Data mining in large sets of complex data

Robson Leonardo Ferreira Cordeiro

Data mining in large sets of complex data[1]

Robson Leonardo Ferreira Cordeiro

Advisor: **Prof. Dr. Caetano Traina Jr.**
Co-advisor: **Prof. Dr. Christos Faloutsos**

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

USP – São Carlos
October 2011

[1] This work has been supported by FAPESP (Process Number 2007/01639-4), CNPq and CAPES.

Ficha Catalográfica elaborada pela Seção de Tratamento da Informação da Biblioteca Prof. Achille Bassi- Instituto de Ciências Matemáticas e de Computação – ICMC/USP.

Cordeiro, Robson Leonardo Ferreira
C794d Data mining in large sets of complex data / Robson Leonardo Ferreira Cordeiro. São Carlos, 2011.
135 p.

Tese (Doutorado – Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional). - - Instituto de Ciências Matemática e de Computação, Universidade de São Paulo, 2011.
Orientador: Caetano Traina Jr.
Co-orientador: Christos Faloutsos
1. Correlation Clustering. 2. Moderate-to-high dimensionality data. 3. Terabyte-scale data mining. 4. MapReduce. 5. Labeling and Summarization. I. Traina Jr., Caetano, orient. II. Faloutsos, Christos, co-orient. III. Título.

Abstract

Due to the increasing amount and complexity of the data stored in the enterprises' databases, the task of knowledge discovery is nowadays vital to support strategic decisions. However, the mining techniques used in the process usually have high computational costs that come from the need to explore several alternative solutions, in different combinations, to obtain the desired knowledge. The most common mining tasks include data classification, labeling and clustering, outlier detection and missing data prediction. Traditionally, the data are represented by numerical or categorical attributes in a table that describes one element in each tuple. Although the same tasks applied to traditional data are also necessary for more complex data, such as images, graphs, audio and long texts, the complexity and the computational costs associated to handling large amounts of these complex data increase considerably, making most of the existing techniques impractical. Therefore, especial data mining techniques for this kind of data need to be developed. This Ph.D. work focuses on the development of new data mining techniques for large sets of complex data, especially for the task of clustering, tightly associated to other data mining tasks that are performed together. Specifically, this Doctoral dissertation presents three novel, fast and scalable data mining algorithms well-suited to analyze large sets of complex data: the method *Halite* for correlation clustering; the method *BoW* for clustering Terabyte-scale datasets; and the method *QMAS* for labeling and summarization. Our algorithms were evaluated on real, very large datasets with up to *billions* of complex elements, and they always presented highly accurate results, being at least one order of magnitude faster than the fastest related works in almost all cases. The real data used come from the following applications: automatic breast cancer diagnosis, satellite imagery analysis, and graph mining on a large web graph crawled by Yahoo! and also on the graph with all users and their connections from the Twitter social network. Such results indicate that our algorithms allow the development of *real time applications* that, potentially, could not be developed without this Ph.D. work, like a software to aid on the fly the diagnosis process in a worldwide Healthcare Information System, or a system to look for deforestation within the Amazon Rainforest in real time.

Resumo

O crescimento em quantidade e complexidade dos dados armazenados nas organizações torna a extração de conhecimento utilizando técnicas de mineração uma tarefa ao mesmo tempo fundamental para aproveitar bem esses dados na tomada de decisões estratégicas e de alto custo computacional. O custo vem da necessidade de se explorar uma grande quantidade de casos de estudo, em diferentes combinações, para se obter o conhecimento desejado. Tradicionalmente, os dados a explorar são representados como atributos numéricos ou categóricos em uma tabela, que descreve em cada tupla um caso de teste do conjunto sob análise. Embora as mesmas tarefas desenvolvidas para dados tradicionais sejam também necessárias para dados mais complexos, como imagens, grafos, áudio e textos longos, a complexidade das análises e o custo computacional envolvidos aumentam significativamente, inviabilizando a maioria das técnicas de análise atuais quando aplicadas a grandes quantidades desses dados complexos. Assim, técnicas de mineração especiais devem ser desenvolvidas. Este Trabalho de Doutorado visa a criação de novas técnicas de mineração para grandes bases de dados complexos. Especificamente, foram desenvolvidas duas novas técnicas de agrupamento e uma nova técnica de rotulação e sumarização que são rápidas, escaláveis e bem adequadas à análise de grandes bases de dados complexos. As técnicas propostas foram avaliadas para a análise de bases de dados reais, em escala de Terabytes de dados, contendo até *bilhões* de objetos complexos, e elas sempre apresentaram resultados de alta qualidade, sendo em quase todos os casos pelo menos uma ordem de magnitude mais rápidas do que os trabalhos relacionados mais eficientes. Os dados reais utilizados vêm das seguintes aplicações: diagnóstico automático de câncer de mama, análise de imagens de satélites, e mineração de grafos aplicada a um grande grafo da web coletado pelo Yahoo! e também a um grafo com todos os usuários da rede social Twitter e suas conexões. Tais resultados indicam que nossos algoritmos permitem a criação de *aplicações em tempo real* que, potencialmente, não poderiam ser desenvolvidas sem a existência deste Trabalho de Doutorado, como por exemplo, um sistema em escala global para o auxílio ao diagnóstico médico em tempo real, ou um sistema para a busca por áreas de desmatamento na Floresta Amazônica em tempo real.

Título: *Mineração de Dados em Grandes Conjuntos de Dados Complexos*
Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. VERSÃO REVISADA.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

Halite	*Method Halite for Correlation Clustering*
BoW	*The **B**est **o**f both **W**orlds Method*
QMAS	*Method for **Q**uerying, **M**ining **A**nd **S**ummarizing Multi-dimensional Databases*
KDD	*Knowledge Discovery in Databases*
DB	*Database*
HDFS	*Hadoop Distributed File System*
RWR	*Random Walk with Restart*
MDL	*Minimum Description Length*
SDSS	*Sloan Digital Sky Survey*
API	*Application Programming Interface*
I/O	*Input / Output*
MBR	*Minimum Bounding Rectangle*
MAM	*Metric Access Methods*
CPU	*Central Processing Unit*
PB	*Petabyte(s)*
TB	*Terabyte(s)*
GB	*Gigabyte(s)*
MB	*Megabyte(s)*
RAM	*Random-Access Memory*
GHz	*Gigahertz*
GBdI	*Databases and Images Group*
USP	*University of São Paulo*
CMU	*Carnegie Mellon University*

List of Symbols

${}^{d}\mathbb{S}$	A d-dimensional space.
${}^{\square}_{\square}S_{\square}$	A set of d-dimensional points. ${}^{\square}_{\square}S_{\square} \subset {}^{d}\mathbb{S}$
	${}^{d}S$ - A full d-dimensional dataset.
	${}^{\delta}_{\gamma}S_k$ - Points of cluster ${}^{\delta}_{\gamma}C_k$. ${}^{\delta}_{\gamma}S_k \subseteq {}^{d}S$
${}_{\square}E_{\square}$	A set of axes.
	E - Full set of axes for ${}^{d}S$.
	$E = \{e_1, e_2 \ldots e_d\},\ \lvert E \rvert = d$
	${}_{\gamma}E_k$ - Axes relevant to a cluster ${}^{\delta}_{\gamma}C_k$.
	${}_{\gamma}E_k \subseteq E,\ \lvert {}_{\gamma}E_k \rvert = \delta$
d	Dimensionality of dataset ${}^{d}S$.
η	Number of points in dataset ${}^{d}S$. $\eta = \left\lvert {}^{d}S \right\rvert$
s_i	A point of dataset ${}^{d}S$. $s_i \in {}^{d}S$
s_{ij}	Value in axis e_j of point s_i. $s_{ij} \in [0, 1)$
${}^{\delta}_{\gamma}C_k$	A correlation cluster. ${}^{\delta}_{\gamma}C_k = \left\langle {}_{\gamma}E_k, {}^{\delta}_{\gamma}S_k \right\rangle$
δ	Dimensionality of ${}^{\delta}_{\gamma}C_k$.
${}_{\square}k$	Number of clusters in dataset ${}^{d}S$.
	${}_{\gamma}k$ - Number of correlation clusters.
	k - Number of clusters regardless of their types.
T	A Counting-tree.
H	Number of resolutions in T
h	Each level of T.
ξ_h	Side size of cells at level h in T.
a_h	A cell at level h in T.
α	Significance level for the statistical test.
r	Number of reducers for parallel run.
m	Number of mappers for parallel run.

F_s	Database file size in bytes.
D_s	Disk speed, i.e. disk transfer rate in bytes per second.
N_s	Network speed, i.e. network transfer rate in bytes per second.
D_r	Dispersion ratio.
R_r	Reduction ratio.
S_r	Sampling ratio.
$start\_up\_cost(t)$	Start-up cost for t `MapReduce` tasks.
$plug\_in\_cost(s)$	Serial clustering cost regarding the data size s.
I	An input collection of complex objects.
I_i	One object from I. $I_i \in I$
N_I	The number of objects in I. $N_I = \|I\|$
L	A collection of known labels.
L_l	One label from L. $L_l \in L$
N_L	The number of labels in L. $N_L = \|L\|$
N_R	The desired number of representatives.
N_O	The desired number of top outliers.
G	The *Knowledge Graph*. $G = (V, X)$
V	The set of vertexes in G.
X	The set of edges in G.
$V(I_i)$	Vertex that represents object I_i in G.
$V(L_l)$	Vertex that represents label L_l in G.
c	The restart probability for the random walk.

Chapter

1

Introduction

This chapter presents an overview of this Doctoral dissertation. It contains brief descriptions of the facts that motivated the work, the problem definition, our main objectives and the central contributions of this Ph.D. work. The following sections describe each one of these topics.

1.1 Motivation

The information generated or collected in digital formats for various application areas is growing not only in the number of objects and attributes, but also in the complexity of the attributes that describe each object [Fayyad, 2007b, Fayyad et al., 1996, Kanth et al., 1998, Korn et al., 2001, Kriegel et al., 2009, Pagel et al., 2000, Sousa, 2006]. This scenario has prompted the development of techniques and tools aimed at, intelligently and automatically, assisting humans to analyze, to understand and to extract knowledge from raw data [Fayyad, 2007b, Fayyad and Uthurusamy, 1996, Sousa, 2006], molding the research area of *Knowledge Discovery in Databases – KDD*.

The increasing amount of data makes the KDD tasks especially interesting, since they allow the data to be considered as useful resources in the decision-making processes of the organizations that own them, instead of being left unused in disks of computers, stored to never be accessed, such as real 'tombs of data' [Fayyad, 2003]. On the other hand, the increasing complexity of the data creates several challenges to the researchers, provided that most of the existing techniques are not appropriate to analyze complex data, such as images, audio, graphs and long texts. Common knowledge discovery tasks are clustering, classification and labeling, identifying measurement errors and outliers, inferring association rules and missing data, and dimensionality reduction.

1.2 Problem Definition and Main Objectives

The knowledge discovery from data is a complex process that involves high computational costs. The complexity stems from a variety of tasks that can be performed to analyze the data and from the existence of several alternative ways to perform each task. For example, the properties of the various attributes used to describe each data object, such as the fact that they are categorical or continuous, the cardinality of the domains, and the correlations that may exist between different attributes, etc., they all make some techniques more suitable or prevent the use of others. Thus, the analyst must face a wide range of options, leading to a high complexity in the task of choosing appropriate mining strategies to be used for each case.

The high computational cost comes from the need to explore several data elements in different combinations to obtain the desired knowledge. Traditionally, the data to be analyzed are represented as numerical or categorical attributes in a table where each tuple describes an element in the set. The performance of the algorithms that implement the various tasks of data analysis commonly depend on the number of elements in the set, on the number of attributes in the table, and on the different ways in which both tuples and attributes interact with their peers. Most algorithms exhibit super-linear complexity regarding these factors, and thus, the computational cost increases fast with increasing amounts of data.

The discovery of knowledge from complex data, such as images, audio, graphs and long texts, usually includes a preprocessing step, when relevant features are extracted from each object. The features extracted must properly describe and identify each object, since they are actually used in search and comparison operations, instead of the complex object itself. Many features are commonly used to represent each object. The resulting collection of features is named the feature vector.

This Ph.D. work aims at the development of knowledge discovery techniques well-suited to analyze large collections of complex objects described *exclusively* by their feature vectors, especially for the task of clustering, tightly associated to other data mining tasks that are performed together. Thus, we have explored the following thesis:

Thesis: *Although the same tasks commonly performed for traditional data are generally also necessary for the analysis of feature vectors from complex objects, the complexity of the analysis and the computational cost associated increase significantly, preventing the use of most of the traditional techniques. Thus, knowledge discovery techniques well-suited to analyze large, complex datasets described exclusively by feature vectors need to be created. As specific properties of feature vectors can be taken into account, it is possible to reduce the complexity and the computational cost involved, which, in the case of high dimensional vectors, are naturally higher than those involved in traditional data.*

Therefore, this work is focused on developing techniques well-suited to analyze large collections of complex objects represented *exclusively* by their feature vectors, automatically extracted by preprocessing algorithms. Nevertheless, the proposed techniques can be applied to any kind of complex data, from which sets of numerical attributes of equal dimensionalities can be extracted. Thus, the analysis of multi-dimensional datasets is the scope of our work. The definitions related to this kind of data, which are used throughout this Doctoral dissertation, are presented as follows.

Definition 1 *A* ***multi-dimensional dataset*** $^{d}S = \{s_1, s_2, \ldots s_\eta\}$ *is a set of* η *points in a d-dimensional space* $^{d}\mathbb{S}$, $^{d}S \subset {}^{d}\mathbb{S}$, *over the set of axes* $E = \{e_1, e_2, \ldots e_d\}$, *where* d *is the* ***dimensionality*** *of the dataset, and* η *is its* ***cardinality****.*

Definition 2 *A* ***dimension*** *(also called a feature or an attribute)* $e_j \in E$ *is an axis of the space where the dataset is embedded. Every axis related to a dataset must be orthogonal to the other axes.*

Definition 3 *A* ***point*** $s_i \in {}^{d}S$ *is a vector* $s_i = (s_{i1}, s_{i2}, \ldots s_{id})$ *that represents a data element in the space* $^{d}\mathbb{S}$. *Each value* $s_{ij} \in s_i$ *is a number in* $\mathbb{R}$. *Thus, the entire dataset is embedded in the d-dimensional hyper-cube* $\mathbb{R}^d$.

1.3 Main Contributions of this Ph.D. Work

With regard to the task of clustering large sets of complex data, an analysis of the literature (see the upcoming Chapter 3) leads us to come to one main conclusion. In spite of the several qualities found in the existing works, to the best of our knowledge, there is no method published in the literature, and well-suited to look for clusters in sets of complex objects, that has *any* of the following desirable properties: (i) **linear or quasi-linear complexity** – to scale linearly or quasi-linearly in terms of memory requirement and execution time with regard to increasing numbers of points and axes, and; (ii) **Terabyte-scale data analysis** – to be able to handle datasets of Terabyte-scale in feasible time. On the other hand, examples of applications with Terabytes of high-dimensionality data abound: weather monitoring systems and climate change models, where we want to record wind speed, temperature, rain, humidity, pollutants, etc; social networks like Facebook TM, with millions of nodes, and several attributes per node (gender, age, number of friends, etc); astrophysics data, such as the SDSS (Sloan Digital Sky Survey), with billions of galaxies and attributes like red-shift, diameter, spectrum, etc. Therefore, the development of novel algorithms aimed at overcoming these two aforementioned limitations is nowadays extremely desirable.

This Doctoral dissertation focuses on overcoming both limitations. Specifically, it presents three novel, fast and scalable data mining algorithms well-suited to analyze large sets of complex data:

1. **The Method *Halite* for Correlation Clustering:** the algorithm *Halite* is a fast and scalable density-based clustering algorithm for multi-dimensional data able to analyze large collections of complex data elements. It creates a multi-dimensional grid all over the data space and counts the number of points lying at each hyper-cubic cell provided by the grid. A hyper-quad-tree-like structure, called the Counting-tree, is used to store the counts. The tree is thereafter submitted to a filtering process able to identify regions that are, in a statistical sense, denser than its neighboring regions regarding at least one dimension, which leads to the final clustering result. The algorithm is fast and it has linear or quasi-linear time and space complexity regarding both the data size and the dimensionality. Therefore, *Halite* tackles the problem of **linear or quasi-linear complexity**.

2. **The Method *BoW* for Clustering Terabyte-scale Datasets:** the method *BoW* focuses on the problem of finding clusters in Terabytes of moderate-to-high dimensionality data, such as features extracted from billions of complex data elements. In these cases, a serial processing strategy is usually impractical. Just to read a single Terabyte of data (at 5GB/min on a single modern eSATA disk) one takes more than 3 hours. *BoW* explores parallelism and can treat as plug-in almost any of the serial clustering methods, including our own algorithm *Halite*. The major research challenges addressed are (a) how to minimize the I/O cost, taking care of the *already existing* data partition (e.g., on disks), and (b) how to minimize the network cost among processing nodes. Either of them may become the bottleneck. Our method automatically spots the bottleneck and chooses a good strategy, one of them uses a novel *sampling-and-ignore* idea to reduce the network traffic. Specifically, *BoW* combines (a) potentially any serial algorithm used as a plug-in and (b) makes the plug-in run efficiently in parallel, by adaptively balancing the cost for disk accesses and network accesses, which allows *BoW* to achieve a very good tradeoff between these two possible bottlenecks. Therefore, *BoW* tackles the problem of **Terabyte-scale data analysis**.

3. **The Method *QMAS* for Labeling and Summarization:** the algorithm *QMAS* uses the background knowledge provided by the clustering algorithms designed in this Ph.D. work to focus on two distinct data mining tasks – the tasks of labeling and summarizing large sets of complex data. Specifically, *QMAS* is a fast and scalable solution to two problems (a) low-labor labeling – given a large collection of complex objects, very few of which are labeled with keywords, find the most suitable labels for the remaining ones, and (b) mining and attention routing – in the same setting, find clusters, the top-N_O outlier objects, and the top-N_R representative objects. The algorithm is fast and it scales linearly with the data size, besides working even with tiny initial label sets.

Our algorithms were evaluated on *real, very large datasets* with up to *billions* of complex elements, and they always presented highly accurate results, being at least one order of magnitude faster than the fastest related works in almost all cases. The real life data used come from the following applications: *automatic breast cancer diagnosis*, *satellite imagery analysis*, and *graph mining* on a large web graph crawled by Yahoo![1] and also on the graph with all users and their connections from the Twitter[2] social network. In extreme cases, the work presented in this Doctoral dissertation allowed us to spot in only two seconds the clusters present in a large set of satellite images, while the related works took two days to perform the same task, achieving similar accuracy. Such results indicate that our algorithms allow the development of *real time applications* that, potentially, could not be developed without this Ph.D. work, like a software to aid on the fly the diagnosis process in a worldwide Healthcare Information System, or a system to look for deforestation within the Amazon Rainforest in real time.

1.4 Conclusions

This chapter presented an overview of this Doctoral dissertation with brief descriptions of the facts that motivated the work, the problem definition, our main objectives and the central contributions of this Ph.D. work. The remaining chapters are structured as follows. In Chapter 2, an analysis of the literature is presented, including a description of the main concepts used as a basis for the work. Some of the relevant works found in literature for the task of clustering multi-dimensional data with more than five or so dimensions are described in Chapter 3. Chapters 4, 5 and 6 contain the central part of this Doctoral dissertation. They present the knowledge discovery techniques designed during this Ph.D. work, as well as the experiments performed. Finally, the conclusions and ideas for future work are given in Chapter 7.

[1] www.yahoo.com
[2] twitter.com

Chapter

2

Related Work and Concepts

This chapter presents the main background knowledge related to the doctoral project. The first two sections describe the areas of processing complex data and knowledge discovery in traditional databases. The task of clustering complex databases is discussed in Section 2.3, while the task of labeling such kind of data is described in Section 2.4. Section 2.5 introduces the `MapReduce` framework, a promising tool for large scale data analysis, which has been proven to offer a valuable support to the execution of data mining algorithms in a parallel processing environment. The last section concludes the chapter.

2.1 Processing Complex Data

Database systems work efficiently with traditional numeric or textual data, but they usually do not provide complete support for complex data, such as images, videos, audio, graphs, long texts, fingerprints, geo-referenced data, among others. However, efficient methods for storing and retrieving complex data are increasingly needed [Mehrotra et al., 1997]. Therefore, many researchers have been working to make database systems more suited to complex data processing and analysis.

The most common strategy is the manipulation of complex data based on features extracted automatically or semi-automatically from the data. This involves the application of techniques that aim at obtaining a set of features (the feature vector) to describe the complex element. Each feature is typically a value or an array of numerical values. The vector resulting from this process should properly describe the complex data, because the mining algorithms rely only on the extracted features to perform their tasks. It is common to find vectors containing hundreds or even thousands of features.

For example, the extraction of features from images is usually based on the analysis of colors, textures, objects' shapes and their relationship. Due to its simplicity and low computational cost, the most used color descriptor is the histogram, that counts the numbers of pixels of each color in an image [Long et al., 2002]. The color coherence vector [Pass et al., 1996], the color correlogram [Huang et al., 1997], the metric histogram [Traina et al., 2003] and the cells histogram [Stehling et al., 2003] are other well-known color descriptors. Texture corresponds to the statistical distribution of how the color varies in the neighborhood of each pixel of the image. Texture analysis is not a trivial task and it usually leads to higher computational costs than the color analysis does. Statistical methods [Duda et al., 2001, Rangayyan, 2005] analyze properties, such as granularity, contrast and periodicity to differentiate textures, while syntactic methods [Duda et al., 2001] perform this task by identifying elements in the image and analyzing their spatial arrangements. Co-occurrence matrices [Haralick et al., 1973], Gabor [Daugman, 1985, Rangayyan, 2005] and wavelet transforms [Chan and Shen, 2005] are examples of such methods. The descriptors of shapes commonly have the highest computational costs compared to the other descriptors, and therefore, they are used mainly in specific applications [Pentland et al., 1994]. There are two main techniques to detect shapes: the geometric methods of edge detection [Blicher, 1984, Rangayyan, 2005], which analyze length, curvature and signature of the edges, and the scalar methods for region detection [Sonka et al., 1998], which analyze the area, "eccentricity", and "rectangularity".

It is possible to say that, besides the feature extraction process, there are still two main problems to be addressed in order to allow efficient management of complex data. The first is the fact that the extractors usually generate many features (hundreds or even thousands). As described in the upcoming Section 2.2, it impairs the data storage and retrieval techniques due to the *"curse of dimensionality"* [Beyer et al., 1999, Korn et al., 2001, Kriegel et al., 2009, Moise et al., 2009, Parsons et al., 2004]. Therefore, dimensionality reduction techniques are vital to the success of strategies for indexing, retrieving and analyzing complex data. The second problem stems from the need to compare complex data by similarity, because it usually does not make sense to compare them by equality, as it is commonly done for traditional data [Vieira et al., 2004]. Moreover, the total ordering property does not hold among complex data elements - one can only say that two elements are equal or different, since there is no explicit rule to sort the elements. This fact distinguishes complex data elements even more from the traditional elements. The access methods based on the total ordering property do not support queries involving comparisons by similarity. Therefore, a new class of access methods was created, known as the Metric Access Methods (MAM), aimed at allowing searches by similarity. Examples of such methods are the *M-tree* [Ciaccia et al., 1997], the *Slim-tree* [Traina Jr. et al., 2000], the *DF-tree* [Traina Jr. et al., 2002] and the *DBM-tree*

[Vieira et al., 2004], which are considered to be dynamic methods, since they allow data updates without the need to rebuild the structure.

The main principle for these techniques is the representation of the data in a metric space. The similarity between two elements is calculated by a distance function acting as a metric applied to the pair of elements in the same domain. The definitions of a metric space and the main types of queries applied to it are as follows.

Definition 4 *A **metric space** is defined as a pair $\langle \mathbb{S}, m \rangle$, where $\mathbb{S}$ is the data domain and $m : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+$ a distance function acting as a metric. Given any $s_1, s_2, s_3 \in \mathbb{S}$, this function must respect the following properties: (i) symmetry, $m(s_1, s_2) = m(s_2, s_1)$; (ii) non-negativity, $0 < m(s_1, s_2) < \infty, \forall\ s_1 \neq s_2$; (iii) identity, $m(s_1, s_1) = 0$; and (iv) triangle inequality, $m(s_1, s_2) \leq m(s_1, s_3) + m(s_3, s_2), \forall\ s_1, s_2, s_3 \in \mathbb{S}$.*

Definition 5 *A **range query** in a metric space receives as input an object $s_1 \in \mathbb{S}$ and one specific range ϵ. It returns all objects s_i, provided that $m(s_i, s_1) \leq \epsilon$, based on the distance function m.*

Definition 6 *A k **nearest neighbor query** in a metric space receives as input an object $s_1 \in \mathbb{S}$ and an integer value $k \geq 1$. It returns the set of the k objects closest to s_1, based on the distance function m.*

One particular type of metric space is the d-dimensional space, for which a distance function is defined, denoted as $\left\langle {}^d\mathbb{S}, m \right\rangle$. This case is especially interesting for this Doctoral dissertation, since it allows posing similarity queries over complex objects represented by feature vectors of equal cardinality in a multi-dimensional dataset.

2.2 Knowledge Discovery in Traditional Data

The task of *Knowledge Discovery in Databases - KDD* is defined as: *"The nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."* [Fayyad et al., 1996]. This process is commonly partitioned into three steps: *Preprocessing*, *Mining* and *Result Evaluation* [Rezende, 2002]. In order to obtain high-level information from raw data, the data to be analyzed is usually represented as a set of points in a d-dimensional space for which a distance function acting as a metric is specified, as described in Section 2.1. The attributes of the dataset indicate dimensions and the data objects represent points in the space, while the similarity between pairs of objects is measured in terms of the respective distance function applied to the data space.

With the increasing quantity and complexity of the data generated or collected in digital systems, the task of *Preprocessing* has become essential to the whole KDD process [Sousa et al., 2007]. In this step, the data are reduced and prepared by cleaning, integrating, selecting and transforming the objects to the subsequent mining step. A

major problem to be minimized is the *"curse of dimensionality"* [Beyer et al., 1999, Korn et al., 2001, Kriegel et al., 2009, Moise et al., 2009, Parsons et al., 2004], a term referring to the fact that increasing the number of attributes in the objects quickly leads to significant degradation of the performance and accuracy of existing techniques to access, store and process data. This occurs because data represented in high dimensional spaces tend to be extremely sparse and all the distances between any pair of points tend to be very similar, with respect to various distance functions and data distributions [Beyer et al., 1999, Kriegel et al., 2009, Parsons et al., 2004]. Dimensionality reduction is the most common technique applied to minimize this problem. It aims at obtaining a set of relevant and non-correlated attributes that allow representing the data in a space of lower dimensionality with minimum loss of information. The existing approaches are: feature selection, which discards among the original attributes, the ones that contribute with less information to the data objects; and feature extraction, which creates a reduced set of new features, formed by linear combinations of the original attributes, able to represent the data with little loss of information [Dash et al., 1997].

The mining task is a major step in the KDD process. It involves the application of data mining algorithms chosen according to the goal to be achieved. Such tasks are classified as: predictive tasks, which seek a model to predict the value of an attribute based on the values of other attributes, by generalizing known examples, and descriptive tasks, which look for patterns that describe the intrinsic data behavior [Sousa, 2006].

Classification is a major predictive mining task. It considers the existence of a training set with records classified according to the value of an attribute (target attribute or class) and a test set, in which the class of each record is unknown. The main goal is to predict the values of the target attribute (class) in the database records to be classified. The algorithms perform data classification by defining rules to describe correlations between the class attribute and the others. Examples of such algorithms are genetic algorithms [Ando and Iba, 2004, Zhou et al., 2003] and algorithms based on decision trees [Breiman et al., 1984], on neural networks [Duda et al., 2000] or on the Bayes theorem (Bayesian classification) [Zhang, 2004].

Clustering is an important descriptive task. It is defined as: *"The process of grouping the data into classes or clusters, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters."* [Han and Kamber, 2006]. Traditional clustering algorithms are commonly divided into: (i) hierarchical algorithms, which define a hierarchy between the clusters in a process that may be initiated with a single cluster, recursively partitioned in follow-up steps (*top-down*), or considering at first that each data object belongs to a distinct cluster, recursively merging the clusters latter (*bottom-up*); or (ii) partitioning algorithms, which divide η objects into k clusters, $k \leq \eta$, such that each object belongs to at most one cluster and each cluster contains at least one object. Examples of well-known clustering methods

are k-Means [Lloyd, 1982, MacQueen, 1967, Steinhaus, 1956], k-Harmonic Means [Zhang et al., 2000], DBSCAN [Ester et al., 1996] and STING [Wang et al., 1997].

The last step in the process of knowledge discovery is the result evaluation. At this stage, the patterns discovered in the previous step are interpreted and evaluated. If the patterns refer to satisfactory results (valid, novel, potentially useful and ultimately understandable), the knowledge is consolidated. Otherwise, the process returns to a previous stage to improve the results.

2.3 Clustering Complex Data

Complex data are usually represented by vectors with hundreds or even thousands of features in a multi-dimensional space, as described in Section 2.1. Each feature represents a dimension of the space. Due to the curse of dimensionality, traditional clustering methods, such as k-Means [Lloyd, 1982, MacQueen, 1967, Steinhaus, 1956], k-Harmonic Means [Zhang et al., 2000], DBSCAN [Ester et al., 1996] and STING [Wang et al., 1997], are commonly inefficient and ineffective for these data [Aggarwal and Yu, 2000, Aggarwal et al., 1999, Agrawal et al., 1998, 2005]. The main factor that leads to their inefficiency is that they often have super-linear computational complexity on both cardinality and dimensionality. Traditional methods are also often ineffective, when applied to high dimensional data, as the data tend to be very sparse in the multi-dimensional space and the distances between any pair of points usually become very similar to each other, regarding several data distributions and distance functions [Beyer et al., 1999, Kriegel et al., 2009, Moise et al., 2009, Parsons et al., 2004]. Thus, traditional methods do not solve the problem of clustering large, complex datasets [Aggarwal and Yu, 2000, Aggarwal et al., 1999, Agrawal et al., 1998, 2005, Domeniconi et al., 2007, Kriegel et al., 2009, Moise et al., 2009, Parsons et al., 2004, Tung et al., 2005].

Dimensionality reduction methods minimize the effects of the dimensionality curse by finding a new set of orthogonal dimensions, of cardinality smaller than the original set's one, composed of non-correlated dimensions relevant to characterize the data. The elements of this set can be original dimensions, in the case of feature selection methods, or linear combinations of them, for feature extraction methods. Notice however that only global correlations are identified by such methods. In other words, dimensionality reduction methods look for correlations that occur for *all* dataset elements regarding a set of dimensions. Nevertheless, high dimensional data often present correlations local to subsets of the data elements and dimensions [Domeniconi et al., 2007, Tung et al., 2005]. Thus, distinct groups of data points correlated with different sets of dimensions may exist. Many of these correlations can also be non-linear. Therefore, it is clear that traditional dimensionality reduction techniques do not identify all possible correlations, as they evaluate correlations in the entire dataset, and thus, when used as a preprocessing

step for traditional clustering, they do not solve the problem of clustering large, complex datasets. [Aggarwal and Yu, 2000, Aggarwal et al., 1999, Agrawal et al., 1998, 2005, Domeniconi et al., 2007].

Since high dimensional data often present correlations local to subsets of elements and dimensions, the data are likely to present clusters that only exist in subspaces of the original data space. In other words, although high dimensional data usually do not present clusters in the space formed by all dimensions, the data tend to form clusters when the points are projected into subspaces generated by reduced sets of original dimensions or linear combinations of them. Moreover, different clusters may be formed in distinct subspaces. Several recent studies support this idea and a recent survey on this area is presented at [Kriegel et al., 2009].

Figure 2.1 exemplifies the existence of clusters in subspaces of four 3-dimensional databases over the axes $E = \{x, y, z\}$. Figure 2.1a shows a 3-dimensional dataset projected onto axes x and y, while Figure 2.1b shows the same dataset projected onto axes x and z. There exist two clusters in this data, C_1 and C_2. None of the clusters present a high density of points in the 3-dimensional space, but each cluster is a dense, elliptical object in one subspace. Thus, the clusters exist in subspaces only. Cluster C_1 exists in the subspace formed by axes x and z, while cluster C_2 exists in the subspace $\{x, y\}$. Besides elliptical clusters, real data may have clusters that assume *any* shape in their respective subspaces. The clusters must only be dense in that subspace. To illustrate this fact, we present 'star-shaped' and 'triangle-shaped' clusters in another dataset (Figure 2.1c: x-y projection; Figure 2.1d: x-z projection) and 'curved' clusters in one third example dataset (Figure 2.1e: x-y projection; Figure 2.1f: x-z projection).

Such clusters may also exist in subspaces formed by linear combinations of original axes. That is, clusters like the ones in our previous examples may be arbitrarily *rotated* in the space, thus not being aligned to the original axes. For example, Figures 2.1g and 2.1h respectively plot x-y and x-z projections of one last 3-dimensional example dataset. Similarly to clusters C_1 and C_2, the clusters in this data are also dense, elliptical objects in subspaces, but, in this case, the subspaces are planes generated by linear combinations of the axes $\{x, y, z\}$. Traditional clustering is likely to fail when analyzing this data, as each cluster is spread over an axis (generated by linear combinations of the original axes) in the 3-dimensional space. Also, dimensionality reduction applied to the entire dataset does not help, as no 1- or 2-dimensional projection, axis aligned or not, keeps the clusters apart. This problem tends to be even worse in spaces of higher dimensionality. In fact, there is an interdependence between traditional clustering and dimensionality reduction that prevents them from solving the problem of clustering complex data. It is a fact that traditional clustering depends on a prior dimensionality reduction to analyze this kind of data. On the other hand, dimensionality reduction treats only global correlations, the ones that happen with regard to all data elements. Correlations local to subsets of the data cannot

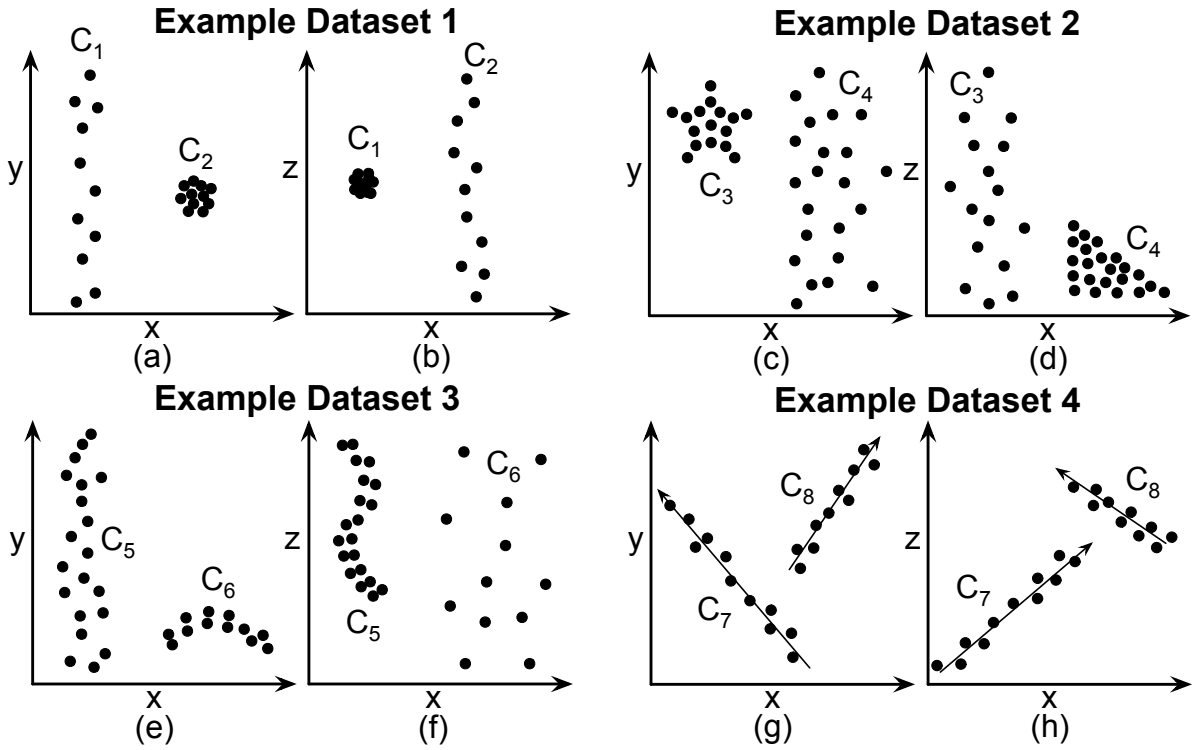


Figure 2.1: x-y and x-z projections of four 3-dimensional datasets over axes $\{x, y, z\}$. From (a) to (f): clusters in the subspaces $\{x, y\}$ and $\{x, z\}$. (g) and (h): clusters in subspaces formed by linear combinations of $\{x, y, z\}$.

be identified without knowing the respective subsets. In other words, clustering complex data is not possible due to the high dimensionality, but the dimensionality reduction is incomplete without the prior knowledge of the data clusters where local correlations occur.

The most common strategy used to untangle this knotty problem is to unify both tasks, clustering and dimensionality reduction, creating a single task. Several methods have used this idea in order to look for clusters together with the subspaces where the clusters exist. According to a recent survey [Kriegel et al., 2009], such methods differ from each other in two major aspects: (i) the search strategy used in the process, which can be *top-down* or *bottom-up*, and (ii) the characteristics of the clusters sought.

The *Bottom-up* algorithms usually rely on a property named *downward closure* or *monotonicity*. This property, valid for some criteria of clustering characterization, warrants that: if there is at least one cluster in a d-dimensional dataset, at least one cluster will stand out when the data are projected into each of the possible subspaces formed by the original dimensions [Agrawal et al., 2005, Kriegel et al., 2005, 2009, Parsons et al., 2004]. As an example, this property is valid when the criterion used for clustering characterization is the *minimum density threshold*. By this criterion, a part of the data space contains a cluster if its density of points meet or exceed the specified minimum bound. The density of points never reduces when data are projected onto subspaces of the original space. Thus, if there is one part of a d-dimensional space, whose density of points is sufficient to characterize a cluster, it is possible to affirm that at least one

cluster will stand out, i.e. at least one space region will be dense enough, when the data are projected into any subspace formed by the original dimensions.

Based on a clustering characterization criterion in which the *downward closure* property applies, *bottom-up* algorithms assume that: if a cluster exists in a space of high dimensionality, it has to exist or to be part of some cluster in all subspaces of lower dimensionality formed by original dimensions [Agrawal et al., 2005, Kriegel et al., 2009, Parsons et al., 2004]. These methods start analyzing low dimensional subspaces, usually 1-dimensional ones, in order to identify the subspaces that contain clusters. The subspaces selected are then united in a recursive procedure that allows the identification of subspaces of higher dimensionality in which clusters also exist. Various techniques are used to spot and to prune subspaces without clusters so that they are ignored in the process, minimizing the computational cost involved, but, in general, the main shortcomings of *bottom-up* methods are: (i) they often have super-linear computational complexity or even exponential complexity regarding the dimensionality of the subspaces analyzed, and (ii) fixed density thresholds are commonly used, assuming that clusters in high dimensional spaces are as dense as clusters in subspaces of smaller dimensionality, which is unlikely to be true in several cases.

The *Top-down* algorithms assume that the analysis of the space with all dimensions can identify patterns that lead to the discovery of clusters existing in lower dimensional subspaces only [Parsons et al., 2004]. This assumption is known in literature as the *locality assumption* [Kriegel et al., 2009]. After identifying a pattern, the distribution of points surrounding the pattern in the space with all dimensions is analyzed to define whether or not the pattern refers to a cluster and the subspace in which the possible cluster is better characterized. The main drawbacks of *top-down* algorithms are: (i) they often have super-linear computational complexity, though not exponential complexity, with regard to the data dimensionality, and; (ii) there is no guarantee that the analysis of the data distribution in the space with all dimensions is always sufficient to identify clusters that exist in subspaces only.

Clustering algorithms for high dimensional data also differ from each other in the characteristics of the clusters that they look for. Some algorithms look for clusters that form a dataset partition, together with a set of outliers, while other algorithms partition the database regarding specific subspaces, so that the same data element can be part of two or more clusters that overlap with each other in the space with all dimensions, as long as the clusters are formed in different subspaces [Parsons et al., 2004]. Finally, the subspaces analyzed by these methods may be limited to subsets of the original dimensions or may not be limited to them, thus including subspaces formed by linear combinations of the original dimensions [Kriegel et al., 2009].

Subspace clustering algorithms aim at analyzing projections of the dataset into spaces formed by subsets of the *original dimensions only.* Given a subspace and its

corresponding data projection, these algorithms operate similarly to traditional clustering algorithms, partitioning the data into disjoint sets of elements, named **subspace clusters**, and a set of outliers. A data point can belong to more than one subspace cluster, as long as the clusters exist in different subspaces. Therefore, subspace clusters are *not necessarily disjoint*. Examples of subspace clustering algorithms are: CLIQUE [Agrawal et al., 1998, 2005], ENCLUS [Cheng et al., 1999], SUBCLU [Kailing et al., 2004], FIRES [Kriegel et al., 2005], P3C [Moise et al., 2006, 2008] and STATPC [Moise and Sander, 2008].

Projected clustering algorithms aim at partitioning the dataset into *disjoint* sets of elements, named **projected clusters**, and a set of outliers. A subspace formed by the *original dimensions* is assigned to each cluster, and the cluster elements are densely grouped, when projected into the respective subspace. Examples of projected clustering algorithms in literature are: PROCLUS [Aggarwal et al., 1999], DOC/FASTDOC [Procopiuc et al., 2002], PreDeCon [Bohm et al., 2004], COSA [Friedman and Meulman, 2004], FINDIT [Woo et al., 2004], HARP [Yip and Ng, 2004], EPCH [Ng and Fu, 2002, Ng et al., 2005], SSPC [Yip et al., 2005], P3C [Moise et al., 2006, 2008] and LAC [Al-Razgan and Domeniconi, 2006, Domeniconi et al., 2004, 2007].

Correlation clustering algorithms aim at partitioning the database in a manner analogous to what occurs with projected clustering techniques - to identify *disjoint* clusters and a set of outliers. However, the clusters identified by such methods, named **correlation clusters**, are composed of densely grouped elements in subspaces formed by the original dimensions of the database, *or by their linear combinations*. Examples of correlation clustering algorithms found in literature are: ORCLUS [Aggarwal and Yu, 2002, 2000], 4C [Bohm et al., 2004], CURLER [Tung et al., 2005], COPAC [Achtert et al., 2007] and CASH [Achtert et al., 2008]. Notice that, to the best of our knowledge, CURLER is the only method found in literature that spots clusters formed by non-linear, local correlations, besides the ones formed by linear, local-correlations.

2.4 Labeling Complex Data

In this section, we assume that the clustering algorithms aimed at analyzing complex data, such as the ones cited in the previous section, can also serve as a basis to perform one distinct data mining task – the task of labeling large sets of complex objects, as we will discuss in the upcoming Chapter 6. For that reason, the following paragraphs introduce some background knowledge related to the task of labeling complex data, i.e., the task of analyzing a given collection of complex objects, in which a few objects have labels, in order to spot appropriate labels for the remaining majority.

Specifically, the task of labeling is one predictive data mining task that considers the existence of a training set containing records labeled with keywords and a test set, in which the labels of each record are unknown. Its main goal is to assign appropriate keywords to

the database records to be labeled. Unlike other data mining tasks, the task of labeling is not completely defined in the literature and slight conceptual divergences exist in the definitions provided by distinct authors, while some authors consider that labeling is one type of classification or use other names to refer to it (e.g., captioning). In this Ph.D. work, we consider that labeling refers to a generalized version of the classification task, in which the restriction of mandatorily assigning one and only one label to every data object does not exist. Specifically, we assume that labeling generalizes the task of classification with regard to at most three aspects: (i) it may consider that the dataset commonly contains objects that differ too much from the labeled training examples, which should be returned to the user as outliers that potentially deserve a new label of their own; (ii) it may allow any object to receive more than one appropriate label; and (iii) it may use hierarchies of labels, in a way that each object can be assigned to entire paths in the hierarchy, instead of being linked to individual labels only.

