Effective and unsupervised fractal-based feature
selection for very large datasets: removing linear
and non-linear attribute correlations

Antonio Canabrava Fraideinberze

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Antonio Canabrava Fraideinberze

Effective and unsupervised fractal-based feature selection
for very large datasets: removing linear and non-linear
attribute correlations

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-USP,
in partial fulfillment of the requirements for the degree
of the Master Program in Computer Science and
Computational Mathematics. *EXAMINATION BOARD
PRESENTATION COPY*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Robson Leonardo Ferreira Cordeiro

USP – São Carlos
June 2017

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

F812c Fraideinberze, Antonio Canabrava
 Effective and unsupervised fractal-based feature
 selection for very large datasets: removing linear
 and non-linear attribute correlations / Antonio
 Canabrava Fraideinberze; orientador Robson Leonardo
 Ferreira Cordeiro. - São Carlos - SP, 2017.
 90 p.

 Dissertação (Mestrado - Programa de Pós-Graduação
 em Ciências de Computação e Matemática Computacional)
 - Instituto de Ciências Matemáticas e de Computação,
 Universidade de São Paulo, 2017.

 1. Massive parallel processing - Feature selection
 - Non-linear attribute correlations - Big Data
 - Fractal Theory. I. Cordeiro, Robson Leonardo
 Ferreira, orient. II. Título.

Antonio Canabrava Fraideinberze

**Seleção de atributos efetiva e não-supervisionada em
grandes bases de dados: aplicando a Teoria de Fractais
para remover correlações lineares e não-lineares**

Dissertação apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Mestre Program in Computer Science and
Computational Mathematics. *EXEMPLAR DE
DEFESA*

Área de Concentração: Computer Science and
Computational Mathematics

Orientador: Prof. Dr. Robson Leonardo
Ferreira Cordeiro

USP – São Carlos
June de 2017

ACKNOWLEDGEMENTS

I would like to start thanking the São Paulo Research Foundation (FAPESP), the Coordination for the improvement of Higher Education Personnel (CAPES) and the National Council for Scientific and Technological Development (CNPq) for their support to this MSc work. Additionally, I would like to thank Amazon Web Services (AWS) and Microsoft Azure for the services credits provided, without them this work could not have been gone this far.

I, also, would like to thank all the great people from the Databases and Images Group (GBDI) for the countless hours we spent together at the laboratory as well as outside of it. For all their support, conversations and laughs. I also would like to thank all the colleagues from the time of under-graduation that extends till now.

I would specially like to thank Robson for his patience and understanding mostly during these last months of work. The experiments not always performed as well as we expected frustrating both of us. I thank him also for the time we spent writing up to 3 a.m. to submit a paper. I believe the method of having a meeting a week is very important and works excellently in order to have some progress in the work, almost like the daily meetings in the scrum method and for that I also thank him.

I would like to thank my parents for the understanding of the path I am choosing for myself and their support no matter what. They have always been my base and will always be, and for that I thank them.

Lastly, I would like to thank Chao Tsai Ping, for these years we have been together. We work as a team and her support is singularly important to me, thank you.

ABSTRACT

FRAIDEINBERZE, A. C.. **Effective and unsupervised fractal-based feature selection for very large datasets: removing linear and non-linear attribute correlations**. 2017. 90 f. Monograph (Master student – Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Given a **very large** dataset of moderate-to-high dimensionality, how to mine useful patterns from it? In such cases, dimensionality reduction is essential to overcome the well-known “curse of dimensionality”. Although there exist algorithms to reduce the dimensionality of Big Data, unfortunately, they all fail to identify/eliminate **non-linear** correlations that may occur between the attributes. This MSc work tackles the problem by exploring concepts of the Fractal Theory and massive parallel processing to present *Curl-Remover*, a novel dimensionality reduction technique for very large datasets. Our contributions are: (a) *Curl-Remover* eliminates linear and non-linear attribute correlations as well as irrelevant attributes; (b) it is unsupervised and suits for analytical tasks in general – not only classification; (c) it presents linear scale-up on both the data size and the number of machines used; (d) it does not require the user to guess the number of attributes to be removed, and; (e) it preserves the attributes’ semantics by performing feature selection, not feature extraction. We executed experiments on synthetic and real data spanning up to 1.1 billion points, and report that our proposed *Curl-Remover* outperformed two PCA-based algorithms from the state-of-the-art, being in average up to 8% more accurate.

Key-words: Massive parallel processing – Feature selection – Non-linear attribute correlations – Big Data – Fractal Theory.

RESUMO

FRAIDEINBERZE, A. C.. **Seleção de atributos efetiva e não-supervisionada em grandes bases de dados: aplicando a Teoria de Fractais para remover correlações lineares e não-lineares**. 2017. 90 f. Master dissertation (Master student Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Dada uma **grande** base de dados de dimensionalidade moderada a alta, como identificar padrões úteis nos objetos de dados? Nesses casos, a redução de dimensionalidade é essencial para superar um fenômeno conhecido na literatura como a “maldição da alta dimensionalidade”. Embora existam algoritmos capazes de reduzir a dimensionalidade de conjuntos de dados na escala de *Terabytes*, infelizmente, todos falham em relação à identificação/eliminação de correlações **não lineares** entre os atributos. Este trabalho de Mestrado trata o problema explorando conceitos da Teoria de Fractais e processamento paralelo em massa para apresentar *Curl-Remover*, uma nova técnica de redução de dimensionalidade bem adequada ao pré-processamento de *Big Data*. Suas principais contribuições são: (a) *Curl-Remover* elimina correlações lineares e não lineares entre atributos, bem como atributos irrelevantes; (b) não depende de supervisão do usuário e é útil para tarefas analíticas em geral – não apenas para a classificação; (c) apresenta escalabilidade linear tanto em relação ao número de objetos de dados quanto ao número de máquinas utilizadas; (d) não requer que o usuário sugira um número de atributos para serem removidos, e; (e) mantém a semântica dos atributos por ser uma técnica de seleção de atributos, não de extração de atributos. Experimentos foram executados em conjuntos de dados sintéticos e reais contendo até 1,1 bilhões de pontos, e a nova técnica *Curl-Remover* apresentou desempenho superior comparada a dois algoritmos do estado da arte baseados em PCA, obtendo em média até 8% a mais em acurácia de resultados.

Palavras-chave: Processamento paralelo em massa – Seleção de atributos – Correlações não lineares entre atributos – *Big Data* – Teoria de Fractais.

LIST OF FIGURES

Figure 1 – Examples of synthetic and real fractals with exact or statistical self-similarity.	26
Figure 2 – (a) Points distributed over a line (1-dimensional object). (b) Points distributed over a plane (2-dimensional object). They are all embedded in a 3-dimensional space.	26
Figure 3 – Bi-dimensional space divided by different sized grids.	27
Figure 4 – MapReduce parallel programming model.	28
Figure 5 – Workflow of our algorithm.	33
Figure 6 – (a) 5 bi-dimensional points spread over the space with the box-counting cells representation. (b) The corresponding quad-tree-like data structure. Both the bi-dimensional space and the tree structure are divided into two partitions: one in blue-dotted lines and the other in red-continuous lines.	35
Figure 7 – Adapted <i>map/reduce</i> stage with a merging step.	39
Figure 8 – Left column: log-log plots used to compute D_2 . Note that our datasets are indeed fractals. Right column: D_2 after removing one attribute at a time, from the least relevant ones to the most relevant ones. Plots were built from right to left. Arrows point to the least relevant attributes that were not discarded in each dataset. In average, our <i>Curl-Remover</i> shrank the volume of data in <u>66%</u> and only <u>12%</u> of information was lost.	45
Figure 9 – Additional results presented as in Figure 8.	46
Figure 10 – Additional results presented as in Figure 8.	47
Figure 11 – Scale-up on the data size. <i>Curl-Remover</i> scales linearly as the dataset increases.	49
Figure 12 – Scale-up on the number of machines used for parallel processing. The runtime of <i>Curl-Remover</i> decreases linearly as the number of machines increases.	49
Figure 13 – 2-dimensional hypercube cells and the corresponding <i>Counting Tree</i>	66
Figure 14 – Sliding window of size 100 units of time ($n_p = 4$ and $n_e = 25$) over a 3-dimensional data stream.	67
Figure 15 – Dynamic evolution of the attribute space versus tree coverage. Depicting events taken into account before (top) and after (bottom) the time window slides. (a): the space covered by the <i>Counting Tree</i> is still adequate; (b): coverage must be expanded; (c): coverage should be contracted for better representation.	70

Figure 16 – Examples of expansion and contraction of the attribute space covered by a <i>Counting Tree</i> , and how to efficiently implement them.	73
Figure 17 – Accuracy in synthetic stream.	75
Figure 18 – Runtime in synthetic stream.	75
Figure 19 – Runtime in synthetic stream: building the tree versus spotting clusters. . . .	75
Figure 20 – Runtime in real climatic stream.	75
Figure 21 – Scenario of a typical crisis situation considering our architecture for crisis management.	83
Figure 22 – The DCCM architecture, consisting of the tasks: classification (A), filtering (B) and historical retrieval (C).	84
Figure 23 – Precision-Recall in the process of filtering incoming data in the buffer. . . .	87
Figure 24 – Precision-Recall for retrieving historical data.	88
Figure 25 – Time to extract features from one image and insert them into the database. .	89

LIST OF ALGORITHMS

Algoritmo 1 – Computes the Correlation Fractal Dimension for a dataset A (box-count approach).	31
Algoritmo 2 – Curl-Remover	34
Algoritmo 3 – Mapper-Dataset.	36
Algoritmo 4 – Reducer.	38
Algoritmo 5 – Mapper-Tree.	40
Algoritmo 6 – Moves the sliding window.	69
Algoritmo 7 – Expands the coverage of the tree.	71
Algoritmo 8 – Contracts the coverage of the tree.	72

LIST OF TABLES

Table 1	– Results of <i>Curl-Remover</i> for the datasets studied.	43
Table 2	– Comparing <i>Curl-Remover</i> , sPCA and Kernel PCA. The smaller the error/run-time the better. <i>Curl-Remover</i> led to <u>4.2%</u> better classification than Kernel PCA and <u>8%</u> better than sPCA.	48
Table 3	– Comparing <i>Curl-Remover</i> , sPCA and Kernel PCA regarding the amount of information retained after the reduction of dimensionality. <i>Curl-Remover</i> preserved in average 88.1% of the original values of D_2 , while sPCA retained in average solely 66.9% and Kernel PCA 72.0%.	48
Table 4	– Overall performance of DCCM over Flickr-Fire.	89

CONTENTS

1	INTRODUCTION	21
1.1	Context	21
1.2	Problem and Motivation	21
1.3	Contributions	22
1.4	Final Considerations	23
2	FUNDAMENTAL CONCEPTS	25
2.1	Initial Considerations	25
2.2	Fractal Theory Applied to Databases	25
2.3	MapReduce	27
2.4	Final Considerations	28
3	RELATED WORK	29
3.1	Initial Considerations	29
3.2	Dimensionality Reduction	29
3.3	Feature Selection	30
3.4	Feature Extraction	31
3.5	Final Considerations	32
4	PROPOSED METHOD	33
4.1	Initial Considerations	33
4.2	Sample	34
4.3	Shrink	34
4.3.1	<i>Mappers – building the trees</i>	35
4.3.2	<i>Mappers – speeding up</i>	37
4.3.3	<i>Reducers</i>	38
4.3.4	<i>Merge</i>	38
4.3.5	<i>Time complexity analysis</i>	39
4.4	Final Considerations	39
5	EVALUATION	41
5.1	Initial Considerations	41
5.2	Methodology	41
5.3	Results	43

5.3.1	<i>Comparison</i>	44
5.3.2	<i>Scale-up experiments</i>	44
5.4	Final Considerations	49
6	CONCLUSIONS	51
BIBLIOGRAPHY		53
APPENDIX A FAST AND SCALABLE SUBSPACE CLUSTERING FOR MULTIDIMENSIONAL DATA STREAMS		
A.1	Initial Considerations	61
A.2	Problem and Motivation	61
A.3	Contributions	62
A.4	Background Concepts and Related Works	63
A.5	The base algorithm	64
A.5.1	<i>First phase</i>	64
A.5.2	<i>Second phase</i>	65
A.6	Proposed method	66
A.6.1	<i>Dealing with one sliding window of time</i>	66
A.6.2	<i>Non-normalized data analysis</i>	69
A.6.3	<i>Efficiently representing space expansions and contractions</i>	70
A.7	Experiments	72
A.7.1	<i>System configuration</i>	73
A.7.2	<i>Experiments on synthetic data</i>	74
A.7.3	<i>Experiments on real climatic data</i>	75
A.8	Conclusion	76
APPENDIX B ON THE SUPPORT OF A SIMILARITY-ENABLED RELATIONAL DATABASE MANAGEMENT SYSTEM IN CIVILIAN CRISIS SITUATION		
B.1	Initial Considerations	77
B.2	Problem and Motivation	77
B.3	Contributions	78
B.4	Related Work	79
B.5	Background	80
B.5.1	<i>Content-Based Retrieval</i>	80
B.5.2	<i>kNN Classifier</i>	81
B.5.3	<i>Feature Extractors</i>	81
B.5.4	<i>Evaluation Functions</i>	82
B.5.5	<i>Similarity Support on RDBMS</i>	82

B.6	Proposed Architecture	83
B.6.1	<i>Crisis Management Scenario</i>	83
B.6.2	<i>Data-Centric Crisis Management</i>	83
B.7	Case Study	85
B.7.1	<i>Implementation of DCCM</i>	85
B.7.2	<i>Classification of Incoming Data</i>	85
B.7.2.1	<i>Methodology</i>	85
B.7.2.2	<i>Experimentation and Results</i>	86
B.7.3	<i>Filtering of Incoming Data</i>	86
B.7.3.1	<i>Methodology</i>	86
B.7.3.2	<i>Experimentation and Results</i>	86
B.7.4	<i>Retrieval of Historical Data</i>	87
B.7.4.1	<i>Methodology</i>	87
B.7.4.2	<i>Experimentation and Results</i>	88
B.7.5	<i>Overall Performance</i>	88
B.8	Conclusions	89

INTRODUCTION

1.1 Context

In the past few years, a number of organizations in diverse areas of science have been storing huge amounts of data, in many cases, without having a clear idea of its potential value (BOLON-CANEDO; SÁNCHEZ-MARONO; ALONSO-BETANZOS, 2015). To find useful information within such data is usually essential to monetize it. This panorama has motivated the development of techniques and tools aiming to support decision making by means of data mining, information retrieval and many other strategies of analysis. Today, to analyze, to comprehend and to extract knowledge from large datasets is one of the Biggest Challenges in Computer Science, according to the Brazilian Computer Society (SALGADO; MOTTA; SANTORO, 2015), especially in the context of complex data, such as image collections, large texts, audio, large graphs extracted from the web or from social networks, fingerprints, DNA sequences, climate data, and a number of other “non-traditional” types of data.

1.2 Problem and Motivation

Existing analytical algorithms that are well-suited to process Big-Data¹ usually tend to be inefficient and/or ineffective when the data handled have moderate-to-high dimensionality, say five or more dimensions (CORDEIRO; FALOUTSOS; TRAINA-JR., 2013). This is a well-known problem, commonly referred to as the “curse of dimensionality” (BELLMAN, 2013). *Given a very large dataset of moderate-to-high dimensionality, how to mine useful patterns from its points? How to make a profit from it?* In such cases, dimensionality reduction is essential both to reduce the drawbacks of the “curse of dimensionality” and also to shrink the amount of data to be analyzed. Previous work aimed at tackling this problem (ELGAMAL *et al.*, 2015;

¹ In this work, we use the term Big Data to refer to databases containing a huge number of instances, i.e., in the order of *billions* or elements of more.

BALCAN *et al.*, 2015). Notwithstanding, to the best of our knowledge, the state-of-the-art algorithms aimed at reducing the dimensionality of Big Data present one central drawback: they are usually unable to identify and eliminate **non-linear** correlations among attributes. On the other hand, correlations of this type are very likely to exist in data coming from many real applications (TAYLOR *et al.*, 1998; TUNG; XU; OOI, 2005; ÖZCAN, 2005). Obviously, this fact compromises the accuracy of the dimensionality reduction task as a whole.

It is interesting to note that the aforementioned drawback has already been tackled in the context of very small datasets, with up to a few thousands of elements, by using concepts of the Theory of Fractals applied to data analysis in a serial processing environment (TRAINA-JR *et al.*, 2000). Taking this fact as a premise, the central hypothesis to be explored in this MSc work is:

Hypothesis 1: The use of concepts from the Theory of Fractals applied to data analysis in a massive parallel processing environment makes it feasible to identify and eliminate both linear and non-linear attribute correlations in billion-element-scale datasets, as well as irrelevant attributes.

1.3 Contributions

This MSc work investigates the aforementioned hypothesis focused on datasets of moderate dimensionality, i.e., data spanning from 5 up to 150 attributes. Specifically, we present a novel dimensionality reduction technique for billion-element-scale datasets – the new algorithm *Curl-Remover*. Our main contributions are:

1. **Accuracy** – as opposed to most techniques from the state-of-the-art, *Curl-Remover* eliminates both linear and non-linear attribute correlations, besides irrelevant attributes;
2. **Scalability** – it presents linear scale-up regarding the data size and also according to the number of machines used for parallel processing;
3. **Usability** – it is unsupervised, so it does not require the user to guess the number of attributes to be removed neither requires a training set, which are both rarely available for many real datasets;
4. **Semantics** – it is a feature selection algorithm, thus it maintains the semantics of the attributes, and;
5. **Generality** – it suits for analytical tasks in general, and not only for classification.

We performed experiments on synthetic data and on real data from a number of applications – e.g., Physics, Chemistry, Astrophysics and Web-data flow – spanning up to *1.1 billion* data points. *Curl-Remover* was in average 8% more accurate than two PCA-based algorithms

from the state-of-the-art. The experiments also indicate the linear scale-up of our algorithm, corroborating a theoretical complexity analysis that we developed for it.

1.4 Final Considerations

This chapter introduced the problem and the main reasons that motivated this MSc work, as well as its main contributions. The rest of this document follows a traditional organization with fundamental concepts (Chapter 2), related work (Chapter 3), proposed method (Chapter 4), evaluation (Chapter 5), and conclusions (Chapter 6). Appendices A and B present two additional work that were developed in parallel with the main work of the MSc program, in collaboration with other researchers. The candidate student's individual contributions in each of these work are also reported.

FUNDAMENTAL CONCEPTS

2.1 Initial Considerations

This chapter overviews the main background concepts used in the MSc work. First, we describe concepts of the Fractal Theory applied Databases. Then, we present the MapReduce model for massive parallel processing.

2.2 Fractal Theory Applied to Databases

A fractal is an object that presents the property of being self-similar, which means that it has approximately the same characteristics when analyzed in different resolutions ([SCHROEDER, 2012](#)). Self-similarity can be classified as exact or statistical. The former refers to intrinsic patterns that repeat exactly in different resolutions, while the latter denotes invariant scale independent statistical properties ([TAYLOR *et al.*, 1998](#)). Figure 1 presents examples of synthetic fractals with exact self-similarity, such as the Peano-Gosper, Koch and Vicsek curves, and the Sierpinski triangle. It also illustrates real fractals with statistical self-similarity, as the shape of mountains, clouds, river networks, coasts of countries, the Romanesco broccoli and some species of ferns.

