AFRATI, F. N.; KOLAITIS, P. G. Repair Checking in Inconsistent Databases: Algorithms and Complexity. In: **Proceedings of the 12th International Conference on Database Theory**. New York, NY, USA: Association for Computing Machinery, 2009. (ICDT '09), p. 31–41. ISBN 9781605584232. Available: <https://doi.org/10.1145/1514894.1514899>. Citation on page 42.

ALAGIANNIS, I.; BOROVICA, R.; BRANCO, M.; IDREOS, S.; AILAMAKI, A. NoDB: Efficient Query Execution on Raw Data Files. In: **39th ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2012. (SIGMOD '12), p. 241–252. Citation on page 111.

AMARI, S.-i. **Differential-Geometrical Methods in Statistics**. New York: Springer Science & Business Media, 2012. Citation on page 35.

AOUCHICHE, M.; HANSEN, P. Distance Spectra of Graphs: A Survey. **Linear Algebra and Its Applications**, Elsevier, v. 458, p. 301–386, 2014. Citation on page 35.

ARENAS, M.; BERTOSSI, L.; CHOMICKI, J. Consistent Query Answers in Inconsistent Databases. In: **Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems**. New York, NY, USA: Association for Computing Machinery, 1999. (PODS '99), p. 68–79. ISBN 1581130627. Available: <https://doi.org/10.1145/303976.303983>. Citation on page 42.

AROCENA, P. C.; GLAVIC, B.; MECCA, G.; MILLER, R. J.; PAPOTTI, P.; SANTORO, D. Messing Up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 2, p. 36–47, Oct. 2015. ISSN 2150-8097. Available: <https://doi.org/10.14778/2850578.2850579>. Citation on page 62.

BAYLOR, D.; BRECK, E.; CHENG, H.-T.; FIEDEL, N.; FOO, C. Y.; HAQUE, Z.; HAYKAL, S.; ISPIR, M.; JAIN, V.; KOC, L.; KOO, C. Y.; LEW, L.; MEWALD, C.; MODI, A. N.; POLYZOTIS, N.; RAMESH, S.; ROY, S.; WHANG, S. E.; WICKE, M.; WILKIEWICZ, J.; ZHANG,

X.; ZINKEVICH, M. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In: **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2017. (KDD '17), p. 1387–1395. Available: <https://doi.org/10.1145/3097983.3098021>. Citation on page 50.

BESKALES, G.; ILYAS, I. F.; GOLAB, L.; GALIULLIN, A. On the Relative Trust between Inconsistent Data and Inaccurate Constraints. In: **Proceedings of the 29th International Conference on Data Engineering**. Los Alamitos, CA, USA: IEEE Computer Society, 2013. (ICDE '13), p. 541–552. ISBN 9781467349093. Available: <https://doi.org/10.1109/ICDE.2013.6544854>. Citations on pages 41 and 50.

BLEIFUSS, T.; KRUSE, S.; NAUMANN, F. Efficient Denial Constraint Discovery with Hydra. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 11, n. 3, p. 311–323, Nov. 2017. ISSN 2150-8097. Available: <https://doi.org/10.14778/3157794.3157800>. Citation on page 39.

BLODGETT, T. M.; MELTZER, C. C.; TOWNSEND, D. W. PET/CT: Form and Function. **Radiology**, Radiological Society of North America, v. 242, n. 2, p. 360–385, 2007. Citation on page 37.

BOHANNON, P.; FAN, W.; GEERTS, F.; JIA, X.; KEMENTSIETSIDIS, A. Conditional Functional Dependencies for Data Cleaning. In: **Proceedings of the 23rd International Conference on Data Engineering**. [S.l.]: IEEE Computer Society, 2007. (ICDE '07), p. 746–755. Citations on pages 28 and 40.

BRESSAN, R. S.; ALVES, D. H. A.; VALERIO, L. M.; BUGATTI, P. H.; SAITO, P. T. M. DOCToR: The Role of Deep Features in Content-Based Mammographic Image Retrieval. In: IEEE. **Proceedings of the 31st International Symposium on Computer-Based Medical Systems**. [S.l.], 2018. (CBMS '18), p. 158–163. Citation on page 38.

BUGATTI, P. H.; TRAINA, A. J. M.; FELIPE, J. C.; TRAINA-JR., C. A New Method to Efficiently Reduce Histogram Dimensionality. In: GIGER, M. L.; KARSSEMEIJER, N. (Ed.). **Medical Imaging 2008: Computer-Aided Diagnosis**. SPIE, 2008. v. 6915, p. 907–915. Available: <https://doi.org/10.1117/12.770512>. Citation on page 105.

BURKOV, A. **The Hundred-Page Machine Learning Book**. Quebec City: Andriy Burkov, 2019. Citation on page 25.

CAO, P.; GOWDA, B.; LAKSHMI, S.; NARASIMHADEVARA, C.; NGUYEN, P.; POELMAN, J.; POESS, M.; RABL, T. From BigBench to TPCx-BB: Standardization of a Big Data Benchmark. In: **Technology Conference on Performance Evaluation and Benchmarking**. New York, NY, USA: Springer, 2016. p. 24–44. Citation on page 123.

CAPPUZZO, R.; PAPOTTI, P.; THIRUMURUGANATHAN, S. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In: **47th ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2020. (SIGMOD '20), p. 1335–1349. ISBN 9781450367356. Available: <https://doi.org/10.1145/3318464.3389742>. Citation on page 41.

CARVALHO, L. O.; SERAPHIM, E.; SERAPHIM, T. F. P.; TRAINA, A. J. M.; TRAINA-JR., C. MedInject: A General-Purpose Information Retrieval Framework Applied in a Medical Context.

In: **27th IEEE International Symposium on Computer-Based Medical Systems**. [S.l.]: IEEE, 2014. (CBMS '14), p. 308–313. ISBN 978-1-4799-4435-4. ISSN 1063-7125. Citation on page 38.

CHAMANARA, J.; KÖNIG-RIES, B.; JAGADISH, H. V. QUIS: In-Situ Heterogeneous Data Source Querying. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 10, n. 12, p. 1877–1880, 2017. ISSN 2150-8097. Citation on page 111.

CHAPELLE, O.; SCHLKOPF, B.; ZIEN, A. **Semi-Supervised Learning**. 1st. ed. Cambridge: The MIT Press, 2010. ISBN 0262514125. Citation on page 43.

CHASSEUR, C.; LI, Y.; PATEL, J. Enabling JSON Document Stores in Relational Systems. In: **International Workshop on the Web and Databases**. New York, NY, USA: ACM, 2013. p. 1–6. Citation on page 111.

CHEN, L.; GAO, Y.; SONG, X.; LI, Z.; MIAO, X.; JENSEN, C. S. Indexing Metric Spaces for Exact Similarity Search. **CoRR**, abs/2005.03468, 2020. Available: <https://arxiv.org/abs/2005.03468>. Citation on page 33.

CHINO, D. Y. T.; SCABORA, L. C.; CAZZOLATO, M. T.; JORGE, A. E. S.; TRAINA-JR., C.; TRAINA, A. J. M. Segmenting Skin Ulcers and Measuring the Wound Area Using Deep Convolutional Networks. **Computer Methods and Programs in Biomedicine**, Elsevier, v. 191, p. 1–11, 2020. Citation on page 38.

CHOMICKI, J.; MARCINKOWSKI, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. **Information and Computation**, Academic Press, Inc., USA, v. 197, n. 1-2, p. 90–121, Feb. 2005. ISSN 0890-5401. Citation on page 42.

CHU, X.; ILYAS, I. F. Qualitative Data Cleaning. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 13, p. 1605–1608, Sep. 2016. ISSN 2150-8097. Available: <https://doi.org/10.14778/3007263.3007320>. Citation on page 28.

CHU, X.; ILYAS, I. F.; KRISHNAN, S.; WANG, J. Data Cleaning: Overview and Emerging Challenges. In: **Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2016. (SIGMOD '16), p. 2201–2206. ISBN 9781450335317. Available: <https://doi.org/10.1145/2882903.2912574>. Citation on page 28.

CHU, X.; ILYAS, I. F.; PAPOTTI, P. Discovering Denial Constraints. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 6, n. 13, p. 1498–1509, Aug. 2013. ISSN 2150-8097. Available: <https://doi.org/10.14778/2536258.2536262>. Citations on pages 39 and 40.