Regarding complex data, labeling has been mostly applied to image datasets and also to sets of image regions segmented or arbitrarily extracted from larger images. There is an extensive body of work on the labeling of unlabeled regions extracted from partially labeled images in the computer vision field, such as image segmentation and region classification [Lazebnik and Raginsky, 2009, Shotton et al., 2006, Torralba et al., 2008]. The Conditional Random Fields (CRF) and boosting approach [Shotton et al., 2006] shows a competitive accuracy for multi-label labeling and segmentation, but it is relatively slow and requires many training examples. The KNN classifier [Torralba et al., 2008] may be the fastest way for image region labeling, but it is not robust against outliers. Also, the Empirical Bayes approach [Lazebnik and Raginsky, 2009] proposes to learn contextual information from unlabeled data. However, it may be difficult to learn the context from several types of complex data, such as from satellite images.

The Random Walk with Restart (RWR) [Tong et al., 2008] algorithm has served as a basis to other labeling methods. The idea is to perform labeling by creating a graph to represent the input complex objects to be labeled, the given example labels and the similarities existing between the objects. Then, random walks in this graph allow spotting the most appropriate labels for the remaining unlabeled objects. In general, RWR consists into performing walks in a graph according to the following strategy: a random walker starts a walk from a vertex V of the graph, and, at each time step, the walker either goes back to the initial vertex V, with a user-defined probability c, or it goes to a randomly chosen vertex that shares an edge with the current vertex, with probability $1 - c$. The intuition is that this procedure provides an appropriate relevance score between two graph nodes, since the steady state probability that a random walker will find itself in a vertex V', always restarting the walk from a vertex V, is a way to measure the closeness between V and V'. For labeling, RWR is commonly used to measure the closeness between the graph nodes that represent data objects and the ones that represent keywords.

GCap [Pan et al., 2004] is one of the most famous labeling methods that uses random walks with restarts as a basis. GCap proposes a graph-based strategy for automatic image labeling, which can also be applied to any set of multimedia objects. It represents images and label keywords by multiple layers of nodes in a graph and captures the similarities between pairs of images by creating edges to link the nodes that refer to similar images. The known labels become links between the respective images and keywords. This procedure creates a tri-partite graph that represents the input images and labels, besides the existing similarities between the images. Given an image node of interest, random walks with restarts (RWR) are used to perform proximity queries in this graph, allowing GCap to automatically find the best annotation keyword for the respective image. Unfortunately, GCap remains rather inefficient, since it searches for the nearest neighbors of every image in the image feature space to create edges between similar image nodes, and this operation is super-linear even with the speed up offered by many approximate nearest-neighbor finding algorithms (e.g., the ANN Library [Mount and Arya]).

2.5 MapReduce

The large amounts of data collected by the enterprises are accumulating data, and today it is already feasible to have Terabyte- or even Petabyte-scale datasets that must be submitted for data mining processes (e.g., Twitter crawl: > 12 TB, Yahoo! operational data: 5 *Petabytes* [Fayyad, 2007a]), such as the processes of clustering and labeling complex data. However, the use of serial data mining algorithms, like the ones described in the previous sections, to analyze such huge amounts of data is clearly an impractical task. Just to read a single Terabyte of data (at 5GB/min on a single modern eSATA disk) one takes more than 3 hours. Therefore, to improve the existing serial data mining methods in order to make them run efficiently in parallel is nowadays extremely desirable. With that in mind, this section describes the `MapReduce` framework, a promising tool for large-scale, parallel data analysis, which has been proving to offer a valuable support to the execution of data mining algorithms in a parallel processing environment.

`MapReduce` is a programming framework [Dean and Ghemawat, 2004] fostered by Google[1] to process large-scale data in a massively parallel way. `MapReduce` has two major advantages: the programmer is oblivious of the details related to the data storage, distribution, replication, load balancing, etc.; and furthermore, it adopts the familiar concept of functional programming. The programmer needs to specify only two functions, a *map* and a *reduce*. The typical framework is as follows [Lämmel, 2008]: (a) the *map* stage passes over the input file and outputs (key, value) pairs; (b) the *shuffling* stage transfers the mappers output to the reducers based on the key; (c) the *reduce* stage processes the received pairs and outputs the final result. Due to its scalability, simplicity

[1] www.google.com

and the low cost to build large clouds of computers, `MapReduce` is a very promising tool for large scale, parallel data analysis, which has already being reflected in the academia (e.g., [Papadimitriou and Sun, 2008] [Kang et al., 2009] [Kang et al., 2010]).

`Hadoop` is the open source implementation of `MapReduce`. `Hadoop` provides the Hadoop Distributed File System (HDFS) [Had], HBase [Wiki], which is a way to efficiently store and handle semi-structured data as Google's BigTable storage system [Chang et al., 2006], and PIG, a high level language for data analysis [Olston et al., 2008].

2.6 Conclusions

In this chapter we presented an overview of the concepts that are the basis for the development of this Doctoral dissertation. We described the research areas of processing complex data and knowledge discovery in traditional databases, besides the main factors that distinguish the tasks of traditional clustering and clustering complex data. The task of labeling large sets of complex objects was also discussed. Finally, we introduced the `MapReduce` framework, a promising tool for large scale data analysis, which has been proving to offer a valuable support to the execution of data mining algorithms in a parallel processing environment. The next chapter describes some of the most relevant clustering methods available in literature for multi-dimensional data with more than five or so dimensions, which look for clusters formed by local correlations.

Chapter

3

Clustering Methods for Moderate-to-High Dimensionality Data

Traditional clustering methods are usually inefficient and ineffective over data with more than five or so dimensions. In Section 2.3, we discuss the main reasons that lead to this fact. It is also mentioned that the use of dimensionality reduction methods does not solve the problem, since it allows one to treat only the global correlations in the data. Correlations local to subsets of the data cannot be identified without the prior identification of the data clusters where they occur. Thus, algorithms that combine dimensionality reduction and clustering into a single task have been developed to look for clusters together with the subspaces of the original space where they exist. Some of these algorithms are described in this chapter. Specifically, we first present a brief survey on the existing algorithms, and later we detail four of the most relevant ones. Then, in order to help one to evaluate and to compare the algorithms, we conclude the chapter by presenting a table to link some of the most relevant techniques with the main desirable properties that any clustering technique for moderate-to-high dimensionality data should have. The general goal of the chapter is to identify the main strategies already used to deal with the problem, besides the key limitations of the existing techniques.

Remark: the concepts considered by each technique are not equal, in spite of being similar in several cases, and thus, slight conceptual divergences are contemplated by the original notation. To avoid conflicts in the concepts, this chapter describes each algorithm following the original notation used by its authors. Consequently, the list of symbols shown at the beginning of this Doctoral dissertation should be disregarded in this chapter.

3.1 Brief Survey

CLIQUE [Agrawal et al., 1998, 2005] was probably the first technique aimed at finding clusters in subspaces of multi-dimensional data. It uses a bottom-up approach, dividing 1-dimensional data projections into a user-defined number of partitions and merging dense partitions to spot clusters in subspaces of higher dimensionality. CLIQUE scales exponentially regarding the cluster dimensionality and it relies on a fixed density threshold that assumes high-dimensional clusters to be as dense as low-dimensional ones, an assumption that is often unrealistic. Several posterior works, such as ENCLUS [Cheng et al., 1999], EPCH [Ng et al., 2005], P3C [Moise et al., 2006, 2008], SUBCLU [Kröger et al., 2004] and FIRES [Kriegel et al., 2005] improve the ideas of CLIQUE to reduce its drawbacks, but they are still typically super-linear in space or in running time.

PROCLUS [Aggarwal et al., 1999] proposed a top-down clustering strategy, assuming what is known in the literature as the *locality assumption*: the analysis of the space with all dimensions is sufficient to find patterns that lead to clusters that only exist in subspaces. PROCLUS is a k-medoid method that assigns to each medoid a subspace of the original axes. Each point is assigned to the closest medoid in its subspace. An iterative process analyzes the points distribution of each cluster in each axis and the axes in which the cluster is denser form its subspace. PROCLUS scales super-linearly on the number of points and axes and it only finds clusters in subspaces formed by the original axes.

ORCLUS [Aggarwal and Yu, 2002, 2000] and CURLER [Tung et al., 2005] improve the ideas of PROCLUS to find arbitrarily oriented correlation clusters. They analyze each cluster's orientation, based on the data attributes eigenvector with the biggest eigenvalue, in an iterative process that merges close clusters with similar orientations. CURLER spots even non-linear, local correlations, but it has a quadratic time complexity regarding the number of clusters and their dimensionalities, and its complexity is cubic with respect to the data dimensionality. Other well-known, top-down methods are: DOC/FASTDOC [Procopiuc et al., 2002], PkM [Agarwal and Mustafa, 2004], LAC [Domeniconi et al., 2007], RIC [Böhm et al., 2006], LWC/CLWC [Cheng et al., 2008], PreDeCon [Bohm et al., 2004], OCI [Böhm et al., 2008], FPC/CFPC [Yiu and Mamoulis, 2005], COSA [Friedman and Meulman, 2004], HARP [Yip et al., 2004] and STATPC [Moise and Sander, 2008].

Density-based strategies have also been used for correlation clustering. 4C [Böhm et al., 2004] finds arbitrarily oriented clusters, by extending each cluster from a seed as long as a density-criterion is fulfilled. Otherwise, it picks another seed, until all points are classified. The density criterion used is a minimal required number of points in the neighborhood of each point, and the neighborhood is defined with a distance function that uses as a basis the eigensystems of its input points. 4C tends to uncover clusters of a single, user-defined dimensionality at each run, but the algorithm COPAC [Achtert et al., 2007] improves the ideas of 4C to fix this issue.

CASH [Achtert et al., 2008] uses a novel idea that does not rely on the *locality assumption*. Based on the Hough transform [Hough, 1962], it proposes to map the data space into a parameter space defining the set of all possible arbitrarily oriented subspaces. Then, it analyzes the parameter space to find those among all the possible subspaces that accommodate many database objects, leading to the identification of correlation clusters. Unfortunately, CASH remains rather inefficient, as it has a cubic average time complexity regarding the data dimensionality d and its worst case time complexity is $O\left(2^d\right)$.

CARE [Zhang et al., 2008] formalizes the correlation clustering problem as the problem of finding feature subsets strongly correlated with regard to large portions of the input dataset. It uses the Spectrum Theory to study the monotonicity properties of the problem, which allows the proposal of heuristics to prune the problem's search space. However, CARE has four user-defined input parameters and it usually scales quadratically on the data dimensionality, with an exponential, theoretical worst case time complexity.

This section presented a brief survey on the existing algorithms well-suited to analyze moderate-to-high dimensionality data. A recent survey on this area is given by [Kriegel et al., 2009]. In the next sections, four of the most relevant algorithms are detailed.

3.2 CLIQUE

The method CLIQUE [Agrawal et al., 1998, 2005] was probably the first method aimed at finding clusters in subspaces of multi-dimensional data. It proposes a *bottom-up* search strategy, to identify *subspace clusters*.

The process starts by analyzing the input data projected into the 1-dimensional subspaces formed by each of the original dimensions. In this step, a data scan is performed to partition each projection into ξ equal sized intervals, which enables the creation of histograms that represent the points distribution regarding each interval and dimension. An interval is considered to be a *dense unit* when the percentage of elements inside it, with regard to the total number of elements, is greater than or equal to a density threshold τ. The values of ξ and τ are user-defined input parameters.

Examples of dense units are illustrated in Figure 3.1, which was adapted from [Agrawal et al., 2005]. Figures 3.1a, 3.1b and 3.1c respectively present an example dataset over the set of dimensions $A = \{x, y\}$ and its projections in the 1-dimensional subspaces formed by dimensions x and y. The database contains a total of 14 elements and each 1-dimensional projection is partitioned into $\xi = 10$ equal sized intervals. Given a density threshold $\tau = 20\%$ and one of these intervals, CLIQUE assumes that the interval is a dense unit if and only if it has at least 3 points. Consequently, after a data scan, the algorithm finds that there are three dense units in our toy dataset with regard to the projection in x and one dense unit in the projection in y. These are identified as $(2 \leq x < 3)$, $(5 \leq x < 6)$, $(6 \leq x < 7)$ and $(55 \leq y < 60)$, according to their respective ranges of values in x or in y.

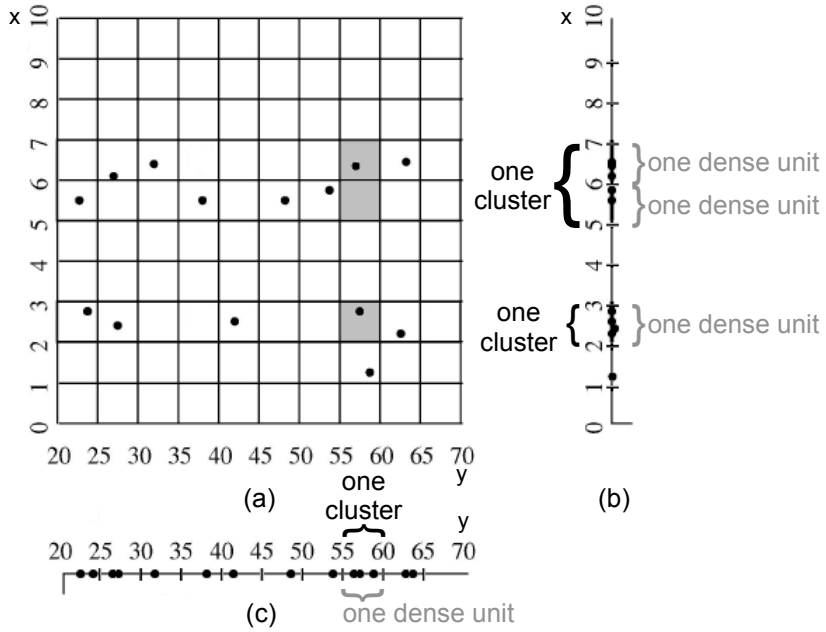


Figure 3.1: Examples of 1-dimensional dense units and clusters existing in a toy database over the dimensions $A = \{x, y\}$. The figure was adapted from [Agrawal et al., 2005].

Once the 1-dimensional dense units are identified, the process continues in order to find k-dimensional dense units, for $k > 1$. A pair of dense units in subspaces with $k-1$ dimensions generates a candidate k-dimensional dense unit if and only if the respective subspaces share exactly $k-2$ dimensions and both units are in the same positions regarding the shared dimensions. After identifying all candidate k-dimensional dense units, a data scan allows the algorithm to verify if the candidates are actually dense units. A candidate is said to be a dense unit if the number of points that fall into it is enough with regard to the threshold τ. The process continues recursively and it ends when, in an iteration: new dense units are not found, or; the space with all dimensions is analyzed.

In our running example from Figure 3.1, the 1-dimensional dense units identified lead to the existence of three candidates, 2-dimensional dense units, filled in gray in Figure 3.1a. These are described as $(2 \leq x < 3) \wedge (55 \leq y < 60)$, $(5 \leq x < 6) \wedge (55 \leq y < 60)$ and $(6 \leq x < 7) \wedge (55 \leq y < 60)$. However, after a data scan, it is possible to notice that none of the candidates has three or more points, which is the minimum number of points needed to spot a dense unit in this example. Thus, the recursive process is terminated.

CLIQUE also proposes an algorithm to prune subspaces in order to minimize its computational costs. It considers that the larger the sum of points in the dense units of a subspace analyzed, the more likely that this subspace will be useful in the next steps of the process. Thus, some subspaces analyzed by the method that have small sums of points in their dense units may be ignored in follow-up steps. For example, consider the dense units in our toy dataset from Figure 3.1. The sums of points in the dense units of the 1-dimensional subspaces formed by dimensions x and y are 13 and 3 respectively.

Thus, CLIQUE considers that, if needed, it is better to prune the subspace formed by dimension y than the one formed by x.

Pruning is performed as follows: at each iteration of the process, the dense units found are grouped according to their respective subspaces. The subspaces are then put into a list, sorted in descending order regarding the sum of points in their dense units. The principle *Minimum Description Length (MDL)* [Grunwald et al., 2005, Rissanen, 1989] is then used to partition this list, creating two sublists: *interesting subspaces* and *uninteresting subspaces*. The MDL principle allows maximizing the homogeneity of the values in the sublists regarding the sum of points in the dense units of each subspace. After this step is finished, the dense units that belong to the *uninteresting subspaces* are discarded and the recursive process continues, considering only the remaining dense units.

Once the recursive process is completed, the dense units identified are merged so that maximum sets of dense units adjacent in one subspace indicate clusters in that subspace. The problem of finding maximum sets of adjacent, dense units is equivalent to a well-known problem in the Graph Theory, the search for connected subgraphs. This is verified considering a graph whose vertices correspond to dense units in one subspace and edges connect two vertices related to adjacent, dense units, i.e., the ones that have a common face in the respective subspace. Thus, a *depth-first* search algorithm [Aho et al., 1974] is applied to find the maximum sets of adjacent, dense units defining the clusters and their corresponding subspaces. In our example, the dense units found in Figure 3.1b form two clusters in the subspace defined by dimension x. One cluster is related to the dense unit ($2 \leq x < 3$) while the other refers to the pair of adjacent dense units ($5 \leq x < 6$) and ($6 \leq x < 7$). Finally, the subspace of dimension y contains a single cluster, represented by the dense unit ($55 \leq y < 60$), and no cluster exists in the 2-dimensional space.

The main shortcomings of CLIQUE are: (i) even considering the pruning of subspaces proposed, the time complexity of the algorithm is still exponential with regard to the dimensionality of the clusters found; (ii) due to the pruning technique used, there is no guarantee that all clusters will be found; (iii) there is no policy suggested to define appropriate values to the parameters ξ and τ, and; (iv) the density parameter τ assumes that clusters in subspaces of high dimensionality should be as dense as clusters in subspaces of lower dimensionality, a fact which is often unrealistic.

3.3 P3C

P3C [Moise et al., 2006, 2008] is a well-known method for clustering moderate-to-high dimensional data. It uses a *bottom-up* search strategy that allows finding *subspace clusters* or *projected clusters*. The method assumes that the initial analysis of only attributes with non-uniform distribution is enough to identify clusters that exist in subspaces of the original space, and the first step of the process is the identification of such attributes.

Therefore, based on well-known statistical principles, projections of the data in each of the original axes are partitioned into $\lfloor 1 + log_2 n \rfloor$ equal-sized intervals each, where n is the total number of data elements. The statistical test *Chi-square goodness-of-fit* [Snedecor and Cochran, 1989] is then applied on each axis in order to identify and discard, with confidence level $\alpha = 0.001$, the axes that have uniform data distribution.

The non-discarded axes are then processed as follows: given one of such axes, its intervals are again tested by the *Chi-square* test, this time ignoring the interval with the largest number of elements. If the test indicates a non-uniform distribution in the intervals analyzed, the same statistical test is applied once more to all intervals, except to the ones with the first and the second largest numbers of points. This process continues recursively, ignoring the intervals with the largest numbers of points as needed, until the test indicates one uniform distribution in the intervals analyzed. After that, the intervals ignored in the last test performed are called 1-*signatures*. Notice two remarks: the number one in the 1-*signatures* refers to the fact that the corresponding intervals are 1-dimensional, and; adjacent intervals are said to form a single 1-*signature*.

After the recursive procedure is finished, the 1-*signatures* identified serve as a basis to spot *p-signatures*, for $p > 1$. In other words, P3C analyzes sets of dense, 1-dimensional intervals to identify sets of dense intervals of higher dimensionality. As an example, the toy dataset shown in previous illustrations is used once more, this time to illustrate P3C. Figure 3.2, adapted from [Agrawal et al., 2005], illustrates this dataset. It contains 14 elements over the set of dimensions $A = \{x, y\}$. Figures 3.2a, 3.2b and 3.2c respectively show the dataset and its projections in dimensions x and y. In this example, P3C would probably find the 1-*signatures* $\mathbf{S_1}$, $\mathbf{S_2}$ and $\mathbf{S_3}$. Note that the 1-*signatures* identify possible 2-*signatures* in the example data, i.e., possible dense 2-dimensional intervals. In our running example, there are identified those formed by the sets of 1-*signatures* $\{\mathbf{S_1}, \mathbf{S_3}\}$ and $\{\mathbf{S_2}, \mathbf{S_3}\}$, which delimit the search space into two parts, filled in gray in our illustration.

In order to find *p-signatures*, for $p > 1$, P3C analyzes pairs $(\mathbf{S}, S_{p'})$, where $\mathbf{S}$ is a $(p-1)$-*signature* that contains intervals in $p-1$ distinct dimensions, and $S_{p'}$ represents an interval in dimension j, such that none of the intervals in $\mathbf{S}$ refer to dimension j. Specifically, P3C analyzes the distribution of the points within the intervals of $\mathbf{S}$ regarding dimension j to confirm a possible *p-signature* formed by $\mathbf{S} \cup \{S_{p'}\}$. Let us consider that there are $Supp(\mathbf{S})$ elements within the intervals of $\mathbf{S}$ and that the interval $S_{p'}$ represents $width(S_{p'})$ percent of the domain of attribute j. For the uniform distribution, the expected number of elements that are within the intervals of $\mathbf{S}$ and are also within the interval $S_{p'}$ is given by $ESupp(\mathbf{S} \cup \{S_{p'}\} \,|\mathbf{S}) = Supp(\mathbf{S}) * (width(S_{p'})/100)$. That is, for the case where the points within the intervals of $\mathbf{S}$ are randomly distributed through dimension j, $ESupp(\mathbf{S} \cup \{S_{p'}\} \,|\mathbf{S})$ defines the number of points expected by chance to be also within the interval $S_{p'}$. P3C assumes that the union $\mathbf{S} \cup \{S_{p'}\}$ forms a *p-signature*, if and only if the number of data elements within the intervals of the possible *p-signature*, $Supp(\mathbf{S} \cup \{S_{p'}\})$,

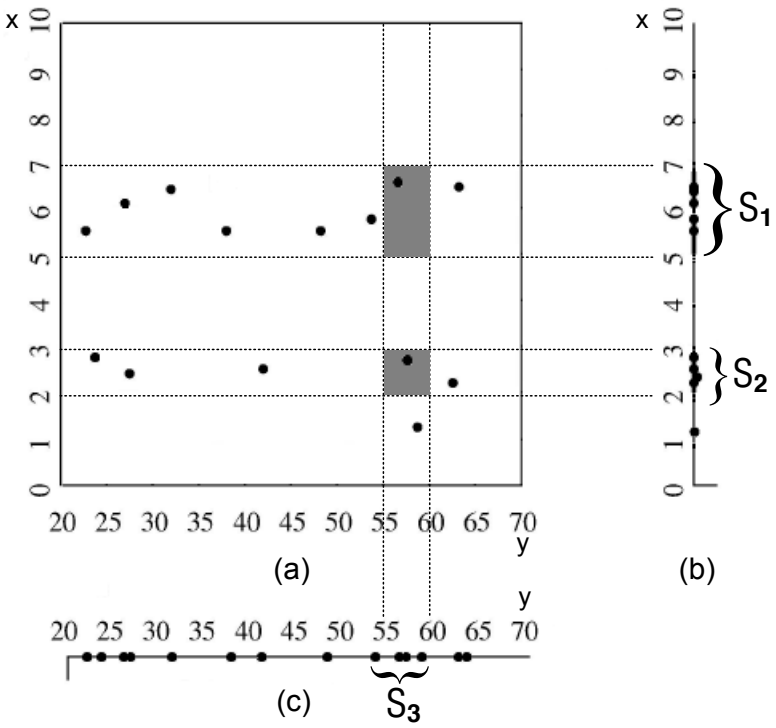


Figure 3.2: Examples of *p-signatures* in a database over the dimensions $A = \{x, y\}$. The figure was adapted from [Agrawal et al., 2005].

is sufficiently greater than $ESupp(\mathbf{S} \cup \{S_{p'}\} | \mathbf{S})$, the number expected for the uniform distribution. For this task, the Poisson distribution [Snedecor and Cochran, 1989] is used to measure the probability of observing $Supp(\mathbf{S} \cup \{S_{p'}\})$ elements in a given space region, when the expected number of elements for the region is $ESupp(\mathbf{S} \cup \{S_{p'}\} | \mathbf{S})$, under the uniform distribution. If $Supp(\mathbf{S} \cup \{S_{p'}\}) > ESupp(\mathbf{S} \cup \{S_{p'}\} | \mathbf{S})$ and the probability is small enough, smaller than or equal to one user-defined threshold named $PoissonThreshold$, P3C considers that the number of elements $Supp(\mathbf{S} \cup \{S_{p'}\})$ is sufficiently greater than the number expected by chance $ESupp(\mathbf{S} \cup \{S_{p'}\} | \mathbf{S})$ and thus the union $\mathbf{S} \cup \{S_{p'}\}$ is considered to be a *p-signature*. Otherwise, the union $\mathbf{S} \cup \{S_{p'}\}$ does not form a *p-signature*.

In the subsequent step, each *p-signature* $\mathbf{S}$, together with the set of data elements delimited by its intervals, may be considered as a first draft of a cluster, thus being named a *cluster core*. $\mathbf{S}$ defines a *cluster core* if and only if there is no interval $S_{p'}$, not already included in $\mathbf{S}$, whose union with $\mathbf{S}$ allows the creation of a $(p+1)$-*signature*.

After all *cluster cores* had been identified, the dataset is projected into the subspace formed by the axes not discarded in the first stage, i.e., the axes considered to have non-uniform data distributions. In this subspace, any data element not yet belonging to a *cluster core* is inserted into the set of elements of its nearest *cluster core*, the one with the nearest centroid considering the *Mahalanobis* distance function. Then, P3C identifies outliers by using a traditional method [Rousseeuw and van Zomeren, 1990], still considering the same subspace and distance function.

Finally, P3C mounts a matrix M with n rows and k columns, where n is the number of elements in the database and k is the number of *cluster cores* found. $M[i][l]$ receives the value 0 if the data element i does not belong to the *cluster core* l. Otherwise, the value in $M[i][l]$ indicates the fraction of *cluster cores* that contain the element i. This matrix is used as input for the algorithm *Expectation/Maximization* (EM) [Dempster et al., 1977] to compute the probability of each data element to belong to each of the *cluster cores* found. In this way, P3C is able to find: *projected clusters*, by naming each data element after its *cluster core* of maximum likelihood, or; *subspace clusters*, by considering the *cluster cores* to be the final clustering result, and thus, ignoring the probability computation.

The main shortcomings of P3C are: (i) its computational cost is exponential with regard to the dimensionality of the clusters found; (ii) it does not find clusters that exist in subspaces whose all dimensions, individually have uniform data distributions; and (iii) the method is highly sensitive to rotation, because the clusters found must exist in subspaces formed by the original dimensions only.

3.4 LAC

LAC [Al-Razgan and Domeniconi, 2006, Domeniconi et al., 2004, 2007] is a k-means-based method aimed at clustering moderate-to-high dimensionality data. Given a user-defined number of clusters k, the algorithm partitions the dataset into k disjoint sets of elements, disregarding the possible existence of outliers. Its main difference from the traditional k-means method is to consider that any original dimension can be more relevant or less relevant than the other ones to characterize each cluster. The more the points of a cluster are densely grouped when projected into a dimension, the highest is the relevance of that dimension to the cluster. In this way, one dimension may be highly relevant to one cluster and, at the same time, it may have a lower relevance with regard to other clusters.

The final clustering result is a set of k data clusters and k weighting vectors. The weighting vectors represent the relevance of each original dimension with regard to each cluster found, in a way that the elements in each cluster are densely grouped into the space with all dimensions according to the L_2 distance function, weighted by the corresponding vector. Provided that LAC defines a dataset partition by only analyzing original dimensions, most authors from the literature consider that it finds clusters similar to *projected clusters*, besides the fact that LAC does not clearly define the subspaces where the clusters found exist.

Given a database S with N points on an Euclidean D-dimensional space, the algorithm computes a set of k centroids $\{c_1, c_2, \ldots c_k\}$, for $c_j \in \mathbb{R}^D$ and $1 \leq j \leq k$, together with a set of weighting vectors $\{w_1, w_2, \ldots w_k\}$, for $w_j \in \mathbb{R}^D$ and $1 \leq j \leq k$. The centroids and their weighting vectors define k clusters $\{S_1, S_2, \ldots\ S_k\}$ in a way that the clusters minimize the sum of the squared L_2 weighted distances between each data point and the

centroid of its cluster, according to the corresponding weighting vector. The proposed procedure ensures that $S = S_1 \cup S_2 \cup \ldots S_k$ will always be true, and it also guarantees that, given any two clusters, S_a and S_b, the expression $S_a \cap S_b = \emptyset$ will always apply, for $1 \leq a \leq k$, $1 \leq b \leq k$ and $a \neq b$.

LAC uses a *top-down* clustering strategy that starts with the choice of k centroids. The first centroid is chosen at random. The second one maximizes the L_2 distance to the first centroid. The third centroid maximizes the L_2 distances to the two centroids chosen before, and so on. Equal weights are initially defined for all centroids regarding all dimensions. Therefore, the initial dataset partition is performed by assigning each data point to the cluster of its nearest centroid, based on the unweighted L_2 distance function.

After the initial clusters are defined, LAC improves the centroids and the weighting vectors, by minimizing the sum of the squared L_2 distances between each data point and the centroid of its cluster, weighted by the corresponding weighting vector. At first, the weighting vectors are updated by analyzing the distribution of the points of each cluster projected into each original dimension individually. The more the points of a cluster are densely grouped when projected into a dimension, the biggest is the weight of that dimension to the cluster. During this process, LAC uses a user-defined parameter h, $0 \leq h \leq \infty$, to control how much the distribution of the values in each weighting vector will deviate from the uniform distribution. Setting $h = 0$, concentrates all the weight that refers to a cluster j on a single axis (the axis i in which the points of j projected into i are best clustered, compared to when these points are projected into each one of the other axes), whereas setting $h = \infty$ forces all axes to be given equal weights for cluster j. Values of h between 0 and ∞ lead to intermediate results.

Once the weighting vectors are updated for the first time, the data is partitioned again by assigning each point to the cluster of its closest centroid according to the L_2 distance function, weighted by the updated centroid's weighting vector. This process defines new centroids, and thus, it creates new clusters, whose point distribution must be analyzed for each dimension to update the weighting vectors one more time. It leads to a recursive process that stops when, after one iteration, none of the weighting vectors change.

The main shortcomings of LAC are: (i) the number of clusters k is user-defined; (ii) the algorithm is non-deterministic, since the first centroid is randomly chosen at the beginning of the process; (iii) although the authors present a formal proof of convergence for LAC, there is no guarantee that the convergence will occur in feasible time for all cases, and; (iv) LAC disregards the possible existence of outliers in the data, which is considered to be a shortcoming, since traditional methods for outlier detection do not treat the case where clusters exist in subspaces of the original space.

3.5 CURLER

CURLER [Tung et al., 2005] is a *correlation clustering* algorithm that looks for clusters using a *top-down* strategy. To the best of our knowledge, it is the only method found in literature that spots clusters formed by local, non-linear correlations, besides clusters formed by local, linear correlations. Although the positive aspects of the method are relevant, CURLER cannot be fully automated, depending on a semi-automatic process where the user visualizes unrefined clusters found by the algorithm to produce the final result. As well as other correlation clustering algorithms, CURLER starts the clustering procedure by analyzing the space with all original axes to find tiny sets of points densely grouped, which are named microclusters. Then, some of the microclusters are merged to form correlation clusters.

In general, microclusters that are close to each other in specific subspaces and have similar orientations should be merged, where the orientation of a microcluster is the eigenvector with the biggest eigenvalue found for the corresponding data objects. However, CURLER points out that the microclusters to be merged must be carefully identified, especially when analyzing data with clusters formed by non-linear, local correlations, since these clusters usually have one global orientation and several distinct local orientations. The global orientation is the eigenvector with the biggest eigenvalue obtained when analyzing all points in the cluster, while the local orientations are obtained when considering specific subsets of these points. Figure 3.3 [Tung et al., 2005] exemplifies this fact for a toy 2-dimensional database that contains a cluster following a sinusoidal curve. As it can be seen, when considering specific parts of the cluster, one can find considerably distinct local orientations, as in the cases of the small neighborhoods centered at points r and q. The global orientation is found only when larger parts of the cluster are analyzed, as in the case of the large neighborhood centered at the point p.

Therefore, pairs of microclusters that belong to the same correlation cluster and have very distinct orientations are likely to exist, despite of the fact that they are close to each other in one subspace. It is also possible that two microclusters of a single cluster have similar orientations, although they are far apart in all subspaces. Thus, the specification of levels of importance for the analyses of proximity and orientation in the merging process is a complex task and, at the same time, it is a vital task for the definition of what microclusters should be merged in order to generate an accurate clustering result. CURLER defines a novel similarity measure for pairs of microclusters, named *co-sharing level*, to tackle this difficult problem.

The algorithm searches for microclusters based on the Fuzzy Logic and on the Expectation/Maximization (EM) technique [Banfield and Raftery, 1993]. Similarly to the k-means method, the proposed strategy finds k_0 clusters, in this case named microclusters, through an iterative process that aims at maximizing the quality of the microclusters

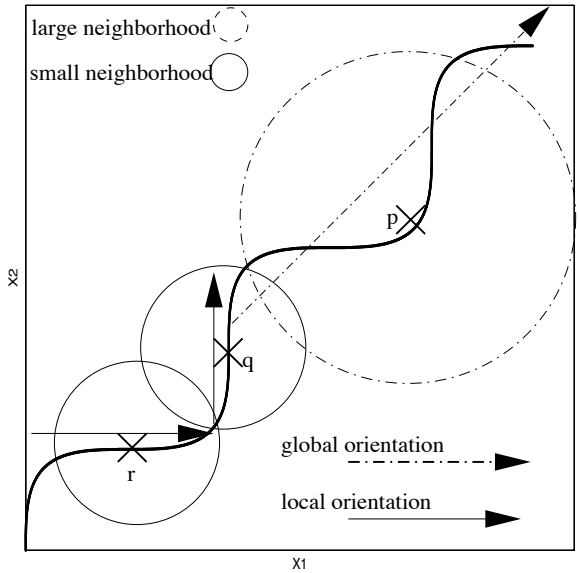


Figure 3.3: Examples of global and local orientation in a toy 2-dimensional database [Tung et al., 2005].

found. The value of k_0 is user-defined and the microclusters found are not necessarily disjoint. Quality is measured based on the adequacy of the microclusters found with regard to a predefined clustering model. The clustering model used is a Gaussian mixture model in which each microcluster M_i is represented by a probability distribution with density parameters $\theta_i = \{\mu_i, \sum_i\}$, where μ_i and $\sum_i$ represent respectively the centroid and the covariance matrix of the data elements that belong to M_i. A vector W is also defined, where each value W_i represents the fraction of the database that belongs to each microcluster M_i.

The iterative process starts with the creation of k_0 microclusters using identity matrices and random vectors to represent their covariance matrices and centroids respectively. The vector W is initially built by setting $W_i = \frac{1}{k_0}$ for each microcluster M_i created. Similarly to any method based on the Expectation/Maximization (EM) technique, the iterative process has two steps per iteration: the *Expectation step* and the *Maximization step*. The first one computes the probabilities of each data element to belong to each microcluster. In the second step, the density parameters of each microcluster are updated. The process stops when, after one iteration, the increase in the clustering quality is smaller than the parameter $^{\epsilon}likelihood$ or the maximum number of iterations $MaxLoopNum$ is reached. The values for $^{\epsilon}likelihood$ and $MaxLoopNum$ are user-defined. The clustering quality obtained after one iteration is computed by $E(\theta_1, \theta_2, \ldots \theta_{k_0} \mid D) = \sum_{x \in D} log[\sum_{i=1}^{k_0} W_i.PR(M_i|x)]$, where x is an element of the database D and $PR(M_i|x)$ is the probability of an element x to belong to microcluster M_i. The final result of the iterative process is a set of k_0 microclusters, defined by their corresponding

centroids and covariance matrices, as well as the probabilities of each data element to belong to each microcluster.

Once the iterative process is finished, the resulting microclusters are analyzed to define the ones that should be merged. For this task, CURLER proposes to use a novel similarity measure for pairs of microclusters, named *co-sharing level*. It is computed by the function $coshare(M_i, M_j) = \sum_{x \in D} [PR(M_i|x) . PR(M_j|x)]$, where x is an element of the database D, $PR(M_i|x)$ is the probability of element x to belong to microcluster M_i and $PR(M_j|x)$ is the probability of element x to belong to the microcluster M_j. Based on the $coshare(\ )$ function, a recursive process merges pairs of microclusters to form clusters, including merged clusters in further mergers. Specifically, a pair of microclusters (M_i, M_j) is merged if and only if $coshare(M_i, M_j) \geq {}^{\epsilon}coshare$, where the value of ${}^{\epsilon}coshare$ is user-defined.

After the merging step is completed, the resulting clusters are refined in one semi-automatic process that involves their visualization. This step is motivated by the fact that the automatic process may tend to spot only the global orientations of the clusters and the visualization step allows the user to find clusters that need to be further explored. We shall omit the details of this process here, as this Doctoral dissertation focus on automatic knowledge discovery methods only.

The main shortcomings of CURLER are: (i) the process is not fully automated; (ii) the method disregards the possible existence of outliers in the data, which is considered to be a drawback, since traditional methods for outlier detection do not analyze subspaces of the original space; (iii) there is no guarantee that the iterative process will converge in feasible time for all cases. Notice that the user-defined maximum number of iterations does not actually solves the problem; and (iv) the automatic part of the process has a quadratic time complexity regarding the number of microclusters and their dimensionalities, and the complexity is cubic with respect to the data dimensionality.

3.6 Conclusions

In this chapter we analyzed some of the well-known methods found in literature that search for clusters in multi-dimensional data with more than five or so dimensions. This analysis aimed at identifying the strategies already used to tackle the problem, as well as the major limitations of the existing, state-of-the-art techniques. Specifically, several well-known methods were briefly described and four of the most relevant ones were detailed. A recent survey on this area is found in [Kriegel et al., 2009].

We conclude the chapter by summarizing in Table 3.1 some of the most relevant techniques with regard to the main desirable properties that any clustering technique for moderate-to-high dimensionality data should have. Specifically, Table 3.1 uses checkmarks to link techniques with their desirable properties in order to help one to evaluate and to

compare the techniques listed. That table was partially[1] obtained from [Kriegel et al., 2009]. In general, the more checkmarks an algorithm features, the better and/or the more general is the algorithm. The properties considered in the table are described as follows:

- **Arbitrarily oriented clusters:** the ability to spot clusters that exist in subspaces formed by linear combinations of the original dimensions, besides the ones existing in subspaces that refer to subsets of the original dimensions;
- **Not relying on the** *locality assumption***:** the ability to be independent of the *locality assumption* [Kriegel et al., 2009], i.e., not to assume that the analysis of the space with all dimensions is sufficient to identify patterns that lead to clusters that only exist in subspaces;
- **Adaptive density threshold:** the ability to be independent of a fixed density threshold, i.e., not to assume high-dimensional clusters to be as dense as low-dimensional ones;
- **Independent wrt the order of the attributes:** the ability to generate the exact same clustering result for a given dataset, regardless of the order of its attributes;
- **Independent wrt the order of the objects:** the ability to generate the exact same clustering result for a given dataset, regardless of the order of its objects;
- **Deterministic:** the ability generate the exact same clustering result for a given dataset every time that the algorithm is executed;
- **Arbitrary number of clusters:** the ability to automatically identify the number of clusters existing in the input dataset, i.e., not to take the number of clusters as an user-defined input parameter;
- **Overlapping clusters (soft clustering):** the ability to identify data objects that belong to two or more overlapping clusters;
- **Arbitrary subspace dimensionality:** the ability to automatically spot clusters in subspaces of distinct dimensionalities in a single run of the algorithm, without taking the dimensionality of the clusters as an user-defined input parameter;
- **Avoiding complete enumeration:** the ability to avoid the analysis of all possible subspaces of a d-dimensional space, even for the worst case scenario;
- **Robust to noise:** the ability to obtain accurate clustering results from noisy data;

[1] Table 3.1 includes a summary of the table found in [Kriegel et al., 2009], i.e., Table 3.1 includes a selection of most relevant desirable properties and most closely related works from the original table. Table 3.1 also includes two novel desirable properties not found in [Kriegel et al., 2009] – **Linear or quasi-linear complexity** and **Terabyte-scale data analysis**.

- **Linear or quasi-linear complexity:** the ability to scale linearly or quasi-linearly in terms of memory requirement and execution time with regard to increasing numbers of points and axes, besides increasing clusters' dimensionalities;
- **Terabyte-scale data analysis:** the ability to handle datasets of Terabyte-scale in feasible time.

Clustering Algorithm	Arbitrarily oriented clusters	Not relying on the *locality assumption*	Adaptive density threshold	Independent wrt the order of the attributes	Independent wrt the order of the objects	Deterministic	Arbitrary number of clusters	Overlapping clusters (soft clustering)	Arbitrary subspace dimensionality	Avoiding complete enumeration	Robust to noise	Linear or quasi-linear complexity	Terabyte-scale data analysis
Axes parallel clustering													
CLIQUE [Agrawal et al., 1998, 2005]		✓		✓	✓	✓	✓	✓	✓		✓		
ENCLUS [Cheng et al., 1999]		✓		✓	✓	✓	✓	✓	✓		✓		
SUBCLU [Kröger et al., 2004]		✓		✓	✓	✓	✓	✓	✓		✓		
PROCLUS [Aggarwal et al., 1999]			✓							✓			
PreDeCon [Bohm et al., 2004]				✓	✓	✓	✓			✓	✓		
P3C [Moise et al., 2006, 2008]		✓	✓	✓	✓	✓	✓	✓	✓		✓		
COSA [Friedman and Meulman, 2004]				✓	✓	✓			✓	✓	✓		
DOC/FASTDOC [Procopiuc et al., 2002]		✓		✓	✓		✓	✓	✓				
FIRES [Kriegel et al., 2005]		✓	✓		✓	✓	✓	✓	✓	✓	✓		
Correlation clustering													
ORCLUS [Aggarwal and Yu, 2002, 2000]	✓			✓						✓			
4C [Böhm et al., 2004]	✓			✓	✓	✓	✓			✓	✓		
COPAC [Achtert et al., 2007]	✓			✓	✓	✓	✓		✓	✓	✓		
CASH [Achtert et al., 2008]	✓	✓	*n a*		✓	✓	✓		✓		✓		

Table 3.1: Properties of clustering algorithms well-suited to analyze moderate-to-high dimensional data. The table was partially obtained from [Kriegel et al., 2009]. *n a*: not applicable.

In spite of the several qualities found in the existing works, the analysis of the literature summarized in Table 3.1 led us to identify one important problem. To the best of our knowledge, among the methods published in the literature and designed to look for clusters in subspaces, no one has *any* of the following desirable properties:

- **Linear or quasi-linear complexity;**
- **Terabyte-scale data analysis.**

A complete justification for this statement is found in the descriptions provided during this chapter and also in the previous Chapter 2. Especially, please note that the existing techniques are unable to cluster datasets of Terabyte-scale in feasible time mainly because they propose serial processing strategies, besides the fact that they do not provide linear or quasi-linear scalability. Details on this topic are found in Section 2.5 from Chapter 2.

The next three chapters contain the central part of this Doctoral dissertation, which includes our contributions aimed at tackling the aforementioned problem. Specifically, we present the three knowledge discovery methods developed during this Ph.D. work.

Chapter

4

Halite

In the previous chapter, we provide a brief description of representative methods for clustering moderate-to-high-dimensional data, and summarize our analysis of the literature in Table 3.1. It allows us to identify two main desirable properties that are still missing from the existing techniques – (i) **Linear or quasi-linear complexity**, and; (ii) **Terabyte-scale data analysis**. Here we focus on tackling the former problem. Specifically, this chapter presents the new **method *Halite* for correlation clustering**, the first knowledge discovery algorithm designed in this Ph.D. work. *Halite* is a novel *correlation clustering* method for multi-dimensional data, whose main strengths are that it is fast and scalable with regard to increasing numbers of objects and axes, besides increasing dimensionalities of the clusters. The following sections describe our new method in detail.