In the scope of databases, [Faloutsos e Kamel \(1994\)](#) have shown that real datasets exhibit fractal behavior, i.e., the spatial object formed from the data points exhibit exact or statistical self-similarity. Hence, the data can be investigated by using concepts and tools of the Fractal Theory. In fact, fractals have been serving as a basis to tackle several problems in the areas of Databases and Data Mining, such as for selectivity estimation ([BAIOCO; TRAINA; TRAINA-JR., 2007](#); [BÖHM, 2000](#); [FALOUTSOS *et al.*, 2000](#)), clustering ([BARBARA; CHEN, 2010](#); [BARBARA; CHEN, 2000](#); [CORDEIRO *et al.*, 2010](#); [CORDEIRO *et al.*, 2013](#)), time series forecasting ([CHAKRABARTI; FALOUTSOS, 2002](#)), correlation detection and analysis of data streams ([NUNES *et al.*, 2010](#); [SOUSA *et al.*, 2007b](#)). The Fractal Theory has also been used

Figure 1 – Examples of synthetic and real fractals with exact or statistical self-similarity.

to support the applications of high impact. In Medicine, for example, it has been used to help psychological analyses ([MESHCHERYAKOVA; LARIONOVA, 2017](#)), lung cancer detection ([REZAIIE; HABIBOGHLI, 2017](#); [LEE; CHANG; HSIEH, 2016](#)) and analyses of Magnetic Resonance Images (MRI) ([LAHMIRI, 2017](#); [ANAMI; UNKI, 2016](#)).

From the Fractal Theory comes the concept of Correlation Fractal Dimension D_2 . It is useful in Databases and in Data Mining to estimate the minimum number of dimensions (i.e., the intrinsic dimensionality) required to lossless represent the points of one given dataset, regardless of the number of attributes present in the data (i.e., the embedded dimensionality) ([TRAINA-JR. et al., 2010](#)). For example, Figures 2a and 2b respectively illustrate 3-dimensional points distributed over a line and over a plane. Although they are all embedded in a 3-dimensional space, one single dimension – formed by a linear combination of x , y and z – is necessary to perfectly represent the line, while two dimensions are enough for the plane. To this extent, we say that the intrinsic dimensionality of the line is 1 and it is 2 for the plane. The embedded dimensionality is 3 in both cases.

Figure 2 – (a) Points distributed over a line (1-dimensional object). (b) Points distributed over a plane (2-dimensional object). They are all embedded in a 3-dimensional space.

The value of D_2 measures the non-uniform behavior of a given dataset ignoring the effects of any polynomial and even non-polynomial correlation that may exist between its attributes ([FALOUTSOS; KAMEL, 1994](#); [TRAINA-JR. et al., 2010](#)). There exist two primary methods to compute D_2 : the exact approach and the approximate box-counting approach ([SCHROEDER, 2012](#)). The former presents quadratic complexity on the number of

data points, so it is out of the scope of this MSc work by being impractical for datasets with hundreds of millions or even billions of points. The latter can be computed with linear complexity on the data size (TRAINA-JR. *et al.*, 2010). Following Equation 2.1 and Figure 3, the box-counting approach lays hyper grids with different side sizes over the dataset's attribute space, then it counts the points that fall within each cell of each grid to compute D_2 . In the equation, $[r_1, r_2]$ is a range of distances that is representative for the data, r is the cell grid size and $C_{r,i}$ is the number of data points that fall within the i -th cell that has side size r . Note that this technique can only be used for data that present self-similarity, i.e., datasets for which plotting $\log(r)$ versus $\log\left(\sum_i C_{r,i}^2\right)$ results in a curve that nicely approximates a line segment for the distances in $[r_1, r_2]$. The slope of this line segmented defines D_2 .

Figure 3 – Bi-dimensional space divided by different sized grids.

$$D_2 \equiv \frac{\partial \log\left(\sum_i C_{r,i}^2\right)}{\partial \log(r)} \quad r \in [r_1, r_2] \quad (2.1)$$

2.3 MapReduce

Diverse tools have been used to parallelize processing and storage for Big Data analysis, such as GPU (GUILLÉN *et al.*, 2014) and parallel programming models, like the MPI (ASFOOR *et al.*, 2014). However, most of the existing tools require the user to implement solutions for data distribution and replication as well as for load balance and fault tolerance, or, at least, be aware of them in detail. The high complexity of these tasks motivated the development of simpler models, like MapReduce.

MapReduce (DEAN; GHEMAWAT, 2004) is a parallel programming model aimed at processing large volumes of data in a simple manner and at feasible runtime and cost. It provides solutions that hide from the user the complexity related to parallelism, such as data storage, distribution and replication, fault tolerance, load balance and the like.

A typical MapReduce job is represented in Figure 4. The dataset stored in a distributed file system is divided into non-overlapping subsets – commonly referred to as splits. Each split is then sent to a *map* process. The *mappers* process the data received and emit pairs containing a key and a value each. These pairs are sorted and grouped by their key to be sent to *reduce* processes, in such a way that pairs sharing the same key are always processed together. The *reducers* handle the keys and values received and store results back in the distributed file system. Due to its scalability, simplicity and the low cost to build large cluster of computers, MapReduce is today a promising tool to analyze Big Data (QIAN *et al.*, 2015; FARAHAT *et al.*, 2013).

Figure 4 – MapReduce parallel programming model.

Apache Hadoop¹ is one of the most used implementations for MapReduce. It provides a wide set of tools, such as a distributed file system with data replication – the Hadoop Distributed File System (HDFS), and a resource manager with node fault tolerance – the Yet Another Resource Negotiator (YARN), among others.

Apache Spark² is a general engine for large scale processing that includes a MapReduce implementation. Its main basis is the concept of Resilient Distributed Datasets (RDD) (ZAHARIA *et al.*, 2012), which corresponds to an abstraction layer for distributed memory computing in a fault tolerant manner. Spark has two main types of operations: *transformations* and *actions*. Transformations are usually stacked up to be evaluated all at once when an action takes place, in a process known as *lazy-evaluation*. Differently from Hadoop that works in a batch-mode, Spark is mainly focused on iterative processing.

2.4 Final Considerations

This chapter introduced background concepts that are used as a basis for the MSc work. In the next chapter, we present related work regarding the task of dimensionality reduction, especially focused on those techniques that can handle large volumes of data.

¹ <https://hadoop.apache.org/>

² <https://spark.apache.org/>

RELATED WORK

3.1 Initial Considerations

In the previous chapter, we presented two concepts that are essential for this work: (i) the Fractal Theory and its applications to data management and; (ii) the parallel programming model MapReduce that supports the development and deployment of complex algorithms for large clusters of computers. This chapter presents the task of dimensionality reduction as well as relevant works found in the literature for it, especially those that can deal with Big Data.

3.2 Dimensionality Reduction

Knowledge Discovery in Databases (KDD) is the process of identifying valid, new and essentially useful patterns embedded in raw data ([FAYYAD; PIATETSKY-SHAPIRO; RAMASAMY, 2003](#)). It is divided in three main steps: preprocessing, data mining and result evaluation. The task of data mining refers to the use of analytical algorithms to spot the patterns, therefore it is the main part of the KDD process. Nevertheless, it highly depends on receiving adequately preprocessed data as input.

During the preprocessing phase, the data are reduced and/or prepared, through cleaning, integration, selection and transformation, for further pattern discovery. One of the main problems tackled here is the “curse of dimensionality” ([BELLMAN, 2013](#)). This term refers to the fact that many analytical algorithms suffer from significant degradation in efficiency and/or in efficacy, as the number of data attributes increase. The main technique used to mitigate this issue is the dimensionality reduction, which aims at obtaining a new set of attributes with lower cardinality or to select a subset of the original attributes to represent the data with minimum information loss. To make it possible, the resulting set cannot have irrelevant nor correlated attributes.

According to ([BOLON-CANEDO; SÁNCHEZ-MARONO; ALONSO-BETANZOS,](#)

2015), dimensionality reduction methods can be categorized in filter, embedded and wrapper methods. Filter methods are task independent, meaning that they are well suited to preprocess data for use by analytical algorithms in general. They are also unsupervised. On the other hand, embedded and wrapper methods are tailored to preprocess data for specific data mining tasks, i.e., classification or clustering, aimed at obtaining better results at the cost of worse generality. Dimensionality reduction algorithms are also categorized in two main classes (BOLON-CANEDO; SÁNCHEZ-MARONO; ALONSO-BETANZOS, 2015): feature selection and feature extraction/transformation. While the former selects the most relevant attributes among the original ones, thus preserving the semantics of the data, the latter creates a reduced set of new attributes to better represent the data in such a way that each new attribute is a combination of original attributes, however, losing the original semantics of the attributes.

Due to its importance as a preprocessing step for data mining, machine learning, computer vision and many other research areas, dimensionality reduction has been in constant development in the past few decades (WANG *et al.*, 2016). The following sections discuss some of the works in the state-of-the-art.

3.3 Feature Selection

There is an extensive body of work on using the Theory of Rough-sets (PAWLAK, 1982) for feature selection in Big Data, commonly focused on datasets with missing and/or uncertain attribute values (BOLON-CANEDO; SÁNCHEZ-MARONO; ALONSO-BETANZOS, 2015). Some relevant examples are (YANYUN *et al.*, 2012; ZHU *et al.*, 2013; SUN; LI, 2014; QIAN *et al.*, 2014) and (ZHAO *et al.*, 2013). In spite of the many qualities of these works, to the best of our knowledge, they all depend on interactive user supervision, thus having limited usability for many real applications. Also, most of these works cannot handle data of high cardinality, e.g., billions of data objects. We consider them to be out of the scope of this MSc. proposal since we focus on non-supervised dimensionality reduction for Big Data.

In (ORDOZGOITI; CANAVAL; MOZO, 2015), a parallel feature selection algorithm is presented. Based on the Column Subset Selection Problem (CSSP) (BOUTSIDIS; MAHONEY; DRINEAS, 2009), it selects original attributes incrementally, according to their relevance, by verifying changes that may occur in the data variance when considering distinct attribute subsets. Unfortunately, this algorithm is limited to only identify/remove linear correlations between the attributes. It also requires the user to guess the number of attributes to be removed, despite the fact that this number is commonly unknown for most real datasets.

The algorithm Fractal Dimension Reduction (FDR) (TRAINA-JR *et al.*, 2000; TRAINA-JR. *et al.*, 2010; TRAINA-JR.; TRAINA; FALOUTSOS, 2010) is a basis for this MSc work. Both FDR and the new algorithm that we propose use the Correlation Fractal Dimension (D_2) to pick the most relevant attributes from multidimensional data, being able to detect linear and

non-linear attribute correlations. Note, however, that FDR is limited to process relatively small datasets due to two main reasons: (a) it proposes a serial processing strategy, and; (b) it requires a volume of main memory that is commonly tens of times larger than the size of the input data; FDR has a linear memory complexity on the data size, but the constant values in the equation are large.

Algorithm FDR works as follows. First, it computes D_2 for the whole dataset and sets all attributes as relevant. Then it calculates E Partial Correlation Fractal Dimension (pD_i) values ignoring one relevant attribute i at a time, in which E is the number of attributes – the embedded dimensionality. All the pD_i values are sorted and the attribute leading to the smaller difference between pD_i and D_2 is removed as irrelevant. The general idea/assumption here is that this attribute is the one that adds the least amount of information to the data. The aforementioned process is repeated until obtaining a set with $\lceil D_2 \rceil$ relevant attributes, which is provided to the user as the final output.

FDR computes D_2 and pD_i using a fast and scalable implementation – Algorithm 1 – for the box-counting approach. First, hyper grids with distinct side sizes are laid over the attribute space, using a quad-tree-like data structure to speed-up the process, and the data points within each cell of each grid are counted. Then, a log-log plot is created with the grid side sizes versus the sum of the squared counts of points for each grid. The slope of the linear part of the plot approximates D_2 .

Algorithm 1: Computes the Correlation Fractal Dimension for a dataset A (box-count approach).

Input: Normalized dataset A with N rows and E attributes; number n of distinct resolutions (grid side sizes)

Result: D_2

```

1 begin
2   foreach point of the dataset  $A$  do
3     foreach grid side size  $r = \frac{1}{2^j}, j \in [1, n]$  do
4       Decide which grid cell of side size  $r$  the current point falls in (say, the  $i$ -th cell)
         using a quad-tree-like structure to represent the grids;
5       Increment the count  $C_{r,i}$ , that is, insert the point in the current level of the tree
         structure;
6   Compute the sum of occupancies  $S(r) = \sum C_{r,i}^2$ , for each grid side size  $r$ ;
7   Use the values of  $\log(r)$  and  $\log(S(r))$  to generate a plot;
8   Return the slope of the linear part of the plot as the value of  $D_2$  for dataset  $A$ ;

```

3.4 Feature Extraction

Principal Component Analysis (PCA) (JOLLIFFE, 1987) and Singular Value Decomposition (SVD) (STEWART, 1993) are the most popular techniques for feature extraction,

being considered as valuable tools in diverse areas such as image processing and information retrieval (WANG *et al.*, 2016). They were both used in the context of Big Data. In (DING *et al.*, 2011), SVD was implemented using Message Passing Interface (MPI) and the library ARPACK (LEHOUCQ; SORENSEN; YANG, 1998). Unfortunately, the algorithm has a cubic complexity on the dimensionality of the output data, and it is limited to identify/remove linear correlations only. Also, as a feature extractor, this algorithm does not preserve the meaning of the original attributes.

Algorithm Scalable PCA (sPCA) (ELGAMAL *et al.*, 2015) follows the Probabilistic PCA (PPCA) approach (TIPPING; BISHOP, 1999) implemented in a distributed environment with MapReduce. sPCA includes optimization strategies mainly focused on minimizing the volume of the intermediate data that it creates. However, both sPCA and PPCA exhibit the same inherent drawbacks from PCA, being limited to find/remove linear correlations only and do not preserving the meaning of the original attributes.

Kernel PCA (KPCA) (SCHÖLKOPF; SMOLA; MÜLLER, 1998) extends PCA to identify both linear and non-linear attribute correlations. In (BALCAN *et al.*, 2015), a version of KPCA was proposed to analyze multidimensional data in a multi-machine, distributed fashion. It includes different heuristics aimed at reducing the communication between machines of the cluster, such as running PCA locally on each worker node before sending results to a master node, and using dynamic point sampling. Unfortunately, this technique cannot handle data with very large cardinality – e.g., billions of points – in feasible time.

3.5 Final Considerations

After revising the literature, we conclude that: in spite of the many qualities of the related works, to the best of our knowledge, there is no feature selection algorithm well suited to identify and remove **non-linear** attribute correlations in the context of **Big Data**. This MSc. work tackles this relevant limitation focused on datasets of *medium dimensionality*, i.e., data in the range of around 5 to 150 axes, aimed at supporting analytical algorithms by preprocessing data using dimensionality reduction in a more effective way. In the next chapter, we present the novel algorithm that was developed in this MSc. program.

PROPOSED METHOD

4.1 Initial Considerations

In the previous chapters, we discussed the motivations, contributions and fundamental concepts related to this MSc work. Then we presented related works existing in the literature for dimensionality reduction, mainly focused on works that are well suited to process Big Data.

Here, we detail the main contribution of this MSc work: the new algorithm *Curl-Remover* for feature selection in a distributed, MapReduce-based environment. Figure 5 illustrates our proposed workflow; the corresponding pseudo-code is in Algorithm 2. In a nutshell, the new algorithm has two main phases: *Sample* and *Shrink*. At first, we analyze a tiny sample extracted from the input dataset to obtain a rough estimate of its attributes' relevances. The final set of relevant attributes is then computed from the full dataset using the initial estimation to: (i) minimize processing, disk accesses and network traffic among machines of the cluster, and; (ii) balance the workload on these machines. The following subsections detail our proposal.

Figure 5 – Workflow of our algorithm.

Algorithm 2: Curl-Remover**Input:** Dataset A **Result:** List of relevant attributes

```

1 begin
2   Execute the Sample step;
3   Considering the list of attributes obtained from the sample, identify the  $M$  probably
      most relevant attributes to be used in the next phases;
4   Execute a Reduce step using the Mapper-Dataset;
5    $D_{current} \leftarrow pD_i$  that gives the smaller value ( $D_2 - pD_i$ );
6   Set attribute  $i$  as irrelevant, and ignore it from now on;
7   while  $D_{current} \geq \lceil D_2 \rceil$  do
8     if the last level of the tree exists in the distributed file system then
9       | Execute a Reduce step using the Mapper-Tree;
10    else if then
11      | Execute a Reduce step using the Mapper-Dataset;
12       $D_{current} = pD_i$ ;
13      Set attribute  $i$  as irrelevant, and ignore it from now on;
14  Return a list with the non-ignored attributes;

```

4.2 Sample

The phase of sampling is straightforward. The *mappers* read the input dataset from the distributed file system, and randomly select a tiny subset of points to be sent to a single reducer. Then, the *reducer* processes the sample to return a rough estimate of the attributes' relevances by using one serial processing, feature selection algorithm as a plugged-in subroutine. Any of the numerous serial processing algorithms available in the literature for this task can be used here, preferably one algorithm that is able to identify non-linear correlations, such as the algorithm FDR. Note, however, that the results obtained from the sample are used exclusively to speed-up the next phase of our workflow, in which we process the full dataset – they have no influence on the final set of features selected by our method.

4.3 Shrink

The second phase of *Curl-Remover* analyzes the entire set of objects. It takes advantage of concepts from the Theory of Fractals to look for one irrelevant attribute at a time, in ascending order of relevance, until the $E - \lceil D_2 \rceil$ least relevant attributes are identified. E and D_2 respectively refer to the dataset's embedded dimensionality, i.e., the number of attributes, and its Correlation Fractal Dimension, i.e., the approximate intrinsic dimensionality. An efficient quad-tree-based implementation for the box-counting approach allows us to compute D_2 with linear scalability on the data size.

Figure 6 – (a) 5 bi-dimensional points spread over the space with the box-counting cells representation. (b) The corresponding quad-tree-like data structure. Both the bi-dimensional space and the tree structure are divided into two partitions: one in blue-dotted lines and the other in red-continuous lines.

4.3.1 Mappers – building the trees

As it can be seen in Figure 5, the second phase of our algorithm begins with a *map/reduce* stage, which is detailed in Algorithms 3 and 4. Each *mapper* inserts/represents its chunk of data points into a multidimensional quad-tree data structure that only stores counts of points and cell IDs. To illustrate the tree, let us consider the toy dataset in Figure 6. It shows 5 bi-dimensional data points (Figure 6a), and the corresponding tree up to three levels of resolution (Figure 6b). The feature space is recursively divided into cells of distinct sizes, each of which storing an ID, the count of points C that falls within the cell, and a pointer P to the next tree level.

The tree is built in main memory, and each of its nodes can be implemented as a linked list of cells, or a memory-based, key-value index structure like a red-black tree using cell IDs as the keys and point counts as the values. Although the number of regions to divide the space “explodes” at $O(2^{EH})$ for E attributes and H tree levels, we only store/subdivide cells with at least one point. So, each tree level has in fact at most one cell per point. Empirical evidences show that nearly $H = 15$ levels are enough to accurately calculate D_2 , as we demonstrate in the upcoming Chapter 5.

