

UNIVERSITÀ DELLA CALABRIA



Dipartimento di ELETTRONICA,  
INFORMATICA E SISTEMISTICA

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,  
Informatica e Sistemistica

Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo

*Tesi di Dottorato*

# Discovering Exceptional Individuals and Properties in Data

Fabio Fassetti





UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,  
Informatica e Sistemistica

Dottorato di Ricerca in  
Ingegneria dei Sistemi e Informatica  
XX ciclo

*Tesi di Dottorato*

Discovering Exceptional Individuals  
and Properties in Data

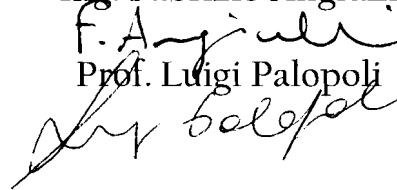
Fabio Fassetti



Coordinatore  
Prof. Domenico Talia



Supervisor  
Ing. Fabrizio Angiulli  
Prof. Luigi Palopoli



DEIS

DEIS – DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA  
Novembre 2007

Settore Scientifico Disciplinare: ING-INF/05

---

## Acknowledgments

*I wish to gratefully acknowledge my Ph.D. supervisor Luigi Palopoli for his teachings and for having provided me with ideas and suggestions; my supervisor Fabrizio Angiulli for having been a guidance for me, for his encouragements and for his support as a friend. I want to thank my Ph.D. coordinator, Domenico Talia, and all the colleagues I worked with during these years, for many valuable comments and stimulating discussions. My deepest gratitude goes to my family and to all my friends for their help, their unlimited support in any difficult time and for making pleasant my days.*



---

# Contents

<b>1</b>	<b>Introduction</b> .....	7
1.1	Overview on Exceptional Object Discovery .....	9
1.1.1	Outlier Detection .....	9
1.1.2	Anomalies in Streams of Data .....	11
1.2	Overview on Exceptional Property Discovery .....	12

---

## Part I Outlier Detection

---

<b>2</b>	<b>The Outlier Detection Problem</b> .....	17
2.1	Introduction .....	17
2.2	Related Work .....	20
<b>3</b>	<b>Outlier Detection in Data</b> .....	25
3.1	Contribution .....	25
3.2	Algorithm .....	26
3.3	Analysis of the algorithm .....	29
3.3.1	Spatial cost analysis .....	29
3.3.2	Implementation and time cost analysis .....	32
3.4	Experimental Results .....	33
<b>4</b>	<b>Outlier Detection in Streams of Data</b> .....	41
4.1	Contribution .....	41
4.2	Statement of the Problem .....	41
4.3	Algorithm .....	43
4.3.1	Exact algorithm .....	44
4.3.2	Approximate algorithm .....	46
4.4	Analysis of the Algorithm .....	48
4.4.1	Approximation Error Bounds .....	48
4.4.2	Implementation and Cost Analysis .....	53
4.5	Experimental results .....	55

---

**Part II Outlying Property Detection**

---

<b>5</b>	<b>Detecting Outlying Properties of Exceptional Objects</b>	61
5.1	Introduction	61
5.2	Related Work	62
5.3	Contribution	67
5.4	Outlying Properties	69
5.4.1	Preliminaries	69
5.4.2	Outlierness	71
5.4.3	Outlier Explanations	75
5.5	The Outlying Property Detection Problem	76
5.6	Complexity of The Outlying Property Detection Problem	76
5.6.1	Preliminaries on Computational Complexity	77
5.6.2	Complexity analysis	78
5.7	Upper Bound Properties	87
5.8	Algorithms	96
5.8.1	Global outlying properties	96
5.8.2	Local Outlying Properties	98
5.9	Algorithm implementation details, time and spatial cost	100
5.9.1	Data structures	100
5.9.2	Temporal cost	105
5.9.3	Spatial cost	106
5.10	Experimental results	107
5.10.1	Scalability	107
5.10.2	Sensitivity to parameters	109
5.10.3	About mined knowledge	110
5.10.4	Random data	114
	<b>Conclusion</b>	115
	<b>References</b>	117



---

## List of Figures

3.1	The DOLPHIN distance-based outlier detection algorithm. . . . .	26
3.2	Analysis of the algorithm. . . . .	29
3.3	Course of the size of INDEX. . . . .	35
3.4	Execution time and effectiveness of pruning rules. . . . .	36
3.5	Sensitivity to parameters $R$ and $k$ . . . . .	37
3.6	Comparison with other methods. . . . .	38
4.1	Exact data stream distance-based outlier detection algorithm. . . . .	44
4.2	Approximate data stream distance-based outlier detection algorithm. . . . .	45
4.3	Precision and Recall of approx-STORM. . . . .	54
4.4	Number of nearest neighbors associated with the misclassified objects of the <i>Rain</i> data set. . . . .	56
5.1	<i>Zoo database</i> (A=hair, B=feathers, C=eggs, D=milk, E=airborne, F=aquatic, G=predator, H=toothed, I=backbone, J=breathes, K=venomous, L=fins, M=legs (set of values: 0,2,4,5,6,8), N=tail, O=domestic, P=catsize, Q=type (integer values in range [1,7])). . . . .	63
5.2	Example Database . . . . .	70
5.3	Histograms of the example data base. . . . .	71
5.4	The areas associated with the curve of the cumulated frequency histogram. . . . .	72
5.5	Example of outlierness computation. . . . .	73
5.6	Example of Dataset with Functional Dependencies. . . . .	75
5.7	An example of the reduction used in Theorem 3 . . . . .	79
5.8	Example of reduction used in Theorem 4 . . . . .	81
5.9	Example of upper bounds obtained on the database $DB^{ex}$ . . . . .	94
5.10	The algorithm <i>FindOutliers</i> . . . . .	97
5.11	The algorithm <i>FindLocalOutliers</i> . . . . .	98
5.12	Example of data structure <i>db</i> . . . . .	101

5.13	Index structure .....	102
5.14	Example of frequency histogram management. ....	104
5.15	Explanation tree example. ....	106
5.16	Experimental results on the synthetical data set family. ....	108
5.17	Mean number of node visited (starting from the top of each figure the curves are for $\sigma = 0.25, 0.5, 0.75,$ and $1.0$ ). Notice that, on the Voting records database, curves for $\sigma = 0.75$ and $\sigma = 1.0$ overlap. ....	109
5.18	Mean execution times per node (milliseconds). ....	111

---

## List of Tables

3.1	Experiments on a massive dataset.....	39
4.1	Elaboration time per single object [msec]. ....	57
5.1	Computational complexity results concerning the outlying property detection problem. ....	87
5.2	Breast cancer data: attribute domains. ....	113
5.3	Random dataset family: experimental results. ....	114



---

## List Of Publications

Fabio Fassetti, Gianluigi Greco and Giorgio Terracina. Efficient discovery of loosely structured motifs in biological data. In *Proceedings of the 2006 ACM Symposium on Applied Computing(SAC)*, pages 151–155, 2006.

Fabrizio Angiulli, Fabio Fassetti and Luigi Palopoli. Un metodo per la scoperta di proprietà inattese. In *Proceedings of the Fourteenth Italian Symposium on Advanced Database Systems (SEBD)*, pages 321–328, 2006.

Fabio Fassetti, Gianluigi Greco and Giorgio Terracina. L-SME: A Tool for the Efficient Discovery of Loosely Structured Motifs in Biological Data. In *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems (SEBD)*, pages 389–396, 2007.

Fabio Fassetti and Bettina Fazzinga. FOX: Inference of Approximate Functional Dependencies from XML Data. In *Proceedings of Eighteenth International Workshop on Database and Expert Systems Applications (DEXA)*, pages 10–14, 2007.

Fabio Fassetti and Bettina Fazzinga. Approximate Functional Dependencies for XML Data. In *Local Proceedings of Eleventh East-European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 86–95, 2007.

Fabrizio Angiulli and Fabio Fassetti. Very Efficient Mining of Distance-Based Outlier. In *Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management*, pages 791–800, 2007.

Fabrizio Angiulli and Fabio Fassetti. Detecting Distance-Based Outliers in Streams of Data. In *Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management*, pages 811–820, 2007.

Fabrizio Angiulli, Fabio Fassetti and Luigi Palopoli. Detecting Outlying Properties of Exceptional Objects. *Submitted to an international journal*, 2007.

Fabio Fassetti, Gianluigi Greco and Giorgio Terracina. Mining Loosely Structured Motifs from Biological Data. *Submitted to an international journal*, 2007.



## Introduction

This thesis aims at providing novel techniques and methods in the complex scenario of knowledge discovery. This research field can be defined as “the nontrivial extraction of implicit, previously unknown, and potentially useful information from data”, and it has witnessed increasing interest in the last few years and a lot of research efforts have been spent on related problems. Knowledge discovery tasks mainly belong to four general categories:

1. dependency inference
2. class identification
3. class description
4. outlier detection

The former three categories focus on the search of characteristics applicable to a large percentage of objects. Many data mining tasks belong to these categories as *association rules*, *classification* and *data clustering*. Conversely, the fourth category deals with a very small percentage of dataset objects, which are often regarded as noise. Nevertheless, “one person’s noise may be another person’s signal”. Indeed, in a lot of real world domains, an object that, for some reasons, is different from the majority of the population which it belongs to, is the real interesting knowledge to be inferred from the population. Think, for example, an anomalous traffic pattern in a computer network that could signal the presence of a hacked computer. Similarly, anomalies in credit card transaction data could represent an illegal action, or in monitoring athlete performances, the detection of data which do not conform to expected normal behavior, is the main issue. Many clustering, classification, and dependency detection methods produce outliers as a by-product of their main task. For example, in classification, mislabeled objects are considered outliers and thus they are removed from the training set to improve the accuracy of the resulting classifier, while, in clustering, objects that do not strongly belong to any cluster are considered outliers. Nevertheless, searching for outliers through techniques specifically designed for tasks different from outlier detection could not be advantageous. As an example, clusters can be distorted by outliers

and, thus, the quality of the outliers returned is affected by their presence. Moreover, other than returning a solution of higher quality, outlier detection algorithms can be vastly more efficient than non ad-hoc algorithms.

The design of novel and efficient outlier detection techniques is the field in which the first part of this thesis aims to provide novel contributions. Here, the discovery of exceptionalities is tackled by considering a classical setting.

On the contrary, the second part of the thesis is concerned with a novel data mining task related to abnormality search. In classical meaning, outlier detection consists of mining objects which are anomalous, for some reasons, with respect to the population which they belong to. Conversely, here the anomalous object is known in advance and the goal is to find the features it possesses that can justify its exceptionality. In more details, it is assumed that you are given a significantly large data population characterized by a certain number of attributes, and you are provided with the information that one of the individuals in that data population is abnormal, but no reason whatsoever is given to you as to why this particular individual is to be considered abnormal. The interest here is precisely to single out such reasons.

This thesis is organized as follows. In the following of this chapter some data mining concepts and issues are briefly surveyed. Particular attention is dedicated to outlier detection tasks, introducing the basic terminology and some problems that arise in this scenario. Finally, the main contribution of the thesis are presented.

The remainder of the thesis is divided in two parts. The first one regards the detection of anomalous objects in a given population. In particular, Chapter 2 introduces the outlier detection problem and surveys related work; Chapter 3 presents a novel efficient technique for mining outlier objects coming from huge disk-resident data sets. The statistical foundation of the efficiency of the proposed method is investigated. An algorithm, called DOLPHIN, is presented and its performance and its complexity are described. Finally, the results of a large experimental campaign are reported in order to show the behavior of DOLPHIN on both synthetic and real data cases. Furthermore, comparisons with existing methods have been conducted, showing DOLPHIN to outperform the current state-of-the-art algorithms. Chapter 4 presents a method to detect outliers in streams of data, namely, when data objects are continuously delivered. Two algorithms are presented. The first one exactly answers outlier queries, but has larger space requirements. The second algorithm is directly derived from the exact one, has limited memory requirements and returns an approximate answer based on accurate estimations with a statistical guarantee. Several experiments have been accomplished, confirming the effectiveness of the proposed approach and the high quality of approximate solutions.

The second part of the thesis is concerned with the problem of discovering sets of attributes that account for the (a-priori stated) abnormality of an individual within a given data population. A criterion is presented to measure the abnormality of combinations of attribute values featured by the given ab-



normal individual with respect to the reference population. The problem of individuating abnormal properties is formally stated and analyzed, and the involved computational complexity is discussed. Such a formal characterization is then exploited in order to devise efficient algorithms for detecting outlying properties. Experimental evidence, which is also accounted for, shows that the algorithms are able to mine meaningful information and to accomplish the computational task by examining a negligible fraction of the search space.

## 1.1 Overview on Exceptional Object Discovery

In exceptional object discovery, the purpose is to find the objects that are different from most of the other objects. These objects are referred to as outliers. In many applications, singling out exceptional objects is much more interesting than detecting common characteristics. For example, consider fraud detection, commerce monitoring, athlete performances, and so on. Traditionally, the goal of the outlier detection task is to find outliers in a given population; nevertheless, there are many other emerging applications, such as network flow monitoring, telecommunications, data management, etc., in which the data set is not given, but data arrive continuously and it is either unnecessary or impractical to store all incoming objects. In this context, a hard challenge becomes that of finding the most exceptional objects in the data stream. In the following of this chapter, some preliminary concepts about both outlier detection and data streams are presented.

### 1.1.1 Outlier Detection

*Outlier Detection* aims at singling out exceptional objects. A natural question arises, that is: “What is an exceptional object (an outlier)?”. Although there is not a formal and general definition of an outlier, the Hawkins’ definition well capture the essence of an outlier: “an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism” [Hawkins, 1980]. Former works in outlier detection rely on statistics, and more than one hundred discordancy tests have been developed to identify outliers. Nevertheless, this kind of approach is not generally applicable either because there may not be a suitable test for a distribution or because no known distribution suitably models the data. Moreover, since in the data mining context data distribution is almost always unknown, a suitable standard distribution modeling data has to be inferred from data and the cost of this operation is often high, especially for large dataset. Thus, in recent years, numerous efforts have been made in this outstanding context to overcome these shortcomings.

Proposed approaches to outlier detection can be mainly classified in supervised, semi-supervised, and unsupervised. In supervised methods, an already

classified data set is available and it is employed as training set. The objects belonging to it are already known to be normal or abnormal, and they are exploited to learn a model to correctly classify any other object. In semi-supervised methods, instead, only a set of normal objects is given and available as the training set. The aim of this methods is to find a rule to partition the object space into regions, an *accepting* region and a *rejecting* one. The former contains the normal objects, whereas the latter contains the objects that significantly deviate from the training set. Finally, in unsupervised methods, no training set is given and the goal of finding outliers in a given data set is pursued by computing a score for each object suited to reflect its degree of abnormality. These scores are usually based on the comparison between an object and its neighborhood.

Data mining researchers have largely focused on unsupervised approaches. These approaches can be mainly further classified in *deviation-based* [Arning et al., 1996], *density-based* [Breunig et al., 2000], *MDEF-based* [Papadimitriou et al., 2003], and *distance-based* [Knorr and Ng, 1998].

Deviation-based techniques [Arning et al., 1996] identify as exceptional the subset  $I_x$  of the overall data set  $I$  whose removal maximizes the similarity among the objects in  $I \setminus I_x$ .

Density-based methods, introduced in [Breunig et al., 2000], are based on the notion of *local outlier*. Informally, the Local Outlier Factor (LOF) measures the outlierness degree of an object by comparing the density in its neighborhood with the average density in the neighborhood of its neighbors. The density of an object is related to the distance to its  $k^{th}$  nearest neighbor. Density-based methods are useful when the data set is composed of subpopulations with markedly different characteristics.

The multi-granularity deviation factor (MDEF), introduced in [Papadimitriou et al., 2003], is similar to the LOF score, but the neighborhood of an object consists of the objects within an user-provided radius and the density of an object is defined on the basis of the number of objects lying in its neighborhood.

Distance-based outlier detection has been introduced by Knorr and Ng [1998] to overcome the limitations of statistical methods, this novel notion of outliers, is a very general one and is not based on statistical considerations. They define outliers as follows:

**Definition 1.** *Let  $\mathcal{DB}$  be a set of objects,  $k$  a positive integer, and  $R$  a positive real number. An object  $o$  of  $\mathcal{DB}$  is a distance-based outlier (or, simply, an outlier) if less than  $k$  objects in  $\mathcal{DB}$  lie within distance  $R$  from  $o$ .*

The term *DB( $k, R$ )-outlier* is a shorthand for Distance-Based outlier detected using parameters  $k$  and  $R$ . Objects lying at distance not greater than  $R$  from  $o$  are called *neighbors* of  $o$ . The object  $o$  is considered a neighbor of itself.

This definition is very intuitive since it defines an outlier as an object lying distant from others dataset objects.

Moreover, Definition 1 is a solid one, since it generalizes many notions of outliers defined in statistics for standard distributions and supported by statistical outlier tests. In other words, for almost all discordancy tests, if an object  $o$  is an outlier according to a specific test, then  $o$  is also a  $DB(k, R)$ -outlier for suitable values of parameters  $k$  and  $R$ . Nevertheless, this definition can not replace all statistical outlier tests. For example, if no distance can be defined between dataset objects, DB-outlier notion cannot be employed.

Some variants of the original definition have been subsequently introduced in the literature. In the previous one, the number of DB-outliers is not fixed. In some applications can be useful to find the *top* –  $n$  outliers, namely the  $n$  outliers scoring the highest score of abnormality. Then, a score function able to rank outliers is needed. In particular, in order to rank the outliers, Ramaswamy et al. [2000] introduced the following definition:

**Definition 2.** *Given two integers  $k$  and  $n$ , an object  $o$  is an outlier if less than  $n$  objects have higher value for  $D^k$  than  $o$ , where  $D^k$  denotes the distance of the  $k^{\text{th}}$  nearest neighbor of the object.*

Subsequently, Angiulli and Pizzuti [2002], with the aim of taking into account the whole neighborhood of the objects, proposed to rank them on the basis of the sum of the distances from the  $k$  nearest neighbors, rather than considering solely the distance to the  $k^{\text{th}}$  nearest neighbor. Therefore, they introduced the following definition:

**Definition 3.** *Given two integers  $k$  and  $n$ , an object is an outlier if less than  $n$  objects have higher value for  $w_k$ , where  $w_k$  denotes the sum of the distances between the object and its  $k$  nearest neighbors*

Last definition was also used by Eskin et al. [2002].

The three definitions above are closely related to one another. In particular, there exist values for the parameters such that the outliers found by using the first definition are the same as those obtained by using the second definition, but not the third one. In this work we will deal with the original definition provided by Knorr and Ng [1998], even if we will compare also with approaches following the definition given by Ramaswamy et al. [2000].

### 1.1.2 Anomalies in Streams of Data

A *data stream* is a large volume of data coming as an unbounded sequence where, typically, older data objects are less significant than more recent ones. This is because the characteristics of the data may change over time, and then the most recent behavior should be given larger weight.

Therefore, data mining on evolving data streams is often performed based on certain time intervals, called windows. Two main different data streams window models have been introduced in literature: *landmark window* and *sliding window* [Golab and Özsu, 2003].

In the first model, some time points (called landmarks) are identified in the data stream, and analysis are performed only for the stream portion which falls between the last landmark and the current time. Then, the window is identified by a fixed endpoint and a moving endpoint.

In contrast, in the sliding window model, the window is identified by two sliding endpoints. In this approach, old data points are thrown out as new points arrive. In particular, a *decay function* is defined, that determines the weight of each point as a function of elapsed time since the point was observed. A meaningful decay function is the step function. Let  $W$  be a parameter defining the window size and let  $t$  denote the current time, the step function evaluates to 1 in the temporal interval  $[t - W + 1, t]$ , and 0 elsewhere.

In all window models the main task is to analyze the portion of the stream within the current window, in order to mine data stream properties or to single out objects conforming with characteristics of interest.

Due to the intrinsic characteristics of a data stream, and in particular since it is neverending, finding outliers in a stream of data becomes a hard challenge.

## 1.2 Overview on Exceptional Property Discovery

The anomaly detection issues introduced in the previous two sections is a prominent research topic in data mining, as the numerous proposed approaches witness. A lot of efforts have been paid to discover unexpected elements in data populations and several techniques have been presented [Tan et al., 2005].

The classical problem in outlier detection is to identify anomalous objects in a given population. But, what does the term *anomalous* mean? When an object can be defined anomalous? As discussed in the previous section, there are many definition to establish the exceptionality of an object. An object is identified as outlier if it possesses some characteristics, established by the chosen definition, that the other dataset objects do not possess, or if it does not possess some characteristics that the other dataset objects possess. Hence, in words, the characteristics distinguishing inliers from outliers are a-priori chosen, and then, outliers are detected according to the choice. Nevertheless, in many real situations, a novel but related problem arises, which is somehow the inverse of the previous one: it is a-priori known which objects are inliers and which ones are outliers and the characteristics distinguishing them have to be detected. A limited attention has been paid till now to this problem. This problem has many practical applications. For instance, in analyzing health parameters of a sick patient, if a history of healthy patients is given and if the same set of parameters is available, then it is relevant to single out that subset of those parameters that mostly differentiate the sick patient from the healthy population. As another example, the history of the characteristics athlete that has established an exceptional performance can be analyzed to

detect those characteristics distinguishing the last exceptional performance from the previous ones.

Hence, in this context, a population is given. It can consist of entries referred to the same individual (performance history of an athlete) or to different individuals (sick and healthy patients), and for each entry several attributes are stored. The outlier is known and the goal is to single out properties it possesses, that justify its abnormality. Often, such properties are subsets of attribute values featured by the given outlier, that are anomalous with respect to the population the outlier belongs to. Naturally, the following question arises: what does the term *anomalous* mean? When a property can be defined anomalous? Also in this context, a lot of definitions can be introduced. For example, a property can be defined as anomalous, if it is different from the rest of the population, and this difference is statistically significant. Moreover, different definitions characterize properties referred to numerical attributes and properties referred to categorical attribute. For example, for numerical attributes a distance among properties can be defined, while for categorical attributes the frequency of a property can be meaningful.



**Outlier Detection**





## The Outlier Detection Problem

### 2.1 Introduction

In this part of the thesis, the outlier detection problem is addressed. In particular, two scenarios are considered. In the first one, outliers have to be discovered in disk-resident data sets. These data sets can be accessed many times, even if they are assumed to be very large and then the cost to be spent to read them from disk is relevant. In the second scenario considered in this thesis, data are assumed to come continuously from a given data source. Thus, they can be read only once, and this adds further difficulties and novel constraints to be taken in account.

There exist several approaches to the problem of singling out the objects mostly deviating from a given collection of data [Barnett and Lewis, 1994, Arning et al., 1996, Knorr and Ng, 1998, Breunig et al., 2000, Aggarwal and Yu, 2001, Jin et al., 2001, Papadimitriou et al., 2003]. In particular, distance-based approaches [Knorr and Ng, 1998] exploit the availability of a distance function relating each pair of objects of the collection. These approaches identify as outliers the objects lying in the most sparse regions of the feature space.

Distance-based definitions [Knorr and Ng, 1998, Ramaswamy et al., 2000, Angiulli and Pizzuti, 2002] represent an useful tool for data analysis [Knorr and Ng, 1999, Eskin et al., 2002, Lazarevic et al., 2003]. They have robust theoretical foundations, since they generalize diverse statistical tests. Furthermore, these definitions are computationally efficient, as distance-based outlier scores are monotonic non-increasing with the portion of the database already explored.

In recent years, several clever algorithms have been proposed to fast detect distance-based outliers [Knorr et al., 2000, Ramaswamy et al., 2000, Bay and Schwabacher, 2003, Angiulli and Pizzuti, 2005, Ghoting et al., 2006, Tao et al., 2006]. Some of them are very efficient in terms of CPU cost, while some others are mainly focused on minimizing the I/O cost. Nevertheless, it is worth to

notice that none of them is able to simultaneously achieve the two previously mentioned goals when the dataset does not fit in main memory.

In this thesis a novel technique for mining distance-based outliers in an high dimensional very large disk-resident dataset, with both near linear CPU and I/O cost, is presented.

The proposed algorithm, DOLPHIN, performs only two sequential scans of the data. It needs to maintain in main memory a subset of the dataset and employs it as a summary of the objects already seen. This subset allows to efficiently search for neighbors of data set objects and, thus, to fast determine whether an object is an inlier or not. An important feature of the method is that memory-resident data is indexed by using a suitable proximity search approach. Furthermore, DOLPHIN exploits some effective pruning rules to early detect inliers, without needing to find  $k$  neighbors for each of them. Importantly, both theoretical justifications and empirical evidences that the amount of main memory required by DOLPHIN is only a small fraction of the dataset are provided. Therefore, the approach is feasible on very large disk-resident datasets.

The I/O cost of DOLPHIN corresponds to the cost of sequentially reading two times the input dataset file. This cost is negligible even for very large datasets. As for the CPU cost, the algorithm performs in quadratic time but with a little multiplicative constant, due to the introduced optimizations. In practice, the algorithm needs to compute only a little fraction of the overall number of distances (which is quadratic with respect to the size of the data set) in order to accomplish its task. DOLPHIN has been compared with state of the art methods, showing that it outperforms existing ones of at least one order of magnitude.

As previously stated, in the data stream context, new challenges arise along with the outlier detection problems. First of all, data come continuously. Second, with each data stream object, a life time is associated. While in data sets each object is equally relevant, in data streams an objects *expires* after some periods of time, namely, it is no more relevant in the analysis. Third, the characteristics of the stream, and then the characteristics of the relevant population may change during time.

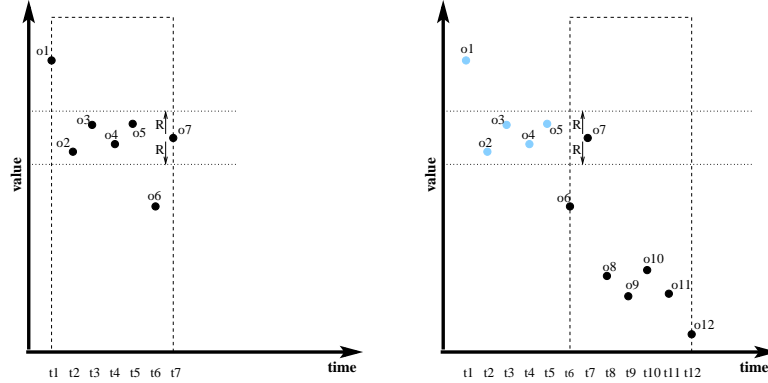
In this thesis, a contribution in a data stream setting is given. Specifically, the problem of outlier detection adopting the sliding window model with the step function as the decay function is addressed.

The proposed approach introduces a novel concept of *querying* for outliers. Specifically, previous work deals with *continuous queries*, that are queries evaluated continuously as data stream objects arrive; conversely, here, *one-time queries* are considered. This kind of queries are evaluated once over a point-in-time (for a survey on data streams query models refer to [Babcock et al., 2002]). The underlying intuition is that, due to stream evolution, object properties can change over time and, hence, evaluating an object for outlierness when it arrives, although meaningful, can be reductive in some contexts and

sometimes misleading. On the contrary, by classifying single objects when a data analysis is required, data *concept drift* typical of streams can be captured. To this aim, it is needed to support queries at arbitrary points-in-time, called *query times*, which classify the whole population in the current window instead of the single incoming data stream object.

The example below shows how concept drift can affect the outlieriness of data stream objects.

*Example 1.* Consider the following figure.



The two diagrams represent the evolution of a data stream of 1-dimensional objects. The abscissa reports the time of arrival of the objects, while the ordinate reports the value assumed by each object. Let the number  $k$  of nearest neighbors to consider be equal to 3, and let the window size  $W$  be equal to 7. The dashed line represents the current window.

The left diagram reports the current window at time  $t_7$  (comprehending the interval  $[t_1, t_7]$ ), whereas the right diagram reports the current window at time  $t_{12}$  (comprehending the interval  $[t_6, t_{12}]$ ).

First of all, consider the diagram on the left. Due to the data distribution in the current window at time  $t_7$ , the object  $o_7$  is an inlier, since it has four neighbors in the window. Then, if an analysis were required at time  $t_7$ , the object  $o_7$  would be recognized as an inlier. Note that  $o_7$  belongs to a very dense region.