4.1 Introduction

The **method *Halite* for correlation clustering** is a fast and scalable algorithm that looks for *correlation clusters* in multi-dimensional data using a *top-down*, multi-resolution strategy. It analyzes the distribution of points in the space with all dimensions by performing a multi-resolution, recursive partitioning of that space, which helps distinguishing clusters covering regions with varying sizes, density, correlated axes and number of points. Existing methods are typically super-linear in either space or execution time. *Halite* is fast and scalable, and gives highly accurate results. In details, the main contributions of *Halite* are:

1. **Scalability:** it is linear in running time and in memory usage with regard to the data size and to the dimensionality of the subspaces where clusters exist. *Halite* is

also linear in memory usage and quasi-linear in running time regarding the space dimensionality;

2. **Usability:** it is deterministic, robust to noise, does not have the number of clusters as a parameter and finds clusters in subspaces generated by the original axes or by their linear combinations, including space rotation;

3. **Effectiveness:** it is accurate, providing results with equal or better quality compared to top related works;

4. **Generality:** it includes a soft clustering approach, which allows points to be part of two or more clusters that overlap in the data space.

The proposed method is named after the mineral Halite. Halite, or rock salt, is the mineral form of sodium chloride ($NaCl$). One specimen of naturally crystallized Halite is shown in Figure 4.1[1]. Halite forms isometric crystals, i.e., crystal systems of any shape and size formed from the union of overlapping, rectangular unit cells. In this chapter, we propose to generalize the structure of these systems to the d-dimensional case in order to describe correlation clusters of any shape and size, hence the name of our method *Halite*.

Figure 4.1: Example of an isometric crystal system commonly found in the nature – one specimen of the mineral Halite that was naturally crystallized.

The method *Halite* uses spatial convolution masks in a novel way to efficiently detect density variations in a multi-scale grid structure that represents the input data, thus

[1] The image is publicly available at 'http://en.wikipedia.org/wiki/Halite'. Notice that this specimen was obtained from the Stassfurt Potash deposit, Saxony-Anhalt, Germany.

spotting clusters. These masks are extensively used in digital image processing to detect patterns in images. However, to the best of our knowledge, this is the first work to apply such masks over data in five or more axes. *Halite* also uses the Minimum Description Length (MDL) principle in a novel way. The main idea is to encode an input dataset, selecting a minimal code length. Specifically, *Halite* uses MDL to automatically tune a density threshold with regard to the data distribution, which helps spotting the clusters' subspaces. Finally, *Halite* includes a compression-based analysis to spot points that most likely belong to two or more clusters that overlap in the space. It allows soft clustering results, i.e., points can be part of two or more overlapping clusters.

We present a theoretical study on the time and space complexity of *Halite* and report an extensive experimental evaluation performed over synthetic and real data spanning up to 1 *million* elements to corroborate the desirable properties of our method regarding its Scalability, Usability, Effectiveness and Generality. Specifically, we report experiments comparing *Halite* with seven representative works. On synthetic data, *Halite* was consistently the fastest method, always presenting highly accurate results. Regarding real data, *Halite* analyzed 25-dimensional data for breast cancer diagnosis (KDD Cup 2008) at least 11 times faster than five previous works, increasing their accuracy in up to 35, while the last two related works failed. Details are found in the upcoming Section 4.7.

Remark: *Halite* is well-suited to analyze data in the range of 5 to 30 axes. In our experience, the intrinsic dimensionalities of real datasets are frequently smaller than 30, mainly due to the existence of several global correlations [Traina Jr. et al., 2002]. Therefore, if one dataset has more than 30 or so axes, it is possible to apply some distance preserving dimensionality reduction or feature selection algorithm to remove the global correlations, such as PCA or FDR, and then apply *Halite* to treat the correlations local to specific data clusters.

4.2 General Proposal

In this chapter, we tackle the problem of correlation clustering based on the following fact: clusters of *any* shape, existing only in subspaces of a d-dimensional space, create bumps (spikes) in the point density of their respective subspaces, but, commonly, these bumps *still exist* in the space with all dimensions, besides being weakened or diluted by the dimensions that do not belong to the respective clusters' subspaces. To find the clusters, in the same way as most related works do (see Chapter 3 for details), we assume what is known in literature as the *locality assumption*: one can still spot such diluted bumps by analyzing the space with all dimensions.

Assuming *locality*, and inspired by the structure of isometric crystal systems, such as the specimen of the mineral Halite shown in Figure 4.1, our general proposal for correlation clustering is twofold:

- **Bump Hunting:** to spot bumps in the point density of the space with all axes, defining each bump as a d-dimensional, axes-aligned hyper-rectangle that is considerably denser than its neighboring space regions (local neighborhood). We name each bump as a β-cluster;

- **Correlation Clustering:** to describe correlation clusters of *any* shape, existing in subspaces of a d-dimensional space, axis-aligned or not, as maximal sets of overlapping β-clusters.

Let us illustrate these concepts on the 3-dimensional toy datasets that were previously introduced in Section 2.3 from Chapter 2. Figure 4.2 reprints the same datasets in this chapter, this time *including* dotted rectangles into the illustration to represent β-clusters. Remember: the datasets contain clusters of several shapes (i.e., elliptical clusters, 'star-shaped' clusters and 'triangle-shaped' clusters) existing only in subspaces that are formed by subsets of the axes $\{x, y, z\}$, or by linear combinations of these axes, and we plot x-y and x-z projections of the toy data in the illustration. Notice that a single axes-aligned, 3-dimensional hyper-rectangle embodies each cluster in some cases, but certain clusters, especially the ones not aligned to the original axes, need more than one overlapping hyper-rectangle to be covered. Thus, maximal sets of overlapping β-clusters properly describe the clusters, even the ones not aligned to the axes $\{x, y, z\}$.

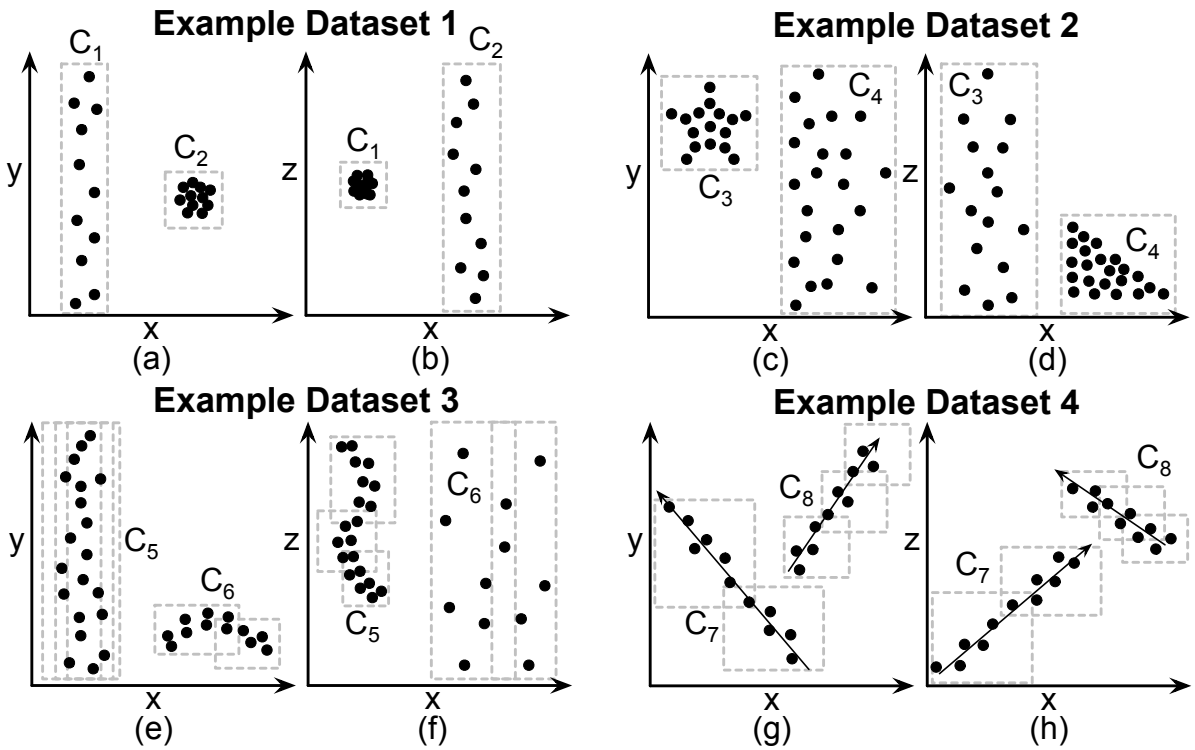


Figure 4.2: x-y and x-z projections of four 3-dimensional datasets over axes $\{x, y, z\}$. From (a) to (f): clusters in the subspaces $\{x, y\}$ and $\{x, z\}$. (g) and (h): clusters in subspaces formed by linear combinations of $\{x, y, z\}$.

According to the general proposal presented, we provide formal definitions for the β-clusters and for the correlation clusters.

Definition 7 *Let ${}^dS$ be a multi-dimensional dataset on the axes E. Then a* ***β-cluster*** *in ${}^dS$, ${}^{\delta}_{\beta}C_k = \langle {}_{\beta}E_k, {}^{\delta}_{\beta}S_k \rangle$ is a set ${}_{\beta}E_k \subseteq E$ of δ axes together with a set of points ${}^{\delta}_{\beta}S_k \subseteq {}^dS$ that, in a statistical sense, form one* **bump** *(one spike) in the point density of the d-dimensional space. The axes in ${}_{\beta}E_k$,* ***relevant to the β-cluster****, are the original axes that cause the bump, i.e., ${}^{\delta}_{\beta}S_k$ is densely clustered in the axes-aligned subspace formed by ${}_{\beta}E_k$. The axes in $E -_{\beta} E_k$ are* ***irrelevant to*** *${}^{\delta}_{\beta}C_k$. ${}^{\delta}_{\beta}S_k$ must be within an axes-aligned, d-dimensional hyper-rectangle, with the upper and lower bounds at each axis e_j being $U[k][j]$ and $L[k][j]$ respectively.*

Definition 8 $-$ **β-cluster overlapping:** *Given any two β-clusters ${}^{\delta'}_{\beta}C_{k'}$ and ${}^{\delta''}_{\beta}C_{k''}$, one can say that the β-clusters overlap to each other if $U[k'][j] \geq L[k''][j] \wedge L[k'][j] \leq U[k''][j]$ is valid for every original axis e_j.*

Definition 9 *Let ${}^dS$ be a multi-dimensional dataset on the axes E. Then a* ***correlation cluster*** *in ${}^dS$, ${}^{\delta}_{\gamma}C_k = \langle {}_{\gamma}E_k, {}^{\delta}_{\gamma}S_k \rangle$ is defined as one maximally connected component in the graph, whose nodes are the β-clusters that exist in ${}^dS$, and there is an edge between two nodes, if the respective β-clusters overlap. For the β-clusters referring to the nodes of this component, ${}^{\delta}_{\gamma}S_k = \bigcup {}^{\delta'}_{\beta}S_{k'}$ and ${}_{\gamma}E_k = \bigcup {}_{\beta}E_{k'}$. The axes in ${}_{\gamma}E_k$ are said to be* ***relevant to the cluster****, and the axes in $E -_{\gamma} E_k$ are* ***irrelevant to the cluster****. The cardinality $\delta = |{}_{\gamma}E_k|$ is the* ***dimensionality of the cluster****.*

Notice one *important* remark: as it can be seen in Definition 9, we assume that the axes irrelevant to a cluster ${}^{\delta}_{\gamma}C_k = \langle {}_{\gamma}E_k, {}^{\delta}_{\gamma}S_k \rangle$ are the original axes in which ${}^{\delta}_{\gamma}C_k$ spreads parallel to. The axes relevant to ${}^{\delta}_{\gamma}C_k$ are the remaining *original* axes. Thus, if ${}^{\delta}_{\gamma}C_k$ does *not* spread parallel to any original axis, *all* original axes are relevant to the cluster, i.e., ${}_{\gamma}E_k = E$. Finally, once a cluster ${}^{\delta}_{\gamma}C_k$ is found, and thus its corresponding local correlation was already detected, the **subspace** in which the cluster exists can be easily defined by using a traditional feature extraction method, such as PCA, applied over the points ${}^{\delta}_{\gamma}S_k$ projected onto the axes ${}_{\gamma}E_k$. ${}_{\gamma}E_k$ defines the subspace of a cluster only for axes-aligned clusters. As an example, cluster C_6 in Figure 4.2 spreads parallel to z, while both its relevant axes and subspace are given by the axes $\{x, y\}$. On the other hand, cluster C_8 in Figure 4.2 does not spread parallel to any original axis, thus its relevant axes are $\{x, y, z\}$. The subspace of C_8 is found by feature extraction applied only on the points of the cluster, projected into its relevant axes $\{x, y, z\}$.

4.3 Proposed Method – Basics

This section presents the basic implementation of our clustering method[2], which we shall refer to as $Halite_0$. We start with $Halite_0$ for clarity of description; in later sections, we present the additional improvements, that lead to our optimized, and finally proposed method *Halite*.

The main idea in our approach is to identify clusters based on the variation of the data density over the space in a multi-resolution way, dynamically changing the partitioning size of the analyzed regions. Multi-resolution is explored applying d-dimensional hyper-grids with cells of several side sizes over the data space and counting the points in each grid cell. Theoretically, the number of cells increases exponentially to the dimensionality as the cell size shrinks, so the grid sizes dividing each region are carefully chosen in a way that only the cells that hold at least one data element are stored, limiting this number to the dataset cardinality. The grid densities are stored in a quad-tree-like data structure, the Counting-tree, where each level represents the data as a hyper-grid in a specific resolution.

Spatial convolution masks are applied over each level of the Counting-tree, to identify bumps in the data distribution regarding each resolution. Applying the masks to the needed tree levels allows spotting clusters with different sizes. Given a tree level, $Halite_0$ applies a mask to find the regions of the space with all dimensions that refer to the largest changes in the point density. The regions found may indicate clusters that only exist in subspaces of the analyzed data space. The neighborhoods of these regions are analyzed to define if they stand out in the data in a statistical sense, thus indicating clusters. The axes in which the points in an analyzed neighborhood are close to each other are said to be relevant to the respective cluster, while the axes in which these points are sparsely distributed are said to be irrelevant to the cluster. The Minimum Description Length (MDL) principle is used in this process to automatically tune a threshold able to define relevant and irrelevant axes, based on the data distribution. The following subsections detail the method.

4.3.1 Building the Counting-tree

The first phase of $Halite_0$ builds the Counting-tree, representing a dataset ${}^dS$ with d axes and η points as a set of hyper-grids of d-dimensional cells in several resolutions. The tree root (level zero) corresponds to a hyper-cube embodying the full dataset. The next level divides the space into a set of 2^d hyper-cubes, each of which fulfilling a "hyper-quadrant" whose side size is equal to half the size of the previous level. The resulting hyper-cubes are divided again, generating the tree structure. Therefore, each level h of the Counting-tree

[2] A first implementation of $Halite_0$ was initially named as the method *MrCC* (after Multi-resolution Correlation Clustering) in [Cordeiro et al., 2010b], an earlier conference publication of the work presented in this chapter. Latter, it was renamed to $Halite_0$ for clarity, since several improvements on the basic implementation were included into a journal paper [Cordeiro et al., 2011a].

represents ${}^dS$ as a hyper-grid of d-dimensional cells of side size $\xi_h = 1/2^h, h = 0, 1, ..., H - 1$, where H is the number of resolutions. Each cell can either be refined in the next level or not, according to the presence or to the absence of points in the cell, so the Counting-tree can be unbalanced.

Without loss of generality, we assume in the following description that the dataset ${}^dS$ is in a unitary hyper-cube $[0, 1)^d$. Thus, we refer to each cell in the Counting-tree as a hyper-cube. However, for data not scaled between 0 and 1, as long as we know the minimum and the maximum bounds of the axes, the Counting-tree can still be built by dividing each axis in half, through a recursive procedure that creates cells referring to hyper-rectangles and all our proposed strategies still apply.

The structure of each tree cell is defined as $< loc, n, P[d], usedCell, ptr >$, where loc is the cell spatial position inside its parent cell, n is the number of points in the cell, $P[\,]$ is a d-dimensional array of half-space counts, $usedCell$ is a boolean flag and ptr is a pointer to the next tree level. The cell position loc locates the cell inside its parent cell. It is a binary number with d bits of the form $[bb \ldots b]$, where the j-bit sets the cell in the lower (0) or upper (1) half of axis e_j relative to its immediate parent cell. Each half-space count $P[j]$ counts the number of points in the lower half of the cell with regard to axis e_j. The flag $usedCell$ determines whether or not the cell's density of points has already been analyzed in the clustering procedure. This analysis occurs only in the second phase of $Halite_0$, and thus, every $usedCell$ flag is initially set to $false$. Figure 4.3a shows 2-dimensional grids in up to five resolutions, while Figures 4.3b and 4.3c respectively illustrate a grid over a 2-dimensional dataset in four resolutions and the respective Counting-tree. The $usedCell$ flags are not shown to reduce cluttering in the figure.

In a Counting-tree, a given cell a at level h is referred to as a_h. The immediate parent of a_h is a_{h-1} and so on. The cell position loc of a_h corresponds to the relative position regarding its immediate parent cell, and it is referred to as $a_h.loc$. The parent cell is at relative position $a_{h-1}.loc$ and so on. Hence, the absolute position of cell a_h is obtained by following the sequence of positions $[a_1.loc \downarrow a_2.loc \downarrow \ldots \downarrow a_{h-1}.loc \downarrow a_h.loc]$. For example, the cell marked as A_2 in level 2 of Figure 4.3c has relative position $A_2.loc = [00]$ and absolute position up to level 2 as $[11 \downarrow 00]$. A similar notation is used to refer to the other cells' attributes, n, $P[\,]$, $usedCell$ and ptr. As an example, given the cell a_h, its number of points is $a_h.n$, the number of points in its parent cell is $a_{h-1}.n$ and so on.

The Counting-tree is created in main memory, and an efficient implementation to deal with the case when memory is not enough is presented in Section 4.4. Each tree node is implemented as a linked list of cells. Thus, although the number of regions to divide the space grows exponentially at $O(2^{dH})$, we only store the regions where there is at least one point and each tree level has in fact at most η cells. However, without loss of generality, this section describes each node as an array of cells for clarity.

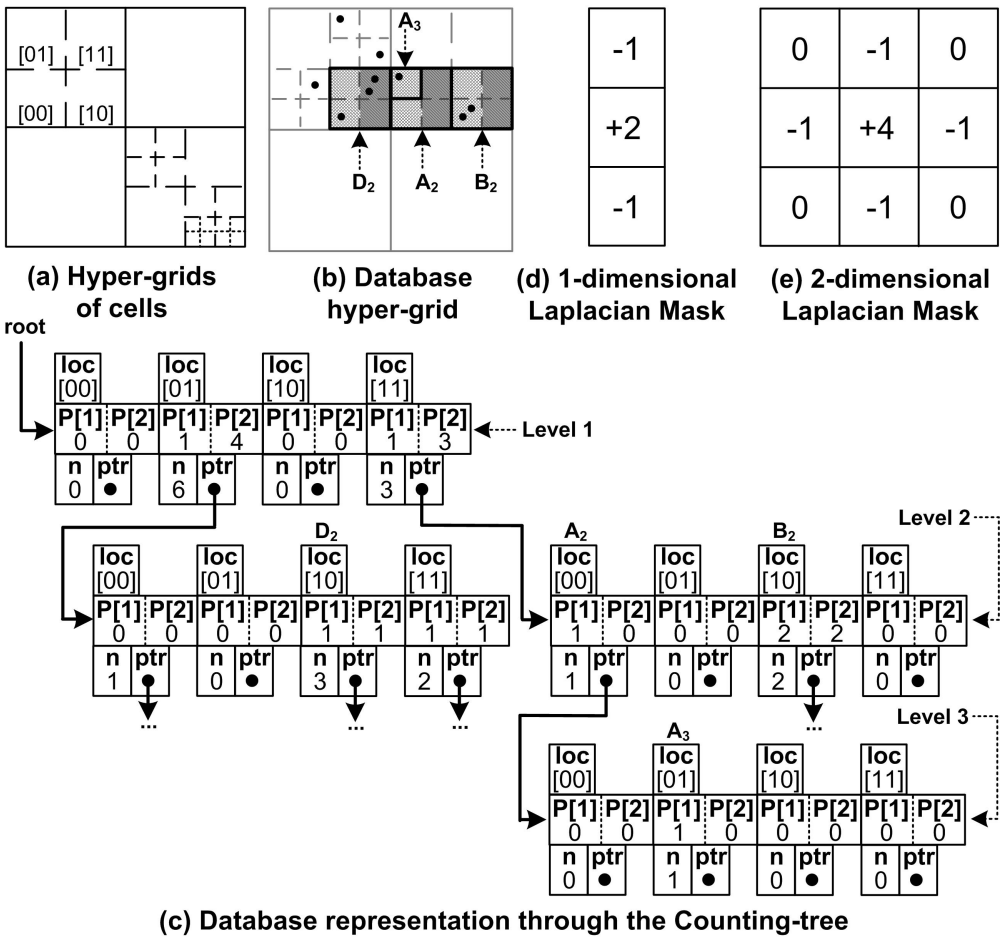


Figure 4.3: Examples of Laplacian Masks, 2-dimensional hyper-grid cells and the corresponding Counting-tree. Grids of several resolutions are applied over the dataset in (b) and the tree in (c) stores the resulting point counts. A convolution mask helps spotting correlation clusters.

Algorithm 1 shows how to build a Counting-tree. It receives the dataset normalized to a d-dimensional hyper-cube $[0, 1)^d$ and the desired number of resolutions H. This number must be greater than or equal to 3, as the tree contains $H - 1$ levels and at least 2 levels are required to look for clusters (see details in Algorithms 1 and 2). $Halite_0$ performs a single data scan, counting each point in every corresponding cell at each tree level as it is read. Each point is also counted in each half-space count $P[j]$ for every axis e_j when it is at the lower half of a cell in e_j.

Time and Space Complexity: Algorithm 1 reads each of the η data points once. When a point is read, it is counted in all the $H - 1$ tree levels, based on its position in all the d axes. Thus, the time complexity of Algorithm 1 is $O(\eta\ H\ d)$. The tree has $H - 1$ levels. Each level has at most η cells, which contain an array with d positions each. Thus, the space complexity of Algorithm 1 is $O(H\ \eta\ d)$.

Algorithm 1 : Building the Counting-tree.

Input: normalized dataset ${}^{d}S$,
number of distinct resolutions H
Output: Counting-tree T

```
1: for each point s_i ∈ ^dS do
2:    start at the root node;
3:    for h = 1, 2, ..., H − 1 do
4:       decide which hyper-grid cell in the current node of the Counting-tree covers s_i
         (let it be the cell a_h);
5:       a_h.n = a_h.n + 1;
6:       a_h.usedCell = false;
7:       if h > 1, update the half-space counts in a_{h−1};
8:       access the tree node pointed by a_h.ptr;
9:    end for
10:   update the half-space counts in a_h;
11: end for
```

4.3.2 Finding β-clusters

The second phase of *Halite*$_0$ uses the counts in the tree to spot bumps in the space with all axes that indicate β-clusters. A β-cluster ${}^{\delta}_{\beta}C_k = \langle {}_{\beta}E_k, {}^{\delta}_{\beta}S_k \rangle$ follows Definition 7. *Halite*$_0$ uses three matrices L, U and V to describe β-clusters. Let ${}_{\beta}k$ be the number of β-clusters found so far. Each matrix has ${}_{\beta}k$ lines and d columns, and the description of a β-cluster ${}^{\delta}_{\beta}C_k$ is in arrays $L[k]$, $U[k]$ and $V[k]$. $L[k]$ and $U[k]$ respectively store the lower and the upper bounds of β-cluster ${}^{\delta}_{\beta}C_k$ in each axis, while $V[k]$ has the value *true* in $V[k][j]$ if axis e_j belongs to ${}_{\beta}E_k$, and the value $false$ otherwise.

Halite$_0$ looks for β-clusters by applying convolution masks over each level of the Counting-tree. We name this task *"Bump Hunting"*. The masks are integer approximations of the Laplacian filter, a second-derivative operator that reacts to transitions in density. Figures 4.3d and 4.3e show examples of 1- and 2-dimensional Laplacian masks respectively. In a nutshell, the *"Bump Hunting"* task refers to: to apply for each level of the Counting-tree one d-dimensional Laplacian mask over the respective grid to spot bumps in the respective resolution. Figure 4.4a illustrates the process on a toy 1-dimensional dataset with grids in four resolutions. To spot bumps, for each cell of each resolution, the 1-dimensional mask from Figure 4.3d is applied as follows: multiply the count of points of the cell by the center value in the mask; multiply the point count of each neighbor of the cell by the respective mask value, and; get the convoluted value for the cell by summing the results of the multiplications. After visiting all cells in one resolution, the cell with the largest convoluted value represents the clearest bump in that resolution, i.e., the largest positive magnitude of the density gradient. In Figure 4.4a, for each resolution, one dark-gray arrow points to this cell.

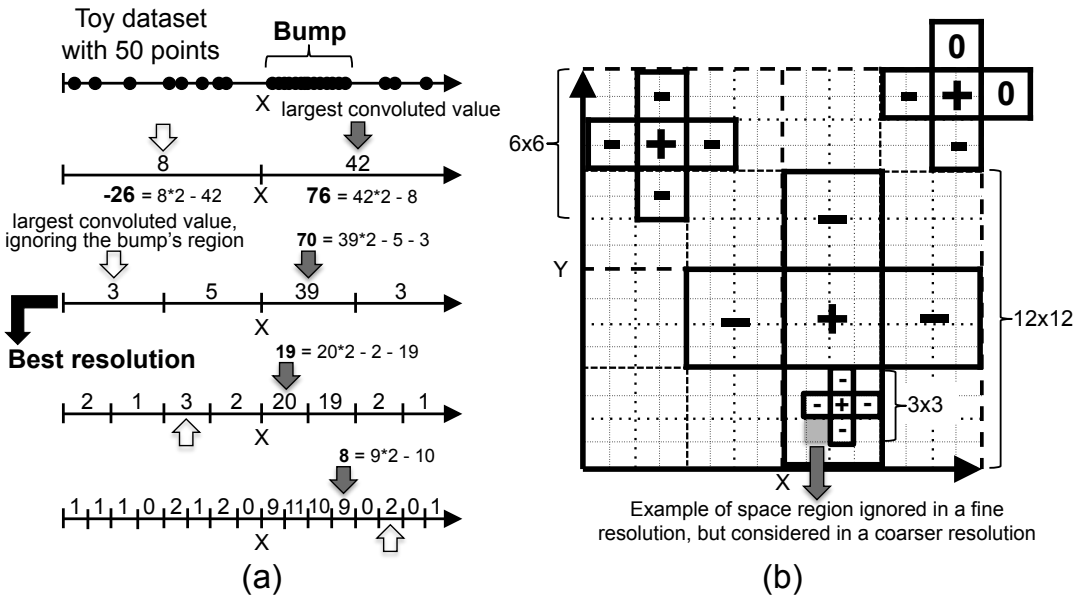


Figure 4.4: (a) one mask applied to 1-dimensional data. An statistical test finds the best resolution for a bump, and the time to stop spotting bumps; (b) intuition on the success of the used masks. The multi-resolution allows using 3^d masks to "simulate" larger masks, while the space regions ignored by the mask borders in a resolution tend to be considered in coarser resolutions.

For performance purposes, $Halite_0$ uses only masks of order 3, that is, matrices of sizes 3^d. In such masks, regardless of their dimensionality, there is always one center element (the convolution pivot), $2d$ center-faces elements (or just face elements, for short) and $3^d - 2d - 1$ corner elements. Applying a mask over all cells at a Counting-tree level can become prohibitively expensive in datasets with several dimensions – for example, a 10-dimensional cell has 59,028 corner elements. However, $Halite_0$ uses Laplacian masks having non-zero values only at the center and the facing elements, that is $2d$ for the center and -1 for the face elements, as in the examples in Figures 4.3d and 4.3e. A 10-dimensional cell has only 20 face elements. Therefore, it is possible to convolute each level of a Counting-tree with linear complexity regarding the dataset dimensionality d.

Notice one *important* remark: we chose to use masks of order 3, as they are the smallest available for convolution. Experimentally we found that the clustering accuracy of $Halite_0$ improves a little (at most 5 percent when applied over the datasets from the upcoming Section 4.7.1) when we use masks of order ϕ, $\phi > 3$, having non-zero values at all elements (center, face and corner elements), but the time required increases too much – in the order of $O(\phi^d)$ as compared to $O(d)$ when using masks of order 3 having non-zero values only at the center and the facing elements. Thus, we always use masks of order $\phi = 3$. To explain the success of the used masks, we point to a fact: the multi-resolution allows $Halite_0$ to use masks of a low order to efficiently "simulate" masks of higher orders. Figure 4.4b gives an intuition on this fact by illustrating masks applied to grids in distinct resolutions over a 2-dimensional space. As it can be seen, one mask of order 3 applied to a coarse resolution "simulates" masks of higher orders (i.e., $6, 12, ...$) applied to finer

resolutions. Furthermore, Figure 4.4b also shows that, in our multi-resolution setting, the space regions ignored in fine resolutions due to the mask's zero values are commonly considered in coarser resolutions.

Bump Hunting: As it can be seen in Algorithm 2, phase two starts applying the mask to level two of the Counting-tree, starting at a coarse resolution and refining as needed. It allows $Halite_0$ to find β-clusters of different sizes. When analyzing a resolution level h, the mask is convoluted over every cell at this level, excluding those already used for a β-cluster found before. The cell with the largest convoluted value refers to the clearest bump in the data space analyzed regarding level h, i.e., the largest positive magnitude of the density gradient. Thus, it may indicate a new β-cluster.

Applying a convolution mask to level h requires a partial walk over the Counting-tree, but no node deeper than h is visited. The walk starts going down from the root until reaching a cell b_h at level h that may need to be convoluted. The neighbors of this cell are named after the convolution matrix: face and corner neighbors. The center element corresponds to the cell itself. If the $b_h.usedCell$ flag is $true$ or the cell shares the data space with a previously found β-cluster, the cell is skipped. Otherwise, $Halite_0$ finds the face neighbors of b_h and applies the mask centered at b_h. After visiting all cells in level h, the cell with the largest convoluted value has the $usedCell$ flag set to $true$.

Each cell b_h at resolution level h is itself a d-dimensional hyper-cube and it can be divided into other 2^d cells in the next level, splitting each axis in half. Therefore, from the two face neighbors at axis e_j, one is stored in the same node of cell b_h, while the other is in a sibling node of cell b_h. We call the face neighbor stored at the same node as the internal neighbor of cell b_h, and the other as its external neighbor regarding axis e_j. For example, cell A_2 in resolution level 2 of the Counting-tree in Figure 4.3c has the internal neighbor B_2 and the external neighbor D_2 regarding axis e_1. The internal neighbor of a cell b_h in axis e_j at resolution level h and its point count are respectively referred as $NI(b_h, e_j)$ and $NI(b_h, e_j).n$. Similarly, for the same axis and resolution, the external neighbor of b_h and its points count are $NE(b_h, e_j)$ and $NE(b_h, e_j).n$ respectively. $Halite_0$ analyzes the absolute cell positions in a Counting-tree to spot external and internal neighbors.

Confirming the β-cluster: The *"Bump Hunting"* task allows to efficiently spot the clearest bumps in a dataset. However, two open questions still prevent its use for clustering: (i) *what is the best resolution to spot each bump?* and (ii) *when should the "Bump Hunting" stop?* To give an intuition on both questions, we use again the 1-dimensional data with grids in four resolutions from Figure 4.4a. In the illustration, a dark-gray arrow points to the cell with the largest convoluted value, which describes the clearest bump in the data for each resolution. Notice: we did not show yet a procedure to automatically identify the best resolution to spot a bump, which refers to question one. In the example, the bump is best described in the second coarsest resolution, as its bounds are overestimated in the coarsest resolution and only its borders are spotted in

Algorithm 2 : Finding the β-clusters.

Input: Counting-tree T, significance level α
Output: matrices of β-clusters L, U and V,
number of β-clusters ${}_\beta k$

1: ${}_\beta k = 0$;
2: **repeat**
3: $h = 1$;
4: **repeat**
5: $h = h + 1$;
6: **for** each cell b_h in level h of T **do**
7: **if** $b_h.usedCell = false \quad \wedge \quad b_h$ does not overlap with a previously found β-cluster **then**
8: find the face neighbors of b_h in all axes;
9: apply the Laplacian mask centered in b_h, using the point counts of b_h and of the found neighbors;
10: $a_h = b_h$, if the resulting convoluted value is the largest one found in the current iteration;
11: **end if**
12: **end for**
13: $a_h.usedCell = true$;
14: centered on a_h and based on α, compute cP_j, nP_j and θ_j^α for every axis e_j;
15: **if** $cP_j > \theta_j^\alpha$ for at least one axis e_j **then**
16: ${}_\beta k =_\beta k + 1$; {a new β-cluster was found}
17: **end if**
18: **until** a new β-cluster is found $\quad \vee \quad h = H - 1$
19: **if** a new β-cluster was found **then**
20: compute $r[\,]$ and $cThreshold$;
21: **for** $j = 1, 2, ..., d$ **do**
22: $V[{}_\beta k][j] = r\,[j] \geq cThreshold$;
23: **if** $V[{}_\beta k][j] = true$ **then**
24: compute $L[{}_\beta k][j]$ and $U[{}_\beta k][j]$ based on the lower and upper bounds of a_h and of its face neighbors regarding axis e_j;
25: **else** $L[{}_\beta k][j] = 0$; $U[{}_\beta k][j] = 1$;
26: **end if**
27: **end for**
28: **end if**
29: **until** no new β-cluster was found

the two finest resolutions. Now, we refer to question two. Assuming that the exemplified bump was properly described in its best resolution, should we keep looking for bumps or should we stop? A procedure to automatically answer this question, after spotting each bump, is also missing. In the example, ignoring the bump's region, a white arrow points to the cell with the largest convoluted value for each resolution, which, clearly, does not lead to a cluster. Thus, the *"Bump Hunting"* should stop.

Notice that the *"Bump Hunting"* spots cells on the "positive" side of the largest, local density changes, but it does not guarantee that these changes are statistically significant - some of them could have been created by chance. Even when analyzing only points randomly distributed through a d-dimensional space, the mask will return bumps. Therefore, we propose to automatically answer both questions previously posed by ignoring bumps that, potentially, were created by chance, assuming that only statistically significant bumps lead to β-clusters. This strategy allows $Halite_0$ to identify the best resolution to spot each bump, since this resolution is the one in which the corresponding local density change is more intense, thus, it avoids overestimating the clusters bounds and spotting only clusters' borders. The proposed strategy also spots the correct moment to stop the *"Bump Hunting"* - it stops once the clearest bump was potentially created by chance in all resolutions.

To state if a new β-cluster exists in level h, $Halite_0$ searches for the cell a_h with the largest convoluted value and analyzes its neighbors. The intuition is to use an statistical test to verify if the possible cluster, represented by cell a_h, is *significantly* denser than its neighboring data space regions, thus confirming or not the cluster existence. The Counting-tree structure gives us a single feasible option for the analysis of the neighboring regions: to analyze the data distribution in one predecessor of cell a_h and in the face neighbors of that predecessor. Our experiments reported in the upcoming Section 4.7 show that top clustering accuracy is obtained by choosing the first (direct) predecessor of a_h, cell a_{h-1}, which gives us *at most six* regions to analyze per axis. Thus, $Halite_0$ uses this option. Other options are: to analyze 12 regions per axis, by choosing predecessor a_{h-2}; to analyze 24 regions per axis, by choosing predecessor a_{h-3}; and so on. But, these options would force $Halite_0$ to require more than two levels in the tree, the minimum requirement when choosing a_{h-1}, which would make the method slower.

For axis e_j, the neighbor cells to be analyzed are the predecessor a_{h-1} of a_h, its internal neighbor $NI(a_{h-1}, e_j)$ and its external neighbor $NE(a_{h-1}, e_j)$. Together, they have $nP_j = a_{h-1}.n + NI(a_{h-1}, e_j).n + NE(a_{h-1}, e_j).n$ points. The half-space counts in these three cells show how the points are distributed in *at most six* consecutive, equal-sized regions in axis e_j, whose densities we shall analyze. The point count in the center region, the one containing a_h, is given by: $cP_j = a_{h-1}.P[j]$, if the j-bit in $a_h.loc$ is 0, or by $cP_j = a_{h-1}.n - a_{h-1}.P[j]$ otherwise. For example, wrt cell A_3 in Figure 4.3b and axis e_1, the six analyzed regions are presented in distinct texture, $cP_1 = 1$ and $nP_1 = 6$.

If at least one axis e_j of cell a_h has cP_j significantly greater than the expected average number of points $\frac{nP_j}{6}$, $Halite_0$ assumes that a new β-cluster was found. Thus, for each axis e_j, the null hypothesis test is applied to compute the probability that the central region contains cP_j points if nP_j points are uniformly distributed in the six analyzed regions. The critical value for the test is a threshold to which cP_j must be compared to determine whether or not it is statistically significant to reject the null hypothesis. The statistic significance is a user-defined probability α of wrongly rejecting the null hypothesis. For a one-sided test, the critical value θ_j^α is computed as $\alpha = Probability(cP_j \geq \theta_j^\alpha)$. The probability is computed assuming the Binomial distribution with the parameters nP_j and $\frac{1}{6}$, since $cP_j \sim Binomial(nP_j, \frac{1}{6})$, under the null hypothesis and $\frac{1}{6}$ is the probability that one point falls into the central region, when it is randomly assigned to one of the six analyzed regions. If $cP_j > \theta_j^\alpha$ for at least one axis e_j, we assume a_h to be the center cell of a new β-cluster and increment ${}_\beta k$. Otherwise, the next tree level is processed.

Describing the β-cluster: Once a new β-cluster was found, $Halite_0$ generates the relevances array $r = [r_1, r_2, \ldots r_d]$, where $r\,[j]$ is a real value in $(0, 100]$ representing the relevance of axis e_j regarding the β-cluster centered in a_h. The relevance $r\,[j]$ is given by $(100 * cP_j)/nP_j$. Then, $Halite_0$ automatically tunes a threshold to mark each axis as relevant or irrelevant to the β-cluster. The relevances in $r[\,]$ are sorted in ascending order into array $o = [o_1, o_2, \ldots o_d]$, which is analyzed to find the best cut position p, $1 \leq p \leq d$ that maximizes the homogeneity of values in the partitions of $o[\,]$, $[o_1, \ldots o_{p-1}]$ and $[o_p, \ldots o_d]$. The value $cThreshold = o[p]$ defines axis e_j as relevant or irrelevant, by setting $V[{}_\beta k][j] = true$ if $r\,[j] \geq cThreshold$, and false otherwise.

In order to identify the value of p, based on the MDL principle, $Halite_0$ analyzes the homogeneity of values in partitions of $o[\,]$, $[o_1, \ldots o_{p'-1}]$ and $[o_{p'}, \ldots o_d]$, for all possible cut positions p', integer values between 1 and d. The idea is to compress each possible partition, representing it by its mean value and the differences of each of its elements to the mean. A partition with high homogeneity tends to allow good compression, since its variance is small and small numbers need less bits to be represented than large ones do. Thus, the best cut position p is the one that creates the partitions that compress best.

For a given p', we compute the number of bits required to represent the respective compressed partitions of $o[\,]$ using the following equation.

$$size\,(p') \;=\; b\,(\mu_L) \;+ \sum_{1 \leq j < p'} b\,(o[j] - \mu_L) + \;\; b\,(\mu_R) \;\; + \sum_{p' \leq j \leq d} b\,(o[j] - \mu_R) \qquad (4.1)$$

In Equation 4.1, $b\,(\,)$ is a function that returns the number of bits required to represent the value received as input, μ_L is the mean of $[o_1, \ldots o_{p'-1}]$ and μ_R is the mean of $[o_{p'}, \ldots o_d]$. We assume $b\,(\mu_L) = 0$ for $p' = 1$. Minimizing $size(\,)$, through all possible cut positions p', integer values between 1 and d, leads $Halite_0$ to find the best cut position p.

The last step required to identify the new β-cluster finds its lower and upper bounds in each axis. These bounds are respectively set to $L[_\beta k][j] = 0$ and $U[_\beta k][j] = 1$ for every axis e_j, having $V[_\beta k][j] = false$. For the other axes, the relevant ones, these bounds are first set equal to the lower and upper bounds of a_h in these axes. Then, they are refined analyzing the neighbors of a_h. Considering a relevant axis e_j, if there exists a non-empty face neighbor of a_h whose lower bound is smaller than the lower bound of a_h, then $L[_\beta k][j]$ is decreased by $1/2^h$. In the same way, if there exists a non-empty face neighbor of a_h whose upper bound is bigger than the upper bound of a_h, then $U[_\beta k][j]$ is increased by $1/2^h$. In this way, $Halite_0$ completes the description of the β-cluster and restarts applying the mask from level two of the tree to find another β-cluster. The process stops when the mask is applied to every tree level and no new β-cluster is found.

Time and Space Complexity: Algorithm 2 identifies $_\beta k$ β-clusters. When looking for each β-cluster, at most $H - 2$ tree levels are analyzed, which have at most η cells each. For each tree level, the cells that do not belong to a previously found β-cluster are the convolution pivots to apply the mask. Finally, the neighborhood of the cell with the largest convoluted value is analyzed to find if it is the center of a new β-cluster. Thus, the time complexity of this part of Algorithm 2 (lines 3-18) is $O(_\beta k^2\ H^2\ \eta\ d)$. After finding each new β-cluster, the relevance level array with d real values in $(0, 100]$ is built in $O(d)$ time and sorted in $O(d\ log\ d)$ time, the method MDL is used in $O(d)$ time and the new β-cluster is described in $O(d\ H)$ time. Thus, the time complexity of this part of Algorithm 2 (lines 19-28) is $O(d\ _\beta k(log\ d + H))$. However, each iteration step of the first part of Algorithm 2 consumes a time t_1 that is much larger than the time t_2 consumed by each iteration step of the second part. Thus, the total time of Algorithm 2 is $O(_\beta k^2\ H^2\ \eta\ d)\ t_1 + O(d\ _\beta k(log\ d + H))\ t_2$. Given that t_1 and t_2 are constant values and $t_1 \gg t_2$, we argue that $Halite_0$ is quasi-linear in d and our experiments corroborate this claim. See Section 4.7 for details. The space complexity of Algorithm 2 is $O(d\ _\beta k + d + H)$, as it builds the matrices L, U and V and it uses arrays with either d or H positions each.

4.3.3 Building the Correlation Clusters

The final phase of $Halite_0$ builds $_\gamma k$ correlation clusters based on the β-clusters found before. According to Definition 9, $Halite_0$ analyzes pairs of β-clusters and those that overlap are merged into a single cluster, including merged clusters in further mergers. Algorithm 3 details this phase.

Time and Space Complexity: Algorithm 3 analyzes, at cost $d_\beta k^2$, all the $_\beta k$ β-clusters found before to identify and to merge, among all possible pairs, the ones that overlap in the d-dimensional space. Then, it defines, at cost $d_\gamma k\ _\beta k$, the axes relevant to each of the $_\gamma k$ merged clusters, based on the relevant axes of the $_\beta k$ β-clusters. Thus, the time complexity of Algorithm 3 is $O(d\ (_\beta k^2 +_\gamma k\ _\beta k))$. During the process, an array with $_\beta k$ positions links β-clusters to correlation clusters, while a matrix with $_\gamma k$ lines

Algorithm 3 : Building the correlation clusters.

Input: matrices of β-clusters L, U and V,
number of β-clusters ${}_{\beta}k$

Output: set of correlation clusters C,
number of correlation clusters ${}_{\gamma}k$

1: identify all pairs of β-clusters that overlap;
2: merge each pair of overlapping β-clusters into a single cluster, including merged clusters in further mergers;
3: define the points that belong to each merged cluster, as the ones that belong to at least one of its β-clusters;
4: define the dimensions relevant to each merged cluster, as those relevant to at least one of its β-clusters;
5: C = the merged clusters;
6: ${}_{\gamma}k$ = the number of merged clusters;

and d columns indicates the relevant axes to the clusters. Thus, the space complexity of Algorithm 3 is $O({}_{\beta}k + {}_{\gamma}k\ d)$.