CHU, X.; MORCOS, J.; ILYAS, I. F.; OUZZANI, M.; PAPOTTI, P.; TANG, N.; YE, Y. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In: **Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2015. (SIGMOD '15), p. 1247–1261. ISBN 9781450327589. Available: <https://doi.org/10.1145/2723372.2749431>. Citation on page 41.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: **23rd International Conference on Very Large Data Bases**. San Francisco: Morgan Kaufmann Publishers, Inc., 1997. (VLDB '97), p. 426–435. ISBN 1-55860-470-7. Citation on page 93.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Communications of the ACM**, ACM, New York, NY, USA, v. 13, n. 6, p. 377–387, 1970. Available: <http://doi.acm.org/10.1145/362384.362685>. Citations on pages 25, 77, and 79.

CODY, W. F.; HAAS, L. M.; NIBLACK, W.; ARYA, M.; CAREY, M. J.; FAGIN, R.; FLICKNER, M.; LEE, D.; PETKOVIC, D.; SCHWARZ, P. M.; THOMAS, J.; ROTH, M. T.; WILLIAMS, J. H.; WIMMERS, E. L. Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project. In: **3rd IFIP WG2.6 Working Conference on Visual Database Systems 3 (VDB-3)**. London: Chapman & Hall, Ltd., 1995. p. 17–35. ISBN 0-412-72170-8. Citation on page 38.

COMER, D. The Ubiquitous B-tree. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 11, n. 2, p. 121–137, 1979. ISSN 0360-0300. Available: <http://doi.acm.org/10.1145/356770.356776>. Citation on page 80.

CONSONNI, C.; SOTTOVIA, P.; MONTRESOR, A.; VELEGRAKIS, Y. Discovering Order Dependencies through Order Compatibility. In: HERSCHEL, M.; GALHARDAS, H.; REINWALD, B.; FUNDULAKI, I.; BINNIG, C.; KAOUDI, Z. (Ed.). **Proceedings of the 22nd International Conference on Extending Database Technology**. OpenProceedings.org, 2019. (EDBT '19), p. 409–420. Available: <https://doi.org/10.5441/002/edbt.2019.36>. Citation on page 39.

COVER, T. M.; THOMAS, J. A. **Elements of Information Theory**. Hoboken: John Wiley & Sons, 2006. Citations on pages 31, 32, and 35.

CSISZÁR, I. Information-Type Measures of Difference of Probability Distributions and Indirect Observations. **Studia Scientiarum Mathematicarum Hungarica**, v. 2, p. 229–318, 1967. Citation on page 35.

DAI, J.; ZHANG, M.; CHEN, G.; FAN, J.; NGIAM, K. Y.; OOI, B. C. Fine-Grained Concept Linking Using Neural Networks in Healthcare. In: **45th ACM SIGMOD International Conference on Management of Data**. New York: ACM, 2018. (SIGMOD '18), p. 51–66. ISBN 978-1-4503-4703-7. Available: <http://doi.acm.org/10.1145/3183713.3196907>. Citation on page 37.

DAS, K.; SCHNEIDER, J. Detecting Anomalous Records in Categorical Datasets. In: **Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2007. (KDD '07), p. 220–229. ISBN 9781595936097. Available: <https://doi.org/10.1145/1281192.1281219>. Citation on page 41.

DAS, P.; NEELIMA, A. An Overview of Approaches for Content-Based Medical Image Retrieval. **International Journal of Multimedia Information Retrieval**, Springer, v. 6, n. 4, p. 271–280, 2017. Citation on page 38.

DESELAERS, T.; KEYSERS, D.; NEY, H. Features for Image Retrieval: An Experimental Comparison. **Information Retrieval**, Kluwer Academic Publishers, Hingham, v. 11, n. 2, p. 77–107, 2008. ISSN 1386-4564. Citation on page 36.

DOBSON, A. J.; BARNETT, A. **An Introduction to Generalized Linear Models**. 4th. ed. New York: CRC Press, 2018. (Chapman & Hall/CRC Texts in Statistical Science). ISBN 9781138741683. Citation on page 108.

DOHNAL, V.; GENNARO, C.; ZEZULA, P. Efficiency and Scalability Issues in Metric Access Methods. In: **Computational Intelligence in Medical Informatics**. [S.l.: s.n.], 2008. p. 235–263. Citation on page 103.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 7. ed. Hoboken: Pearson, 2015. ISBN 0-13-397077-9, 978-0-13-397077-7. Citations on pages 77 and 111.

FALOUTSOS, C. **Searching Multimedia Databases by Content**. Norwell: Kluwer Academic Publishers, 1996. 1–155 p. (Advances in Database Systems, v. 3). ISBN 978-0-7923-9777-9. Citation on page 36.

FAN, W.; GEERTS, F. **Foundations of Data Quality Management**. San Rafael: Morgan & Claypool Publishers, 2012. ISBN 160845777X. Citations on pages 39 and 40.

FAN, W.; GEERTS, F.; JIA, X.; KEMENTSIETSIDIS, A. Conditional Functional Dependencies for Capturing Data Inconsistencies. **ACM Transactions on Database Systems**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 2, Jun. 2008. ISSN 0362-5915. Available: <https://doi.org/10.1145/1366102.1366103>. Citation on page 44.

FAN, W.; JIA, X.; LI, J.; MA, S. Reasoning About Record Matching Rules. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 2, n. 1, p. 407–418, Aug. 2009. ISSN 2150-8097. Available: <https://doi.org/10.14778/1687627.1687674>. Citation on page 41.

FAN, W.; LI, J.; TANG, N.; YU, W. Incremental Detection of Inconsistencies in Distributed Data. **IEEE Transactions on Knowledge and Data Engineering**, IEEE Educational Activities Department, USA, v. 26, n. 6, p. 1367–1383, Jun. 2014. ISSN 1041-4347. Available: <https://doi.org/10.1109/TKDE.2012.138>. Citations on pages 26 and 45.

GEERTS, F.; MECCA, G.; PAPOTTI, P.; SANTORO, D. Cleaning data with Llunatic. **The VLDB Journal**, Springer, p. 1–26, 2019. Citation on page 41.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2. ed. Sebastopol: O'Reilly Media, 2019. Citation on page 25.

GETOOR, L.; MACHANAVAJJHALA, A. Entity Resolution: Theory, Practice & Open Challenges. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 5, n. 12, p. 2018–2019, Aug. 2012. ISSN 2150-8097. Available: <https://doi.org/10.14778/2367502.2367564>. Citation on page 41.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge: The MIT Press, 2016. (Adaptive Computation and Machine Learning). Citation on page 130.

GRANDVALET, Y.; BENGIO, Y. Semi-Supervised Learning by Entropy Minimization. In: **Advances in Neural Information Processing Systems 17**. [S.l.: s.n.], 2004. (NIPS '04), p. 529–536. Citation on page 43.

GRUENHEID, A.; DONG, X. L.; SRIVASTAVA, D. Incremental Record Linkage. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 7, n. 9, p. 697–708, May 2014. ISSN 2150-8097. Available: <https://doi.org/10.14778/2732939.2732943>. Citations on pages 26 and 45.

HE, J.; VELTRI, E.; SANTORO, D.; LI, G.; MECCA, G.; PAPOTTI, P.; TANG, N. Interactive and Deterministic Data Cleaning. In: **Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2016. (SIGMOD '16), p. 893–907. ISBN 9781450335317. Available: <https://doi.org/10.1145/2882903.2915242>. Citations on pages 26 and 45.

HEATH, M.; BOWYER, K.; KOPANS, D.; MOORE, R.; KEGELMEYER, W. P. The Digital Database for Screening Mammography. In: **5th International Workshop on Digital Mammography**. Madison: Medical Physics Publishing, 2001. (IWDM '01), p. 212–218. ISBN 1-930524-00-5. Citation on page 103.

HEIDARI, A.; MCGRATH, J.; ILYAS, I. F.; REKATSINAS, T. HoloDetect: Few-Shot Learning for Error Detection. In: **Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2019. (SIGMOD '19), p. 829–846. ISBN 9781450356435. Available: <https://doi.org/10.1145/3299869.3319888>. Citations on pages 26, 28, and 50.

HEIDARI, A.; MICHALOPOULOS, G.; KUSHAGRA, S.; ILYAS, I. F.; REKATSINAS, T. Record Fusion: A Learning Approach. **CoRR**, abs/2006.10208, p. 1–18, 2020. Available: <https://arxiv.org/abs/2006.10208>. Citation on page 41.

HEISE, A.; QUIANé-RUIZ, J.-A.; ABEDJAN, Z.; JENTZSCH, A.; NAUMANN, F. Scalable Discovery of Unique Column Combinations. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 7, n. 4, p. 301–312, Dec. 2013. ISSN 2150-8097. Available: <https://doi.org/10.14778/2732240.2732248>. Citation on page 39.