Nevertheless, when stream evolves the data distribution change. The region, which  $o_7$  belongs to, becomes sparse, and data stream objects assume lower values. The right figure shows the evolution of the stream until time  $t_{12}$ . In the novel distribution,  $o_7$  has not any neighbor. Then, if an analysis were required at time instant  $t_{12}$ ,  $o_7$  should be recognized as an outlier. Note that now  $o_7$  belongs to a very sparse region.

## 2.2 Related Work

*Distance-based outliers* have been first introduced by Knorr and Ng [1998], which also presented the early three algorithms to detect distance-based outliers. The first one is a block nested loop algorithm that runs in  $O(dN^2)$  time, where  $N$  is the number of objects and  $d$  the dimensionality of the data set. It extends the naïve nested loop algorithm by adopting a strategy to reduce the number of data blocks to be read from the disk. This kind of approach has a quadratic cost with respect to  $N$ , that can be impractical for handling large data sets.

The second and the third one are two cell-based algorithms, which are linear with respect to  $N$ , but exponential in  $d$ . Then, they are fast only if  $d$  is small. In particular, the former is designed to work with memory-resident data sets, whereas the latter deals with disk-resident ones and, then, aims at minimizing the number of passes over the data sets (indeed it guarantees at most three passes over the data sets). Basically, the main idea of cell-based algorithms is to partition the space into cells of length  $\frac{R}{2\sqrt{d}}$ , and counting the number of objects within cells. The number of neighbors of an object can be determined by examining only the cells close to the cell which the object belongs to.

A shortcoming of this technique is that the number of cells is exponential with respect to the number of dimensions  $d$ .

Therefore, the latter kind of techniques is impractical if  $d$  is not small, and the former approach does not scale well w.r.t.  $N$ ; then, efforts for developing efficient algorithms scaling well to large datasets have been subsequently made.

Ramaswamy et al. [2000] present two novel algorithms to detect outliers. The first assumes the dataset to be stored in a spatial index, like the *R\*-tree* [Beckmann et al., 1990], and uses it to compute the distance of each dataset object from its  $k^{th}$  nearest neighbor. Pruning optimizations to reduce the number of distance computations while querying the index are exploited. The authors noted that this method is computationally expensive and introduced a partition-based algorithm to reduce the computational cost. The second algorithm first partitions the input points using a clustering algorithm, and then prunes the partitions that cannot contain outliers. Experiments were reported only up to ten dimensions.

Bay and Schwabacher [2003] introduce the distance-based outlier detection algorithm *ORCA*. Basically, *ORCA* enhances the naive block nested loop algorithm with a simple pruning rule and randomization, obtaining a near linear scaling on large and high dimensional data sets. The major merit of this work is to show that the CPU time of the their schema is often approximately linear in the dataset size.

In order to improve performances of *ORCA*, Ghoting et al. [2006] propose the algorithm *RBRP* (Recursive Binning and Re-Projection). The method has two phases. During the first phase, a divisive hierarchical clustering algorithm

is used to partition objects into bins, i.e. group of objects likely to be close to each other. Then, objects in the same bin are reorganized according to the projection along the principal component. During the second phase, the strategy of ORCA is employed on the clustered data, obtaining improved performances.

Recently, Tao et al. [2006] point out that for typical values of the parameters, ORCA has quadratic I/O cost. Then, they present an algorithm, named *SNIF* (for ScaN with prIoritized Flushing), intended to work with datasets that do not fit into main memory, and whose major goal is to achieve linear I/O cost. They propose two algorithms, the first one retrieves the outliers by scanning the dataset three times but has smaller space requirements, the second algorithm needs to perform in some cases only two data set scans, but has larger space requirements. Roughly speaking, the first algorithm initially accommodates in memory a sample set of objects of fixed size  $s$  and then counts and stores the number of neighbors within distance  $\frac{R}{2}$  of these objects by scanning the data set. Using the stored information, during a second scan, some objects are recognized as inliers, and all the other objects are stored in memory.

The authors show that, in practice, the total amount of memory required is smaller than the available memory. Then, they propose a second algorithm, which stores much more objects in memory, in order to reduce the probability of the third scan. In particular, dataset objects, and for each of them the number of neighbors found till now, are stored. Whenever the available memory gets full, the algorithm halves the occupied memory, by discarding recognized inliers and by storing objects with lower priority in a *verification file*. The priority is the probability that the object is an outlier. Next, a second scan is performed to classify the objects in memory and the objects of the verification file that are loaded in memory. If the memory becomes full before the verification file is exhausted, then the second scan is performed to empty the memory and a third scan is required for the remaining verification file objects. Authors show that, in practice, the verification file contains a very small number of objects and in some cases is empty, then SNIF needs only two scans to accomplish its task. Furthermore, authors show that the I/O cost of their algorithm is low and insensitive to the parameters, but time performances of the method were not deeply investigated.

The *HilOut* algorithm, [Angiulli and Pizzuti, 2005], detects the top distance-based outliers, according to the weight score, in a numerical data sets. It makes use of the Hilbert space-filling curve in order to linearize the data set and consists of two phases: the first phase guarantees at least an approximate solution, by scanning at most  $d+1$  times the data set and with temporal cost quadratic in  $d$  and linear in  $N$ , where  $d$  is the number of dimensions of the data set and  $N$  is the number of data set objects. If needed, the second phase is performed. It provides the exact solution after a second data set scan examining the candidates outlier returned by the first phase. Experimental results show that the

algorithm always stops, reporting the exact solution, during the first phase after much less than  $d + 1$  steps.

In domains where distance computations are very expensive, e.g. the edit distance between subsequences or the quadratic distance between image color histograms, determining the exact solution may become prohibitive. Wu and Jermaine [2006] considered this scenario and describe a sampling algorithm for detecting distance-based outliers with accuracy guarantees. Roughly speaking, the algorithm works as follows. For each data point,  $\alpha$  points are randomly sampled from the data set. Using the user-specified distance function, the  $k^{\text{th}}$ - $NN$  distance of the data set objects in those  $\alpha$  samples are computed. When the data set objects end, the sampling algorithm returns the objects whose sampled  $k^{\text{th}}$ - $NN$  distance is the greatest. The total number of computed distances is  $\alpha N$ , where  $N$  is the number of data set objects. Authors analyze the statistical properties of the proposed algorithm to provide accuracy guarantees thereof.

This thesis presents a novel distance-based outlier detection algorithm, named *DOLPHIN* (for Detecting OutLiers PusHing data into an INdex), whose goal is to achieve both near linear CPU and I/O cost on very large disk-resident datasets, with a small usage of main memory. It gains efficiency by integrating pruning rules and state of the art database index technologies.

It must be preliminary noted that none of the existing methods is able to achieve both these goals. Some algorithms exploit indexing or clustering [Ramaswamy et al., 2000, Ghoting et al., 2006], but require to build the index, or to perform clustering, on the whole dataset, and, in some cases, to store the clustered data in main memory. On the other hand, the technique of detecting outliers by directly exploiting existing indexing techniques [Bentley, 1975, Beckmann et al., 1990, Berchtold et al., 1996, Böhm et al., 2001, Chávez et al., 2001] in order to search for the  $k^{\text{th}}$  nearest neighbor of an object, suffers of the drawback that all the dataset objects have to be stored into the index structure. Besides, note that the approach of computing the  $k$  nearest neighbors of each object is not very efficient for outlier detection, since, for a lot of objects, this task can be avoided by using clever algorithms. Furthermore, the approach in [Bay and Schwabacher, 2003] works directly on disk-resident data and is efficient in CPU time, being able to achieve roughly near linear time for some combinations of the parameters but, as pointed out by Tao et al. [2006], its I/O cost may be quadratic. On the other hand, the approach in [Tao et al., 2006] keeps low the I/O cost, but it is not as efficient from the point of view of the CPU cost (for example, note that, SNIF compares each dataset object to all the  $s$  centroids, but  $s = 1,000$ , or greater, is a typical value for this parameter), and cannot be used in nonmetric spaces.

*DOLPHIN* detects outliers in disk-resident datasets. It performs two sequential scans of input dataset file. During the first scan, it maintains a data structure storing a small subset of the dataset objects in main memory, together with some additional information. This memory-resident data structure represents a summary of the already seen objects, and it is used to determine

whether the object currently read from disk is an inlier or not. If it cannot be determined that the current object is an inlier, then it is added to the memory-resident data. At the same time, objects already stored in main memory could be recognized as inliers and may be discarded. By retaining a moderate fraction of proved inliers, DOLPHIN is able to effectively exploit the triangular inequality to early prune inliers, without the need of computing  $k$  distances per object.

Outlier detection methods previously discussed are designed to work in a batch framework, namely under the assumption that the whole data set is stored in secondary memory and multiple passes over the data can be accomplished. Hence, they are not suitable for the online paradigm or for processing data streams. While the majority of the approaches to detect anomalies in data mining consider the batch framework, some researchers have attempted to address the problem of online outlier detection. In [Yamanishi et al., 2000], the *SmartSifter* system is presented, addressing the problem from the viewpoint of statistical learning theory. The system employs an online discounting learning algorithm to learn a probabilistic model representing the data source. An important feature of the employed algorithm is that it discounts the effect of past data in the on-line process by means of a decay function. It assigns a score to the datum, measuring how large the model has changed after learning. Every time a datum is input, SmartSifter updates the model and assigns a score to the input datum on the basis of the model. In particular, SmartSifter measures how large the model updated with the new datum has moved from the one learned before. The algorithm returns as outliers the data having high scores, that have, then, an high probability of being statistical outliers.

In [Ghoting et al., 2004], authors present *LOADED* (Link-based Outlier and Anomaly Detectin in Evolving Data Sets), an algorithm for outlier detection in evolving data sets. The authors focus on two main aspects: first, their algorithm accomplishes only one-pass over the data set and, then, it is employable for on-the-fly outlier detection; second, their algorithm deals with both categorical and continuous attributes. In particular, authors define a metric that is able to determine dependencies between both categorical and continuous attributes. As for categorical attributes, they define that there is a link between two objects if they have a pair attribute-value in common. The strength of the link between two objects is the number of links they possess. Conversely, for continuous objects, they employ correlation coefficients between each pair of continuous attributes. Based on these dependencies, they define that an object is linked to another one in the mixed attribute space if there is a link between them in the categorical attribute subspace, and if their continuous attributes adhere according to the correlation coefficients. Objects having few objects linked to them are considered outliers.

In [Aggarwal, 2005], the focus is on detecting rare events in a data stream. Their technique is able to detect exceptional events in a stream in which also some other anomalies are present. These other anomalies are called spurious

abnormalities and affect the stream in a similar way as rare events. Then, they deal with the further difficulty to capture the subtle differences between rare events of interest and other similar, but more frequent and less interesting, anomalies. Their method is a supervised one, and performs statistical analysis on the stream. In particular, the incoming objects are unlabeled but when an event is recognized as “rare” by external mechanism (for example the user can recognize the rare event for its actual consequences) this information is given to the system for improving the accuracy of the abnormality detection. This algorithm continuously detects events using the data from a history, and rare events are defined on the basis of their deviation from expected values computed on historical trends.

In [Subramaniam et al., 2006] a distributed framework to detect outliers in a sensor network is presented. The main focus of the work is to deal with sensors, that are characterized by limited resource capabilities. In the proposed settings, each sensor stores a model that approximates the distribution of the data it receives. In particular, since the sliding window model is adopted, the model refers only to data in the current window. To compute an approximation of data distributions, kernel estimators are employed. Based on these estimations, each sensor can detect the outliers among the data it receives. Next, in order to mine outliers of the overall network, the outliers coming by single sensors are combined. This work detects outliers according two outlier definitions, i.e. distance-based and MDEF-based [Papadimitriou et al., 2003]. According to the latter definition, an object  $o$  is an outlier if its number of neighbors is statistically significantly different from the average number of neighbors of the objects in a random sample of the neighborhood of  $o$ . It must be said that this method is specifically designed to support sensor networks.

Moreover, all the techniques discussed above, detect anomalies online as they arrive, and one-time queries are not supported.

In this thesis, a novel technique, called *STORM*, is proposed and two algorithms are presented, an exact and an approximate one. The algorithms are designed to mine distance-based outliers in data streams under the sliding window model, and *outlier queries* are performed in order to detect anomalies in the current window. The exact algorithm always returns all and only the outliers in the window actual at query time. The approximate algorithm returns an approximate set of outliers, has smaller space requirements than the exact one, and anyway guarantees an high accuracy of the provided answer.



## Outlier Detection in Data

### 3.1 Contribution

In this chapter the problem of outlier detection in data is addressed, and an efficient algorithm for solving it is presented and analyzed in details.

This chapter is organized as follows. In this section the contribution given by this thesis is stated. Subsequent section 4.3 describes the DOLPHIN algorithm. Section 3.3 analyzes the spatial and temporal cost of DOLPHIN. Finally, section 3.4 presents a thorough experimental activity, including comparison with state of the art outlier detection methods.

The contribution of this thesis in the context of the outlier detection in data can be summarized as follows:

- DOLPHIN, a novel distance-based outlier detection algorithm, is presented, capable on working on huge disk-resident datasets, and having I/O cost corresponding only to the cost of sequentially reading two times the input dataset file;
- both theoretical justification and experimental evidence that, for meaningful combinations of the parameters  $R$  and  $k$ , the number of objects to be retained in memory by DOLPHIN in order to accomplish its task amounts to a small fraction of the dataset, is provided;
- the DOLPHIN algorithm easily integrates database indexing techniques. Indeed, by indexing the objects stored in main memory, the neighbors of the dataset objects are searched as efficiently as possible. Importantly, this task is accomplished without needing to preliminarily index the whole dataset as done by other methods;
- the strategy pursued by DOLPHIN allows to have expected near linear CPU time, for suitable combinations of the parameters  $R$  and  $k$ ;
- DOLPHIN is very simple to implement and it can be used with any type of data;

**Algorithm DOLPHIN**

1. Build an empty DBO-index INDEX
2. Perform the first scan of the data set  $\mathcal{DB}$ :
  - a) for each object  $obj$  of  $\mathcal{DB}$ :
    - i. associate with it a DBO-node  $n_{curr}$
    - ii. perform a range query search with center  $obj$  and radius  $R$  into INDEX. For each node  $n_{index}$  returned by the range query:
      - A. if  $\text{dist}(obj, n_{index}.obj) \leq R - n_{index}.rad$ , then stop search and report  $obj$  as an inlier [PR1]
      - B. if  $\text{dist}(obj, n_{index}.obj) \leq R$ , then
        - $oldrad = n_{index}.rad$
        - update the list  $n_{index}.nn$  with the object  $obj$
        - if  $oldrad > R$  and  $n_{index}.rad \leq R$  then, with probability  $1 - p_{inliers}$ , remove the node  $n_{index}$  from INDEX [PR2]
        - update the list  $n_{curr}.nn$  with the object  $n_{index}.obj$
        - if  $n_{curr}.rad \leq R$  then stop search and report  $obj$  as an inlier [PR3]
    - iii. if the range query finishes without reporting  $obj$  as an inlier, then insert  $n_{curr}$  into INDEX
  - b) Remove from INDEX all the nodes  $n$  such that  $n.rad \leq R$
3. Perform the second scan of the data set  $\mathcal{DB}$ :
  - a) For each object  $obj$  of  $\mathcal{DB}$ , perform a range query search with center  $obj$  and radius  $R$  into INDEX. For each node  $n_{index}$  returned by the range query:
    - i. if  $\text{dist}(obj, n_{index}.obj) \leq R$  then
      - A. update the list  $n_{index}.nn$  with  $obj$ , taking care of avoiding duplicates
      - B. if  $n_{index}.rad \leq R$  the delete  $n_{index}$  from INDEX
4. The objects  $n.obj$  stored into the remaining nodes  $n$  of INDEX are the outliers of  $\mathcal{DB}$

**Fig. 3.1.** The DOLPHIN distance-based outlier detection algorithm.

- DOLPHIN has been compared with state of the art distance-based outlier detection algorithms, specifically SNIF, ORCA, and RBRP, proving itself more efficient than these ones.

### 3.2 Algorithm

In this section the algorithm DOLPHIN is described. The algorithm uses a data structure called DBO-index (where DBO stands for Distance Based Outlier) defined next.

Let  $\mathcal{DB}$  be a disk-resident dataset. First of all the definition of DBO-node is provided.

**Definition 4.** A *DBO-node*  $n$  is a data structure containing the following information:

- $n.obj$ : an object of  $\mathcal{DB}$ ;

- $n.id$ : the record identifier of  $n.obj$  in  $\mathcal{DB}$ ;
- $n.nn$ : a list consisting of at most  $k - 1$  pairs  $(id, dst)$ , where  $id$  is the identifier of an object of  $\mathcal{DB}$  lying at distance  $dst$ , not greater than  $R$ , from  $n.obj$ ;
- $n.rad$ : the greatest distance  $dst$  stored in  $n.nn$  ( $n.rad$  is  $+\infty$  if less than  $k - 1$  pairs are stored in  $n.nn$ ).

A DBO-index is a data structure based on *DBO*-nodes, as defined in the following.

**Definition 5.** A *DBO-index* INDEX is a data structure storing *DBO*-nodes and providing a method `range query search` that, given an object  $obj$  and a real number  $R > 0$ , returns a (possibly strict) superset of the nodes in INDEX associated with objects whose distance from  $obj$  is not greater than  $R$ .<sup>1</sup>

Figure 3.1 shows the algorithm DOLPHIN. The algorithm performs only *two* sequential scans of the dataset.

During the first scan, the DBO-index INDEX is employed to maintain a summary of the portion of the dataset already examined. In particular, for each dataset object  $obj$ , the nodes already stored in INDEX are exploited in order to attempt to prove that  $obj$  is an inlier. The object  $obj$  will be inserted into INDEX only if, according to the schema described in the following, it will be impossible to determine that it is an inlier. Note that, by adopting this strategy, it is guaranteed that INDEX contains at least all the true outliers encountered while scanning the dataset.

After having picked the next object from the dataset, first of all, a range query search with center  $obj$  and radius  $R$  is performed into INDEX, and, for each DBO-node  $n_{index}$  encountered during the search, the distance  $dst$  between  $obj$  and  $n_{index}.obj$  is computed.

Since  $n_{index}.rad$  is the radius of a hyper-sphere centered in  $n_{index}.obj$  and containing at least  $k - 1$  dataset objects other than  $n_{index}.obj$ , if  $dst \leq R - n_{index}.rad$  then within distance  $R$  from  $obj$  there are at least  $k$  objects and  $obj$  is not an outlier. In this case, the range query is stopped and the next dataset object is considered.

Thus, this rule is used to *early prune* inliers. The more densely populated is the region the object lies in, the higher the probability of being recognized as an inlier through this rule. This can be intuitively explained by noticing that the radius of the hyper-spheres associated to the objects lying in its proximity is inversely proportional to the density of the region. This is the first rule used by the algorithm to recognize inliers. Since other rules will be used to reach the same goal, this one will be called Pruning Rule 1 (PR1 for short).

---

<sup>1</sup> More precisely, assume that the method *range query search* is implemented through two functions, i.e.  $getFirst(obj, R)$ , returning the first node of the result set, and  $getNext(obj, R)$ , returning the next node of the result set, so that candidate neighbors are computed one at a time.

Otherwise, if  $dst \leq R$  then the list  $nn_{index.nn}$  ( $nn_{curr.nn}$ , resp.) of the nearest neighbors of  $n_{index.obj}$ , ( $n_{curr.obj}$  resp.) is updated with  $n.obj$  ( $n_{index.obj}$  resp.). In particular, updating a  $n.nn$  list with a neighbor  $obj$  of  $n.obj$ , consists in inserting in  $n.nn$  the pair  $(obj, dst)$ , where  $dst$  is the distance between  $obj$  and  $nn.obj$ . Furthermore, if after this insertion  $n.nn$  contains more than  $k - 1$  object, than the pair  $(obj', dst')$  in  $n.nn$  having the greatest value for  $dst'$  must be deleted from  $n.nn$ .

After having updated the nearest neighbors lists, if the radius  $n_{index.rad}$  becomes less than  $R$ , then  $n_{index.obj}$  is recognized as an inlier. For clarity, these objects are called in the following *proved inliers*. In this case there are two basic strategies to adopt.

According to the first one, the node  $n_{index}$  is removed from INDEX since it is no longer a candidate outlier (recall that an object is inserted into INDEX only if it is not possible to determine that it is an inlier). This strategy has the advantage of releasing space as soon as it is not strictly needed, and maybe of making cheaper the cost of the range query search (when its cost is related to the size of INDEX), but it may degrade inlier detection capabilities since the PR1 becomes ineffective. Indeed, if this strategy is used, the field  $n.rad$  of each node  $n$  stored into INDEX is always  $+\infty$ , otherwise the node  $n$  has to be cancelled from INDEX. According to the second strategy, the node  $n_{index}$  is maintained into the index since it can help to detect subsequent dataset inliers through the PR1.

In between the above two strategies, there is a third intermediate one, that is to maintain only a percentage of the proved inliers. In particular, even though the latter strategy makes the PR1 effective, it must be said that it may introduce an high degree of *redundancy*. Being real data clustered, often objects share neighbors with many other dataset objects. Thus, it is better to maintain not all the proved inliers, but only a portion of them, say  $p_{inliers}$  percent. According to this third strategy, if  $n_{index.rad}$  is greater than  $R$  before updating  $n_{index.nn}$ , but becomes less or equal to  $R$  after updating  $n_{index.nn}$ , then, with probability  $p_{inliers}$ , the node  $n_{index}$  is maintained into INDEX, while with probability  $(1 - p_{inliers})$  it is removed from INDEX. This pruning rule will be referred to, in the following, as PR2. The effect of the parameter  $p_{inliers}$ , on the size of INDEX and on the ability in early recognizing inliers, will be studied in the following.

As for the current dataset object  $obj$ , if  $n_{curr.rad}$  becomes less or equal to  $R$ , then it is recognized as an inlier. In this case the range query is stopped, the object is not inserted into INDEX (this is the third pruning rule of inliers, PR3, for short, in the following), and the next dataset object is considered.

This completes the description of the first dataset scan. When the first dataset scan terminates, INDEX contains a *superset* of the dataset outliers. The goal of the second scan is to single out the true outliers among the objects stored in INDEX. Since the proved inliers stored in INDEX at the end of the first scan are no longer useful, they are removed from INDEX before starting the second dataset scan.

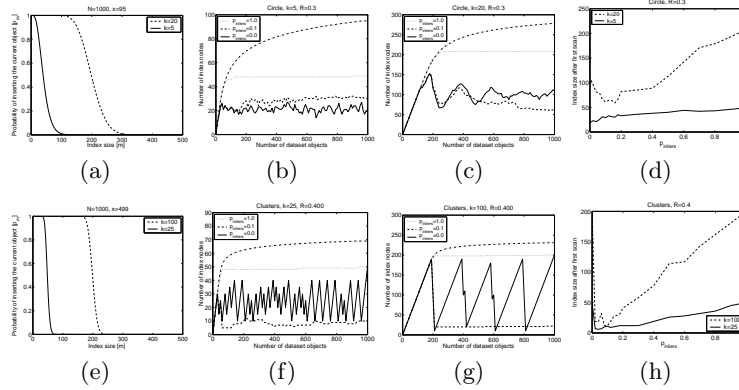


Fig. 3.2. Analysis of the algorithm.

During the second scan, for each dataset object  $obj$ , a range query search with center  $obj$  and radius  $R$  is performed into INDEX. This returns at least all the objects  $n_{index.obj}$  of INDEX such that  $obj$  lies in their neighborhood of radius  $R$ . Thus, if  $dst$ , the distance between  $n_{index.obj}$  and  $obj$ , is less or equal to  $R$ , the list of the nearest neighbors of  $n_{index.obj}$  can be updated using  $obj$ . If  $n_{index.rad}$  becomes less or equal to  $R$  then  $n_{index.obj}$  is a proved inlier and it is removed from INDEX.

At the end of the second dataset scan, INDEX contains all and only the outliers of  $\mathcal{DB}$ . It can be concluded that, provided INDEX is a DBO-index, DOLPHIN returns all and only the outliers of  $\mathcal{DB}$  after two sequential scans of  $\mathcal{DB}$ .

### 3.3 Analysis of the algorithm

It immediately follows from the description of the algorithm, that the *I/O cost* of DOLPHIN corresponds to the *cost of sequentially reading the input dataset file twice*. This cost is really negligible even for very large datasets.

As for the CPU cost, it is related to the size of INDEX. Next it is investigated how the size of INDEX changes during the execution of the algorithm.

#### 3.3.1 Spatial cost analysis

Interestingly, it can be provided evidence that, for meaningful combinations of the parameters  $k$  and  $R$ , even in the worst case setting where objects are not clustered and are never removed from INDEX, the size of any DBO-index INDEX at the end of DOLPHIN is only a fraction of the overall dataset.

Let  $N$  be the number of objects of the dataset  $\mathcal{DB}$ . Say  $p_m$  the probability that the current object of  $\mathcal{DB}$  will be inserted into INDEX when it has size

$m$ , and assume that  $p_{inliers}$  is one, so that no node inserted into INDEX is removed during the first scan.

Let  $Y_m$  be a random variable representing the number of objects to be scanned to insert a novel object into INDEX when it already contains  $m$  objects. Assuming that the neighbors of a generic object are randomly picked from the dataset, i.e. that no relationship holds among the neighbors of the dataset objects, the problem is equivalent to a set of independent Bernoulli trials, and

$$\text{pr}(Y_m = y) = p_m(1 - p_m)^{(y-1)}.$$

Hence, the expected number of objects to be scanned before inserting a novel object, when the index has size  $m$  is

$$\mathbf{E}[Y_m] = \sum_{y=1}^N y \cdot \text{pr}(Y_m = y) = \frac{1}{p_m}.$$

Consequently

$$s_m = \sum_{i=1}^m \mathbf{E}[Y_i] = \frac{1}{p_1} + \dots + \frac{1}{p_m} \quad (3.1)$$

is the expected number of dataset objects to be scanned in order to insert  $m$  nodes into INDEX. It can be concluded that the expected size of INDEX at the end of the first phase is  $S_N = \max\{m \mid 1 \leq m \leq N \text{ and } s_m \leq N\}$ .

Now we are interested in validating the above analysis by studying the growth of this function, and in empirically demonstrating that the value  $S_N$  is a worst case, since the size of INDEX is noticeable smaller for  $p_{inliers}$  less than one.

For simplicity, assume that each dataset object has approximately  $\bar{x}$  neighbors. Also, assume that the outliers form a small fraction of the dataset, so that their presence can be ignored in the analysis. With this assumption, the probability that an inlier of  $\mathcal{DB}$ , but out of INDEX, is inserted into INDEX (recall that it is supposed the neighbors of a generic object are randomly picked objects of the dataset) is

$$p_m = \frac{\sum_{j=0}^{k-2} \binom{n-\bar{x}}{m-j} \binom{\bar{x}-1}{j}}{\binom{n-1}{m}} \quad (3.2)$$

which is the probability, conditioned to the fact that the current object is an inlier, that among the  $m$  objects of INDEX there are less than  $k-1$  neighbors of  $obj$  ( $p_1 = \dots = p_{k-2} = 1$ , by definition).

Two synthetical datasets and a radius  $R$  such that  $\bar{x}$  is almost constant for all the objects, were considered. The first dataset, named *Circle*, is composed by 1,000 points equally ranged over a circumference of diameter 1.0, plus a single outlier in the center of the circle. For  $R = 0.3$  each point has  $\bar{x} = 95$  neighbors. The second dataset, named *Clusters*, is composed by two well-separated uniform clusters, having 499 points each, plus two outliers. For

$R = 0.4$  each object has  $\bar{x} = 499$  neighbors, that are all the objects of the cluster it belongs to.