4.4 Proposed Method – The Algorithm Halite

In this section we propose the ***Halite* method for correlation clustering**. It improves the basic $Halite_0$ by providing an optimized implementation strategy for the Counting-tree, even for the case when it does not fit in main memory. The $Halite_0$ algorithm has linear space complexity with regard to the number of points, axes and clusters. However, using the recommended configuration, the amount of memory required by it in our experiments (see Section 4.7.1 for details) varied between 25 and 50 percent of the data size, depending on the points distribution. Thus, for large datasets, the use of Operational System's disk cache may become a considerable bottleneck. In order to overcome this problem, *Halite* has a table-based implementation that never uses disk cache, regardless of the input dataset. Therefore, it allows us to efficiently analyze large amounts of data.

The idea is to represent the Counting-tree by tables stored in main memory and/or disk. Each table represents one tree level, by storing in key/value entries the data related to all non-empty cells of that level. Remember that $Halite_0$ uses cells with the structure $< loc, n, P[d], usedCell, ptr >$, where loc is the cell spatial position inside its parent cell, n is the number of points in the cell, $P[\,]$ is an array of half-space counts, $usedCell$ is a boolean flag and ptr is a pointer to the next tree level. For *Halite*, this cell structure was slightly modified. Here, the pointer ptr does not exist and loc has the **absolute** position for the cell. In each key/value pair, loc is the key, and the other attributes form the value.

Figure 4.5 exemplifies the data storage for *Halite*. The tables shown consist in a different way of storing the Counting-tree of Figure 4.3c. Both approaches represent the same data, the 2-dimensional dataset from Figure 4.3b, the one used in our examples of

Section 4.3. To reduce cluttering in the figure, the *usedCell* flags are not shown. Notice, as an example, that the cell A_3 from Figure 4.3b has a single point. This information is stored in both versions of the tree as $A_3.n = 1$. The data space position of A_3 is given by $A_3.loc = [11 \downarrow 00 \downarrow 01]$ in Figure 4.5. The same information is found in Figure 4.3c as $[A_1.loc \downarrow A_2.loc \downarrow A_3.loc] = [11 \downarrow 00 \downarrow 01]$. Finally, the half-space counts are represented by $A_3.P[1] = 1$ and $A_3.P[2] = 0$ in both data structures.

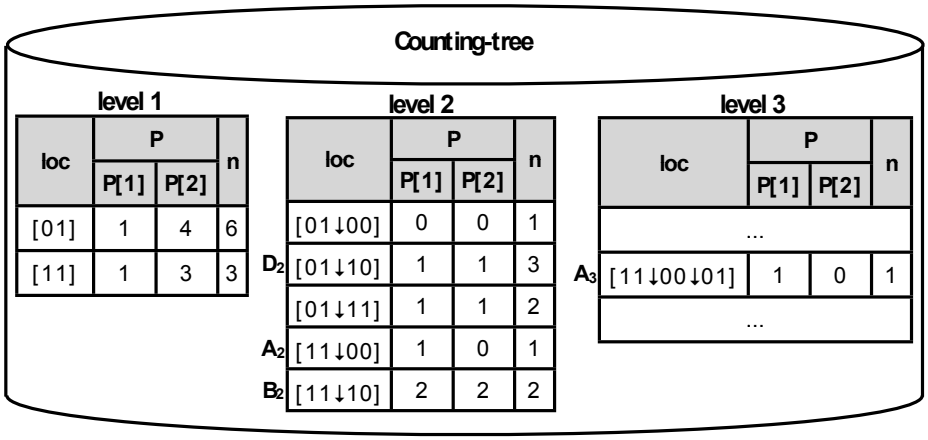


Figure 4.5: The Counting-tree by tables of key/value pairs in main memory and/or disk. It allows *Halite* to efficiently cluster large datasets, even when the tree does not fit in main memory. Notice that, both this tree and the one in Figure 4.3c represent our example dataset from Figure 4.3b.

Provided the $Halite_0$ algorithm and the fact that the tree can be stored in tables with key/value entries, the implementation of *Halite* is done as follows: we use a traditional approach to store tables with key/value entries in main memory and/or disk for the tree storage, and the same strategies used by $Halite_0$, described in Algorithms 1, 2 and 3, are applied to *Halite*. Notice that, between the used data structures, the Counting-tree is the only one that may have large changes in size with regard to the input dataset. Thus, considering our table-based implementation, it is possible to affirm that *Halite* never uses disk cache, regardless of the input dataset.

Our current implementation for *Halite* stores the Counting-tree by using the Oracle Berkeley DB 11g[3], configured for simple data storage to avoid data locks. The cache size is set according to the available main memory. Finally, we currently have hash tables storing the key/value pairs, but other structures could also be used, such as B-trees.

4.5 Proposed Method – Soft Clustering

The *Halite* algorithm is a hard clustering method, i.e., it defines a dataset partition by ensuring that each point belongs to at most one cluster. Hard clustering methods lead to high quality results for most datasets. Also, several applications require the definition of a

[3] www.oracle.com/technology/products/berkeley-db/

dataset partition. However, hard clustering is not the best solution for some specific cases, in which the clusters have high probability to overlap. Consider the 2-dimensional dataset in Figure 4.6a. The data contains a pair of clusters that overlap, making *any* dataset partition not a good choice, provided that the points in light-gray should belong to both clusters. In cases like that, the so-called soft clustering methods are more appropriate, since they allow points in the overlapping spaces to belong to more than one cluster. For that reason, this section proposes the ***Halite***$_s$ **method for soft correlation clustering**, a soft clustering approach for *Halite*.

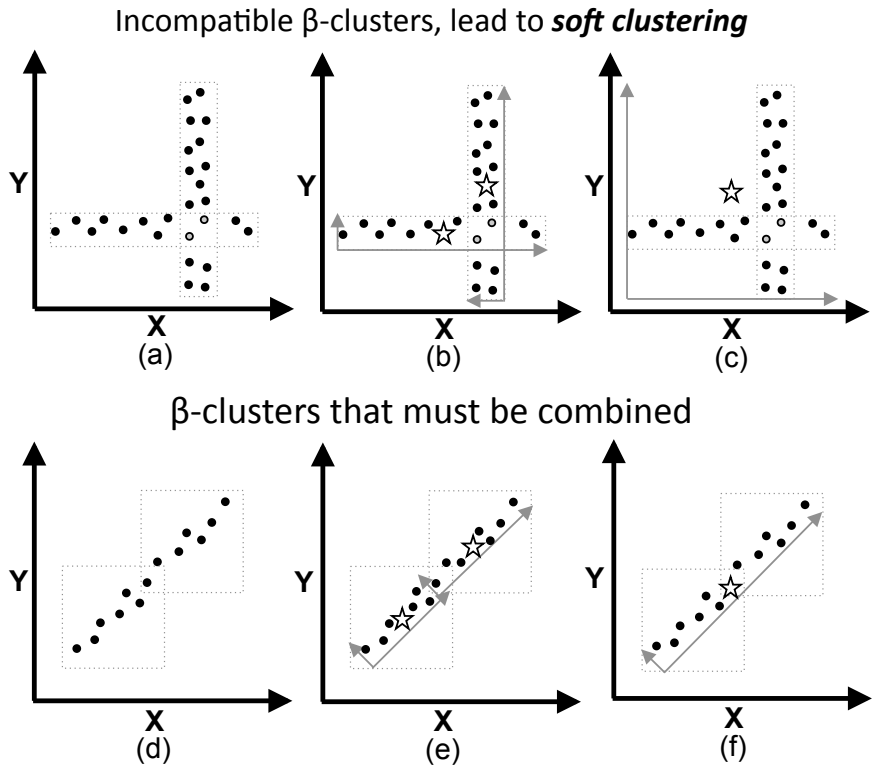


Figure 4.6: Illustration of our soft clustering method *Halite*$_s$: β-clusters (dotted rectangles) may stay apart (top) if they are incompatible, resulting in soft clustering (light-gray circles in (a)); or merged together (bottom). The compression-based formulas of *Halite*$_s$ automatically make the right choice. In all cases, a 'star' indicates the center of the respective clusters.

As a real example, let us consider the clustering analysis of satellite images. In this scenario, a topographer wants to analyze terrains in a set of images, usually assuming that each image is split into tiles (say, 32x32 pixels), from which features are extracted. The topographer expects to find clusters of 'water' tiles, 'concrete' tiles, 'forest' tiles, etc., but the used procedure may create many hybrid tiles, as a bridge (both 'water' and 'concrete') or a park ('water', 'forest' and 'concrete'), which should belong to more than one cluster. In other words, there is a high probability that the clusters overlap in the data space. Therefore, a hard clustering method is semantically inappropriate to the case.

Halite$_s$ is a fast and scalable algorithm carefully designed to spot points that should belong to two or more clusters, being recommended to scenarios where the probability of

cluster overlap is high, as in our previous example with satellite imagery. The algorithm has three phases. The first two are the same ones used for hard clustering: Algorithms 1 and 2, including the improvements presented in Section 4.4. The third phase is new. It takes β-clusters as input and uses a compression-based analysis to combine some of the ones that overlap into a soft clustering result. Figure 4.6 illustrates the problem. Distinct 2-dimensional datasets are shown in Figures 4.6a and 4.6d. Each dataset contains a pair of overlapping β-clusters, described by dotted rectangles. The β-clusters in Figure 4.6a clearly form two clusters and, thus they should remain apart. However, the ones in Figure 4.6d should be combined into a single cluster. *Halite*$_s$ automatically makes the right choice in both cases.

The full pseudo-code for the new phase three is shown in Algorithm 4. The idea is to use the Minimum Description Length (MDL) principle and to analyze the compression ratio of pairs of overlapping β-clusters, where the clusters in each pair can be compressed apart or combined. *Halite*$_s$ picks the option that compresses best. If the combined cluster allows better compression than the β-clusters do in separate, the β-clusters are merged; otherwise, the β-clusters remain apart, allowing points in the overlapping spaces to belong to both clusters.

Algorithm 4 : Building soft clusters.

Input: matrices of β-clusters L, U and V,
number of β-clusters ${}_\beta k$

Output: set of correlation clusters C,
number of correlation clusters ${}_\gamma k$

1: **for** $k' = 1, 2, ..., {}_\beta k$ **do**
2: **for** $k'' = k' + 1, k' + 2, ..., {}_\beta k$ **do**
3: **if** β-clusters ${}^{\delta'}_{\beta}C_{k'}$ and ${}^{\delta''}_{\beta}C_{k''}$ overlap **then**
4: $size_{k'} = compress({}^{\delta'}_{\beta}S_{k'})$, **if** not done yet;
5: $size_{k''} = compress({}^{\delta''}_{\beta}S_{k''})$, **if** not done yet;
6: $size_{k' \cup k''} = compress({}^{\delta'}_{\beta}S_{k'} \cup {}^{\delta''}_{\beta}S_{k''})$;
7: **if** $(size_{k'} + size_{k''}) > size_{k' \cup k''}$ **then**
8: // the combined cluster compresses best
9: merge β-clusters ${}^{\delta'}_{\beta}C_{k'}$ and ${}^{\delta''}_{\beta}C_{k''}$ into a single cluster, including merged clusters in further mergers, as it is done in Algorithm 3;
10: **end if**
11: **end if**
12: **end for**
13: **end for**

The strategy used for compression is described in Algorithm 5, which refers to the function *compress*( ) used in Algorithm 4. The function receives a set of points ${}^{\delta}_{\gamma}S_k$, related to a possible cluster ${}^{\delta}_{\gamma}C_k$ and returns the size of the input data in bits, after compression. Notice: in order to avoid additional disk accesses, ${}^{\delta}_{\gamma}S_k$ is approximated by the tree level where the respective β-clusters were found, assuming points in the center of

the cluster's cells. The compression is performed as follows. The principal components of ${}^{\delta}_{\gamma}S_k$ are computed, and the points of ${}^{\delta}_{\gamma}S_k$ are projected into **all** d principal components. Let the projected points be in a set ${}^{\delta}_{\gamma}P_k$ and the d principal components computed be in a set ${}_{\gamma}A_k$. Notice that this step states that ${}^{\delta}_{\gamma}P_k$ and ${}_{\gamma}A_k$ define the possible cluster ${}^{\delta}_{\gamma}C_k$ as ${}^{\delta}_{\gamma}C_k = \langle {}_{\gamma}A_k, {}^{\delta}_{\gamma}P_k \rangle$ after its projection into the principal components of ${}^{\delta}_{\gamma}S_k$. Then, the maximum max_j and minimum min_j values of ${}^{\delta}_{\gamma}P_k$ in each principal component $a_j \in {}_{\gamma}A_k$ are found. After that, we are able to represent the input data ${}^{\delta}_{\gamma}S_k$ in a compressed way by storing: the descriptions of each principal component a_j, besides their related values for max_j and min_j, and; the differences to the center, $p_{ij} - (((max_j - min_j)/2) + min_j)$, for each projected point $p_i \in {}^{\delta}_{\gamma}P_k$, with regard to each principal component $a_j \in {}_{\gamma}A_k$. The size in bits needed to store these items is the output. Large numbers need more bits to be stored than small numbers do. Thus, a set of points in which the stored differences to the center are small tend to lead to a good compression.

Algorithm 5 : Function *compress*( ).

Input: set of points ${}^{\delta}_{\gamma}S_k$ for a possible cluster ${}^{\delta}_{\gamma}C_k$
Output: compressed size for ${}^{\delta}_{\gamma}S_k$

1: compute the principal components of ${}^{\delta}_{\gamma}S_k$;
2: ${}^{\delta}_{\gamma}P_k =$ ${}^{\delta}_{\gamma}S_k$, projected into **all** d principal components computed;
3: ${}_{\gamma}A_k =$ the set of d principal components computed;
// Notice that Lines 2 and 3 state that ${}^{\delta}_{\gamma}P_k$ and ${}_{\gamma}A_k$ define the possible cluster ${}^{\delta}_{\gamma}C_k$ as ${}^{\delta}_{\gamma}C_k = \langle {}_{\gamma}A_k, {}^{\delta}_{\gamma}P_k \rangle$ after its projection into the principal components of ${}^{\delta}_{\gamma}S_k$.
4: max_j, min_j = the maximum and the minimum values of ${}^{\delta}_{\gamma}P_k$, in each principal component $a_j \in {}_{\gamma}A_k$;
5: $size$ = the number of bits needed to store the descriptions of each principal component $a_j \in {}_{\gamma}A_k$, besides its related values for max_j and min_j;
6: **for** every projected point $p_i \in {}^{\delta}_{\gamma}P_k$ **do**
7: **for** $j = 1, 2, ..., d$ **do**
8: // difference to the center wrt principal component $a_j \in {}_{\gamma}A_k$
9: $size = size +$ number of bits needed to store $(p_{ij} - (((max_j - min_j)/2) + min_j))$;
10: **end for**
11: **end for**
12: **return** size;

Let us illustrate how the compression idea works, by using the example datasets in Figure 4.6. Dotted rectangles, gray arrows and stars refer to β-clusters, principal components and cluster centers respectively. First, consider the data in Figure 4.6a. For the β-clusters apart, Figure 4.6b shows that we have 'medium compression', since the differences to the centers are small in one principal component of each cluster and large in the other. However, Figure 4.6c shows that the compression is worse for the combined cluster, since the differences to the center are large in both principal components. Thus, *Halite*$_s$ keeps the β-clusters apart. Notice that this step defines that the points in gray will belong to both clusters in the final result. For the data in Figure 4.6d, the Figures 4.6e

and 4.6f respectively show that the differences to the centers are small in one principal component and large in the other, both with the β-clusters in separate and combined. However, in the first case, we store the descriptions of two sets of principal components and the related values of max_j and min_j for each component. A single set is stored for the combined cluster, which leads to better compression. Therefore, $Halite_s$ decides to combine the β-clusters into a single correlation cluster.

4.6 Implementation Discussion

A possible bottleneck in our algorithms is related to computing the critical value θ_j^α for the statistical test, line 14 of Algorithm 2. As shown in Section 4.3, we carefully identify data space regions that refer to bumps in the point density and verify if these regions stand out in the data in a statistical sense, thus spotting clusters. The Binomial distribution $Binomial(n, p)$ is the base for our statistical test. But, computing the critical value with the exact Binomial distribution may become a bottleneck for large values of n. Fortunately, an efficient approximation to the $Binomial(n, p)$ is given by the Normal distribution $Normal(n.p,\ n.p.(1-p))$. Also, it is common sense in statistics that the approximation quality is excellent when $n.p > 5\ \wedge\ n.(1-p) > 5$. Thus, we compute θ_j^α for both *Halite* and $Halite_s$ using the normal approximation to the Binomial distribution whenever this rule applies. The exact computation is very efficient in all other cases.

4.7 Experimental Results

This section presents the experiments performed to test the algorithms proposed in this chapter. We intend to answer the following questions:

1. Compared with seven of the recent and related works, how good is the clustering method *Halite*?

2. How do the proposed techniques scale up?

3. How sensitive to the input parameters are our techniques?

4. What are the effects of soft clustering in data with high probability of cluster overlap and, compared with a well-known, soft clustering algorithm, how good is $Halite_s$?

All experiments were made in a machine with $8.00GB$ of RAM using a $2.33GHz$ core. Our methods were tuned with a fixed configuration in all experiments, i.e., default values for $\alpha = 1.0E-10$ and $H = 4$. The justification for this choice is in Section 4.7.3. Finally, notice that results on memory usage are not reported neither for *Halite* nor for $Halite_s$, since both allow the efficient use of data partially stored in disk, as described in Section

4.4. Note however that we do compare here the memory needs of the related works with those of *Halite*$_0$. Moreover, remember that *Halite*$_0$, *Halite* and *Halite*$_s$ have similar space complexity, the difference being that *Halite* and *Halite*$_s$ do manage the memory in disk if the Counting-tree does not fit in main memory.

4.7.1 Comparing hard clustering approaches

This section compares *Halite* with seven of the top related works over synthetic and real data. The techniques are: ORCLUS, COPAC, CFPC, HARP, LAC, EPCH and P3C. All methods were tuned to find disjoint clusters. The code of ORCLUS was kindly provided by Kevin Y. Yip and the project Biosphere. The source codes for all other methods were kindly provided by their original authors (i.e., Arthur Zimek and Elke Achtert provided COPAC; Man Lung Yiu and Nikos Mamoulis provided CFPC; Kevin Y. Yip provided HARP; Carlotta Domeniconi provided LAC; Raymond Chi-Wing Wong provided EPCH, and; Gabriela Moise provided P3C.). We also report results for *Halite*$_0$, which allows us to evaluate the improvements proposed on our basic algorithm.

Evaluating the Results: the quality of each result given by each technique was measured based on the well-known precision and recall measurements. We distinguish between the clusters known to exist in a dataset ${}^dS$, which we call real clusters, and those that a technique found, which we call found clusters. A real cluster ${}^{\delta}_{r}C_k = \langle {}_rA_k, {}^{\delta}_{r}S_k \rangle$ is defined as a set ${}_rA_k$ of δ axes, aligned or not to the original axes, together with a set of points ${}^{\delta}_{r}S_k$ densely clustered when projected into the subspace formed by ${}_rA_k$. Notice that the symbol A is used here in place of the symbol E to represent a set of axes, since the axes in ${}_rA_k$ can be original axes, but they can also be linear combinations of the original axes, i.e., ${}_rA_k$ is not necessarily a subset of E. A found cluster ${}^{\delta}_{f}C_k = \langle {}_fA_k, {}^{\delta}_{f}S_k \rangle$ follows the same structure of a real cluster, using the symbol f instead of r. Finally, ${}_fk$ and ${}_rk$ respectively refer to the numbers of found and real clusters existing in dataset ${}^dS$.

For each found cluster ${}^{\delta}_{f}C_k$, its most dominant real cluster ${}^{\delta'}_{r}C_{k'}$ is identified by the following equation.

$$ {}^{\delta'}_{r}C_{k'} \mid \ |{}^{\delta}_{f}S_k \cap {}^{\delta'}_{r}S_{k'}| = max(|{}^{\delta}_{f}S_k \cap {}^{\delta''}_{r}S_{k''}|),\ 1 \leq k'' \leq {}_rk \tag{4.2} $$

Similarly, for each real cluster ${}^{\delta'}_{r}C_{k'}$, its most dominant found cluster ${}^{\delta}_{f}C_k$ is identified by the equation as follows.

$$ {}^{\delta}_{f}C_k \mid \ |{}^{\delta'}_{r}S_{k'} \cap {}^{\delta}_{f}S_k| = max(|{}^{\delta'}_{r}S_{k'} \cap {}^{\delta''}_{f}S_{k''}|),\ 1 \leq k'' \leq {}_fk \tag{4.3} $$

The precision and the recall between a found cluster ${}^{\delta}_{f}C_k$ and a real cluster ${}^{\delta'}_{r}C_{k'}$ are computed as follows.

$$precision = \frac{\left|{}^{\delta}_{f}S_k \cap {}^{\delta'}_{r}S_{k'}\right|}{\left|{}^{\delta}_{f}S_k\right|} \tag{4.4}$$

$$recall = \frac{\left|{}^{\delta}_{f}S_k \cap {}^{\delta'}_{r}S_{k'}\right|}{\left|{}^{\delta'}_{r}S_{k'}\right|} \tag{4.5}$$

To evaluate the quality of a clustering result, we averaged the *precision* (Equation 4.4) for all found clusters and their respective most dominant real clusters. Also, we averaged the *recall* (Equation 4.5) for all real clusters and their respective most dominant found clusters. These two averaged values are closely related to well-known measurements. The first one is directly proportional to the dominant ratio [Aggarwal and Yu, 2002, Ng et al., 2005], while the second one is directly proportional to the coverage ratio [Ng et al., 2005]. We name as ***Quality*** the harmonic mean of these averaged values. The evaluation of a clustering result with regard to the quality of the subspaces uncovered is similar. We also computed the harmonic mean of the averaged *precision* for all found clusters and the averaged *recall* for all real clusters, but we exchanged the sets of points (S sets) in the two last equations, Equations 4.4 and 4.5, by sets of axes (A sets). We name this harmonic mean as ***Subspaces Quality***.

Finally, in the cases where a technique did not find clusters in a dataset, the value zero was assumed for both qualities.

System Configuration: *Halite* uses fixed input parameter values, as defined in the upcoming Section 4.7.3. $Halite_0$ was tuned in the same way. The other algorithms were tuned as follows. ORCLUS, LAC, EPCH, CFPC and HARP received as input the number of clusters present in each dataset. Also, the known percent of noise for each dataset was informed to HARP. The extra parameters of the previous works were tuned as in their original authors' instructions. LAC was tested with integer values from 1 to 11, for the parameter $1/h$. However, its run time differed considerably with distinct values of $1/h$. Thus, a time out of three hours was specified for LAC executions. All configurations that exceeded this time limit were interrupted. EPCH was tuned with integer values from 1 to 5 for the dimensionalities of its histograms and several real values varying from 0 to 1 were tried for the outliers threshold. For the tests in P3C, the values $1.0E-1$, $1.0E-2$, $1.0E-3$, $1.0E-4$, $1.0E-5$, $1.0E-7$, $1.0E-10$ and $1.0E-15$ were tried for the *Poisson threshold*. HARP was tested with the Conga line data structure. CFPC was tuned with values 5, 10, 15, 20, 25, 30 and 35 for w, values 0.05, 0.10, 0.15, 0.20 and 0.25 for α, values 0.15, 0.20, 0.25, 0.30 and 0.35 for β and 50 for *maxout*. ORCLUS was tested with its default values for $\alpha = 0.5$ and $k_0 = 15k$, where k is the known number of clusters present in each dataset. It also received as input the known average cluster dimensionality of each synthetic dataset, and all possible dimensionalities were tested for the real data. COPAC was tuned with its default values for $\alpha = 0.85$ and $k = 3d$ and its default distance function was used. Its parameter ϵ was defined as suggested in COPAC's

original publication and μ received the smallest value between k and the known size of the smallest cluster present in each dataset, since COPAC demands $\mu \leq k$.

Notice two remarks: (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 for the methods we compare with refer to the configurations that led to the best Quality value, over all possible parameters tuning.

Synthetic Data Generation: we created synthetic data following standard procedures used by most methods described in Chapter 3, including the tested methods. The details are in Algorithm 6. In a nutshell: (i) we initially created axes-aligned, elliptical clusters of random sizes that follow normal distributions with random means and random variances in at least 50% of the axes (relevant axes), spreading through at most 15% of these axes domains. In other axes, the irrelevant ones, all clusters follow the uniform distribution, spreading through the whole axes domains; and (ii) an optional data rotation allowed creating clusters not aligned to the original axes. In this step, each dataset was rotated four times in random planes and random degrees.

Algorithm 6 was used to create synthetic datasets organized in several groups. A first group of datasets was created to analyze each tested method with regard to increasing numbers of points, axes and clusters. It contains 7 non-rotated datasets with d, η and $_{\gamma}k$ growing together from 6 to 18, $12k$ to $120k$ and 2 to 17 respectively. Noise percentile was fixed at 15%. For identification purposes, the datasets are named $6d$, $8d$, $10d$, $12d$, $14d$, $16d$ and $18d$ according to their dimensionalities. Rotated versions of these datasets were also created to analyze each method over clusters not aligned to the original axes. These datasets are named $6d\_r$, $8d\_r$, $10d\_r$, $12d\_r$, $14d\_r$, $16d\_r$ and $18d\_r$.

The strategy employed to create the $14d$ dataset (14 axes, $90k$ points, 17 clusters, 15% noise and non-rotated) was the base for the creation of the other groups of datasets. Based on it, 3 groups of datasets were created varying one of these characteristics: numbers of points, axes or clusters. Each group has datasets, created by Algorithm 6, in which a single characteristic changes, while all others remain the same as the ones in the $14d$ dataset. The number of points grows from $50k$ to $250k$, the dimensionality grows from 5 to 30 and the number of clusters grows from 5 to 25. The names Xk, Xc and Xd_s refer respectively to datasets in the groups changing numbers of points, clusters or axes. For example, dataset $30d\_s$ differs from $14d$ only because it has 30 axes instead of 14.

Results on Synthetic Data: This section compares the methods on synthetic data. We show results for clustering quality, memory consumption and run time. For easy reading the graphs of the section, we linked the values obtained for each method and plotted all vertical axes related to run time or to memory consumption in *log scale*. When a method found no cluster in a dataset, despite it was ran several times for each possible parameter configuration, the respective values measured for run time and also for memory

Algorithm 6 : Function *generate_one_dataset*( ).

Input: dimensionality d,
cardinality η,
number of clusters ${}_{\gamma}k$,
percent of noise pN,
choice between axes-aligned and arbitrarily oriented clusters *rotate*

Output: one synthetic dataset ${}^{d}S$

```
1: // generates the noise points
   ^dS = η * (pN/100) random points in [0,1)^d;
2: // defines the sizes of the clusters
   c[ ] = array of _γk random integers, such that Σ_k c[k] = η * ((100 − pN)/100);
3: for k = 1, 2, ..., _γ k do
4:    // empty matrix (new cluster)
      define cluster[c[k]][d];
5:    // δ ≥ 50% of d
      δ = random integer in [d/2, d];
6:    // relevant axes, at least 50%
      randomly pick δ axes;
7:    for j = 1, 2, ..., d do
8:       if axis e_j was picked then
9:          // the cluster will spread through at most 15% of the domain of e_j
            choose random mean and variance, such that the Normal(mean, variance)
            distribution generates values in [0,1) that differ at most 0.15 from each other;
10:         cluster[:][j] = c[k] real numbers following Normal(mean, variance);
11:      else cluster[:][j] = c[k] random values in [0,1);
12:      end if
13:   end for
14:   insert each row of cluster[ ][ ] as a point in ^dS;
15: end for
16: if rotate then
17:   // optional step of rotation
      rotate ^dS four times in random planes and in random degrees, and then normalize
      ^dS to [0,1)^d;
18: end if
19: return ^dS;
```

consumption were ignored, as in most cases each run led to different values measured due to the distinct configurations used. When a method used disk cache, its run time was ignored too. In these cases, no lines link the values obtained for the respective methods in the graphs.

Figure 4.7 presents the results for run time and for clustering accuracy. Figure 4.7a shows that *Halite*, $Halite_0$, EPCH, HARP and LAC presented similar high Quality values for all the datasets in the first group. CFPC presented a clear decrease in Quality when the dimensionality was higher than 12. P3C, ORCLUS and COPAC had worse Quality results. Regarding run time, Figure 4.7b shows that *Halite* was in general the fastest algorithm, loosing by little to $Halite_0$ for small data. As an example, for the biggest dataset $18d$, *Halite* was respectively 6, 17, 54, 145, 515, $1,440$ and $10,255$ times faster than $Halite_0$, CFPC, EPCH, LAC, P3C, ORCLUS and COPAC.

The results for data with individual changes in numbers of points, axes and clusters are shown in Figure 4.7, from 4.7c up to 4.7h. Notice that, *Halite*, $Halite_0$, LAC and EPCH performed well in Quality for all cases, exchanging positions but being in general within 10 percent from each other with no one being prevalent. Notice also that *Halite* and $Halite_0$ always showed the same Quality. ORCLUS, COPAC, CFPC, HARP and P3C performed worse than that. *Halite* was again the fastest method in almost all cases, only tying with $Halite_0$ for low dimensional datasets. As an example, for the dataset with the highest dimensionality, $30d\_s$, *Halite* ran respectively 9, 25, 36, 419, $1,542$, $3,742$ and $5,891$ times faster than $Halite_0$, CFPC, LAC, P3C, ORCLUS, HARP and COPAC.

Another experiment refers to the rotated data. It analyzes each method's abilities to find clusters in subspaces formed by linear combinations of original axes. The results are in Figures 4.7i and 4.7j. *Halite*, $Halite_0$ and LAC were only marginally affected by rotation, varying at most 5% in their respective Quality values, compared to the results of the same data without rotation. All others had considerable decreased or increased Quality values for at least one case. Run time results were similar to those obtained for non-rotated data.

The results on memory usage are presented in Figure 4.8. We report results for P3C, LAC, EPCH, CFPC, HARP, and also for our method $Halite_0$. Figure 4.8 shows that, in all cases, there was a huge discrepancy between HARP and EPCH face to the others with regard to memory usage. Note the *log scale* in every Y-axis. As an example, for dataset $18d$, the biggest one into the first group of datasets, HARP used approximately $34.4GB$ of memory and EPCH used 7.7 percent of this amount, while $Halite_0$ used only 0.3 percent of the memory required by HARP.

The quality of relevant axes was also evaluated. LAC, COPAC and ORCLUS were not tested here, as they do not return a set of original axes to define the axes relevant to each cluster. The results for the first group of datasets are in Figure 4.9. The Subspaces

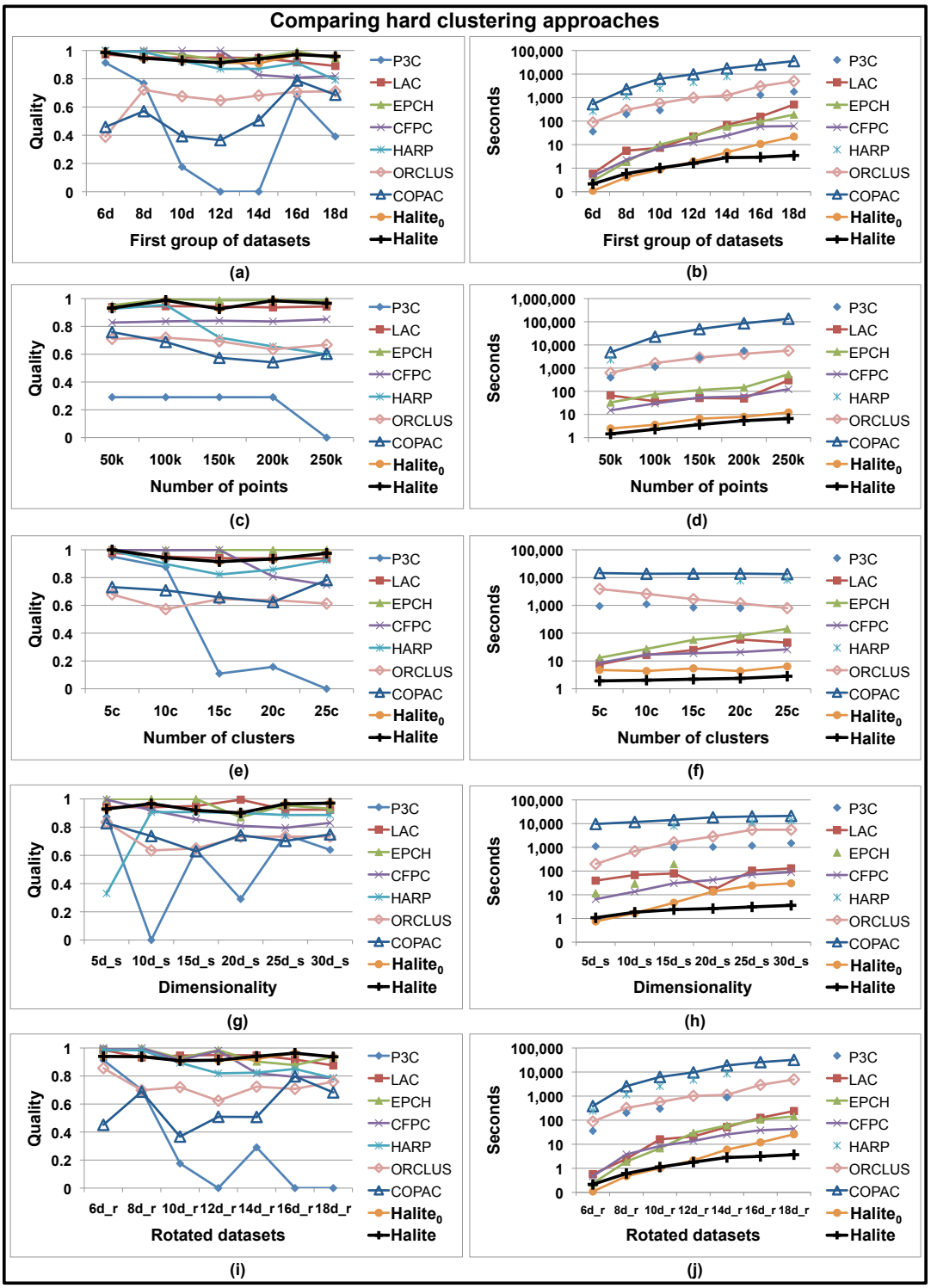


Figure 4.7: *Halite* is shown in black vertical crossing lines. Left column: Quality; Right column: wall-clock time in *log scale*. Comparison of approaches for hard clustering – *Halite* was in average at least 12 times faster than seven top related works, always providing high quality clusters.

Quality values are similar for *Halite*, $Halite_0$ and EPCH. All others had worse results. The same pattern was seen in the other datasets.

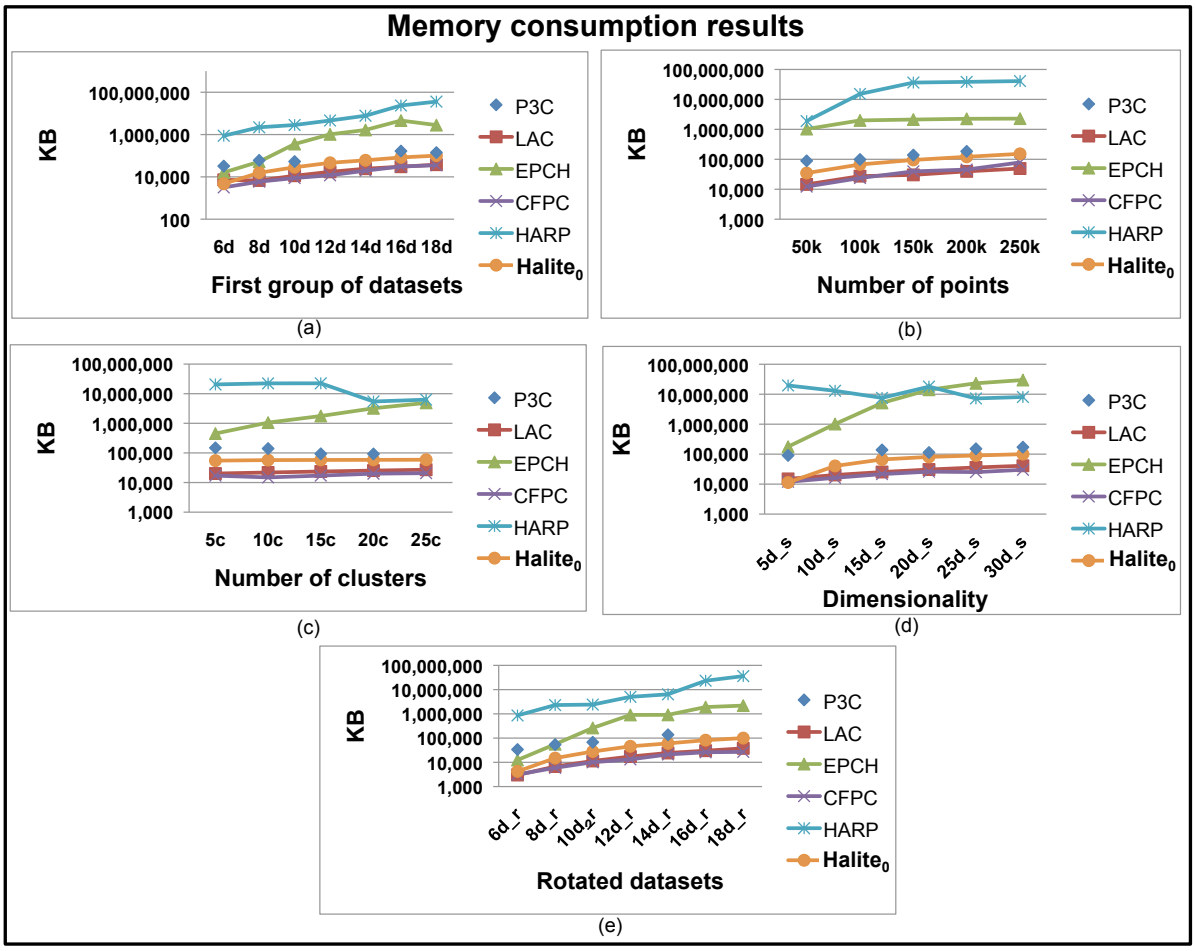


Figure 4.8: Results on memory consumption for synthetic data.

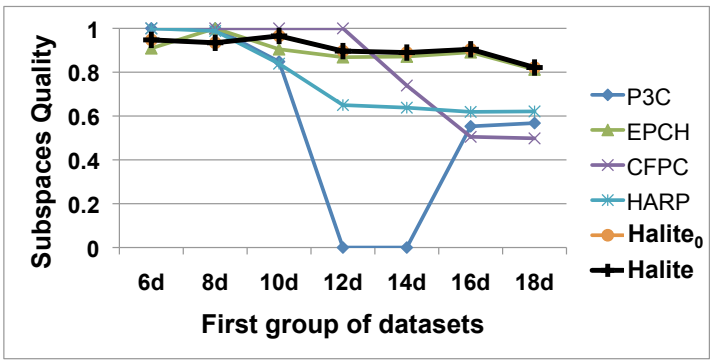


Figure 4.9: Subspace Quality for synthetic data.

Concluding, P3C had the worst Quality values in most cases. HARP, CFPC, ORCLUS and COPAC provided average Quality values for some datasets, but the results were not good in several cases. *Halite*, $Halite_0$, LAC and EPCH had the best results, in general tying with regard to Quality values. However, contrasting to *Halite* and also to $Halite_0$, the methods LAC and EPCH demanded guessing the number of clusters and required distinct threshold tuning to each dataset to obtain their best results reported. Regarding memory needs, remember that $Halite_0$, *Halite* and $Halite_s$ have similar space

complexity, the difference being that *Halite* and $Halite_s$ do manage the memory in disk if the Counting-tree does not fit in main memory. With that in mind, we compared the memory needs of the related works with those of $Halite_0$. This comparison shows that CFPC in general required the least amount of memory, followed by LAC, $Halite_0$, P3C, EPCH and HARP which respectively required 1.2, 2.8, 6.5, 112 and 600 times more memory than CFPC in average. Therefore, the memory consumption of $Halite_0$ was similar to those of top related works. Regarding run time, *Halite* was the fastest method in almost all cases, only loosing by little to $Halite_0$ in some small datasets. The improvements on our basic implementation allowed *Halite* to be about one order of magnitude faster than $Halite_0$ for large data. *Halite* also avoids the use of disk cache. Finally, when comparing to the other works, *Halite* was in average 12, 26, 32, 475, 756, $2,704$ and $7,218$ times faster than CFPC, LAC, EPCH, P3C, ORCLUS, HARP and COPAC respectively. Notice that *Halite* was in average at least 12 times faster than seven recent and related works, always providing high quality clusters.

Real Data: the real dataset used to test the methods is the training data provided for the Siemens KDD Cup 2008[4]. It was created for automatic breast cancer diagnosis, consisting of 25 of the most significant features extracted automatically from 102,294 Regions of Interest (ROIs) present in X-ray breast images of 118 malignant cases and 1,594 normal cases. Ground truth is also provided. The data was partitioned into four datasets, each containing features extracted from homogeneous images, i.e., each dataset has features extracted from $\sim 25,000$ ROIs related to images taken from one breast, left or right, in one of the possible directions, CC or MLO. The Quality results were computed based on the ground truth class label of each ROI.

Results on Real Data: all methods were tested with the real data. However, LAC and P3C failed for all four datasets in all tested configurations. LAC always grouped all points into a single cluster. P3C did not finish within a week for all cases. Thus, they are not reported. The results for left breast images in one MLO view are shown in Figure 4.10. *Halite* was at least 11 times faster than the previous works, increasing their accuracy in up to 35%. The other three real datasets led to similar results.

4.7.2 Scalability

This section analyzes the scalability of *Halite* and $Halite_s$ with regard to increasing data sizes and dimensionality. Synthetic datasets were created by Algorithm 6. The data sizes and dimensionality vary from $100k$ to 1 *million* points and from 5 to 30 axes respectively. The datasets have 10 axes-aligned clusters each and 15% of noise. Notice in Figure 4.11 that the proposed techniques scale as expected, according to the theoretical complexity analysis, presented in Section 4.3.