HELLERSTEIN, J. M. Quantitative Data Cleaning for Farge Databases. **United Nations Economic Commission for Europe (UNECE)**, v. 25, 2008. Citations on pages 28 and 41.

HOLT, A.; BICHINDARITZ, I.; SCHMIDT, R.; PERNER, P. Medical Applications in Case-Based Reasoning. **The Knowledge Engineering Review**, Cambridge University Press, v. 20, n. 3, p. 289–292, 2005. Citation on page 36.

HRIPCSAK, G.; DUKE, J. D.; SHAH, N. H.; REICH, C. G.; HUSER, V.; SCHUEMIE, M. J.; SUCHARD, M. A.; PARK, R. W.; WONG, I. C. K.; RIJNBEEK, P. R.; LEI, J. van der; PRATT, N.; NORéN, G. N.; LI, Y.-C.; STANG, P. E.; MADIGAN, D.; RYAN, P. B. Observational Health Data Sciences and Informatics (OHDSI): Opportunities for Observational Researchers. **Studies in Health Technology and Informatics**, NIH Public Access, v. 216, p. 574, 2015. Citation on page 37.

HSU, W.; ANTANI, S.; LONG, L. R.; NEVE, L.; THOMA, G. R. SPIRS: A Web-Based Image Retrieval System for Large Biomedical Databases. **International Journal of Medical Informatics**, v. 78, n. 1, p. S13–S24, 2009. Citation on page 38.

HUANG, H. K. **PACS-Based Multimedia Imaging Informatics: Basic Principles and Applications**. Hoboken: John Wiley & Sons, 2019. Citation on page 37.

ILYAS, I. F. Effective Data Cleaning with Continuous Evaluation. **IEEE Data Engineering Bulletin**, v. 39, n. 2, p. 38–46, 2016. Available: <http://sites.computer.org/debull/A16june/p38.pdf>. Citation on page 49.

ILYAS, I. F.; CHU, X. Trends in Cleaning Relational Data: Consistency and Deduplication. **Foundations and Trends in Databases**, Now Publishers, Inc., v. 5, n. 4, p. 281–393, 2015. Citations on pages 28, 40, and 49.

____. **Data Cleaning**. New York: Association for Computing Machinery, 2019. ISBN 9781450371520. Citations on pages 40, 49, and 50.

JACCARD, P. The Distribution of the Flora in the Alpine Zone. **New Phytologist**, Wiley Online Library, v. 11, n. 2, p. 37–50, 1912. Citation on page 34.

JOHNSON, A. E. W.; POLLARD, T. J.; SHEN, L.; LEHMAN, L. H.; FENG, M.; GHASSEMI, M.; MOODY, B.; SZOLOVITS, P.; CELI, L. A.; MARK, R. G. MIMIC-III, a Freely Accessible Critical Care Database. **Scientific Data**, Nature Publishing Group, v. 3, n. 160035, p. 1–9, 2016. Citation on page 37.

JOHNSON, J.; DOUZE, M.; JÉGOU, H. Billion-Scale Similarity Search with GPUs. **IEEE Transactions on Big Data**, IEEE, 2019. Citation on page 25.

KANDEL, S.; PAEPCKE, A.; HELLERSTEIN, J.; HEER, J. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In: **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2011. (CHI '11), p. 3363–3372. ISBN 9781450302289. Available: <https://doi.org/10.1145/1978942.1979444>. Citation on page 28.

KASTER, D. S.; BUGATTI, P. H.; PONCIANO-SILVA, M.; TRAINA, A. J. M.; MARQUES, P. M. A.; SANTOS, A. C.; TRAINA-JR., C. MedFMI-SiR: A Powerful DBMS Solution for Large-Scale Medical Image Retrieval. In: BOHM, C.; KHURI, S.; LHOTSKA, L.; PISANTI, N. (Ed.). **2nd International Conference on Information Technology in Bio- and Medical Informatics**. Heidelberg: Springer, 2011, (LNCS, v. 6865). p. 16–30. ISBN 978-3-642-23207-7. Citations on pages 38 and 92.

KINGMA, D. P.; WELLING, M. An Introduction to Variational Autoencoders. **Foundations and Trends® in Machine Learning**, v. 12, n. 4, p. 307–392, 2019. ISSN 1935-8237. Available: <http://dx.doi.org/10.1561/2200000056>. Citation on page 130.

KOLAHI, S.; LAKSHMANAN, L. V. S. On Approximating Optimum Repairs for Functional Dependency Violations. In: **Proceedings of the 12th International Conference on Database Theory**. New York, NY, USA: Association for Computing Machinery, 2009. (ICDT '09), p. 53–62. Available: <https://doi.org/10.1145/1514894.1514901>. Citation on page 40.

KOLLER, D.; FRIEDMAN, N. **Probabilistic Graphical Models: Principles and Techniques**. Cambridge: The MIT Press, 2009. ISBN 0262013193. Citation on page 42.

KOSUB, S. A Note on the Triangle Inequality for the Jaccard Distance. **Pattern Recognition Letters**, Elsevier, v. 120, p. 36–38, 2019. Citation on page 34.

KOUDAS, N.; SAHA, A.; SRIVASTAVA, D.; VENKATASUBRAMANIAN, S. Metric Functional Dependencies. In: **Proceedings of the 25th IEEE International Conference on Data Engineering**. USA: IEEE Computer Society, 2009. (ICDE '09), p. 1275–1278. ISBN 9780769535456. Available: <https://doi.org/10.1109/ICDE.2009.219>. Citation on page 39.

KOUDAS, N.; SARAWAGI, S.; SRIVASTAVA, D. Record Linkage: Similarity Measures and Algorithms. In: **Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2006. (SIGMOD '06), p. 802–803. ISBN 1595934340. Available: <https://doi.org/10.1145/1142473.1142599>. Citation on page 41.

KRISHNAN, S.; WANG, J.; WU, E.; FRANKLIN, M. J.; GOLDBERG, K. ActiveClean: Interactive Data Cleaning for Statistical Modeling. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 9, n. 12, p. 948–959, Aug. 2016. ISSN 2150-8097. Available: <https://doi.org/10.14778/2994509.2994514>. Citations on pages 26, 45, and 50.

KULLBACK, S.; LEIBLER, R. A. On Information and Sufficiency. **The Annals of Mathematical Statistics**, JSTOR, v. 22, n. 1, p. 79–86, 1951. Citation on page 35.

KUMAR, A.; KIM, J.; CAI, W.; FULHAM, M.; FENG, D. Content-Based Medical Image Retrieval: A Survey of Applications to Multidimensional and Multimodality Data. **Journal of digital imaging**, Springer, v. 26, n. 6, p. 1025–1039, 2013. Citations on pages 36 and 37.

KUNDU, M. K.; CHOWDHURY, M.; DAS, S. Interactive Radiographic Image Retrieval System. **Computer Methods and Programs in Biomedicine**, Elsevier, v. 139, p. 209–220, 2017. Citation on page 38.

KUROSE, J.; ROSS, K. **Computer Networking: A Top-Down Approach**. 7th. ed. London: Pearson, 2016. ISBN 9780133594140. Citations on pages 84 and 85.

KYTE, T.; KUHN, D. **Expert Oracle Database Architecture**. 3rd. ed. Berkeley: Apress, 2014. ISBN 978-1-4302-6299-2. Citation on page 47.

LAHDENMAKI, T.; LEACH, M. **Relational Database Index Design and the Optimizers: DB2, Oracle, SQL Server, et al.** Hoboken: John Wiley & Sons, 2005. Citation on page 47.

LEHMANN, T. M.; GULD, M. O.; THIES, C.; FISCHER, B.; SPITZER, K.; KEYSERS, D.; NEY, H.; KOHNEN, M.; SCHUBERT, H.; WEIN, B. B. Content-Based Image Retrieval in Medical Applications. **Methods of Informatics in Medicine**, v. 43, n. 4, p. 354–361, 2004. Citation on page 38.

LEVANDOWSKY, M.; WINTER, D. Distance between Sets. **Nature**, Springer, v. 234, n. 5323, p. 34–35, 1971. Citation on page 34.

LIPKUS, A. H. A Proof of the Triangle Inequality for the Tanimoto Distance. **Journal of Mathematical Chemistry**, Springer, v. 26, n. 1-3, p. 263–265, 1999. Citation on page 34.