Figure 3.2(a) shows, for the *Circle* dataset, the probability  $p_m$  of inserting an inlier into INDEX, computed by using formula (3.2) with  $N = 1,000$ ,  $\bar{x} = 95$ , and  $k = 5$  (solid line) and  $k = 20$  (dashed line). Note that there exists a limit on the index size beyond which the probability of inserting becomes close to zero. Figures 3.2(b) and 3.2(c) show the expected size of INDEX versus the number of dataset objects processed (dashed-pointed line) computed by using formula (3.1). Figures 3.2(b) and 3.2(c) also show the actual size of INDEX for varying values of the parameter  $p_{inliers}$ , that is  $p_{inliers} = 0$  (solid line),  $p_{inliers} = 0.1$  (dashed line), and  $p_{inliers} = 1$  (dotted line).

Interestingly, if objects inserted in INDEX are never removed ( $p_{inliers} = 1$ ), then the observed final size of INDEX is below the value  $S_N$ . This behavior can be explained by noting that in real data common neighbors are biased. Moreover, if  $p_{inliers}$  is decreased, then the final size of INDEX may be noticeably smaller than the value  $S_N$ , even a little fraction of the overall dataset. This behavior is confirmed by Figure 3.2(d), showing the size of INDEX at the end of the first phase of DOLPHIN as a function of  $p_{inliers}$  for  $k = 5$  (solid line) and  $k = 20$  (dashed line).

Even though one may expect that by deleting all the proved inliers ( $p_{inliers} = 0$ ) the size of INDEX should be smaller (see Figure 3.2(b) for an example of this behavior), it was observed that, in terms of maximum size of the index, the best value of  $p_{inliers}$  is either a value close to 0.1 or exactly zero, depending on the characteristics of the dataset.

Since, for meaningful combinations of the parameters  $k$  and  $R$ , very often the mean number of objects lying in spheres of radius  $R$  is high if compared with  $k$ , as we shall see, a value for  $p_{inliers}$  slightly greater than zero is optimal in practice in terms of execution time, and, sometimes, also in terms of index size. As an example, see Figure 3.2(c). For  $p_{inliers} = 0$ , after INDEX has accumulated enough objects to recognize inliers, the algorithm deletes the proved inliers from INDEX and forgets what has already seen. The size of the index becomes thus oscillating. The greater is the value of  $k$ , and the greater is the fluctuation of the size. On the contrary, for  $p_{inliers} = 0.1$ , after having accumulated in INDEX enough objects to recognize inliers, a large portion of proved inliers is removed but, since the algorithm has *learned* the dense regions of the feature space (this information is stored together with the proved inliers left into the index), the PR1 is applied efficiently and the size of the index stabilizes on a small fraction of the dataset. Using a greater value for  $p_{inliers}$  has the effect of increasing the size of INDEX, but does not augment the number of objects pruned by the PR1.

This behavior is much more evident on the dataset *Clusters*. Figures 3.2(e), 3.2(f), 3.2(g), and 3.2(h) report, on the *Clusters* dataset, the same kind of experiments described above ( $R = 0.4$ ,  $\bar{x} = 499$ ). For  $p_{inliers} = 0$ , the fluctuation is now more pronounced as the objects are strongly clustered and the parameter  $k$  is relatively large. Note that, for  $p_{inliers} = 0.1$  the final size of INDEX

is a small fraction of the dataset. It can be observed in Figure 3.2(h) that for  $p_{inliers} = 0$  the size of the index is exactly  $2k$ , as there are two clusters and at least  $k$  objects have to be seen in each cluster in order to recognize all the inliers.

Summarizing, in practice, the size of INDEX does not reach the value  $S_N$  (we measured this value only by simulating a dataset where nearest neighbors are randomly picked objects). Furthermore, as real data is clustered, by properly tuning the parameter  $p_{inliers}$ , only a small fraction of the dataset is inserted into the index. As will be confirmed in the following, setting approximately  $p_{inliers}$  to 0.1 is a good trade-off between index size (amount of redundant information into the index) and execution time (efficient application of the PR1).

### 3.3.2 Implementation and time cost analysis

DOLPHIN uses as INDEX data structure a *pivot-based* index [Chávez et al., 2001]. A pivot-based index is a data structure performing proximity search in any metric space. It makes use of a certain number of objects (also called *pivots*), usually selected among the objects stored into the index. Distances among pivots and objects stored into the index are precalculated at indexing time. When a range query with center  $obj$  and radius  $R$  is submitted to the index, the distances between the pivots and the query object  $obj$  are computed. The precalculated distances are exploited to recognize the index objects lying at distance greater than  $R$  from  $obj$ . For each index object  $obj'$ , if there exists a pivot  $p$ , such that  $|\text{dist}(obj, p) - \text{dist}(obj', p)| > R$ , then, by the triangular inequality, it is known that  $\text{dist}(obj, obj') > R$ , and  $obj'$  is ignored. Otherwise it is returned as candidate neighbor of  $obj$ . By using this kind of technique the range query search returns a list of candidates that is a superset of the objects that truly satisfy the query. The performances of pivot-based indexes are related to the number of employed pivots. In particular, the larger is the number of pivots, the more accurate (and then smaller) is the list of returned candidates, but the cost of querying and building the index clearly increases.

Now we want to depict a qualitative analysis of the expected temporal cost of DOLPHIN. The execution time of the first scan of DOLPHIN is given by the sum of the following three terms<sup>2</sup>:

$$N \cdot (C_{query} + S_{query} \cdot d) + N_{ins} \cdot C_{ins} + N_{del} \cdot C_{del} \quad (3.3)$$

where  $N$  is the number of dataset objects,  $C_{query}$  is the mean temporal cost of executing a range query search during the first phase,  $S_{query}$  is the mean number of candidate neighbors examined per dataset object during the first phase,  $d$  is the cost of computing a distance,  $N_{ins}$  ( $N_{del}$ , resp.) is the number of insertions (deletions, resp.) into INDEX during the first scan, and  $C_{ins}$

<sup>2</sup> This formula ignores minor costs due to non dominating operations, like updating the list of nearest neighbors and so on.



( $C_{del}$ , resp.) is the mean cost of insertion (deletion, resp.) during the first phase.

The cost of the second scan can be expressed with a similar formula.

Consider the first term  $N \cdot (C_{query} + S_{query} \cdot d)$  of Formula (3.3). It follows from the discussion of previous section, that we can roughly approximate the maximum size of INDEX to  $\mathcal{O}(\alpha N)$ , where  $\alpha$  is a small constant, and, hence, also the term  $S_{query}$  is upper bounded by  $\mathcal{O}(\alpha N)$ . The cost  $C_{query}$  of executing the range query search amounts to computing the distances between a dataset object and all the pivots plus the cost of pruning index objects. Since the number of pivots we used is logarithmic in the size of the index this cost is upper bounded by  $\mathcal{O}(d \log \alpha N + d\alpha N)$ . As a whole the term  $N \cdot (C_{query} + S_{query} \cdot d)$  can be approximated by  $\mathcal{O}(d\alpha N^2)$ . As for the cost of removing an object from a pivot-based index, note that it is constant (practically it consists in flagging as empty the entry of an array), thus the term  $N_{del} \cdot C_{del}$  of Formula (3.3) can be ignored. Finally, as for the insertion an object  $obj$  into the index, it requires to compute the distances among  $obj$  and all the pivots. However, since insertion is always performed after the range query search, these distances are already available and, then, insertion has virtually no cost.

Summarizing, the temporal cost of the algorithm can be roughly approximated to  $\mathcal{O}(d\alpha N^2)$ . Basically, this expression assumes that all the objects in the dataset are compared with all the objects in INDEX and, then, it does not take into account that the range query search returns only a subset of the nodes of INDEX. Furthermore, the above analysis does not take into account that DOLPHIN does not need to find  $k$  neighbors since it is able to exploit the triangular inequality (through the PR1) to early prune inliers.

Bay and Schwabacher [2003] showed that, if the dataset is randomized, then the expected number of objects to be compared with each dataset object in order to find its  $k$  nearest neighbors is a constant depending only on the values of  $k$  and  $\bar{x}$ . Based on this property, ORCA performs in near linear CPU time. The above upper bound on the number object comparisons is also applicable to DOLPHIN, since INDEX stores a random sample of inliers together with the true outliers. Nevertheless, DOLPHIN greatly reduces this number by exploiting both range searching and pruning rules. Furthermore, ORCA performs a block nested loop reading of disk pages which may lead to a quadratic I/O cost, while DOLPHIN needs only two sequential readings of the dataset.

This expected behavior will be confirmed by experimental results reported in the following section.

### 3.4 Experimental Results

Experiments are organized as follows. First of all, it is analyzed the course of the size of INDEX. Then, it is determined how the parameter  $p_{inliers}$  affects

the execution time, and a study of the sensitivity to the parameters  $R$  and  $k$  is accomplished. Next, DOLPHIN is compared with other outlier detection algorithms and, finally, experiments on a massive dataset are presented.

**Employed datasets.** Datasets employed in the experiments are summarized in the following table and briefly described next.

<i>Dataset</i>	<i>Objects</i>	<i>Attributes</i>
<i>Color Histogram</i>	68,040	32
<i>DARPA 1998</i>	499,467	23
<i>Forest Cover Type</i>	581,012	54
<i>Household</i>	1,000,000	3
<i>Landsat</i>	275,465	60
<i>Server</i>	500,000	5
<i>Mixed Gauss</i>	18,000,000	30

*Color Histogram* contains image features extracted from a Corel image collection<sup>3</sup>. *DARPA 1998* consists in network connection records, from five weeks of training data, of intrusions simulated in a military network environment [DARPA, 1998] (also referred to as *Weeks* in the following). *Forest Cover Type* contains forest cover type data determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data<sup>4</sup>. *Household* is released by the US Census Bureau and contains the annual expenditure of American families on electricity, gas, and water, as described in [Tao et al., 2006]. *Landsat* contains normalized feature vectors associated with tiles of a collection of large aerial photos<sup>5</sup>. *Server* is an excerpt of the KDD Cup 1999 data, as described in [Tao et al., 2006]. Finally, *Mixed Gauss* is a synthetic dataset described in the last part of the section.

**Course of the index size.** According the methodology suggested in [Tao et al., 2006], in most of the experiments reported in the following, for the parameter  $k$  it will be employed a value ranging from the 0.02% to the 0.1% of the dataset size, while, for the parameter  $R$ , the value employed will range in the interval  $[R_{\min}, R_{\max}]$ , where  $R_{\min}$  ( $R_{\max}$ , resp.) is the radius corresponding to exactly one outlier (the 0.1% dataset size of outliers, resp.) when  $k$  is set to the 0.05%.

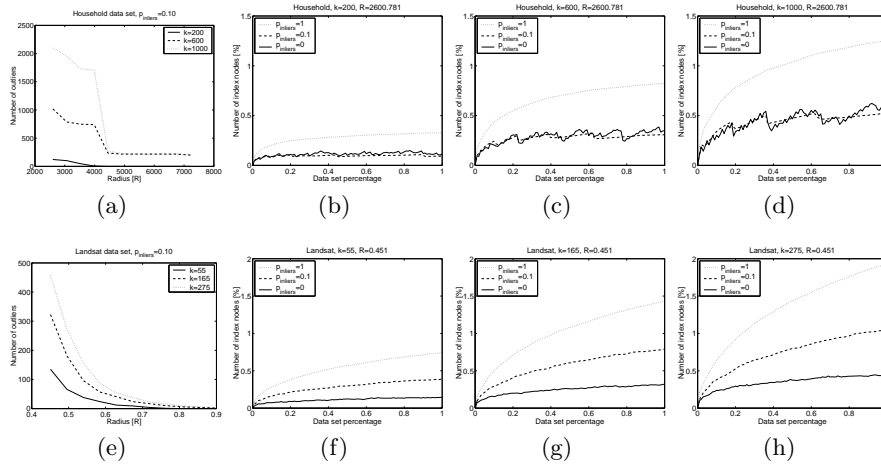
Next, it is studied the course of the size of INDEX during the execution of DOLPHIN on the two real datasets *Household* and *Landsat*. In particular, Figures 3.3(a) and 3.3(e) show, for three values of  $k$  in the range above mentioned, the number of outliers versus the radius  $R$ . These curves provide an idea of the severity of the problem in correspondence of various combinations of the parameters  $R$  and  $k$ .

Figures 3.3(b), 3.3(c), 3.3(d), and 3.3(f), 3.3(g), 3.3(h) show the size of INDEX versus the percentage of objects read from the input dataset. Note

<sup>3</sup> See <http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html>.

<sup>4</sup> See <http://kdd.ics.uci.edu/databases/coverttype/coverttype.html>.

<sup>5</sup> See <http://vision.ece.ucsb.edu>



**Fig. 3.3.** Course of the size of INDEX.

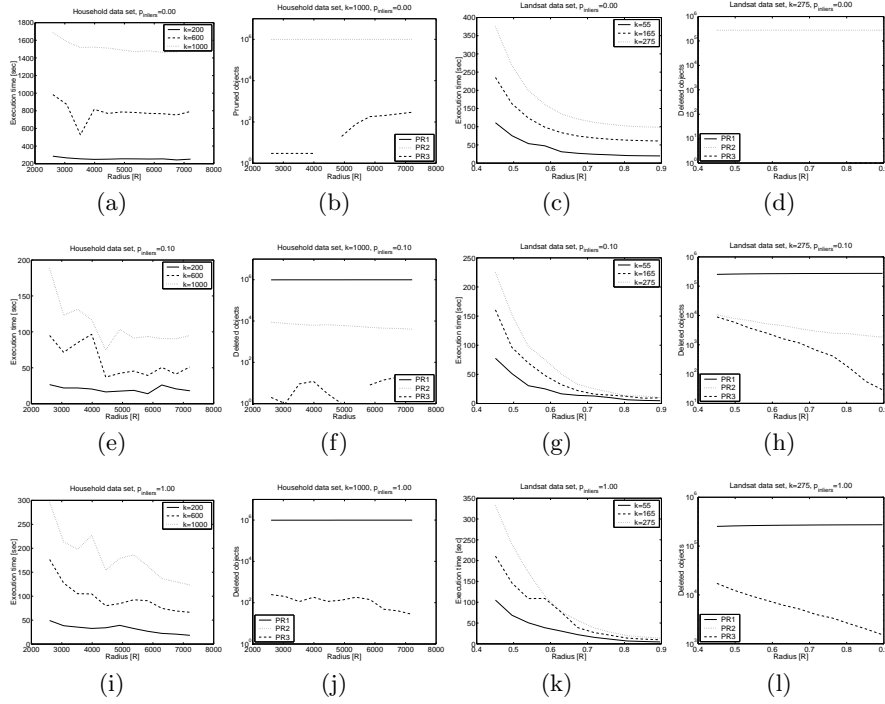
that in all cases, as predicted, the growth of the size of INDEX slows down as the percentage of seen dataset objects increases.

On the *Household* dataset, by using  $k = 1,000$  and  $R = 2,600$  (for these parameters there are more than 2,000 outliers, see Figure 3.3(a)), when  $p_{inliers} = 1$  INDEX contains at most the 2% of the dataset objects, while for  $p_{inliers}$  equal to 0 or 0.1 the maximum size of INDEX is approximatively the 0.5% of the dataset (see Figure 3.3(d)). For  $p_{inliers} = 0$  the size of INDEX is oscillating, while for  $p_{inliers} = 0.1$  it is more stable, confirming the analysis of the previous section. From these curves it is clear that, on this dataset, in terms of index size, setting  $p_{inliers}$  to 0.1 is better than having it set to 0, as the former value avoids fluctuations of the size. For smaller values of  $k$ , the maximum size of INDEX sensibly decreases (see Figures 3.3(b) and 3.3(c)). For example, when  $k = 200$  the size is always below 0.5%.

As for the *Landsat* dataset (see Figures 3.3(f), 3.3(g), and 3.3(h)), the maximum size of INDEX is always directly proportional to the value of the parameter  $p_{inliers}$ . This behavior will be observed also on other datasets. For  $R = 0.451$  and  $k = 275$  (see Figure 3.3(h)), in the worst case, the size of index reaches the 2%, but it is about the 1% for  $p_{inliers} = 0.1$ , and about the 0.5% for  $p_{inliers} = 0$ .

**Execution time and effectiveness of pruning rules.** Figure 3.4 shows the execution time<sup>6</sup> of DOLPHIN on the *Household* and *Landsat* datasets, together with the number of times the pruning rules PR1, PR2, and PR3 are applied during the first scan of the algorithm. From the top to the bottom, curves displayed are obtained by setting  $p_{inliers}$  to 0, 0.1, and 1, respectively.

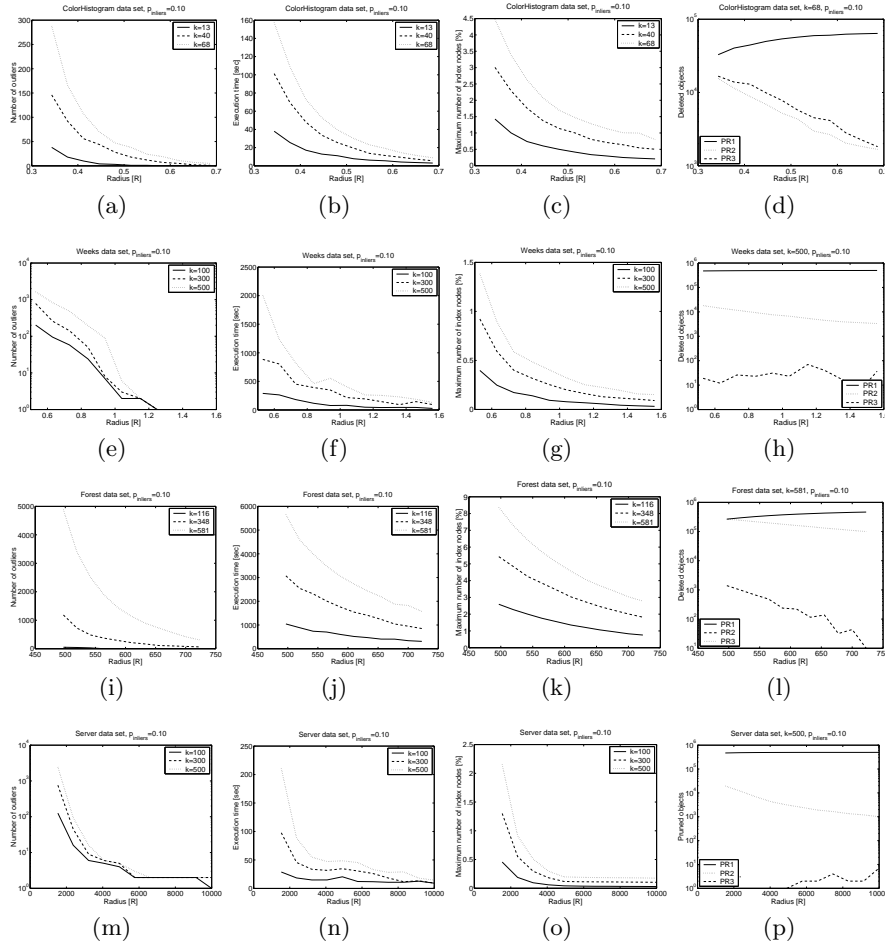
<sup>6</sup> We used a Pentium IV Dual Core 3.4GHz based machine with 1GByte of main memory and the Linux operating system.



**Fig. 3.4.** Execution time and effectiveness of pruning rules.

Note that  $p_{inliers} = 0$  gives the worst performance, while  $p_{inliers} = 0.1$  gives the best one (see Figures 3.4(a), 3.4(e), 3.4(i), for the *Household* dataset, and 3.4(c), 3.4(g), 3.4(k), for the *Landsat* dataset).

In order to understand this behavior, it is useful to study how frequently the pruning rules, i.e. PR1, PR2, and PR3, are used (see Figures 3.4(b), 3.4(f), 3.4(j), and 3.4(d), 3.4(h), 3.4(l)). For  $p_{inliers} = 0$ , almost all the objects are pruned by PR2, that is, almost all the objects read from disk are firstly inserted into INDEX, and, subsequently, deleted from INDEX after having seen their  $k$  neighbors. On the contrary, by using  $p_{inliers} = 0.1$ , almost all the inliers are pruned by the PR1, but also the PR2 and PR3 may be effective, even if the number of objects pruned by the last two rules is more than an order of magnitude smaller. Finally, using a greater value for  $p_{inliers}$ , has the effect of increasing the size of INDEX, but without significantly augmenting the number of objects pruned by the PR1. Thus, due to the presence of redundant objects, the range query performs worse, and the total execution time increases. Interestingly, by forgetting most of the information already seen, both the spatial and temporal complexities are improved. The best trade-off between space occupancy and execution time appears to be achieved for  $p_{inliers} \approx 0.1$ , and this value will be used in the subsequent experiments.

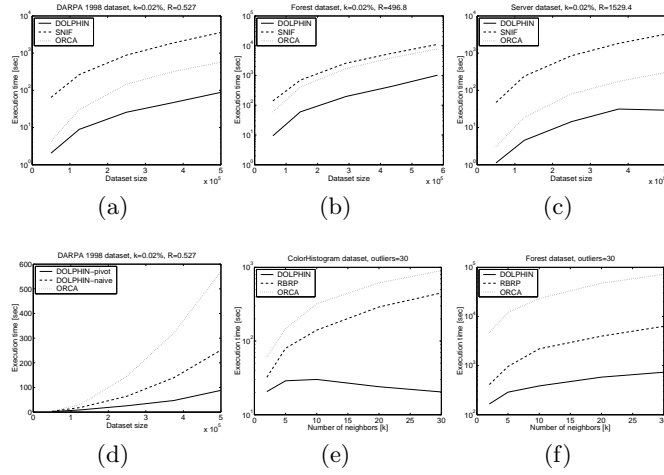


**Fig. 3.5.** Sensitivity to parameters  $R$  and  $k$ .

Note that when the parameter  $k$  is relatively small,  $p_{inliers} = 0$  can give performances similar to  $p_{inliers} = 0.1$  (see, e.g., Figures 3.4(c) and 3.4(g) for  $k = 55$ ). Thus, in some cases, deleting all proved inliers may provide advantages in terms of space<sup>7</sup> with a moderate loss in terms of execution time. Nevertheless, as  $k$  increases, deleting all the proved inliers may greatly degrade performances. See, for example, Figures 3.4(a) and 3.4(e) for  $k = 1,000$ .

**Sensitivity to parameters  $R$  and  $k$ .** Figure 3.5 shows the behavior of DOLPHIN on the datasets *ColorHistogram*, *DARPA 1998*, *Forest Cover Type*, and *Server* for various values of  $k$  and  $R$ .

<sup>7</sup> Indeed, the number of nodes in INDEX is fewer, and it is not required to store the list  $n.nn$  in the nodes  $n$  of INDEX.



**Fig. 3.6.** Comparison with other methods.

Figures 3.5(a), 3.5(e), 3.5(i), and 3.5(m) show the number of found outliers. The curves of execution time (see Figures 3.5(b), 3.5(f), 3.5(j), and 3.5(n)) confirm the behavior previously observed on other datasets. The algorithm has good performances even when large values of  $k$  and small values of  $R$  are employed, and, consequently, a considerable number of outliers is found.

The maximum size of INDEX (see Figures 3.5(c), 3.5(g), 3.5(k), and 3.5(o)) is always a small fraction of the dataset, up to the 1.5% or 5% depending on the dataset and the parameters. There is a peak of the 8.5% for the *Forest Cover Type* dataset in correspondence of  $k = 581$  and  $R = 496.8$ . This is due to the very large number of outliers detected (about 5,000 objects). Even in this case, if a smaller number of outliers is searched for, e.g. one thousand, then the maximum index size is about the 4%.

As for the pruning rules (see Figures 3.5(d), 3.5(h), 3.5(l), and 3.5(p)), the PR1 is effective on all the combinations of the parameters. For *Forest Cover Type* with  $k = 581$ , when the radius  $R$  decreases, the PR3 is applied as frequently as the PR1. This indicates that an high fraction of the data lies in relatively sparse regions of the space for that value of  $R$ , and explains the very high number of outliers.

**Comparison with other methods.** Figure 3.6 shows the comparison with the distance-based outlier detection algorithms SNIF, ORCA, and RBRP.

We compared DOLPHIN with SNIF<sup>8</sup> and ORCA<sup>9</sup> through scaling analysis. Figures 3.6(a), 3.6(b), and 3.6(c) show the execution time of DOLPHIN, SNIF, and ORCA on the *DARPA 1998*, *Forest Cover Type*, and *Server* datasets, respectively. The parameter  $k$  was set to the 0.02% of the dataset

<sup>8</sup> <http://www.cse.cuhk.edu.hk/~taoyf/paper/kdd06.html>

<sup>9</sup> <http://www.isle.org/~sbay/software/orca/>

size, while  $R$  was fixed to the value  $R_{\min}$  previously described. The number of centroids employed by SNIF was 1,000 while the memory buffer used was the 10% of the dataset size, as suggested by the authors. As for ORCA, we set the cutoff value to  $R$ , and the number of outliers to be found to a value greater than the number of distance outliers present in the dataset. All the experiments showed the same behavior, that is DOLPHIN is faster than ORCA of about one order of magnitude, while ORCA is faster than SNIF. In two experiments DOLPHIN is faster than SNIF of about two orders of magnitude, while in the remaining one of about one order of magnitude.

We performed the same type of experiment on the *DARPA 1998* dataset, but with an implementation of DOLPHIN that uses the *naive index*. The curve of the execution time of this implementation (called DOLPHIN-naive) is reported in Figure 3.6(d) together with the curves of DOLPHIN using the pivot-based index (there called DOLPHIN-pivot) and of ORCA. It is clear that DOLPHIN gains in efficiency by exploiting indexing techniques though, even if it is not used, it always maintains great performances and performs noticeably better than ORCA. This can be explained since the I/O cost of DOLPHIN is lower than that of ORCA, and since, even if DOLPHIN uses the naive index, it is still able to apply the pruning rules (specifically, the PR1)

We compared DOLPHIN also with RBRP, even if it must be recalled that *RBRP indexes all the dataset into the main memory*. We considered the experimental results described in [Ghoting et al., 2006] on the *ColorHistogram* and *Forest Cover Type* datasets<sup>10</sup>. Figures 3.6(e) and 3.6(f) show the execution time of DOLPHIN, RBRP, and ORCA, for various values of  $k$  when the number  $n$  of outliers to mine is 30. In order to run DOLPHIN, we computed the radiuses  $R$  corresponding to various combination of the parameters  $k \in [2, 30]$  and  $n = 30$ , and measured its execution time. The curves show that, on this type of experiment, DOLPHIN outperformed RBRP of at least one order of magnitude.

$k$	Outliers	Time [sec]	Index size	Memory usage [MB]
5	6	930.7	248	0.05
200	118	3,006.0	1,173	1.97
500	141	7,284.8	1,479	5.87
1,000	160	10,320.2	1,922	14.96

**Table 3.1.** Experiments on a massive dataset.

**Experiments on massive datasets.** We considered the *Mixed Gauss* dataset, composed by 18,000,000 objects consisting in points of  $\mathbb{R}^{30}$ . The

<sup>10</sup> We used a machine similar to that there employed in order to compare execution times.

dataset is composed of 10 gaussian clusters, having different sizes, whose centers are randomly generated in the square  $[-25, +25]^{30}$ , and whose variances are randomly chosen in the range  $[0.5, 2]$ , plus 1,000 randomly generated points in the square  $[-30, +30]^{30}$ . The dataset occupies more than 2GBytes of secondary memory and cannot be entirely stored in the main memory of the machine.

We fixed the parameter  $R$  to 25.0,<sup>11</sup> and gradually increased the value of the parameter  $k$ . Table 3.1 shows the results of these experiment. Interestingly, the memory usage was very limited due to the small number of objects that are to be stored in INDEX. The number of outliers found is significative, confirming that the parameters used are meaningful.

---

<sup>11</sup> This value amounts to about the 7% of the maximum estimated distance separating two points in the dataset.



## Outlier Detection in Streams of Data

### 4.1 Contribution

In this chapter a method for detecting distance-based outliers in data streams is proposed. The two proposed algorithms, an exact and an approximate one, are analyzed in details.

This chapter is organized as follows. This section describes the contribution given by this thesis in the outlier detection in data stream context. The following section formally states the data stream outlier query problem. Section 4.3 describes both the exact and the approximate algorithm. Subsequent section 4.4 analyzes in details the algorithms. Finally, section 4.5 illustrates experimental results.