[4] http://www.kddcup2008.com

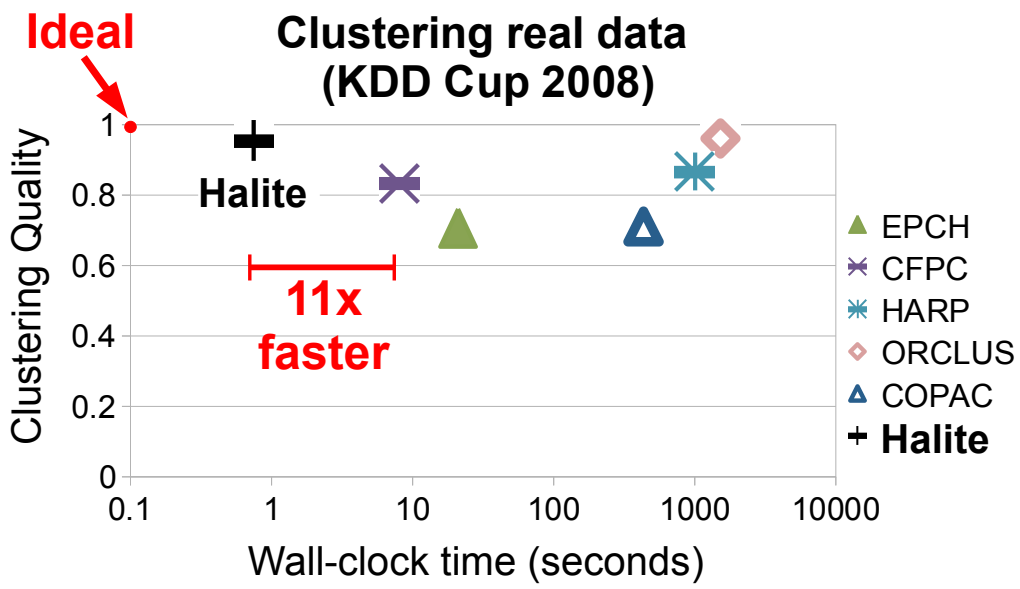
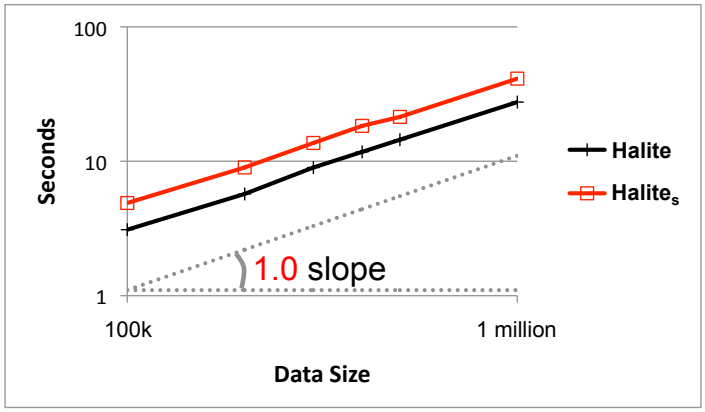


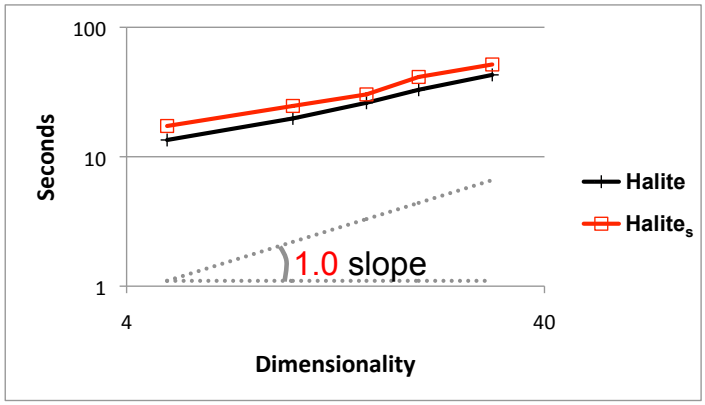
Figure 4.10: Quality versus run time in *linear-log scale* over 25-dimensional data for breast cancer diagnosis (KDD Cup 2008). *Halite* was at least 11 times faster than 5 previous works (2 other failed), increasing their accuracy in up to 35%. Similar behavior occurred in synthetic data.

4.7.3 Sensitivity Analysis

The behavior of our techniques varies based on two parameters: α and H. This section analyzes how they affect our methods. We varied both parameters for *Halite*, $Halite_0$ and $Halite_s$ to maximize the Quality values obtained, defining the best configuration for each of our datasets and techniques. Then, for each dataset and technique, we modified the best configuration, changing one parameter at a time, and analyzed the technique's behavior. For example, when varying H for a dataset and technique, the value of α was fixed at the value in the respective best configuration. The tested values of α and H vary from $1.0E-3$ to $1.0E-160$ and from 4 to 80 respectively. Figure 4.12 reports the results of $Halite_0$. Figures 4.12a and 4.12c present the results wrt α. Notice that, the values of α that led to the best Quality vary from $1.0E-5$ to $1.0E-20$ and the run time was barely affected by changes in α. Concerning H, Figures 4.12b and 4.12d show that the Quality does not increase significantly for H higher than 4. However, the run time increased as expected wrt H. Thus, small values for H, such as 4, are enough for most datasets. Similar results were obtained by *Halite* and $Halite_s$ for all synthetic and real datasets. Therefore, we define the values $\alpha = 1.0E-10$ and $H = 4$ to be the default configuration for our proposed techniques. These fixed values were used in all other experiments reported here.



(a) scalability wrt the Data Size



(b) scalability wrt the Dimensionality

Figure 4.11: Scalability of *Halite* and *Halite*$_s$ on synthetic data of varying sizes and dimensionality. Plots in *log-log* scale. Notice: both methods scale as expected, according to the theoretical complexity analysis in Section 4.3.

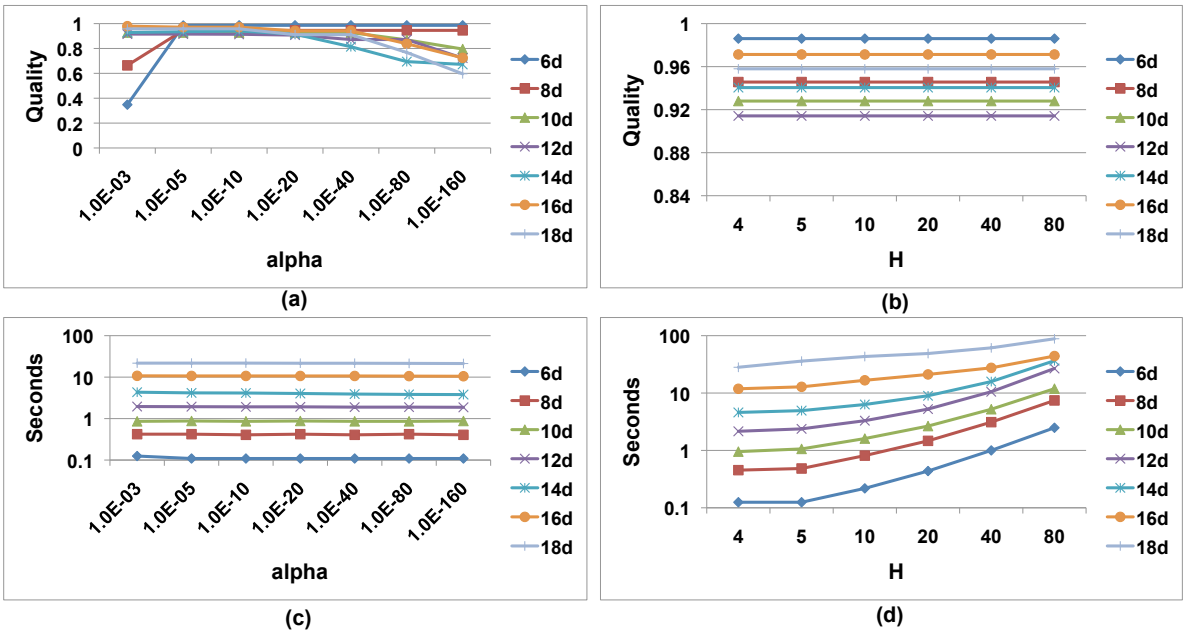


Figure 4.12: Sensitivity analysis. It led to the definition of our default configuration $\alpha = 1E-10$ and $H = 4$.

4.7.4 Soft Clustering

This section compares *Halite*$_s$ with STATPC, a well-known, state-of-the-art soft clustering method. The original code for STATPC was used, which was gracefully provided by Gabriela Moise. As suggested by the authors, the input parameter values used were the default values, $\alpha_0 = 1.0E - 10$ and $\alpha_K = \alpha_H = 0.001$. *Halite*$_s$ uses fixed input parameter values, as defined in Section 4.7.3. The methods were compared in a scenario with high probability of cluster overlap, our example scenario from Section 4.5. We analyzed 14 high quality satellite images from famous cities, as the city of Hong Kong in Figure 4.13a. The images, available at 'geoeye.com', have a combined size of $17MB$. Each image was divided into equal-sized rectangular tiles, from which Haar wavelets features were extracted. The process led to a 10-dimensional dataset of $14,336$ points.

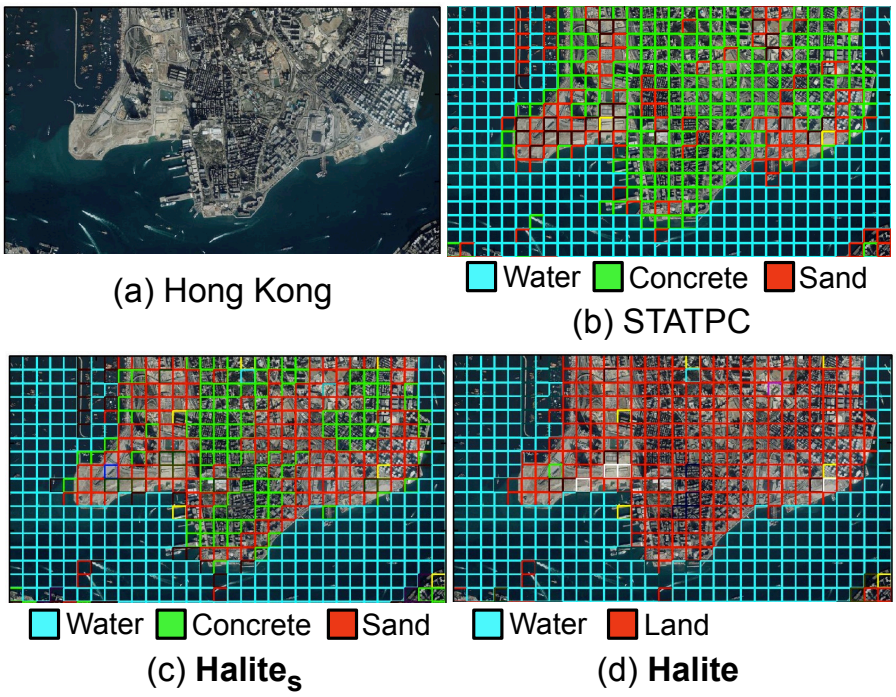


Figure 4.13: Comparing *Halite* and *Halite*$_s$ with STATPC on data with cluster overlap (best viewed in color). As expected, hard clustering leads to correct, but less detailed results: roughly speaking, *Halite*$_s$ and STATPC found clusters of 'Water' tiles (cyan), 'Sand' tiles (red) and 'Concrete' tiles (green); *Halite* merged the last two clusters into a cluster of 'Land' tiles (red). The results for *Halite*$_s$ and STATPC are similar. Notice, however, that *Halite*$_s$ found the clusters in only two *seconds*, while STATPC took two *days* to perform the same task. So, our solution can be used in *real time applications*.

Figures 4.13b and 4.13c respectively exemplify the results for STATPC and *Halite*$_s$ over this data by coloring each tile from the example image of Hong Kong according to its cluster. As expected, some tiles belong to more than one cluster. These were colored according to their first assigned clusters. Notice that both results are similar, with clusters that represent the main patterns apparent in the example image. However, STATPC took

two *days* to find the clusters, while *Halite*$_s$ performed the same task in only two *seconds*. Similar results were obtained for the other 13 images.

Notice that such results indicate that our solution allows the development of *real time applications* that, potentially, could not be developed without it, like a software to aid on the fly the diagnosis process in a worldwide Healthcare Information System, or a system to look for deforestation within the Amazon Rainforest in real time.

Finally, we report results for *Halite* (Figure 4.13d), which, as expected, are still correct, but provide less details compared to the soft clustering ones. Roughly speaking, *Halite*$_s$ and STATPC found clusters of 'water' tiles (cyan), 'sand' tiles (red) and 'concrete' tiles (green), whereas *Halite* merged the last two clusters into a single cluster of 'land' tiles (red). These results corroborate our conjecture from Section 4.5, that soft clustering is more appropriate to this kind of data.

4.8 Discussion

This section provides a discussion on some specific characteristics of the clustering methods proposed. As a first topic to discuss, notice that our method looks for dense space regions, and thus it works for any data distribution. We illustrated it on rotated Gaussians, as well as on real data of unknown distribution.

Also, we claim that the quadratic behavior on the number of β-clusters found is not a crucial concern for our techniques. Experimental evaluation showed that this number closely follows the ground truth number of clusters existing in the tested datasets. In our experiments, the biggest number of β-clusters found over all synthetic and real datasets was 33. Notice that, the biggest number of clusters present in these data is 25. The results for the first group of synthetic datasets are in Figure 4.14, which shows a plot with the Y-axis referring to each dataset tested, and horizontal bars representing the respective number of β-clusters found by *Halite*, besides the ground-truth number of clusters. The same pattern was seen in all of our other datasets. Furthermore, the analysis of data with many clusters is usually meaningless, as it is very hard to the user to obtain semantic interpretations from a large number of clusters.

We also claim that the quadratic behavior on H is not a relevant concern for *Halite*, since very small values for H are commonly sufficient to obtain accurate clustering results. Remember that a Counting-tree describes a d-dimensional dataset in H distinct resolutions. Each cell in the first resolution level $h = 0$ is divided in 2^d cells in level $h+1$, which are divided again in 2^d cells in level $h+2$ each, and so on. The process stops when $h = H-1$. Thus, for moderate-to-high dimensionality data, the maximum count of points in a cell of tree level h converges exponentially to 1, as the value of h increases to reach $H-1$. After the point of convergence, even for a skewed dataset, more levels tend to be

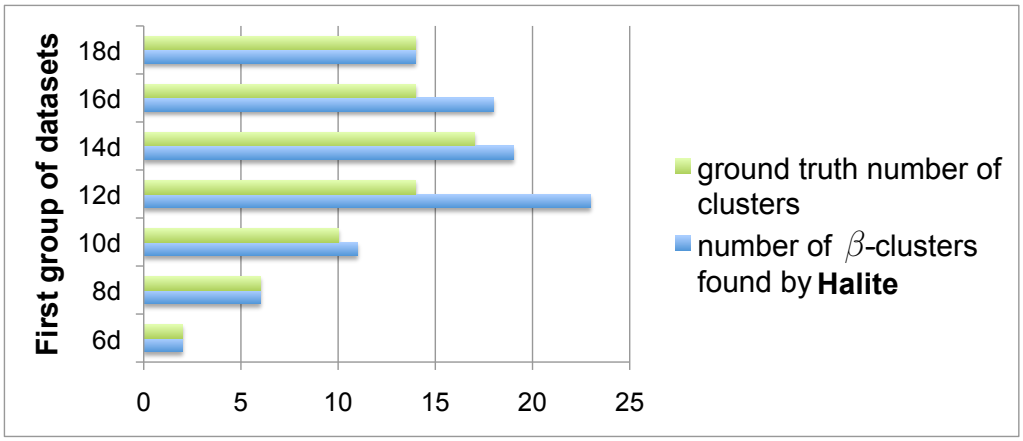


Figure 4.14: Ground truth number of clusters versus the number of β-clusters found by *Halite* over synthetic data.

useless, since they would not help to better describe the data. The sensitivity analysis in Section 4.7.3 corroborate this claim.

Additionally, notice that, *Halite* is limited to the size of the clusters that it finds. Our method analyses the points distribution in specific regions of the data space with all dimensions using an statistical hypothesis test to identify β-clusters, which lead to the clusters. However, these regions must have a minimum amount of points to reject the null hypothesis. In this way, *Halite* may miss clusters with small amount of points present in low-dimensional subspaces, since they tend to be extremely sparse in spaces with several dimensions. On the other hand, the clustering results tend to be better as the number of points in the clusters increase. Thus, *Halite* is suitable for large, multi-dimensional datasets and as it scales linearly with the dataset size, it becomes better as the dataset increases. In this way, *Halite* tends to be able to spot accurate clusters even from datasets with more than 30 axes, when they are large enough.

Notice one last remark: the *traditional* clustering method STING [Wang et al., 1997] is a basis to the work presented in this chapter. Similarly to *Halite*, STING also does multi-resolution space division in a statistical approach for clustering. However, STING is a *traditional* clustering method that proposes to only analyze two or very low dimensional data. It is *not* suitable for moderate-to-high dimensionality data clustering, since STING does *not* spot clusters that only exist in subspaces of the original data space. Also, STING uses a fixed density threshold to find clusters, whereas *Halite* applies a novel spike detection strategy based on convolution masks to find possible clusters, and then *Halite* confirms the clusters by using a statistical test to identify the spikes that are significantly denser than their neighboring space regions. Finally, STING does not include a soft clustering approach.

4.9 Conclusions

This chapter presented the new ***Halite* method for correlation clustering** that we developed as part of this Ph.D. work. Existing methods are typically super-linear in space or execution time. The main strengths of *Halite* are that it is fast and scalable, while still giving highly accurate results. In details, the main contributions of *Halite* are:

1. **Scalability:** it is linear in time and space with regard to the data size and to the dimensionality of the clusters. *Halite* is also linear in memory usage and quasi-linear in running time regarding the space dimensionality;

2. **Usability:** it is deterministic, robust to noise, does not have the number of clusters as a parameter and finds clusters in subspaces formed by original axes or their linear combinations, including space rotation;

3. **Effectiveness:** it is accurate, providing results with equal or better quality compared to top related works;

4. **Generality:** it includes a soft clustering approach, which allows points to be part of two or more clusters that overlap in the data space. Specifically, we introduced: $Halite_0$, a basic implementation for our method; *Halite*, our optimized, and finally proposed method for hard clustering, and; $Halite_s$, our recommended implementation for soft clustering.

A theoretical study on the time and space complexity of *Halite*, presented in Section 4.3, as well as an extensive experimental evaluation performed over synthetic and real data spanning up to 1 *million* elements corroborate these properties. Specifically, the experiments compared *Halite* with seven representative works. On synthetic data, *Halite* was consistently the fastest method, always presenting highly accurate results. Regarding real data, *Halite* analyzed 25-dimensional data for breast cancer diagnosis (KDD Cup 2008) at least 11 times faster than five previous works, increasing their accuracy in up to 35, while the last two related works failed.

Halite is the first knowledge discovery algorithm designed in this Ph.D. work. The next two chapters describe the other algorithms developed in this Ph.D. work.

Chapter

5

BoW

The large amounts of data collected by the enterprises are accumulating data, and today it is already feasible to have Terabyte- or even Petabyte-scale datasets that must be submitted for data mining processes. However, given a *Terabyte-scale* dataset of moderate-to-high dimensionality, how could one cluster its points? Numerous successful, serial clustering algorithms for data in five or more dimensions exist in literature, including our own algorithm *Halite*. However, the existing algorithms are impractical for datasets spanning Terabytes and Petabytes, and examples of applications with such huge amounts of data in five or more dimensions abound (e.g., Twitter crawl: $>$ 12 TB, Yahoo! operational data: 5 *Petabytes* [Fayyad, 2007a]). This limitation was previously summarized in Table 3.1 from Chapter 3. For datasets that do not even fit on a single disk, parallelism is a first class option, and thus we must re-think, re-design and re-implement existing serial algorithms in order to allow for parallel processing. In this chapter we explore parallelism using `MapReduce` for clustering huge datasets. Specifically, we present the second knowledge discovery algorithm designed in this Ph.D. work.

5.1 Introduction

Given a *Terabyte-scale* dataset of moderate-to-high dimensional elements, how could one cluster them? Numerous successful, serial clustering algorithms for data in five or more dimensions exist in literature, including our own algorithm *Halite*. However, the existing algorithms are impractical for data spanning Terabytes and Petabytes (e.g., Twitter crawl: $>$ 12 TB, Yahoo! operational data: 5 *Petabytes* [Fayyad, 2007a]). In such cases, the data are *already* stored on multiple disks, as the largest modern disks are 1-2TB. Just to read a single Terabyte of data (at 5GB/min on a single modern eSATA disk) one takes

more than 3 hours. Thus, parallelism is not another option – it is by far the best choice. Nevertheless, good, serial clustering algorithms and strategies are still extremely valuable, because we can (and should) use them as 'plug-ins' for parallel clustering. Naturally, the best algorithm is the one that combines (a) a fast, scalable serial algorithm and (b) makes it run efficiently in parallel. This is exactly what our proposed method does.

Examples of applications with Terabytes of data in five or more dimensions abound: weather monitoring systems and climate change models, where we want to record wind speed, temperature, rain, humidity, pollutants, etc; social networks like Facebook TM, with millions of nodes, and several attributes per node (gender, age, number of friends, etc); astrophysics data, such as the SDSS (Sloan Digital Sky Survey), with billions of galaxies and attributes like red-shift, diameter, spectrum, etc.

In this chapter we focus on the problem of finding clusters in subspaces of *huge, moderate-to-high dimensionality* datasets. Our proposed method uses `MapReduce`, and can treat as plug-in almost any of the serial clustering methods, including our own algorithm *Halite*. The major research challenges addressed here are (a) how to minimize the I/O cost, taking care of the *already existing* data partition (e.g., on disks), and (b) how to minimize the network cost among processing nodes. Either of them may become the bottleneck. Thus, we propose the ***B****est* ***o****f both* ***W****orlds* – *BoW* method, that automatically spots the bottleneck and chooses a good strategy. The main contributions of *BoW* are as follows:

1. *Algorithm design and analysis:* the *BoW* method is based on a novel, adaptive algorithm that automatically picks the best of two strategies and good parameters for it, hence its name ***B****est* ***o****f both* ***W****orlds*. One of the strategies uses our proposed *sampling-and-ignore* idea that reduces the network traffic;

2. *Effectiveness, scalability and generality:* we show that *BoW* can use most serial clustering methods as plug-ins, including our own method *Halite*. *BoW* requires no user defined parameters (due to our defaults) and it maintains the serial clustering quality, with near-linear scale-up;

3. *Experiments:* we report experiments on real and synthetic data with *billions* of elements, using hundreds of machines running in parallel.

We report results obtained from a combination of large real and synthetic datasets, including the *Yahoo! web* one.[1] To the best of our knowledge, the Yahoo! web is the largest real dataset for which results have ever been reported in the database clustering literature for data in five or more axes. Although spanning 0.2 TB of multi-dimensional data, *BoW* took only 8 minutes to cluster it, using 128 cores. We also report experiments that used $1,024$ cores, also the highest such number published in the clustering literature for moderate-to-high dimensional data.

[1] Provided by Yahoo! Research (www.yahoo.com).

Notice one *important* remark: *BoW* is *tailored to spot clusters in subspaces* of moderate-to-high dimensionality data and can handle most serial algorithms as plug-ins, since the only required API is that the serial algorithm should return clusters of points in hyper-rectangles, which we shall refer to as β-clusters, whose definition follows the same one previously employed for the *Halite* algorithm, but which may also be provided by many other existing algorithms. Overlapping β-clusters are then merged to form clusters. Indeed, the intuition is to generalize the structure of isometric crystal systems to the d-dimensional case in order to describe clusters of any shape and size, existing in subspaces only, as it was extensively discussed in the previous Chapter 4. Remember that the clustering methods well-suited to analyze moderate-to-high dimensionality data spot clusters that exist only in subspaces of the original, d-dimensional space (i.e., spaces formed by subsets of the original axes or linear combinations of them). Thus, the *natural* shape of the clusters in the original space facilitates their representation with hyper-rectangles, as the points of each cluster spread linearly through several axes (original axes or linear combinations of them) in the original space. For that reason, many of the existing serial, clustering methods (e.g., CLIQUE, FPC/CFPC, P3C, STATPC, and also our method *Halite*) return clusters in hyper-rectangles, and adapting others to work with *BoW* tends to be facilitated by the natural shape of the clusters. Nevertheless, besides focusing on spotting clusters in subspaces of moderate-to-high dimensionality data, *BoW* also works with traditional clustering methods and low dimensional data, if the plug-in returns clusters in hyper-rectangles.

5.2 Proposed Main Ideas – Reducing Bottlenecks

The major research problems for clustering *very large* datasets with `MapReduce` are (a) how to minimize the I/O cost, and (b) how to minimize the network cost among processing nodes. *Should we split the data points at random, across machines? What should each node do, and how should we combine the results? Do we lose accuracy (if any), compared to a serial algorithm on a huge-memory machine?*

Our proposed method answers all of those questions, by careful design and by adaptively trading-off disk delay and network delay. Specifically, we propose a novel, adaptive algorithm named *BoW* that is a hybrid between two strategies presented in this section: (i) the *ParC* method that does data partitioning and merges the results; and (ii) the *SnI* method that does some sampling first, to reduce communication cost at the expense of higher I/O cost. There is no universal winner between *ParC* and *SnI*, since it depends on the environment used and also on the dataset characteristics (see Section 5.5 for details). *BoW* automatically picks the best option, and good parameters for it. The reason for the success of *BoW* is our upcoming cost-estimation formulas (Equations 5.4 and 5.5), which help *BoW* to pick the best alternative and to set proper parameters for

the chosen environment, while requiring nimble computational effort. Next, we describe the methods *ParC* and *SnI* in detail.

5.2.1 Parallel Clustering – ParC

The *ParC* algorithm has three steps: (1) appropriately partition the input data and assign each data partition to one machine, (2) each machine finds clusters in its assigned partition, named as β-clusters, and, (3) merge the β-clusters found to get the final clusters. There are subtle issues on how to merge the results once we do clustering on each machine, which are detailed in Section 5.4.

We give the details in Section 5.4, but in a nutshell, we consider three options for data partitioning: (a) *random data partitioning*: elements are assigned to machines at random, striving for load balance; (b) *address-space data partitioning*: eventually, nearby elements in the data space often end up in the same machine, trading-off load balance to achieve better merging of the β-clusters; and (c) *arrival order or 'file-based' data partitioning*: the first several elements in the collection go to one machine, the next batch goes to the second, and so on, achieving perfect load balance. The rationale is that it may *also* facilitate the merging of the β-clusters, because data elements that are stored consecutively on the disk, may also be nearby in address space, due to locality: For example, galaxy records from the Sloan Digital Sky Survey (SDSS) are scanned every night with smooth moves of the telescope, and thus galaxies close in (2-d) address space, often result in records that are stored in nearby locations on the disk.

As described in Section 2.5 of Chapter 2, a `MapReduce`-based application has at least two modules: the map and the reduce. Our *ParC* method partitions the data through `MapReduce` mappers and does the clustering in `MapReduce` reducers. The final merging is performed serially, since it only processes the clusters descriptions, which consist of a tiny amount of data and processing. Figure 5.1a (5.1b will be explained latter) illustrates the process. It starts in phase **P1** with m mappers reading the data in parallel from the `MapReduce` distributed file system. In this phase, each mapper receives a data element at a time, computes its key, according to the data partition strategy used, and outputs a pair $\langle key, point \rangle$. All elements with the same key are forwarded in phase **P2** to be processed together, by the same reducer, and the elements with distinct keys are processed apart, by distinct reducers.

In phase **P3**, each reducer receives its assigned set of elements and normalizes them to a unitary hyper-cube. Each reducer then applies the plugged-in serial clustering algorithm over the normalized elements, aiming to spot β-clusters. For each β-cluster found, the reducer outputs, in phase **P4**, a pair $\langle key, cluster\_description \rangle$. The key concatenates the reducer identification and a cluster identification. The reducer identification is the input key. The cluster identification is a sequential number according to the order in which

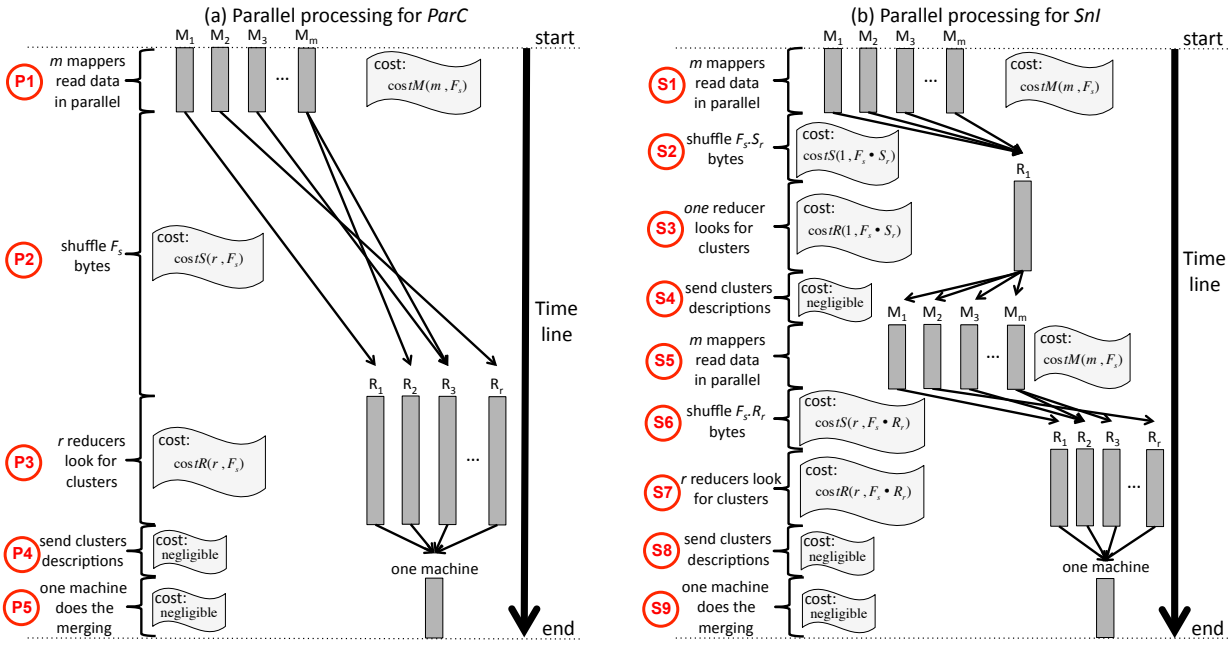


Figure 5.1: *Which one is best?* Parallel run overview for *ParC* (left) and *SnI* (right - with sampling). *ParC* executes the map (*P1*), shuffle (*P2*) and reduce (*P3*) phases once, on the full dataset. *SnI* uses sampling (phases *S1-S4*) to get rough cluster estimates and then uses phases *S5-S9* to cluster the remaining points (see Section 5.2.2 for details). Their clustering qualities are similar (see Section 5.5). The winning approach depends on the environment; *BoW* uses cost-based optimization to automatically pick the best.

the β-cluster was found in the corresponding reducer. A β-cluster description consists of the unnormalized minimum/maximum limits of the cluster in each dimension, defining a hyper-rectangle in the data space. Notice that this is a tiny amount of data, amounting to two float values per axis, per β-cluster.

The last step is phase **P5**, that involves merging and/or stitching the β-clusters provided by all the reducers to calculate the final answer. This step is performed serially, as it processes only the tiny amount of data (the bounds of each β-cluster found) received from phase **P4**, and *not* the data elements themselves. The best strategy to follow in this step is highly dependent on the criteria used by the mapper to partition the data. Thus, *ParC* uses distinct procedures for distinct data partitioning criteria. The procedures used for each of the partitioning strategies that we studied are detailed in Section 5.4.

5.2.2 Sample and Ignore – SnI

Our initial implementation for parallel clustering, the *ParC* algorithm, reads the dataset once, aiming at minimizing disk accesses, which is the most common strategy used by serial algorithms to shrink computational costs. However, this strategy does not address the issue of minimizing the network traffic: in the shuffle phase of the *ParC* algorithm (phase **P2** of Figure 5.1a), almost all of the records have to be shipped over the network,

to the appropriate reducer. It may become a considerable bottleneck. *How can we reduce this network traffic?*

The main idea in this section is to minimize the network traffic for parallel clustering by exploiting the skewed distribution of cluster sizes that typically appears in real datasets. Most of the elements usually belong to a few large clusters, and these are exactly the elements that we try to *avoid* processing. Thus, we propose *SnI*, a parallel clustering algorithm that consists of: (a) a novel *sample-and-ignore* preprocessing step; and (b) the *ParC* algorithm from Section 5.2.1. The *sample-and-ignore* step works on a small dataset sample, spots the major clusters and ignores their members in the follow-up steps. It significantly reduces the amount of data moved in the shuffling phases of *SnI*, with consequent savings for the network traffic, as well as the I/O cost for the intermediate results and processing cost for the receiving reduce tasks. Notice that the proposed *sample-and-ignore* idea is an alternative general strategy that can improve many clustering methods, and not only *ParC*.

The *SnI* method is defined in Algorithm 7 and the process is illustrated in Figure 5.1b. At a high-level, in Phase I (steps **S1-S4** in the figure, and lines **1-3** in the algorithm) the method *samples* the input data and builds an initial set of clusters. In the second phase (steps **S5-S9** in the figure, and lines **4-8** in the algorithm), the input data is filtered, so that we only include *unclassified* elements, that is, those that do not belong to any of the clusters found in Phase I. These unclassified elements are then clustered using *ParC*.

Algorithm 7 : Multi-phase Sample-and-Ignore (SnI) Method.

Input: dataset ${}^{d}S$
sampling ratio S_r
Output: *clusters*

1: // Phase 1 – Sample
2: m mappers read data and send elements to one reducer with probability S_r;
3: one reducer uses plug-in to find clusters in $\sim \eta.S_r$ received elements, and passes clusters descriptions to m mappers;
4: // Phase 2 – Ignore
5: m mappers read data, ignore elements from clusters found in sample and send the rest to r reducers, according to the Data Partition Approach;
6: r reducers use plug-in to find clusters in the received elements, and send clusters descriptions to one machine;
7: one machine merges clusters received and the ones from sample, let the merged result be *clusters*;
8: **return** *clusters*

Figure 5.2 illustrates the *SnI* approach over a toy dataset, assuming that we have $r = 2$ reducers available for parallel processing. The top part of the figure shows Phase-I. First, in Phase-I (a) the input dataset is read in parallel by m map tasks, each mapper passes the input elements to the same reducer with some probability, for example, 0.5 for the case shown in the figure. A single reducer builds clusters using the sample elements

in Phase-I (b). In this case two clusters were found and are denoted by the gray boxes around the elements. The summary descriptors of the clusters found in Phase-I, i.e., the minimum/maximum limits of the clusters wrt each dimension, are passed to Phase-II.

In Phase-II (a), m mappers perform a second pass over the data, this time filtering out points that fall in the clusters found in Phase-I, which are denoted by the black boxes. The elements that do not fall into clusters are passed to the two reducers available, as shown in Phase-II (b) and (c), in which we assume that the used partitioning strategy divided the elements into 'black points' and 'white points'. Each reducer finds new clusters, denoted by the points surrounded by dotted boxes. In Phase-II (d), the clusters found by the reducers are merged with the clusters from the sampling phase using the upcoming merging/stitching strategies described in Section 5.4. The global set of clusters, containing three clusters represented in Phase-II (e) by distinct gray levels, is the final output.

The main benefit of the *SnI* approach is realized in the shuffle/reduce stages. In Phases **S2** and **S3** of Figure 5.1b, only a small sample is shuffled and processed by a receiving reducer. In Phases **S6** and **S7** of Figure 5.1b, only the *non-ignored* elements may need to be shuffled through the network to other machines and processed. This means that most elements belonging to the major clusters spotted in the sample are ignored, *never* being shuffled through the network *nor* processed by a reducer. Compared to the *ParC* algorithm, *SnI* significantly minimizes the network cost and the reducers processing, at the cost of reading the whole dataset twice. In other words, *ParC* does a single pass over the data, but almost all of the records have to be shipped over the network (in phase **P2** of Figure 5.1a), to be processed by the appropriate reducer. On the other hand, *SnI* minimizes the shuffle/reduce cost, at the expense of reading the dataset one extra time. *What approach is the best?* The answer is given in Section 5.3.

5.3 Proposed Cost-based Optimization

This section proposes an adaptive, hybrid method named ***BoW*** *(**B**est **o**f both **W**orlds)* that exploits the advantages of the previously described approaches, *ParC* and *SnI*, taking the best of them. There is no universal winner, since it depends on the environment and on the dataset characteristics. See Section 5.5 for a complete explanation. Therefore, the main question here is: *When should our sampling-and-ignore idea be used and when should it be avoided? ParC* executes the map, shuffle and reduce phases only once on the whole dataset. *SnI* reduces the amount of data to be shipped to and processed by the reducers, at the expense of a second pass on the input data (in the map phase). We propose a cost-based optimization that uses simple analytics models to estimate the running time of each clustering strategy. *BoW* picks the strategy with the lowest estimated cost.

The environmental parameters required by *BoW* are presented in Table 5.1. They describe the hardware characteristics (i.e., the specs of the available `MapReduce` cluster),

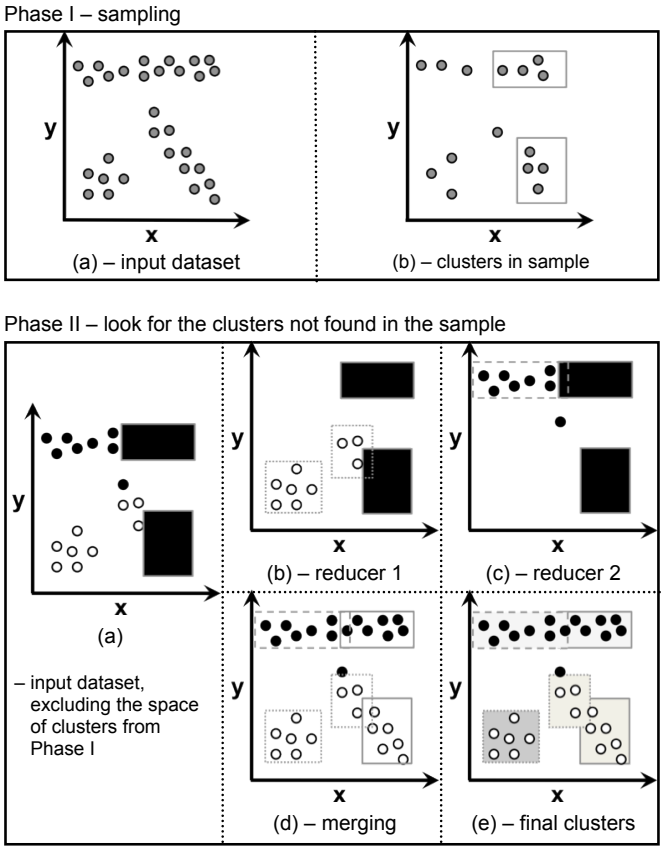


Figure 5.2: Overview of the Multi-phase Sample-and-Ignore (SnI) Method. Phase-I finds clusters on a sample of the input data. Phase-II ignores elements that fall within a previously found cluster and finds clusters using the remaining elements only.

the total amount of data to be processed, and the cost estimate for the plugged-in serial clustering method. Setting the value for F_s is straightforward. D_s, N_s and $start\_up\_cost(t)$ are inferred by analyzing the cloud of computers' logs, while $plug\_in\_cost(s)$ is defined based on the plugged-in method's original time complexity analysis and/or experiments, or measured by the user in a simple experiment. Notice: each machine in the cloud may run many `MapReduce` tasks (mappers and/or reducers) in parallel, *sharing* the machine's disks and network connection. Therefore, N_s and D_s are expected to be *smaller* than the effective network bandwidth and disk transfer rate respectively.

Two other parameters are used, shown in Table 5.2. We provide reasonable default values for them based on empirical evaluation. Notice one *important* remark: As is the common knowledge in database query optimization, at the cross-over point of two strategies, the wall-clock-time performances usually create *flat plateaus*, being not much sensitive to parameter variations. This occurs in our setting, and the results in the upcoming Figures 5.9a, 5.10a and 5.10d exemplify it (notice the log-log scale). Thus, tuning exact values to our parameters barely affects *BoW*'s results and the suggested values are expected to work well in most cases.

Parameter	Meaning	Explanation
F_s	data file size (bytes)	Size of the dataset to be clustered.
D_s	disk speed (bytes/sec.)	Average number of bytes per second that a `MapReduce` task (mapper or reducer) is able to read from local disks, i.e. the average disk transfer rate per `MapReduce` task.
N_s	network speed (bytes/sec.)	Average bytes/sec. that a `MapReduce` task (mapper or reducer) is able to read from other computers in the cloud, i.e. the average network transfer rate per `MapReduce` task.
$start\_up\_cost(t)$	start-up cost (seconds)	Average time to start-up t `MapReduce` tasks (mappers or reducers).
$plug\_in\_cost(s)$	plug-in cost (seconds)	Average time to run the plugged-in serial method over s data bytes on a standard computer in the cloud.

Table 5.1: Environmental parameters.

Parameter	Meaning	Explanation	Our defaults
D_r	dispersion ratio	Ratio of data transferred in the shuffle phase through the network (distinct machines) relative to the total amount of data processed.	0.5
R_r	reduction ratio	Ratio of data that do not belong to the major clusters found in the sampling phase of *SnI* relative to the full data size F_s.	0.1

Table 5.2: Other parameters.

The following lemmas and proofs define the equations of the cost-based optimization. First, we describe the expected costs for complete map, shuffle and reduce phases relative to the number of mappers and/or reducers available and to the amount of data involved. Then, we infer the costs for: *ParC*, which minimizes disk accesses, and; *SnI*, which aims at shrinking the network cost. For clarity, consider again Figure 5.1 that provides a graphical overview of the parallel execution of both methods, including their expected cost equations.

Lemma 1 *Map Cost – the expected cost for the map phase of the parallel clustering approaches is a function of the number of mappers m used and the involved data size s, given by:*

$$costM(m, s) = start\_up\_cost(m) + \frac{s}{m} \cdot \frac{1}{D_s} \tag{5.1}$$

Proof: In the map phase, m mappers are started-up at the cost of $start\_up\_cost(m)$. Additionally, the majority of the time spent is related to reading the input dataset from disk. In our case, s bytes of data will be read in parallel by m mappers, which are able to read D_s bytes per second each. Thus, the total reading time is given by: $\frac{s}{m} \cdot \frac{1}{D_s}$. ■

Lemma 2 *Shuffle Cost – the expected shuffle cost of the parallel clustering approach is a function of the number of reducers r to receive the data and the amount of data to be shuffled s, which is given by:*

$$costS(r, s) = \frac{s \cdot D_r}{r} \cdot \frac{1}{N_s} \tag{5.2}$$

Proof: The majority of the shuffling cost is related to shipping data between distinct machines through the network. Whenever possible, `MapReduce` minimizes this cost by assigning reduce tasks to the machines that already have the required data in local disks. D_r is the ratio of data actually shipped between distinct machines relative to the total amount of data processed. Thus, the total amount of data to be shipped is $s \cdot D_r$ bytes. The data will be received in parallel by r reducers, each one receiving in average N_s bytes per second. Thus, the total cost is given by: $\frac{s \cdot D_r}{r} \cdot \frac{1}{N_s}$. ■

Lemma 3 *Reduce Cost – the expected cost for the reduce phase is a function of the number of reducers r used for parallel processing and the size s of the data involved, which is:*

$$\begin{aligned} costR(r, s) &= start\_up\_cost(r) + \frac{s}{r} \cdot \frac{1}{D_s} + \\ & \quad plug\_in\_cost(\frac{s}{r}) \end{aligned} \tag{5.3}$$

Proof: In the reduce phase, r reducers are started-up at cost $start\_up\_cost(r)$. After the start-up process, the reducers will read from disk s bytes in parallel at the individual cost of D_s bytes per second. Thus, the total reading time is $\frac{s}{r} \cdot \frac{1}{D_s}$. Finally, the plugged-in serial clustering method will be executed in parallel over partitions of the data, whose average sizes are $\frac{s}{r}$. Therefore, the approximate clustering cost is $plug\_in\_cost(\frac{s}{r})$. ■

Lemma 4 *ParC Cost – the expected cost of the* ParC *algorithm is given by:*

$$costC = costM(m, F_s) + costS(r, F_s) + costR(r, F_s) \tag{5.4}$$

Proof: The parallel processing for *ParC* is as follows: (i) F_s bytes of data are processed in the map phase, by m mappers; (ii) F_s bytes of data are shuffled to r reducers in the shuffling phase; (iii) F_s bytes of data are processed in the reduce phase by r reducers, and; (iv) a single machine merges all the β-clusters found. The last step has a negligible cost, since it performs simple computations over data amounting to two float values per β-cluster, per dimension. Thus, summing the costs of the three initial phases leads to the expected cost for *ParC*. ■

Lemma 5 *SnI Cost – the expected cost for the* SnI *algorithm is given by:*

$$\begin{aligned} costCs &= 2 \cdot costM(m, F_s) + \\ & costS(1, F_s \cdot S_r) + costR(1, F_s \cdot S_r) + \\ & costS(r, F_s \cdot R_r) + costR(r, F_s \cdot R_r) \end{aligned} \tag{5.5}$$

Proof: *SnI* runs two complete map, shuffle and reduce phases. In both map phases, the full dataset is processed by m mappers, at combined cost: $2 \cdot costM(m, F_s)$. In the first shuffle phase, a data sample of size $F_s \cdot S_r$ bytes is shuffled to a single reducer, at cost $costS(1, F_s \cdot S_r)$. The reduce cost to process this sample is: $costR(1, F_s \cdot S_r)$. R_r is the ratio of data that does not belong to the major clusters, the ones found in the sampling phase, relative to F_s. That is, $F_s \cdot (1 - R_r)$ bytes are *ignored* in the Second Phase of *SnI*, while $F_s \cdot R_r$ bytes of data are *not ignored*, being processed after clustering the sample. Both second shuffle and reduce phases involve r reducers. Thus, their combined costs are: $costS(r, F_s \cdot R_r) + costR(r, F_s \cdot R_r)$. The costs for shipping and processing β-clusters descriptions is negligible, since the involved amount of data and processing is tiny. ■

Remark: when our algorithms are executed, the number of distinct key values to be sorted by the `MapReduce` framework is extremely small; it is *always* the number r of reducers used only. Each reducer handles a single key, so it does not need to do sorting. Thus, the sorting cost is negligible for our approaches. The I/O and network costs are the real bottlenecks. The wall-clock time results measured in all of our experiments (see Section 5.5) confirm this assertion.