Despite the linear memory usage on the data size, the limited amount of memory available for each *mapper* may not be enough for the entire tree. We tackle this problem by monitoring the memory usage while inserting points in the tree, and whenever a *mapper* is close to run out of memory, it sends the tree computed so far to the *reducers*, then it cleans up and builds

Algorithm 3: Mapper-Dataset.

Input: Chunk of E -dimensional points of A , # of levels N and the M probably most relevant dimensions

Result: Pairs (x_ID, C) and $(x_ID_p, point)$

```

1 begin
2   Sorts the points in  $A$ ;
3   for  $x := 0$  to  $E$  do
4     // Building the tree that ignores attribute  $x$ . Does not ignore
      any attribute when  $x = 0$ 
5     Initializes the tree root with point count  $C = 0$ ;
6     foreach point of  $A$  do
7       Sums 1 to the point count  $C$  of the root node;
8       foreach grid size  $h = 1, 2, \dots, N$  do
9         Decides which tree cell from level  $h$  the current point belongs to;
10        Sums 1 to this cell's count  $C$ ;
11      Emits the pairs  $(x\_ID_p, point)$  of the current point;
12      if memory full then
13        Emits the pair  $(x\_ID, C)$  of all the nodes currently in the tree;
14        Deletes the tree;
15        Initializes one new root with point count  $C = 0$ ;
16      Emits the pair  $(x\_ID, C)$  of all cells currently in the tree;
17      Deletes the tree;

```

a whole new tree from the next points to be inserted. A crucial preprocessing step must be emphasized here: before building any tree, i.e., right after receiving its chunk of data points, each *mapper* sorts the points in main memory using multiple criteria. That is, 1st criterion: values of the 1st attribute; 2nd criterion: values of the 2nd attribute, and so on. Then, it builds the tree by processing the points in order. This simple procedure leads us to create in distinct moments tree branches that mostly represent distinct subregions of the feature space, thus avoiding to reprocess space regions in the event of running out of memory. To illustrate this fact, let us consider again our running example from Figure 6. By processing the points in order we create the tree partition highlighted with a blue-dotted line first, then the one surrounded by the red-continuous line, so these two branches could be created separately with very little overhead – only the tree root would have to be represented twice. Note that it happens when considering any of the two options of dual criteria: x -axis first, y -axis latter; or the opposite way.

In fact, each *mapper* creates and reports to *reducers* $E + 1$ independent tree structures, one at a time. The first tree stores point counts in the full, E -dimensional feature space, while the other E trees count the same points projected in subspaces of dimensionality $E - 1$, each subspace containing all but one of the attributes. In this way, we provide enough information to identify the least relevant attribute at the end of the *map/reduce* stage by comparing the impact on D_2 caused by the elimination of each attribute individually.

4.3.2 Mappers – speeding up

We struggle to minimize the amount of data transferred among machines of the cluster, as well as the consequent processing to shuffle the data and the many disk accesses used on temporary data files. Ideally, we should tackle this problem by creating only a top few levels of the $E + 1$ trees in the *mappers*, say the $N = 2$ or 3 levels of lower resolution. Then, the results would be combined in the *reducers* to build the remaining $H - N$ levels for each tree. In this way, the first N levels would help us distribute balanced workloads to *reducers*, while the vast majority of tree cells – lower level cells tend to be much more numerous than those from higher levels – would be built and used locally in each *reducer*.

But, *how to combine the results? And, how to balance the workload among the reducers?* Remember that we have many *mappers* extracting partial point counts from each data chunk, including counts for the full E -dimensional space and those for E distinct subspaces. Each tree cell created in each *mapper* may have counterparts in all other *mappers*. Additionally, as we mentioned before, our algorithm identifies one irrelevant feature at a time, so in the next iteration, we must deal with point counts for the best $E - 1$ -dimensional subspace and for its $E - 1$ subspaces of dimensionality $E - 2$.

To tackle this problem, we take advantage of the preliminary result obtained from the data sample analyzed at the very beginning of the process – see Subsection 4.2. Specifically, we propose to have *mappers* emitting (*key*, *value*) pairs of two distinct types: one type to represent tree cells, from the N levels of lower resolution, of course, and another type for data points. The pairs representing cells have the form (x_ID, C) , in which x_ID uniquely identifies the cell among all cells of all trees created in the *mapper* and C is the corresponding count of points. Note that cells from distinct *mappers* may share the same x_ID . On the other hand, the pairs referring to data points have the form $(x_ID_p, point)$. The value of x_ID_p for a data point is the ID of the cell in which the point falls into, considering level N of a tree that represents a given M -dimensional subspace. The actual data point is in *point*. Here, we identify each point considering a subspace of very low dimensionality, say $M = 2$ or 3 dimensions, formed by the M most relevant attributes obtained from the rough estimate performed before in the phase of sampling – see Subsection 4.2.

The idea here is to use this particular M -dimensional subspace as a fixed information shared by all *mappers* in order to send nearby points to the same *reducer* independently of any matter of parallelization, like data partitioning, synchronization, and others. This strategy makes it possible to create the remaining $H - N$ levels of each tree in the *reducers*, thus minimizing processing, disk accesses and network traffic among machines of the cluster, as we discussed before. Note that it also allows us to balance the workload on these machines since we can use an appropriate maximum number of possible values for x_ID_p by tuning parameter M to make 2^{M*N} be only a few times larger than the number of reducers available for parallel processing. The main disadvantage of this strategy is that the *mappers* may have to create the whole trees if

one of the M most relevant attributes spotted in the sample turns out to be considered irrelevant in the full data. We consider this situation as an extreme and unlike case since the usual values of M (2 or 3) are much smaller than D_2 for most real datasets. In fact, it did not happen in any of the experiments that we performed.

4.3.3 Reducers

The *reduce* stage receives the pairs sent by the *mappers* and handles each pair according to its type. As it is shown in Algorithm 4, each *reducer* performs simple computations to obtain and emit, one $(key, value)$ pair for each level of each tree with the corresponding side size of cells, and the sum of the squared counts of points of the level.

Algorithm 4: Reducer.

Input: Pairs (x_ID, C) and $(x_ID_p, point)$

Result: Pairs $(\log(r), \sum C^2)$

1 **begin**

2 **if** x_ID_p **then** // Builds the bottom of the tree

3 **foreach** pair $(x_ID_p, point)$ **do**

4 **foreach** $h = N + 1, N + 2, \dots, H$ **do**

5 Decides which tree node in the level h the current point belongs to;

6 Sums 1 to the points count C ;

7 **for** $x := 0$ **to** E **do**

8 **foreach** pair (ID, C) **do**

9 Sums C for the same ID ;

10 **foreach** $h = 1, 2, \dots, H$ **do**

11 Sums C^2 ;

12 **foreach** $h = 1, 2, \dots, H$ **do**

13 Emits the pair $(x \log(r), \sum C^2)$

4.3.4 Merge

Once the reduce phase is complete, a serial processing step that we name *merge* is executed. Figure 7 illustrates a *map/reduce* stage adapted with the merging step. This step takes as input the tiny amount of data returned by each reducer, i.e., solely $2H(E + 1)$ numeric values, and computes D_2 for the full E -dimensional dataset, and also the Partial Correlation Fractal Dimension (pD_i) for each of its E subspaces of dimensionality $E - 1$ that ignores an attribute i . The attribute i that leads to the highest pD_i (i.e., the one that contributes the least to the Correlation Fractal Dimension of the full E -dimensional dataset) is then assumed to be the least relevant attribute, and it is ignored in a new *map/reduce* stage to be initiated. In this way we spot one irrelevant attribute at a time, in ascending order of relevance, until the $E - \lceil D_2 \rceil$ least relevant attributes are identified.

Figure 7 – Adapted *map/reduce* stage with a merging step.

The *merge* step plays one other important role in our algorithm. Besides spotting the least relevant attribute, it also estimates the time necessary to read the dataset again in one future map/reduce stage and compares it with the estimated time necessary to use a possibly better strategy: to recover the data from the tree. If the predicate shown in Equation 4.1 is true, the next *reduce* stage will store the last tree level in the distributed file system, and our algorithm will read this data later on instead of the original data points. In this case, Algorithm 5 would replace Algorithm 3 for the *mappers*, and no further modification would be necessary. In the equation, t_{store} is the estimated time to store the last level of the tree in the distributed file system; $t_{transfer}$ is the estimated time to transfer the last level of the tree from *mappers* to *reducers*; t_{read} is the estimated time to read the last level of the tree from the distributed file system; T_{read} is the average time spent to read the input dataset from the distributed file system; and, $R = E - \lceil D_2 \rceil - \#_of_irrelevant_attributes_identified_so_far$.

$$t_{store} + t_{transfer} + R \times t_{read} < R \times T_{read} \quad (4.1)$$

4.3.5 Time complexity analysis

From the proposed algorithm, we perform as follows a theoretical analysis in terms of disk I/O and network traffic. The number of read operations is $(E - \lceil D_2 \rceil + 1) \times number_of_points$, thus $O(E \times number_of_points)$. The network traffic is bounded by $O(E \times N \times number_of_points)$.

4.4 Final Considerations

In this chapter, we presented the new algorithm *Curl-Remover* for feature selection in a distributed, MapReduce-based environment. It is the main contribution of this MSc work. In the next chapter, we present an extensive set of experiments performed on real and synthetic datasets to evaluate the new method, including results obtained when comparing it with existing techniques from the state-of-the-art, and results that corroborate our complexity analysis.

Algorithm 5: Mapper-Tree.

Input: Chunk of pairs (ID, C)
Result: Pairs (x_ID, C) and $(x_ID_p, point)$

```

1 begin
2   for  $x := 0$  to  $E$  do
3     // Building the tree that ignores attribute  $x$ . Does not ignore
4     // any attribute when  $x = 0$ 
5     Initializes the tree root with point count  $C = 0$ ;
6     foreach pair  $(ID, C)$  do
7       Builds the tree bottom-up, incrementing the value of  $C$ ;
8       if reduce printing then
9         Removes the attributes previously removed;
10        Emits the pair  $(ID, C)$ ;
11      Emits the pairs  $(x\_ID_p, point)$  of the current point;
12      if memory full then
13        Emits the pair  $(x\_ID, C)$  of all the nodes currently in the tree;
14        Deletes the tree;
15        Initializes one new root with point count  $C = 0$ ;
16    Emits the pair  $(x\_ID, C)$  of all cells currently in the tree;
17    Deletes the tree;

```

EVALUATION

5.1 Initial Considerations

The previous chapter presented in detail the main contribution of this MSc work: the new algorithm *Curl-Remover* for feature selection. Here, we report the experiments performed to evaluate *Curl-Remover* on a variety of real and synthetic datasets. We aimed to answer two central questions:

1. *Compared with two PCA-based related works from the state-of-the-art, how accurate is our new algorithm?*
2. *How does it scale-up regarding the data size and the cluster size?*

5.2 Methodology

Curl-Remover was implemented in Hadoop 2.6.0¹. Our experiments used a Microsoft Azure cluster with 21 machines: one *master* machine with 2 cores, 3.5 GB of RAM and 60 GB of disk, and; 20 *worker* machines, each one with 8 cores, 14 GB of RAM and 600 GB of disk. We configured the machines with GNU/Linux CentOS 6.5. The HDFS block size was set to 256 MB. The algorithms used for comparison with our proposal are the sPCA and the Kernel PCA, which were implemented using Apache Spark, with the MLlib implementation of PCA and kmeans++. Each *mapper/reducer* had 3.1 GB of RAM, allowing two cores per process and the spawn of 4 processes at the same time per machine. The source codes of *Curl-Remover*, sPCA and Kernel PCA, as well as the datasets used in our experiments, are available in <<https://www.dropbox.com/sh/ky395s7134u5sox/AACtwTESy2fvugKkFj7Y2oana?dl=0>>

We studied the synthetic and real datasets described as follows:

¹ <hadoop.apache.org>

- i *Sierpinski* is a group of synthetic datasets with the same characteristics, except for their sizes that vary from 1 million to **1.1 billion** data points. There are 5 dimensions: two attributes representing the Sierpinski Triangle, a and b ; one attribute linearly correlated to the first two, $c = (a + b)$ and; two others following non-linear correlations, $d = (a^2 + b^2)$ and $e = (a^2 - b^2)$;
- ii *Hybrid Sierpinski* is another group of synthetic datasets with sizes that vary from 1 million to **1.1 billion** data points. All datasets have 5 dimensions: two attributes for the Sierpinski Triangle, a and b ; one attribute non-linearly correlated to the first two, $c = (a^2 + b^2)$ and; two attributes with random data, $d = \text{random}()$ and $e = \text{random}()$;
- iii *Yahoo! Network Flows*² is a real dataset containing communication patterns between end-users in the Web. It has 562 million points and 12 attributes;
- iv *Astro* is a one billion-elements snapshot (time slice) of a high-resolution cosmological simulation. It describes **1.07 billion** astrophysical particles at three distinct timestamps with 6 attributes;
- v *Susy* (BALDI; SADOWSKI; WHITESON, 2014) is a well-known dataset used by physicists to classify subatomic particles into: (a) supersymmetric particles produced by a signal process, and; (b) particles produced by a background process. It has 5 million data points and 18 attributes;
- vi *Susy 2x* is a modified version of *Susy* containing 36 attributes. The 18 additional attributes were generated from non-linear correlations between the original ones;
- vii *Susy 8x* is a modified version of *Susy* containing **144 attributes**. The 126 additional attributes were generated from non-linear correlations between the original ones;
- viii *Hepmass* is another physics-related dataset. It regards the classification of particles of unknown mass. It has 28 attributes and 10.5 million instances;
- ix *Hepmass 2x* is a modified version of *Hepmass* containing 56 attributes. The 28 additional attributes were generated from non-linear correlations between the original ones;
- x *Hepmass 4x* is a modified version of *Hepmass* containing **112 attributes**. The 84 additional attributes were generated from non-linear correlations between the original ones;
- xi *Ethylene* (FONOLLOSA *et al.*, 2015) includes records of 16 chemical sensors exposed to two dynamic gas mixtures. It has 19 dimensions in total and 8.3 million instances;
- xii *Ethylene 2x* is a modified version of *Ethylene* containing 38 attributes. The 19 additional attributes were generated from non-linear correlations between the original ones;

² <<http://webscope.sandbox.yahoo.com/catalog.php?datatype=g&did=18>>

Table 1 – Results of *Curl-Remover* for the datasets studied.

Name	Embedded Dim.	Int. Dim.	D_2	Number of points	Reduction (%)
Sierpinski	5	2	1.63	from 1M to 1.1B	60.0
Hybrid Sierpinski	5	4	3.76	from 1M to 1.1B	20.0
Yahoo! Net. Flows	12	4	3.10	562M	66.6
Astro	6	4	4.00	1.1B	33.3
Susy	18	7	6.06	5.0M	61.1
Susy 2x	36	7	6.35	5.0M	80.5
Susy 8x	144	7	6.35	5.0M	97.2
Hepmass	28	14	13.25	10.5M	50.0
Hepmass 2x	56	14	13.75	10.5M	75.0
Hepmass 4x	112	16	15.29	10.5M	85.7
Ethylene	19	4	3.46	8.3M	78.9
Ethylene 2x	38	4	3.52	8.3M	89.4
				Average	66.48

5.3 Results

We executed *Curl-Remover* on the aforementioned datasets and report results in Table 1. For each dataset, we present the embedded, intrinsic and D_2 dimensionality, its number of points and the relative reduction in the volume of data compared to the full dataset.

For the synthetic datasets, it is worth to note that the dimensions identified as relevant were the ones that we expected. For the *Sierpinski* datasets, they were a and b , those that generate the Sierpinski triangle and the algorithm removed the three correlated dimensions; for the *Hybrid Sierpinski*, *Curl-Remover* selected a and b that generate the Sierpinski triangle, as well as d and e that are uniformly distributed. Attribute c was correctly discarded.

Despite the fact that *Curl-Remover* reduced, in average, the datasets to 34% of their original sizes, the results presented in Figures 8, 9 and 10 indicate that very little information was lost. In average, 88% of the original values of D_2 were preserved. The left column of each figure confirms that the datasets are indeed fractals, since the curves in the log-log plots derived from Equation 2.1 – see Section 2.2 – nicely approximate line segments for representative ranges of distances. In the right column of each figure, we plot the values of D_2 after removing one attribute at a time from each dataset, from the least relevant attributes to the most relevant ones, according to the order of relevance provided by *Curl-Remover*. Note that these plots were built from the right to the left. The arrows point to the least relevant attributes that were not discarded in each dataset.

5.3.1 Comparison

We compared the accuracy of *Curl-Remover* with that obtained by algorithms sPCA and Kernel PCA. To do so, we reduced the dimensionality of datasets Susy, Susy 2x, Susy 8x, Hepmass, Hepmass 2x, Hepmass 4x, Ethylene and Ethylene 2x using one algorithm at a time and classified the resulting data. sPCA and Kernel PCA require the user to guess the best dimensionality for the reduced data. For each dataset, its $\lceil D_2 \rceil$ principal components were used in this experiment. The classifiers were Decision Trees, Gradient Boosted, Random Forest and Logistic Regression, from library MLlib³.

For the Decision Tree, the maximum depth of the tree equals to 5 and the maximum number of bins equals to 32. For the Gradient Boosted, the maximum depth of the tree equals to 5. The Random Forest was a with 5 trees of maximum depth of 5 and maximum number of bins set to 32. Finally, the Logistic Regression algorithm did not require any parameter configuration.

Table 2 reports the error generated by each classifier. As we can see, the classification using *Curl-Remover* as a preprocessor was in average **8%** more accurate than that using sPCA and **4.2%** more accurate than that with Kernel PCA. We also used the Student's t-test to better understand the improvements obtained by *Curl-Remover*, and found out that there exists indeed statistically significant difference between the results of our method and those of each of its competitors. On the other hand, the difference between *Curl-Remover*'s results and those obtained from the original, full-dimensionality datasets is not statistically significant.

Table 3 presents the percentage of D_2 preserved after the dimensionality reduction performed by each of the competing algorithms. Here, we aim at indicating how much information was retained within the reduced set of attributes. As it can be seen: *Curl-Remover* was able to retain in average **88.1%** of the original Fractal Dimensionality D_2 , while sPCA retained 66.9% and Kernel PCA 72.0% of it.

5.3.2 Scale-up experiments

We performed experiments to test the scalability of *Curl-Remover* on increasing dataset sizes. Figure 11 reports runtime results for the *Sierpinski* and the *Hybrid Sierpinski* groups of datasets. All results are the average of five distinct runs. Note that standard deviation values are too small to be shown. As we can see, *Curl-Remover* demonstrated linear scalability in both cases.

We also performed scale-up experiments varying the number of machines used for parallel processing. Figure 12 reports the results. The datasets are the *Sierpinski* and the *Hybrid Sierpinski* with 100 million points each. The number of machines increases from 5 to 20. The results presented are the average of 3 distinct runs for each setting of machines. Again, standard

³ <https://spark.apache.org/mllib/>

(a) Sierpinski D_2 .

(b) Sierpinski attribute's influence based on the reverse removing order.

(c) H. Sierpinski D_2 .

(d) H. Sierpinski attribute's influence based on the reverse removing order.

(e) Susy D_2 .

(f) Susy attribute's influence based on the reverse removing order.

(g) Susy 2x D_2 .

(h) Susy 2x attribute's influence based on the reverse removing order.