The contribution of the thesis in this context can be summarized as follows:

- the novel task of data stream outlier query is introduced;
- an exact algorithm to efficiently detect distance-based outliers in the introduced model is presented;
- an approximate algorithm is derived from the exact one, based on a trade off between spatial requirements and answer accuracy; the method approximates object outlierness with a statistical guarantee;
- by means of experiments on both real and synthetic data sets, the efficiency and the accuracy of the proposed techniques are shown.

### 4.2 Statement of the Problem

Next, the formalism employed will be presented. First of all, the formal definition of distance-based outlier is recalled [Knorr and Ng, 1998].

**Definition 6 (Distance-Based Outlier).**

*Let  $S$  be a set of objects,  $obj$  an object of  $S$ ,  $k$  a positive integer, and  $R$  a positive real number. Then,  $obj$  is a distance-based outlier (or, simply, an outlier) if less than  $k$  objects in  $S$  lie within distance  $R$  from  $obj$ .*

Objects lying at distance not greater than  $R$  from  $obj$  are called *neighbors* of  $obj$ . The object  $obj$  is not considered a neighbor of itself.

A *data stream*  $\mathbf{DS}$  is a possible infinite series of objects  $\dots, obj_{t-2}, obj_{t-1}, obj_t, \dots$ , where  $obj_t$  denotes the object observed at time  $t$ . It will be interchangeably used the term *identifier* and the term *time of arrival* to refer to the time  $t$  at which the object  $obj_t$  was observed for the first time.

Data stream objects are assumed to be elements of a metric space on which a distance function is defined. In the following  $d$  will be used to denote the space required to store an object, and  $\Delta$  to denote the temporal cost required for computing the distance between two objects.

Given two object identifiers  $t_m$  and  $t_n$ , with  $t_m \leq t_n$ , the *window*  $\mathbf{DS}[t_m, t_n]$ , is the set of  $n - m + 1$  objects  $obj_{t_m}, obj_{t_m+1}, \dots, obj_{t_n}$ . The *size* of the window  $\mathbf{DS}[t_m, t_n]$  is  $n - m + 1$ .

Given a window size  $W$ , the *current window* is the window  $\mathbf{DS}[t-W+1, t]$ , where  $t$  is the time of arrival of the last observed data stream object.

An *expired object*  $obj$  is an object whose identifier  $id$  is less than the lower limit of the current window, i.e. such that  $id < t - W + 1$ .

Now, the main problem to be solved can be defined.

**Definition 7 (Data Stream Outlier Query).**

*Given a data stream  $\mathbf{DS}$ , a window size  $W$ , and fixed parameters  $R$  and  $k$ , the Data Stream Outlier Query is: return the distance based outliers in the current window.*

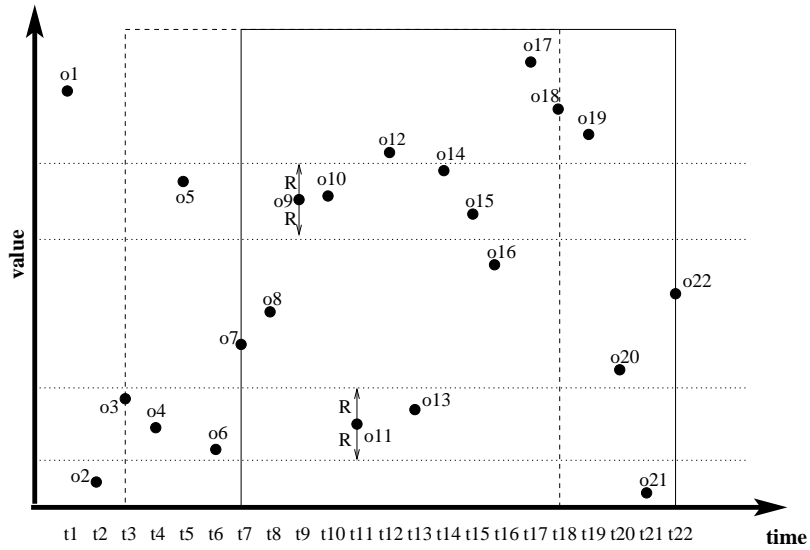
The time  $t$ , which a data stream outlier query is requested at, is called *query time*.

In the following the neighbors of an object  $obj$  that precede  $obj$  in the stream and belong to the current window are called *preceding neighbors* of  $obj$ . Furthermore, the neighbors of an object  $obj$  that follow  $obj$  in the stream and belong to the current window are called *succeeding neighbors* of  $obj$ .

According to Definition 6, an *inlier* is an object  $obj$  having at least  $k$  neighbors in the current window. In other words, let  $\alpha$  be the number of preceding neighbors of  $obj$  and  $\beta$  be the number of succeeding neighbors of  $obj$ ,  $obj$  is an inlier if  $\alpha + \beta \geq k$ .

Let  $obj$  be an inlier. Since during stream evolution objects expire, the number of preceding neighbors of  $obj$  decreases. Therefore, if the number of succeeding neighbors of  $obj$  is less than  $k$ ,  $obj$  could become an outlier depending on the stream evolution. Conversely, since  $obj$  will expire before its succeeding neighbors, inliers having at least  $k$  succeeding neighbors will be inliers for any stream evolution. Such inliers are called *safe inliers*.

*Example 2.* The following diagram represents the evolution of a one-dimensional data stream. Let  $k$  be 3, and let  $W$  be 16.



Consider the current window (the dashed one) at time  $t_{18}$ : both  $o_9$  and  $o_{11}$  are inliers, since  $o_9$  has four neighbors ( $o_5, o_{10}, o_{14}, o_{15}$ ), and also  $o_{11}$  has four neighbors ( $o_3, o_4, o_6, o_{13}$ ). Moreover, since  $o_9$  has three succeeding neighbors, it is a safe inlier, while  $o_{11}$  is not a safe inlier.

Indeed, consider instant  $t_{22}$ . The object  $o_9$  is still an inlier: object  $o_5$  expired, but  $o_9$  has still three (succeeding) neighbors. Conversely,  $o_{11}$  is now an outlier: objects  $o_3, o_4$  and  $o_6$  expired, and now it has only one neighbor.

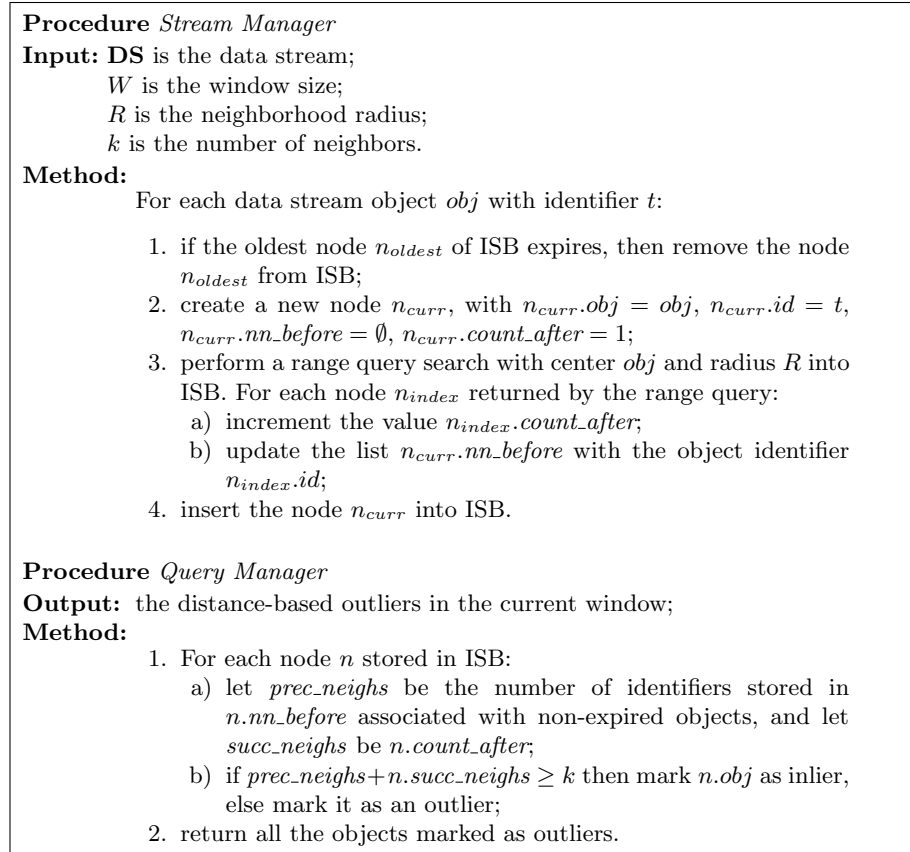
### 4.3 Algorithm

In this section the algorithm STORM, standing for SStream Outlier Miner, is described. Two variants of the method are presented.

When the entire window can be allocated in memory, the exact answer of the data stream outlier query can be computed. In the above setting, the algorithm exact-STORM is able to exactly answer outlier queries at any time (section 4.3.1).

Nevertheless, there are some applications in which only limited memory can be allocated, or interesting windows can be so large that do not fit in memory. In this case approximations must be employed. The algorithm approx-STORM is designed to work in the latter setting by introducing effective approximations in exact-STORM (section 4.3.2). These approximations guarantee highly accurate answers with limited memory requirements (section 4.4.1).

After having described the two methods, temporal and spatial costs required to obtain exact and approximate answers will be stated (section 4.4.2).



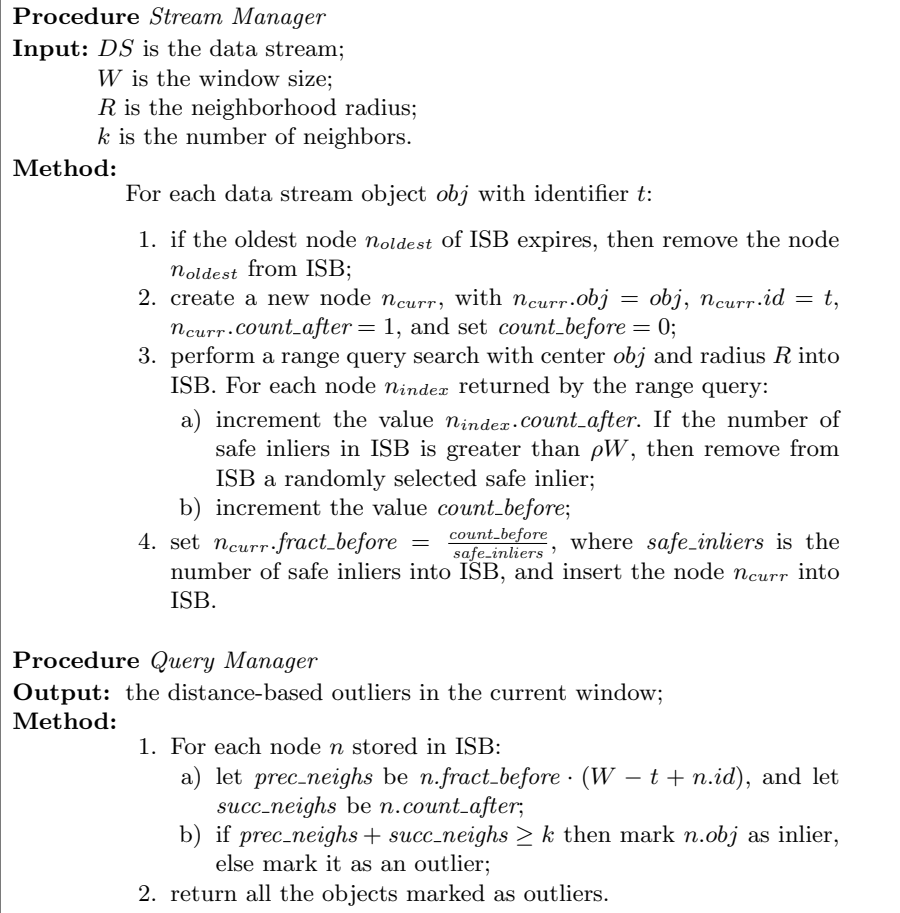
**Fig. 4.1.** Exact data stream distance-based outlier detection algorithm.

### 4.3.1 Exact algorithm

The algorithm exact-STORM is shown in Figure 4.1. This algorithm consists of two procedures: the *Stream Manager* and the *Query Manager*. The former procedure receives the incoming data stream objects and efficiently updates a suitable data structure that will be exploited by the latter procedure to effectively answer queries.

In order to maintain a summary of the current window, a data structure, called ISB (standing for *Indexed Stream Buffer*), storing *nodes* is employed (nodes are defined next). Each node is associated with a different data stream object.

ISB provides a *range query search* method, that, given an object  $obj$  and a real number  $R \geq 0$ , returns the nodes in ISB associated with objects whose distance from  $obj$  is not greater than  $R$ . The implementation of ISB will be described in section 4.4.2.



**Fig. 4.2.** Approximate data stream distance-based outlier detection algorithm.

Now, the definition of node is provided. A *node*  $n$  is a record consisting of the following information:

- $n.obj$ : a data stream object.
- $n.id$ : the identifier of  $n.obj$ , that is the arrival time of  $n.obj$ .
- $n.count\_after$ : the number of succeeding neighbors of  $n.obj$ . This field is exploited to recognize safe inliers.
- $n.nn\_before$ : a list, having size at most  $k$ , containing the identifiers of the most recent preceding neighbors of  $n.obj$ . At query time, this list is exploited to recognize the number of preceding neighbors of  $n.obj$ . Both the operation of *ordered insertion* of a novel identifier in the list and the operation of *search* of an identifier in the list are assumed to be executed in time  $\mathcal{O}(\log k)$  (see [Knuth, 1973] for a suitable implementation).

The *Stream Manager* takes as input a data stream  $\mathbf{DS}$ , a window size  $W$ , a radius  $R$ , and the number  $k$  of nearest neighbors to consider.

For each incoming data stream object  $obj$ , a new node  $n_{curr}$  is created with  $n_{curr}.obj = obj$ . Then a range query search with center  $n_{curr}.obj$  and radius  $R$  is performed in ISB, that returns the nodes associated with the preceding neighbors of  $obj$  stored in ISB.

For each node  $n_{index}$  returned by the range query search, since the object  $obj$  is a succeeding neighbor of  $n_{index}.obj$ , the counter  $n_{index}.count\_after$  is incremented. Moreover, since the object  $n_{index}.obj$  is a preceding neighbor of  $obj$ , the list  $n_{curr}.nn\_before$  is updated with  $n_{index}.id$ .

If the counter  $n_{index}.count\_after$  becomes equal to  $k$ , then the object  $n_{index}.obj$  becomes a safe inlier. Thus, it will not belong to the answer of any future outlier query. Despite this important property, a safe inlier cannot be discarded from ISB, since it may be a preceding neighbor of a future stream object. Finally, the node  $n_{curr}$  is inserted into ISB. This terminates the description of the procedure *Stream Manager*.

In order to efficiently answer queries, when invoked by the user, the *Query Manager* performs a single scan of ISB. In particular, for each node  $n$  of ISB, the number  $prec\_neighs$  of identifiers stored in  $n.nn\_before$  associated with non-expired objects is determined. This is accomplished in  $\mathcal{O}(\log k)$  time by performing a search in the list  $n.nn\_before$  of the identifier closest to the value  $t - W + 1$ , that is the identifier of the oldest object in  $n.nn\_before$  belonging to the current window.

The number  $succ\_neighs$  of succeeding neighbors of  $n.obj$  is stored in  $count\_after$ . Thus, if  $prec\_neighs + succ\_neighs \geq k$  then the object  $n.obj$  is recognized as an inlier, otherwise it is an outlier and is included in the answer of the outlier query.

### 4.3.2 Approximate algorithm

Figure 4.2 shows the algorithm approx-STORM. The exact algorithm requires to store all the window objects. If the window is so huge that does not fit in memory, or only limited memory can be allocated, the exact algorithm might be not employable. However, as described in the following, the algorithm described in the previous section can be readily modified to reduce the required space.

With this aim, two approximations are introduced.

Firstly, in order to significantly reduce the occupied space, not all the window objects are stored in ISB. In particular, objects belonging to ISB can be partitioned in outliers and inliers. Among the latter kind of objects there are safe inliers, that are objects that will be inliers in any future stream evolution. As already observed, despite safe inliers cannot be returned by any future outlier query, they have to be kept in ISB in order to correctly recognize outliers, since they may be preceding neighbors of future incoming objects.

However, as shown in subsequent section 4.4.1, it is sufficient to retain in ISB only a fraction of safe inliers to guarantee an highly accurate answer to the outlier query. Thus, in order to maintain in ISB a controlled fraction  $\rho$  ( $0 \leq \rho \leq 1$ ) of safe inliers, the following strategy is adopted.

During stream evolution, an object  $obj$  of the stream becomes a safe inlier when exactly  $k$  succeeding neighbors of  $obj$  arrive. At that time, if the total number of safe inliers into ISB exceeds  $\rho W$ , then a randomly selected object of ISB is removed. The random selection policy adopted guarantees that safe inliers surviving into ISB are uniformly distributed in the window.

To answer one-time queries both outliers and unsafe inliers, which are objects candidate to become outliers if the stream characteristics change, have to be maintained in ISB. Note that, meaningful combinations of the parameters  $R$  and  $k$  are only those for which the number of outliers, and hence of unsafe inliers, amounts to a negligible fraction of the overall population. Thus, the number of nodes in ISB can be assumed approximately equal to  $\rho W$ . In the following section it will be discussed how to compute an optimal value for  $\rho$  in order to obtain a statistical guarantee on the approximation error of the estimation of the number of preceding neighbors of each data stream object.

The second approximation consists in reducing the size of each node by avoiding storing the list of the  $k$  most recent preceding neighbors. This is accomplished by storing in each node  $n$ , instead of the list  $n.nn\_before$ , just the fraction  $n.fract\_before$  of previous neighbors of  $n.obj$  observed in ISB at the arrival time  $n.id$  of the object  $n.obj$ . The value  $n.fract\_before$  is determined as the ratio between the number of preceding neighbors of  $n.obj$  in ISB which are safe inliers and the total number of safe inliers in ISB, at the arrival time of  $n.obj$ .

At query time, in order to recognize outliers, a scan of ISB is performed and, for each stored node  $n$ , the number of neighbors of  $n.obj$  in the current window has to be evaluated. Since only the fraction  $n.fract\_before$  is stored now in  $n$ , the number of preceding neighbors of  $n.obj$  in the whole window at the current time  $t$  has to be estimated.

Let  $\alpha$  be the number of preceding neighbors of  $n.obj$  at the arrival time of  $n.obj$ . Assuming that they are uniformly distributed along the window, the number of preceding neighbors of  $n.obj$  at the query time  $t$  can be estimated as

$$prec\_neighs = \alpha \cdot \frac{W - t + n.id}{W}.$$

Note that  $n.fract\_before$  does not directly give the value  $\alpha$ , since it is computed by considering only the objects stored in ISB and, thus, it does not take into account removed safe inliers preceding neighbors of  $n.obj$ . However,  $\alpha$  can be safely (see next section) estimated as

$$\alpha \approx n.fract\_before \cdot W.$$

Summarizing, the number of preceding neighbors of  $n.obj$  at the query time  $t$  can be estimated as

$$prec\_neighs = n.fract\_before \cdot (W - t + n.id).$$

Recall that, to classify objects the sum between the estimated number of its preceding neighbors and the number of succeeding neighbors is computed. It is worth to point out that the number of succeeding neighbors is not estimated, since  $n.count\_before$  provides the *true* number of succeeding neighbors of  $n.obj$ . Therefore as stream evolves, the sum above approaches the true number of neighbors in the window.

## 4.4 Analysis of the Algorithm

### 4.4.1 Approximation Error Bounds

Now, how to set the parameter  $\rho$  in order to obtain safe bounds on the approximation error of the estimation is discussed.

Let  $W$  be the window size, let  $w$  be the number of safe inliers in ISB, let  $\alpha$  be the exact number of preceding neighbors of an object at its time of arrival, let  $\tilde{\alpha}$  be the number of preceding neighbors of the object in ISB which are safe inliers at its time of arrival, and let  $\mu$  denote the ratio  $\frac{\alpha}{W}$ .

In order to determine an optimal value for  $\rho$ , a value for  $w$ , such that  $\frac{\tilde{\alpha}}{w}$  is a good approximation for  $\mu$ , has to be determined. Formally, the following property has to be satisfied. *For given  $\delta > 0$  and  $0 < \epsilon < 1$ , the following should hold:*

$$\Pr \left[ \left| \frac{\tilde{\alpha}}{w} - \mu \right| \leq \epsilon \right] > 1 - \delta. \quad (4.1)$$

Since ISB contains a random sample of the window safe inliers, a safe bound for  $w$  can be obtained from the well known *Lyapounov Central Limit Theorem*.

This theorem asserts that, for any  $\lambda$

$$\lim_{w \rightarrow \infty} \Pr \left[ \frac{\tilde{\alpha} - w\mu}{\sqrt{w\mu(1-\mu)}} \leq \lambda \right] = \Phi(\lambda)$$

where  $\Phi(\lambda)$  denotes the cumulative distribution function of the normal distribution.

Thus, if  $w$  is large enough, then the following relation holds:

$$\Pr \left[ \frac{\tilde{\alpha} - w\mu}{\sqrt{w\mu(1-\mu)}} \leq \lambda \right] \approx \Phi(\lambda). \quad (4.2)$$

Now, the result that will allow us to get the needful safe bound for  $w$  can be formally presented.



**Theorem 1.** For any  $\delta > 0$  and  $0 < \epsilon < 1$ , if  $w$  satisfies the following inequality

$$w > \frac{\mu(1-\mu)}{\epsilon^2} \left( \Phi^{-1} \left( 1 - \frac{\delta}{2} \right) \right)^2 \quad (4.3)$$

then it satisfies (4.1).

*Proof.* Theorem 1 is a direct consequence of the central limit theorem (see [Watanabe, 2000] for details).

Starting from relation (4.2), and setting

$$\lambda = \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}}$$

it is easy to obtain that, for any  $0 < \epsilon < 1$ :

$$\Pr \left[ \frac{\tilde{\alpha}}{w} > \mu + \epsilon \right] \approx 1 - \Phi \left( \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}} \right) \quad (4.4)$$

$$\Pr \left[ \frac{\tilde{\alpha}}{w} < \mu - \epsilon \right] \approx 1 - \Phi \left( \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}} \right) \quad (4.5)$$

The goal (4.1) is equivalent to:

$$\Pr \left[ \left| \frac{\tilde{\alpha}}{w} - \mu \right| > \epsilon \right] < \delta$$

and, then, to:

$$\Pr \left[ \frac{\tilde{\alpha}}{w} > \mu + \epsilon \right] + \Pr \left[ \frac{\tilde{\alpha}}{w} < \mu - \epsilon \right] < \delta$$

Thus, using relations (4.4) and (4.5), it can be rewritten as:

$$\left( 1 - \Phi \left( \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}} \right) \right) + \left( 1 - \Phi \left( \frac{\epsilon\sqrt{w}}{\sqrt{\mu(1-\mu)}} \right) \right) < \delta$$

Finally, from the above inequality, it is easy to get relation (4.3).

Now, the bound stated in the above theorem is examined. Although the provided bound depends on the unknown value  $\alpha$ , it can be safely applied by setting  $\mu$  to  $\frac{1}{2}$ . Therefore, in order to satisfy (4.1), being  $w = \rho W$ , it is sufficient to set  $\rho$  to the value

$$\rho = \frac{1}{4\epsilon^2 W} \left( \Phi^{-1} \left( 1 - \frac{\delta}{2} \right) \right)^2. \quad (4.6)$$

It is worth noting that the bound for  $w$  given by expression (4.3) does not depend on the window size  $W$ . Furthermore, since in expression (4.6) the unknown value  $\mu$  is safely set to  $\frac{1}{2}$ , whenever  $\mu$  is different to  $\frac{1}{2}$ , the property

(4.1) is guaranteed for values of  $\epsilon$  and  $\delta$  better than those used to compute  $w$ . In particular, the two following inequalities hold:

$$Pr \left[ \left| \frac{\tilde{\alpha}}{w} - \mu \right| \leq \epsilon \right] > 1 - \delta^*, \text{ and} \quad (4.7)$$

$$Pr \left[ \left| \frac{\tilde{\alpha}}{w} - \mu \right| \leq \epsilon^* \right] > 1 - \delta. \quad (4.8)$$

In the first inequality,  $\delta^*$  is obtained from the following equation:

$$\frac{1}{4\epsilon^2} \left( \Phi^{-1} \left( 1 - \frac{\delta}{2} \right) \right)^2 = \frac{\mu(1-\mu)}{\epsilon^2} \left( \Phi^{-1} \left( 1 - \frac{\delta^*}{2} \right) \right)^2$$

whereas, in the second one,  $\epsilon^*$  is obtained from

$$\frac{1}{4\epsilon^2} \left( \Phi^{-1} \left( 1 - \frac{\delta}{2} \right) \right)^2 = \frac{\mu(1-\mu)}{\epsilon^{*2}} \left( \Phi^{-1} \left( 1 - \frac{\delta}{2} \right) \right)^2.$$

Note that, if the true value for  $\mu$  is  $\frac{1}{2}$ , then  $\delta^* = \delta$  and  $\epsilon^* = \epsilon$ .

Equation (4.8) can be rewritten as

$$Pr \left[ \frac{\tilde{\alpha}}{w} - \epsilon^* \leq \frac{\alpha}{W} \leq \frac{\tilde{\alpha}}{w} + \epsilon^* \right] > 1 - \delta.$$

It follows that the maximum error *err* made in computing  $\alpha$  is

$$err = W\epsilon^* = W \cdot 2\epsilon\sqrt{\mu(1-\mu)}.$$

This value provides the maximum error made in estimating the total number of neighbors of an object when it arrives.

Now, it must be determined when the above error will cause a misclassification, i.e. when an inlier (resp., an outlier) will be estimated to be an outlier (resp., an inlier).

For an object *obj* having identifier *id*, when it arrives, the number of true preceding neighbors is  $\mu W$ . As stream evolves, some preceding neighbors expire, and, assuming they are uniformly distributed along the window, in the portion of the stream preceding *obj* in the current window their number becomes  $\mu(W - t + id)$ .

To correctly classify *obj*, if the sum between the number  $\alpha$  of preceding neighbors of *obj* and the number  $\beta$  of succeeding neighbors is greater (resp. smaller) than  $k$  in the current window, then also the estimated value for  $\alpha$  plus the number  $\beta$  of succeeding neighbors should be greater (resp. smaller) than  $k$ . Formally, let  $\bar{W} = W - t + id$  be the number of objects of the current window preceding the object *obj* with identifier *id*, let  $\alpha = \mu\bar{W}$  be the true value of preceding neighbors of *obj* in the current window, let  $\beta$  be the number of its succeeding neighbors, and let  $2\bar{W}\epsilon\sqrt{\mu(1-\mu)}$  be the error *err*. If  $\alpha + \beta$  is greater than  $k$ , then the following inequality should hold:

$$\mu\overline{W} - 2\overline{W}\epsilon\sqrt{\mu(1-\mu)} + \beta > k$$

Assuming that the distribution of succeeding neighbors is the same as the distribution of preceding neighbors,  $\beta$  can be approximated to  $\mu(W - \overline{W})$ , where  $(W - \overline{W})$  is the portion of the stream in the current window succeeding *obj*. Thus, for

$$\mu > \frac{kW + 2\overline{W}^2\epsilon^2 + \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{up} \quad (4.9)$$

an inlier is recognized with probability  $(1 - \delta)$ .

Analogously, if  $\alpha + \beta < k$ , starting from

$$\mu\overline{W} + 2\overline{W}\epsilon\sqrt{\mu(1-\mu)} + \beta < k$$

with the same assumption stated above, it holds that for

$$\mu < \frac{kW + 2\overline{W}^2\epsilon^2 - \sqrt{(kW + 2\epsilon^2\overline{W}^2)^2 - k^2(W^2 + 4\overline{W}^2\epsilon^2)}}{W^2 + 4\overline{W}^2\epsilon^2} = \mu_{down} \quad (4.10)$$

an outlier is recognized with probability  $(1 - \delta)$ .