LIU, J.; LI, J.; LIU, C.; CHEN, Y. Discover Dependencies from Data — A Review. **IEEE Transactions on Knowledge and Data Engineering**, IEEE Educational Activities Department, USA, v. 24, n. 2, p. 251–264, Feb. 2012. ISSN 1041-4347. Available: <https://doi.org/10.1109/TKDE.2010.197>. Citation on page 39.

LIU, Y.; ZHANG, D.; LU, G.; MA, W.-Y. A Survey of Content-Based Image Retrieval with High-Level Semantics. **Pattern Recognition**, Elsevier, New York, v. 40, n. 1, p. 262–282, Jan. 2007. ISSN 0031-3203. Citation on page 101.

LIU, Z. H.; HAMMERSCHMIDT, B.; MCMAHON, D. JSON Data Management: Supporting Schema-less Development in RDBMS. In: **41st ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2014. (SIGMOD '14), p. 1247–1258. ISBN 978-1-4503-2376-5. Citation on page 111.

LIU, Z. H.; HAMMERSCHMIDT, B.; MCMAHON, D.; LIU, Y.; CHANG, H. J. Closing the Functional and Performance Gap Between SQL and NoSQL. In: **43rd ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2016. (SIGMOD '16), p. 227–238. ISBN 978-1-4503-3531-7. Citation on page 111.

LONG, L. R.; ANTANI, S.; DESERNO, T. M.; THOMA, G. R. Content-Based Image Retrieval in Medicine: Retrospective Assessment, State of the Art, and Future Directions. **International Journal of Healthcare Information Systems and Informatics**, v. 4, n. 1, p. 1–16, 2009. Citation on page 101.

LORICA, B.; NATHAN, P. **AI Adoption in the Enterprise**. 1. ed. Sebastopol: O'Reilly Media, Inc., 2019. Citation on page 26.

MA, L.; LIU, X.; GAO, Y.; ZHAO, Y.; ZHAO, X.; ZHOU, C. A New Method of Content Based Medical Image Retrieval and Its Applications to CT Imaging Sign Retrieval. **Journal of Biomedical Informatics**, Elsevier, v. 66, p. 148–158, 2017. Citation on page 38.

MAYFIELD, C.; NEVILLE, J.; PRABHAKAR, S. ERACER: A Database Approach for Statistical Inference and Data Cleaning. In: **Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2010. (SIGMOD '10), p. 75–86. ISBN 9781450300322. Available: <https://doi.org/10.1145/1807167.1807178>. Citations on pages 41 and 50.

MEIER, A.; KAUFMANN, M. **SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management**. Wiesbaden: Springer Vieweg, 2019. Citation on page 25.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient Estimation of Word Representations in Vector Space. In: BENGIO, Y.; LECUN, Y. (Ed.). **Proceedings of the 1st International Conference on Learning Representations**. [s.n.], 2013. (ICLR '13). Available: <http://arxiv.org/abs/1301.3781>. Citation on page 54.

MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G.; DEAN, J. Distributed Representations of Words and Phrases and Their Compositionality. In: **Proceedings of the 26th International Conference on Neural Information Processing Systems — Volume 2**. Red Hook, NY, USA: Curran Associates Inc., 2013. (NIPS '13), p. 3111–3119. Citation on page 54.

MOLINARO, A.; GRAAF, R. de. **SQL Cookbook: Query Solutions and Techniques for All SQL Users**. 2. ed. Sebastopol: O'Reilly Media, 2020. Citation on page 25.

MOREIRA, I. C.; BACELAR-SILVA, G. M.; RODRIGUES, P. P. Compliance of Publicly Available Mammographic Databases with Established Case Selection and Annotation Requirements. In: **5th International Conference on Health Informatics**. [S.l.: s.n.], 2012. p. 337–340. Citation on page 101.

MULLER, H.; KALPATHY–CRAMER, J.; CAPUTO, B.; SYEDA-MAHMOOD, T.; WANG, F. Overview of the First Workshop on Medical Content–Based Retrieval for Clinical Decision Support at MICCAI 2009. In: **Proceedings of the 1st MICCAI International Conference on Medical Content-Based Retrieval for Clinical Decision Support**. Berlin, Heidelberg: Springer-Verlag, 2009. (MCBR-CDS '09), p. 1–17. ISBN 3642117686. Available: <https://doi.org/10.1007/978-3-642-11769-5_1>. Citation on page 37.

MULLER, H.; ZHOU, X.; DEPEURSINGE, A.; PITKANEN, M.; IAVINDRASANA, J.; GEISSBUHLER, A. Medical Visual Information Retrieval: State of the Art and Challenges Ahead. In: **Proceedings of the 8th IEEE International Conference on Multimedia and Expo**. [S.l.]: IEEE, 2007. p. 683–686. Citation on page 37.

MURPHY, K. P. **Machine Learning: A Probabilistic Perspective**. Cambridge: The MIT Press, 2012. ISBN 0262018020. Citation on page 49.

NAUMANN, F.; HERSCHEL, M. **An Introduction to Duplicate Detection**. San Rafael: Morgan & Claypool Publishers, 2010. ISBN 1608452204. Citation on page 41.

NESSO-JR., M. R.; CAZZOLATO, M. T.; SCABORA, L. C.; OLIVEIRA, P. H.; SPADON, G.; SOUZA, J. A. de; OLIVEIRA, W. D.; CHINO, D. Y. T.; RODRIGUES-JR., J. F.; TRAINA, A. J. M.; TRAINA-JR., C. RAFIKI: Retrieval-Based Application for Imaging and Knowledge Investigation. In: **31st IEEE International Symposium on Computer-Based Medical Systems**. [S.l.: s.n.], 2018. (CBMS '18), p. 71–76. Citations on pages 103 and 132.

OLIVEIRA, P. H.; FRAIDEINBERZE, A. C.; LAVERDE, N. A.; GUALDRON, H.; GONZAGA, A. S.; FERREIRA, L. D.; OLIVEIRA, W. D.; RODRIGUES-JR., J. F.; CORDEIRO, R. L. F.; TRAINA-JR., C.; TRAINA, A. J. M.; SOUSA, E. P. M. On the Support of a Similarity-Enabled Relational Database Management System in Civilian Crisis Situations. In: **18th International Conference on Enterprise Information Systems**. Setúbal, Portugal: SCITEPRESS, 2016. (ICEIS '16), p. 119–126. ISBN 9789897581878. Available: <https://doi.org/10.5220/0005816701190126>. Citations on pages 103 and 132.

OLIVEIRA, P. H.; KASTER, D. S.; TRAINA-JR., C.; ILYAS, I. F. Batchwise Probabilistic Incremental Data Cleaning. **CoRR**, abs/2011.04730, 2020. Available: <https://arxiv.org/abs/2011.04730>. Citations on pages 29 and 131.

OLIVEIRA, P. H.; SCABORA, L. C.; CAZZOLATO, M. T.; OLIVEIRA, W. D.; TRAINA, A. J. M.; TRAINA-JR., C. Efficiently Indexing Multiple Repositories of Medical Image Databases. In: **30th IEEE International Symposium on Computer-Based Medical Systems**. [S.l.: s.n.], 2017. (CBMS '17), p. 286–291. Citations on pages 29, 102, and 131.

OLIVEIRA, P. H.; SCABORA, L. C.; CAZZOLATO, M. T.; BEDO, M. V. N.; TRAINA, A. J. M.; TRAINA-JR., C. MAMMOSET: An Enhanced Dataset of Mammograms. In: **Proceedings of the Satellite Events of the 32nd Brazilian Symposium on Databases**. Porto Alegre, RS, Brazil: SBC, 2017. (DSW '17), p. 256–266. Citations on pages 104 and 131.

OLIVEIRA, P. H.; SCABORA, L. C.; CAZZOLATO, M. T.; OLIVEIRA, W. D.; PAIXÃO, R. S.; TRAINA, A. J. M.; TRAINA-JR., C. Employing Domain Indexes to Efficiently Query Medical Data from Multiple Repositories. **IEEE Journal of Biomedical and Health Informatics**, IEEE, v. 23, n. 6, p. 2220–2229, Nov. 2019. ISSN 2168-2208. Citations on pages 29, 82, 96, 97, 99, 102, 104, 105, 106, 107, 108, and 131.

OLIVEIRA, P. H.; SCABORA, L. C.; KASTER, D. S.; TRAINA-JR., C. One Index to Dominate Them All: Domain Indexes for Improving Queries Across Multiple Tables. In: **32nd ACM Symposium on Applied Computing**. New York: ACM, 2017. (SAC '17), p. 900–905. Citations on pages 29, 84, 85, 87, 89, 90, 91, 93, 101, 102, and 131.