Figure 8 – Left column: log-log plots used to compute D_2 . Note that our datasets are indeed fractals. Right column: D_2 after removing one attribute at a time, from the least relevant ones to the most relevant ones. Plots were built from right to left. Arrows point to the least relevant attributes that were not discarded in each dataset. In average, our *Curl-Remover* shrank the volume of data in 66% and only 12% of information was lost.

(a) Susy 8x D_2 .

(b) Susy 8x attribute's influence based on the reverse removing order.

(c) Yahoo! Net. Flows D_2 .

(d) Yahoo! attribute's influence based on the reverse removing order.

(e) Astro D_2 .

(f) Astro attribute's influence based on the reverse removing order.

(g) Ethylene D_2 .

(h) Ethylene attribute's influence based on the reverse removing order.

Figure 9 – Additional results presented as in Figure 8.

(a) Ethylene 2x D_2 .

(b) Ethylene 2x attribute's influence based on the reverse removing order.

(c) Hepmass D_2 .

(d) Hepmass attribute's influence based on the reverse removing order.

(e) Hepmass 2x D_2 .

(f) Hepmass 2x attribute's influence based on the reverse removing order.

(g) Hepmass 4x D_2 .

(h) Hepmass 4x attribute's influence based on the reverse removing order.

Figure 10 – Additional results presented as in Figure 8.

Table 2 – Comparing *Curl-Remover*, sPCA and Kernel PCA. The smaller the error/runtime the better. *Curl-Remover* led to **4.2%** better classification than Kernel PCA and **8%** better than sPCA.

Dataset	Method	Dec. Tree		Grad. Boosted		Rand. Forest		Log. Regr.		Avg. Error	Avg. Time (s)	
		Error	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)			
Susy	Raw	2.17E-01	46.826s	2.17E-01	46.8263s	2.30E-01	30.4667s	2.11E-01	62.1627s	2.19E-01	46.571s	
	sPCA	3.33E-01	27.68	3.07E-01	45.04	3.16E-01	32.11	2.90E-01	72.18	3.11E-01	44.25	
	Kernel PCA	3.49E-01	43.99	3.49E-01	43.99	3.60E-01	32.75	3.41E-01	54.01	3.50E-01	43.68	
	Curl-Remover	2.80E-01	44.95	2.80E-01	44.95	2.84E-01	32.47	3.38E-01	35.94	2.96E-01	39.58	
Susy 2x	Raw	2.64E-01	58.351s	2.64E-01	58.3513s	2.50E-01	48.2137s	2.54E-01	101.5597s	2.58E-01	66.619s	
	sPCA	3.59E-01	28.47	3.16E-01	43.91	3.26E-01	33.11	2.74E-01	63.67	3.19E-01	42.29	
	Kernel PCA	4.03E-01	43.05	4.03E-01	43.05	3.68E-01	31.87	4.59E-01	44.57	4.08E-01	40.63	
	Curl-Remover	2.80E-01	44.95	2.80E-01	44.95	2.84E-01	32.47	3.38E-01	35.94	2.96E-01	39.58	
Susy 8x	Raw	2.42E-01	72.648s	2.19E-01	96.578s	2.38E-01	82.172s	2.06E-01	158.218s	2.26E-01	102.404s	
	sPCA	3.48E-01	30.66	3.10E-01	51.48	3.19E-01	34.66	2.92E-01	73.36	3.17E-01	47.53	
	Kernel PCA	3.22E-01	30.23	2.92E-01	49.54	3.08E-01	34.72	2.83E-01	65.53	3.01E-01	44.01	
	Curl-Remover	3.08E-01	26.76	2.49E-01	44.62	2.54E-01	31.97	2.58E-01	33.46	2.67E-01	34.20	
Hep.	Raw	1.81E-01	41.978s	1.67E-01	66.8653s	1.92E-01	46.5407s	1.77E-01	66.8883s	1.79E-01	55.568s	
	sPCA	3.58E-01	41.70	3.26E-01	47.38	3.44E-01	43.10	2.71E-01	74.47	3.25E-01	51.66	
	Kernel PCA	2.82E-01	41.19	2.64E-01	51.88	2.73E-01	43.88	2.60E-01	107.35	2.70E-01	61.08	
	Curl-Remover	3.73E-01	39.57	2.95E-01	46.01	2.96E-01	42.15	3.29E-01	46.98	3.23E-01	43.68	
Hep. 2x	Raw	2.06E-01	62.484s	1.90E-01	85.0733s	1.84E-01	68.9630s	1.49E-01	145.7463s	1.82E-01	90.567s	
	sPCA	4.07E-01	32.34	4.45E-01	49.40	4.42E-01	37.39	2.85E-01	80.13	3.95E-01	49.82	
	Kernel PCA	2.92E-01	39.98	2.77E-01	46.31	2.83E-01	43.90	2.89E-01	141.07s	2.85E-01	67.81	
	Curl-Remover	2.80E-01	44.95	2.80E-01	44.95	2.84E-01	32.47	3.38E-01	35.93	2.96E-01	39.57	
Hep. 4x	Raw	2.06E-01	62.484s	1.90E-01	85.0733s	1.84E-01	68.9630s	1.49E-01	145.7463s	1.82E-01	90.567s	
	sPCA	4.07E-01	32.34	4.45E-01	49.40	4.42E-01	37.39	2.85E-01	80.13	3.95E-01	49.81	
	Kernel PCA	2.92E-01	39.98	2.77E-01	46.31	2.83E-01	43.90	2.89E-01	141.06	2.85E-01	67.81	
	Curl-Remover	2.46E-01	40.19	1.96E-01	54.80	2.18E-01	43.49	2.98E-01	61.91	2.40E-01	50.10	
Ethy.	Raw	2.75E-03	44.097s	9.98E-05	52.8863s	1.25E-04	47.6003s	0.00E+00	65.6217s	7.45E-04	52.551s	
	sPCA	3.29E-01	32.58	1.93E-01	49.43	2.25E-01	36.40	3.38E-01	40.56	2.71E-01	39.74	
	Kernel PCA	2.75E-01	32.55	1.05E-01	64.81	1.30E-01	35.78	5.97E-02	55.53	1.42E-01	47.17	
	Curl-Remover	2.48E-01	35.18	5.72E-02	50.25	7.71E-02	40.29	1.93E-01	45.46	1.44E-01	42.79	
Ethy. 2x	Raw	4.29E-04	59.587s	0.00E+00	71.0127s	1.33E-07	64.2057s	0.00E+00	78.9223s	1.07E-04	68.432s	
	sPCA	3.43E-01	31.70	1.83E-01	49.35	1.92E-01	36.55	3.36E-01	44.84	2.64E-01	40.61	
	Kernel PCA	2.86E-01	33.83	8.41E-02	48.68	1.08E-01	35.48	9.54E-02	54.17	1.43E-01	43.03	
	Curl-Remover	1.91E-01	32.76	1.68E-02	48.91	1.80E-02	35.99	7.69E-02	43.76	7.58E-02	40.36	
									Avg.	Raw	1.40E-01	63.38
									Avg. per	sPCA	3.14E-01	44.73
									method	KPCA	2.66E-01	50.57
										CR	2.24E-01	48.60

Table 3 – Comparing *Curl-Remover*, sPCA and Kernel PCA regarding the amount of information retained after the reduction of dimensionality. *Curl-Remover* preserved in average **88.1%** of the original values of D_2 , while sPCA retained in average solely 66.9% and Kernel PCA 72.0%.

Datasets	Curl-Remover	sPCA	KPCA
Sierpinski	98.7%	85.6%	96.9%
H. Sierpinski	96.2%	47.8%	93.6%
Yahoo! Net. Flows	85.1%	87.7%	-
Astro	94.5%	78.5%	-
Susy	95.2%	83.4%	97.0%
Susy 2x	98.3%	97.9%	97.9%
Susy 8x	93.8%	41.1%	45.1%
Hep.	83.3%	66.6%	66.6%
Hep. 2x	82.2%	68.4%	68.4%
Hep. 4x	73.7%	35.7%	44.3%
Ethy.	78.0%	55.7%	55.7%
Ethy. 2x	78.9%	54.8%	54.8%
Average	88.1%	66.9%	72.0%

deviation results are too small to be show. As we can see, there is a linear trend with the runtime

Figure 11 – Scale-up on the data size. *Curl-Remover* scales linearly as the dataset increases.

Figure 12 – Scale-up on the number of machines used for parallel processing. The runtime of *Curl-Remover* decreases linearly as the number of machines increases.

of *Curl-Remover* decreasing as the number of machines increases.

5.4 Final Considerations

This chapter presented an extensive set of experiments performed on real and synthetic datasets to evaluate the new method *Curl-Remover*, including results obtained when comparing it with existing dimensionality reduction techniques from the state-of-the-art, and scalability results that corroborate our complexity analysis. The next chapter presents the conclusions of this MSc work.

CONCLUSIONS

This MSc work investigated the hypothesis as follows:

“The use of concepts from the Theory of Fractals applied to data analysis in a massive parallel processing environment makes it feasible to identify and eliminate both linear and **non-linear** attribute correlations in billion-element-scale datasets, as well as irrelevant attributes.” (Hypothesis 1 from Chapter 1)

As a main result, we corroborated this hypothesis by presenting the new algorithm *Curl-Remover*: the first feature selection algorithm well-suited to find and remove **non-linear** attribute correlations from datasets with hundreds of millions and even billions of elements. Our main contributions are:

1. **Accuracy** – as opposed to the works in the state-of-the-art, *Curl-Remover* eliminates both linear and non-linear attribute correlations, besides irrelevant attributes;
2. **Scalability** – it presents linear scale-up on the data size and also on the number of machines used for parallel processing;
3. **Usability** – it is unsupervised, so it does not require the user to guess the number of attributes to be removed neither requires a training set, which are both rarely available for many real datasets;
4. **Semantics** – it is a feature selection algorithm, thus it maintains the original semantics of the attributes, and;
5. **Generality** – it suits for analytical tasks in general, and not only for classification.

We performed experiments on synthetic data as well as on real data from Chemistry, Physics, Astrophysics and Web-data flow applications spanning up to 1.1 billion data points. Our proposed algorithm achieved up to **8%** better accuracy compared with two PCA-based

algorithms from the state-of-the-art. In average, *Curl-Remover*, shrank the datasets studied to 34% of their original volumes, while still maintaining 88% of their Fractal Dimensionality D_2 (information). The experiments also demonstrate the linear scale-up of our algorithm, corroborating the theoretical complexity analysis that we developed for it.

Additionally, this MSc work has demonstrated the applicability of MapReduce over a major machine learning problem in the context of very large datasets. Our contributions shall enable the preprocessing of real large data, easing the application of data mining algorithms that, otherwise, would strive against issues that are intrinsic to the high dimensionality.

The work performed during this MSc program generated three publications:

- Antonio C. Fraideinberze, Jose F. Rodrigues Jr., Robson L. F. Cordeiro: Effective and Un-supervised Fractal-Based Feature Selection for Very Large Datasets: Removing Linear and Non-linear Attribute Correlations. **IEEE ICDM Workshops 2016**: 615-622 (workshop of an International Conference – **Qualis A1**)
- Afonso Expedito Da Silva, Lucas L. Sanches, Antonio C. Fraideinberze, Robson L. F. Cordeiro: Haliteds: Fast and Scalable Subspace Clustering for Multidimensional Data Streams. **SIAM SDM 2016**: 351-359 (International Conference – **Qualis A1**)
- Paulo H. Oliveira, Antonio C. Fraideinberze, Natan A. Laverde, Hugo Gualdrón, Andre S. Gonzaga, Lucas D. Ferreira, Willian D. Oliveira, Jose F. Rodrigues Jr., Robson L. F. Cordeiro, Caetano Traina Jr., Agma J. M. Traina, Elaine P. M. de Sousa: On the Support of a Similarity-enabled Relational Database Management System in Civilian Crisis Situations. **ICEIS (1) 2016**: 119-126 (International Conference – **Qualis B2**)

The first publication presents algorithm *Curl-Remover* – the main contribution of this MSc work. An improved and expanded version of this article is currently being evaluated for publication at the Elsevier Information Sciences Journal (Qualis A1). The other two publications derive from additional works that were conducted in collaboration with other researchers from the Database and Image Group – GBdI at ICMC/USP. These additional works are presented in Appendixes A and B. The candidate student's individual contributions in each of these works are also reported.

BIBLIOGRAPHY

AHA, D.; KIBLER, D.; ALBERT, M. Instance-based learning algorithms. **Machine Learning**, 1991. Cited on page 81.

ANAMI, B. S.; UNKI, P. H. Multilevel thresholding and fractal analysis based approach for classification of brain MRI images into tumour and non-tumour. **IJMEI**, v. 8, n. 1, p. 1–13, 2016. Disponível em: <<http://dx.doi.org/10.1504/IJMEI.2016.073651>>. Cited on page 26.

ASFOOR, H.; SRINIVASAN, R.; VASUDEVAN, G.; VERBIEST, N.; CORNELLS, C.; TOLENTINO, M.; TEREDESAI, A.; De Cock, M. Computing fuzzy rough approximations in large scale information systems. **2014 IEEE Int. Conf. Big Data (Big Data)**, p. 9–16, 2014. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7004350>>. Cited on page 27.

BAIOCO, G. B.; TRAINA, A. J. M.; TRAINA-JR., C. MAMCost: Global and local estimates leading to robust cost estimation of similarity queries. **Proc. Int. Conf. Sci. Stat. Database Manag. SSDBM**, n. Ssdbm, 2007. ISSN 10993371. Cited on page 25.

BALCAN, M.-F.; LIANG, Y.; SONG, L.; WOODRUFF, D.; XIE, B. Communication Efficient Distributed Kernel Principal Component Analysis. **KDD'16**, p. 1–15, 2015. Disponível em: <http://www.kdd.org/kdd2016/papers/files/Paper_{_}967.pdf?searchterm=principal+Definit>. Cited 3 times on pages 21, 22, and 32.

BALDI, P.; SADOWSKI, P.; WHITESON, D. Searching for exotic particles in high-energy physics with deep learning. **Nat Commun**, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 5, jul 2014. Disponível em: <<http://dx.doi.org/10.1038/ncomms5308>>. Cited on page 42.

BARBARA, D.; CHEN, P. Using the Fractal Dimension to Cluster Datasets. **Proceeding KDD '00 Proc. sixth ACM SIGKDD Int. Conf. Knowl. Discov. data Min.**, p. 260–264, 2000. Cited on page 25.

_____. Fractal mining-self similarity-based clustering and its applications. In: **Data Min. Knowl. Discov. Handb.** [S.l.]: Springer, 2010. p. 573–589. Cited on page 25.

BARIONI, M.; KASTER, D.; RAZENTE, H.; TRAINA, A.; Traina Jr., C. Querying Multimedia Data by Similarity in Relational DBMS. In: **Advanced Database Query Systems**. [S.l.: s.n.], 2011. Cited 3 times on pages 80, 82, and 85.

BEDO, M.; BLANCO, G.; OLIVEIRA, W.; CAZZOLATO, M.; COSTA, A.; Rodrigues Jr., J.; TRAINA, A.; Traina Jr., C. Techniques for effective and efficient fire detection from social media images. In: . [S.l.: s.n.], 2015. (ICEIS '15). ISBN 978-989-758-096-3. Cited 3 times on pages 78, 85, and 86.

BEDO, M.; TRAINA, A.; Traina Jr., C. Seamless integration of distance functions and feature vectors for similarity-queries processing. **JIDM**, 2014. Cited on page 82.

BELLMAN, R. **Dynamic Programming**. Dover Publications, 2013. (Dover Books on Computer Science). ISBN 9780486317199. Disponível em: <<https://books.google.it/books?id=CG7CAgAAQBAJ>>. Cited 2 times on pages 21 and 29.

BÖHM, C. A cost model for query processing in high dimensional data spaces. **ACM Trans. Database Syst.**, v. 25, n. 2, p. 129–178, 2000. ISSN 03625915. Cited on page 25.

BOLON-CANEDO, V.; SÁNCHEZ-MARONO, N.; ALONSO-BETANZOS, A. Recent advances and emerging challenges of feature selection in the context of Big Data. **Knowledge-Based Syst.**, v. 86, p. 33–45, may 2015. ISSN 09507051. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0950705115002002>>. Cited 2 times on pages 21 and 30.

BOUTSIDIS, C.; MAHONEY, M.; DRINEAS, P. An improved approximation algorithm for the column subset selection problem. **SODA '09 Proc. Twent. Annu. ACM-SIAM Symp. Discret. Algorithms**, p. 17, 2009. Disponível em: <[http://arxiv.org/abs/0812.4293\\$\\delimeter"026E30F\\$nhhttp://dl.acm.org/citation.cfm?id=1496875](http://arxiv.org/abs/0812.4293$\\delimeter)>. Cited on page 30.

CELIK, T.; OZKARAMANLI, H.; DEMIREL, H. Fire and smoke detection without sensors: Image processing based approach. In: . [S.l.: s.n.], 2007. (EUSIPCO '07). Cited on page 78.

CHAKRABARTI, D.; FALOUTSOS, C. F4: large-scale automated forecasting using fractals. **Int. Conf. Inf. Knowl. Manag.**, v. 1, p. 2–9, 2002. Cited on page 25.

CORDEIRO, R. L. F.; FALOUTSOS, C.; TRAINA-JR., C. **Data Mining in Large Sets of Complex Data**. [S.l.: s.n.], 2013. ISBN 9781447148890. Cited on page 21.

CORDEIRO, R. L. F.; FALOUTSOS, C.; Traina Jr., C. **Data Mining in Large Sets of Complex Data**. [S.l.]: Springer, 2013a. (SpringerBriefs in Computer Science). ISBN 9781447148890. Cited 4 times on pages 61, 62, 63, and 64.

CORDEIRO, R. L. F.; TRAINA, A. J. M.; FALOUTSOS, C.; TRAINA-JR., C. Finding clusters in subspaces of very large, multi-dimensional datasets. **Proc. - Int. Conf. Data Eng.**, p. 625–636, 2010. ISSN 10844627. Cited on page 25.

_____. Halite: Fast and scalable multiresolution local-correlation clustering. **IEEE Trans. Knowl. Data Eng.**, v. 25, n. 2, p. 387–401, 2013. ISSN 10414347. Cited on page 25.

CORDEIRO, R. L. F.; TRAINA, A. J. M.; FALOUTSOS, C.; Traina Jr., C. Halite: Fast and scalable multiresolution local-correlation clustering. **IEEE TKDE**, v. 25, n. 2, p. 387–401, 2013b. Cited 4 times on pages 62, 63, 64, and 74.

CORDEIRO, R. L. F.; Traina Jr., C.; TRAINA, A. J. M.; LÓPEZ, J.; KANG, U.; FALOUTSOS, C. Clustering very large multi-dimensional datasets with mapreduce. In: **KDD**. [S.l.]: ACM, 2011. p. 690–698. ISBN 978-1-4503-0813-7. Cited on page 61.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. **Proc. 6th Symp. Oper. Syst. Des. Implement.**, p. 137–149, 2004. ISSN 00010782. Cited on page 27.

DING, R.; WANG, Q.; DANG, Y.; FU, Q.; ZHANG, H.; ZHANG, D. YADING: Fast clustering of large-scale time series data. **VLDB**, VLDB Endowment, v. 8, n. 5, p. 473–484, jan. 2015. ISSN 2150-8097. Disponível em: <<http://dx.doi.org/10.14778/2735479.2735481>>. Cited on page 64.