It can be concluded that, if an object *obj* has more than  $\mu_{up}\overline{W}$  or less than  $\mu_{down}\overline{W}$  neighbors, it is correctly classified with probability  $1 - \delta$ .

Contrariwise, if the number of neighbors of *obj* is in the range  $[\mu_{down}\overline{W}, \mu_{up}\overline{W}]$  then the estimation error at most is equal to  $2\overline{W}\epsilon\sqrt{\mu(1-\mu)}$ , that could lead to a misclassification. Both the error and the range are small and directly proportional to  $\epsilon$ . Moreover, they depend on the current time  $t$ . As  $t$  increases, the error goes to zero and the range tends to be empty.

Before concluding, it is worth to recall that object classification depends also on the number of succeeding neighbors of the object, whose true value is known.

Next, the analysis of the expected distribution of misclassified objects is conducted. In particular, it is analyzed the number of inliers that are incorrectly recognize as outliers. The inverse case, namely the number of outliers that are incorrectly recognized as inliers, is not detailed, since it can be trivially get by the following analysis.

Suppose that the real number of inliers  $\mu W$  in the current window is greater than  $k$ . Then, also the estimated number of inliers should be greater than  $k$ . Therefore, the error  $err = \frac{\alpha}{w}\overline{W} - \mu\overline{W}$  should be such that  $\mu W - err \geq k$ .

Equation (4.2) can be rewritten as:

$$Pr \left[ \frac{\sqrt{w}}{\overline{W}} \frac{\overline{W}(\tilde{\alpha} - w\mu)}{w\sqrt{\mu(1-\mu)}} \leq \lambda \right] \approx \Phi(\lambda)$$

and, then, as:

$$Pr \left[ \frac{err \cdot \sqrt{w}}{\bar{W} \sqrt{\mu(1-\mu)}} \leq \lambda \right] \approx \Phi(\lambda)$$

Finally:

$$Pr \left[ err \leq \lambda \frac{\bar{W} \sqrt{\mu(1-\mu)}}{\sqrt{w}} \right] \approx \Phi(\lambda)$$

By setting

$$\gamma = \lambda \frac{\bar{W} \sqrt{\mu(1-\mu)}}{\sqrt{w}}$$

$$Pr [err \leq \gamma] \approx \Phi \left( \frac{\gamma \sqrt{w}}{\bar{W} \sqrt{\mu(1-\mu)}} \right)$$

follows.

Recall that the aim is to estimate the probability that  $err \leq \mu W - k$ , hence, setting  $\gamma$  to  $\mu W - k$ , it holds

$$Pr [err \leq \mu W - k] \approx \Phi \left( \frac{(\mu W - k) \sqrt{w}}{\bar{W} \sqrt{\mu(1-\mu)}} \right)$$

From above equation, the probability of misclassifying objects can be obtained as:

$$Pr [err > \mu W - k] \approx 1 - \Phi \left( \frac{(\mu W - k) \sqrt{w}}{\bar{W} \sqrt{\mu(1-\mu)}} \right) \quad (4.11)$$

The misclassification distribution depends on the object age, in particular the probability of misclassifying an object  $o$  decreases with the age of  $o$ . Then, the Equation (4.11) represents the probability of a misclassification, given the object age.

Hence, the probability of misclassifying objects can be estimated by exploiting the *law of total probability*:

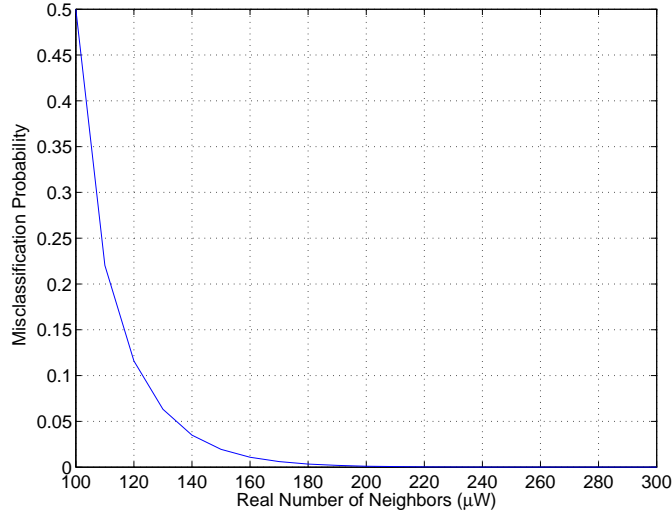
$$Pr [misclass] = \sum_{\eta} Pr [missclass \mid age = \eta] Pr [age = \eta]$$

The term  $Pr [missclass \mid age = \eta]$  is computable by means of Equation (4.11), whereas  $Pr [age = \eta]$  can be assumed equal to  $1/W$ .

Finally:

$$Pr [misclass] = \frac{1}{W} \sum_{\eta=1}^W \left[ 1 - \Phi \left( \frac{(\mu W - k) \sqrt{w}}{\eta \sqrt{\mu(1-\mu)}} \right) \right] \quad (4.12)$$

*Example 3.* For example, consider a window size equal to 10000,  $\delta = 0.1$ ,  $\epsilon = 0.016$  and  $k = 100$ . From Equation 4.3, it follows that  $w = 1088$ . The distribution of misclassified objects is reported in the following figure.



The previous example shows that if the distribution of misclassified objects is very tight to the value of the parameter  $k$ , then only objects with a number of neighbors very close to  $k$  are expected to be misclassified. This behavior will be confirmed in the experimental section (Section 4.5).

#### 4.4.2 Implementation and Cost Analysis

In this section the implementation of ISB is detailed, and then, temporal and spatial costs of STORM are analyzed.

**Implementation details.** The ISB data structure is a *pivot-based* index. This data structure has been already described in Section 3.3.2.

Recall that it performs proximity search in any metric space making and that its performances are related to the number of employed pivots. In particular, the larger is the number of pivots, the more accurate is the list of candidates and, then, the lower is the number of computed distances. Nevertheless, the cost of querying and building the index increases. In this work the number of pivots used is logarithmic with respect to the index size. In order to face concept drift, the older pivot is periodically replaced with an incoming object.

Now, the temporal cost of operations to be executed on the ISB data structure is analyzed. Recall that the cost of computing the distance between two objects is  $\Delta$ . Assume that the number of nodes stored in ISB is  $N$ .

The cost of performing a range query search corresponds to the cost of computing the distances between an object and all the pivots plus the cost of determining true neighbors. Since the number of used pivots is logarithmic in the size of the index, the former cost is  $\mathcal{O}(\Delta \log N)$ . As for the latter cost, let  $\eta$  ( $0 \leq \eta \leq 1$ ) be the mean fraction of index objects marked as candidate

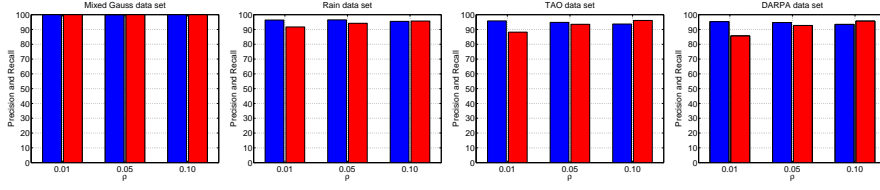


Fig. 4.3. Precision and Recall of approx-STORM.

neighbors when the radius is set to  $R$ . In order to determine if a candidate is a true neighbor, its distance from the query object has to be computed. Then, the cost is  $\mathcal{O}(\Delta\eta N)$ . Supposing that the latter cost is always greater than the former one, the total cost for the range query search is  $\mathcal{O}(\Delta\eta N)$ .

The cost of removing an object from the index is constant, since it practically consists in flagging as empty the entry of an array.

Finally, as for the insertion of an object  $obj$  into ISB, it requires to compute the distances among  $obj$  and all the pivots. However, since insertion is always performed after the range query search, these distances are already available and, then, the insertion cost is constant too.

**Spatial analysis.** For exact-STORM, ISB stores all the  $W$  objects of the current window and, for each of them, the list  $nm\_before$  of the  $k$  most recent preceding neighbors, and two integer numbers. Recall that each object requires space  $d$  and each list requires space  $k$ .

For approx-STORM, assuming meaningful combinations of the parameters, ISB stores approximately  $\rho W$  objects of the current window and, for each of them, a counter of preceding neighbors, and two integer numbers.

Summarizing, the spatial cost of the algorithm STORM is

- $\mathcal{O}(W(d+k))$  for exact-STORM, and
- $\mathcal{O}(\rho W d)$  for approx-STORM.

**Temporal analysis.** The procedure *Stream Manager* consists of four steps (see Figures 4.1 and 4.2). The cost of the step 1 corresponds to the cost of removing an object from ISB, which, from the discussion above, is constant. Also step 2 has a constant cost.

Step 3 performs a range query search, whose cost is  $\mathcal{O}(\Delta\eta N)$ . For each of the  $\eta N$  objects returned by the search, the exact-STORM performs an ordered insertion into the list  $nm\_before$ , that costs  $\mathcal{O}(\log k)$  (see Section 4.3.1), and executes some other operations having constant cost. Contrariwise, the approx-STORM possibly removes a node from ISB and executes some other operations. Both the removal and the other operations have constant cost.

Finally, step 4 inserts a node into ISB and hence has constant cost. Summarizing, the cost of the procedure *Stream Manager* is

- $\mathcal{O}(\Delta\eta W \log k)$  for exact-STORM, and
- $\mathcal{O}(\Delta\eta \rho W)$  for approx-STORM.

The procedure *Query Manager* consists of a scan of the ISB structure. For each node  $n$ , the exact-STORM computes *prec.neighs* in time  $\mathcal{O}(\log k)$  (see Section 4.3.1) by determining the number of identifiers in  $n.nn\_before$  associated with non-expired objects. Conversely, the approx-STORM performs only steps having constant cost. Summarizing, the cost of the procedure *Query Manager* is

- $\mathcal{O}(W \log k)$  for exact-STORM, and
- $\mathcal{O}(\rho W)$  for approx-STORM.

## 4.5 Experimental results

In this section, results obtained by experimenting the proposed techniques on both synthetic and real data sets are presented.

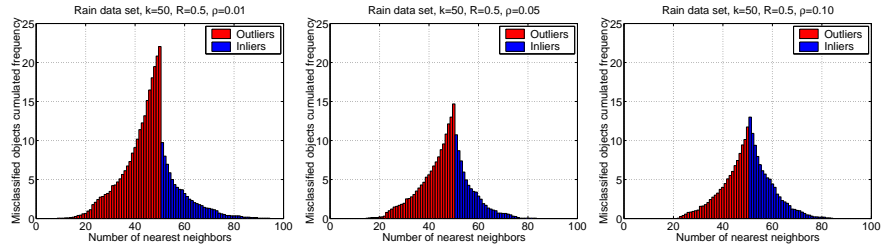
First of all, the data set employed are described. The *Gauss* data set is a synthetically generated time sequence of 35,000 one dimensional observations, also used by Papadimitriou et al. [2003]. It consists of a mixture of three Gaussian distributions with uniform noise.

Also some public real data from the Pacific Marine Environmental Laboratory of the U.S. National Oceanic & Atmospheric Administration (NOAA) are used. Data consist of temporal series collected in the context of the Tropical Atmosphere Ocean project (TAO)<sup>1</sup>. This project collects real-time data from moored ocean buoys for improved detection, understanding and prediction of El Niño and La Niña, which are oscillations of the ocean-atmosphere system in the tropical Pacific having important consequences for weather around the globe. The measurements used in experiments have been gathered each ten minutes, from January 2006 to September 2006, by a moored buoy located in the Tropical Pacific.

Both a one and a three dimensional data stream have been considered. The *Rain* data set consists of 42,961 rain measurements. The *TAO* data set consists of 37,841 triplets (SST, RH, Prec), where SST is the sea surface temperature, measured in units of degrees centigrade at a depth of 1 meter, RH is the relative humidity, measured in units of percent at a height of 3 meters above mean sea level, and Prec is the precipitation, measured in units of millimeters per hour at a height of 3.5 meters above mean sea level. The three attributes were normalized with respect to their standard deviation.

Finally, *1998 DARPA Intrusion Detection Evaluation Data* [DARPA, 1998] has been employed, it has been extensively used to evaluate intrusion detection algorithms. The data set consists of network connection records of several intrusions simulated in a military network environment. The TCP connections have been elaborated to construct a data set of 23 numerical features. 50,000 TCP connection records from about one week of data have been used.

<sup>1</sup> See <http://www.pmel.noaa.gov/tao/>. A gratefully acknowledgement is due to the TAO Project Office for making available the collected measurements.



**Fig. 4.4.** Number of nearest neighbors associated with the misclassified objects of the *Rain* data set.

In all experiments, the window size  $W$  was set to 10,000 and the parameter  $k$  was set to  $0.005 \cdot W = 50$ . The parameter  $R$  was selected to achieve a few percent of outliers in the current window ( $R = 0.1$  for *Gauss*,  $R = 0.5$  for *Rain*,  $R = 1$  for *TAO*, and  $R = 1,000$  for *DARPA*).

Furthermore, it was submitted an outlier query every one hundred objects. Measures reported in the sequel are averaged over the total number of queries submitted. The first query was submitted only after having observed the first  $W$  data stream objects.

The classification accuracy of the method was evaluated. It is worth to recall that exact-STORM exactly detects distance-based outliers in the current window. Thus, the answer returned by this algorithm was used to evaluate the quality of the approximate solution returned by approx-STORM.

The *precision* and *recall* measures were employed. *Precision* represents the fraction of objects reported by the algorithm as outliers that are true outliers. *Recall* represents the fraction of true outliers correctly identified by the algorithm.

Figure 4.3 shows precision (dark bars, on the left) and recall (light bars, on the right) achieved by approx-STORM on the four considered data sets, for increasing values of  $\rho$ , that is  $\rho = 0.01$ ,  $\rho = 0.05$ , and  $\rho = 0.10$ .

Interestingly, on the *Gauss* data set the method practically returned all and only the true outliers. This is because, in this data set, outliers are represented by noise which is well separated from the data distribution. Notice that other methods were not able to obtain this very good classification result.

As for the data sets from the TAO Project, since outliers there contained are associated to large oscillations of earth parameters, they lie on the boundary of the overall measurement distribution and are not completely separated from the rest of the population. Thus, there exists a region of transition where the approximate algorithm can fail to exactly recognize outliers (see below in this section for an evaluation of the characteristics of objects on which classification errors are made).

It is clear by the diagrams that, by augmenting the parameter  $\rho$ , the precision tends to decrease while the recall tends to increase. This can be explained since by using a small sample size the number of nearest neighbors



tends to be overestimated. Anyway, the classification accuracy was very good, e.g. precision 0.965 and recall 0.942 on the *Rain* data set, and precision 0.948 and recall 0.935 on the *TAO* data set, for  $\rho = 0.05$ .

The *DARPA* data set represents a challenging classification task due the considerable number of attributes it is characterized by. The precision-recall trade-off previously observed is confirmed also on this data set. Moreover, the classification accuracy is of remarkable quality: for  $\rho = 0.05$ , precision 0.947 and recall 0.956 were achieved.

Data set	$\rho = 0.01$	$\rho = 0.05$	$\rho = 0.10$	Exact
<i>Gauss</i>	0.17	0.43	0.84	7.52
<i>Rain</i>	0.17	0.47	0.81	7.86
<i>TAO</i>	0.17	0.42	0.62	3.94
<i>DARPA</i>	0.17	0.29	0.47	3.28

**Table 4.1.** Elaboration time per single object [msec].

Table 4.1 reports the time (in milliseconds) employed by approx-STORM for various values of  $\rho$  (first three columns), and by exact-STORM (fourth column) to process an incoming data stream object<sup>2</sup>. Approx-STORM guarantees time savings with respect to exact-STORM which are in most cases proportional to the parameter  $\rho$ . Differences in performances among the various experiments are justified by the different characteristics of the data sets and, among them, particularly, by the mean fraction  $\eta$  of objects falling in the neighborhood of radius  $R$  of data stream objects.

Figure 4.4 shows the distribution of the number of nearest neighbors associated with objects of the *Rain* data set which are misclassified by exact-STORM. These diagrams are interesting to comprehend the nature of the misclassified returned objects and the quality of the approximation. From left to right, diagrams are associated with increasing values of  $\rho$ .

The abscissa reports the number of nearest neighbors, while the ordinate the cumulated absolute frequency of misclassified objects. Two cumulated histograms are included in each diagram, one concerning outliers and the other concerning inliers.

Light bars (on the left) represent the mean number of outliers which are reported as inliers. Thus, these misclassifications concern the recall measure. Specifically, a bar of position  $k_0$  and height  $h_0$  represents the following information: among the objects having at most  $k_0 (< 50)$  nearest neighbors (and hence outliers), on the average,  $h_0$  of them have been recognized as inliers.

<sup>2</sup> Experiments were executed on a Core 2 Duo based machine having 2GB of main memory.

Dark bars (on the right) represent the mean number of inliers which are reported as outliers. Thus, these misclassifications concern the precision measure. Specifically, a bar of position  $k_0$  and height  $h_0$  represents the following information: among the objects having at least  $k_0 (\geq 50)$  nearest neighbors (and hence inliers), on the average,  $h_0$  of them have been recognized as outliers.

These diagrams show that for small sample sizes the number of errors is biased towards the outliers, due to the overestimation effect. Moreover, more interestingly, they show the nature of the misclassified objects. Indeed, as predicted by the analysis of section 4.4.1, for an object the probability of being misclassified greatly decreases with the distance  $|k_0 - k|$  between the true number  $k_0$  of its nearest neighbors and the parameter  $k$ .

Indeed, the majority of the misclassified inliers have a number of neighbors close to  $k$ . For example, when  $\rho = 0.05$ , almost all the misclassified outliers have at most 60 neighbors (compare this value with  $k = 50$ ).

The quality of the approximate answer is thus very high. Although these objects are not outliers according to Definition 6, from the point of view of a surveillance application, they could be as interesting as true outliers, since they anyhow lie in a relatively sparse region of the feature space.

**Outlying Property Detection**



## Detecting Outlying Properties of Exceptional Objects

In this chapter the problem of discovering sets of attributes that account for the abnormality of an individual that is known to be an outlier with respect to a given data population is addressed. A novel way to evaluate the abnormality of subsets of attributes (properties) is introduced and motivated. Next, the problem is analyzed from the complexity point of view, and clever algorithms to solve it are presented. This way, a fully automated support can be provided to decode those properties determining the abnormality of the given object within the reference data context. Finally, some experiments on both real and synthetic data set have been conducted in order to show the algorithm performances and the meaningfulness of the information mined.

### 5.1 Introduction

Let a data population be given, stored into a database  $DB$  represented in the form of a relational table. Let  $o$  be an object of the population, which is known to be abnormal on the basis of available external knowledge. The objective here is that of devising techniques by which it is possible and effective to single out those combinations of attributes that justify the abnormality of the object with the highest plausibility.

Several criteria to measure the abnormality of properties can be defined. In this thesis a suitable one is presented. The proposed measure does not require the definition of a distance relating pairs of objects but, rather, it is based on a simple concept of relative frequency.

In particular, the point of view adopted is that a property, or set of attributes, is abnormal for the object  $o$  if the combination of values  $o$  assumes on these attributes is very infrequent with respect to the overall distribution of the attribute values in the data set: to this end, in the following, it is introduced a measure by which, how much a set of attributes should be considered relevant as to explain the abnormality of the given individual can be faithfully

captured. A discussion on the peculiarities of the measure and its relationship with related measures will be presented later in the chapter (see Section 5.2). From this discussion, and from some of the results of the experiments, it shall clearly turn out that our measure is a sensible and significant one in the context of the analyzed abnormality explanation problems. Both global and local forms of abnormal properties can be defined, where local properties are ones justified by some others playing the role of explanation for the former, and algorithms to mine the top most relevant ones are described.

As an example of knowledge that can be mined by using the approach here described, consider the *Zoo database* from the UCI Machine Learning repository [Newman et al., 1998] reported in Figure 5.1. This database stores information about animals. In particular it contains 15 boolean-valued and two numerical attributes representing animal features. Attributes are described in the caption of the figure. It is known that the platypus is an exceptional animal, since it is a mammal, but it lays eggs. This intuitive notion can be formally illustrated on the database by noticing that among dataset objects having value “y” for the attribute C, that is *eggs*, the platypus is the only animal having value “y” for the attribute D, that is *milk*. Obviously the value “y” for the attribute *milk* is not an exceptional feature “per se”, but it becomes surprising if we restrict our attention to the animals which lay eggs. This is a case where a local property is individuated, where the attribute *eggs* plays the role of *explanation* for the *outlying property*, *milk*, of the platypus.

## 5.2 Related Work

The mining technique illustrated here is of the unsupervised kind.

Methods discussed in Paragraph 2.2 search for objects which can be regarded as anomalous by looking at the full dimensional space. In many real situations an object shows exceptional characteristics only when the attention is restricted to a subset of the features. Searching for outliers in subspaces is a relatively novel research topic. In the rest of the section, works on outlier detection dealing with a search space consisting in a subset of the overall set of data set attributes are briefly surveyed.

In [Sarawagi et al., 1998], a *discovery-driven* technique for the discovery of anomalies into OLAP *data cubes* is presented. The analyst search for anomalies is guided by precomputed indicators of exceptions at various levels of detail in the cube. Loosely speaking, a value  $y$  in the cube is recognized as an exception if it differs significantly from the value  $\hat{y}$  predicted by a statistical model taking into account all the higher-level aggregations the value  $y$  belongs to. The absolute difference  $r = |y - \hat{y}|$  between the actual value  $y$  and the value  $\hat{y}$  anticipated by the model is called the *residual*  $r$ . If the standardized residual  $s = \frac{|y - \hat{y}|}{\sigma}$ , where  $\sigma$  is the anticipated standard deviation associated with residuals, is higher than some threshold  $\tau$ , then the value  $y$  is called an *exception*. The value  $\tau = 2.5$ , corresponding to a probability of 99% in the

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
bass	n	n	y	n	n	y	y	y	n	n	y	0	y	n	n	4	toad	n	n	y	n	n	y	n	y	y	n	n	4	n	n	n	5		
carp	n	n	y	n	n	y	n	y	n	n	y	0	y	n	4	tortoise	n	n	y	n	n	n	n	n	y	y	n	n	4	y	n	y	3		
catfish	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	tuatara	n	n	y	n	n	n	y	y	y	n	n	4	y	n	n	3			
chicken	n	y	y	n	y	n	n	n	y	n	n	2	y	n	2	tuna	n	n	y	n	n	y	y	y	n	n	y	0	y	n	y	4			
chub	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	vulture	n	y	y	n	y	n	y	y	n	n	2	y	n	y	2				
clam	n	n	y	n	n	n	y	n	n	n	n	0	n	n	7	wasp	y	n	y	n	y	n	n	n	y	n	6	n	n	n	6				
crab	n	n	y	n	n	y	n	n	n	n	n	4	n	n	7	worm	n	n	y	n	n	n	n	n	y	n	0	n	n	n	7				
crayfish	n	n	y	n	n	y	n	n	n	n	n	6	n	n	7	wren	n	y	y	n	y	n	n	y	n	n	2	y	n	n	2				
crow	n	y	y	n	y	n	y	n	y	n	n	2	y	n	2	aardvark	y	n	n	y	n	n	y	y	y	n	n	4	n	n	y	1			
dogfish	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	antelope	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1			
dove	n	y	y	n	y	n	n	n	y	n	n	2	y	n	2	bear	y	n	n	y	n	n	y	y	y	n	n	4	n	n	y	1			
duck	n	y	y	n	y	n	n	y	n	n	2	y	n	2	boar	y	n	n	y	n	n	y	y	y	n	n	4	y	n	y	1				
flamingo	n	y	y	n	y	n	n	n	y	n	n	2	y	n	2	buffalo	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1			
flea	n	n	y	n	n	n	n	n	y	n	n	6	n	n	6	calf	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1			
frog	n	n	y	n	n	y	y	y	n	n	4	n	n	5	cavy	y	n	n	n	n	y	y	y	n	n	4	n	y	n	1					
frog	n	n	y	n	n	y	y	y	y	n	4	n	n	5	cheetah	y	n	n	y	n	n	y	y	y	n	n	4	y	n	y	1				
gnat	n	n	y	n	n	n	n	y	n	n	6	n	n	6	deer	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1				
gull	n	y	y	n	y	y	n	y	n	n	2	y	n	2	dolphin	n	n	n	y	n	y	y	y	n	y	0	y	n	y	1					
haddock	n	n	y	n	n	y	n	y	n	n	y	0	y	n	4	elephant	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1			
hawk	n	y	y	n	y	n	y	n	y	n	2	y	n	2	fruitbat	y	n	n	y	n	n	y	y	n	n	2	y	n	n	1					
herring	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	giraffe	y	n	n	n	n	y	y	y	n	n	4	y	n	y	1				
honeybee	y	n	y	n	y	n	n	n	n	y	n	6	n	y	n	6	girl	y	n	n	y	n	y	y	n	n	2	n	y	y	1				
housefly	y	n	y	n	n	n	n	n	y	n	6	n	n	6	goat	y	n	n	n	n	y	y	n	n	4	y	n	y	1						
kiwi	n	y	y	n	n	n	y	n	y	n	2	y	n	2	gorilla	y	n	n	n	n	y	y	n	n	2	n	n	y	1						
ladybird	n	n	y	n	y	n	n	y	n	n	6	n	n	6	hamster	y	n	n	y	n	n	n	y	y	n	n	4	y	n	y	1				
lark	n	y	y	n	n	n	y	n	n	2	y	n	2	hare	y	n	n	n	n	y	y	n	n	4	y	n	n	1							
lobster	n	n	y	n	n	y	n	n	n	n	6	n	n	7	leopard	y	n	n	n	y	y	y	n	n	4	y	n	y	1						
moth	y	n	y	n	n	n	n	n	y	n	6	n	n	6	lion	y	n	n	y	n	y	y	n	n	4	y	n	y	1						
newt	n	n	y	n	n	y	y	y	n	n	4	y	n	5	lynx	y	n	n	n	y	y	y	n	n	4	y	n	y	1						
octopus	n	n	y	n	n	y	n	n	n	n	8	n	n	7	mink	y	n	n	y	n	y	y	y	n	n	4	y	n	y	1					
ostrich	n	y	n	n	n	n	n	y	n	n	2	y	n	2	mole	y	n	n	n	y	y	y	n	n	4	y	n	n	1						
parakeet	n	y	y	n	n	n	n	y	n	n	2	y	n	2	mongoose	y	n	n	n	y	y	y	n	n	4	y	n	y	1						
penguin	n	y	y	n	n	y	n	y	n	n	2	y	n	2	opossum	y	n	n	n	n	y	y	y	n	n	4	y	n	n	1					
pheasant	n	y	y	n	n	n	y	n	n	2	y	n	2	oryx	y	n	n	n	n	y	y	n	n	4	y	n	y	1							
pike	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	polecat	y	n	n	n	y	y	y	n	n	4	y	n	y	1					
piranha	n	n	y	n	n	y	y	y	n	n	y	0	y	n	4	pony	y	n	n	n	n	y	y	n	n	4	y	y	y	1					
pitviper	n	n	y	n	n	n	y	y	y	n	0	y	n	3	porpoise	n	n	n	y	y	y	y	n	y	0	y	n	y	1						
platypus	y	n	y	y	n	y	n	y	n	y	n	4	y	n	y	1	puma	y	n	n	n	y	y	y	n	n	4	y	n	y	1				
rhea	n	y	y	n	n	n	n	y	n	y	n	2	y	n	2	pussycat	y	n	n	n	n	y	y	y	n	n	4	y	y	y	1				
seahorse	n	n	y	n	n	y	n	y	n	n	y	0	y	n	4	raccoon	y	n	n	n	y	y	y	n	n	4	y	n	y	1					
seawasp	n	n	y	n	n	y	n	n	n	y	n	0	n	n	7	reindeer	y	n	n	n	n	y	y	n	n	4	y	y	y	1					
skimmer	n	y	y	n	y	y	n	y	n	n	2	y	n	2	scorpion	n	n	n	n	n	y	n	n	y	n	8	y	n	n	7					
skua	n	y	n	y	y	n	y	n	y	n	2	y	n	2	seal	y	n	n	y	y	y	y	n	y	0	n	n	y	1						
slowworm	n	n	y	n	n	n	y	y	y	n	n	0	y	n	3	sealion	y	n	n	n	y	y	y	n	y	2	y	n	y	1					
slug	n	n	y	n	n	n	n	n	n	n	0	n	n	7	seasnake	n	n	n	n	n	y	y	y	n	n	0	y	n	n	3					
sole	n	n	y	n	n	y	n	y	n	n	y	0	y	n	4	squirrel	y	n	n	n	n	y	y	n	n	2	y	n	n	1					
sparrow	n	y	y	n	y	n	n	n	y	n	n	2	y	n	2	vampire	y	n	n	y	n	n	y	y	n	n	2	y	n	n	1				
starfish	n	n	y	n	n	y	n	n	n	n	5	n	n	7	vole	y	n	n	n	n	y	y	n	n	4	y	n	n	1						
stringray	n	n	y	n	n	y	y	n	n	y	0	y	n	4	wallaby	y	n	n	n	n	y	y	n	n	2	y	n	y	1						
swan	n	y	y	n	y	n	n	y	n	n	2	y	n	2	wolf	y	n	n	n	n	y	y	y	n	n	4	y	n	y	1					
termite	n	n	y	n	n	n	n	n	n	y	n	6	n	n	6																				

**Fig. 5.1.** Zoo database (A=hair, B=feathers, C=eggs, D=milk, E=airborne, F=aquatic, G=predator, H=toothed, I=backbone, J=breathes, K=venomous, L=fins, M=legs (set of values: 0,2,4,5,6,8), N=tail, O=domestic, P=catsize, Q=type (integer values in range [1,7])).

normal distribution, is used. In order to summarize exceptions in lower levels of the cube as single values at higher levels of the cube, some quantities, called SelfExp, InExp, and PathExp, are associated with each cell. SelfExp denotes

the value  $s$  described above,  $\mathcal{I}nExp$  represents the degree of surprise somewhere beneath this cell if the cell is drilled down, and  $\mathcal{P}athExp$  represents the degree of surprise for each drill-down path from the cell. These precomputed quantities are used to lead the analyst to interesting regions of the cube during navigation. This technique is specifically designed for OLAP data cubes [Chaudhuri and Dayal, 1997] and works only on numerical attributes. Furthermore, it is able to discover single exceptional values at cell level or at an aggregate level, and then it cannot be focused on a single tuple to search for anomalous values associated with it.