Algorithm 8 describes the main steps of *BoW*. In summary, *ParC* executes the map, shuffle and reduce phases once, involving the full dataset. *SnI* runs these phases twice, but involving less data. *What is the fastest approach?* It depends on your environment. *BoW* takes the environment description as input and uses cost-based optimization to automatically choose the fastest, prior to the real execution. Provided that the clustering accuracies are similar for both approaches (see Section 5.5 for a complete explanation), *BoW* actually picks the *'Best of both Worlds'*.

5.4 Finishing Touches – Partitioning the Dataset and Stitching the Clusters

This section describes three reasonable approaches proposed for data partitioning and consequent merging and/or stitching of the clusters found in each partition. Notice that *BoW* works with any of the three partitioning approaches described and, *potentially*, works with any user-defined partitioning strategy.

Algorithm 8 : The ***B****est* ***o****f both* ***W****orlds* – ***BoW*** Method.

Input: dataset ${}^{d}S$
environmental parameters (Table 5.1),
other parameters (Table 5.2),
number of reducers r,
number of mappers m,
sampling ratio S_r

Output: *clusters*

1: compute *costC* from Equation 5.4;
2: compute *costCs* from Equation 5.5;
3: **if** $costC > costCs$ **then**
4: // use the *sampling-and-ignore* idea
clusters = result of *SnI* over ${}^{d}S$;
5: **else**
6: // no sampling
clusters = result of *ParC* over ${}^{d}S$;
7: **end if**
8: **return** *clusters*

5.4.1 Random-Based Data Partition

The first alternative is the **Random-Based Data Partition.** Mappers randomly assign data elements to reducers, striving for load balance. Each reducer receives a random sample of the dataset, looks for β-clusters on it, and reports the β-clusters it finds, in terms of their MBRs (Minimum Bounding Rectangles).

The final step of the computation merges every pair of β-clusters that overlap in the data space. Notice that, to spot an overlap, we need *only* the descriptions of the β-clusters (MBRs), and *not* the elements themselves. Two clusters overlap if they overlap in every axis j: Let u_{ij} and l_{ij} represent respectively the upper and lower bounds of cluster i at axis j. Similarly, let $u_{i'j}$ and $l_{i'j}$ represent the bounds of cluster i' at axis j. Two β-clusters i and i' overlap if $u_{ij} \geq l_{i'j} \ \wedge \ l_{ij} \leq u_{i'j}$ holds for every axis j.

Figure 5.3(I) illustrates a simulation of this process assuming that we have $r = 2$ reducers. The first reducer gets the nodes indicated as 'white-circles' and the second one gets the 'black-circles'; both reducers run a typical clustering algorithm, returning the MBRs (Minimum Bounding Rectangles) of the β-clusters they discover (Figure 5.3(I)(b,c)). Then, we merge the overlapping β-clusters (Figure 5.3(I)(d)), and return the results, indicated as the shaded areas of (Figure 5.3(I)(e)). Notice that some data points may be left as outliers, which is a possibility for all the parallel methods that we present, as well as for most serial clustering algorithms.

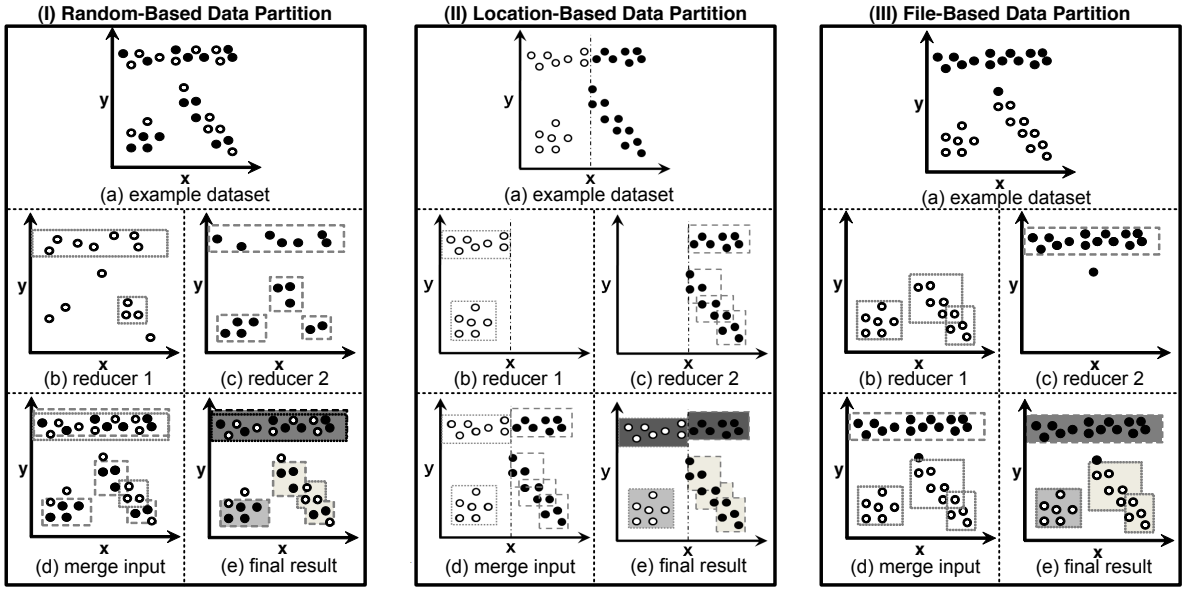


Figure 5.3: 'file-based' wins. Clustering examples for the three data partitioning approaches. We assume exactly the same 2-d input dataset, with r=2 reducers. (I–left) assigns elements at random to reducers, and merges the resulting β-clusters that overlap. (II–middle) divides the address space in disjoint regions, assigns each region to a reducer, and then either merges, or *stitches* the appropriate β-clusters (see Section 5.4.2 for details). (III–right) assigns elements to reducers according to their position in the data file, and hopes that, due to locality, the resulting β-clusters will have little overlap. As shown in Section 5.5, the 'file-based' strategy outperforms the first two alternatives.

5.4.2 Location-Based Data Partition

The second alternative is the **Location-Based Data Partition.** The idea here is to divide the *address space*, trading-off load balance to achieve better merging of the β-clusters. Specifically, we partition the address space into r disjoint regions (say, hyper-rectangles, by bi-secting some coordinate axes), where r is the number of reducers. The mappers are given the boundaries of every region, and direct each element accordingly. In our current implementation, we have r to be a power of two, since the partitions are created by dividing each dimension in half as needed.

Figure 5.3(II) illustrates a simulation of the process, using the same toy dataset of Figure 5.3(I) that we used to illustrate the previous approach. The data elements are assigned to reducers according to their location (vertical dashed line). Again, each of the two reducers generates MBRs of the β-clusters it finds. Then we (a) merge those that overlap and (b) stitch those that touch, like the two β-clusters on the top of Figure 5.3(II)(d).

The stitching step requires a careful design. We want to stitch together the clusters that touch in partitioned positions with respect to one or more axes, and have "enough

Algorithm 9 : Stitching β-clusters i and i'.

Input: u_{ij} and l_{ij}, upper and lower bounds of β-cluster i in each axis j
$u_{i'j}$ and $l_{i'j}$, upper and lower bounds of β-cluster i' in each axis j
Output: merge

```
1: merge = true;
2: for each axis j do
3:    if (not (u_ij ≥ l_i'j ∧ l_ij ≤ u_i'j)) ∧ (not (axis j was partitioned ∧
      (u_ij = l_i'j = partitioned_position ∨ l_ij = u_i'j = partitioned_position))) then
4:       // do not overlap neither touch in a partitioned position in j
5:       merge = false;
6:    end if
7: end for
8: if merge then
9:    for each axis j do
10:      if (u_ij ≥ l_i'j ∧ l_ij ≤ u_i'j) then
11:         compute h_i, h_i' and h_i∩i' wrt j;
12:         if h_i∩i' ≤ (h_i − h_i∩i') + (h_i' − h_i∩i') then
13:            merge = false; // not "enough touching area" in j
14:         end if
15:      end if
16:   end for
17: end if
18: return merge
```

touching area" with regard to all other axes. In our running example, Figure 5.4 shows the input for this step. The β-clusters i and i' touch in a partitioned position of axis x. We propose to stitch two β-clusters if the area that they jointly touch is larger than the disjoint areas. In more detail, in the example of Figure 5.4, their "touching area" with regard to axis y is $h_{i \cap i'}$. As in the illustration, let h_i and $h_{i'}$ be the individual 1-dimensional heights wrt axis y of the cluster i and i', respectively. Our method consider this "touching area" as being "large enough" for stitching if the common part is larger than the union of the non-common parts, for each axis that do not touch in a partitioned position. It is defined by the following equation.

$$h_{i \cap i'} > (h_i - h_{i \cap i'}) + (h_{i'} - h_{i \cap i'}) \tag{5.6}$$

Notice that our "large enough" criterion is *parameter-free*. Algorithm 9 gives the full pseudo-code. In our running example, Figure 5.3(II.e) shows the final output for the merging / stitching procedure, assuming that the upper two β-clusters were stitched. The intermediate set of six β-clusters is summarized into three clusters, represented in three distinct gray levels in the illustration.

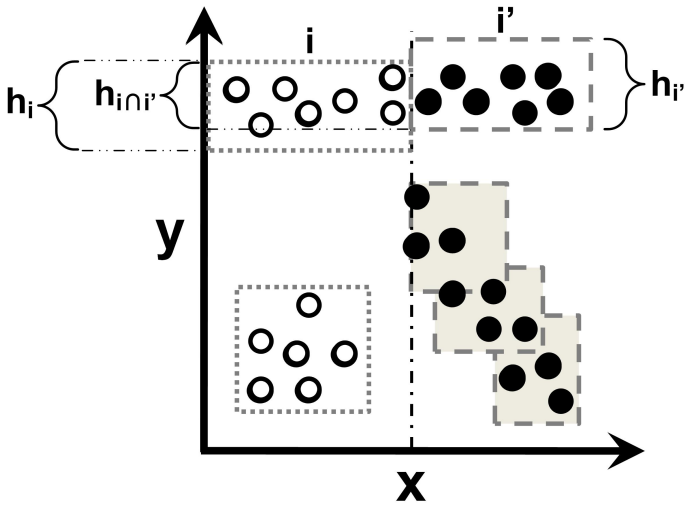


Figure 5.4: Merging and Stitching for the Location-based approach. **Merging**: the three lower-right β-clusters are merged, since they overlap. **Stitching**: the β-clusters i and i' are stitched to form a bigger cluster, since the height of the "touching area is large enough" compared to the heights of the β-clusters.

5.4.3 File-Based Data Partition

The third approach is the **File-Based Data Partition.** This approach has perfect load balance, assigning the first $1/r$ portion of the records to the first reducer, the second $1/r$ portion to the second one, and so one. The rationale is that it may *also* facilitate the merging of the β-clusters, because data elements that are stored consecutively on the disk, may also be nearby in address space, due to locality.

The specific steps are as follows: we want to divide the input file into r pieces of nearly equal sizes, whose elements are sequentially stored in the file. The `MapReduce` mappers receive the total number of elements η and the total number of reducers r available for parallel processing as input parameters. When an element is received, a mapper takes into account the physical order o of the element in the input file to define its appropriate key. The key is computed by the following equation: $floor(o/ceil((\eta+1)/r))$, assuring an even amount of elements to each partition. Thus, each reducer receives a set of elements sequentially stored in the input file, and then looks for β-clusters on it. The final step of the computation is identical to the random-based data partitioning approach: we merge every pair of β-clusters that overlap in the address space.

Figure 5.3(III) illustrates a simulation of the process assuming that we have $r = 2$ reducers. It follows the same process as in the random-based approach, except for the first step, where the data elements are assigned to reducers according to their location in the file. Assuming locality, we expect most of the black circles to be close in space, and similarly for the white circles. Each reducer reports its MBRs, and then the β-clusters with overlapping MBRs are merged. The hope is that, due to locality, there will be much

fewer pairs of overlapping β-clusters than in the random case, while enjoying even better load balancing.

5.5 Experimental Results

In this section, we describe the experiments performed to test the algorithms proposed in the chapter. We aimed at answering the following questions:

Q1 Among the reasonable choices proposed in Section 5.4, what is the best data partitioning approach?

Q2 How much (if at all) does the parallelism affect the clustering quality?

Q3 How does our method scale-up?

Q4 How accurate are the equations used in our cost-based optimization?

All experiments were performed using the `Hadoop`[2] implementation for the `MapReduce` framework, on two `Hadoop` clusters: the M45 by Yahoo! and the DISC/Cloud by Parallel Data Lab in the Carnegie Mellon University. The M45 is one of the top 50 supercomputers in the world totaling 400 machines ($3,200$ cores), 1.5 PB of storage and 3.5 TB of main memory. The DISC/Cloud has 512 cores, distributed in 64 machines, 1TB of RAM and 256 TB of raw disk storage. We used our own algorithm *Halite* as the serial clustering method for the plug-in for all experiments.

The methods were tested over the real and synthetic datasets listed in Table 5.3, which are detailed as follows.

- `YahooEig`: The top 6 eigenvectors from the adjacency matrix of one of the largest web graphs. The web graph was crawled by Yahoo![3] in 2002 and contains 1.4 *billion* nodes and 6.6 *billion* edges. The eigenvectors amount to 0.2 *TB*.

- `TwitterEig`: The top 10 eigenvectors from the adjacency matrix of the Twitter[4] graph, that represents 62 million users and their relationships. The eigenvectors amount to 14 GB.

- `Synthetic`: A group of datasets with sizes varying from 100 thousand up to 100 million 15-dimensional points, containing 10 clusters each, and no noise. Clusters in subspaces of the original 15-dimensional space were created following standard procedures used by most of the clustering algorithms described in Chapter 3, including the plugged-in serial clustering method used in our experiments.

[2] www.hadoop.com
[3] www.yahoo.com
[4] http://twitter.com/

Specifically, Algorithm 6 from the previous Chapter 4 was used again to generate the synthetic data. Axes-aligned clusters were created. Remember that the clusters generated by Algorithm 6 follow normal distributions with random mean and random standard deviation in at least 50% of the axes (relevant axes), spreading through at most 15% of the axes domains. In the other axes, the irrelevant ones, all clusters follow the uniform distribution, spreading through the whole axes domains.

Dataset	Number of Points	Number of Axes	File Size
`YahooEig`	1.4 *billion*	6	0.2 *TB*
`TwitterEig`	62 million	10	14 GB
`Synthetic`	up to 100 million	15	up to 14 GB

Table 5.3: Summary of datasets. TB: Terabytes; GB: Gigabytes.

Notice one remark: to evaluate how much (if at all) parallelism affects the serial clustering quality, the ideal strategy is to use as ground truth the clustering results obtained by running the plugged-in algorithm serially on any dataset, synthetic or real, and to compare these results to the ones obtained with parallel processing. However, for most of our large datasets, to run a serial algorithm (*Halite* or, potentially, any other serial clustering method for moderate-to-high dimensionality data) is an impractical task – it would require impractical amounts of main memory and/or take a very long time. Thus, in practice, the `Synthetic` datasets are the only ones from which we have clustering ground truth, and they were used to evaluate the quality of all tested techniques in all experiments performed.

For a fair comparison with the plugged-in serial algorithm, the quality is computed following the same procedure used in Section 4.7.1 of Chapter 4. That is, the quality is computed by comparing the results provided by each technique to the ground truth, based on the averaged precision and recall of all clusters.

The File-based Data Partitioning strategy may provide distinct quality results according to the order in which the input data is physically stored. Obviously, the best results appear when the data is totally ordered, i.e., the points of each cluster are sequentially stored in the data file. On the other hand, when the points are randomly distributed in the file, the qualities tend to be similar to those obtained by the approaches that use the Random-based Data Partitioning. For a fair analysis, we created each dataset from the `Synthetic` group considering an average case, i.e., 50% of the elements from the totally ordered case were randomly repositioned throughout the data file.

All experiments involving *BoW* were performed at M45. The parameters used are presented in Table 5.4. F_s refers to the data file size. D_s, N_s and $start\_up\_cost(t)$ were inferred by analyzing the logs of the M45 machines, while $plug\_in\_cost(s)$ was defined

based on the time complexity analysis and experiments of the plugged-in method, which were previously presented in Chapter 4.

Parameter	Value
F_s	data file size
D_s	$40MB/sec$
N_s	$20MB/sec$
$start\_up\_cost(t)$	$0.1t$
$plug\_in\_cost(s)$	$1.4E^{-7}s$

Table 5.4: Environmental parameters for M45.

The results on quality and wall-clock time reported for all experiments are the average of 10 *distinct runs*. We decided to use a sample size of nearly one million elements (i.e., $S_r = \frac{1\ million}{\eta}$) in all experiments. Also, in every experiment the number of mappers m used was automatically defined by `Hadoop`.

5.5.1 Comparing the Data Partitioning Strategies

This section compares the data partitioning approaches. Here we want to answer question $Q1$: Among the reasonable choices in Section 5.4, what is the best data partitioning approach? In order to answer it, we decided to use *ParC*, our most straightforward parallel algorithm. This decision aims at avoiding that algorithmic characteristics influence the results, which should be related to the pros and to the cons of the data partitioning strategies only. That is, we want to avoid that the used algorithm leads to biased results.

Figures 5.5(a) and 5.5(b) show the quality of *ParC* using distinct data partitioning approaches (*ParC-F*– file-based , *ParC-R* – random-based and *ParC-L* – location-based) versus the wall-clock time over the `Synthetic` datasets, varying η and r respectively. The data sizes vary from 100 thousand to 100 million elements, while the number of reducers r starts at 2 and goes up to 16. The glyph sizes reflect the dataset size (Figure 5.5a) or the number of reducers (Figure 5.5b). Obviously, the ideal elements are in the top left of both plots, which represent 100% quality obtained in zero time. Thus, we compared the strategies by analyzing how close they are to these elements in all cases. Notice that the three strategies present good quality, with some few exceptions for the Location and the Random-based ones. However, the File-based strategy consistently outperformed the others, presenting top quality and being the fastest one in all cases. The other `Synthetic` datasets generated very similar results. Thus, the *File-based Data Partition* is the partitioning strategy that we recommend to be used with *BoW*. The experiments presented in the rest of this chapter *always* employ this strategy.

We think that the main reason for the success of the 'file-based' approach is that, when using this approach, the reducers process continuous pieces of the input file. This

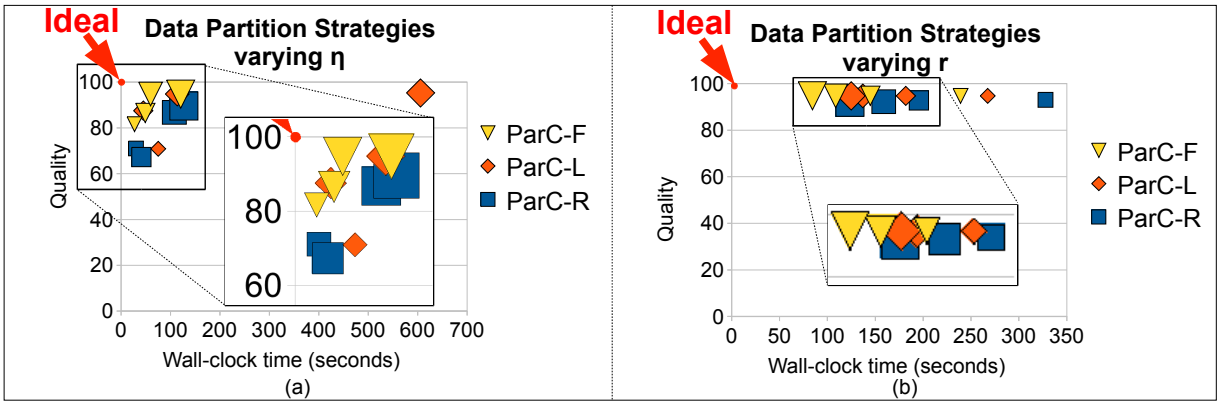


Figure 5.5: File-based wins. Quality versus run time for *ParC* using distinct data partitioning approaches (*ParC-F*– file-based: yellow triangles, *ParC-R* – random-based: blue squares and *ParC-L* – location-based: orange diamonds). Left, 64 reducers, varying the data size $\eta = 100K, 1M, 10M, 100M$. Right, 10 million elements dataset, varying the number of reducers $r = 2, 4, 8, 16$. The glyph sizes reflect the dataset size (a) or the number of reducers (b). Top-left is the ideal element - notice that *ParC-F* is consistently closer to it than the others. Thus, we recommend the File-based data partitioning approach.

helps `Hadoop` to assign reduce tasks to the machines that already have the required data in local disks, turning the 'file-based' approach into the fastest one. Also, we confirmed in all of our experiments the hope that, due to locality, there will be much fewer pairs of overlapping β-clusters than in the random case, while enjoying even better load balancing.

5.5.2 Quality of Results

This section intends to answer question $Q2$: How much (if at all) does the parallelism affect the clustering quality? Figure 5.6 presents the quality results obtained by *ParC*, *SnI* and *BoW* over our `Synthetic` dataset with 10 million elements. All tested methods presented top quality, even for large numbers of reducers, like $1,024$. Notice, that the serial execution quality of the plugged-in clustering method is the one obtained when using a single reducer ($r = 1$, extreme left elements in the plot). Similar results were observed with all `Synthetic` datasets.

An interesting observation is that the quality may decrease for small datasets, when using a large number of reducers. The obvious reason is that, in those cases, we are partitioning a small amount of data through a large number of reducers, which actually receive too little data, not enough to represent the patterns existing in the dataset. This fact was confirmed in all of our experiments, and they lead us to recommend at least $\sim 150k$ points per reducer in average. That is, we recommend to set $r \leq \frac{\eta}{150k}$.

According to our experiments, the answer to question $Q2$ is: as long as you have enough data, the clustering quality is barely affected by the parallelism, even for extremely large

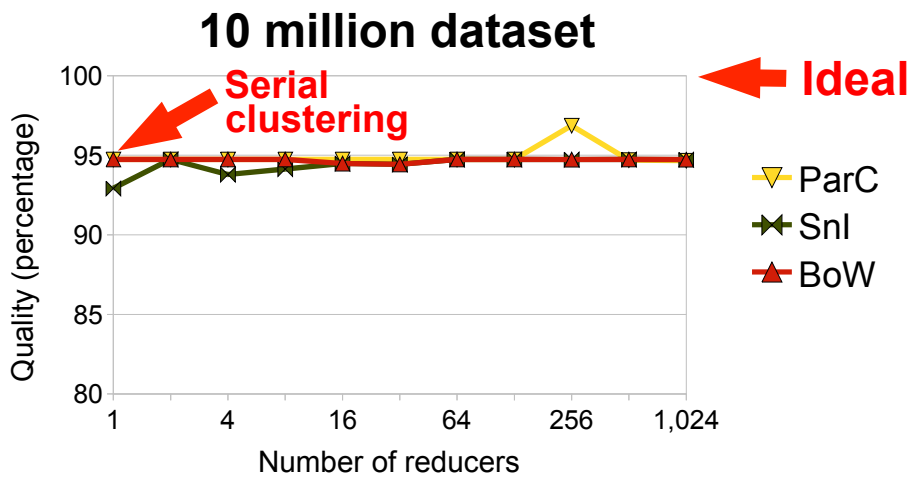


Figure 5.6: All our variants give high quality results. 10 million dataset; quality versus number r of reducers for *ParC*, *SnI* and *BoW*. All methods match the quality of the serial clustering method (top left), for all values of r, like $1,024$. The default, 'file-based' partitioning was used for all cases.

numbers of reducers, such as, $1,024$. Thus, our method *BoW* allowed us to obtain top quality clusters in very little time from all of our very large datasets.

5.5.3 Scale-up Results

This section intends to answer question $Q3$: How does our method scale-up? Scale-up results with different numbers of reducers are in Figure 5.7. Here we used the `TwitterEig` eigenvectors and the `Synthetic` dataset with 100 million points. The plots show X-axes as the number of reducers r, and the Y-axes as the relative performance with n reducers compared to using 1 reducer (`TwitterEig`) or with 4 reducers (`Synthetic`). A fixed number of mappers $m =\sim 700$ was used. The results reported are the average of 10 distinct runs. We picked 4 reducers for our `Synthetic` dataset, as the running time using just one reducer was impractical. Note that our method exhibits the expected behavior: it starts with near-linear scale-up, and then flattens. Similar scale-up results were obtained for all other datasets.

The scale-up results with different data sizes are in Figure 5.8. The `YahooEig` dataset is used. Random samples of the data with increasing sizes, up to the full dataset (1.4 *billion* elements) were generated to perform this experiment. We plot wall clock time versus data size. The wall-clock time shown is the average time for 10 distinct runs. Fixed numbers of reducers and mappers ($r = 128$ and $m =\sim 700$) were used. As shown, our method has the expected scalability, scaling-up linearly with the data size.

It took only ~ 8 *minutes* to cluster the full dataset, which amounts to *200 Gigabytes*. Let us provide some context to this result by characterizing the time taken at different stages in the process: (a) the mappers took 47 seconds to read the data from disks; (b)

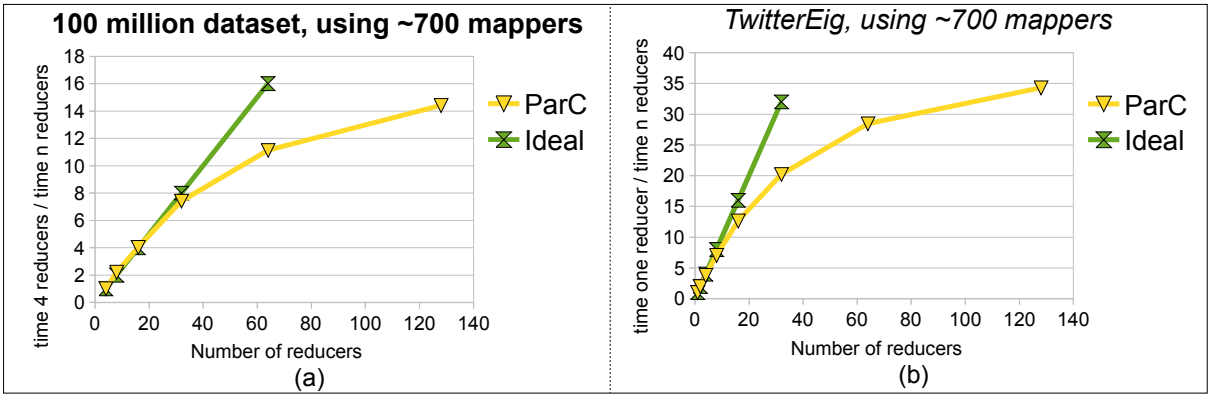


Figure 5.7: Near-linear scale-up. Scale-up results regarding the number of reducers r. Our proposed method exhibits the expected behavior: it starts with near-linear scale-up, and then flattens. Numbers are the average of 10 runs, for real and synthetic data. 100 million dataset (left); `TwitterEig` (right). The X-axes show the number of reducers r, and the Y-axes the relative performance with r reducers compared to using 1 reducer (right) or 4 reducers (left), in lin-lin scales. Using one reducer in the left case requires prohibitively long time. Number of mappers: $m =\sim 700$. The default, 'file-based' partitioning was used for all cases.

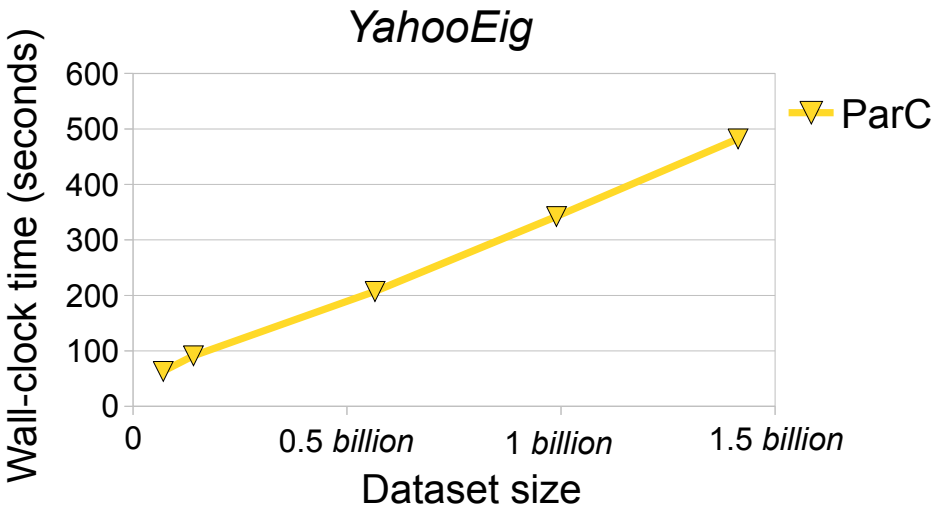


Figure 5.8: Scale-up results: our method is linear on the dataset size. Wall-clock time (average of 10 runs) versus data size in lin-lin scales. Random samples from `YahooEig`, up to the full dataset (1.4 *billion* elements). Fixed number of reducers and mappers ($r = 128$ and $m =\sim 700$). The default, 'file-based' partitioning was used.

65 seconds were taken to shuffle the data; and (c) the reduce stage took 330 seconds. To estimate the time taken by the serial method in item (c), we clustered a random sample of the `YahooEig` dataset, of size $\frac{F_s}{r} = \frac{0.2\ TB}{128}$, by running the plug-in on a single machine (one core), similar to the ones of the used cloud of computes. The serial clustering time

was 192 seconds. This indicates that the plug-in took ∼ 43% of the total time. Similar scale-up results were obtained for all other datasets.

5.5.4 Accuracy of our Cost Equations

Here we illustrate the accuracy of our cost formulas, (Equations 5.4 and 5.5) from Section 5.3, and the ability of *BoW* to choose the correct alternative.

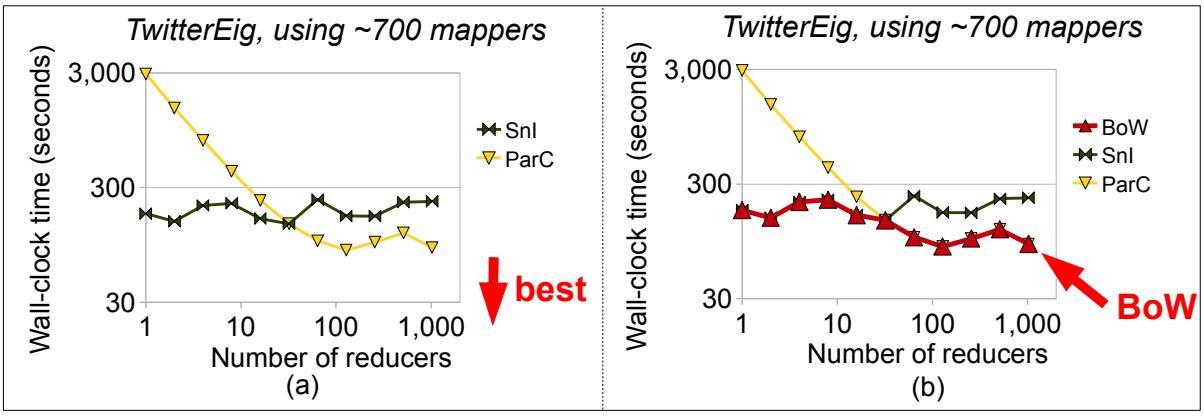


Figure 5.9: *BoW* wins. Results for real data from Twitter. Wall-clock time versus number of reducers in log-log scale. ∼ 700 `MapReduce` mappers were used for all runs. Left: *ParC* (yellow down-triangles) and *SnI* (green butterflies). The latter uses our *sampling-and-ignore* idea; Right: the same, *including* our proposed *BoW* (in red up-triangles). *BoW* achieves the best of both worlds, using cost-based optimization to pick the winning strategy and good parameters for it, and thus practically over-writes the corresponding curve on the graph.

Figure 5.9 shows an example of *BoW*'s results on the `TwitterEig` dataset. It plots the wall-clock-time (average of 10 runs) versus the number of reducers, in log-log scales. Figure 5.9(a) shows the results for *ParC*, in yellow down-triangles, and *SnI*, in green 'butterfly' glyphs. The latter uses our proposed *sampling-and-ignore* idea. Notice that there is no universal winner, with a cross-over point at about 30 machines for this setting. Figure 5.9(b) shows exactly the same results, this time including the wall-clock time of our proposed *BoW*, in red up-triangles. Notice that *BoW* locks onto the best of the two alternatives. The reason for its success is our cost-estimation formulas (Eq. (5.4) and (5.5)), which help *BoW* to pick the best alternative and set good parameters for the chosen environment, while requiring nimble computational effort. Furthermore, notice that the two curves in Figure 5.9(a) intersect at a narrow angle, which means that the optimal curve has a smooth plateau, and thus the cost is rather robust with respect to small variations of the environment parameters.

Figure 5.10 details the results for the Twitter data and also reports results for the `Synthetic` (100 million points) dataset, in the left and right columns, respectively. The six plots give the wall-clock times (average of 10 runs) versus the number of reducers r, in

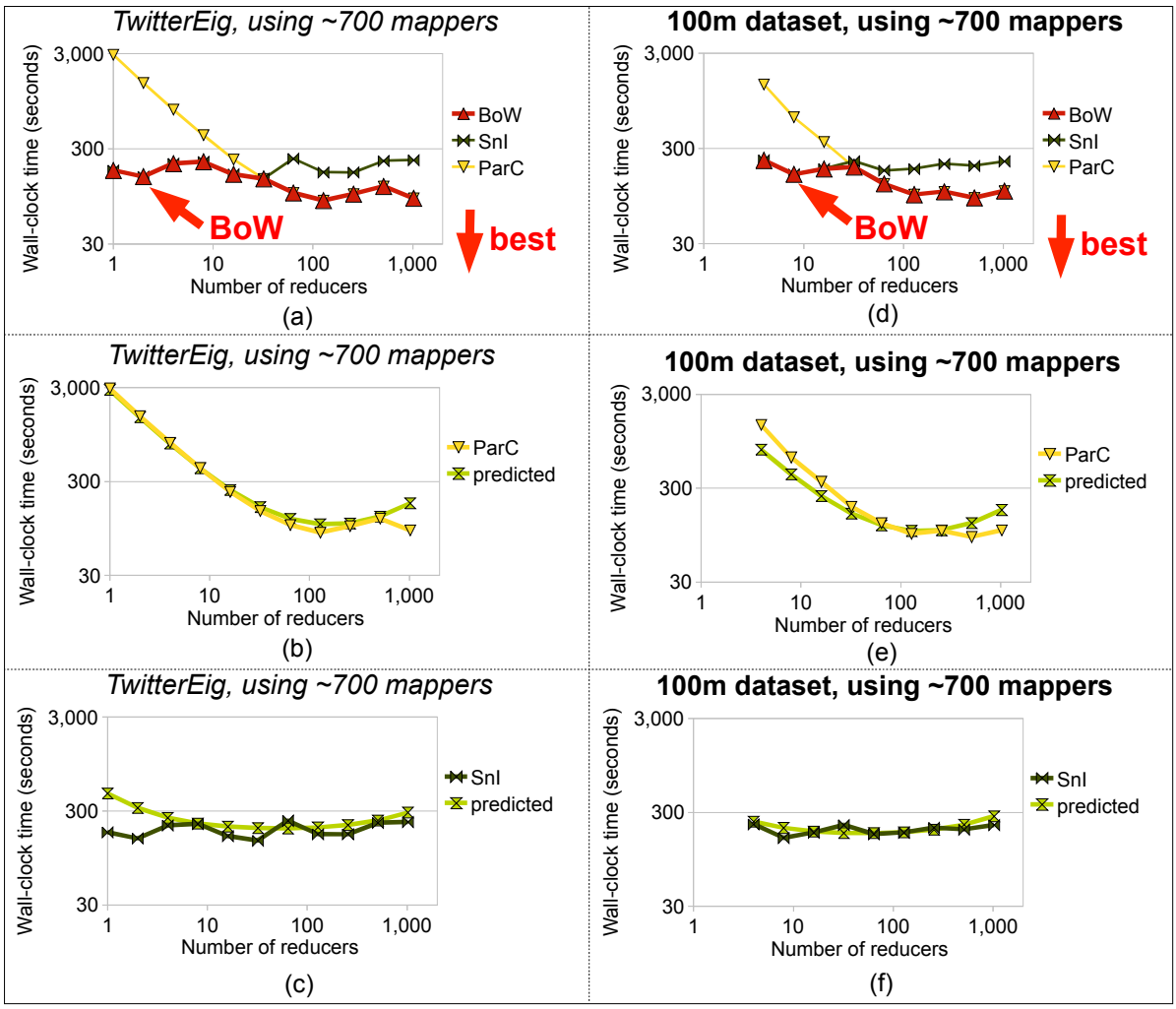


Figure 5.10: *BoW* indeed achieves the *Best of both Worlds*. *BoW*'s results on the `TwitterEig` (left) and on the `Synthetic` 100 million (right) datasets. Time (average of 10 runs) versus number of reducers in log-log scale. m is ~ 700 for all runs. Top line: illustration of *BoW*'s ability to pick the winner. Results for *ParC* (yellow down-triangles), for *SnI* (green butterflies) and for our proposed *BoW* (red up-triangles). Notice that *BoW* achieves the best of both worlds, consistengly choosing the winning strategy, and thus practically over-writing the corresponding curve on those graphs. Bottom two rows: illustration of the accuracy of our Equations 5.4 and 5.5 for *ParC* and *SnI* respectively. In all cases, the green hour-glass shapes stand for our formulas; notice how close they are to the actual measurements (yellow triangles, and dark-green butterfly shapes, respectively).

log-log scales. The top row ((a) and (d)) shows that *BoW*, in red up-triangles, consistently picks the winning strategy among the two options: *ParC* (yellow down-triangles) and *SnI* (dark-green butterflies). For both datasets, *BoW* gives results so close to the winner, that its curve practically overwrites the winner's curve; the only overhead of *BoW* is the CPU time required to run the cost equations, which is obviously negligible.

The next two rows of Figure 5.10 illustrate the accuracy of our cost formulas. Light-green hour-glasses indicate our theoretical prediction; yellow triangles stand for *ParC* in the middle row, and dark-green butterflies stand for *SnI* in the bottom row. Notice that the theory and the measurements usually agree very well. All other datasets provided similar results.

5.6 Conclusions

Given a *very large* moderate-to-high dimensionality dataset, how could one cluster its points? For data that do not fit even on a single disk, parallelism is mandatory. In this chapter we explored it using `MapReduce` for clustering huge datasets. Specifically, we presented *BoW*, the second knowledge discovery algorithm designed in this Ph.D. work. The main contributions of *BoW* are:

1. *Algorithm design and analysis*: We proposed *BoW* and carefully derived its cost functions, which allow to perform the automatic, dynamic trade-off between disk delay and network delay;

2. *Effectiveness, scalability and generality*: We showed that *BoW* has many desirable features: it can use almost any serial method as a plug-in (the only requirement: clusters described by hyper-rectangles), it uses no user defined parameters (due to our defaults), it matches the clustering quality of the serial algorithm, and it has near-linear scale-up;

3. *Experiments*: We report experiments on both real and synthetic data including *billions* of points, using up to $1,024$ cores in parallel. To the best of our knowledge, the *Yahoo! web* is the *largest real dataset ever reported* in the database clustering literature for moderate-to-high dimensionality data. *BoW* clustered its *200 Gigabytes* in only 8 minutes, using 128 cores. We also report experiments that used $1,024$ cores, also the highest such number published in the clustering literature for moderate-to-high dimensionality data.

The next chapter introduces the third and last data mining algorithm developed during this Ph.D. work.

Chapter

6

QMAS

In this chapter, we use the background knowledge provided by the clustering algorithms designed in this Ph.D. work to focus on two distinct data mining tasks – the tasks of labeling and summarizing large sets of complex data. Given a large collection of complex objects, very few of which have labels, how can we guess the labels of the remaining majority, and how can we spot those objects that may need brand new labels, different from the existing ones? This chapter provides answers to these questions. Specifically, we present *QMAS*, the third and last algorithm designed in this Ph.D. work, which is one fast and scalable solution to the problem of automatically analyzing, labeling and understanding large collections of complex objects.

6.1 Introduction

The problem of automatically analyzing, labeling and understanding large collections of complex objects appears in numerous fields. One example application is related to satellite imagery, involving a scenario in which a topographer wants to analyze the terrains in a collection of satellite images. We assume that each image is divided into tiles (say, 16x16 pixels). Such a user would like to label a small number of tiles ('Water', 'Concrete' etc), and then the ideal system would automatically find labels for all the rest. The user would also like to know what strange pieces of land exist in the analyzed regions, since they may indicate anomalies (e.g., de-forested areas, potential environmental hazards, etc.), or errors in the data collection process. Finally, the user would like to have a few tiles that best represent each kind of terrain.

Figure 6.1 illustrates the problem on the example application of satellite images. It shows an example image from the city of Annapolis, MD, USA[1], decomposed into $1,024$ (32x32) tiles, very few (4) of which were manually labeled as "City" (red), "Water" (cyan), "Urban Trees" (green) or "Forest" (black). With this input set, we want to automatically assign the most appropriate labels to the unlabeled tiles, and provide a summarized description of the data by finding clusters of tiles, the N_R best representatives for the data patterns and the top-N_O outlier tiles.

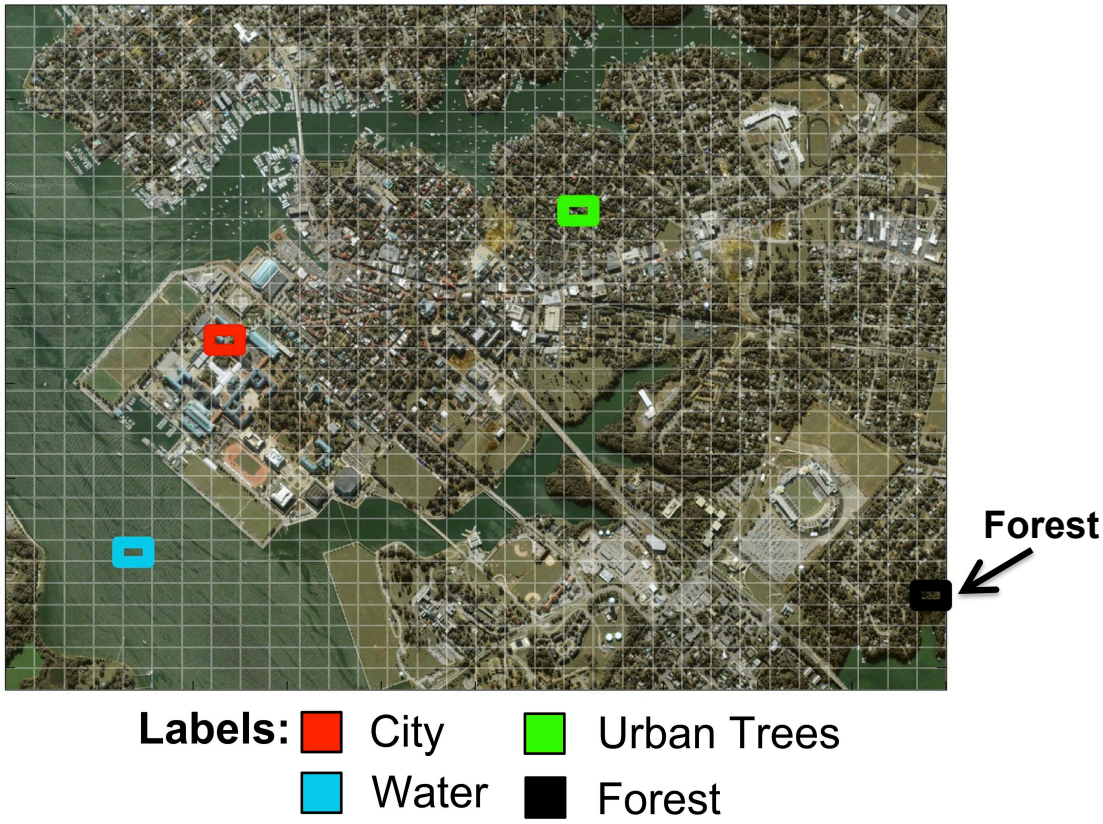


Figure 6.1: One example satellite image of Annapolis (MD, USA), divided into $1,024$ (32x32) tiles, only 4 of which are labeled with keywords, "City" (red), "Water" (cyan), "Urban Trees" (green) or "Forest" (black) (best viewed in color).