DING, Y.; ZHU, G.; CUI, C.; ZHOU, J.; TAO, L. A Parallel Implementation of Singular Value Decomposition based on Map-Reduce and PARPACK. **Int. Conf. Comput. Sci. Netw. Technol.**, p. 739–741, 2011. Cited on page 32.

DOMENICONI, C.; GUNOPULOS, D.; MA, S.; YAN, B.; AL-RAZGAN, M.; PAPADOPOULOS, D. Locally adaptive metrics for clustering high dimensional data. **DMKD**, Hingham, MA, USA, v. 14, n. 1, p. 63–97, 2007. ISSN 1384-5810. Cited on page 63.

ELGAMAL, T.; YABANDEH, M.; ABOULNAGA, A.; MUSTAFA, W.; HEFEEDA, M. sPCA : Scalable Principle Component Analysis for Big Data. In: **Sigmod'15**. [S.l.: s.n.], 2015. p. 79–91. ISBN 9781450327589. Cited 3 times on pages 21, 22, and 32.

FALOUTSOS, C.; KAMEL, I. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. **Proc. Thirteen. ACM SIGACT-SIGMOD-SIGART Symp. Princ. database Syst.**, v. 8958546, p. 1–18, 1994. Disponível em: <<http://dl.acm.org/citation.cfm?id=182593>>. Cited 2 times on pages 25 and 26.

FALOUTSOS, C.; SEEGER, B.; TRAINA, A.; TRAINA-JR., C. Spatial join selectivity using power laws. **ACM SIGMOD Rec.**, v. 29, n. 2, p. 177–188, 2000. ISSN 01635808. Cited on page 25.

FARAHAT, A. K.; ELGOHARY, A.; GHODSI, A.; KAMEL, M. S. Distributed Column Subset Selection on MapReduce. **2013 IEEE 13th Int. Conf. Data Min.**, p. 171–180, 2013. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6729501>>. Cited on page 28.

FAYYAD, U. M.; PIATETSKY-SHAPIO, G.; RAMASAMY, U. Summary from the KDD-03 panel: data mining: the next 10 years. **ACM SIGKDD Explor. Newsl.**, v. 5, n. 2, p. 191–196, 2003. Cited on page 29.

FIX, E.; Hodges Jr., J. **Discriminatory analysis — Nonparametric discrimination: Consistency properties**. [S.l.], 1951. Cited on page 81.

FONOLLOSA, J.; SHEIK, S.; HUERTA, R.; MARCO, S. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. **Sensors Actuators B Chem.**, v. 215, p. 618–629, 2015. ISSN 09254005. Cited on page 42.

GHAHREMANLOU, L.; SHERCHAN, W.; THOM, J. Geotagging Twitter messages in crisis management. **The Computer Journal**, 2015. Cited on page 79.

GIBSON, H.; ANDREWS, S.; DOMDOUZIS, K.; HIRSCH, L.; AKHGAR, B. Combining big social media data and FCA for crisis response. In: . [S.l.: s.n.], 2014. (UCC '14). Cited on page 79.

GUILLÉN, A.; ARENAS, M. I. G.; HEESWIJK, M. van; SOVILJ, D.; LENDASSE, A.; HERRERA, L. J.; POMARES, H.; ROJAS, I. Fast feature selection in a GPU cluster using the delta test. **Entropy**, v. 16, n. 2, p. 854–869, 2014. ISSN 10994300. Cited on page 27.

HALDER, B. Crowdsourcing collection of data for crisis governance in the post-2015 world: Potential offers and crucial challenges. In: . [S.l.: s.n.], 2014. (ICEGOV '14). ISBN 978-1-60558-611-3. Cited on page 80.

HAMMING, R. **Error detecting and error correcting codes**. 1950. 147–160 p. Disponível em: <http://www.caip.rutgers.edu/~bushnell/dsdwebsite/hamming.p>. Cited on page 82.

HASSANI, M.; SPAUS, P.; GABER, M. M.; SEIDL, T. Density-based projected clustering of data streams. In: **SUM**. Springer-Verlag, 2012. p. 311–324. ISBN 978-3-642-33361-3. Disponível em: http://dx.doi.org/10.1007/978-3-642-33362-0_24. Cited on page 63.

HE, D.; ZHOU, Y.; SHOU, L.; CHEN, G. Cluster based rank query over multidimensional data streams. In: **CIKM**. ACM, 2009. p. 1493–1496. ISBN 978-1-60558-512-3. Disponível em: <http://doi.acm.org/10.1145/1645953.1646154>. Cited on page 61.

HUANG, M.; SMILOWITZ, K.; BALCIK, B. A continuous approximation approach for assessment routing in disaster relief. **Transportation Research Part B**, 2013. Cited on page 79.

JOLLIFFE, I. T. Principal Component Analysis. **Stat.**, v. 36, n. 4, p. 432, 1987. ISSN 00390526. Disponível em: <http://www.jstor.org/stable/10.2307/2348864?origin=crossref>. Cited on page 31.

KANG, U.; MEEDER, B.; PAPALEXAKIS, E. E.; FALOUTSOS, C. Heigen: Spectral analysis for billion-scale graphs. **IEEE TKDE**, v. 26, n. 2, p. 350–362, 2014. Cited on page 61.

KASTER, D.; BUGATTI, P.; PONCIANO-SILVA, M.; TRAINA, A.; MARQUES, P.; SANTOS, A.; Traina Jr., C. MedFMI-SiR: A powerful DBMS solution for large-scale medical image retrieval. In: . [S.l.: s.n.], 2011, (ITBAM '11). Cited on page 82.

KRIEGEL, H.-P.; KRÖGER, P.; ZIMEK, A. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. **ACM TKDD**, ACM, New York, NY, USA, v. 3, n. 1, p. 1:58, 2009. ISSN 1556-4681. Disponível em: <http://doi.acm.org/10.1145/1497577.1497578>. Cited 2 times on pages 62 and 63.

KUDYBA, S. **Big Data, Mining, and Analytics: Components of Strategic Decision Making**. [S.l.: s.n.], 2014. Cited on page 77.

LAHMIRI, S. Glioma detection based on multi-fractal features of segmented brain MRI by particle swarm optimization techniques. **Biomed. Signal Proc. and Control**, v. 31, p. 148–155, 2017. Disponível em: <http://dx.doi.org/10.1016/j.bspc.2016.07.008>. Cited on page 26.

LEE, J. W.; LEE, W. S. A coarse-grain grid-based subspace clustering method for online multi-dimensional data streams. In: **CIKM**. ACM, 2008. p. 1521–1522. ISBN 978-1-59593-991-3. Disponível em: <http://doi.acm.org/10.1145/1458082.1458366>. Cited 2 times on pages 61 and 63.

LEE, W.; CHANG, K.; HSIEH, K. Unsupervised segmentation of lung fields in chest radiographs using multiresolution fractal feature vector and deformable models. **Med. Biol. Engineering and Computing**, v. 54, n. 9, p. 1409–1422, 2016. Disponível em: <http://dx.doi.org/10.1007/s11517-015-1412-6>. Cited on page 26.

LEHOUCQ, R.; SORENSEN, D.; YANG, C. **ARPACK Users' Guide**. Society for Industrial and Applied Mathematics, 1998. Disponível em: <http://epubs.siam.org/doi/abs/10.1137/1.9780898719628>. Cited on page 32.

MEHROTRA, S.; BUTTS, C.; KALASHNIKOV, D.; VENKATASUBRAMANIAN, N.; RAO, R.; CHOCKALINGAM, G.; EGUCHI, R.; ADAMS, B.; HUYCK, C. Project Rescue: Challenges in responding to the unexpected. In: . [S.l.: s.n.], 2004. (EI '04). Cited on page 80.

MESHCHERYAKOVA, E. I.; LARIONOVA, A. V. Fractal computer visualization in psychological research. **AI Soc.**, v. 32, n. 1, p. 121–133, 2017. Disponível em: <<http://dx.doi.org/10.1007/s00146-016-0658-3>>. Cited on page 26.

MILLER, Z.; HU, W. Data stream subspace clustering for anomalous network packet detection. **J. Information Security**, v. 3, n. 3, p. 215–223, 2012. Cited on page 63.

MOISE, G.; SANDER, J. Finding non-redundant, statistically significant regions in high dimensional data. In: **KDD**. [S.l.: s.n.], 2008. p. 533–541. Cited on page 63.

MOISE, G.; SANDER, J.; ESTER, M. Robust projected clustering. **KIS**, Springer, New York, NY, USA, v. 14, n. 3, p. 273–298, 2008. ISSN 0219-1377. Cited on page 63.

MULTIMEDIA, I. MPEG-7: The generic multimedia content description standard, p. 1. **IEEE MultiMedia**, 2002. Cited on page 81.

NG, E. K. K.; FU, A. W. chee; WONG, R. C.-W. Projective clustering by histograms. **TKDE**, Piscataway, NJ, USA, v. 17, n. 3, p. 369–383, 2005. ISSN 1041-4347. Cited on page 63.

NTOUTSI, I.; ZIMEK, A.; PALPANAS, T.; KRÖGER, P.; KRIEGEL, H. Density-based projected clustering over high dimensional data streams. In: **SDM**. SIAM, 2012. p. 987–998. ISBN 978-1-61197-232-0. Disponível em: <<http://dx.doi.org/10.1137/1.9781611972825.85>>. Cited 3 times on pages 61, 62, and 63.

NUNES, S. a.; ROMANI, L. a. S.; AVILA, A. M. H.; COLTRI, P. P.; CORDEIRO, R. L. F.; SOUSA, E. P. M. D.; TRAINA, A. J. M. Analysis of Large Scale Climate Data: How Well Climate Change Models and Data from Real Sensor Networks Agree? **Proceeding WWW '13 Companion Proc. 22nd Int. Conf. World Wide Web**, p. 517–526, 2010. Cited on page 25.

NUNES, S. A.; ROMANI, L. A. S.; de Ávila, A. M. H.; COLTRI, P. P.; Traina Jr., C.; CORDEIRO, R. L. F.; SOUSA, E. P. M. de; TRAINA, A. J. M. Analysis of large scale climate data: how well climate change models and data from real sensor networks agree? In: **WWW Companion Volume**. ACM, 2013. p. 517–526. ISBN 978-1-4503-2038-2. Disponível em: <<http://dl.acm.org/citation.cfm?id=2487986>>. Cited on page 61.

OLIVEIRA, P. H.; FRAIDEINBERZE, A. C.; LAVERDE, N. A.; GUALDRON, H.; GONZAGA, A. S.; FERREIRA, L. D.; OLIVEIRA, W. D.; JR., J. F. R.; CORDEIRO, R. L. F.; JR., C. T.; TRAINA, A. J. M.; SOUSA, E. P. M. de. On the support of a similarity-enabled relational database management system in civilian crisis situations. In: **ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems, Volume 1, Rome, Italy, April 25-28, 2016**. [s.n.], 2016. p. 119–126. Disponível em: <<http://dx.doi.org/10.5220/0005816701190126>>. Cited on page 77.

ORDOZGOITI, B.; CANAVAL, S. G.; MOZO, A. Massively Parallel Unsupervised Feature Selection on Spark. **New Trends Databases Inf. Syst.**, v. 539, p. 186–196, 2015. Cited on page 30.

ÖZCAN, F. (Ed.). **Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005**. [S.l.]: ACM, 2005. ISBN 1-59593-060-4. Cited on page 22.

PARK, N. H.; LEE, W. S. Grid-based subspace clustering over data streams. In: **CIKM**. ACM, 2007. p. 801–810. ISBN 978-1-59593-803-9. Disponível em: <<http://doi.acm.org/10.1145/1321440.1321551>>. Cited on page 63.

PAWLAK, Z. Rough sets. **Int. J. Comput. Inf. Sci.**, p. 1–51, 1982. ISSN 1558-0032. Disponível em: <<http://link.springer.com/article/10.1007/BF01001956>>. Cited on page 30.

QIAN, J.; LV, P.; YUE, X.; LIU, C.; JING, Z. Hierarchical attribute reduction algorithms for big data using MapReduce. **Knowledge-Based Syst.**, v. 73, p. 18–31, jan 2015. ISSN 09507051. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0950705114003311>>. Cited on page 28.

QIAN, J.; MIAO, D.; ZHANG, Z.; YUE, X. Parallel attribute reduction algorithms using MapReduce. **Inf. Sci. (Ny)**, v. 279, p. 671–690, sep 2014. ISSN 00200255. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0020025514004666>>. Cited on page 30.

QIAN, Q.; XIAO, C.; ZHANG, R. Grid-based data stream clustering for intrusion detection. **I. J. Network Security**, v. 15, n. 1, p. 1–8, 2013. Disponível em: <<http://ijns.femto.com.tw/contents/ijns-v15-n1/ijns-2013-v15-n1-p1-8.pdf>>. Cited on page 64.

REZAIE, A. A.; HABIBOGLI, A. Detection of lung nodules on medical images by the use of fractal segmentation. **IJIMAI**, v. 4, n. 5, p. 15–19, 2017. Disponível em: <<http://dx.doi.org/10.9781/ijimai.2017.452>>. Cited on page 26.

REZNIK, T.; HORAKOVA, B.; SZTURC, R. Advanced methods of cell phone localization for crisis and emergency management applications. **IJDE**, 2015. Cited on page 79.

SALGADO, A. C.; MOTTA, C. L. R.; SANTORO, F. M. Grandes Desafios da Computação no Brasil. 2015. Disponível em: <<http://sbc.org.br/documentos-da-sbc/send/141-grandes-desafios/798-grandesdesafios-portugues>>. Cited on page 21.

SCHÖLKOPF, B.; SMOLA, A.; MÜLLER, K.-R. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. **Neural Comput.**, v. 10, n. 5, p. 1299–1319, 1998. ISSN 0899-7667. Cited on page 32.

SCHROEDER, M. R. **Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise**. Dover Publications, Incorporated, 2012. ISBN 9780486134789. Disponível em: <<https://books.google.com.br/books?id=Qpa77Jl2rvQC>>. Cited 2 times on pages 25 and 26.

SIKORA, T. The MPEG-7 visual standard for content description — An overview. **IEEE Trans. Cir. Sys. Vid.**, 2001. Cited on page 81.

SILVA, A. E. D.; SANCHES, L. L.; FRAIDEINBERZE, A. C.; CORDEIRO, R. L. F. *Halite_{ds}*: Fast and scalable subspace clustering for multidimensional data streams. In: **Proceedings of the 2016 SIAM International Conference on Data Mining, Miami, Florida, USA, May 5-7, 2016**. [s.n.], 2016. p. 351–359. Disponível em: <<http://dx.doi.org/10.1137/1.9781611974348.40>>. Cited on page 61.

SILVA, Y.; ALY, A.; AREF, W.; LARSON, P. SimDB: A similarity-aware database system. In: . [S.l.: s.n.], 2010. (SIGMOD '10). ISBN 978-1-4503-0032-2. Cited on page 82.

SOUSA, E. P.; TRAINA, A. J.; Traina Jr., C.; FALOUTSOS, C. Measuring evolving data streams' behavior through their intrinsic dimension. **New Generation Computing**, Verlag Omsha Tokio, v. 25, n. 1, p. 33–59, 2007. ISSN 0288-3635. Disponível em: <<http://dx.doi.org/10.1007/s00354-006-0003-3>>. Cited on page 64.

SOUSA, E. P. M. D.; TRAINA, A. J. M.; TRAINA-JR., C.; FALOUTSOS, C. Measuring Evolving Data Streams' Behavior through Their Intrinsic Dimension. **New Gener. Comput.**, v. 25, n. 1, p. 33–60, 2007. ISSN 0288-3635. Cited on page 25.

STEWART, G. W. On the early history of the singular value decomposition. **SIAM Rev.**, SIAM, v. 35, n. 4, p. 551–566, 1993. Cited on page 31.

SUN, Z.; LI, Z. Data Intensive Parallel Feature Selection Method Study. **Neural Networks (IJCNN), 2014 Int. Jt. Conf.**, 2014. Cited on page 30.

TAYLOR, R. P.; MICOLICH, A. P.; NEWBURY, R.; BIRD, J. P.; FROMHOLD, T. M.; COOPER, J.; AOYAGI, Y.; SUGANO, T. Exact and statistical self-similarity in magnetoconductance fluctuations: A unified picture. **Phys. Rev. B**, v. 58, n. 17, p. 11107–11110, 1998. ISSN 0163-1829. Disponível em: <<http://link.aps.org/doi/10.1103/PhysRevB.58.11107>>. Cited 2 times on pages 22 and 25.

TIPPING, M. E.; BISHOP, C. M. Probabilistic Principal Component Analysis. **J. R. Stat. Soc.**, n. iii, p. 611–622, 1999. Cited on page 32.

TRAINA-JR, C.; TRAINA, A.; WU, L.; FALOUTSOS, C. Fast Feature Selection using Fractal Dimension. **J. Inf. Data Manag.**, v. 1, n. 1, p. 3–16, 2000. Disponível em: <<http://repository.cmu.edu/compsci/580/>>. Cited 2 times on pages 22 and 30.

TRAINA-JR., C.; TRAINA, A. J. M.; FALOUTSOS, C. Fast Feature Selection using Fractal Dimension - Ten Years Later. **JIDM**, v. 1, n. 1, p. 17–20, 2010. Disponível em: <<http://seer.lcc.ufmg.br/index.php/jidm/article/view/29>>. Cited on page 30.

TRAINA-JR., C.; TRAINA, A. J. M.; WU, L.; FALOUTSOS, C. Fast feature selection using fractal dimension. **JIDM**, v. 1, n. 1, p. 3–16, 2010. Disponível em: <<http://seer.lcc.ufmg.br/index.php/jidm/article/view/4>>. Cited 3 times on pages 26, 27, and 30.

TUNG, A. K. H.; XU, X.; OOI, B. C. CURLER: finding and visualizing nonlinear correlated clusters. In: ÖZCAN, F. (Ed.). **Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005**. ACM, 2005. p. 467–478. ISBN 1-59593-060-4. Disponível em: <<http://doi.acm.org/10.1145/1066157.1066211>>. Cited on page 22.

WANG, X.; LIU, W.; LI, J.; GAO, X. A novel dimensionality reduction method with discriminative generalized eigen-decomposition. **Neurocomputing**, v. 173, p. 163–171, 2016. Cited 2 times on pages 30 and 32.

WILSON, D.; MARTINEZ, T. Improved heterogeneous distance functions. **J. Artif. Int. Res.**, 1997. Cited on page 82.

XU, C. A novel spatial clustering method based on wavelet network and density analysis for data stream. **Journal of Computers**, v. 8, n. 8, p. 2139–2143, 2013. Disponível em: <<http://www.ojs.academypublisher.com/index.php/jcp/article/view/jcp080821392143>>. Cited on page 64.

YANYUN, C.; JIANLIN, Q.; JIANPING, C.; LI, C.; YANG, P. A parallel rough set attribute reduction algorithm based on attribute frequency. **Fuzzy Syst. Knowl. Discov. (FSKD)**, 2012 **9th Int. Conf.**, n. Fskd, p. 211–215, 2012. Cited on page 30.

ZAHARIA, M.; CHOWDHURY, M.; DAS, T.; DAVE, A. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. **NSDI'12 Proc. 9th USENIX Conf. Networked Syst. Des. Implement.**, p. 2–2, 2012. ISSN 00221112. Disponível em: <<https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf>>. Cited on page 28.