[Knorr and Ng, 1999] focus on the identification of the intensional knowledge associated with distance-based outliers. First, they detect the distance-based outliers  $P$  in the full attribute space and, then, for each outlier  $P$ , they search for the subspaces that better explain why it is exceptional. In particular, this subspace coincides with the minimal subspace in which  $P$  is still an outlier. Note that the distance-based outlier measure is monotonic with respect to the subset inclusion relationship: if an object is an outlier in a subspace, then it will be an outlier in all its supersets. Although meaningful when dealing with homogeneous numerical attributes, this kind of monotonicity may not be in general suitable for categorical attributes, where often objects exhibit outlierness only in characterizing subspaces. Also, according to this approach, the exceptional object is not provided in input, but it belongs to the set of distance-based outliers of the data set in the full attribute space. Furthermore, this setting models outliers which are exceptional with respect to the whole population, but does not capture objects which are exceptional only with respect to homogeneous subpopulations, e.g. those identified by objects with similar characteristics.

In [Aggarwal and Yu, 2001], anomalies are detected by searching for subspaces in which the data density is exceptionally lower than the mean density of the whole data set. An abnormal lower projection is one in which the density of the data is exceptionally lower than the average. Assume that each attribute of the data is divided into  $\phi$  equi-depth ranges, and let  $f = 1/\phi$ . Let us assume that there are a total of  $N$  points in the data set, and let  $n(\mathcal{D})$  be the number of points in the  $k$ -dimensional cube  $\mathcal{D}$ . Then the sparsity coefficient  $S(\mathcal{D})$  of the cube  $\mathcal{D}$  is defined as follows:

$$S(\mathcal{D}) = \frac{n(\mathcal{D}) - N \cdot f^k}{\sqrt{N \cdot f^k \cdot (1 - f)^k}}, \quad (5.1)$$

where the term  $N \cdot f^k$  denotes the expected number of points in  $\mathcal{D}$ , and the denominator of  $S(\mathcal{D})$  denotes the expected standard deviation. Negative sparsity coefficients indicate cubes in which the presence of the points is significantly lower than expected. In order to search the exponential space of possible projections, a technique based on evolutionary algorithms is presented. The user must specify the number  $m$  of abnormal projections to be determined and, also, the dimensionality  $k$  of the projection which is used in order to determine the outliers. After having determined  $m$  abnormal projections and,



hence,  $m$  sparse cubes, a post-processing phase retrieves the data set objects lying in these cubes and reports them as outliers. This approach requires some non-intuitive parameters to be specified, e.g. the the dimensionality  $k$  of the abnormal projections. It is well suited for numerical attributes, and cannot be focused on a single individual to search for its outlying properties.

In [Zhu et al., 2005], the authors' goal is to find the example-based outliers in a data set. Given an input set of example outliers, i.e. of objects known to be outliers, they search for the objects of the data set which mostly exhibit the same exceptional characteristics as the example outliers. In order to single out these objects, they search for the subspace  $S$  maximizing the average value of sparsity coefficients of Expression (5.1) of cubes containing user examples. This subspace is detected by means of a genetic algorithm. After finding  $S$ , the method returns the objects contained in the cubes which are sparser than or as sparse as cubes containing examples in the subspace  $S$ . As the preceding one, this method is suited for numerical attributes and does not take into account subpopulations. Furthermore, the interest is in detecting outliers showing a behavior similar to a set of known individuals used as examples of exceptionality.

In [Zhang and Wang, 2006], the algorithm *HighDOD* is presented. The interest there is on finding the subspaces in which a given point is an outlier. The outlying degree (OD) of a point is measured using the sum of the distances between this point and its  $k$  nearest neighbors [Angiulli and Pizzuti, 2002]. OD has a monotonic property: if a point is an outlier in a subspace, then it will be an outlier in any superset of that subspace. Two different strategies to identify outlying subspaces are presented. The *Global-T* pruning strategy adopts a unique distance threshold for all the subspaces. Contrariwise, the *Local-T* pruning strategy adopts different distance thresholds for different subspaces in order to take into account the varied denseness of points there. This *Local-T* strategy was introduced since it could mine more interesting subspaces than those found by the *Global-T* one. The *Local-T* strategy is comparatively slow and less scalable to high-dimensional data sets than the *Global-T* strategy, while this latter strategy may be able to prune a larger number of subspaces than the former one. The algorithm *HighDOD* is a dynamic subspace search method that utilizes a sampling-based learning process to identify the subspaces in which a given point is an outlier. Informally, the Total Saving Factor (TSF) of a  $m$ -dimensional subspace is defined as the expected (in terms of probabilities of applying upward and downward pruning; these probabilities are estimated through a sampling-based learning process) savings (in terms of subspaces to be visited) obtained after having visited the subspace and having applied the pruning rules. The basic idea of the dynamic subspace search method is to search beginning with those subspaces with the same dimension that has the highest current TSF value. The search terminates when all the subspaces have been evaluated or pruned. The proposed algorithm is compared with top-down and bottom-up like techniques.

Zhang et al. [2006] present a variant of the above described task. They focus on outlying subspace detection, that is, finding subsets of attributes in which a given data point significantly deviates from the rest of the population. Let  $p$  be a data set object,  $k$  be a positive integer, and  $s$  be a subspace. Then  $D_s^k(p)$  denotes the distance from  $p$  to its  $k$ -th nearest neighbor in the data set projected on the subspace  $s$ . The *Subspace Outlying Factor* (SOF) of a subspace  $s$  w.r.t. a given object  $p$  is defined as the ratio of  $D_s^k(p)$  against average  $D_s^k$  for points in the data set. Outlying subspaces are detected by using a genetic algorithm.

Both the approaches outlined above use the distance-based outlier measure which is not suitable for categorical or non homogeneous attributes. Furthermore, the algorithms used to single out subspaces are non-optimal.

Wei et al. [2003] deal with outlier detection for categorical data. They present a novel definition of outlier based on an hypergraph model. By means of this modeling they aim at capturing the characteristics of data distribution in subspaces. In particular, they focus on detecting outliers by discovering sets of categorical attributes, called *common attributes*, being able to single out a portion of the data set in which the value assumed by some objects on a single additional attribute, called *exceptional attribute*, becomes infrequent with respect to the mean frequency of the values in the domain of that attribute. The degree of infrequency for a value  $v$  in the attribute  $A$  is measured through the *deviation* measure  $\frac{f-\mu}{\sigma}$ , where  $f$  is the absolute frequency of the value  $v$ ,  $\mu$  is the mean absolute frequency associated with values of  $A$ , and  $\sigma$  is their standard deviation. The objects whose value on the exceptional attribute exhibits a meaningful deviation are called outliers. Common attributes are determined by selecting the sets of frequent attributes of the data base. In [Wei et al., 2003] an algorithm is presented, called *HypergraphBasedOutlierTest (HOT)*, to mine hypergraph-based outliers. HOT works by executing the following steps. First, all the frequent itemsets, having a user-specified support *min\_sup*, are mined by using the Apriori algorithm [Agrawal and Srikant, 1994]. Then, these frequent itemsets are arranged in a hierarchy according to the containment relationship. Next, the hierarchy is visited using a bottom-up strategy. Frequent itemsets  $I$  represent common attributes, while each attribute  $A$  not in  $I$  represents a potential exceptional attribute. For each itemset  $I$ , the histogram of frequencies associated with each attribute  $A$  not in  $I$  is stored, and used to compute the deviation of each value taken by  $A$  in the database. Finally, the objects assuming a value on the attribute  $A$  whose deviation is smaller than a user-provided threshold are returned as outliers. Both temporal and spatial costs of this method are  $\mathcal{O}(\|HE\| \cdot k \cdot \max\{\|he\|\})$ , where  $\|HE\|$  is the number of frequent itemsets returned by Apriori,  $k$  is the number of attributes, and  $\max\{\|he\|\}$  is the size of the largest set of tuples selected by a frequent itemset.

The objective of Wei et al. [2003] is to find out a single exceptional attribute. Furthermore, they mine outliers as exceptional objects on an attribute with respect to a frequent itemset which they belong to. Moreover, the first

step of the method consists in the Apriori algorithm. Thus, HOT does not exploit pruning rules to reduce the portion of search space to be explored. Also, its space requirements are impractical, since the number  $\|HE\|$  of frequent itemsets is exponential in the number of attributes.

The *Subgroup Discovery Task* (SDT) [Klösgen, 1996] aims at finding an interesting subgroup of objects with common characteristics with respect to a given attribute, called *target variable*. Typically a subgroup is identified by using a conjunction of selection expressions. For example, for the target variable *coronary heart disease = true*, an interesting subgroup can be that including the objects selected by the expression: *smoke = true AND family history = true*. The usefulness of the subgroup found is typically obtained by means of a statistical test. The SDT outputs a group of exceptional objects and the target variable is only one and it is given as input.

As pointed out earlier in this section, all these methods share the fact of dealing with the search space composed of the subsets of the data set attributes, but major differences exist among them. Before leaving the section, it is thus of interest to summarize the main differences among our approach and approaches referred to above, which is accounted for in the following. In [Sarawagi et al., 1998, Aggarwal and Yu, 2001, Zhu et al., 2005, Wei et al., 2003, Klösgen, 1996] the task considered is different from the task investigated here, since the goal of these works is to detect outliers in subspaces or single out interesting subgroups, though Zhu et al. [2005] make use of examples to guide the detection, whereas in our approach the outlier object is part of the input and the goal is to single out the attributes that better justify its exceptionality. Wei et al. [2003] use the concept of common attributes to single out relevant subpopulations, which is similar to our concept of explanation, but exceptional properties are single attributes, while outlying properties are sets of attributes. Knorr and Ng [1999], Zhang and Wang [2006], Zhang et al. [2006] consider the problem of singling out subspaces in which given points are outliers. All these methods use distance-based definitions and are suitable for data sets having homogenous numerical attributes, but not for categorical attributes, which our technique conversely manages. In [Knorr and Ng, 1999], the exceptional objects are the distance-based outliers of the data set in the full attribute space and, hence, cannot be provided as input. Moreover, Zhang and Wang [2006], Zhang et al. [2006] use local search methods, which are non-optimal. Furthermore, all those three methods do not mine objects which are exceptional with respect to homogeneous subpopulations, e.g. those identified by objects with similar characteristics, like it is done when singling out local properties.

### 5.3 Contribution

The contributions of this thesis, given in the illustrated context, can be summarized as follows:

- the somewhat novel problem of singling out outlying properties with associated explanations for an object given as abnormal within a categorical data set is presented;
- a measure of outlierness for detecting very infrequent combinations of attribute values for an object in a data set is introduced;
- computational complexity figures of the problem of singling out top outlying properties of a given object, with complexity classes ranging from  $F\Delta_2^P[\mathcal{O}(\log n)]$  to  $F\Sigma_2^P$  are precisely depicted;
- accurate upper bounds for the outlierness of supersets of a property are provided;
- linear space algorithms for computing top outlying properties are presented;
- the results of experiments conducted on several domains are presented, they show that, despite the high theoretical complexity associated with handled problems, the algorithms presented here are, on the average, capable to converge and yield the result within a reasonable amount of time; also, experimental evidence demonstrates the effectiveness of our algorithms in mining significant information.

The rest of the chapter is organized according to the following road map.

Paragraph 5.4 defines the measure of abnormality of a property, called *outlierness*, and discusses about the intuition underlying this definition. Major differences with other known measures are commented upon. Moreover, the concept of explanation is introduced, which leads to the definition of local properties.

Paragraph 5.5 formally states the *Outlying Property Detection Problem* (or OPD), which is the problem of knowledge discovery to be solved. In particular, the problem consists in searching for the outlying properties mostly characterizing a given outlier object, that are the properties scoring the greatest values of outlierness.

Paragraph 5.6 studies the computation complexity of the OPD. In order to precisely characterize the complexity of the OPD, several variants of it are defined and studied, namely: OPD(D), its decision version, OPD(O), its optimization version, and OPD(S), its recognition version. The complexity study reveals that the general OPD problem is  $F\Delta_2^P[\mathcal{O}(\log n)]$ -hard, hence intractable, but within  $F\Sigma_2^P$  if the number of properties the user is interested in discovering is polynomially bounded in the number of attributes.

In Paragraph 5.7, some properties of the measure employed here are derived, which are needful to design a smart search algorithm capable of delivering the result by visiting an as-much-restricted-as-possible part of the search space. Basically, given a property and an explanation, the two following questions are answered: (A) how to obtain an upper bound to the outlierness of any superset of the given property justified by the given explanation? (B) how to obtain an upper bound to the outlierness of any superset of the given property justified by any superset of the given explanation?

Equipped with the results presented in Paragraph 5.7, the following Paragraph 5.8 presents algorithms for computing the most unexpected properties associated with a given object. Algorithms are designed with two goals in mind. First, the optimal solution of OPD has to be obtained. Also, to make the method as practical as possible, the algorithms has to run in linear space. In order to prune unfruitful subspaces and speed up computation, algorithms exploit the upper bound properties derived in the previous section.

Paragraph 5.9 describes data structures exploited in order to implement the methods in a way as to comply with the linear space requirements. Both spatial and temporal concrete cost of algorithms are analyzed in detail.

Finally, Paragraph 5.10 shows and discusses experimental results. In particular, the scalability of the method and the sensitivity to parameters variations are studied, and examples of mined knowledge are shown.

## 5.4 Outlying Properties

### 5.4.1 Preliminaries

An *attribute*  $a$  is an identifier with an associated domain. Let  $\mathbf{A} = a_1, \dots, a_m$  be a set of  $m$  attributes<sup>1</sup>. Then, an *object*  $o$  on  $\mathbf{A}$  is a tuple  $o = \langle v_1, \dots, v_m \rangle$  of  $m$  values, such that each  $v_i$  is a value in the domain of  $a_i$ . The *value*  $v_i$  of the attribute  $a_i$  in the object  $o$  will be denoted by  $o[a_i]$ . A *database*  $DB$  on a set of attributes  $\mathbf{A}$  is a multi-set (that is, duplicate elements are allowed), of objects on  $\mathbf{A}$ .

In the following,  $DB$  will denote a database on the set of attributes  $\mathbf{A}$  and  $o$  will be an object of  $DB$ . Moreover, let  $\mathbf{S} = a_1, \dots, a_l$  denote a subset of the attributes in  $\mathbf{A}$ .

The *projection*  $o[\mathbf{S}]$  of the object  $o$  on the subset of attributes  $\mathbf{S}$ , is the new object  $\langle o[a_1], \dots, o[a_l] \rangle$ . The *projection*  $DB[\mathbf{S}]$  of the database  $DB$  on the set of attributes  $\mathbf{S}$  is the new database  $\{o[\mathbf{S}] \mid o \in DB\}$  where it is assumed that possible duplicate objects are maintained. Accordingly, the *selection*  $DB_{o[\mathbf{S}]}$  of the database  $DB$  w.r.t.  $o$  and  $\mathbf{S}$  is the database  $\{o' \in DB \mid o'[\mathbf{S}] = o[\mathbf{S}]\}$ .

In the following, for simplicity of notation and whenever the database is clear from the context, when a function having database  $DB$  among its parameters is denoted, the parameter  $DB$  will be omitted. If it is not the case, then the database of reference will be specified as a superscript of the function.

The *frequency*  $\text{freq}_{\mathbf{S}}(o)$  of  $o$  in the database  $DB$  w.r.t. the subset of attributes  $\mathbf{S} \subseteq \mathbf{A}$ , is the rational number  $|DB_{o[\mathbf{S}]}|/|DB|$ .

Next, the definitions of frequency histogram, cumulated frequency histogram, and marginal frequency histogram are given, which will then be used to define the outlierness measure of an object.

<sup>1</sup> For the sake of simplicity and w.l.o.g., it is assumed that an arbitrary ordering of the attributes in  $\mathbf{A}$  has been fixed.

Car ID	Tyres	Weight	Pit-Stops	Cylinders	Engine Failures
c <sub>1</sub>	B	605	2	10	3
c <sub>2</sub>	B	605	2	8	1
c <sub>3</sub>	B	605	2	10	1
c <sub>4</sub>	B	605	2	10	2
<b>c<sub>5</sub></b>	<b>B</b>	<b>605</b>	<b>3</b>	<b>8</b>	<b>0</b>
c <sub>6</sub>	M	605	2	8	1
c <sub>7</sub>	M	605	2	10	4
c <sub>8</sub>	M	605	2	8	1
c <sub>9</sub>	M	605	3	10	0
c <sub>10</sub>	M	605	3	10	5
c <sub>11</sub>	M	605	3	8	1

Objects	Freq
$\langle B, 3 \rangle$	1/11
$\langle M, 2 \rangle$	3/11
$\langle M, 3 \rangle$	3/11
$\langle B, 2 \rangle$	4/11

(b)  $DB^{ex}$  objects and associated frequencies.

(a) An example database  $DB^{ex}$ .

Fig. 5.2. Example Database

The *frequency histogram*  $\text{hist}_{\mathbf{S}}$  of  $DB$  w.r.t.  $\mathbf{S}$  is the ordered multiset of the rational numbers  $\{f_1, \dots, f_n\}$ , where each  $f_i$  is the frequency  $\text{freq}_{\mathbf{S}}(o_i)$  associated to a distinct object  $o_i \in DB[\mathbf{S}]$ , such that for each  $i \in \{2, 3, \dots, n\}$ , it holds that  $f_{i-1} \leq f_i$ .

*Example 4.* The database  $DB^{ex}$  of Figure 5.2(a) encodes information about Formula 1 car performances. Suppose it is known that the car  $c_5$  is exceptional since it established a striking new total time race record on a certain circuit. Consider the set  $\mathbf{S}$  including the attributes  $\{Tyres, Pit-Stops\}$ . Then the distinct objects of  $DB^{ex}[\mathbf{S}]$  are  $o_1 = \langle B, 2 \rangle$ ,  $o_2 = \langle B, 3 \rangle$ ,  $o_3 = \langle M, 2 \rangle$ , and  $o_4 = \langle M, 3 \rangle$ , and the frequency histogram  $\text{hist}_{\mathbf{S}}^{DB^{ex}}$  is the set  $\{\frac{1}{11}, \frac{3}{11}, \frac{3}{11}, \frac{4}{11}\}$ ; this frequency histogram is reported on the right column of the table in Figure 5.2(b).

Let  $h$  be a frequency histogram. The *cumulated frequency histogram*  $|h|$  of  $h$ , is the set consisting of the pairs  $(0, 0)$  and  $(1, 1)$  plus the pairs  $(f_j, g_j)$ , where  $f_j$  is a distinct frequency of  $h$  and  $g_j$  is  $\sum_{h_i \leq f_j} h_i$ . In the following, the value  $g_j$  will be denoted by  $|h|(f_j)$ .

The *diagram* associated to the cumulated frequency histogram  $\{(f_1, g_1), \dots, (f_n, g_n)\}$  is the curve on the plane drawn by the  $2(n-1) - 1$  segments linking the point  $(f_i, g_i)$  to the point  $(f_{i+1}, g_i)$ , and the point  $(f_{i+1}, g_i)$  to the point  $(f_{i+1}, g_{i+1})$ , respectively.

*Example 4 (continued).* The cumulated frequency histogram  $|\text{hist}_{\mathbf{S}}|$  of  $DB^{ex}$  w.r.t. the set of attributes  $\mathbf{S} = \{Tyres, Pit-Stops\}$  is reported in the second and third columns of the table in Figure 5.3(a), whereas its associated curve is depicted in Figure 5.3(b).  $\square$

Let  $h$  be a frequency histogram. The *marginal frequency histogram*  $\|h\|$  of  $h$ , is the set of pairs  $(f_j, F_j)$ , where  $(f_j, g_j)$  is a pair of the cumulated frequency histogram  $|h|$  and  $F_j$  is  $\sum_{f_k > f_j} (f_k - f_{k-1})g_{k-1}$ . In the following, the value  $F_j$  will be denoted by  $\|h\|(f_j)$ . It immediately follows from this definition that

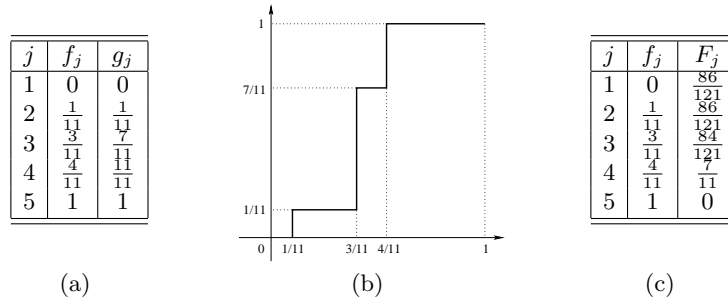


Fig. 5.3. Histograms of the example data base.

the value  $\|h\|(f_j)$  equals the area below the portion of curve of the cumulated frequency histogram included between the frequencies  $f_j$  and 1.

Example 4 (continued). Consider again the database  $DB^{ex}$ . The marginal frequency histogram of  $DB^{ex}$  w.r.t.  $\mathbf{S} = \{ Tyres, Pit-Stops \}$  is reported in the last two columns of the table in Figure 5.3(c).  $\square$

### 5.4.2 Outlierness

Next, the concept of *outlierness*, that is, the abnormality measure employed throughout the chapter, is introduced.

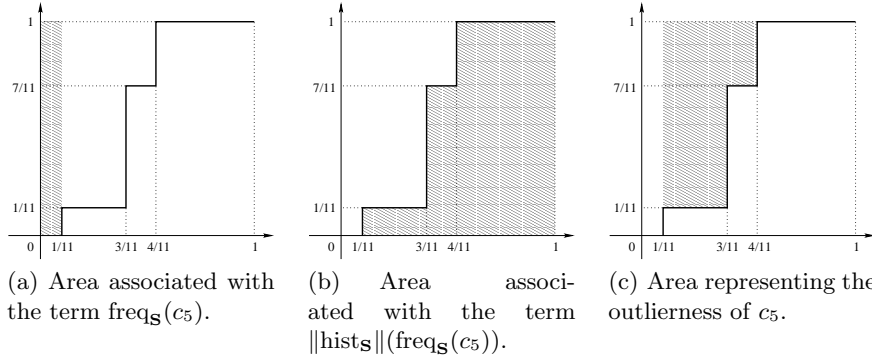
The intuition underlying the definition provided below is that a set of attributes makes an object exceptional in an object w.r.t. a database if the frequency of the combination of values assumed by that object on those attributes is rare if compared to the frequencies associated with the other combinations of values assumed on the same attributes by the other objects of the database.

A way to quantify the “degree of unbalanceness” between the frequency of the object under consideration  $o$  and the frequencies of the rest of the database is to measure the area *above* the cumulated frequency histogram of the database w.r.t. the set of attributes of interest, starting from the frequency of  $o$ . Indeed, the larger this area is, the smaller the frequency of  $o$  is w.r.t. the frequencies associated with the other data set objects.

Thus, the *outlierness*  $out_{\mathbf{S}}(o)$  of the set of attributes  $\mathbf{S}$  in  $o$  w.r.t.  $DB$  is defined as follows

$$out_{\mathbf{S}}(o) = 1 - [ freq_{\mathbf{S}}(o) + \|hist_{\mathbf{S}}\|(freq_{\mathbf{S}}(o)) ].$$

The outlierness takes values in the range  $[0, 1]$  and it is zero when the property can be considered usual, whereas the more the property denoted by the given set of attributes is outlying, the more it will get close to one. Given a threshold



**Fig. 5.4.** The areas associated with the curve of the cumulated frequency histogram.

$\theta \in [0, 1]$ ,  $\mathbf{S}$  is said a  $\theta$ -outlying property, or simply an outlying property, for  $o$  if  $\text{out}_{\mathbf{S}}(o) \geq \theta$ .