Similar requirements appear in several other settings that involve distinct types of complex data, such as, social networks, and medical image or biological image applications. In a social network, one user wants to find other users that share similar interests with himself/herself or with his/her contacts, while the network administrator wants to spot a few example users that best represent both the most typical and the most strange types of users. In medicine, a doctor wants to find tomographies or x-rays similar to the images of his/her patient's as well as a few examples that best represent both the most typical and the most strange image patterns. In biology, given a collection of fly

[1] The image is publicly available at 'geoeye.com'.

embryos or protein localization patterns or cat retina images and their labels, we want a system to answer the same types of questions.

Our goals are summarized in two research problems:

Problem 1 *low-labor labeling (L3)* – **Given** *an input set* $I = \{I_1, I_2, I_3, ..., I_{N_I}\}$ *of* N_I *complex objects,* very few *of which are labeled with keywords,* **find** *the most appropriate labels for the remaining ones.*

Remark: we have coined the term *"low-labor labeling (L3)"* in this Ph.D. work.

Problem 2 *mining and attention routing* – **Given** *an input set* $I = \{I_1, I_2, I_3, ..., I_{N_I}\}$ *of* N_I *partially labeled complex objects,* **find** *clusters, the* N_R *objects from* I *that best represent the data patterns and the top-*N_O *outlier objects.*

In this chapter we propose *QMAS*: ***Q****uerying,* ***M****ining* ***A****nd* ***S****ummarizing Multi-dimensional Databases.* Our method is a fast ($O(N)$) solution to the aforementioned problems. Its main contributions, supported by experiments on real satellite images, spanning up to more than 2.25 *Gigabytes*, are summarized as follows:

1. **Speed**: *QMAS* is fast and it scales linearly on the database size, being up to 40 times faster than top related works on the same subject;
2. **Quality**: It can do *low-labor labeling (L3)*, providing results with better or equal quality when compared to top related works;
3. **Non-labor intensive**: It works even when we are given very few labels – *it can still extrapolate from tiny sets of pre-labeled data*;
4. **Functionality**: Contrasting to other methods, *QMAS* encompasses extra mining tasks such as clustering and outlier and representatives detection as well as summarization. It also spots data objects that potentially require new labels;

6.2 Proposed Method

This section describes *QMAS*, our proposed solution to the problems of *low-labor labeling (L3)* (Problem 1) and *mining and attention routing* (Problem 2). It assumes that a feature extraction process is first applied over the input set of complex objects I, turning the set into a multi-dimensional dataset. Next, we describe our proposal in detail.

6.2.1 Mining and Attention Routing

In this section we present our solution to the problem of *mining and attention routing* (Problem 2). The general idea is as follows: First we do clustering on the input set of

complex objects I; then we find (a) the subset of objects $R = \{R_1, R_2, R_3, ..., R_{N_R}\} \mid R \subseteq I$ that best represent I, and (b) the array with the top-N_O outlier objects $O = (O_1, O_2, O_3, ..., O_{N_O}) \mid O_o \in I, \forall\ 1 \leq o \leq N_O$, sorted according to the confidence degree of each object O_o be an outlier. Algorithm 10 provides a general view of our solution to Problem 2. The details are described as follows.

Algorithm 10 : *QMAS-mining.*

Input: input set of complex objects I;
desired number of representatives N_R;
desired number of top outliers N_O.

Output: clustering result C;
set of representatives $R = \{R_1, R_2, R_3, ..., R_{N_R}\} \mid R \subseteq I$;
top-N_O outliers $O = (O_1, O_2, O_3, ..., O_{N_O}) \mid O_o \in I, \forall\ 1 \leq o \leq N_O$,
in sorted order.

1: do clustering on I, let the result be C;
2: R = random N_R complex objects from I;
3: $error = E_{QMAS}(I, R)$; // from Equation 6.2
4: **repeat**
5: improve the representatives in R;
6: $old\_error = error$;
7: $error = E_{QMAS}(I, R)$; // from Equation 6.2
8: **until** $error == old\_error$
9: O = the N_O complex objects from I worst represented by R, sorted according to the confidence degree of each object O_o in O be an outlier;
10: **return** C, R and O;

Clustering

The clustering step over the input set of objects I is performed using our algorithm *Halite*. As described in Chapter 4, *Halite* is a fast and scalable clustering algorithm well-suited to spot clusters in large collections of complex data. Here, we ignore the merging step of *Halite* (Algorithm 3 from Chapter 4) and use the β-clusters found so far as the final clustering result, considering that each object can belong to one or more clusters with equal probabilities. This strategy is used by *QMAS* to find clusters in the input set of complex objects I.

Finding Representatives

Now we focus on the problem of selecting a set $R = \{R_1, R_2, R_3, ..., R_{N_R}\} \mid R \subseteq I$ of objects with cardinality $N_R = |R|$ to represent the input set of complex objects I. First, we discuss the desirable properties for a set of representatives, then we work on two possible approaches to actually find the representatives.

An appropriate set of representatives R for the objects in I must have the following property: *there is a large similarity between every object $I_i \in I$ and its most similar*

representative R_r. Obviously, the set of representatives that best represent I is the full set of complex objects, $N_R = N_I \Rightarrow R = I$. In this case, the similarity is maximal between each object I_i and its most similar representative R_r, which is the complex object itself, $I_i = R_r$. However, for $N_R < N_I$, how should one evaluate the quality of a given set of representatives?

A simple way to evaluate the quality of a collection of representatives is to sum the squared distances between each object I_i and its closest representative R_r. This gives us an error function that should be minimized in order to achieve the best set of representatives R for the input set of complex objects I. It is not a coincidence that this is the same error function minimized by the classic clustering algorithm K-Means, which is formally defined by the following equation.

$$E_{KM}(I, R) = \sum_{I_i \in I} MIN\{\|I_i - R_r\|^2 \mid R_r \in R\} \tag{6.1}$$

In the equation, $\|I_i - R_r\|$ is the distance between the objects I_i and R_r, and MIN is a function that returns the minimum value within its input set of values. Without loss of generality, the Euclidean distance L_2 is considered here.

Based on this idea, when we ask K-Means for N_R clusters, the centroids of the clusters are good indicators of the data space positions where we should look for representatives. Then, we have a set of representatives for K-Means by: (i) finding, for each centroid, the data object I_i from I that is the closest one to the respective centroid, and; (ii) defining R to be the complete set of objects found.

Figure 6.2a shows a synthetic dataset containing three clusters. The clusters and their sizes follow skewed distributions. The sizes are $30,000$, $3,000$ and $1,000$ for the clusters in the bottom left, bottom right and top of the data space respectively. Additionally, 500 points are uniformly distributed through the data space in order to represent noise.

Figure 6.2b shows the representatives selected for our example dataset by using K-Means and considering N_R as 10 (top) and 20 (bottom). The presented results are the best ones over 50 runs, i.e., the ones with the smallest error, computed by Equation 6.1. Notice that, in all cases, the representatives selected are excessively concentrated in the bottom right cluster, the biggest one, while the other two clusters are poorly represented, having only a few representatives each. These results indicate that K-Means is sensitive to the data distribution, commonly presenting unsatisfactory results for representative picking, especially for skewed data distributions.

We propose to ***use*** the traditional K-Harmonic Means clustering algorithm in *QMAS*, since it is almost insensitive to skewed distributions, data imbalance, and bad seed initialization. Thus, it provides to us a robust way to look for representatives, again by asking for N_R clusters and, for each cluster, picking as a representative the object I_i

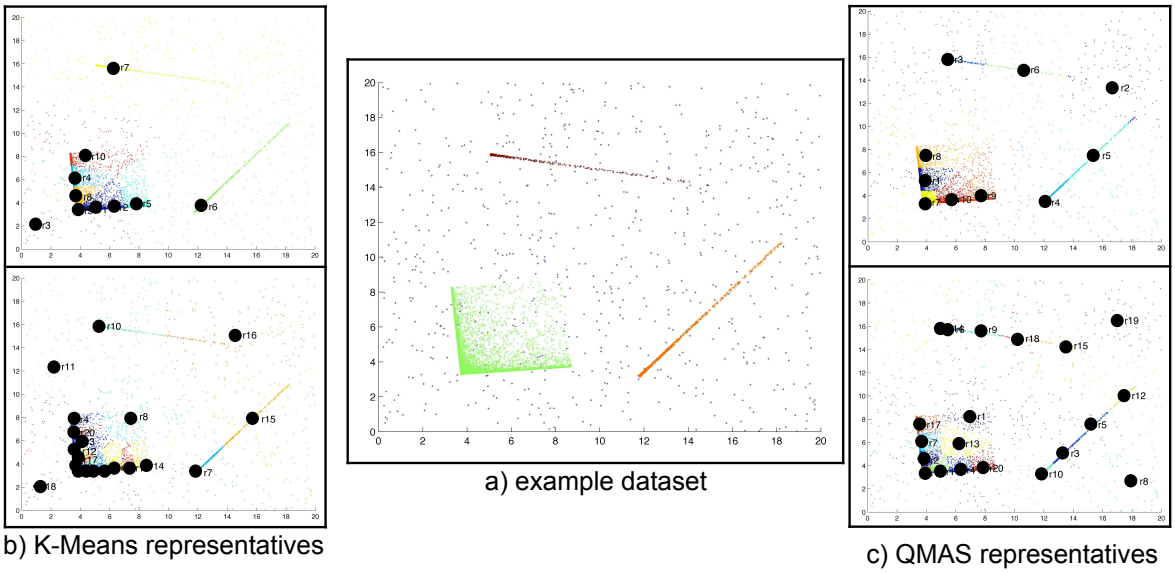


Figure 6.2: Examples of representatives spotted in synthetic data. Center: example dataset with 3 clusters following skewed distributions; Borders: representatives selected by K-Means (left) and *QMAS* (right), for $N_R = 10$ (top) and 20 (bottom). These are the results with the smallest error over 50 runs.

from I that is the closest object to the respective cluster's centroid. The minimization error function is presented as follows.

$$E_{QMAS}(I, R) = \sum_{I_i \in I} HAR\{\|I_i - R_r\|^2 \mid R_r \in R\} \tag{6.2}$$

$$= \sum_{I_i \in I} \frac{N_R}{\sum_{R_r \in R} \frac{1}{\|I_i - R_r\|^2}}$$

In the equation, $\|I_i - R_r\|$ is the distance between the data objects I_i and R_r, and HAR is a function that returns the harmonic mean of its input values. The Euclidean distance L_2 is used once more, without loss of generality.

Figure 6.2c shows the representatives selected by *QMAS* for our example dataset, again considering N_R as 10 (top) and 20 (bottom). Once more, the presented results are the best ones over 50 runs, this time considering the error function in Equation 6.2. Notice that the representatives chosen are now well distributed among the three clusters, providing to the user a summary that better describes the data patterns.

Finding the Top-N_O Outliers

The final task related to the problem of *mining and attention routing* is to find the top-N_O outliers $O = (O_1, O_2, O_3, ..., O_{N_O}) \mid O_o \in I, \forall\ 1 \leq o \leq N_O$, for the input set of complex objects I. In other words, O contains the N_O objects of I that diverge the most from the

main data patterns. The outliers must be sorted in such a way that we identify the top $1st$ outlier, the top $2nd$ outlier and so on, according to the confidence degree of each one being an outlier.

In order to achieve this goal, we take the representatives found in the previous section as a base for the outliers definition. Assuming that a set of representatives R is a good summary for I, the N_O objects from I that are the worst represented by R are said to be the top-N_O outliers. Let us consider again the error function in Equation 6.2. Notice that the minimized error is the summation of the individual errors for each object $I_i \in I$, where the individual error with respect to I_i is given by the following equation.

$$IE_{QMAS}(I_i, R) = \frac{N_R}{\sum_{R_r \in R} \frac{1}{\|I_i - R_r\|^2}} \tag{6.3}$$

This equation is the harmonic mean of the squared distances between one object I_i and each one of the representative objects in R. The object $I_i \in I$ with the greatest individual error is the one that is worst represented by R, which is the object considered to be the top $1st$ outlier of I. The top $2nd$ outlier is the object with the second greatest individual error, and so on. In this way, *QMAS* defines the array O containing the top-N_O outliers, in sorted order.

Figure 6.3 shows the top-10 outliers that *QMAS* found for the example dataset in Figure 6.2a, considering $N_O = 10$ and $N_R = 10$. As we can see, the top outliers are actually the most extreme cases for this data.

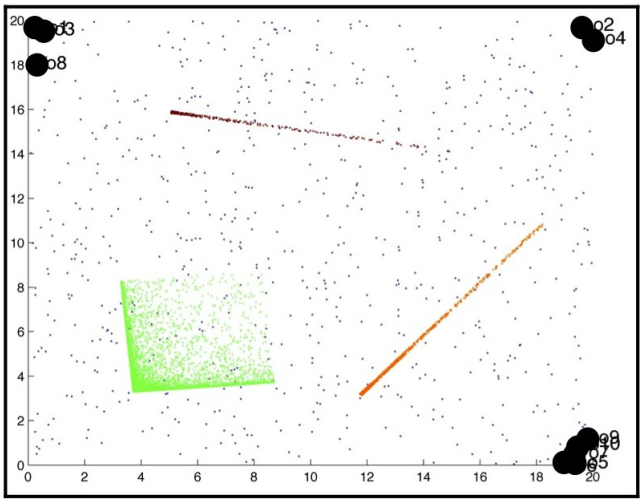


Figure 6.3: Top-10 outliers for the example dataset in Figure 6.2a, considering the QMAS representatives from Figure 6.2c (top). As we can see, the top outliers are actually the most extreme cases for this data.

6.2.2 Low-labor Labeling (L3)

In this section we discuss our solution to the problem of *low-labor labeling (L3)* (Problem 1). That is, given the input set $I = \{I_1, I_2, I_3, ..., I_{N_I}\}$ of N_I complex objects, very few of which are labeled with keywords, we want to find the most appropriate labels for the remaining ones. In order to solve this problem, we first represent the input complex objects and labels as a graph G, which we name as the *Knowledge Graph*. Then, random walks with restarts over G allow us to find the most suitable labels for each unlabeled object. Algorithm 11 provides a general view of our solution to Problem 1. The details are described as follows.

Algorithm 11 : *QMAS-labeling.*

Input: input collection of complex objects I;
collection of known labels L;
restart probability c;
clustering result C. // from Algorithm 10

Output: full set of labels LF.

1: use I, L and C to build the *Knowledge Graph* G;
2: **for** each unlabeled object $I_i \in I$ **do**
3: do random walks with restarts in G, using c and always restarting the walk from vertex $V(I_i)$;
4: compute the affinity between each label in the collection L and the object I_i. Let L_l be the label with the biggest affinity to I_i;
5: set in LF: L_l is the appropriate label for object I_i;
6: **end for**
7: **return** LF;

G is a tri-partite graph composed of a set of vertexes V and a set of edges X, i.e., $G = (V, X)$. To build the graph, the input sets of complex objects I and known labels L are used, as well as the clustering results obtained in Section 6.2.1. V consists of one vertex for each data object, for each cluster, and for each label. The edges link complex objects to their respective clusters and labels. In our notation, $V(I_i)$ and $V(L_l)$ represent the vertexes of G related to object I_i and to label L_l respectively. Provided the clustering results for the objects in I, the process of building G is very simple, having linear time and memory complexities relative to the number of objects, labels and clusters.

Figure 6.4 shows the *Knowledge Graph* G for a small example dataset with seven complex objects, two labels, and three clusters. In this figure, data objects, labels, and clusters are represented by nodes with shape of squares, circles, and triangles, respectively. The graph indicates, for example, that cluster C_1 contains the objects I_1, I_2, and I_3. Object I_3 also belongs to cluster C_2 in this setting. In addition, the graph shows that object I_1 has the known label L_1, while the objects I_4 and I_7 have the known label L_2.

In order to look for the most suitable label for an unlabeled complex object I_i, we use random walks with restarts over graph G. This process is described as follows: a random

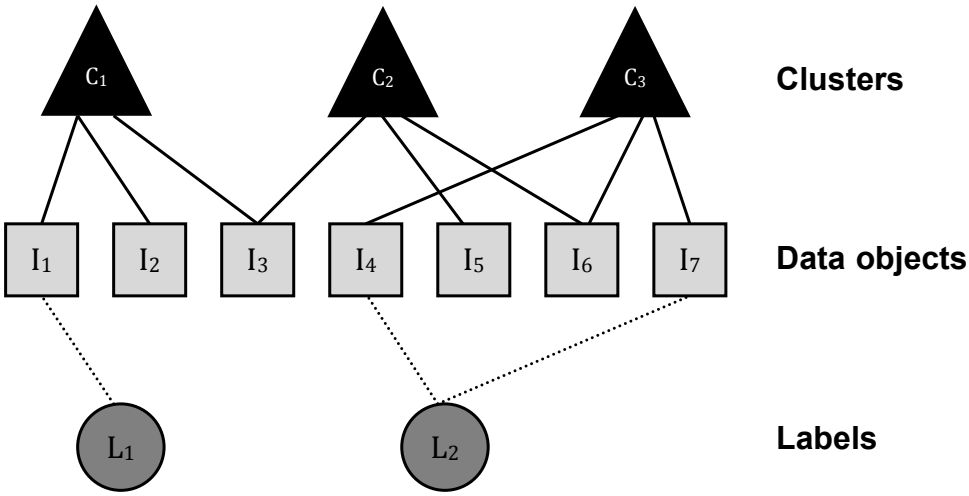


Figure 6.4: The *Knowledge Graph G* for a toy dataset. Nodes with shape of squares, circles, and triangles represent data objects, labels, and clusters respectively. The edges link objects to their corresponding clusters and known labels.

walker starts from vertex $V(I_i)$. At each step, the walker either goes back to the initial vertex $V(I_i)$, with probability c, or to a randomly chosen vertex that shares an edge with the current vertex, with probability $1 - c$. The value of c is user defined, and may be determined by cross validation. It is set to 0.15 in our experiments. The probability of choosing a neighboring vertex is proportional to the degree of that vertex, *i.e.*, the walker favors smaller clusters and more specific labels in this process. The affinity between I_i and a label L_l is given by the steady state probability that our random walker will find itself at vertex $V(L_l)$, always restarting from $V(I_i)$. Finally, the label L_l with the largest affinity with object I_i is considered to be the most suitable label for I_i.

The intuition behind this procedure is that the steady state probability that a random walker will find itself in vertex $V(L_l)$, always restarting the walk from vertex $V(I_i)$, is a way to measure the closeness between $V(I_i)$ and $V(L_l)$. If the computed probability is high, the vertexes are probably linked by short paths. On the other hand, if the probability is low, it is likely that no short path links them.

This idea can be better understood through our example in Figure 6.4. Let us assume that we want to find the most appropriate label for object I_2. There is a high probability that a random walker will reach L_1, always restarting the walk from I_2, mainly because there exists a three-step path linking I_2 to L_1. On the other hand, there is a lower probability that the walker will find itself at L_2, always restarting the walk from I_2, especially because the shortest path between I_2 and L_2 has seven steps. This fact leads us to conclude that, in our example, the most appropriate label for I_2 is L_1.

6.3 Experimental Results

This section presents the experiments performed to test the *QMAS* algorithm. To validate our method, we decided to analyze large collections of satellite images. First we report *QMAS* results on our initial example from the introductory Section 6.1. Then we report the experiments performed to support our contributions stated in that section, regarding Speed, Quality, Non-labor intensive capability, and Functionality.

Three sets of real satellite images were used in the experiments. They are summarized in Table 6.1 and described as follows:

- *GeoEye*[2] – this public dataset contains 14 high-quality satellite images in the jpeg format extracted from famous cities around the world, such as the city of Annapolis (MD, USA), illustrated in Figure 6.1. The total data size is about 17 MB. We divided each image into equal-sized rectangular tiles and the entire dataset contains $14,336$ tiles, from which Haar wavelets features in 2 resolution levels were extracted, plus the mean value of each band of the tiles;

- *SAT1.5GB* – this proprietary dataset contains 3 large satellite images of around 500 MB each in the GeoTIFF lossless data format. The total data size is about 1.5 *Gigabytes*. Each image was divided into equal-sized rectangular tiles. The 3 images combined form a set of $721,408$ tiles, from which Haar wavelets features in 2 resolution levels were extracted, plus the mean value of each band of the tiles;

- *SATLARGE* – this proprietary dataset contains a pan QuickBird image of size 1.8 *Gigabytes*, and its matching 4-band multispectral image of size 450 MB. These images were combined and 2,570,055 hexagonal tiles generated, from which we extracted features. Mean, variance, moments and GBT texture features [Gibson and Lucas, 1982] were extracted from each tile. The final feature set of a tile comprises a 30-dimensional vector. The details about this process are found at [Cordeiro et al., 2010a].

Dataset	Number of Tiles	Data Size
GeoEye	$14,336$	17 MB
SAT1.5GB	$721,409$	1.5 GB
SATLARGE	2,570,055	2.25 GB

Table 6.1: Summary of datasets. MB: Megabytes; GB: Gigabytes.

The experimental environment is a server with Fedora® Core 7 (Red Hat, Inc.), a 2.8 GHz core and 4GB of RAM. We compared *QMAS* with one of the best competitors: the

[2] The data is publicly available at: 'geoeye.com'.

GCap method that we described in Section 2.4 from Chapter 2. GCap was implemented in two versions with different nearest neighbor finding algorithms: one version uses the basic quadratic algorithm (GCap) and one other version spots approximate nearest neighbors (GCap-ANN), using the ANN Library[3]. The number of nearest neighbors is set to seven in all experiments. All three approaches share the same implementation of the random walks algorithm using the *power iteration method* [Golub and Van Loan, 1996], with the restart parameter set as $c = 0.15$.

6.3.1 Results on our Initial Example

This section reports the results obtained for our example satellite image from Figure 6.1, presented in the introductory Section 6.1. The image, also shown in Figure 6.5(a), refers to the city of Annapolis, MD, USA. As in the introductory example, we decomposed it into $1,024$ (32x32) tiles, *only four* of which were manually labeled as "City" (red), "Water" (cyan), "Urban Trees" (green) or "Forest" (black). From each tile Haar wavelets features in 2 resolution levels were extracted, plus the mean value of each band of the tile.

Figure 6.5(b) shows the solution proposed by *QMAS* to the problem of *low-labor labeling (L3)* (Problem 1) on the example satellite image. Notice two remarks: (a) the vast majority of the tiles were correctly labeled and (b) there are few outlier tiles marked in yellow that *QMAS* judges as too different from the labeled ones (i.e., there is no path linking the image and one label in the *Knowledge Graph*), and thus are returned to the user as outliers that potentially deserve a new label of their own. Closer inspection shows that the outlier tiles tend to be on the border of, say, "Water" and "City" (because they contain a bridge).

Our solution to the problem of *mining and attention routing* (Problem 2) on the example image is presented in Figure 6.5c and Figure 6.5d. *QMAS* pointed out the 3 tiles that best represent the data patterns and the top-2 outliers. Notice that the representatives actually cover the 3 major keywords ("City", "Urban Trees", and "Water"), while the top outliers are hybrid tiles, like the bottom right which is a bridge (both "Water" and "City").

Note that *QMAS* goes even further by summarizing the results: besides the representatives and top outliers, *QMAS* found clusters in the data, ignoring the user-provided labels. This has two advantages. The first is that it indicates to the user what, if any, changes have to be done to the labels: new labels may need to be created (to handle some clusters or outliers), and/or labels may need to be merged (e.g., "Forest" and "Urban trees"), and/or labels that are too general may need to be divided in two or more ("Shallow Water" and "Deep Sea", instead of just "Water"). The second advantage is that these

[3] http://www.cs.umd.edu/m̃ount/ANN/

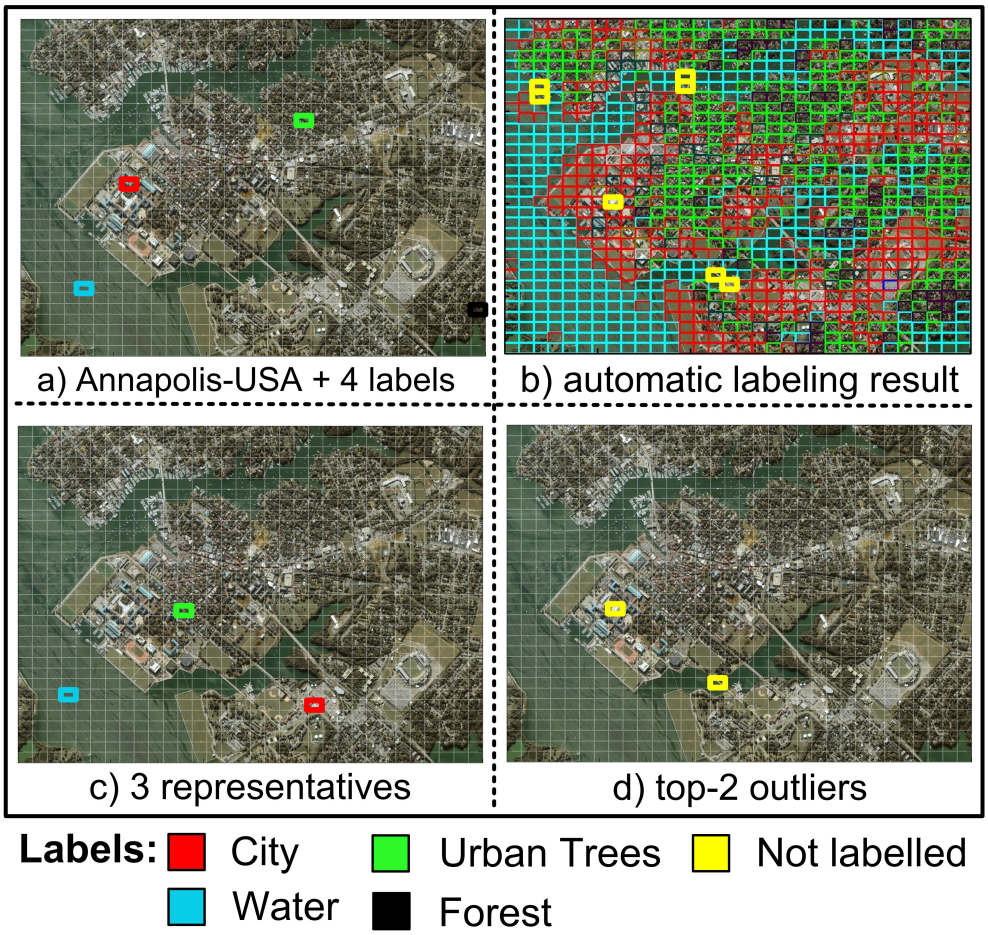


Figure 6.5: Our solution to Problem 1 – *low-labor labeling* and to Problem 2 – *mining and attention routing* on an example satellite image (best viewed in color). Top Left: the input satellite image of Annapolis (MD, USA), divided into 1,024 (32x32) tiles, only 4 of which are labeled with keywords ("City" in red, etc). Top Right: the labels that *QMAS* proposes; yellow indicates outliers. Bottom Left: the 3 tiles that best represent the data, which actually cover the 3 major keywords. Bottom Right: the top-2 outlier tiles, where appropriate labels do not exist (hybrid tiles, like the bottom right which is a bridge = both "Water" and "City").

results can also be used for group labeling, since the user can decide to assign labels to entire clusters rather than labeling individual tiles one at a time.

6.3.2 Speed

This section supports the following claim: *QMAS* is a fast solution to the problems investigated, scaling linearly on the data size, and being several times faster than top related works. Figure 6.6 shows how the tested methods scale with increasing data sizes. Random samples from our *SAT1.5GB* dataset were used. As it can be seen, the log-log curve for QMAS has the slope equal to one, so *QMAS* scales linearly with the input data size, while the slope of log-log curves are 2.1 and 1.5 for GCap and GCap-ANN,

respectively. For the full *SAT1.5GB* dataset, *QMAS* is 40 times faster than GCap-ANN, while running GCap would take hours long (not shown in the figure).

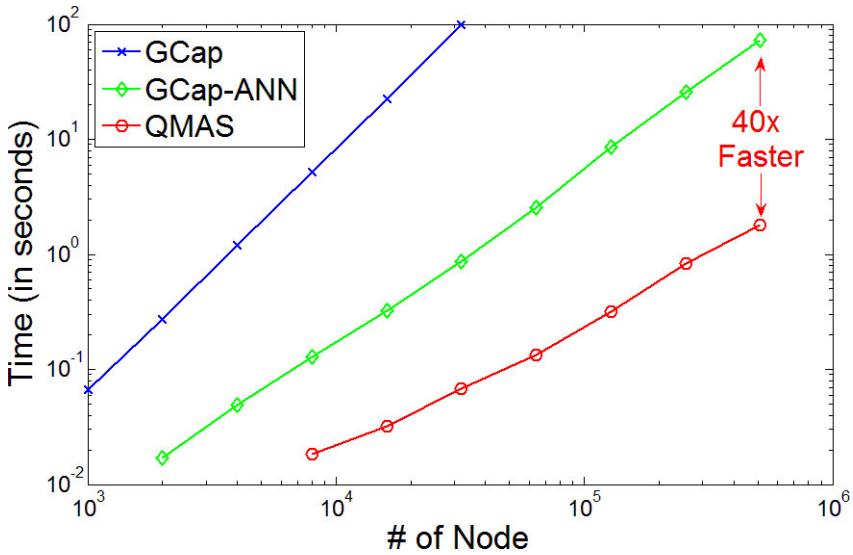


Figure 6.6: Time versus number of tiles for random samples of the *SAT1.5GB* dataset. *QMAS*: red circles; GCap: blue crosses; GCap-ANN: green diamonds. Wall-clock time results are averaged over 10 runs; error bars are too small to be shown.

Notice one *important* remark: as stated in Section 2.4 of Chapter 2, most previous works, including GCap, searches for nearest neighbors in the feature space. This operation is super-linear even with the use of approximate nearest-neighbor finding algorithms. On the other hand, *QMAS* avoids the nearest neighbor searches by using clusters to connect similar image nodes in the *Knowledge Graph*. This approach allows *QMAS* to scale linearly on the data size, being up to 40 times faster than the top competitors.

6.3.3 Quality and Non-labor Intensive

We labeled 256 tiles in the *SAT1.5GB* dataset via manual curation. Some few ground truth labels were randomly selected from each class as the input labels and the remaining ones were used for the quality test. Figure 6.7 illustrates the labeling accuracy for the GCap-ANN and for the *QMAS* approaches in box plots obtained from 10 repetitive runs. As it can be seen, *QMAS* does not sacrifice quality for speed compared with GCap-ANN and it performs even better when the pre-labeled data size is limited. Note that the accuracy of *QMAS* is barely affected by the number of the pre-labeled examples in each label class, when the number of examples given goes above 2, while the quality of GCap-ANN was considerably worse with small sets of pre-labeled examples. The fact that *QMAS* can still extrapolate from tiny sets of pre-labeled data ensures its non-labor intensive capability.

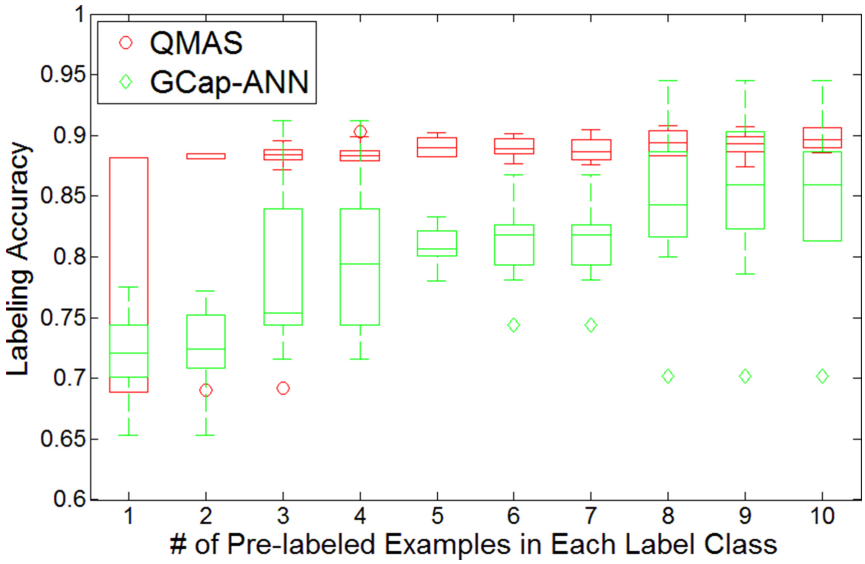


Figure 6.7: Comparison of approaches in box plots – quality versus size of the pre-labeled data. Top left is the ideal point. *QMAS*: red circles; GCap-ANN: green diamonds. Accuracy values of *QMAS* are barely affected by the size of the pre-labeled data. Results are obtained over 10 runs.

6.3.4 Functionality

This section evaluates the following claim: in contrast to the related works, *QMAS* includes other mining tasks such as clustering, detection of top outliers and representatives, besides summarization. In other words, *QMAS* tackles both the problem of *low-labor labeling (L3)* (Problem 1) and the problem of *mining and attention routing* (Problem 2), while the related works address only the former. To evaluate this claim, we analyzed the functionality of our method regarding its ability to spot clusters, representatives and top outliers. The *GeoEye* dataset was used in all experiments reported in this section.

Figure 6.8 shows some screenshots of the clustering results obtained with *QMAS*. Yellow tiles represent outliers. Closer inspection shows that these outlier tiles tend to be on the border of areas like "Water" and "City" (because they contain a bridge). The remaining tiles are colored according to its cluster. As expected, a few tiles belong to more than one cluster, since we do soft clustering in *QMAS*. These tiles were colored according to their first assigned clusters. Notice: the clustering results reported indeed represent the main patterns apparent in the analyzed images.

Finally, Figure 6.9 reports the results obtained by *QMAS* with respect to representatives. $N_R = 6$ representatives are shown, colored according to their clusters. Notice that these few representatives cover the main clusters previously presented in Figure 6.8. Also, these 6 representatives were used as a basis to the detection of top outliers. Figure 6.10 shows the top-3 outliers spotted. By comparing these results with the clusters presented in Figure 6.8, one can notice that the 3 outliers spotted, together with the 6 representatives

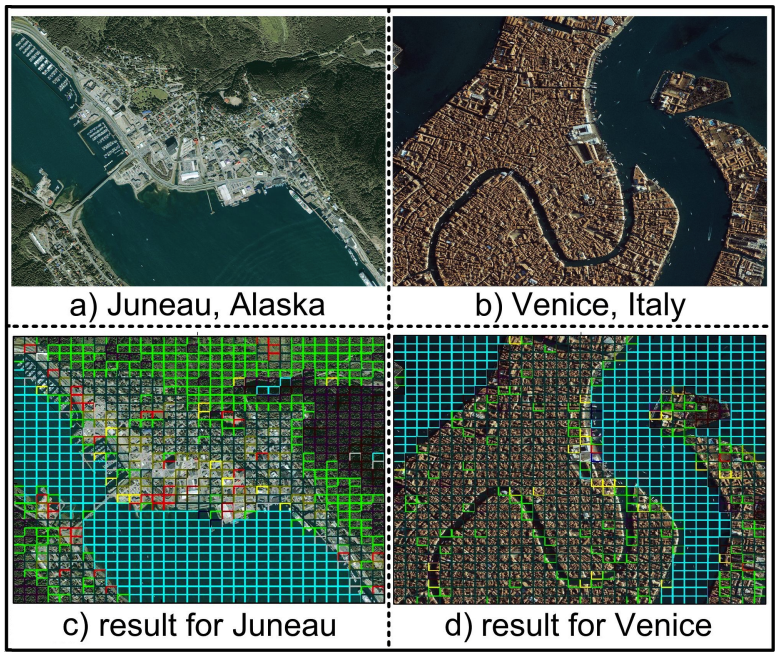


Figure 6.8: Clustering results provided by *QMAS* for the *GeoEye* dataset (best viewed in color). Top: the real satellite images; Bottom: the corresponding results, shown by coloring each tile after its cluster. Yellow tiles represent outliers. Notice that the clusters actually represent the main data patterns.

found (only 9 tiles in total) properly summarize the *GeoEye* dataset, which has more than 14 thousand tiles. This fact illustrates the functionality of our method.

6.3.5 Experiments on the *SATLARGE* dataset

Here we present results for the *SATLARGE* dataset, related to *query by examples* experiments; *i.e.*, given a small set of tiles (examples), manually labeled with one keyword, query the unlabeled tiles to find the ones most likely related to that keyword. Figures 6.11, 6.12, 6.13, 6.14, 6.15, 6.16 and 6.17 illustrate the results obtained for several categories ("Water", "Houses", "Trees", etc) to show that *QMAS* returns high-quality results, being almost insensitive to the kind of tile given as input. Additionally, notice that Figures 6.14 and 6.15 show that the results provided by *QMAS* are good even for tiny sets of pre-labeled data. The number of examples provided vary from as many as ~ 50 examples to as few as *two* examples. Varying the amount of labeled data allowed us to observe how the system responds to these changes. In general, labeling only a small number of examples (even less than five) still leads to pretty accurate results. Finally, notice that correct results often look very different from the given examples, *i.e.*, *QMAS* is able to extrapolate from the given examples to other, correct tiles that do not have significant resemblance to the pre-labeled set.

Figure 6.9: $N_R = 6$ representatives found by *QMAS* for the *GeoEye* dataset, colored after their clusters (best viewed in color). By comparing the representatives to the clusters presented in Figure 6.8, it is easy to see that these few representatives nicely cover the main clusters.

6.4 Conclusions

In this chapter we proposed *QMAS*: ***Q****uerying,* ***M****ining* ***A****nd* ***S****ummarizing Multi-dimensional Databases*, the third and last algorithm designed in this Ph.D. work, which is a fast solution to the problem of automatically analyzing, labeling and understanding large collections of complex objects. The main contributions of the method *QMAS*, supported by experiments on real satellite images spanning up to 2.25 *Gigabytes*, are presented as follows:

1. **Speed**: *QMAS* is a fast solution to the presented problems, and it scales linearly on the database size. It is up to 40 times faster than top related works (GCap) on the same subject;

Figure 6.10: Top-3 outliers found by *QMAS* for the *GeoEye* dataset based on the 6 representatives of Figure 6.9 (best viewed in color). The outlier tiles tend to be on the border of areas like "Water" and "City" (because they contain a bridge). Notice that these 3 outliers together with the 6 representatives of Figure 6.9, only 9 tiles in total, nicely summarize the *GeoEye* dataset, which contains more than 14 thousand tiles.

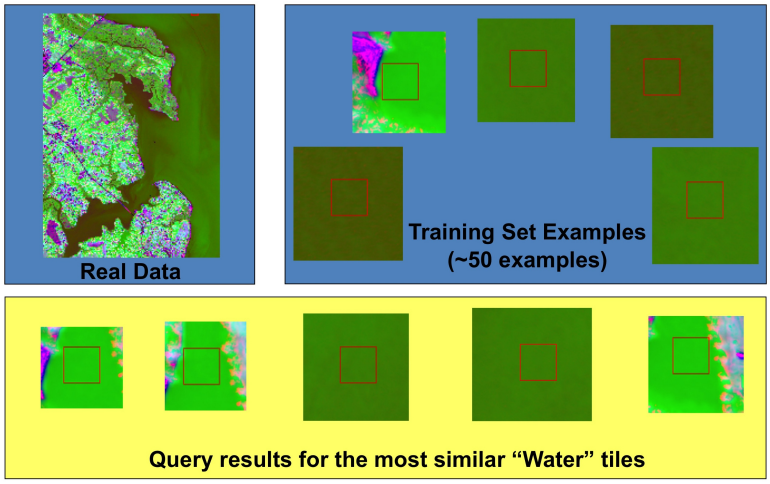


Figure 6.11: Example with water: labeled data and the corresponding results of a query for "Water" tiles (best viewed in color).

2. **Quality**: It can do *low-labor labeling (L3)*, providing results with accuracy better than or equal to the accuracy of the related works;

3. **Non-labor intensive**: It works even when we are given very few labels – *it can still extrapolate from tiny sets of pre-labeled data*;

4. **Functionality**: In contrast to the other methods, *QMAS* spots data objects that potentially require new labels, and encompasses other mining tasks such as clustering, outlier and representatives detection, as well as summarization;

The next chapter presents the conclusions of this Doctoral dissertation and ideas for future works.

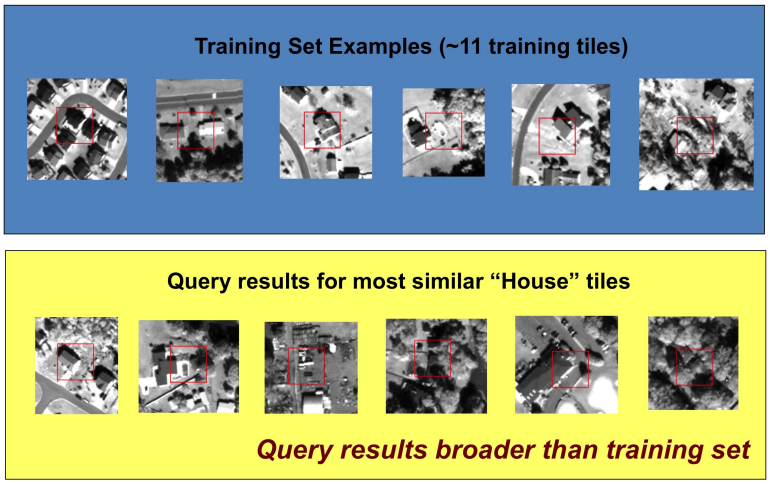


Figure 6.12: Example with houses: labeled data and the corresponding results of a query for "House" tiles (best viewed in color).

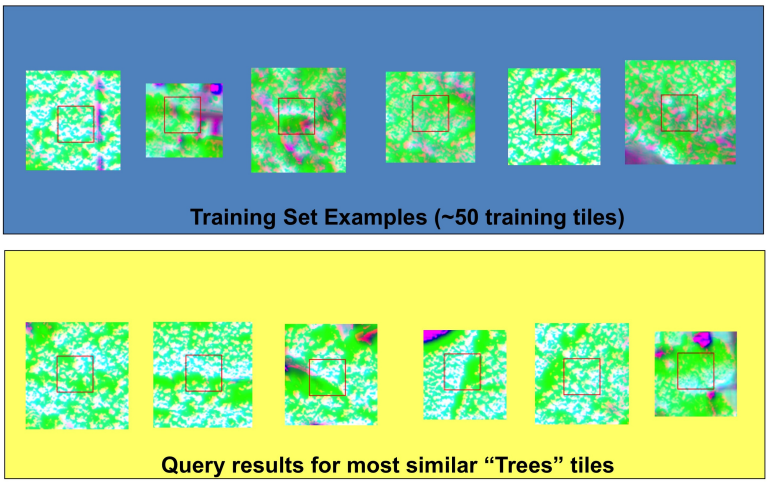


Figure 6.13: Example with trees: labeled data and the corresponding results of a query for "Trees" tiles (best viewed in color).