ZHANG, J.; LI, J.; LIU, Z. Multiple-resource and multiple-depot emergency response problem considering secondary disasters. **Expert Syst. Appl.**, 2012. Cited on page 79.

ZHANG, Q.; LIU, J.; WANG, W. Incremental subspace clustering over multiple data streams. In: **ICDM**. [S.l.: s.n.], 2007. p. 727–732. ISSN 1550-4786. Cited on page 63.

ZHAO, Z.; ZHANG, R.; COX, J.; DULING, D.; SARLE, W. Massively parallel feature selection: An approach based on variance preservation. **Mach. Learn.**, v. 92, n. 1, p. 195–220, 2013. ISSN 08856125. Cited on page 30.

ZHU, H.; DING, S.; XU, X.; XU, L. A parallel attribute reduction algorithm based on Affinity Propagation clustering. **J. Comput.**, v. 8, n. 4, p. 990–997, 2013. ISSN 1796-203X. Disponível em: <<http://ojs.academypublisher.com/index.php/jcp/article/view/7916>>. Cited on page 30.

FAST AND SCALABLE SUBSPACE CLUSTERING FOR MULTIDIMENSIONAL DATA STREAMS

A.1 Initial Considerations

This appendix presents one novel clustering algorithm that was developed in parallel with the main proposal of the MSc program, in collaboration with other researchers from the Database and Image Group – GBdI at ICMC/USP. The work generated a full paper ([SILVA *et al.*, 2016](#)) presented at the SIAM International Conference on Data Mining - SDM 2016 (**Qualis A1**).

The candidate student's individual contributions in this particular work are: (a) fine-tuning and testing the algorithm; (b) performing nearly half of the experimental evaluation reported, and; (c) contributing to write the paper.

A.2 Problem and Motivation

The volume of information generated or collected in diverse areas of science has been increasing not only in the quantity of data objects, but also in the number of attributes used to describe each object as well as in the complexity of each of these attributes ([KANG *et al.*, 2014](#); [CORDEIRO; FALOUTSOS; Traina Jr., 2013a](#); [CORDEIRO *et al.*, 2011](#)). Gathering the data is also in many cases a continuous, repetitive and potentially infinite process, in which the attributes of interest are measured in distinct timestamps. The resulting datasets are known as **multidimensional data streams** ([NUNES *et al.*, 2013](#); [NTOUTSI *et al.*, 2012](#); [HE *et al.*, 2009](#); [LEE; LEE, 2008](#)), according to Definition 1. In this scenario, clustering techniques are among the most useful tools available to help us to analyze, to comprehend and to discover

knowledge from the data. For example, how to cluster decades of frequent measurements of tens of climatic attributes, like temperature, precipitation of rain and so on, aimed at aiding **real time** alert systems in forecasting extreme climatic events, such as floods and hurricanes?

Definition 1. One **multidimensional data stream** is a potentially unbounded sequence of events $\langle e_1, e_2, \dots \rangle$ ordered in time. Each event e_i is a list of d attribute values collected at timestamp i , i.e., $e_i = (a_1, a_2, \dots, a_d)$. d is the dimensionality of the stream.

It is easy to note that any data stream can be seen as a static dataset, as long as a limited period of time is specified for the analysis. Thus, traditional algorithms that were originally developed to analyze static data can also be used for streams, by processing in separate data subsets that correspond to specific time intervals already in the past, in such a way that the analysis of consecutive intervals allows us to understand the temporal evolution of the data. Nevertheless, this approach has one central limitation: the knowledge obtained from previous analyses is usually ignored when analyzing newer data, thus increasing the overall computational cost and decreasing the accuracy of results.

As a consequence, new analytical algorithms have been developed for streams, mainly targeted at minimizing the aforementioned limitation. When considering the task of clustering multidimensional data with more than five or so attributes – known in literature as **subspace clustering** (KRIEGEL; KRÖGER; ZIMEK, 2009), it is easy to note the need for novel algorithms well-suited to process streams, provided that the vast majority of works addressing this subject focuses on static data only (NTOUTSI *et al.*, 2012). Additionally, none of the few existing approaches that target streams is scalable. Therefore, to the best of our knowledge, it is currently unfeasible to process data streams with many attributes and high frequency of events in **real time**.

A.3 Contributions

This work proposes the new algorithm *Halite_{ds}*: one fast, scalable and highly accurate **subspace clustering** algorithm well-suited to analyze **multidimensional data streams** of moderate-to-high dimensionality. It improves upon an existing technique named *Halite* (CORDEIRO; FALOUTSOS; Traina Jr., 2013a; CORDEIRO *et al.*, 2013b) that was originally designed to cluster static (not streams) datasets. Our main contributions are:

1. **Analysis of Data Streams:** our new *Halite_{ds}* takes advantage of the knowledge obtained from clustering past data to easily cluster data in the present. This fact allows *Halite_{ds}* to be considerably faster than its base algorithm, yet obtaining the same accuracy of results;
2. **Real Time Processing:** As opposed to the state-of-the-art, our *Halite_{ds}* is fast and scalable, making it feasible to analyze data streams with many attributes and high frequency of events in real time.

3. **Experiments:** We ran experiments on synthetic data and on a real multidimensional stream with almost one century of climatic data. *Halite_{ds}* was up to 217 times faster than 5 representative works, i.e., its base algorithm plus 4 others from the state-of-the-art, always presenting top-quality results.

The rest of this appendix follows a traditional organization: background concepts and related works (Sections A.4 and A.5); proposed techniques (Section A.6); experimental evaluation (Section A.7); and conclusions (Section A.8).

A.4 Background Concepts and Related Works

Real data of dimensionality above five or so tend to have many **local correlations**, as some points are commonly correlated with regard to a given set of attributes, while other points are correlated in distinct attributes (DOMENICONI *et al.*, 2007; KRIEGEL; KRÖGER; ZIMEK, 2009). As a consequence, the data usually have clusters that exist only in **subspaces** of the original feature space (i.e., sets of orthogonal vectors formed from original attributes or from subset combinations thereof) and each cluster may exist in a distinct subspace (MOISE; SANDER, 2008; NG; FU; WONG, 2005; MOISE; SANDER; ESTER, 2008; KRIEGEL; KRÖGER; ZIMEK, 2009). Many algorithms perform subspace clustering in static (not streams) datasets. One well-known survey is in (KRIEGEL; KRÖGER; ZIMEK, 2009). There are two distinct approaches: bottom-up and top-down. Bottom-up methods, like *P3C* (MOISE; SANDER; ESTER, 2008) and *EPCH* (NG; FU; WONG, 2005), divide 1-dimensional data projections into a user-defined number of partitions and merge dense partitions to spot clusters in subspaces of higher dimensionality. On the other hand, top-down methods like *LAC* (DOMENICONI *et al.*, 2007), *STATPC* (MOISE; SANDER, 2008) and our base algorithm *Halite* (CORDEIRO; FALOUTSOS; Traina Jr., 2013a; CORDEIRO *et al.*, 2013b) analyze the “full dimensional” space looking for patterns that may lead to clusters. The data distribution surrounding these patterns allow the algorithm to confirm the clusters and to spot their subspaces – the axes in which a cluster is denser form its subspace.

Subspace clustering algorithms aimed at analyzing **multidimensional data streams** are rare in the literature. *StreamPreDeCon* (MILLER; HU, 2012), *ST-Tree* (LEE; LEE, 2008; PARK; LEE, 2007), *δ -CC-Cluster* (ZHANG; LIU; WANG, 2007), *HDDStream* (NTOUTSI *et al.*, 2012) and *PreDeConStream* (HASSANI *et al.*, 2012) are among the state-of-the-art. *StreamPreDeCon* focuses specifically on the detection of anomalous data packages in streams of packages from communication networks. *ST-Tree* combines grid-based clustering and frequent itemset mining to analyze streams. *δ -CC-Cluster* is an incremental algorithm with a model to describe clusters and to detect possible changes in clustering assignments with regard to specific subsets of events. *HDDStream* is a density-based clustering algorithm that summarizes both the input events and the subspaces in which these events are grouped together. It keeps the summaries in RAM to

process new events that arrive over time, also considering that old events expire due to aging. *PreDeConStream* uses a two phase strategy for mining streams: the online phase keeps one microcluster data structure updated, which is periodically passed to the offline phase to refine the clustering model.

Algorithms designed to perform traditional clustering (i.e., **not** subspace clustering) in multidimensional data streams also exist. Relevant examples are (DING *et al.*, 2015), (XU, 2013) and (QIAN; XIAO; ZHANG, 2013). Note, however, that the algorithms in this category are **not** well-suited to process streams with more than five or so attributes, since they do not spot clusters that only exist in subspaces of the original data space.

It is worth noting similarities among our proposed *Halite_{ds}* and algorithm *SID-meter* (SOUSA *et al.*, 2007a). Both *Halite_{ds}* and *SID-meter* use a *quad-tree-like* data structure to analyze streams, which is made possible by self-adjusting the tree as new data arrive. Note, however, that *SID-meter* focuses on estimating intrinsic dimensionality by means of fractal data analysis – it does **not** perform clustering.

After revising the literature, we conclude that: to the best of our knowledge, there is no **subspace clustering** algorithm for **multidimensional data streams** that **scales linearly** in runtime and in memory usage with regard to the data size and dimensionality. This work tackles this relevant limitation aimed at making it feasible to process streams with many attributes and high frequency of events in **real time**.

A.5 The base algorithm

The base algorithm *Halite* (CORDEIRO; FALOUTSOS; Traina Jr., 2013a; CORDEIRO *et al.*, 2013b) is a state-of-the-art, subspace clustering algorithm well-suited to process static (not streams) datasets of moderate-to-high dimensionality in a fast, scalable and accurate manner. It has two main phases: the first one builds a multidimensional, *quad-tree-like* structure from the input data; in the second phase, the tree is analyzed to spot clusters formed in subspaces of the original feature space.

A.5.1 First phase

The initial phase reads one static dataset dS containing $n = |^dS|$ objects with d attributes each. Accounting the objects as points in the d -dimensional space, one multidimensional, *quad-tree-like* structure is built in main memory to indicate how the input objects are distributed in that space. The resulting structure is named as the *Counting Tree*. It is assumed that each attribute value in the input dataset dS is a real, normalized value within the range $[0, 1)$. Thus, dS is included in the unitary hypercube $[0, 1)^d$.

The number H of tree levels (or resolutions) is defined according to the needs of the

application. $H = 4$ is the default. The root node (level 0) represents the entire dataset, i.e., the unitary hypercube $[0, 1)^d$. This node is divided in 2^d hypercubes in the next tree level (level 1) following a process that splits it in half with regard to each one of the d dimensions, so that each new hypercube has a side size that is half of the root's side size. Then, each hypercube of level 1 is also divided in 2^d hypercubes for the next tree level (level 2). The division is performed recursively until the last level $H - 1$. Each hypercube in the tree stores the count of points that fall within it, being known as a counting cell. The cell structure has fields loc , n , $P[\]$, $usedCell$ and ptr . The cell position loc locates the cell inside its parent cell. It is a binary number with d bits of the form $[bb \dots b]$, where the j -bit sets the cell in the lower (0) or upper (1) half of axis j relative to its parent. n is the count of points that fall within the cell. $P[\]$ is an array of d integers that stores the count of points within the lower half of the cell, regarding each one of the d dimensions. $usedCell$ is a boolean value used in the clustering phase only. And, ptr is a pointer to the next tree level.

In our notation, b_h is a counting cell b from tree level h . Its “ancestral” cells in the previous levels are $b_{h-1}, b_{h-2}, \dots, b_0$. Cell b_0 is always the tree root. Figure 13a illustrates the 2-dimensional space divided into cells of distinct resolution levels, together with their corresponding values for loc . Let the cell highlighted in gray be b_3 . Cell b_3 is in level 3, and it has $loc = 11$. Its “ancestors” b_2, b_1 and b_0 are the cell with $loc = 11$ in level 2, the cell with $loc = 01$ in level 1 and the tree root, respectively. In the *Counting Tree*, any data point that falls within any cell b_h is also counted in its “ancestors” from all previous levels. The counting of points is exemplified in Figures 13b and 13c. Figure 13b shows nine points plotted in the 2-dimensional space divided up to resolution level 3. The corresponding *Counting Tree* is in Figure 13c, whose nodes have $2^d = 2^2 = 4$ cells. Note that field $usedCell$ is omitted for better visualization.

The tree is created in main memory, and each of its nodes is usually implemented as a linked list of cells, or a memory-based, key-value index structure like a red-black tree using loc as the key. Although the number of regions to divide the space “explodes” at $O(2^{dH})$, the tree only stores/subdivides the cells with at least one point. Thus, each tree level has in fact at most n cells.

A.5.2 Second phase

The second phase takes the *Counting Tree* as input – there is no additional pass over the data objects. The tree is used to spot clusters based on the variation of the data density over the feature space in a multi-resolution way, dynamically changing the partitioning size of the analyzed regions. Distinct resolutions are represented by distinct tree levels. A convolution process using Laplacian filters is performed on each tree level to spot bumps in the data distribution regarding each resolution. Given a tree level, the filter is applied to find the regions in the “full dimensional” space with the largest changes in the point density. They may indicate clusters that only exist

Figure 13 – 2-dimensional hypercube cells and the corresponding *Counting Tree*.

in subspaces of the analyzed space. The neighborhoods of these regions are then analyzed to define if the regions stand out in the data in a statistical sense, thus confirming the clusters, and a compression-based analysis of the data distribution spots each cluster’s subspace. Finally, alternative cluster entropies are evaluated to create both “hard” and “soft” clustering results.

A.6 Proposed method

Here, we present the new algorithm *Halite_{ds}*. It adds a series of improvements on the base algorithm’s first phase to take advantage of the knowledge obtained from clustering past data to easy clustering data in the present. The second phase is reused unaltered. As a consequence, our *Halite_{ds}* is considerably faster, yet obtaining the same accuracy of results. The improvements that we propose are described as follows.

A.6.1 Dealing with one sliding window of time

The base algorithm was designed to process static datasets. It reads the input data objects one by one to build the *Counting Tree*. The clustering itself is initiated only after reading the entire dataset. On the other hand, data streams are potentially infinite by their very definition, which means that one cannot pass by all events from an input stream for posterior analysis. In many applications, it is also useless/meaningless to analyze streams as a whole. To make the analysis possible, one must be able to spot clusters in subsets of events, instead of reading the whole set first.

Our proposed *Halite_{ds}* deals with multidimensional data streams following Definition 1. To analyze one stream over time, it uses a sliding window of time that bounds successive events

Figure 14 – Sliding window of size 100 units of time ($n_p = 4$ and $n_e = 25$) over a 3-dimensional data stream.

to be considered in the search for clusters. The window is divided into n_p periods, each one containing a predetermined number of events n_e (or units of time), such that whenever n_e new events arrive, the n_e oldest ones are discarded and the clustering results are updated. Therefore, $n_p * n_e$ is the length of the window and n_e is the step by which it moves. For example, Figure 14 illustrates a 3-dimensional data stream processed through a sliding window of size 100 units of time ($n_p = 4$ and $n_e = 25$). The clustering results are computed for the events inside the window, being updated right after each of its movements, thus efficiently monitoring the evolving data behavior. The search for clusters is continuously performed, right after each movement of the window. The size of the window and its movement step are user-defined parameters, and they can easily represent time intervals, such as monthly windows/periods according to the needs of the application¹.

The original *Counting Tree* must be improved to implement the sliding window model. After carefully studying the base algorithm described in Section A.5, we noticed that the time required to build the tree refers to nearly 90% of the entire clustering process. Thus, the base algorithm's first phase is by far the main bottleneck. See Figure 19 and the upcoming experimental Section A.7 for details. In our setting with streams of data, this fact clearly indicates that we should be able to analyze consecutive time intervals **without** having to reconstruct the tree for each new interval. In this way, it would be possible to take advantage of the knowledge obtained from clustering past data to easily cluster data in the present. Unfortunately, the base algorithm does not support this procedure.

As we previously described in Section A.5.1, field n of each tree cell stores the count of incident points in the cell, while field $P[]$ stores partial counts of points, i.e., it counts the points that fall within the lower half of the cell with regard to each attribute. Note that these counts alone are not enough to efficiently deal with a sliding window, since they do not indicate the time

¹ The smallest window used in our experiments has $\sim 1k$ events, from which meaningful clusters were successfully found.

period in which each point counted occurred. Here, we overcome this limitation by replacing field n in each cell of the tree with a circular list of n_p independent counts of points, so to keep track of the points occurring in each time period to be considered in the analysis. Similarly, each partial count $P[j]$ referring to each attribute j also becomes a circular list. For a given position of the window, the head of each one of these lists always stores the count of points referring to its oldest time period, while the tail represents the newest period. A movement of the window is efficiently performed by overwriting the head of each list with the correspondent point count for the new time period to be considered, also setting the next element of the list to be the new head, and defining the former head element as the new tail. In our current implementation, an array $n[]$ of n_p integer values replaces field n from the original tree, while field $P[]$ is replaced by a 2-dimensional matrix $P[][]$ with $d * n_p$ integer values.

By improving the tree to implement one sliding window of time, we make our *Halite_{ds}* able to analyze the most recent events from a data stream without the need to reconstruct the whole tree at each new movement of the window, thus minimizing the main bottleneck of the entire process. In fact, the larger is the number of periods in the window the larger is our gain, provided that we overwrite the oldest period previously considered with a new one in each movement of the window, always leaving all the intermediary periods intact in the tree. In this way, we efficiently support applications that receive intermittent events to be clustered without the need to perform relatively expensive operations that were originally developed for the analysis of static datasets. Note, however, that we still must answer two relevant questions to make this improvement feasible for real applications:

1. *How to work with non-normalized data in the Counting Tree?* In real streams of data, we rarely know the minimum and maximum values that each attribute will assume in the future, thus, we cannot work with previously defined normalization, as it is done in the base clustering algorithm.
2. *How to efficiently represent in the tree expansions and contractions of the attribute space?* Provided that we cannot forecast the attributes' minimum and maximum bounds, new arriving events may force us to expand the attribute space covered by the root of the tree, while the removal of old events can lead to space contractions, which must be efficiently represented in the tree.

Algorithm 19 has the pseudo-code that we propose to efficiently move the sliding window, already taking into account solutions that we developed to answer both questions. Our solutions are detailed in the following.