OLIVEIRA, P. H.; TRAINA-JR., C.; KASTER, D. S. CLAP, ACIR and SCOOP: Novel Techniques for Improving the Performance of Dynamic Metric Access Methods. **Information Systems**, v. 72, p. 117–135, 2017. Available: <https://doi.org/10.1016/j.is.2017.10.003>. Citation on page 132.

OUSSOUS, A.; BENJELLOUN, F.-Z.; LAHCEN, A. A.; BELFKIH, S. Big Data Technologies: A Survey. **Journal of King Saud University - Computer and Information Sciences**, Elsevier, v. 30, n. 4, p. 431–448, 2018. Citation on page 25.

PAPENBROCK, T.; NAUMANN, F. A Hybrid Approach to Functional Dependency Discovery. In: **Proceedings of the 2016 International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2016. (SIGMOD '16), p. 821–833. ISBN 9781450335317. Available: <https://doi.org/10.1145/2882903.2915203>. Citation on page 39.

PAPOTTI, P.; CHU, X.; ILYAS, I. F. Holistic Data Cleaning: Putting Violations into Context. In: **Proceedings of the 29th International Conference on Data Engineering**. Los Alamitos, CA, USA: IEEE Computer Society, 2013. (ICDE '13), p. 458–469. ISBN 9781467349093. Available: <https://doi.org/10.1109/ICDE.2013.6544847>. Citations on pages 41 and 50.

PASQUALIN, D.; MACIEL, E.; BONA, L. C. E. de; OLIVEIRA, L.; SUNYE, M. Dados de Monitoramento de Projetos de Inclusão Digital do Ministério da Ciência, Tecnologia, Inovações e Comunicações. In: **Proceedings of the Satellite Events of the 32nd Brazilian Symposium on Databases**. Porto Alegre, RS, Brazil: SBC, 2017. (DSW '17), p. 193–202. Citation on page 119.

RASCHKA, S.; MIRJALILI, V. **Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2**. 3. ed. Birmingham: Packt Publishing, 2019. Citation on page 25.

REKATSINAS, T.; CHU, X.; ILYAS, I. F.; RÉ, C. HoloClean: Holistic Data Repairs with Probabilistic Inference. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 10, n. 11, p. 1190–1201, Aug. 2017. ISSN 2150-8097. Available: <https://doi.org/10.14778/3137628.3137631>. Citations on pages 26, 41, 42, 43, 49, 50, and 51.

REZIG, E. K. **Online Data Cleaning**. Phd Thesis (PhD Thesis) — Purdue University, West Lafayette, IN, USA, Aug. 2018. Citations on pages 44 and 46.

REZIG, E. K.; CAO, L.; STONEBRAKER, M.; SIMONINI, G.; TAO, W.; MADDEN, S.; OUZZANI, M.; TANG, N.; ELMAGARMID, A. K. Data Civilizer 2.0: A Holistic Framework for Data Preparation and Analytics. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 12, n. 12, p. 1954–1957, Aug. 2019. ISSN 2150-8097. Available: <https://doi.org/10.14778/3352063.3352108>. Citation on page 26.

ROGERS, D. J.; TANIMOTO, T. T. A Computer Program for Classifying Plants. **Science**, JSTOR, v. 132, n. 3434, p. 1115–1118, 1960. Citation on page 34.

ROH, Y.; HEO, G.; WHANG, S. E. A Survey on Data Collection for Machine Learning: A Big Data – AI Integration Perspective. **CoRR**, abs/1811.03402, p. 1–20, 2018. Available: <https://arxiv.org/abs/1811.03402>. Citations on pages 42 and 51.

SAMET, H. **Foundations of Multidimensional and Metric Data Structures**. Burlington: Morgan Kaufmann, 2006. Citation on page 33.

SCABORA, L. C.; OLIVEIRA, P. H.; KASTER, D. S.; TRAINA, A. J. M.; TRAINA-JR., C. Relational Graph Data Management on the Edge: Grouping Vertices' Neighborhood with Edge-k. In: HARA, C. S. (Ed.). **Proceedings of the 32nd Brazilian Symposium on Databases**. SBC, 2017. p. 124–135. Available: <http://sbbd.org.br/2017/wp-content/uploads/sites/3/2018/02/p124-135.pdf>. Citation on page 132.

SCABORA, L. C.; OLIVEIRA, P. H.; SPADON, G.; KASTER, D. S.; RODRIGUES-JR., J. F.; TRAINA, A. J. M.; TRAINA-JR., C. Cutting-Edge Relational Graph Data Management with Edge-k: From One to Multiple Edges in the Same Row. **Journal of Information and Data Management**, v. 9, n. 1, p. 20–35, 2018. Available: <https://periodicos.ufmg.br/index.php/jidm/article/view/413>. Citation on page 132.

SCABORA, L. C.; SPADON, G.; OLIVEIRA, P. H.; RODRIGUES-JR., J. F.; TRAINA-JR., C. Enhancing Recursive Graph Querying on RDBMS with Data Clustering Approaches. In: HUNG, C.; CERNÝ, T.; SHIN, D.; BECHINI, A. (Ed.). **Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing**. ACM, 2020. p. 404–411. Available: <https://doi.org/10.1145/3341105.3375770>. Citation on page 132.

SCHELTER, S.; BIESSMANN, F.; JANUSCHOWSKI, T.; SALINAS, D.; SEUFERT, S.; SZARVAS, G. On Challenges in Machine Learning Model Management. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, v. 41, n. 4, p. 5–13, 2018. Available: <http://sites.computer.org/debull/A18dec/p5.pdf>. Citation on page 50.

SCHELTER, S.; LANGE, D.; SCHMIDT, P.; CELIKEL, M.; BIESSMANN, F.; GRAFBERGER, A. Automating Large-Scale Data Quality Verification. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 11, n. 12, p. 1781–1794, Aug. 2018. ISSN 2150-8097. Available: <https://doi.org/10.14778/3229863.3229867>. Citation on page 45.

SCHIRMER, P.; PAPENBROCK, T.; KRUSE, S.; NAUMANN, F.; HEMPFING, D.; MAYER, T.; NEUSCHäFER-RUBE, D. DynFD: Functional Dependency Discovery in Dynamic Datasets. In: **Proceedings of the 22nd International Conference on Extending Database Technology**. Konstanz, Germany: OpenProceedings.org, 2019. (EDBT '19), p. 253–264. Citation on page 45.

SCHUMACHER, T.; STROHMAIER, M.; LEMMERICH, F. A Comparative Evaluation of Quantification Methods. **CoRR**, abs/2103.03223, 2021. Available: <https://arxiv.org/abs/2103.03223>. Citation on page 35.

SEDGHI, S.; SANDERSON, M.; CLOUGH, P. How Do Health Care Professionals Select Medical Images They Need? In: **Aslib Proceedings**. [S.l.]: Emerald Group Publishing Limited, 2012. Citation on page 36.

SHAABANI, N.; MEINEL, C. Incremental Discovery of Inclusion Dependencies. In: **Proceedings of the 29th International Conference on Scientific and Statistical Database Management**. New York, NY, USA: Association for Computing Machinery, 2017. (SSDBM '17). ISBN

9781450352826. Available: <https://doi.org/10.1145/3085504.3085506>. Citations on pages 26 and 45.

____. Incrementally Updating Unary Inclusion Dependencies in Dynamic Data. **Distributed and Parallel Databases**, Kluwer Academic Publishers, USA, v. 37, n. 1, p. 133–176, Mar. 2019. ISSN 0926-8782. Available: <https://doi.org/10.1007/s10619-018-7233-5>. Citations on pages 26 and 45.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding Machine Learning: From Theory to Algorithms**. Cambridge: Cambridge University Press, 2014. Citation on page 43.

SHIN, J.; WU, S.; WANG, F.; SA, C. D.; ZHANG, C.; RÉ, C. Incremental Knowledge Base Construction Using DeepDive. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 8, n. 11, p. 1310–1321, Jul. 2015. ISSN 2150-8097. Available: <https://doi.org/10.14778/2809974.2809991>. Citation on page 43.

SONG, Y.; CAI, W.; EBERL, S.; FULHAM, M. J.; FENG, D. A Content-Based Image Retrieval Framework for Multi-Modality Lung Images. In: **23rd IEEE International Symposium on Computer-Based Medical Systems**. Washington: IEEE, 2010. (CBMS '10), p. 285–290. ISBN 978-1-4244-9167-4. Citation on page 38.