*Example 4* (continued). Recall that, in our example,  $\mathbf{S}$  is the set  $\{\text{Tyres}, \text{Pit-Stops}\}$ . Figure 5.4 shows the diagram of the cumulated frequency histogram  $\|\text{hist}_{\mathbf{S}}\|$ . The area associated to  $\text{freq}_{\mathbf{S}}(c_5)$ , is reported on the left, which corresponds to the value  $\frac{1}{11}$ , while the area associated to  $\|\text{hist}_{\mathbf{S}}\|(\text{freq}_{\mathbf{S}}(c_5))$ , is displayed in the center as Figure 5.4(b), whose value is  $\frac{1}{11} \cdot (\frac{3}{11} - \frac{1}{11}) + \frac{7}{11} \cdot (\frac{4}{11} - \frac{3}{11}) + 1 \cdot (1 - \frac{4}{11}) = \frac{86}{121}$ . Finally, the outlierness  $\text{out}_{\mathbf{S}}(c_5)$  is equal to  $1 - (\frac{1}{11} + \frac{86}{121}) \simeq 0.2$ , which corresponds to the area depicted on the right (Figure 5.4(c)). Notice that, by considering any proper subset  $\mathbf{S}'$  of  $\mathbf{S}$ , the value of  $\text{out}_{\mathbf{S}'}(c_5)$  would be zero.  $\square$

It is easily seen from the definition provided above that, for any  $\mathbf{S}$  and  $o$ , the measure of  $\text{out}_{\mathbf{S}}(o)$  satisfies the following intuitive properties:

- If  $\text{hist}_{\mathbf{S}} = \{\frac{c}{n}, \dots, \frac{c}{n}\}$ , then  $\text{out}_{\mathbf{S}}(o) = 1 - [\frac{c}{n} + (1 - \frac{c}{n})] = 0$ , that is, if the frequency histogram corresponds to a uniform distribution, then any subset of attribute values will be associated to a constant frequency and, hence, accordingly, its outlierness is zero;
- If  $\text{freq}_{\mathbf{S}}(o) = \max(\text{hist}_{\mathbf{S}}(o)) = f$ , then  $\text{out}_{\mathbf{S}}(o) = 1 - [f + (1 - f)] = 0$ , that is, if the frequency of the value assumed by the test object  $o$  corresponds to the maximum frequency then, regardless of the distribution of the frequency histogram, the property encoded by the considered attribute set  $\mathbf{S}$  is going to be regarded as rather usual and, hence, consistently, the corresponding outlierness is zero;
- If  $\text{hist}_{\mathbf{S}} = \{\frac{1}{n}, \frac{n-1}{n}\}$  and  $\text{freq}_{\mathbf{S}}(o) = \frac{1}{n}$ , then  $\text{out}_{\mathbf{S}}(o) = 1 - [\frac{1}{n} + \|\text{hist}_{\mathbf{S}}\|(\frac{1}{n})] = 1 - [\frac{1}{n} + \frac{1}{n} \cdot (\frac{n-1}{n} - \frac{1}{n}) + 1 \cdot (1 - \frac{n-1}{n})] = \frac{(n-1)(n-2)}{n^2}$  and, hence,  $\lim_{n \rightarrow \infty} \text{out}_{\mathbf{S}}(o) = 1$ , that is, if all the database objects assume the same values on the set of attributes  $\mathbf{S}$ , while the test object  $o$  assumes a different value on the same attributes, then, as expected, the



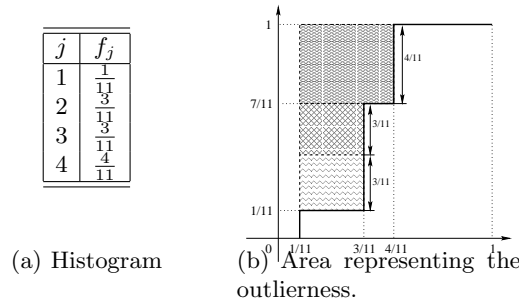
larger the database is, the closer the outlierness associated with  $\mathbf{S}$  for the object  $o$  will get to one.

Conceptually, it is useful and intuitive to define the outlierness in terms of an area obtained from the cumulated frequency histogram; however, for the sake of *computing* the outlierness, it is more convenient to reformulate it, as explained in the following theorem.

**Theorem 2.** *Let  $o$  be an object such that  $\text{freq}_{\mathbf{S}}(o) = f$  and let  $\text{hist}_{\mathbf{S}} = \{f_1, \dots, f_k\}$ . Then  $\text{out}_{\mathbf{S}}(o) = \sum_{f_i > f} f_i(f_i - f)$ .*

*Proof.* By using a graphical argument – see Figure 5.5 for an example – the area above the curve of the cumulated frequency histogram starting from  $f$ , can be obtained by superimposing  $k'$  rectangles, where  $k'$  is the number of frequencies  $f_i$  such that  $f_i > f$ , having height  $f_i$  and breadth  $(f_i - f)$  each. Thus, each single frequency  $f_i > f$  contributes to increase the total area of a quantity  $f_i(f_i - f)$ , and the outlierness  $\text{out}_{\mathbf{S}}(o)$  is thus  $\sum_{f_i > f} f_i(f_i - f)$ .

For example, consider a given frequency histogram w.r.t. a property  $\mathbf{S}$  (Figure 5.5(a)). Let  $\frac{1}{11}$  be the frequency associated to the object  $o$ . It follows from Theorem 2 than the outlierness  $\text{out}_{\mathbf{S}}(o)$  can be obtained by summing up the areas of the three highlighted rectangles.



**Fig. 5.5.** Example of outlierness computation.

**A different measure**

In order to quantify the unexpectedness of the frequency  $f$  of a certain combination of attribute values, a measure based on the cumulated histogram of frequencies is adopted. A supposedly more immediate solution would be instead obtained by combining measures of centrality and dispersion, e.g. like in  $\frac{f-\mu}{\sigma}$  where  $\mu$  is the mean and  $\sigma^2$  is the standard deviation of the frequency histogram (which is similar to that employed in [Wei et al., 2003]).

The rationale underlying this choice is that our measure of outlierness is more suitable for singling out very infrequent combinations of values than the one just recalled, which has instead the undesirable property of associating the same score to very different distributions.

As an example, let  $n \geq 2$  be an integer, and consider the distribution consisting of the  $n$  frequencies  $f_0 < \frac{1}{2n}$ , plus the  $n$  frequencies  $f_1 = \frac{1-nf_0}{n}$ . The mean of this distribution is  $\mu = \frac{1}{2n}$ , while the variance is  $\sigma = \frac{1-2nf_0}{2n}$ . Assume that  $f_0$  is the frequency associated to the object being under examination, that is  $f = f_0$ . As a whole, the term  $\frac{f-\mu}{\sigma}$  is equal to  $-1$ . It is clear thus that, in this case, this measure comes out to be independent of  $f_0$ . Therefore, with a very unbalanced distribution, like, e.g., that obtained for  $n = 10$  and  $f_0 = 0.001$  and  $f_1 = 0.099$ , the same score will be returned as an almost uniform one, like, e.g., that obtained for  $n = 10$  and  $f = 0.049$  and  $f_1 = 0.051$ . Counterwisely, the outlierness would be 0.097 for the former distribution, and 0.001 for the latter.

A similar consideration holds even if the analyzed measure is not normalized, that is, if the distance from the mean  $f - \mu$  is employed as the measure of exceptionality. Indeed, consider the histogram  $h = \{\frac{1}{n}, \frac{c}{n}, \frac{n-c-1}{n}\}$ , and let  $\text{freq}_{\mathbf{S}}(o)$  be  $\frac{1}{n}$ . Since the mean is independent of  $c$ , it would be obtained the same score for very different distributions as, for example, with  $h_1 = \{\frac{1}{n}, \frac{1}{2}, \frac{1}{2} - \frac{1}{n}\}$  and  $h_2 = \{\frac{1}{n}, \frac{1}{n}, \frac{n-2}{n}\}$ , while the outlierness is 0.995 for the former distribution and 0.498 for the latter.

### Functional Dependencies and Outlierness

If *functional dependencies* [Ullman, 1982] are defined on the input database, they can be exploited to prune the search space associated to our problem. In particular, suppose that a functional dependency  $A \rightarrow B$  holds in the input dataset. By definition of functional dependency, if two objects assume the same value on  $A$ , then they will assume the same value on  $B$  and then, also on the pair of attributes  $\{A, B\}$ ; similarly, if two objects assume a different value on  $A$ , they will assume a different value on every superset of  $A$  and then, also on  $\{A, B\}$ . This implies that  $\{A, B\}$  is an outlier property if and only if  $A$  is an outlier property, as well. Then,  $\{A, B\}$  can be safely pruned from the search space while computing outlier properties. Conversely, it is worth noticing that nothing can be deduced about  $B$ , since the outlierness computed for  $B$  can be greater than, smaller than or equal to the outlierness computed for  $A$ . For example consider the following two datasets:

Suppose that  $t_0$  is the exceptional object and that in both datasets the functional dependency  $A \rightarrow B$  holds. It is easy to see that for the dataset in Figure 5.6(a), the outlierness of  $t_0$  w.r.t.  $B$  is greater than the outlierness computed w.r.t.  $A$ . Conversely, for dataset in Figure 5.6(b), the outlierness computed w.r.t.  $B$  is 0 and, then, smaller than the outlierness computed w.r.t.  $A$ .

Obj	A	B
$t_0$	a	a
$t_1$	a	a
$t_2$	b	b
$t_3$	b	b
$t_4$	b	b
$t_5$	c	b
$t_6$	c	b
$t_7$	c	b

(a)

Obj	A	B
$t_0$	a	a
$t_1$	a	a
$t_2$	b	a
$t_3$	b	a
$t_4$	b	a
$t_5$	c	b
$t_6$	c	b
$t_7$	c	b

(b)

**Fig. 5.6.** Example of Dataset with Functional Dependencies

Obviously, if in a dataset both the functional dependencies  $A \rightarrow B$  and  $B \rightarrow A$  hold,  $B$  can be safely discarded from the search space.

### 5.4.3 Outlier Explanations

Sometimes, some properties  $\mathbf{S}$  of an object  $o$  that behave normally w.r.t. the database as a whole, may be unexpected when attention is solely restricted to a portion of the database. Relevant subsets of the database to hand are those obtainable by selecting database objects agreeing with  $o$  on a subset of  $\mathbf{E}$  attributes disjoint from  $\mathbf{S}$ . In such a case, the subset  $\mathbf{E}$  plays the role of *explanation* of the unexpected property  $\mathbf{S}$ . In the following, the formal framework is generalized to outlying property detection under explanations.

The *outlierness* of the set  $\mathbf{S}$  of attributes in  $o$  w.r.t.  $DB$  with *explanation*  $\mathbf{E}$  is defined as

$$\text{out}_{\mathbf{E},\mathbf{S}}(o) = \text{out}_{\mathbf{S}}^{DB_{o[\mathbf{E}]}}(o)$$

Notice that this definition represents a generalization of the analogous definition reported in Section 5.4.2, since the outlierness of  $\mathbf{S}$  in  $o$ , when no explanation is considered, coincide with the outlierness of  $\mathbf{S}$  in  $o$  with explanation  $\mathbf{E} = \emptyset$ .

*Example 5.* Consider the database  $DB^{ex}$  of Figure 5.2, let  $\mathbf{S}$  be  $\{\text{Engine\_Failures}\}$ , and let  $o$  be the object  $c_5$ . Then  $\text{out}_{\mathbf{S}}(c_5) = 1 - [\frac{2}{11} + (\frac{5}{11} - \frac{2}{11}) \cdot \frac{6}{11} + (1 - \frac{5}{11}) \cdot 1] \simeq 0.12$ . However, if the attention is restricted to the portion of the database agreeing with  $c_5$  on the explanation  $\mathbf{E} = \{\text{Cylinders}\}$ , i.e. to the objects  $o$  such that  $o[\mathbf{E}] = \text{"8"}$ , then the outlierness measure computed as  $\text{out}_{\mathbf{E},\mathbf{S}}(c_5)$  for  $c_5$  becomes considerably larger, since  $\text{hist}_{\mathbf{S}}^{DB_{o[\mathbf{E}]}} = \{\frac{1}{5}, \frac{4}{5}\}$  and  $\text{out}_{\mathbf{E},\mathbf{S}}(c_5)$  is, therefore,  $1 - [\frac{1}{5} + (\frac{4}{5} - \frac{1}{5}) \cdot \frac{1}{5} + (1 - \frac{4}{5}) \cdot 1] = 0.48$ .

However, it is apparent that not all the explanations should be considered equally relevant. As a limit case, consider a situation where  $DB_{o[\mathbf{E}]}$  includes one single object. And, in fact, the larger the portion of the database selected by the explanation is the more interesting it should be considered. Thus,

an explanation  $\mathbf{E}$  is said a  $\sigma$ -explanation, where  $\sigma \in [0, 1]$  is a user-defined parameter, if  $\mathbf{E}$  is such that  $\text{freq}_{\mathbf{E}}(o) \geq \sigma$ .

## 5.5 The Outlying Property Detection Problem

In this section, the knowledge discovery task of interest here is defined.

Assume the following be given:

- an object  $o$  of a database  $DB$  on a set of attributes  $\mathbf{A}$ ,
- parameters  $\sigma \in [0, 1]$  and  $\theta \in [0, 1]$ ,
- two disjoint subsets  $\mathbf{S}$  and  $\mathbf{E}$  of  $\mathbf{A}$ .

Then the pair  $(\mathbf{E}, \mathbf{S})$  is an *explanation-outlying property*  $(\sigma, \theta)$ -pair, or simply an *explanation-property pair*, if  $\mathbf{S}$  is a  $\theta$ -outlying property for  $o$  with  $\sigma$ -explanation  $\mathbf{E}$ .

**Definition 8 (Outlying Property Detection Problem).** *Given a data set  $DB$  on the set of attributes  $\mathbf{A}$ , an object  $o$ , thresholds  $\sigma, \theta \in [0, 1]$ , and a positive integer number  $k$ , the problem Outlying Property Detection  $\langle \mathbf{A}, DB, o, \sigma, \theta, k \rangle$  (OPD for short) is defined as follows: among all the pairs  $(\mathbf{E}, \mathbf{S})$  of disjoint subsets of the set of attributes  $\mathbf{A}$  of  $DB$ , find the  $k$  explanation-outlier  $(\sigma, \theta)$ -pairs  $\{(\mathbf{E}_1, \mathbf{S}_1), \dots, (\mathbf{E}_k, \mathbf{S}_k)\}$ , also called top- $k$  pairs, scoring the greatest values of outlierness  $\text{out}_{\mathbf{E}, \mathbf{S}}(o)$ .*

Note that, depending on the particular instance of the problem, the number of  $(\sigma, \theta)$ -pairs might be less than  $k$ . Thus, in general, the solution set of the problem includes *at most*  $k$  pairs.

Besides by setting  $\mathbf{E} = \emptyset$ , explanations can be dropped off the scenario by simply setting  $\sigma = 1$ , since when the frequency threshold  $\sigma$  is set to one, clearly, explanations play no role in the problem at hand. To be more precise, since attributes assuming the same value on all the database objects might exist, an explanation  $\mathbf{E} \neq \emptyset$  can anyway exist even if  $\sigma$  is set to 1; however  $\mathbf{E}$  would represent a *trivial explanation* and, then, they are not considered any further. In the case where explanations are not taken into account, the top  $k$  outlying properties w.r.t. the overall database, in the following also referred to as searching for *global (outlying) properties*, are looked for. Vice versa, if  $\sigma$  is less than one, this means that explanations are actually of interest, and outlying properties will be referred to as *local (outlying) properties*.

## 5.6 Complexity of The Outlying Property Detection Problem

In this section, definitions of computational complexity classes and tools that will be used in the sequel are recalled first (Section 5.6.1) and, then, the computational complexity of the OPD problem is investigated (Section 5.6.2).

### 5.6.1 Preliminaries on Computational Complexity

Some basic definitions about complexity theory are recalled next. The reader is referred to [Garey and Johnson, 1979, Papadimitriou, 1994] for more on this.

*Decision* problems are maps from strings (encoding the input instance over a suitable alphabet) to the set {"yes", "no"}. A (possibly nondeterministic) Turing machine  $M$  answers a decision problem, if on a given input  $x$ , (i) a branch of  $M$  halts in an accepting state iff  $x$  is a "yes" instance, and (ii) all the branches of  $M$  halt in some rejecting state iff  $x$  is a "no" instance.

The class  $P$  is the set of decision problems that can be answered by a deterministic Turing machine in polynomial time. The classes  $\Sigma_k^P$  and  $\Pi_k^P$ , forming the *polynomial hierarchy*, are defined as follows:  $\Sigma_0^P = \Pi_0^P = P$  and, for all  $k \geq 1$ ,  $\Sigma_k^P = NP^{\Sigma_{k-1}^P}$ ,  $\Delta_k^P = P^{\Sigma_{k-1}^P}$ , and  $\Pi_k^P = co\text{-}\Sigma_k^P$  where

- $co\text{-}\Sigma_k^P$  denotes the class of problems whose complementary problems are solvable in  $\Sigma_k^P$ ,
- $\Sigma_k^P$  (resp.  $\Delta_k^P$ ) models computability by a nondeterministic (resp. deterministic) polynomial-time Turing machine which may use an oracle that is, loosely speaking, a subprogram that can be run with no computational cost, for solving a problem in  $\Sigma_{k-1}^P$ .

The class  $\Sigma_1^P$  of decision problems that can be solved by a nondeterministic Turing machine in polynomial time is also denoted by  $NP$ , while the class  $\Pi_1^P$  of decision problems whose complementary problem is in  $NP$ , is denoted by  $co\text{-}NP$ .

*Functions* (also *computation problems*) are (partial) maps from strings to strings, which can be computed by suitable Turing machines, called *transducers*, that have an output tape. In particular, a transducer  $T$  computes a string  $y$  on input  $x$ , if some branch of the computation of  $T$  on  $x$  halts in an accepting state and, in that state,  $y$  is on the output tape of  $T$ . Thus, a function  $f$  is computed by  $T$ , if (i)  $T$  computes  $y$  on input  $x$  iff  $f(x) = y$ , and (ii) all the branches of  $T$  halt in some rejecting state iff  $f(x)$  is undefined.

In this chapter, some classes of computation problems will be referred to which are illustrated next (see, also, [Krentel, 1988, Selman, 1994]). The class  $FP$  is the set of all the polynomial time computable functions, that are functions computed by polynomial-time bounded deterministic transducers. More generally, for each class of decision problems, say  $\mathcal{C}$ ,  $FC$  denotes its functional version; for instance,  $FNP$  denotes the class of functions computed by nondeterministic transducers in polynomial time,  $F\Sigma_2^P$  denotes the class of functions computed in polynomial time by nondeterministic transducers that use an  $NP$  oracle, and  $F\Delta_2^P$  denotes the functions computed, in polynomial time, by a deterministic transducer which uses an  $NP$  oracle.  $F\Delta_2^P[\mathcal{O}(\log n)]$  is the subset of  $F\Delta_2^P$  denoting the class of functions computed in polynomial time by deterministic transducers which, on input  $x$ , query a total of  $\mathcal{O}(\log |x|)$  times the  $NP$  oracle.

Finally, the notion of reduction for decision and computation problems is needed to be recalled. A decision problem  $A_1$  is *polynomially reducible* to a decision problem  $A_2$  if there is a polynomial time computable function  $h$  such that for every  $x$ ,  $h(x)$  is defined and  $A_1$  output “yes” on input  $x$  iff  $A_2$  outputs “yes” on input  $h(x)$ . A decision problem  $A$  is *complete* for the class  $\mathcal{C}$  of the polynomial hierarchy iff  $A$  belongs to  $\mathcal{C}$  and every problem in  $\mathcal{C}$  is polynomially reducible to  $A$ . Moreover, a function  $f_1$  is *reducible* to a function  $f_2$  if there is a pair of polynomial-time computable functions  $h_1, h_2$  such that, for every  $x$ ,  $h_1(x)$  is defined, and  $f_1(x) = h_2(x, w)$  where  $w = f_2(h_1(x))$ . A function  $f$  is hard for a class of functions  $\mathcal{FC}$ , if every  $f' \in \mathcal{F}$  is polynomially reducible to  $f$ , and is complete for  $\mathcal{FC}$ , if it is hard for  $\mathcal{FC}$  and belongs to  $\mathcal{FC}$ .

### 5.6.2 Complexity analysis

In this section, the OPD problem is shown to be intractable. To characterize the complexity of this problem, next three variants of the basic problem are introduced. First, the decision version OPD(D) of the OPD problem is defined.

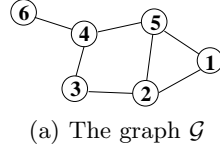
**Definition 9 (Outlying Property Detection Decision Problem).** *Given a set of attributes  $\mathbf{A}$ , a database  $DB$  on  $\mathbf{A}$ , an object  $o$  of  $DB$ , a threshold  $\sigma \in [0, 1]$ , and a threshold  $\theta \in [0, 1]$ , the problem Outlying Property Decision  $\langle \mathbf{A}, DB, o, \sigma, \theta \rangle_{(D)}$  (OPD(D), for short) is: does there exist a pair  $(\mathbf{E}, \mathbf{S})$  of disjoint subsets of  $\mathbf{A}$  that is a  $(\sigma, \theta)$  explanation-property pair?*

The OPD(O) problem, defined next, is the restricted variant of the OPD problem in which the maximum value of outlierness associated with any explanation-property pair for an input object is looked for, and it this formally defined in the following.

**Definition 10 (Maximum Outlierness Value Problem).** *Given a set of attributes  $\mathbf{A}$ , a database  $DB$  on  $\mathbf{A}$ , an object  $o$  of  $DB$ , and a threshold  $\sigma \in [0, 1]$ , the problem Maximum Outlierness Value  $\langle \mathbf{A}, DB, o, \sigma \rangle_{(O)}$  (OPD(O), for short) is: compute the maximum value  $\theta^*$  of outlierness  $\text{out}_{\mathbf{E}, \mathbf{S}}(o)$  associated with any pair  $(\mathbf{E}, \mathbf{S})$  of disjoint subsets of  $\mathbf{A}$  that is a  $(\sigma, \theta)$  explanation-property pair for  $o$ .*

Now, assume that someone provides us the potential solution of the general OPD problem. A further interesting problem thereof is to verify if this provided solution is indeed a correct solution for the problem at hand. Such a checking problem is defined next.

**Definition 11 (Outlying Property Solution).** *Given a data set  $DB$ , an object  $o$  of  $DB$ , an integer  $k$ , thresholds  $\sigma, \theta \in [0, 1]$ , and a set  $Sol = \{(\mathbf{E}_1, \mathbf{S}_1), (\mathbf{E}_2, \mathbf{S}_2), \dots, (\mathbf{E}_h, \mathbf{S}_h)\}$  of at most  $k$  ( $0 \leq h \leq k$ ) explanation-property  $(\sigma, \theta)$ -pairs for  $o$  in  $DB$ , the Outlying Property Solution problem  $\langle \mathbf{A}, DB, o, \sigma, \theta, k, Sol \rangle_{(S)}$ , OPD(S) for short, is defined as follows: is  $Sol$  the solution of the given OPD problem?*



	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$
$t_0$	0	0	0	0	0	0
$t_1$	1	1	0	0	1	0
$t_2$	2	2	2	0	2	0
$t_3$	0	3	3	3	0	0
$t_4$	0	0	4	4	4	4
$t_5$	5	5	0	5	5	0
$t_6$	0	0	0	6	0	6
$t_7$	1	1	0	0	1	0
$t_8$	2	2	2	0	2	0
$t_9$	0	3	3	3	0	0
$t_{10}$	0	0	4	4	4	4
$t_{11}$	5	5	0	5	5	0
$t_{12}$	0	0	0	6	0	6
$t_{13}$	-1	-2	-2	-2	-2	-2
$t_{14}$	-2	-1	-2	-2	-2	-2
$t_{15}$	-2	-2	-1	-2	-2	-2
$t_{16}$	-2	-2	-2	-1	-2	-2
$t_{17}$	-2	-2	-2	-2	-1	-2
$t_{18}$	-2	-2	-2	-2	-2	-1
$t_{19}$	-2	-2	-2	-2	-2	-2
$t_{20}$	-2	-2	-2	-2	-2	-2
$t_{21}$	-2	-2	-2	-2	-2	-2

(b) The database  $DB^{\mathcal{G}}$

Fig. 5.7. An example of the reduction used in Theorem 3

Next, the complexity of the afore-defined problems will be discussed. First, the problem OPD(D) (of Definition 9) for the special case when  $\mathbf{E} = \emptyset$  (or equivalently,  $\sigma = 1$ ) is dealt with.

**Theorem 3.** For  $\sigma = 1$  the OPD(D) problem is NP-complete.

*Proof. (Membership)* A succinct certificate for  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  is given by a property  $\mathbf{S}$ , where  $\mathbf{S}$  is a subset of  $\mathbf{A}$ . Since  $\text{outs}_{\mathbf{S}}(o)$  is computable in polynomial time (see Theorem 2), it can be checked in polynomial time that  $\mathbf{S}$  is a  $\theta$ -outlying property for  $o$  in  $DB$ .

*(Hardness)* The proof is by reduction of the DOMINATING SET PROBLEM, which is known to be NP-complete [Garey and Johnson, 1979]. Let  $\mathcal{G} = \langle V, E \rangle$  be an undirect graph, where  $V = \{v_1, \dots, v_n\}$  is a set of nodes, and  $E = \{e_1, \dots, e_m\}$  is a set of edges  $e_i = \{v_{p_i}, v_{q_i}\}$  s.t.  $p_i, q_i \in \{1, \dots, n\}$ , for  $i = 1, \dots, m$ . Let  $k$  be a given integer. A *dominating set* is a subset  $W$  of  $V$  such that every node of  $V$  not in  $W$  is joined to at least one member of  $W$  by some edge in  $E$ . The DOMINATING SET PROBLEM is: “Does there exist in  $\mathcal{G}$  a dominating set of size at most  $k$ ?”.

The reduction used is as follows.

Given an undirected graph  $\mathcal{G} = \langle V, E \rangle$ , consider the problem  $\langle \mathbf{A}^{\mathcal{G}}, DB^{\mathcal{G}}, o^{\mathcal{G}}, 1, \theta^{\mathcal{G},k} \rangle_{(D)}$ , where (an example of this reduction is shown in Figure 5.7):

- $\mathbf{A}^{\mathcal{G}}$  is the set of attributes  $\{A_1, \dots, A_n\}$ , where  $A_j$  represents the node  $v_j$  of  $\mathcal{G}$ , for each  $j = 1, \dots, n$ ;

- $DB^{\mathcal{G}}$  is a database defined on  $\mathbf{A}$  and composed by  $3n + 4$  objects, in particular:
  - the object  $t_0$ , a tuple of  $n$  zeros;
  - the objects  $t_i$  and  $t_{i+n}$ ,  $i \in \{1, \dots, n\}$ , such that: for each  $j = 1, \dots, n$ , if there exists an edge linking  $v_i$  with  $v_j$  or  $i = j$  then  $t_i[j] = i$ , otherwise  $t_i[j] = 0$ ;
  - the objects  $t_i$ ,  $i \in \{2n + 1, \dots, 3n\}$ , such that: for each  $j \in \{1, \dots, n\}$ ,  $t_i[j] = -1$  if  $j = i$ ,  $t_i[j] = -2$  otherwise;
  - finally, three objects whose attribute values are all equal to  $-2$ .
- $o^{\mathcal{G}}$  is the object  $t_0$ .
- $\theta^{\mathcal{G},k}$  is equal to  $\frac{(n-k)^2+5n-4k}{(3n+4)^2}$ .

Next it is proved that the following holds:

$\mathcal{G}$  has a dominating set of size  $k$  iff  $\langle \mathbf{A}^{\mathcal{G}}, DB^{\mathcal{G}}, o^{\mathcal{G}}, 1, \theta^{\mathcal{G},k} \rangle_{(D)}$  is a “yes” instance of the OPD(D) problem.

( $\Rightarrow$ ) Let  $W = \{v_{w_1}, \dots, v_{w_k}\}$  be a dominating set of  $\mathcal{G}$  having size  $k$ . On the set of attributes  $\mathbf{S}^W = \{A_{w_1}, \dots, A_{w_k}\}$ , the objects  $t_i$  ( $i \in \{1, \dots, 2n+1\}$ ) have a value different from  $\langle 0, \dots, 0 \rangle$ , since, if an object  $t_i$  had such value, the node  $v_i$  would be not in  $W$  and would be not adjacent to any node of  $W$ , and this is impossible since  $W$  encodes a dominating set.

Thus:

- $t_0$  (the object  $o^{\mathcal{G}}$ ) is the unique object assuming value  $\langle 0, \dots, 0 \rangle$  on  $\mathbf{S}^W$ ;
- Each object  $t_i$ ,  $i \in \{1, \dots, n\}$  is equal only to  $t_{i+n}$  on  $\mathbf{S}^W$ ;
- Since  $|W| = k$ , in the set of tuples  $\{t_i \mid 2n+1 \leq i \leq 3n\}$ , there are exactly  $n-k+3$  tuples having value  $\langle -2, \dots, -2 \rangle$  on  $\mathbf{S}^W$ , and  $k$  tuples s.t. each of them has a distinct value on  $\mathbf{S}^W$ ;
- The cumulated frequency histogram of  $\mathbf{S}^W$  in  $DB^{\mathcal{G}}$  consists of three pairs:  $\left(\frac{1}{3n+4}, \frac{k+1}{3n+4}\right)$ ,  $\left(\frac{2}{3n+4}, \frac{2n+k+1}{3n+4}\right)$ , and  $\left(\frac{n-k+3}{3n+4}, \frac{3n+4}{3n+4}\right)$ ; notice that, since  $0 \leq k \leq n$ , then  $\forall k, n-k+3 > 2$ ;
- $\text{out}_{\mathbf{S}^W}(o) = \left(\frac{2}{3n+4} - \frac{1}{3n+4}\right) \cdot \left(\frac{3n+4}{3n+4} - \frac{k+1}{3n+4}\right) + \left(\frac{n-k+3}{3n+4} - \frac{2}{3n+4}\right) \cdot \left(\frac{3n+4}{3n+4} - \frac{2n+k+1}{3n+4}\right) = \frac{(n-k)^2+5n-4k}{(3n+4)^2}$ .