A.6.2 Non-normalized data analysis

The base algorithm takes as input normalized data objects with real values between 0 and 1 in all of their attributes. Then, the objects are represented in the *Counting Tree*, always setting the root node to cover the unitary hypercube $[0, 1)^d$. The spatial bounds of each tree cell are therefore invariant and predefined, **regardless** of the dataset received as input. Nevertheless, many real streams of data have unknown minimum and maximum limit values for their attributes, making it impossible to normalize their events on the fly. *How to efficiently represent non-normalized events in the Counting Tree, dealing with attributes that can assume any real value?* In other words, *how to define the initial attribute space to be covered by the tree, and how to make it dynamic, without rebuilding the tree at each new expansion or contraction of the space?*

Algorithm 6: Moves the sliding window.

```

1 Procedure moveWindow(tree, newEvents)
2 Input: tree: Counting Tree for the window in its
3           previous position;
4           newEvents: set of events occurring in the new
5           time period to be considered;
6 Output: tree: modified tree, now representing the
7           window in its new position;
8 begin
9   Discard events of the oldest period in tree
10  for each event  $e_i$  in newEvents do
11    if  $\exists a_j \in e_i : (a_j < tree.L_j \vee a_j > tree.U_j)$  then
12      //  $e_i$  is outside the coverage of tree
13      expandTree(tree,  $e_i$ ); //expand coverage
14      Insert event  $e_i$  in tree counting it for the new
15      time period to be considered;
16  end
17  if tree.root has one single “child” only then
18    contractTree(tree); // contract coverage
19 end

```

To answer these questions, we propose here another improvement on the original *Counting Tree*. In the new algorithm *Halite_{ds}*, the root of the tree represents one hypercube of varying side size r_0 . The position of the root’s hypercube also varies into the infinite d -dimensional space $\mathbb{R}^d$. The idea is to allow the actual attribute space covered by the tree to change as new events are received and old ones are discarded, thus dynamically representing the temporal evolution of the data. Both the initial value of r_0 and the initial position of the root’s hypercube are defined by the attributes’ active domains in the first set of events read, i.e., those events in the first period of the sliding window at its initial position. Parameter r_0 is initialized by Equation A.1, in which *largest_j* and *smallest_j* are respectively defined in Equations A.2 and A.3. The position of the root’s hypercube into space $\mathbb{R}^d$ is defined by parameters L_j and U_j , respectively referring to its minimum and maximum limits in each attribute j . The root’s initial position is $L_j = \text{smallest}_j$

Figure 15 – Dynamic evolution of the attribute space versus tree coverage. Depicting events taken into account before (top) and after (bottom) the time window slides. (a): the space covered by the *Counting Tree* is still adequate; (b): coverage must be expanded; (c): coverage should be contracted for better representation.

and $U_j = L_j + r0$, for every attribute j . As we mentioned before, the attribute space covered by the *Counting Tree* must vary on the fly to analyze streams, and so we allow the initial values of $r0$, and of L_j and U_j for each attribute j to change over time, by following two novel algorithms that we propose latter, in Section A.6.3. The current values are always stored along with the tree itself.

$$r0 = \max(\{ largest_1 - smallest_1, \\ largest_2 - smallest_2, \dots \\ largest_d - smallest_d \}) \quad (A.1)$$

$$largest_j = \max(\{e_1.a_j, e_2.a_j, \dots e_{n_e}.a_j\}) \quad (A.2)$$

$$smallest_j = \min(\{e_1.a_j, e_2.a_j, \dots e_{n_e}.a_j\}) \quad (A.3)$$

A.6.3 Efficiently representing space expansions and contractions

The attribute space covered by one data stream evolves in time, which may expand, contract or even remain untouched while old events are disposed and new ones arrive. Figure 15 illustrates the three possible categories of evolution: (a) space preservation; (b) space expansion, and; (c) space contraction. In our illustration, the top row represents the events taken into account before one movement of the time window; the grids indicate the attribute space covered by the corresponding *Counting Tree*, and; the bottom row depicts the new reality after the movement. In all categories of evolution, old events are discarded (regions in light-gray), not-so-old events are preserved (in dark-gray) and new events arrive (in black).

In Figure 15a, none of the new events fall outside the already represented attribute space, and the vast majority of that space is still covered by events after the window slides. Thus, the existing coverage can be preserved. Unfortunately, more challenging cases exist, as we illustrate

Algorithm 7: Expands the coverage of the tree.

```

1 Procedure expandTree(tree,  $e_i$ )
2 Input: tree: current Counting Tree;
3        $e_i$ : the new event to be represented;
4 Output: tree: expanded Counting Tree, now
5           well-suited to represent event  $e_i$ ;
6 begin
7   if  $\exists a_j \in e_i : (a_j < \text{tree}.L_j \vee a_j > \text{tree}.U_j)$  then
8     //  $e_i$  is outside the coverage of tree
9     tree.initialLevel++; //used in 2nd phase only
10    tree.H++; // increments tree's height
11    for each coordinate  $a_j \in e_i$  do
12       $OL_j = \text{tree}.L_j$ ; // backups value
13      if  $a_j < \text{tree}.L_j$  then
14        tree.Lj = tree.Lj - tree.r0;
15      else
16        tree.Uj = tree.Uj + tree.r0;
17      end
18    end
19    tree.r0 = tree.r0 * 2;
20     $b = \text{tree}.root$ ; // backups root
21    tree.root = new(cell); //creates new root
22    tree.root.n =  $b.n$ ; tree.root.ptr =  $b$ ;
23    for each coordinate  $a_j \in e_i$  do
24      for each time period  $p$  in tree do
25        if  $a_j < OL_j$  then
26          tree.root.P[j][p] = 0;
27        else
28          tree.root.P[j][p] =  $b.n[p]$ ;
29        end
30    expandTree(tree,  $e_i$ ); // recursive call
31  end
32 end

```

in Figures 15b and 15c. Note that some of the new events in Figure 15b fall outside the attribute space covered by the tree, while in Figure 15c only a small portion of the space is still covered after the slide. To implement the sliding window model, we must therefore be able to expand the coverage of the tree, for cases similar to (b), while in cases resembling (c), we should contract the space for better representation. *How to efficiently expand/contract the coverage of one Counting Tree? Can it be performed by updating a few nodes only, without rebuilding the entire structure?*

To answer these questions, we propose here another improvement on the original *Counting Tree*. Our new method *Halite_{ds}* includes two novel algorithms to **efficiently** implement a self-adjusting coverage: algorithms *expandTree* and *contractTree*. Figure 16a illustrates our proposed strategy for space expansion. The full pseudo-code is in Algorithm 32. The expansion

occurs at each slide of the time window in which the new arriving events fall outside the current coverage of the tree. We propose to adaptively expand the coverage by: (i) creating a new root node for the tree, while setting the old root to be one of the new root’s “children”, thus representing a broader space, and; (ii) recursively performing this procedure until covering all new events to be inserted. In this way, we efficiently expand the coverage of an existing tree to represent **any** future event.

Algorithm 8: Contracts the coverage of the tree.

```

1 Procedure contractTree(tree)
2 Input: tree: current Counting Tree;
3 Output: tree: contracted tree, now well-suited to the
4           new set of events under consideration;
5 begin
6   if tree.root has one single “child” only  $\wedge$ 
7     tree.H > value of H originally chosen by the user
8   then
9     tree.initialLevel--; //used in 2nd phase only
10    tree.H--; // decrements tree’s height
11    b = tree.root.ptr; // root’s single “child”
12    r0 = r0/2;
13    for each dimension j do
14      if j-bit in b.loc is 1 then
15        tree.Lj = tree.Lj + tree.r0;
16      else
17        tree.Uj = tree.Uj - tree.r0;
18      end
19    tree.root = b // copies entire cell (all fields)
20    delete(b);
21    contractTree(tree); // recursive call
22  end
23 end

```

In the opposite way, Figure 16b illustrates our proposal for space contraction. The full pseudo-code is in Algorithm 23. The contraction occurs whenever one single “child” is left for the tree root, due to the disposal of old events in an slide of the window. We propose to adaptively contract the coverage by: (i) removing the tree root node, while setting its single “child” to be the new root, thus representing a smaller space, and; (ii) recursively performing this procedure until obtaining a new root with two or more “children”. In this way, we efficiently contract the coverage of an existing tree to better represent the new reality after the window slides.

A.7 Experiments

This section reports the experiments performed to evaluate our proposed *Halite_{ds}*. We compared it with five related works from the state-of-the-art: *HDDStream*, *EPCH*, *CFPC*, *LAC*

Figure 16 – Examples of expansion and contraction of the attribute space covered by a *Counting Tree*, and how to efficiently implement them.

and the base algorithm. We aimed to answer two main questions:

- **Q1** – How fast is the new method $Halite_{ds}$?
- **Q2** – How accurate is the new method $Halite_{ds}$?

A.7.1 System configuration

The experiments used a 2.83GHz processor from a machine with 8GB of RAM and Linux OS. Both $Halite_{ds}$ and its base algorithm used fixed values for their input parameters in all experiments: $H = 4$ and $\alpha = 1.0E - 10$. These are the default values suggested in the base algorithm's original proposal. Note that α is used only in the search for clusters, i.e., the second phase described in Section A.5.2, which is not modified in this work. The other algorithms were tuned as follows. *LAC* and *CFPC* received the number of clusters present in each dataset. As opposed to our proposal, they demand the user to estimate this number, even for unknown data. The extra parameters of the previous works were tuned as in their original authors' instructions. *CFPC* used its default values: $w = 5$, $\alpha = 0.05$ and $\beta = 0.15$. *HDDStream* was tuned with values 5, 10, 50 and 500 for μ , values 0.00001, 0.0001, 0.001, 0.01 and 0.1 for γ , values 1.8k, 10k, 20k, 40k, 60k and 80k for *InitPoints*, $\beta = 0.5$, $\varepsilon = 0.2$ and $\lambda = 0.5$. *LAC* was tested with integer values from 1 to 11, for the parameter $1/h$. However, its runtime differed considerably with distinct values of $1/h$. Thus, a timeout of 5 hours was specified for all *LAC* executions. *EPCH* was tuned with several integer values between 1 and 100 for the maximum number of clusters, integer values from 1 to 10 for the dimensionalities of its histograms and several real values varying from 0.1 to 1 for the outliers threshold.

Note that: (a) we ran each non-deterministic related work 5 times in each possible configuration and averaged the results. The averaged values were taken as the final result for each

configuration; (b) all results reported refer to the configurations that led to the best clustering accuracy, over all possible parameters tuning.

A.7.2 Experiments on synthetic data

To evaluate our proposed method on synthetic data, we generated one multidimensional data stream with 15 attributes and 1 million events². The events are organized in 100 time periods of $\sim 10k$ events each. To generate each time period we followed the same procedure used for data generation in the base algorithm’s original experiments, i.e., Algorithm 6 from (CORDEIRO *et al.*, 2013b). Specifically, each time period contains 5% of outlier events, and its remaining events form 10 clusters. The clusters exist only in axis-aligned subspaces of the original 15 dimensional space, i.e., they are **subspace clusters**, with subspaces formed by randomly chosen attributes. Each cluster has random size and its data distribution in each attribute is: (a) attribute of the subspace: normal distribution with random mean and random standard deviation; (b) other attributes: random data. Finally, given that the time periods must represent the evolution of one single dataset, each cluster of the first period has its counterpart in every subsequent period, with similar size, subspace and data distribution.

Figures 17, 18 and 19 report the results for the synthetic stream. One sliding window of size 200k events ($n_p = 20$ and $n_e = 10k$) was used. Figure 17 reports the clustering accuracy obtained by each competing algorithm as the stream evolves. We plot accuracy versus the initial period of the window as it slides. As we can see, both Haliteds (in black squares) and its base algorithm (blue circles) presented the highest accuracy obtaining values around 92%, while all other methods were considerably less accurate. Note that our *Halite_{ds}* provided accuracy results so close to the base algorithm, that its curve practically overwrites the base algorithm’s curve. To compute the accuracy we used the same strategy applied in the base algorithm’s original proposal (see Section 8.1 in (CORDEIRO *et al.*, 2013b)), in which precision and recall values are computed by comparing the clustering results provided by each algorithm with the ground truth that is known for the synthetic data.

Figure 18 reports the runtime required by each algorithm as the stream evolves. We plot runtime (log scale) versus the initial period of the window as it slides. Note that our *Halite_{ds}* was considerably faster than all others, being respectively 4, 39, 109 and 217 times faster than *CFPC*, *LAC*, *EPCH* and *HDDStream* in average.

Our *Halite_{ds}* was also 3.4 times faster than its base algorithm. Provided that both algorithms achieved similar accuracy, these results empirically demonstrate the advantages of using the knowledge obtained from clustering past data to easy clustering data in the present. To better understand the results, we highlight in Figure 19 the time required by both algorithms to: (a) build the *Counting Tree*, and; (b) to spot clusters. Note that we omitted the time required to load

² The stream follows Definition 1, so it can also be interpreted as a set of 15 unidimensional streams with simultaneous events.

Figure 17 – Accuracy in synthetic stream.

Figure 18 – Runtime in synthetic stream.

Figure 19 – Runtime in synthetic stream: building the tree versus spotting clusters.

Figure 20 – Runtime in real climatic stream.

the data from disk to avoid cluttering the illustration, as it is always the same for both algorithms. For the base algorithm, to spot clusters is in average 8.4 times faster than to build the tree. Thus, the latter is clearly the bottleneck. On the other hand, our *Halite_{ds}* is 13.6 faster than its base algorithm to build the tree, at the price of being solely 2.3 times slower to spot clusters. *Halite_{ds}* considerably shrinks the bottleneck by reusing the tree as the stream evolves, thus corroborating its expressive improvements.

A.7.3 Experiments on real climatic data

We also studied a real multidimensional stream containing almost one century (i.e., from 1917 to 2010) of frequent measurements for the climatic attributes: minimum and maximum temperatures, and precipitation of rain. The stream was collected in a real weather station at ESALQ-USP, Piracicaba, Brazil. The total number of events is 33,. Figure 20 reports runtime as the stream evolves. One 60th month sliding window was used with $n_p = 60$ and $n_e = \sim 30$. All methods were tested with the real data. However, *LAC* is not reported since it exceeded the timeout of 5 hours in all tested configurations. Again, the new method *Halite_{ds}* was the fastest one, being respectively 2.2, 3.3, 3.7 and 40 times faster than *EPCH*, the base algorithm, *CFPC* and *HDDStream* in average. No clustering ground truth exists for these data, so we cannot report

results on clustering accuracy.

A.8 Conclusion

This appendix presented the new algorithm *Halite_{ds}* – one fast, scalable and highly accurate **subspace clustering** algorithm for **multidimensional data streams**. It improves upon an existing technique that was originally designed to process static (not streams) datasets. Our main contributions are:

- Analysis of Data Streams: the new algorithm takes advantage of the knowledge obtained from clustering past data to easily cluster data in the present. This fact allows our *Halite_{ds}* to be considerably faster than its base algorithm, yet obtaining the same accuracy of results;
- Real Time Processing: as opposed to the state-of-the-art, *Halite_{ds}* is fast and scalable, making it feasible to analyze streams with many attributes and high frequency of events in **real time**.
- Experiments: we performed experiments using synthetic data and a real multidimensional stream with almost one century of climatic data. Our *Halite_{ds}* was up to 217 times faster than 5 works from the state-of-the-art and it always presented top-quality results.

ON THE SUPPORT OF A SIMILARITY-ENABLED RELATIONAL DATABASE MANAGEMENT SYSTEM IN CIVILIAN CRISIS SITUATION

B.1 Initial Considerations

This appendix presents one novel architecture to support decision-making during crisis situations that was developed in parallel with the main proposal of the MSc program, in collaboration with other researchers from the Database and Image Group – GBdI at ICMC/USP. The work generated a full paper(OLIVEIRA *et al.*, 2016) presented at the International Conference on Enterprise Information Systems - ICEIS 2016 (Qualis B2).

The candidate student actively participated in all phases of the project, such as: planning and developing the proposed architecture, preparing and performing the experiments and, writing the paper.

B.2 Problem and Motivation

Crisis situations, such as conflagrations, disasters in crowded events, and workplace accidents in industrial plants, may endanger human life and lead to financial losses. A fast response to this kind of situation is essential to reduce or prevent damage. In this context, software systems aimed at supporting experts in decision-making can be used to better understand and manage crises. A promising line of research is the use of social networks or crowdsourcing (KUDYBA, 2014) to gather information from the crisis site.

Several desirable tasks can be performed by software systems designed for aiding in

decision-making during crises. One of such tasks is to detect the evidences that best depict the crisis situation, so that rescue teams can be aware of it and prepare themselves properly. For instance, identifying fire or smoke on multimedia data, such as images, videos or textual reports, usually points to conflagration. Some relevant proposals in this direction comprehend fire and smoke detection based on image processing approaches (CELIK; OZKARAMANLI; DEMIREL, 2007) and techniques for fire detection designed over image descriptors that focus on detecting fire from social media images (BEDO *et al.*, 2015).

Another important task is to filter the information received from crowdsourcing solutions dedicated to collecting data from crises. When reporting incidents, users might end up sending too much similar information, such as pictures from similar angles of the same object. Such excess of similar data demands a longer time to be processed. Moreover, it turns the decision-making process more time-consuming. Therefore, removing duplicates is an essential task in this context.

The task of searching for similar data in historical databases can support decision-making as well. Take for instance a database that contains images and textual descriptions regarding past crisis situations. If the crowdsourcing system gets, for instance, images depicting fire, a query might be posed on the database to retrieve similar images and the corresponding textual descriptions. Then, based on these results, specialists would potentially infer the kind of material burning in the crisis, by analyzing the color tone of the smoke in the retrieved images and their textual descriptions.

For all those tasks, it is desirable that a commodity system provide functionalities over existing software infrastructure. Commodity systems that can play this role are the Relational Database Management Systems (RDBMS). They are largely available in the current computing technology and are able to bring new functionalities without the need of redesigning the existing software. Moreover, RDBMS provide efficient data storage and retrieval. However, they do not readily support similarity operations, which are needed to address the aforementioned tasks.

Several works in the literature aim at embedding similarity support in RDBMS. Nevertheless, the literature lacks a methodology for employing a similarity-enabled RDBMS in the context of crisis management. This work aims at filling that gap. Our hypothesis is that providing similarity support on an RDBMS helps the decision support in crisis situations.

B.3 Contributions

We contribute with a data-centric architecture for decision-making during crisis situations by means of a similarity-enabled RDBMS. Our proposal is evaluated using an image dataset of real crises from Flickr in performing three tasks:

- **Task 1.** Classification of incoming data regarding current events, detecting the most

relevant information to guide rescue teams in the crisis site;

- **Task 2.** Filtering of incoming data, enhancing the decision support of rescue command centers by removing near-duplicate data;
- **Task 3.** Similarity retrieval from past crisis situations, supporting analytical comprehension of the crisis context.

This work has been conducted to cater to demands of the project *RESCUER: Reliable and Smart Crowdsourcing Solution for Emergency and Crisis Management*¹, supported by the European Union’s Research and Innovation Funding Program FP7.

The results of our experimentation show that the proposed architecture is effective over crisis scenarios which rely on multimedia data. In addition to the high performance achieved, accurate results are obtained when using a proper combination of techniques.

The following sections are structured as follows. Section B.4 presents the related work and Section B.5 presents the main concepts for similarity support on RDBMS. Section B.6 describes the new Data-Centric Crisis Management architecture, on which the proposed methodology is based. Section B.7 presents our methodology, describes the experiments and discusses the results. Finally, the conclusions are presented in Section B.8.

B.4 Related Work

Existing research on crisis management highlights the importance of computer-assisted systems to support this task. The approaches may be categorized into different types according to their purpose.

One type refers to localization, whose purpose is to determine where victims are located during a disaster. There are works that accomplish this task by employing cell phone localization techniques, such as International Mobile Subscriber Identity (IMSI) catchers (REZNIK; HORAKOVA; SZTURC, 2015).

Another type regards logistics. Examples comprehend an integer programming technique for modeling multiple-resource emergency responses (ZHANG; LI; LIU, 2012) and a methodology for routing rescue teams to multiple communities (HUANG; SMILOWITZ; BALCIK, 2013).

A different line of work refers to decision-making based on social media (GIBSON *et al.*, 2014). Most of the work focus on textual data, specially from services like Twitter (GHAHREMANLOU; SHERCHAN; THOM, 2015).