SPADON, G.; SCABORA, L. C.; ARAUJO, M. V. S.; OLIVEIRA, P. H.; MACHADO, B. B.; SOUSA, E. P. M. de; TRAINA-JR., C.; RODRIGUES-JR., J. F. Complex Network Tools to Understand the Behavior of Criminality in Urban Areas. In: **Proceedings of the 14th International Conference on Information Technology - New Generations**. [S.l.]: Springer, 2017. p. 1–6. Citation on page 132.

SPADON, G.; SCABORA, L. C.; OLIVEIRA, P. H.; ARAUJO, M. V. S.; MACHADO, B. B.; SOUSA, E. P. M. de; TRAINA-JR., C.; RODRIGUES-JR., J. F. Behavioral Characterization of Criminality Spread in Cities. In: KOUMOUTSAKOS, P.; LEES, M.; KRZHIZHANOVSKAYA, V. V.; DONGARRA, J. J.; SLOOT, P. M. A. (Ed.). **Proceedings of the 17th International Conference on Computational Science**. Elsevier, 2017. (Procedia Computer Science, v. 108), p. 2537–2541. Available: <https://doi.org/10.1016/j.procs.2017.05.118>. Citation on page 132.

SPEROTTO, A.; SADRE, R.; VLIET, F.; PRAS, A. A Labeled Data Set for Flow-Based Intrusion Detection. In: **9th IEEE International Workshop on IP Operations and Management**. Berlin, Heidelberg: Springer-Verlag, 2009. (IPOM '09), p. 39–50. ISBN 978-3-642-04967-5. Available: <http://dx.doi.org/10.1007/978-3-642-04968-2_4>. Citation on page 84.

STONEBRAKER, M.; BRUCKNER, D.; ILYAS, I. F.; BESKALES, G.; CHERNIACK, M.; ZDONIK, S. B.; PAGAN, A.; XU, S. Data Curation at Scale: The Data Tamer System. In: **Proceedings of the 6th Conference on Innovative Data Systems Research**. [S.l.: s.n.], 2013. (CIDR '13). Citation on page 28.

STRATE, J.; FRITCHEY, G. **Expert Performance Indexing in SQL Server**. 2nd. ed. New York: Apress, 2015. 432 p. ISBN 9781484211199. Available: <http://www.apress.com/9781484211199>. Citation on page 47.

SUBRAMANIAM, S.; JAYANTHI, K. B.; RAJASEKARAN, C.; KUCHELAR, R. Deep Learning Architectures for Medical Image Segmentation. In: IEEE. **Proceedings of the 33rd International Symposium on Computer-Based Medical Systems**. [S.l.], 2020. (CBMS '20), p. 579–584. Citation on page 38.

SUCKLING, J.; PARKER, P.; DANCE, D. R.; ASTLEY, S.; HUTT, I.; BOGGIS, C.; RICKETTS, I. The Mammographic Image Analysis Society Digital Mammogram Database. **Excerpta Medica, International Congress Series**, Elsevier, v. 1069, p. 375–378, 1994. Citation on page 104.

SZLICHTA, J.; GODFREY, P.; GOLAB, L.; KARGAR, M.; SRIVASTAVA, D. Effective and Complete Discovery of Order Dependencies via Set-Based Axiomatization. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 10, n. 7, p. 721–732, Mar. 2017. ISSN 2150-8097. Available: <https://doi.org/10.14778/3067421.3067422>. Citation on page 39.

____. Erratum for Discovering Order Dependencies through Order Compatibility (EDBT 2019). In: BONIFATI, A.; ZHOU, Y.; SALLES, M. A. V.; BÖHM, A.; OLTEANU, D.; FLETCHER, G. H. L.; KHAN, A.; YANG, B. (Ed.). **Proceedings of the 23rd International Conference on Extending Database**. OpenProceedings.org, 2020. (EDBT '20), p. 659–663. Available: <https://doi.org/10.5441/002/edbt.2020.88>. Citation on page 39.

TAHARA, D.; DIAMOND, T.; ABADI, D. J. Sinew: A SQL System for Multi-Structured Data. In: **41st ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2014. (SIGMOD '14), p. 815–826. ISBN 978-1-4503-2376-5. Citation on page 111.

TANIMOTO, T. T. **Elementary Mathematical Theory of Classification and Prediction**. [S.l.], 1958. Citation on page 34.

TRAINA, A. J. M.; TRAINA-JR., C.; BALAN, A. G. R.; RIBEIRO, M. X.; BUGATTI, P. H.; WATANABE, C. Y. V.; AZEVEDO-MARQUES, P. M. Feature Extraction and Selection for Decision Making. In: DESERNO, T. M. (Ed.). **Biomedical Image Processing**. Heidelberg: Springer, 2011, (Biological and Medical Physics, Biomedical Engineering). p. 197–223. Citations on pages 36 and 103.

TRAINA-JR., C.; TRAINA, A. J. M.; FALOUTSOS, C.; SEEGER, B. Fast Indexing and Visualization of Metric Data Sets Using Slim-trees. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, Piscataway, v. 14, n. 2, p. 244–260, 2002. ISSN 1041-4347. Citation on page 36.

TRIEU, V.-H. Getting Value from Business Intelligence Systems: A Review and Research Agenda. **Decision Support Systems**, Elsevier, v. 93, p. 111–124, 2017. Citation on page 25.

URBANO, R. **Oracle Database Administrator's Guide, 12c**. [S.l.], 2016. Citations on pages 46 and 78.

VERBEKE, W.; BAESENS, B.; BRAVO, C. **Profit-Driven Business Analytics: A Practitioner's Guide to Transforming Big Data into Added Value**. Hoboken: John Wiley & Sons, 2017. Citations on pages 25 and 26.

VOLKOVS, M.; CHIANG, F.; SZLICHTA, J.; MILLER, R. J. Continuous Data Cleaning. In: **Proceedings of the 30th International Conference on Data Engineering**. Los Alamitos, CA, USA: IEEE Computer Society, 2014. (ICDE '14, v. 1), p. 244–255. Available: <https://doi.ieeecomputersociety.org/10.1109/ICDE.2014.6816655>. Citations on pages 26, 44, 45, 49, and 50.

WANG, J.; KRISHNAN, S.; FRANKLIN, M. J.; GOLDBERG, K.; KRASKA, T.; MILO, T. A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data. In: **Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2014. (SIGMOD '14), p. 469–480. ISBN 9781450323765. Available: <https://doi.org/10.1145/2588555.2610505>. Citation on page 45.

WEYENBERG, G.; YOSHIDA, R. Reconstructing the Phylogeny: Computational Methods. In: **Algebraic and Discrete Mathematical Methods for Modern Biology**. [S.l.]: Elsevier, 2015. p. 293–319. Citation on page 34.

WHANG, S. E.; GARCIA-MOLINA, H. Entity Resolution with Evolving Rules. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 3, n. 1-2, p. 1326–1337, Sep. 2010. ISSN 2150-8097. Available: <https://doi.org/10.14778/1920841.1921004>. Citation on page 45.

WILSON, D. R.; MARTINEZ, T. R. Improved Heterogeneous Distance Functions. **Journal of Artificial Intelligence Research**, AI Access Foundation, USA, v. 6, n. 1, p. 1–34, 1997. ISSN 1076-9757. Citation on page 34.

WU, R.; ZHANG, A.; ILYAS, I. F.; REKATSINAS, T. Attention-Based Learning for Missing Data Imputation in HoloClean. **Proceedings of Machine Learning and Systems**, p. 307–325, 2020. Citation on page 41.

XUE, Z.; LONG, L. R.; ANTANI, S.; THOMA, G. R. Spine X-Ray Image Retrieval Using Partial Vertebral Boundaries. In: **24th IEEE International Symposium on Computer-Based Medical Systems**. [S.l.: s.n.], 2011. (CBMS '11), p. 1–6. ISSN 1063-7125. Citation on page 38.

YAKOUT, M.; BERTI-ÉQUILLE, L.; ELMAGARMID, A. K. Don't Be SCAREd: Use SCalable Automatic REpairing with Maximal Likelihood and Bounded Changes. In: **Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2013. (SIGMOD '13), p. 553–564. ISBN 9781450320375. Available: <https://doi.org/10.1145/2463676.2463706>. Citations on pages 41 and 50.