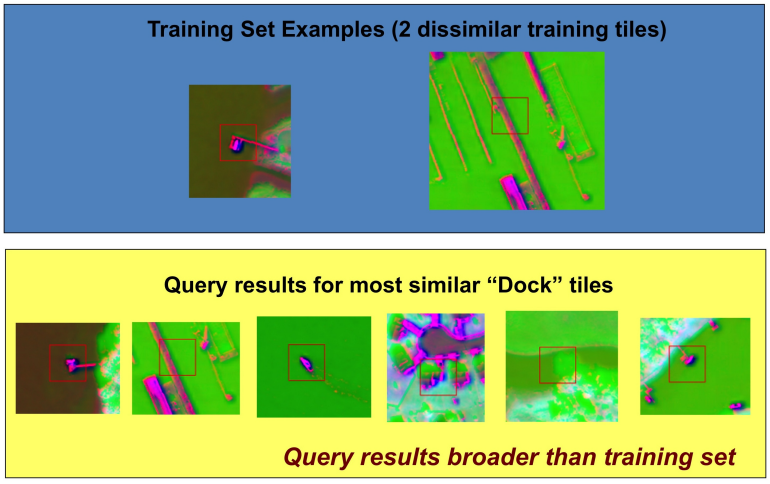


Figure 6.14: Example with docks: labeled data and the corresponding results of a query for "Dock" tiles (best viewed in color).

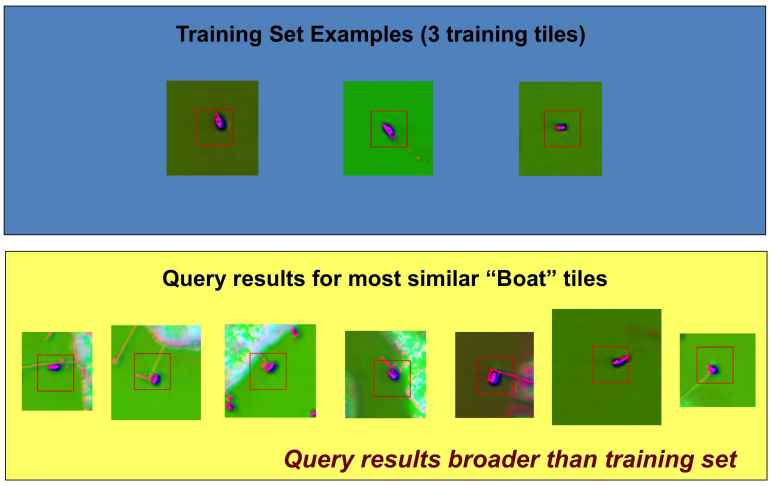


Figure 6.15: Example with boats: labeled data and the corresponding results of a query for "Boat" tiles (best viewed in color).

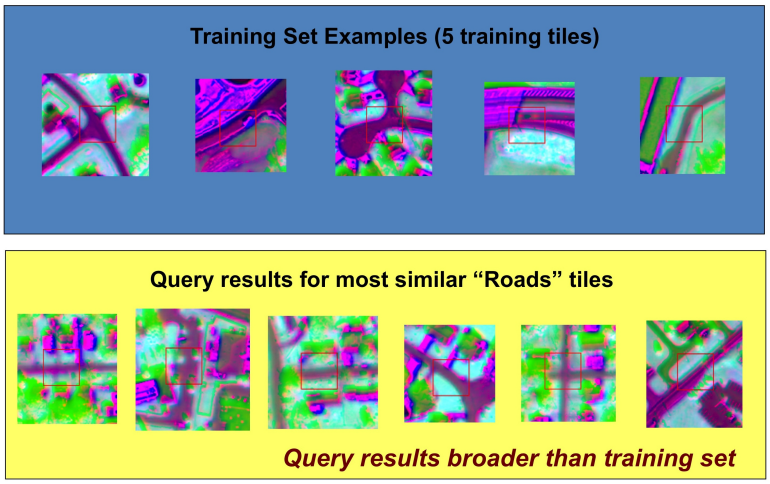


Figure 6.16: Example with roads: labeled data and the corresponding results of a query for "Roads" tiles (best viewed in color).

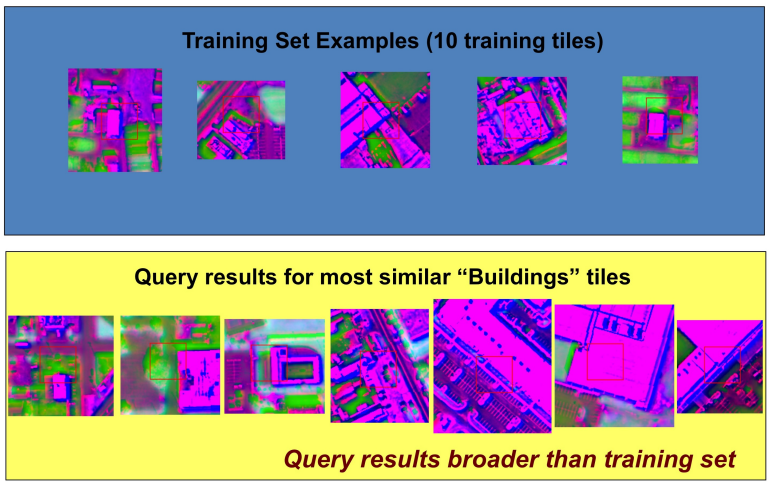


Figure 6.17: Example with buildings: labeled data and the corresponding results of a query for "Buildings" tiles (best viewed in color).

Chapter

7

Conclusion

This Ph.D. work was motivated by the increasing amount and complexity of the dada collected by digital systems in several areas, which turns the task of knowledge discovery out to an essential step in businesses' strategic decisions. The mining techniques used in the process usually have high computational costs and force the analyst to make complex choices. The complexity stems from the diversity of tasks that may be used in the analysis and from the large amount of alternatives to execute each task. The most common data mining tasks include data classification, labeling and clustering, outlier detection and missing data prediction. The large computational cost comes from the need to explore several alternative solutions, in different combinations, to obtain the desired information.

Although the same tasks applied to traditional data are also necessary for more complex data, such as images, graphs, audio and long texts, the complexity and the computational costs associated to handling large amounts of these complex data increase considerably, making the traditional techniques impractical. Therefore, especial data mining techniques for this kind of data need to be developed. This Ph.D. work focused on the development of new data mining techniques for large sets of complex data, especially for the clustering task tightly associated to other data mining tasks that are performed together. Specifically, this Doctoral dissertation presented three novel data mining algorithms well-suited to analyze large sets of complex data: the method *Halite* for correlation clustering; the method *BoW* for clustering Terabyte-scale datasets; and the method *QMAS* for labeling and summarization.

7.1 Main Contributions of this Ph.D. Work

Three data mining techniques were developed during this Ph.D. work. These techniques were evaluated on real, very large datasets with up to *billions* of complex elements, and they always presented highly accurate results, being at least one order of magnitude faster than the fastest related works in almost all cases. The real life data used come from the following applications: automatic breast cancer diagnosis, satellite imagery analysis, and graph mining on a large web graph crawled by Yahoo![1] and also on the graph with all users and their connections from the Twitter[2] social network. The three techniques developed are briefly described as follows.

7.1.1 The Method Halite for Correlation Clustering

The algorithm *Halite* is a fast and scalable density-based clustering algorithm for data of medium dimensionality able to analyze large collections of complex data elements. It creates a multi-dimensional grid all over the data space and counts the number of points lying at each hyper-cubic cell provided by the grid. A hyper-quad-tree-like structure, called the Counting-tree, is used to store the counts. The tree is thereafter submitted to a filtering process able to identify regions that are, in a statistical sense, denser than its neighboring regions regarding at least one dimension, which leads to the final clustering result. The algorithm is fast and has linear or quasi-linear time and space complexity regarding both the data size and the dimensionality.

7.1.2 The Method BoW for Clustering Terabyte-scale Datasets

The method *BoW* focuses on the problem of finding clusters in Terabytes of moderate-to-high dimensionality data, such as features extracted from billions of complex data elements. In these cases, a serial processing strategy is usually impractical. Just to read a single Terabyte of data (at 5GB/min on a single modern eSATA disk) one takes more than 3 hours. *BoW* explores parallelism through `MapReduce` and can treat as plug-in almost any of the serial clustering methods, including our own algorithm *Halite*. The major research challenges addressed are (a) how to minimize the I/O cost, taking care of the *already existing* data partition (e.g., on disks), and (b) how to minimize the network cost among processing nodes. Either of them may become the bottleneck. Our method automatically spots the bottleneck and chooses a good strategy, one of them uses our proposed *sampling-and-ignore* idea to reduce the network traffic.

[1] www.yahoo.com
[2] twitter.com

7.1.3 The Method QMAS for Labeling and Summarization

QMAS is a fast and scalable solution to the following problems:

1. **Low-labor labeling:** given a large collection of complex objects, very few of which are labeled with keywords, find the most suitable labels for the remaining ones;

2. **Mining and attention routing:** in the same setting, find clusters, the top-N_O outlier objects, and the top-N_R representative objects.

The algorithm is fast and it scales linearly with the data size, besides working even with tiny initial label sets.

7.2 Discussion

In the previous Chapter 3 we provide a brief description of representative methods found in the literature, which are aimed at spotting clusters in moderate-to-high dimensionality data. Then, we conclude Chapter 3 by summarizing in Table 3.1 some of the most relevant methods with regard to the main desirable properties that any clustering technique designed to analyze such kind of data should have. That table was partially obtained from [Kriegel et al., 2009]. In this section, we reprint in Table 7.1 the same table presented in Chapter 3, now *including* the methods developed in this Ph.D. work.

Remember that the initial analysis of the literature provided in Chapter 3 led us to come to one main conclusion. In spite of the several qualities found in the existing works, to the best of our knowledge, there is no method published in the literature, and designed to look for clusters in subspaces, that has *any* of the following desirable properties: (i) **to scale linearly or quasi-linearly** in terms of memory requirement and execution time with regard to increasing numbers of points and axes, besides increasing clusters' dimensionalities, and; (ii) to be able to handle data of **Terabyte-scale** in feasible time.

One of the main goals of the work performed in this Ph.D. work is to overcome these two limitations. Specifically, in Chapter 4, we focused on tackling the former problem identified – **linear or quasi-linear complexity**. We presented the method *Halite*, a novel *correlation clustering* algorithm for multi-dimensional data, whose main strengths are that it is fast and it has linear or quasi-linear scalability in time and space with regard to increasing numbers of objects and axes, besides increasing clusters dimensionalities. Therefore, the proposed method *Halite* tackles the problem of **linear or quasi-linear complexity**. A theoretical study on the time and space complexity of *Halite*, presented in Section 4.3, as well as an extensive experimental evaluation performed over synthetic and real data spanning up to 1 *million* elements and comparing *Halite* with seven representative works corroborate this claim.

In Chapter 5, we focused on the later problem – **Terabyte-scale data analysis**. In order to tackle this problem, we presented the method *BoW*, a novel, adaptive clustering method that explores parallelism using `MapReduce` for clustering huge datasets. It combines (a) potentially any serial algorithm used as a plug-in and (b) makes the plug-in run efficiently in parallel, by adaptively balancing the cost for disk accesses and network accesses, which allows *BoW* to achieve a very good tradeoff between these two possible bottlenecks. Therefore, *BoW* tackles the problem of **Terabyte-scale data analysis**. Experiments reported on both real and synthetic data with *billions* of points, and using up to $1,024$ cores in parallel corroborate this claim.

Clustering Algorithm	**Arbitrarily oriented clusters**	**Not relying on the *locality assumption***	**Adaptive density threshold**	**Independent wrt the order of the attributes**	**Independent wrt the order of the objects**	**Deterministic**	**Arbitrary number of clusters**	**Overlapping clusters (soft clustering)**	**Arbitrary subspace dimensionality**	**Avoiding complete enumeration**	**Robust to noise**	**Linear or quasi-linear complexity**	**Terabyte-scale data analysis**
Axes parallel clustering													
CLIQUE [Agrawal et al., 1998, 2005]		✓		✓	✓	✓	✓	✓	✓		✓		
ENCLUS [Cheng et al., 1999]		✓		✓	✓	✓	✓	✓	✓		✓		
SUBCLU [Kröger et al., 2004]		✓		✓	✓	✓	✓	✓	✓		✓		
PROCLUS [Aggarwal et al., 1999]			✓							✓			
PreDeCon [Bohm et al., 2004]				✓	✓	✓	✓			✓	✓		
P3C [Moise et al., 2006, 2008]		✓	✓	✓	✓	✓	✓	✓	✓		✓		
COSA [Friedman and Meulman, 2004]				✓	✓	✓			✓	✓	✓		
DOC/FASTDOC [Procopiuc et al., 2002]		✓		✓	✓		✓	✓	✓				
FIRES [Kriegel et al., 2005]		✓	✓		✓	✓	✓	✓	✓	✓	✓		
Correlation clustering													
ORCLUS [Aggarwal and Yu, 2002, 2000]	✓			✓						✓			
4C [Böhm et al., 2004]	✓			✓	✓	✓	✓			✓	✓		
COPAC [Achtert et al., 2007]	✓			✓	✓	✓	✓		✓	✓	✓		
CASH [Achtert et al., 2008]	✓	✓	*n a*		✓	✓	✓		✓		✓		
***Halite* [Cordeiro et al., 2011a]**	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
***BoW* [Cordeiro et al., 2011b]**	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓

Table 7.1: Properties of methods aimed at clustering moderate-to-high dimensionality data, *including* our methods *Halite* and *BoW* (using *Halite* as plug-in). The table was partially obtained from [Kriegel et al., 2009]. *n a*: not applicable.

Finally, notice in Table 7.1 that, the use of the *Halite* algorithm as a plug-in for the *BoW* method leads to the creation of a powerful tool for clustering moderate-to-high dimensionality data of large scale – this configuration has both the desirable properties sought, **linear or quasi-linear complexity** and **Terabyte-scale data analysis**, still having most of the other properties that any clustering technique designed to analyze moderate-to-high dimensionality data should have.

7.3 Difficulties Tackled

The main difficulty encountered during this Ph.D. work was the lack of large sets of real, complex objects, classified, preferably manually, by experts in the corresponding application area, in order to have a sound ground truth. Experiments with various real datasets are essential, given that they illustrate the practical usability of the mining methods developed.

Fortunately, this Ph.D. work included a visiting sandwich period of one year at the Carnegie Mellon University - USA, under the local supervision of the Ph.D. co-advisor, Prof. Dr. Christos Faloutsos. This activity was of great value for several reasons. Among them, it is important to mention that it provided us with access to large, real databases that were used to test the algorithms developed. In this way, the former difficulty was considerably reduced.

7.4 Future Work

The work performed in this Ph.D. work provides support to future works regarding numerous topics. Some of them are already being targeted by the Ph.D. candidate and his academic peers. The central topics for future works are briefly described as follows.

7.4.1 Using the Fractal Theory and Clustering Techniques to Improve the Climate Change Forecast

One interesting idea for future work is to use the mining techniques designed in this Ph.D. work, as well as Fractal Theory concepts [Schroeder, 1991], to improve the climate change forecast. Climate change forecast allows us to understand, to prevent and to mitigate bad consequences of the human activities to the future weather [Intergovernmental Panel on Climate Change – IPCC, 2007, The National Academies, 2008]. It uses numerical models, known as climate change models, that describe the main physical and dynamical processes of the climate system to simulate future climates as response to changes in external forces [Marengo, 2006].

The climate models are usually evaluated by starting the simulation in a given instant in the past, and using statistical comparison of trend analyses to compare the simulated results to the real recorded data. Today, the analyses of statistical significance indicate that the simulated results closely follow the recorded data, thus giving a strong evidence of the models' correctness. However, at GBdI we have been performing such kind of analysis using Fractal Theory techniques, and we have found that those techniques can clearly differentiate the simulated from the real data [Nunes et al., 2010, 2011, Traina et al., 2010]. Thus, although the current climate change models are appropriate from the statistical point of view, the Fractal Theory shows that they can be improved. Further studies lead us to believe that the clustering techniques for moderate-to-high dimensional data are promising tools to help improving such models. Therefore, one interesting question that we intend to answer in future works is: *How to use Fractal Theory concepts and clustering techniques to improve the climate change models?* In fact, we have already wrote a post-doctoral project on this topic and we will focus on it after this Ph.D. work.

7.4.2 Parallel Clustering

The method *BoW* automatically chooses to use the best of two strategies for parallel clustering, according to the environment in which it is used. Both strategies were carefully designed to allow *BoW* to work with potentially any serial clustering technique and data partitioning strategy. However, if the user is willing to abandon this desirable property, other algorithmic strategies become feasible, providing alternative properties. Specifically we have been working with two possibilities:

- **Map-only clustering:** when we restrict the data partitioning strategy to the File-Based option, described in Section 5.4.3 of Chapter 5, one map-only clustering algorithm becomes possible. The general idea is as follows. The `MapReduce` Framework automatically assigns to the mappers consecutive parts of the input data file. Thus, for the File-Based data partition, we can do clustering directly in the mappers and avoid the large cost of shuffling the data to reducers. This strategy can speed-up the process considerably in many cases. On the other hand, it forces us to limit the number of mappers to be used, also limiting the number of machines that will read the data in parallel from the disks and slowing down the overall process. This is necessary since the clustering quality tends to decrease when the data is partitioned into too many pieces for parallel processing. Therefore, this strategy may be good for some cases and bad for others and thus, one interesting future work is to derive the expected cost formulas and the estimation of the optimal number of machines for the Map-only clustering strategy and to include this option into *BoW*, in a way that, for the File-Based data partitioning, it starts choosing

automatically between three clustering strategies, instead of choosing the best of two strategies;

- **Grid-based clustering:** when we restrict *BoW* to work with grid-based, serial clustering algorithms only, such as our own *Halite* method, another algorithmic strategy becomes possible. The idea is to apply the hyper-grids to the data space directly in the mappers, in such a way that each mapper generates partial counts of points for the cells in the grids regarding its own subset of the dataset elements. Then, the counts of points can be shuffled to the reducers instead of the data elements themselves, minimizing considerably the total shuffling cost. This strategy allows a potentially large speed-up in the parallel clustering process, at the price of limiting the choice of the plug-in, serial algorithm to be used. Therefore, it is an interesting topic for future work.

7.4.3 Feature Selection by Clustering

A novel idea for dimensionality reduction is based on the fact that existing approaches usually rely on the identification of attribute correlations. The traditional strategy to reduce the data dimensionality is to identify and to eliminate most of the correlations, minimizing the effects of the dimensionality curse by finding a new set of orthogonal axes, of reduced cardinality, containing non-correlated dimensions relevant to characterize the data. Notice however that the correlations identified by most existing methods are global correlations, since they must occur for all dataset elements regarding a set of dimensions [Cordeiro et al., 2010b, Domeniconi et al., 2007, Kriegel et al., 2009]. Nevertheless, as described in Section 2.3 of Chapter 2, data in more than ten or so dimensions often present local correlations that refer only to subsets of the data elements and dimensions. In fact, distinct clusters may be correlated with different sets of dimensions. Therefore, it is clear that traditional dimensionality reduction techniques do not identify all possible correlations, as they only evaluate correlations valid for the entire dataset [Cordeiro et al., 2010b, Domeniconi et al., 2007, Kriegel et al., 2009]. A promising strategy to improve the results is to take into account the local correlations as well as the global ones.

The clustering algorithms for moderate-to-high dimensional data may represent a step forward into this idea, since they can be seen as a non-traditional approach to perform feature selection [Traina et al., 2011]. These methods identify local correlations by spotting clusters noticeable only when the data is projected into subspaces. Although, they have been extensively used for clustering multi-dimensional data in more than ten or so dimensions, as in the case of features extracted from complex data elements, feature selection is yet a very recent application for such methods. The general idea is that, instead of eliminating global correlations, one may use clustering to identify local correlations related to specific subsets of the data and assume the dimensions in which

these correlations occur as the most relevant ones, since they are the ones that allow differentiating the distinct categories inside the dataset. In other words, the dimensions participating in at least one of the local correlations spotted must not be discarded, since these dimensions have the highest discrimination power, behaving particularly for elements of one or more given clusters. The other features present uniform behavior to every element in the dataset, do not contributing to categorize the data, and thus they can be eliminated. Consequently, the use of clustering algorithms to select features from data in more than five or so dimensions is a promising, novel strategy to minimize the effects of the dimensionality curse for several data mining tasks.

7.4.4 Multi-labeling and Hierarchical Labeling

The *QMAS* algorithm proposes to find appropriate labels for the elements of a large collection of complex objects. The set of labels is previously defined by the user through examples and each unlabeled object of the collection is assigned to at most one label. Spotting a single label for each object is desirable for several applications, as we described in Chapter 6, but allowing each object to receive more than one appropriate label may be better in some situations. For example, when labeling the tiles generated by dividing satellite images into rectangular, equal-sized pieces, hybrid tiles may exist, as a bridge (both "Water" and "Concrete") or a park ("Water", "Forest" and "Concrete"), which ideally should receive more than one label.

Adding this functionality to the *QMAS* algorithm is an interesting idea for future work. The current algorithm allows sorting the labels according to their appropriateness with regard to each analyzed object. However, how many labels should be assigned for each given object? Some objects should receive a single label, while others should receive two, three or more labels. Thus, defining the correct number of labels to be assigned for each object is not a trivial task and this topic can be explored to extend the functionality of *QMAS*.

Finally, treating hierarchies of labels is another interesting extension for *QMAS*. Today, our algorithm considers the labels to be independent from each other, but the use of labeling hierarchies is desirable for some applications. Considering again the tiles extracted from satellite images, one may desire to define hierarchies of labels to be assigned to the tiles, containing, for example, several types and subtypes of forest, urban areas, and the like, in a way that each tile should be assigned to entire paths in the hierarchy, instead of being linked to individual labels only.

7.5 Publications Generated in this Ph.D. Work

The algorithm *Halite* is described in [Cordeiro et al., 2010b] and in [Cordeiro et al., 2011a]. The first publication refers to a full paper published at the IEEE International Conference on Data Engineering – ICDE (Qualis A2), one of the top quality conferences in the databases research area. The latter publication refers to an extended journal version of the conference paper, which is already *accepted* for publication as a regular paper at the IEEE Transactions on Knowledge and Data Engineering – TKDE journal (Qualis A1). The journal TKDE is one of the leading journals in the databases research area.

The method *BoW* is described in [Cordeiro et al., 2011b], a full paper published at the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, one of the leading conferences in the research areas of data mining and knowledge discovery from data. An extended journal version of the paper is being written.

The algorithm *QMAS* is described in [Cordeiro et al., 2010a], a paper published at the IEEE International Conference on Data Mining – ICDM (Qualis A2), one of the top quality conferences in the data mining and knowledge discovery field. An extended journal version of the paper is being written.

In summary, this Ph.D. work generated four main papers – one of them [Cordeiro et al., 2011a] is *accepted* for publication as a regular paper at the IEEE Transactions on Knowledge and Data Engineering – TKDE journal (Qualis A1). The journal TKDE is one of the leading journals in the databases research area; and the other three papers [Cordeiro et al., 2010b],[Cordeiro et al., 2011b] and [Cordeiro et al., 2010a] were accepted and published at top quality conferences, IEEE ICDE Conference (Qualis A2), ACM SIGKDD Conference and IEEE ICDM Conference (Qualis A2). Also, extended journal versions of the papers [Cordeiro et al., 2011b] and [Cordeiro et al., 2010a] are being written. Finally, additional works were conducted in parallel with the main topic of this Doctoral dissertation aimed at corroborating the applicability of the techniques developed during this Ph.D. work to real-life problems, such as mining data from Healthcare Information Systems and Agrometeorological data. These additional works generated one book chapter [Traina et al., 2011], one paper in an international conference [Traina et al., 2010] and the following other publications [Kaster et al., 2007], [Romani et al., 2010] and [Coltri et al., 2010].

Bibliography

Hadoop information. http://hadoop.apache.org/.

E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek. Robust, complete, and efficient correlation clustering. In *SDM, USA*, 2007.

E. Achtert, C. Böhm, J. David, P. Kröger, and A. Zimek. Global correlation clustering based on the hough transform. *Stat. Anal. Data Min.*, 1:111–127, November 2008. ISSN 1932-1864. doi: 10.1002/sam.v1:3.

P. K. Agarwal and N. H. Mustafa. k-means projective clustering. In *PODS*, pages 155–165, Paris, France, 2004. ACM. ISBN 158113858X. doi: http://doi.acm.org/10.1145/1055558.1055581.

C. Aggarwal and P. Yu. Redefining clustering for high-dimensional applications. *IEEE TKDE*, 14(2):210–225, 2002. ISSN 1041-4347. doi: http://doi.ieeecomputersociety.org/10.1109/69.991713.

C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. *SIGMOD Rec.*, 29(2):70–81, 2000. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/335191.335383.

C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2):61–72, 1999. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/304181.304188.

R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, 1998. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/276305.276314.

R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Min. Knowl. Discov.*, 11(1):5–33, 2005. ISSN 1384-5810. doi: http://dx.doi.org/10.1007/s10618-005-1396-1.

A. V. Aho, J. E. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1974. ISBN 0201000296.

M. Al-Razgan and C. Domeniconi. Weighted clustering ensembles. In J. Ghosh, D. Lambert, D. B. Skillicorn, and J. Srivastava, editors, *SDM*. SIAM, 2006. ISBN 0-89871-611-X.

S. Ando and H. Iba. Classification of gene expression profile using combinatory method of evolutionary computation and machine learning. *Genetic Programming and Evolvable Machines*, 5:145–156, 2004. ISSN 1389-2576. URL `http://dx.doi.org/10.1023/B:GENP.0000023685.83861.69`. 10.1023/B:GENP.0000023685.83861.69.

J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49(3):803–821, 1993.

K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *ICDT*, pages 217–235, UK, 1999. ISBN 3-540-65452-6.

A. P. Blicher. *Edge detection and geometric methods in computer vision (differential topology, perception, artificial intelligence, low-level)*. PhD thesis, University of California, Berkeley, 1984. AAI8512758.

C. Bohm, K. Kailing, H.-P. Kriegel, and P. Kroger. Density connected clustering with local subspace preferences. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 27–34, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2142-8.

C. Böhm, K. Kailing, P. Kröger, and A. Zimek. Computing clusters of correlation connected objects. In *SIGMOD*, pages 455–466, NY, USA, 2004. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007620.

C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant. Robust information-theoretic clustering. In *KDD*, pages 65–75, NY, USA, 2006. ISBN 1-59593-339-5. doi: http://doi.acm.org/10.1145/1150402.1150414.

C. Böhm, C. Faloutsos, and C. Plant. Outlier-robust clustering using independent components. In *SIGMOD*, pages 185–198, USA, 2008. ISBN 978-1-60558-102-6. doi: http://doi.acm.org/10.1145/1376616.1376638.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. ISBN 0-534-98053-8.

T. F. Chan and J. Shen. *Image processing and analysis - variational, PDE, wavelet, and stochastic methods*. SIAM, 2005. ISBN 978-0-89871-589-7.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *USENIX'06*, Berkeley, CA, USA, 2006.

C.-H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *KDD*, pages 84–93, NY, USA, 1999. ISBN 1-58113-143-7. doi: http://doi.acm.org/10.1145/312129.312199.

H. Cheng, K. A. Hua, and K. Vu. Constrained locally weighted clustering. *PVLDB*, 1(1): 90–101, 2008. doi: http://doi.acm.org/10.1145/1453856.1453871.

P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *The VLDB Journal*, pages 426–435, 1997.

P. P. Coltri, R. L. F. Cordeiro, T. T. de Souza, L. A. S. Romani, J. Zullo Jr., C. Traina Jr., and A. J. M. Traina. Classificação de áreas de café em minas gerais por meio do novo algoritmo qmas em imagem espectral geoeye-1. In *XV Simpósio Brasileiro de Sensoriamento Remoto - SBSR*, 2010. 8 pages (in Portuguese).

R. L. F. Cordeiro, F. Guo, D. S. Haverkamp, J. H. Horne, E. K. Hughes, G. Kim, A. J. M. Traina, C. Traina Jr., and C. Faloutsos. Qmas: Querying, mining and summarization of multi-modal databases. In G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, editors, *ICDM*, pages 785–790. IEEE Computer Society, 2010a.

R. L. F. Cordeiro, A. J. M. Traina, C. Faloutsos, and C. Traina Jr. Finding clusters in subspaces of very large, multi-dimensional datasets. In F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, and V. J. Tsotras, editors, *ICDE*, pages 625–636. IEEE, 2010b. ISBN 978-1-4244-5444-0.

R. L. F. Cordeiro, A. J. M. Traina, C. Faloutsos, and C. Traina Jr. Halite: Fast and scalable multi-resolution local-correlation clustering. *IEEE Trans. Knowl. Data Eng.*, 2011a. 15 pages (to appear).

R. L. F. Cordeiro, C. Traina Jr., A. J. M. Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In C. Apté, J. Ghosh, and P. Smyth, editors, *KDD*, pages 690–698. ACM, 2011b. ISBN 978-1-4503-0813-7.

M. Dash, H. Liu, and J. Yao. Dimensionality reduction for unsupervised data. In *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, pages 532–539, November 1997.

J. G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A*, 2:1160–1169, July 1985. doi: 10.1364/JOSAA.2.001160.

J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. doi: 10.2307/2984875.

C. Domeniconi, D. Papadopoulos, D. Gunopulos, and S. Ma. Subspace clustering of high dimensional data. In M. W. Berry, U. Dayal, C. Kamath, and D. B. Skillicorn, editors, *SDM*, 2004. ISBN 0-89871-568-7.

C. Domeniconi, D. Gunopulos, S. Ma, B. Yan, M. Al-Razgan, and D. Papadopoulos. Locally adaptive metrics for clustering high dimensional data. *Data Min. Knowl. Discov.*, 14(1):63–97, 2007. ISSN 1384-5810. doi: http://dx.doi.org/10.1007/s10618-006-0060-8.

R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2001. Second Edition.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000. ISBN 0471056693.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

U. Fayyad. A data miner's story – getting to know the grand challenges. In *Invited Innovation Talk, KDD*, 2007a. Slide 61. Available at: http://videolectures.net/kdd07_fayyad_dms/.

U. Fayyad. A data miner's story – getting to know the grand challenges. In *Invited Innovation Talk, KDD*, 2007b. Available at: http://videolectures.net/kdd07_fayyad_dms/.

U. M. Fayyad. Editorial. *ACM SIGKDD Explorations*, 5(2):1–3, 2003.

U. M. Fayyad and R. Uthurusamy. Data mining and knowledge discovery in databases (introduction to the special section). *Communications of the ACM*, 39(11):24–26, 1996.

U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. 1996.

J. H. Friedman and J. J. Meulman. Clustering objects on subsets of attributes (with discussion). *Journal Of The Royal Statistical Society Series B*, 66(4):815–849, 2004. URL `http://ideas.repec.org/a/bla/jorssb/v66y2004i4p815-849.html`.

L. Gibson and D. Lucas. Spatial Data Processing Using Generalized Balanced Ternary. In *IEEE Conference on Pattern Recognition and Image Analysis*, June 1982.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, USA, 3 edition, 1996. ISBN 0-8018-5413-X.

P. D. Grunwald, I. J. Myung, and M. A. Pitt. *Advances in Minimum Description Length: Theory and Applications (Neural Information Processing)*. The MIT Press, 2005. ISBN 0262072629.

J. Han and M. Kamber. *Data mining: Concepts and techniques*. Morgan Kaufmann, San Francisco, CA, 2 edition, 2006.

R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, 3(6):610 –621, nov. 1973. ISSN 0018-9472. doi: 10.1109/TSMC.1973.4309314.

P. Hough. Method and Means for Recognizing Complex Patterns. U.S. Patent 3.069.654, Dec. 1962.

J. Huang, S. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih. Image indexing using color correlograms. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 762 –768, jun 1997. doi: 10.1109/CVPR.1997.609412.

Intergovernmental Panel on Climate Change – IPCC. *Climate Change 2007: Summary for Policymakers*. Cambridge Univ. Press., 2007. Formally agreed statement of the IPCC concerning key findings and uncertainties contained in the Working Group contributions to the Fourth Assessment Report.

K. Kailing, H. Kriegel, and P. Kroger. Density-connected subspace clustering for highdimensional data, 2004.

U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *ICDM*, 2009.

U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. *SDM*, 2010.

K. V. R. Kanth, D. Agrawal, and A. K. Singh. Dimensionality reduction for similarity searching in dynamic databases. In L. M. Haas and A. Tiwary, editors, *ACM SIGMOD International Conference on Management of Data*, pages 166–176, Seattle, Washington, USA, 1998. ACM Press. Elaine Josiel.

D. S. Kaster, R. L. F. Cordeiro, and R. P. de Mattos Fortes. Um framework web colaborativo baseado em ontologias para recuperação e descoberta de conhecimento em imagens médicas. In *XIII Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia*, 2007. 8 pages. (in Portuguese).

F. Korn, B.-U. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(1):96–111, 2001. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.908983.

H.-P. Kriegel, P. Kröger, M. Renz, and S. Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *ICDM*, pages 250–257, Washington, USA, 2005. ISBN 0-7695-2278-5. doi: http://dx.doi.org/10.1109/ICDM.2005.5.

H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM TKDD*, 3(1):1–58, 2009. ISSN 1556-4681. doi: http://doi.acm.org/10.1145/1497577.1497578.

P. Kröger, H.-P. Kriegel, and K. Kailing. Density-connected subspace clustering for high-dimensional data. In *SDM, USA*, 2004.

R. Lämmel. Google's mapreduce programming model – revisited. *Science of Computer Programming*, 70:1–30, 2008.

S. Lazebnik and M. Raginsky. An empirical bayes approach to contextual region classification. In *CVPR*, pages 2380–2387. IEEE, 2009. ISBN 978-1-4244-3992-8.

S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129 – 137, mar 1982. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489.

F. Long, H. Zhang, and D. D. Feng. Fundamentals of content-based image retrieval. *Multimedia Information Retrieval and Management, Springer*, 2002.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

J. A. Marengo. *Mudanças climáticas globais e seus efeitos sobre a biodiversidade: caracterização do clima atual e definição das alterações climáticas para o território brasileiro ao longo do século XXI.* Ministério do Meio Ambiente, Secretaria de Biodiversidade e Florestas, Brasília, DF, 2006. (in Portuguese).

S. Mehrotra, Y. Rui, K. Chakrabarti, M. Ortega, and T. S. Huang. Multimedia analysis and retrieval system. In *Multimedia Information Systems, 1997. Proceedings., Third International Workshop on*, pages 25–27, September 1997.

G. Moise and J. Sander. Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In *KDD*, pages 533–541, 2008.

G. Moise, J. Sander, and M. Ester. P3C: A robust projected clustering algorithm. In *ICDM*, pages 414–425. IEEE Computer Society, 2006.

G. Moise, J. Sander, and M. Ester. Robust projected clustering. *Knowl. Inf. Syst.*, 14(3): 273–298, 2008. ISSN 0219-1377. doi: http://dx.doi.org/10.1007/s10115-007-0090-6.

G. Moise, A. Zimek, P. Kröger, H.-P. Kriegel, and J. Sander. Subspace and projected clustering: experimental evaluation and analysis. *Knowl. Inf. Syst.*, 21(3):299–326, 2009.

D. M. Mount and S. Arya. Ann: A library for approximate nearest neighbor searching. URL `http://www.cs.umd.edu/ mount/ANN/`.

E. K. K. Ng and A. W. Fu. Efficient algorithm for projected clustering. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 273, Washington, DC, USA, 2002. IEEE Computer Society.

E. K. K. Ng, A. W. chee Fu, and R. C.-W. Wong. Projective clustering by histograms. *TKDE*, 17(3):369–383, 2005. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2005.47.

S. Nunes, L. A. S. Romani, A. A. M. H. d. Ávila, C. Traina Jr., E. P. M. d. Sousa, and A. J. M. Traina. Análise baseada em fractais para identificação de mudanças de tendências em múltiplas séries climáticas. In *25 Simpósio Brasileiro de Bases de Dados (SBBD2010) - Short Papers*, volume 1, page 8 pag., Belo Horizonte, MG, 2010. SBC. (in Portuguese).

S. A. Nunes, L. A. S. Romani, A. M. H. Avila, C. Traina Jr., E. P. M. de Sousa, and A. J. M. Traina. Fractal-based analysis to identify trend changes in multiple climate time series. *Journal of Information and Data Management - JIDM*, 2:1–7, 2011. to appear.

C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08*, pages 1099–1110, 2008.

B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *IEEE International Conference on Data Engineering (ICDE)*, pages 589–598, San Diego, CA, 2000. IEEE Computer Society.

J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Gcap: Graph-based automatic image captioning. In *CVPRW '04: Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 9*, page 146, 2004.

S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with map-reduce. *ICDM*, 2008.

L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, 2004. ISSN 1931-0145. doi: http://doi.acm.org/10.1145/1007730.1007731.

G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *ACM Multimedia*, pages 65–73, 1996.

A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 34–47, 1994.

C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *SIGMOD*, pages 418–427, USA, 2002. ISBN 1-58113-497-5. doi: http://doi.acm.org/10.1145/564691.564739.

R. M. Rangayyan. *Biomedical Image Analysis*. CRC Press, Boca Raton, FL, 2005. ISBN 9780849396953.

S. O. Rezende. *Sistemas Inteligentes Fundamentos e Aplicações*. Editora Manole Ltda, 2002.

J. Rissanen. *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1989. ISBN 9971508591.

L. Romani, H. L. Razente, D. Y. T. Chino, E. P. M. Sousa, M. C. N. Barioni, M. X. Ribeiro, R. Gonçalves, A. M. Avila, J. Zullo, R. L. F. Cordeiro, S. A. Nunes, C. Traina Jr., J. F. Rodrigues Jr., W. D. Oliveira, and A. J. M. Traina. Agrodatamine: Integrating analysis of climate time series and remote sensing images. In *Microsoft Research eScience Workshop*, pages 134–136, 2010.

P. J. Rousseeuw and B. C. van Zomeren. Unmasking multivariate outliers and leverage points. *J. Amer. Stat. Assoc.*, 85(411):633–639, Sept. 1990. ISSN 0162-1459.

M. Schroeder. *Fractals, Chaos, Power Laws*. W. H. Freeman, New York, 6 edition, 1991.

J. Shotton, J. M. Winn, C. Rother, and A. Criminisi. *TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In A. Leonardis, H. Bischof, and A. Pinz, editors, *ECCV (1)*, volume 3951 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006. ISBN 3-540-33832-2.

G. W. Snedecor and W. G. Cochran. *Statistical Methods*. Iowa State University Press, Iowa, USA, 1989.

M. Sonka, V. Hlavac, and R. Boyle. *Image Processing: Analysis and Machine Vision*. Brooks/Cole Pub Co, 2 edition, 1998.

E. P. Sousa, J. Caetano Traina, A. J. Traina, L. Wu, and C. Faloutsos. A fast and effective method to find correlations among attributes in databases. *Data Min. Knowl. Discov.*, 14(3):367–407, 2007. ISSN 1384-5810. doi: http://dx.doi.org/10.1007/s10618-006-0056-4.

E. P. M. Sousa. Identificação de correlações usando a teoria dos fractais. Tese de Doutorado, ICMC/USP, São Carlos, 2006.

R. O. Stehling, M. A. Nascimento, and A. X. Falcão. Cell histograms versus color histograms for image representation and retrieval. *Knowl. Inf. Syst.*, 5: 315–336, September 2003. ISSN 0219-1377. doi: 10.1007/s10115-003-0084-y. URL `http://portal.acm.org/citation.cfm?id=959128.959131`.

H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1: 801–804, 1956. (in French).

The National Academies. *Understanding and Responding to Climate Change: Highlights of National Academies Reports*. The National Academies, 2008.

H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: fast solutions and applications. *Knowl. Inf. Syst.*, 14:327–346, March 2008. ISSN 0219-1377. doi: 10.1007/s10115-007-0094-2. URL `http://portal.acm.org/citation.cfm?id=1357641.1357646`.

A. B. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for non-parametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

A. J. M. Traina, C. Traina, J. M. Bueno, F. J. T. Chino, and P. Azevedo-Marques. Efficient content-based image retrieval through metric histograms. *World Wide Web*, 6:157–185, 2003. ISSN 1386-145X. URL `http://dx.doi.org/10.1023/A:1023670521530`. 10.1023/A:1023670521530.

A. J. M. Traina, L. A. Romani, R. L. F. Cordeiro, E. P. M. d. Sousa, M. X. Ribeiro, A. A. M. H. d. Ávila, J. Zullo Jr., J. F. Rodrigues Jr., and J. Traina, Caetano. How to find relevant patterns in climate data: an efficient and effective framework to mine climate time series and remote sensing images. In B. L. Keyfitz and L. N. Trefethen, editors, *SIAM Annual Meeting 2010*, page 6 pags., Pittsburgh, PA, 2010. SIAM.

A. J. M. Traina, C. Traina Jr., R. L. F. Cordeiro, M. X. Ribeiro, and P. M. Azevedo-Marques. Issues and techniques to mitigate the performance gap in content-based image retrieval systems. *Journal of Healthcare Information Systems and Informatics - IJHISI, special issue on New Technologies for Advancing Healthcare and Clinical Practices*, pages 60–83, 2011.

C. Traina Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *International Conference on Extending Database Technology (EDBT)*, volume 1777 of *Lecture Notes in Computer Science*, pages 51–65, Konstanz, Germany, 2000. Springer Verlag.

C. Traina Jr., A. J. M. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Trans. Knowl. Data Eng.*, 14(2):244–260, 2002.

C. Traina Jr., A. J. M. Traina, R. F. Santos Filho, and C. Faloutsos. How to improve the pruning ability of dynamic metric access methods. In *International Conference on Information and Knowledge Management (CIKM)*, pages 219–226, McLean, VA, USA, 2002. ACM Press.

A. K. H. Tung, X. Xu, and B. C. Ooi. Curler: finding and visualizing nonlinear correlation clusters. In *SIGMOD*, pages 467–478, 2005. ISBN 1-59593-060-4. doi: http://doi.acm.org/10.1145/1066157.1066211.

M. R. Vieira, C. Traina Jr., A. J. M. Traina, and F. J. T. Chino. Dbm-tree: A dynamic metric access method sensitive to local density data. In S. Lifschitz, editor, *Brazilian Symposium on Databases (SBBD)*, volume 1, pages 33–47, Brasília, DF, 2004. SBC.

W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, pages 186–195, 1997. ISBN 1-55860-470-7.

Wiki. http://wiki.apache.org/hadoop/hbase. URL `http://wiki.apache.org/hadoop/Hbase`. Hadoop's Bigtable-like structure.

K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee. Findit: a fast and intelligent subspace clustering algorithm using dimension voting. *Information & Software Technology*, 46 (4):255–271, 2004.

K. Yip, D. Cheung, and M. Ng. Harp: a practical projected clustering algorithm. *TKDE*, 16(11):1387–1397, 2004. ISSN 1041-4347. doi: 10.1109/TKDE.2004.74.

K. Y. Yip and M. K. Ng. Harp: A practical projected clustering algorithm. *IEEE Trans. on Knowl. and Data Eng.*, 16(11):1387–1397, 2004. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2004.74. Member-David W. Cheung.

K. Y. Yip, D. W. Cheung, and M. K. Ng. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering. In *ICDE*, pages 329–340, Washington, USA, 2005. ISBN 0-7695-2285-8. doi: http://dx.doi.org/10.1109/ICDE.2005.96.

M. L. Yiu and N. Mamoulis. Iterative projected clustering by subspace mining. *TKDE*, 17(2):176–189, 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.29.

B. Zhang, M. Hsu, and U. Dayal. K-harmonic means - a spatial clustering algorithm with boosting. In J. F. Roddick and K. Hornsby, editors, *TSDM*, volume 2007 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2000. ISBN 3-540-41773-7.

H. Zhang. The Optimality of Naive Bayes. In V. Barr and Z. Markov, editors, *FLAIRS Conference*. AAAI Press, 2004. URL `http://www.cs.unb.ca/profs/hzhang/publications/FLAIRS04ZhangH.pdf`.

X. Zhang, F. Pan, and W. Wang. Care: Finding local linear correlations in high dimensional data. In *ICDE*, pages 130–139, 2008.

C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson. Evolving accurate and compact classification rules with gene expression programming. *IEEE Transactions on Evolutionary Computation*, 7(6):519–531, 2003.