¹ <<http://www.rescuer-project.org/>>

Although all the aforementioned approaches have been conceived to cater to different requirements, all of them share the characteristic of using Information Communication Technology (ICT) in response to crisis situations. The decision-making systems based on incoming data have one more characteristic: the participation of people somehow involved in the disaster.

Existing work have focused on the importance of crowdsourcing data for crisis management in the post-2015 world (HALDER, 2014). Therefore, to describe our methodology, we assume the existence of crowdsourcing as a subsystem dedicated to gathering input data. Additionally, we assume the existence of a command center, where analysts evaluate the input data in order to guide the efforts of a rescue team at the crisis site.

The work of Mehrotra (MEHROTRA *et al.*, 2004) is the closest approach with respect to our methodology. That work presents an interesting approach, but it focuses mostly on textual data and spatial-temporal information, rather than on more kinds of complex data, such as images. Furthermore, it lacks a methodology for employing content-based operations. We fill those gaps by providing a methodology to perform such operations over disaster-related data and provide useful information to rescue teams.

B.5 Background

B.5.1 Content-Based Retrieval

Complex data is a common term associated with objects such as images, audio, time series, geographical data and large texts. Such data do not present *order relation* and, therefore, are unable to be compared by relational operators ($<$, $\leq$, $\geq$, $>$). Equality operators ($=$, $\neq$) could be used, but they have little or no meaning when employed on such data. Nevertheless, complex data can be compared according to their content by using *similarity* concepts (BARIONI *et al.*, 2011).

The interaction with a content-based retrieval system starts as the user enters a query, providing a complex object as the query example. This complex object is submitted to a feature extractor, which extracts representative characteristics from it and generates a feature vector. The feature vector is sent to an evaluation function, which compares another feature vector stored in the database and returns a value representing the dissimilarity degree (also known as the distance) between both feature vectors. This comparison is repeated over the database, generating the results at the end of the process and sending them to the user.

Two of the most common queries used in content-based retrieval are the Range Query and the k Nearest Neighbor (k NN) Query (BARIONI *et al.*, 2011). Range Query is defined by the function $Rq(s_q, \xi)$, where s_q represents an object from data domain $\mathbb{S}$ and ξ is the radius used as distance constraint. The query returns all objects within a distance ξ from s_q . k NN Query is defined by the function $kNNq(s_q, k)$, where s_q represents an object from the data domain $\mathbb{S}$

and k is the number of elements to be returned. The query returns the k most similar objects to s_q . The k NN Queries are employed in the context of Instance-Based Learning (IBL) algorithms, such as the k NN Classifier, which is used in our proposal and thus discussed in Section B.5.2.

The feature extraction is usually required because the original representation of a given complex object is not prone to useful and efficient computation. Evaluation functions are able to compute the dissimilarity degree of a pair of feature vectors. These subjects are discussed in Sections B.5.3 and B.5.4.

The similarity retrieval process can be performed outside an RDBMS. However, enabling an RDBMS with similarity is a promising approach and there are several ways for doing so, as discussed in Section B.5.5.

B.5.2 *k*NN Classifier

The concept of Instance-Based Learning (IBL) (AHA; KIBLER; ALBERT, 1991) comprehends supervised learning algorithms that make predictions based solely on the instances previously stored in the database. In these algorithms, no model is built. The knowledge is represented by the data instances already stored and classified. Then, new instances are classified in relation to the existing stored instances, according to their similarity. One of the main IBL algorithms is the well-known k NN Classifier (FIX; Hodges Jr., 1951).

For a given unlabeled instance, the k NN Classifier retrieves from the database the k most similar instances. Following, it predicts the label based on the retrieved instances, according to some predefined criterion. A simple one is to assign the label of the prevailing class in the k nearest neighbors. Another one is to weigh the retrieved instances by distance, so the closest ones have a higher influence.

B.5.3 Feature Extractors

One of the main tasks for retrieving complex data by content is the feature extraction process, which maps a high-dimensional input data into a low-dimensional feature space, extracting useful information from raw data while reducing their content. Using proper feature extractors for a complex data domain leads to results closer to what the users expect (SIKORA, 2001).

Several feature extractors have been developed for different application domains. The main characteristics investigated in the context of images are color, texture and shape. There are several feature extractors for such characteristics, some of which are part of the MPEG-7 standard (MULTIMEDIA, 2002). In this work, we employ a color-based and a hash-based extractors.

Color-Based Extractors. Color-based extractors are commonly used as a basis for other extractors. For this reason, they are the most used visual descriptors in content-based image retrieval. The color-based feature extractors in the MPEG-7 standard are commonly employed

in the literature. One of them is the Color Structure Descriptor, which builds a color histogram based on the local features of the image.

Perceptual Hash. An extractor suitable for near-duplicate detection is the Perceptual Hash². It generates a “fingerprint” of a multimedia file derived from various features from its content. These “fingerprints” present the characteristic of being close to one another if the extracted features are similar.

B.5.4 Evaluation Functions

The dissimilarity between two objects is usually determined by a numerical value obtained from an evaluation function. Objects with smaller values are considered to be more alike.

The Minkowski Family comprehends evaluation functions known as L_p metrics that are widely used in content-based retrieval (WILSON; MARTINEZ, 1997). The L_1 metric corresponds to the Manhattan Distance, also noted as City-Block Distance. The L_2 metric is the well-known Euclidean Distance. Finally, there is the L_∞ metric, also noted as the Chebyshev Distance.

The Hamming Distance (HAMMING, 1950), which is another well-known evaluation function, counts the substitutions needed to transform one of the input data into the other. It can be employed in near-duplicate detection tasks, since combining it with the Perceptual Hash leads to accurate results.

B.5.5 Similarity Support on RDBMS

SimDB (SILVA *et al.*, 2010) is a similarity-enabled RDBMS, based on PostgreSQL. The similarity operations and keywords were included in its core. Equivalence rules were also included, which allows alternative query plans. However, similarity queries are only available for numerical data, and traditional queries over other data types are not supported.

SIREN (BARIONI *et al.*, 2011) is a middleware between a client application and the RDBMS. The client sends SQL commands extended with similarity keywords, which are checked by SIREN to identify similarity predicates. First, SIREN evaluates the similarity predicates, accessing an index of feature vectors, then uses the RDBMS for traditional predicates.

FMI-SiR (KASTER *et al.*, 2011) is a framework that operates over the RDBMS Oracle, employing user-defined functions to extract features and to index data. MedFMI-SiR is an extension of FMI-SiR for medical images in the Digital Imaging and Communications in Medicine (DICOM) format.

SimbA (BEDO; TRAINA; Traina Jr., 2014) is a framework that extends the middleware SIREN. SimbA supports the inclusion and combination of feature extractors, evaluation functions

² <<http://www.phash.org/>>

and indexes on demand. The queries are processed just like on SIREN.

B.6 Proposed Architecture

This section presents our architecture for crisis management, named as Data-Centric Crisis Management (DCCM). Section B.6.1 describes the scenario of a typical crisis situation managed by DCCM. Then, Section B.6.2 describes our architecture.

B.6.1 Crisis Management Scenario

Figure 21 shows the scenario of a crisis situation supported by DCCM. In a *Crisis Situation*, eyewitnesses can collect data regarding the event. For instance, they can take pictures, record videos and make textual reports, which are sent to the *Crowdsourcing System*. In Figure 21, the pictures taken by the eyewitnesses are redirected as an *Image Stream* to DCCM. Then, the command center can query DCCM for the *Decision-Making* process.

Figure 21 – Scenario of a typical crisis situation considering our architecture for crisis management.

Additionally, the crowdsourcing system could receive other data, such as metadata (e.g. time and GPS localization) or other data types (e.g. video and text).

B.6.2 Data-Centric Crisis Management

The Data-Centric Crisis Management (DCCM) architecture is represented in Figure 22. The whole mechanism has three processes, each of them depicted in the figure by arrows marked with the letters A, B and C, which represent the tasks introduced in Section B.1.

In a crisis situation, we consider the existence of a crowdsourcing system that receives disaster-related complex objects (A1) and submits them to DCCM.

Each object of the data stream is placed in a *Buffer* and analyzed by the *Filtering Engine*. First, the engine checks whether the object is a near duplicate of some other object currently

Figure 22 – The DCCM architecture, consisting of the tasks: classification (A), filtering (B) and historical retrieval (C).

within the *Buffer*. For the near-duplicate checking, the *Filtering Engine* uses the *Similarity Engine* (A2) to extract a feature vector from the object and compare it to the feature vectors of the other objects within the *Buffer*. The object is marked as a near duplicate when its distance from at least another object is at most ξ , which is a threshold defined by specialists according to the application domain.

If the object is not a near duplicate, then it is submitted to the *Classification Engine* (A3). The classification process uses *Historical Records* in a database to train classifier methods (A4). Based on such training, the *Classification Engine* labels the object regarding the event it represents. For instance, it can be labeled as “fire” or “smoke”. Finally, the *Classification Engine* notifies the specialists with the now-classified object for the *Decision-Making* process (A5).

If the object is a near duplicate, then it is not submitted to the *Classification Engine*. Instead, it stays in the *Buffer* to be compared to others that come later. Moreover, it is associated with the event of the object of which it is a near duplicate. In Figure 22, the objects from the same event have the same color. The white objects marked with “?” have not been analyzed yet.

The *Buffer* may be determined either by a physical size, such as the number of elements it holds, or by a time window. In Figure 22, it is delimited by a time window of length k , beginning at time t and ending at time $t - k$. The *Buffer* is flushed at every k -th time instant. Before flushing, the *Representative Selector* selects the object of each group that best represents its event (B1) according to a predefined criterion.

If a near-duplicate object is selected as the representative, then it gets the label of the classified object of its group. The already-classified object, in turn, is marked as near duplicate. On the other hand, if the selected representative is already the classified object of its group, then no changes are made. Lastly, the classified objects are stored in the database and the near duplicates are discarded, flushing the *Buffer* (B2).

There is another use case for DCCM, which refers to the historical analyses. The *Decision-Making* team may want to provide complex data samples to retrieve similar events from the past. For this purpose, DCCM provides the *Historical Retrieval Engine* (C1). First, the engine extracts the features from each provided sample (C2). Then, it compares the extracted features against the *Historical Records* and provides its findings to the *Decision-Making* team (C3).

B.7 Case Study

In this section, we present the case study for evaluating the DCCM architecture over the three tasks discussed earlier. The experiments were carried out over a real crisis dataset known as Flickr-Fire (BEDO *et al.*, 2015) containing 2,000 images extracted from Flickr, 1,000 labeled as “fire” and 1,000 as “not fire”.

B.7.1 Implementation of DCCM

To implement DCCM, we extended the open-source RDBMS PostgreSQL. Our implementation, named as **Kiara**, supports an SQL extension for building similarity queries over complex data (BARIONI *et al.*, 2011). Also through the SQL extension, Kiara allows managing feature extractors and evaluation functions, which are dynamically (no recompilation) inserted and updated by user-defined functions written in C++. Kiara makes use of metatables to keep track of feature extractors and evaluation functions associated with the attributes of complex data that a user instantiates.

To support the SQL extension, we built a parser that works like a proxy in the core of Kiara. It receives a query and rewrites only the similarity predicates, expressing them through standard SQL operators. Then, it sends the rewritten queries to the core of Kiara.

After inserting a new extractor, the features are automatically extracted from the complex data (e.g. image, video or text) and then stored in user-defined attributes dedicated to representing such data. Similarity queries can be included through PL/pgSQL functions and new indexes can be included through the interface known as Generalized Search Tree (GiST), already present in PostgreSQL. Moreover, Kiara allows exploring alternative query plans involving traditional and similarity predicates.

B.7.2 Classification of Incoming Data

B.7.2.1 Methodology

Classifying disaster-related incoming data is helpful because of two reasons. One of them is to identify the characteristic that best depicts the crisis situation. The other is to store data properly labeled, which improves further queries on a historical database. To do so, the DCCM architecture employs the *k*NN Classifier.

The parameter k can be selected arbitrarily. However, too small values can be noise-sensitive, whereas too large values allow including more instances from other classes, leading to misclassified instances.

B.7.2.2 Experimentation and Results

In this task, we classified the elements of the Flickr-Fire dataset. For a robust experimentation, we used the procedure *10-fold cross-validation* for a k NN classifier using $k = 10$. We used the Manhattan Distance and the extractor Color Structure Descriptor because existing work showed that they allow accurate results for fire detection (BEDO *et al.*, 2015).

After 10 rounds of evaluation, we took the average accuracy and the average F1 score. The result was the same for both measures, which was 0.86. Considering a real event, this capability would be able to automatically group data according to their content, indicating the main characteristics of the crisis and thus saving the command center crucial time for fast response.

B.7.3 Filtering of Incoming Data

B.7.3.1 Methodology

In the task of filtering, we are interested in preventing duplicate information from being classified and subsequently sent to the command center.

To determine whether the incoming data is a near duplicate of existing data, they must be compared by their content. For this purpose, we must employ similarity queries. In this case, though, we are restricted to Range Queries. If the new object is a near duplicate of an object in the buffer, then the distance from each other is at most ξ , which is supposed to be a small threshold (range), since we want to detect pairs of objects that, in essence, represent the same information. Range Queries allow restricting results based on their similarity, differently from k NN Queries, which do it by the number of objects retrieved.

Hence, the DCCM architecture prevents near duplicates by using Range Queries. Each object that arrives in the buffer is submitted to a default feature extractor. Then, a Range Query is performed by using the extracted features as the s_q object. The range value ξ must be predefined as well, according to the application domain. If at least one object from the buffer is retrieved by the Range Query, then the s_q object is marked as a near duplicate.

B.7.3.2 Experimentation and Results

For this experiment, we employed the Hamming Distance with the Perceptual Hash extractor and assumed a buffer size of 80. We filled the buffer with 80 images from Flickr, of which 37 depict “small fire” events and 43 depict “big fire” events. Each of the 80 images was used as the s_q to a Range Query with $\xi = 10$.

The ξ parameter was set to retrieve around half of the images (40 images approximately), in order to be able to return an entire class (“big fire” or “small fire”) of images. This allows evaluating the precision of the queries with the Precision-Recall method.

Figure 23 shows the Precision-Recall curve for this set of queries. The curve falls off only after 80% of retrieval. This late fall-off is characteristic of highly effective retrieval methods. In this result, one can notice a precision above 90% up to 50% of recall.

Figure 23 – Precision-Recall in the process of filtering incoming data in the buffer.

The results show that DCCM is expected to filter out 90% of near-duplicate data. This is a strong indication that such capability can significantly improve both efficiency and efficacy of a command center. Filtering is the most desirable functionality considered in this work. This is because crowdsourcing is highly prone to produce redundant data. Right after a crisis is installed, if filtering is not possible, the flow of information streamed by eyewitnesses may be too high for the command center to make good use of them. However, a similarity-enabled RDBMS in DCCM is able to handle such situation with basic similarity queries.

B.7.4 Retrieval of Historical Data

B.7.4.1 Methodology

In the context of crisis management, the experts from the command center might be willing to analyze data from past events that are similar to the current ones. Such data may lead to decisions about how to proceed with the current crisis. In these situations, similarity queries play an important role.

Considering the DCCM architecture, this task can be performed whenever the *Decision-Making* experts want information similar to the current data. For every notified data at point A5 of Figure 22, they might provide it as the s_q element to the *Historical Retrieval Engine* in order to get similar information.

B.7.4.2 Experimentation and Results

For this experiment, we combined the Color Structure Descriptor and the Manhattan Distance. We performed Range and k NN Queries using each of the 2,000 elements from the Flickr-Fire dataset as the s_q element. We set the k parameter to 1,000 and the ξ parameter to 7.2, retrieving an entire class.

Figure 24 – Precision-Recall for retrieving historical data.

We generated the Precision-Recall curve depicted in Figure 24. From these results, one can observe the high precision around 0.8 when fetching 10% of relevant data, roughly 100 images, and around 0.9 when fetching 5%, nearly 50 images — a more realistic scenario. These results point to an effective retrieval of images based on their class.

From the point of view of a command center user, there would be an ample knowledge bank from which initial considerations could be drawn from the current crisis. This initial knowledge has the potential of saving time of rescue actions by preventing past mistakes and fostering successful decisions.

B.7.5 Overall Performance

Concerning a computer system, it is important to receive the correct response in a timely manner. Therefore, we analyzed the overall performance of DCCM. For this purpose, we carried out one experiment regarding scalability and three regarding the tasks.

Overall Scalability. A solution based on DCCM spends most of its time receiving, storing and indexing data for the sake of similarity retrieval. Therefore, such processing must be efficient. We carried out an experiment to evaluate the time spent extracting features and inserting them into the database. The average time of five rounds is presented in Figure 25.

From the results presented in the figure, one can calculate that the solution is able to process up to 3 images per second — sufficient for many real case scenarios. These numbers refer to a machine with a hard disk of 5,400RPM; such results could be improved by using SSD disks or RAID subsystems.

Figure 25 – Time to extract features from one image and insert them into the database.

From Figure 25, one can also observe that the time spent inserting images is mostly taken by the feature extraction, while the time for inserting the features remains constant. The extraction time varies according to the image resolutions, which range from 300x214 to 3,240x4,290 pixels in the Flickr-Fire dataset.

Table 4 – Overall performance of DCCM over Flickr-Fire.

Task	Average time (sec)
Classification	0.851
Filtering	0.057
Retrieval — Range Query	1.147
Retrieval — k NN Query	0.849

Table 4 presents the performance of DCCM to perform the three tasks of our methodology. We ran the classification and filtering tasks 10 times, whereas the retrieval tasks were performed 2,000 times, once for each element in the dataset. For the classification task, we used the distanced-weighted k NN classifier, with $k = 10$ and performing 10-fold cross-validation. For the filtering task, we used the 80 aforementioned near-duplicate images and the range value ξ was set to 10. Finally, in the retrieval tasks, ξ was set to 2.8 for the Range Queries, retrieving around 50 tuples, and k was set to 50 for the k NN Queries. The results, which represent the average time to perform each task once, indicate that our proposal is feasible in a real-time crisis management application.

B.8 Conclusions

Fast and precise responses are essential characteristics of computational solutions. In this appendix, we presented the architecture of a solution that can achieve these characteristics in crisis management tasks. In the course of our work, we described the use of a similarity-enabled

RDBMS in tasks that could assist a command center in guiding rescue missions. To make it possible, we implemented similarity-based operations within one popular, open-source RDBMS.

The core of our work is related to an innovation project led by the European Union; accordingly, we applied similarity retrieval concepts in an innovative manner, putting together relational and retrieval technologies. To demonstrate our claims, we carried out experiments to evaluate both the efficacy and the efficiency of our proposal. More specifically, we introduced the following functionalities:

- **Classification of Incoming Data.** We proposed to employ k NN classification to classify incoming data, aiming at identifying and characterizing crisis situations faster;
- **Filtering of Incoming Data.** We proposed to employ Range Queries to filter out redundant information, aiming at reducing the data load over the system and over a command center;
- **Retrieval of Historical Data.** We proposed to employ Range and k NN Queries to retrieve data from past crises that are similar to the current one.

The results we obtained for each of these tasks allowed us to claim that a similarity-enabled RDBMS is able to assist in the decision support of command centers when a crisis situation strikes. We conclude by stating that our work demonstrated the use of cutting-edge methods and technologies in a critical scenario, paving the way for similar systems to flourish based on the experiences that we reported.