YAQOOB, I.; HASHEM, I. A. T.; GANI, A.; MOKHTAR, S.; AHMED, E.; ANUAR, N. B.; VASILAKOS, A. V. Big Data: From Beginning to Future. **International Journal of Information Management**, Elsevier, v. 36, n. 6, p. 1231–1247, 2016. Citation on page 25.

YEUNG, R. W. **Information Theory and Network Coding**. New York: Springer Science & Business Media, 2008. Citations on pages 31, 32, and 35.

YU, Z.; CHU, X. PIClean: A Probabilistic and Interactive Data Cleaning System. In: **Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2019. (SIGMOD '19), p. 2021–2024. ISBN 9781450356435. Available: <https://doi.org/10.1145/3299869.3320214>. Citation on page 46.

ZEZULA, P.; AMATO, G.; DOHNAL, V.; BATKO, M. **Similarity Search: The Metric Space Approach**. 1st. ed. New York: Springer, 2006. (Advances in Database Systems, v. 32). ISBN 978-0387-29146-8. Citations on pages 33, 34, and 36.

ZHANG, C.; RÉ, C. DimmWitted: A Study of Main-Memory Statistical Analytics. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 7, n. 12, p. 1283–1294, Aug. 2014. ISSN 2150-8097. Available: <https://doi.org/10.14778/2732977.2733001>. Citation on page 43.

APPENDIX

A

INCREMENTAL CONDITIONAL ENTROPY

We formally present how to incrementally maintain the conditional entropy of each pair of attributes in an evolving dataset.

Definition 1. Let X and Y be two samples. Define their *conditional entropy* as

$$H(X|Y) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{p(x,y)}{p(y)}, \tag{A.1}$$

where

- $p(x,y) = \frac{z}{n}$, n being the total number of tuples and z being the number of times values $x \in X$ and $y \in Y$ co-occur,
- $p(y) = \frac{w}{n}$, w being the number of times value $y \in Y$ occurs.

Then, the conditional entropy equation can be rewritten as

$$H(X|Y) = -\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n}\log\frac{\frac{z_{ij}}{n}}{\frac{w_j}{n}}. \tag{A.2}$$

Lemma 1. Let X and Y be two samples and H_n be their conditional entropy given n tuples. For any positive integer D,

$$-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n+D}\log\frac{\frac{z_{ij}}{n+D}}{\frac{w_j}{n+D}} = \frac{n}{n+D}\cdot H_n. \tag{A.3}$$

Proof. Write $\frac{z_{ij}}{n+D} = \frac{n(z_{ij})}{n(n+D)}$ and $\frac{w_j}{n+D} = \frac{n(w_j)}{n(n+D)}$.

Then, we have

$$-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n+D}\log\frac{\frac{z_{ij}}{n+D}}{\frac{w_j}{n+D}} = \frac{n}{n+D}\left(-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n}\log\frac{\frac{z_{ij}}{n}}{\frac{w_j}{n}}\right) = \frac{n}{n+D}\cdot H_n.$$

□

The following claim provides an update formula for when a new tuple arrives.

Claim 1. Let H_n be the current samples' conditional entropy and consider an incoming tuple having new attribute values $x' \notin X$ and $y' \notin Y$. Assume X' and Y' are the up-to-date versions of samples X and Y, that is, $X' = X \cup \{x'\}$ and $Y' = Y \cup \{y'\}$. Then, we have

$$H_{n+1} = \frac{n}{n+1}\cdot H_n. \tag{A.4}$$

Proof. By definition we have

$$H_{n+1} = -\sum_{i=1}^{|X'|}\sum_{j=1}^{|Y'|}\frac{z_{ij}}{n+1}\log\frac{\frac{z_{ij}}{n+1}}{\frac{w_j}{n+1}}. \tag{A.5}$$

Considering x' and y' as attribute values in the incoming tuple, this equation can be rewritten as

$$H_{n+1} = \left(-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n+1}\log\frac{\frac{z_{ij}}{n+1}}{\frac{w_j}{n+1}}\right) - p(x',y')\log\frac{p(x',y')}{p(y')}.$$

The new tuple accounts for 1 co-occurrence of values $x' \in X'$ and $y' \in Y'$, hence $p(x',y') = \frac{1}{n+1}$. Furthermore, it accounts for 1 occurrence of value $y' \in Y'$, hence $p(y') = \frac{1}{n+1}$. So, we have

$$\begin{aligned} H_{n+1} &= \left(-\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n+1}\log\frac{\frac{z_{ij}}{n+1}}{\frac{w_j}{n+1}}\right) - \frac{1}{n+1}\log\frac{\frac{1}{n+1}}{\frac{1}{n+1}} \\ &= -\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\frac{z_{ij}}{n+1}\log\frac{\frac{z_{ij}}{n+1}}{\frac{w_j}{n+1}}. \end{aligned}$$

Therefore, following from Lemma 1, we have

$$H_{n+1} = \frac{n}{n+1}\cdot H_n. \tag{A.6}$$

□

The following theorem generalizes Claim 1 by providing a formula for the conditional entropy of concatenated samples, given the conditional entropies H_n over n tuples and G_m over m tuples.

Theorem 1. Let $X = \{x_i\}_{i=1}^{|X|}$, $Y = \{y_i\}_{i=1}^{|Y|}$, $A = \{a_i\}_{i=1}^{|A|}$, and $B = \{b_i\}_{i=1}^{|B|}$ be samples. Suppose X is to be concatenated with A, and Y is to be concatenated with B. Samples X and A can either be disjoint or not, whereas $Y \cap B = \emptyset$. Assume n is the number of tuples having attribute values from X and Y, and m is the number of tuples having attribute values from A and B. Let H_n be the conditional entropy of samples X and Y, and let G_m be the conditional entropy of samples A and B. Finally, let $\hat{X} = X \cup A$ and $\hat{Y} = Y \cup B$ be the concatenated samples. Define the variables

- z as the number of times values $x \in X$ and $y \in Y$ co-occur,
- w as the number of times value $y \in Y$ occurs,
- p as the number of times values $a \in A$ and $b \in B$ co-occur,
- q as the number of times value $b \in B$ occurs,
- $\hat{z}$ as the number of times values $\hat{x} \in \hat{X}$ and $\hat{y} \in \hat{Y}$ co-occur,
- $\hat{w}$ as the number of times value $\hat{y} \in \hat{Y}$ occurs,

all of which are greater than 0. Then, we have

$$H_{n+m} = \frac{n}{n+m} \cdot H_n + \frac{m}{n+m} \cdot G_m. \tag{A.7}$$

Proof. Similarly as before, we have

$$\begin{aligned} H_{n+m} &= -\sum_{i=1}^{|\hat{X}|} \sum_{j=1}^{|\hat{Y}|} \frac{\hat{z}_{ij}}{n+m} \log \frac{\frac{\hat{z}_{ij}}{n+m}}{\frac{\hat{w}_j}{n+m}} \\ &= -\sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} \frac{z_{ij}}{n+m} \log \frac{\frac{z_{ij}}{n+m}}{\frac{w_j}{n+m}} - \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \frac{p_{ij}}{n+m} \log \frac{\frac{p_{ij}}{n+m}}{\frac{q_j}{n+m}} \\ &= \frac{n}{n+m} \cdot H_n + \frac{m}{n+m} \cdot G_m, \end{aligned}$$

where the last equality follows from applying Lemma 1 twice. □

The following theorem provides a formula for updating the conditional entropy when a new sample has only unseen attribute values and another new sample has existing attribute values.

Theorem 2. Let X, Y, A, and B be samples. Suppose X is to be concatenated with A, Y is to be concatenated with B, and consider $X \cap A = \emptyset$ and $B \subseteq Y$. Assume n existing tuples have attribute values from X and Y, and m incoming tuples have attribute values from A and B. Let Y^+ be the attribute values $y^+ \in Y \cap B$, whose frequencies will have increased after the samples' concatenation, and let B^* be the attribute values $b^* \in B \setminus Y^+$, which are all unseen values to be concatenated to the values $y \in Y$. Furthermore, let H_n be the conditional entropy of X and Y given n tuples. Define

- z^+ as the number of times values $x \in X$ and $y^+ \in Y^+$ co-occur in the n existing tuples, with $z^+ > 0$,
- w^+ as the number of times value $y^+ \in Y^+$ occurs in the n existing tuples, with $w^+ > 0$,
- p^+ as the number of times values $a \in A$ and $y^+ \in Y^+$ co-occur in the m new tuples, with $p^+ > 0$,
- q^+ as the number of times value $y^+ \in Y^+$ occurs in the m new tuples, with $q^+ > 0$,
- p^* as the number of times values $a \in A$ and $b^* \in B^*$ co-occur in the m new tuples, with $p^* \geq 0$,
- q^* as the number of times value $b^* \in B^*$ occurs in the m new tuples, with $q^* \geq 0$.

Then, the samples' conditional entropy H_{n+m} becomes

$$
\begin{aligned}
H_{n+m} = \frac{n}{n+m} \cdot H_n - \left(\sum_{i=1}^{|X|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} - \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+}{n+m}} \right) \\
- \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|Y^+|} \frac{p_{ij}^+}{n+m} \log \frac{\frac{p_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) - \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|B^*|} \frac{p_{ij}^*}{n+m} \log \frac{\frac{p_{ij}^*}{n+m}}{\frac{q_j^*}{n+m}} \right).
\end{aligned}
\tag{A.8}
$$

Proof. We first adjust the conditional entropy value H_n to the new number of tuples $n+m$, updating the existing amount of entropy related to values $y^+ \in Y^+$ so that it accounts for their frequencies in the m new tuples as well:

$$
H_n^* = \frac{n}{n+m} \cdot H_n - \left(\sum_{i=1}^{|X|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} - \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+}{n+m}} \right).
$$

Then, taking into account the m new tuples, we have

$$H_{n+m} = H_n^* - \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|Y^+|} \frac{p_{ij}^+}{n+m} \log \frac{\frac{p_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) - \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|B^*|} \frac{p_{ij}^*}{n+m} \log \frac{\frac{p_{ij}^*}{n+m}}{\frac{q_j^*}{n+m}} \right).$$

Hence, we have

$$H_{n+m} = \frac{n}{n+m} \cdot H_n - \left(\sum_{i=1}^{|X|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} - \frac{z_{ij}^+}{n+m} \log \frac{\frac{z_{ij}^+}{n+m}}{\frac{w_j^+}{n+m}} \right)$$

$$- \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|Y^+|} \frac{p_{ij}^+}{n+m} \log \frac{\frac{p_{ij}^+}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) - \left(\sum_{i=1}^{|A|} \sum_{j=1}^{|B^*|} \frac{p_{ij}^*}{n+m} \log \frac{\frac{p_{ij}^*}{n+m}}{\frac{q_j^*}{n+m}} \right).$$

□

The next theorem provides a formula for updating the conditional entropy when existing attribute values $x \in X$ and $y \in Y$ have had their frequencies increased, that is, when attribute values from both samples X and Y appear in the incoming tuples.

Theorem 3. Let X and Y be samples with attribute values from n existing tuples, and let A and B be samples with attribute values from m incoming tuples. The conditional entropy of X and Y given the n existing tuples is H_n. Let $X^+ \subset X$ and $Y^+ \subset Y$ be the sets of attribute values that appear both in the n existing tuples and in the m new tuples, and let $X^= = X \setminus X^+$ be the set of attribute values from X whose frequencies remained the same. Finally, let $A^* = A \setminus X^+$ and $B^* = B \setminus Y^+$ be the sets of unseen attribute values in the m new tuples. Define the variables

- $z^{=+}$ as the number of times values $x^= \in X^=$ and $y^+ \in Y^+$ co-occur in the n existing tuples,
- z^{++} as the number of times values $x^+ \in X^+$ and $y^+ \in Y^+$ co-occur in the n existing tuples,
- p^{++} as the number of times values $x^+ \in X^+$ and $y^+ \in Y^+$ co-occur in the m new tuples, with $p^{++} \geq 0$,
- p^{+*} as the number of times values $x^+ \in X^+$ and $b^* \in B^*$ co-occur in the m new tuples, with $p^{+*} > 0$ if $p^{++} = 0$ or $p^{+*} \geq 0$ otherwise,
- r^{*+} as the number of times values $a^* \in A^*$ and $y^+ \in Y^+$ co-occur in the m new tuples, with $r^{*+} > 0$ if $p^{++} = 0$ or $r^{*+} \geq 0$ otherwise,
- r^{**} as the number of times values $a^* \in A^*$ and $b^* \in B^*$ co-occur in the m new tuples, with $r^{**} = 0$ if $p^{++} = 0$ or $r^{**} \geq 0$ otherwise,

- w^+ as the number of times value $y^+ \in Y^+$ occurs in the n existing tuples,
- q^+ as the number of times value $y^+ \in Y^+$ occurs in the m new tuples,
- s^* as the number of times value $b^* \in B^*$ occurs in the m new tuples,

all of which are greater than 0, except when specified otherwise. Then, the conditional entropy H_{n+m} of concatenated samples $\hat{X} = X \cup A$ and $\hat{Y} = Y \cup B$ becomes

$$
\begin{aligned}
H_{n+m} = {} & \frac{n}{n+m} \cdot H_n \\
& - \left(\sum_{i=1}^{|X^=|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{=+}}{n+m} \log \frac{\frac{z_{ij}^{=+}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) + \left(\sum_{i=1}^{|X^=|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{=+}}{n+m} \log \frac{\frac{z_{ij}^{=+}}{n+m}}{\frac{w_j^+}{n+m}} \right) \\
& - \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{++} + p_{ij}^{++}}{n+m} \log \frac{\frac{z_{ij}^{++} + p_{ij}^{++}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) + \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{++}}{n+m} \log \frac{\frac{z_{ij}^{++}}{n+m}}{\frac{w_j^+}{n+m}} \right) \\
& - \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|B^*|} \frac{p_{ij}^{+*}}{n+m} \log \frac{\frac{p_{ij}^{+*}}{n+m}}{\frac{s_j^*}{n+m}} \right) \\
& - \left(\sum_{i=1}^{|A^*|} \sum_{j=1}^{|Y^+|} \frac{r_{ij}^{*+}}{n+m} \log \frac{\frac{r_{ij}^{*+}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) - \left(\sum_{i=1}^{|A^*|} \sum_{j=1}^{|B^*|} \frac{r_{ij}^{**}}{n+m} \log \frac{\frac{r_{ij}^{**}}{n+m}}{\frac{s_j^*}{n+m}} \right).
\end{aligned}
\tag{A.9}
$$

Proof. We begin by adjusting the conditional entropy value H_n to the new number of tuples $n+m$, as well as updating the existing amount of entropy of values $y^+ \in Y^+$ whose frequencies increased and that co-occur with values $x^= \in X^=$:

$$
H_n^* = \frac{n}{n+m} \cdot H_n - \left(\sum_{i=1}^{|X^=|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{=+}}{n+m} \log \frac{\frac{z_{ij}^{=+}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) + \left(\sum_{i=1}^{|X^=|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{=+}}{n+m} \log \frac{\frac{z_{ij}^{=+}}{n+m}}{\frac{w_j^+}{n+m}} \right).
$$

Then, we update the existing amount of entropy regarding values $x^+ \in X^+$ and $y^+ \in Y^+$:

$$
H_n^{**} = H_n^* - \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{++} + p_{ij}^{++}}{n+m} \log \frac{\frac{z_{ij}^{++} + p_{ij}^{++}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right) + \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|Y^+|} \frac{z_{ij}^{++}}{n+m} \log \frac{\frac{z_{ij}^{++}}{n+m}}{\frac{w_j^+}{n+m}} \right).
$$

The next step adds the amount of entropy related to values $x^+ \in X^+$ and $b^* \in B^*$:

$$H_n^{***} = H_n^{**} - \left(\sum_{i=1}^{|X^+|} \sum_{j=1}^{|B^*|} \frac{p_{ij}^{+*}}{n+m} \log \frac{\frac{p_{ij}^{+*}}{n+m}}{\frac{s_j^*}{n+m}} \right).$$

Then, we take into account the unseen values $a^* \in A^*$. We first add the amount of entropy related to values $a^* \in A^*$ and $y^+ \in Y^+$:

$$H_n^{****} = H_n^{***} - \left(\sum_{i=1}^{|A^*|} \sum_{j=1}^{|Y^+|} \frac{r_{ij}^{*+}}{n+m} \log \frac{\frac{r_{ij}^{*+}}{n+m}}{\frac{w_j^+ + q_j^+}{n+m}} \right).$$

Finally, we add the amount of entropy related to values $a^* \in A^*$ and $b^* \in B^*$:

$$H_{n+m} = H_n^{****} - \left(\sum_{i=1}^{|A^*|} \sum_{j=1}^{|B^*|} \frac{r_{ij}^{**}}{n+m} \log \frac{\frac{r_{ij}^{**}}{n+m}}{\frac{s_j^*}{n+m}} \right).$$

□


ICMC
USP
SÃO CARLOS