Then,  $\mathbf{S}^W$  is a property of size  $k$  s.t.  $\text{out}_{\mathbf{S}^W}(o) \geq \theta^{\mathcal{G},k}$ .

( $\Leftarrow$ ) Let  $\mathbf{S}^W = \{A_{s_1}, \dots, A_{s_k}\}$  be a property s.t.  $\text{out}_{\mathbf{S}^W}(o) \geq \sigma = \frac{(n-k)^2+5n-4k}{(3n+4)^2}$ .

If there existed an object  $t_i \neq t_0$  in  $DB^{\mathcal{G}}$  assuming value  $\langle 0, \dots, 0 \rangle$  on  $\mathbf{S}^W$ , then the frequency associated to the object  $o^{\mathcal{G}}$  in the cumulated frequency histogram of  $\mathbf{S}^W$  would be equal to or greater than  $\frac{3}{3n+4}$ .

Since the value assumed by the object  $t_i$ , for  $1 \leq i \leq 2n$ , is taken only by the object  $t_{i+n}$  and by no other objects in  $DB^{\mathcal{G}}$ , the frequency of  $t_i$  is  $\frac{2}{3n+4}$ .



0	<b>DB</b>				
0					
0					
0					
0					
1	$\Phi 1$	...	...	...	$\Phi 1$
...	...	...	...	...	...
1	$\Phi m$	...	...	...	$\Phi m$

**Fig. 5.8.** Example of reduction used in Theorem 4

Furthermore, there are  $n - k + 3$  tuples assuming value  $\langle -2, \dots, -2 \rangle$  on  $\mathbf{S}^W$ , and  $k$  tuples s.t. taking values different from one another.

Summarizing, there were only  $n - k + 3$  tuples with frequency greater than  $\frac{3}{3n+4}$ . Thus  $\text{out}_{\mathbf{S}^W}(o)$  would be at most equal to  $\left(\frac{n-k+3}{3n+4} - \frac{3}{3n+4}\right) \left(\frac{n-k+3}{3n+4}\right) = \frac{(n-k)^2 + 3n - 3k}{(3n+1)^2} < \sigma$ . But this is impossible by the definition of  $\mathbf{S}^W$ , a contradiction.

It can be concluded that no object  $t_i$ , for  $1 \leq i \leq 2n$ , assumes value  $\langle 0, \dots, 0 \rangle$  on  $\mathbf{S}^W$ . Hence, for each node  $v_i$  of  $\mathcal{G}$ , either  $v_i$  belongs to  $W$  or  $v_i$  is adjacent to at least one node in the set  $W = \{v_{s_1}, \dots, v_{s_k}\}$ , so that  $W$  is a dominating set of size  $k$ .

The previous result obtained for  $\sigma = 1$  can be used to determine the complexity of the general OPD(D) problem. The proof is by reduction of the problem  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  to the problem  $\langle \mathbf{A}, DB, o, \sigma, \theta \rangle_{(D)}$ .

**Theorem 4.** *The OPD(D) problem is NP-complete.*

*Proof. (Membership)* A certificate for  $\langle \mathbf{A}, DB, o, \sigma, \theta \rangle_{(D)}$  is given by a pair  $(\mathbf{E}, \mathbf{S})$ , where  $\mathbf{E}$  and  $\mathbf{S}$  are disjoint subsets of  $\mathbf{A}$ . Since  $\text{freq}_{\mathbf{E}}(o)$  and  $\text{out}_{\mathbf{E}, \mathbf{S}}(o)$  can be computed in polynomial time (see Theorem 2), it can be checked in polynomial time that  $\mathbf{E}$  is a  $\sigma$ -explanation and  $\mathbf{S}$  is a  $\theta$ -outlying property for  $o$ .

*(Hardness)* The proof is by reduction of the NP-complete  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  problem.

Given  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$ :

- let  $\mathbf{A}^* = \mathbf{A} \cup \{a_E\}$ , where  $a_E$  is a novel attribute distinct from attributes in  $\mathbf{A}$ ;
- let  $DB^*$  be a database obtained from  $DB$  as follows:
  - let  $DB' = \{t \mid t[a_E] = 0 \wedge t[\mathbf{A}] \in DB\}$ ;
  - let  $DB'' = \{t_i \mid i \in \{1, 2, \dots, m\} \wedge t_i[a_E] = 1 \wedge t_i[\mathbf{A}] = \langle \Phi i, \dots, \Phi i \rangle\}$ , where  $m = \frac{1-\sigma}{\sigma} |DB|$  and for each  $i$  ( $1 \leq i \leq m$ ),  $\Phi i$  is a distinct value not belonging to any active domain of DB attributes.
  - $DB^* = DB' \cup DB''$

An example of such a reduction is reported in Figure 5.8.

Next it is proved that the following holds:  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  is a “yes” instance if and only if  $\langle \mathbf{A}^*, DB^*, o, \sigma, \theta \rangle_{(D)}$  is a “yes” instance.

( $\Rightarrow$ ) Assume that  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  is a YES instance. Then, there exists a subset  $\mathbf{S}$  of  $\mathbf{A}$  that is a  $\theta$ -outlying property for  $o$  w.r.t.  $DB$ . Let  $\mathbf{E}$  be equal to  $\{a_E\}$ , then by construction it follows that  $\text{freq}_{\mathbf{E}}(o) \geq \sigma$  and that  $\mathbf{S}$  is a  $\theta$ -outlying property with explanation  $\{a_E\}$  for  $o$  w.r.t.  $DB^*$ . Hence,  $(\mathbf{E}, \mathbf{S})$  is an explanation-property pair for  $o$  in  $DB^*$ , and then,  $\langle \mathbf{A}^*, DB^*, o, \sigma, \theta \rangle_{(D)}$  is a “yes” instance.

( $\Leftarrow$ ) Assume that  $\langle \mathbf{A}^*, DB^*, o, \sigma, \theta \rangle_{(D)}$  is a YES instance. Then, there exists a  $(\sigma, \theta)$ -pair  $(\mathbf{E}, \mathbf{S})$  of disjoint subsets of  $\mathbf{A}^*$ .

It is recalled that for each attribute  $a$  in  $\mathbf{A}^*$  there is no object in  $DB''$  which agrees with the object  $o$  on  $a$ , that is  $\forall a \in \mathbf{A}^*, \forall i \in \{1, 2, \dots, m\}, o[a] \neq t_i[a]$ .

First of all consider the case in which  $\mathbf{E}$  is empty. Then,  $\mathbf{S} - \{a_E\}$  is a  $\theta$ -outlying property with empty explanation w.r.t.  $DB$ . Indeed, by construction, it holds the following

$$\text{out}_{\mathbf{S}}^{DB^*}(o) \leq \text{out}_{\{a_E\}, \mathbf{S} - \{a_E\}}^{DB^*}(o) \leq \text{out}_{\mathbf{S} - \{a_E\}}^{DB}(o).$$

Assume now that the set  $\mathbf{E}$  is not empty. From what above recalled, it holds that  $\mathbf{E}$  is such that  $DB_{o[\mathbf{E}]}^* \subseteq DB'$ . Moreover, since  $\sigma|DB^*| = |DB'|$ , it follows that  $DB_{o[\mathbf{E}]}^* = DB'$ .

Now, there are two cases to consider:

1. if  $a_E \notin \mathbf{S}$  then  $\mathbf{S}$  is a  $\theta$ -outlying property for  $o$  w.r.t.  $DB'[\mathbf{A}]$ , that is,  $DB$ ;
2. if  $a_E \in \mathbf{S}$ , since  $t_i[a_E] = t_j[a_E]$  for each  $t_i, t_j \in DB'$ ,  $\mathbf{S}' = \mathbf{S} - \{a_E\}$  is also a  $\theta$ -outlying property for  $o$  w.r.t.  $DB'$ . Then,  $\mathbf{S}'$  is a  $\theta$ -outlying property for  $o$  w.r.t.  $DB'[A]$ , that is,  $DB$ .

In all cases, an outlying property for  $o$  in  $DB$  can be determined, and then  $\langle \mathbf{A}, DB, o, 1, \theta \rangle_{(D)}$  is a YES instance.

It immediately follows from the result above that the computation version of our problem (that is, the OPD problem) is NP-hard. This result can however be sharpened. Let us proceed with first characterizing the complexity of the optimization version OPD(O) of the problem.

**Theorem 5.** *The OPD(O) problem is  $F\Delta_2^P[\mathcal{O}(\log n)]$ -complete.*

*Proof.* (Membership) First of all, it can be noted that, given a data set, the range of values on which the outlierness function ranges is polynomially large in the number of objects stored in the data set. Indeed, let  $DB$  be a data set composed of  $n$  objects,  $o$  a generic object of  $DB$ , and  $\mathbf{S}$  a generic property. It follows from Theorem 2 that the outlierness function  $\text{out}_{\mathbf{S}}(o)$  outputs a rational number belonging to the set  $Q_n = \{\frac{i}{n^2} \mid i \in \{0, 1, 2, \dots, n^2 - 1, n^2\}\}$ . Thus, the number of distinct values assumed by  $\text{out}_{\mathbf{S}}(o)$  is upper bounded by

$n^2$ . Consider now a generic explanation  $\mathbf{E}$ . The number of objects  $j$  of the data set  $DB_{o[\mathbf{E}]}$  is always such that  $1 \leq j \leq n$ . Hence, the results returned by the function  $\text{out}_{\mathbf{S},\mathbf{E}}(o)$  belong to the set  $Q = Q_1 \cup Q_2 \cup \dots \cup Q_n$ , whose cardinality  $|Q|$  is  $\sum_{j=1}^n j^2 = 1^2 + 2^2 + \dots + (n-1)^2 + n^2 \leq n^3$ . It can be concluded that the number of distinct values assumed by the function  $\text{out}_{\mathbf{E},\mathbf{S}}(o)$  is upper bounded by  $n^3$ .

Let  $\mathbf{A}$  be the set of attributes of  $DB$ , and let  $\sigma$  be an explanation threshold. Let  $\theta^*$  be the maximum outlierness  $\text{out}_{\mathbf{E},\mathbf{S}}(o)$  associated with a explanation-property pair  $(\mathbf{E}, \mathbf{S})$  for the object  $o$  in  $DB$ , with  $\mathbf{E}$  a  $\sigma$ -explanation for  $o$ . Let  $\bar{Q}$  denote the set  $Q$  sorted in increasing order. Then, by a binary search in the ordered set of rational numbers  $\bar{Q}$ , the value  $\theta^*$  can be computed: at each step of the search, a threshold  $\bar{\theta}$  is given in  $\bar{Q}$ , and it is decided whether the OPD(D) problem  $\langle \mathbf{A}, DB, o, \sigma, \bar{\theta} \rangle_{(D)}$  has some solution. After  $\log |\bar{Q}| = \mathcal{O}(\log n)$  steps at most the procedure ends, and the value  $\theta^*$  can be returned. Since OPD(D) is feasible in NP, it follows that the described procedure is feasible in  $\text{FP}^{\text{NP}[\mathcal{O}(\log n)]}$ .

(Hardness) Recall that computing the size of the maximum clique in a graph is  $\text{F}\Delta_2^P[\mathcal{O}(\log n)]$ -complete [Chen and Toda, 1995]. Next, the NP-complete problem *Clique* is reduced to the OPD Problem.

Let  $\mathcal{G} = (V, E)$  be an undirected graph. By  $\mathcal{G}^c$  the graph  $(V, E^c)$  obtained from  $\mathcal{G}$  by replacing the set of edges  $E$  with  $E^c = \{\{u, v\} \mid u, v \in V \wedge u \neq v \wedge \{u, v\} \notin E\}$  is denoted. The graph  $\mathcal{G}^c$  has  $\frac{|V|^2 - |V|}{2} - |E|$  edges. Furthermore, By  $\bar{\mathcal{G}}$  the graph  $(\bar{V}, \bar{E})$  obtained from  $\mathcal{G}$  as follows:  $\bar{V}$  is the set of nodes  $V \cup \{uv \mid \{u, v\} \in E\}$ , and  $\bar{E}$  is the set of edges  $E \cup \{\{u, uv\}, \{v, uv\} \mid \{u, v\} \in E\}$ , is denoted. The graph  $\bar{\mathcal{G}}$  has  $|V| + |E|$  nodes.

The three following properties are known:

1.  $C$  is a clique of size  $k$  in  $\mathcal{G}$  if and only if  $V - C$  is a vertex cover of size  $|V| - k$  in  $\mathcal{G}^c$  [Karp, 1972],
2.  $C$  is a vertex cover of size  $k$  in  $\mathcal{G}$  if and only if  $C$  is dominating set of size  $k$  in  $\bar{\mathcal{G}}$  [Garey and Johnson, 1979], and
3. by Theorem 3 above,  $C = \{v_1, \dots, v_k\}$  is a dominating set of size  $k$  in  $\mathcal{G} = (V, E)$  if and only if  $\mathbf{S}_C = \{A_1, \dots, A_k\} \subseteq \mathbf{A}^{\mathcal{G}}$  is a  $\theta_{n,k}$ -outlying property for the object  $o^{\mathcal{G}}$  in the data set  $DB^{\mathcal{G}}$  with explanation threshold  $\sigma = 1$ , where  $\theta_{n,k} = \theta^{\mathcal{G},k} = \frac{(n-k)^2 + 5n - 4k}{(3n+4)^2}$  and  $n = |V|$ .

Let  $\mathcal{G} = (V, E)$ ,  $n = |V|$ , and  $m = |E|$ . It follows from properties (1-3) above that  $C$  is a clique of size  $k$  in  $\mathcal{G} \iff V - C$  is a vertex cover of size  $n - k$  in the graph  $\mathcal{G}^c$  (having  $n$  nodes and  $\frac{n^2 - n}{2} - m$  edges)  $\iff V - C$  is a dominating set of size  $n - k$  in the graph  $\bar{\mathcal{G}}^c$  (having  $n' = \frac{n^2 + n}{2} - m$  nodes)  $\iff \mathbf{S}_{V-C}$  is a  $\theta_{n',(n-k)}$ -outlying property for  $t_0$  in the data set  $DB^{\bar{\mathcal{G}}^c}$  with explanation threshold  $\sigma = 1$ .

The outlierness  $\theta_{n',n-k}$  is monotonically increasing with  $k$ , indeed

$$\frac{\partial}{\partial k} \theta_{n', n-k} > 0 \iff k > k_0 = m - \frac{n^2 - n}{2} - 4,$$

but  $m \leq \frac{n^2 - n}{2}$ , and, hence,  $k_0 \leq -4$  and the condition  $k > k_0$  is always satisfied, being  $k \in \{1, 2, \dots, n\}$ .

In order to conclude the proof, observe that the clique  $C^*$  of  $\mathcal{G}$  having maximum size  $k^*$  is in one-to-one correspondence with the outlying property  $\mathbf{S}_{V-C^*}$  for  $t_0$  in  $DB_{\overline{\mathcal{G}}}$  having maximum outlierness  $\theta^* = \theta_{n', n-k^*}$  when the explanation threshold  $\sigma$  is set to 1, and, hence, computing  $\theta^*$  amounts to computing the size of the maximum clique.

Since the OPD(O) problem straightforwardly reduces to the OPD problem, the above theorem immediately establishes even an  $\text{F}\Delta_2^P[\mathcal{O}(\log n)]$ -hardness result for the OPD problem and, hence, a more precise lower bound to its computational complexity. To make this complexity scenario more complete, next also a tight upper bound to the complexity of this problem can be proved. Clear enough, just to compute the topmost explanation-property pair is a relevant special problem on its own. Therefore, the analysis starts from this specific problem, by assuming that  $k = 1$ .

**Theorem 6.** *The Outlying Property Detection problem for  $k = 1$  is in  $\text{F}\Delta_2^P$ .*

*Proof.* Let  $DB$  be a data set on the set of attributes  $\mathbf{A}$ , let  $o$  be an object of  $DB$ , and let  $\sigma, \theta \in [0, 1]$ . The top explanation-property  $(\sigma, \theta)$ -pair  $(\mathbf{E}^*, \mathbf{S}^*)$  can be computed by a deterministic polynomial time Turing machine with an oracle in NP as follows.

First of all, the machine determines the maximum outlierness value  $\theta^*$  associated with an explanation-property pair for  $o$  in  $DB$  with explanation threshold  $\sigma$ . This value can be computed with  $\mathcal{O}(\log |\mathbf{A}|)$  oracle calls, as detailed in the proof of Theorem 5. If  $\theta^*$  is less than  $\theta$ , then the machine terminates returning the empty set. Otherwise, the pair  $(\mathbf{E}^*, \mathbf{S}^*)$  exists, and the machine continues its work by executing the following steps:

1. set  $\mathbf{E}'$  to  $\emptyset$ .
2. for each  $A \in \mathbf{A}$ , if  $|DB_{o[\mathbf{E}' \cup \{A\}]}| \geq \sigma |DB|$ :
  - a) compute the maximum outlierness value  $\theta'$  associated with an explanation-property pair for  $o$  in  $DB_{o[\mathbf{E}' \cup \{A\}]}[\mathbf{A} - (\mathbf{E}' \cup \{A\})]$  with explanation threshold  $\sigma' = \frac{\sigma |DB|}{|DB_{o[\mathbf{E}' \cup \{A\}]}|}$ . The value  $\theta'$  can be computed with  $\mathcal{O}(\log |\mathbf{A}|)$  oracle calls, as detailed in the proof of Theorem 5.
  - b) if  $\theta' = \theta^*$  then set  $\mathbf{E}'$  to  $\mathbf{E}' \cup \{A\}$ .
3. set  $\mathbf{S}'$  to  $\mathbf{A} - \mathbf{E}'$ .
4. for each  $A \in \mathbf{S}'$ :
  - a) compute the maximum outlierness value  $\theta'$  associated with an outlying property for  $o$  in  $DB_{o[\mathbf{E}']}[\mathbf{S}' - \{A\}]$  with explanation threshold  $\sigma' = 1$ . The value  $\theta'$  can be computed with  $\mathcal{O}(\log |\mathbf{A}|)$  oracle calls, as detailed in the proof of Theorem 5.

b) if  $\theta' = \theta^*$  then set  $\mathbf{S}'$  to  $\mathbf{S}' - \{A\}$ .

Let us show that steps 2 and 4 above are correct.

First, it is proved that, after the second step,  $\mathbf{E}'$  is the explanation of an outlying property  $\mathbf{S}'$  such that  $(\mathbf{E}', \mathbf{S}')$  is a  $(\theta^*, \sigma)$ -pair. By construction,  $\mathbf{E}'$  is such that, at each iteration, in  $\mathbf{A} - (\mathbf{E} \cup \{A\})$  there exist disjoint subsets  $\mathbf{E}''$  and  $\mathbf{S}'$  such that the pair  $(\mathbf{E}' \cup \mathbf{E}'', \mathbf{S}')$  is a  $(\theta^*, \sigma)$ -pair. As a consequence,  $\mathbf{E}'$  is contained in the optimum set  $\mathbf{E}^*$ . By contradiction, assume that at the end of step 2 the set  $\mathbf{E}'$  is not equal to  $\mathbf{E}^*$ . Then, there exists an attribute  $A \in \mathbf{E}^*$  not in  $\mathbf{E}'$ . Let  $\mathbf{E}_A$  denote the value of  $\mathbf{E}'$  at the beginning of the iteration in which the attribute  $A$  is considered for insertion. Since  $A$  was not added to  $\mathbf{E}_A$ , it follows that in  $\mathbf{A} - (\mathbf{E}_A \cup \{A\})$  there not exist disjoint subsets  $\mathbf{E}''$  and  $\mathbf{S}'$  such that the pair  $(\mathbf{E}_A \cup \{A\} \cup \mathbf{E}'', \mathbf{S}')$  is a  $(\theta^*, \sigma)$ -pair. But, this is not possible since  $\mathbf{E}^*$  can be always expressed as  $\mathbf{E}_A \cup \{A\} \cup \mathbf{E}''$ .

Now, let us consider step 4. By construction,  $\mathbf{S}'$  is such that, at each iteration, there exists a subset  $\mathbf{S}''$  of  $\mathbf{S}'$  such that the pair  $(\mathbf{E}', \mathbf{S}'')$  is a  $(\theta^*, \sigma)$ -pair. As a consequence,  $\mathbf{S}'$  contains the optimum set  $\mathbf{S}^*$ . By contradiction, assume that at the end of step 4 the set  $\mathbf{S}'$  is not equal to  $\mathbf{S}^*$ . Then, there exists an attribute  $A \in \mathbf{S}'$  not in  $\mathbf{S}^*$ . Let  $\mathbf{S}_A$  denote the value of  $\mathbf{S}'$  at the beginning of the iteration in which the attribute  $A$  is considered for deletion. Since  $A$  was not removed from  $\mathbf{S}_A$ , it follows that there does not exist a subset  $\mathbf{S}''$  of  $\mathbf{S}_A - \{A\}$  such that the pair  $(\mathbf{E}', \mathbf{S}'')$  is a  $(\theta^*, \sigma)$ -pair. But, this is not possible since  $\mathbf{S}^*$  is contained in  $\mathbf{S}_A - \{A\}$ .

Thus, the sets  $\mathbf{E}'$  and  $\mathbf{A}'$  computed by the above procedure represent the sets  $\mathbf{E}^*$  and  $\mathbf{A}^*$ , respectively.

To conclude the proof, note that the total number of oracle calls performed by the Turing machine is  $\mathcal{O}(|\mathbf{A}| \log |\mathbf{A}|)$ , hence polynomial w.r.t. the size of the input.

Next, the recognition version OPD(S) of the OPD problem is analyzed. This result will be exploited in the following in order to characterize the complexity of the OPD problem in a more general setting.

**Theorem 7.** *The Outlying Property Solution problem is co-NP-complete.*

*Proof.* (Membership) A “no” instance of the problem presented in Definition 11 has a succinct disqualification represented by an explanation-property  $(\sigma, \theta)$ -pair  $(\overline{\mathbf{E}}, \overline{\mathbf{S}})$ , not belonging to  $Sol$ , such that either (1)  $\text{out}_{\overline{\mathbf{E}}, \overline{\mathbf{S}}}(o) \geq \theta$ , if  $h < k$ , or (2)  $\text{out}_{\overline{\mathbf{E}}, \overline{\mathbf{S}}}(o) > \min_i \{\text{out}_{\mathbf{E}_i, \mathbf{S}_i}(o)\}$ , if  $h = k$ .

(Hardness) It is recalled that deciding whether a boolean formula in conjunctive normal form is unsatisfiable is a co-NP-complete problem, also known as UNSAT. This problem remains co-NP-complete even if clauses are restricted to contain at most three literals. UNSAT is the complement of the well-known problem SAT.

Now a reduction from the UNSAT problem to the OPD(S) problem can be shown. Let  $\phi$  denote a 3CNF formula with  $n$  variables and  $m$  clauses. It is known that there exists a polynomial time transformation  $\mathcal{G}(\phi)$  from SAT

to the *Vertex Cover* problem [Garey and Johnson, 1979], with  $\mathcal{G}(\phi)$  a graph composed of  $2n + 3m$  nodes and  $n + 6m$  edges, such that  $\phi$  is satisfiable if and only if the graph  $\mathcal{G}(\phi)$  has a vertex cover of size at most  $K = n + 2m$ .

The transformations recalled in the Theorem 5 can be exploited to prove that the formula  $\phi$  is satisfiable  $\iff$  the graph  $\mathcal{G}(\phi)$  has a vertex cover of size at most  $K = n + 2m \iff$  the graph  $\overline{\mathcal{G}(\phi)}$ , having  $N = 3n + 9m$  nodes, has a dominating set of size at most  $K \iff$  there exists a  $\theta^{\overline{\mathcal{G}(\phi)}, K}$ -outlying property for the object  $o^{\overline{\mathcal{G}(\phi)}}$  in the data set  $DB^{\overline{\mathcal{G}(\phi)}}$  with explanation threshold  $\sigma = 1$  (see Theorems 3 and 5).

By construction, it follows that the formula  $\phi$  is unsatisfiable if and only if

$$\langle \mathbf{A}^{\overline{\mathcal{G}(\phi)}}, DB^{\overline{\mathcal{G}(\phi)}}, o^{\overline{\mathcal{G}(\phi)}}, 1, \theta^{\overline{\mathcal{G}(\phi)}, K}, k, \emptyset \rangle_{(S)}$$

is a “yes” instance, with  $k > 0$  an arbitrary integer, that is, if the empty set is the solution of the OPD problem associated with  $\phi$ .

From the practical viewpoint, it is of interest singling out the top- $k$  outlying properties for small values of  $k$  (in several application cases, it will be supposedly enough to consider  $k = 1$ ). The notion of “small” number of solution can be reasonably formalized by requiring that  $k$  does not grow too much with respect to the number  $m$  of attributes of the database, that is, that the value  $k$  is polynomially related to  $m$  as  $k \in \mathcal{O}(m^p)$ , for a fixed positive integer  $p$ . Next, a strict upper bound to complexity of the OPD problem for any such restricted value of  $k$  is proved.

**Theorem 8.** *The OPD problem for  $k = \mathcal{O}(m^p)$  is in  $\text{FS}_2^{\text{P}}$ .*

*Proof.* A nondeterministic polynomial time transducer with an oracle in NP solving the OPD problem is built. Let  $m$  denote the number of attributes of the input data set. The machine performs the following steps:

1. guesses the number  $h \leq k$  of solutions of the OPD problem. Since  $h$  is encoded with  $\mathcal{O}(\log m)$  bits, the required time is logarithmical in the input size;
2. guesses the set  $Sol = \{(\mathbf{E}_1, \mathbf{S}_1), (\mathbf{E}_2, \mathbf{S}_2), \dots, (\mathbf{E}_h, \mathbf{S}_h)\}$  composed of exactly  $h$   $(\sigma, \theta)$ -pairs. The set  $Sol$  can be encoded with at most  $\mathcal{O}(m^{p+1})$  bits, since it is composed of at most  $\mathcal{O}(m^p)$  pairs of disjoint subsets of attributes and each pair can be encoded by means of  $2m$  bits. Hence, this step can be accomplished in polynomial time;
3. verifies in polynomial time that they are actually  $(\sigma, \theta)$ -pairs;
4. calls the oracle to verify that  $Sol$  is the solution set of the OPD problem (see Theorem 7 above);
5. if  $Sol$  is the solution set, writes  $Sol$  on the output tape.

The total number of steps performed by the transducer is polynomially bounded w.r.t. the size of the input, and the machine makes use of an oracle in NP. This completes the proof.

Problem	Complexity
OPD(D)	NP-complete (Th. 3, 4)
OPD(O)	$F\Delta_2^P[\mathcal{O}(\log n)]$ -complete (Th. 5)
OPD(S)	co-NP-complete (Th. 7)
OPD	$F\Delta_2^P[\mathcal{O}(\log n)]$ -hard (Th. 5)
	in $F\Delta_2^P$ , for $k = 1$ (Th. 6)
	in $F\Sigma_2^P$ , for $k = \mathcal{O}(m^p)$ (Th. 8)

**Table 5.1.** Computational complexity results concerning the outlying property detection problem.

Table 5.1 summarizes the computational complexity results concerning the Outlying Property Detection problem.

## 5.7 Upper Bound Properties

In the previous section it has been shown that the OPD problem is intractable. It follows that, in order to detect the top outlying properties, a possibly smart search algorithm is needed and it should be capable of delivering the result by visiting an as-much-restricted-as-possible part of the search space (which, overall, would include all the explanation-property pairs). In other words, it is sensible to devise techniques by which such a potentially demanding search can be effectively pruned. To this end, it is very useful to early detect pairs  $(\mathbf{E}, \mathbf{S})$  that can be disregarded in the search. This issue is the subject of this section.

In the sequel, let  $o$  denote an outlier object of a database  $DB$ ,  $\mathbf{S}$  a property of  $DB$ ,  $\sigma \in [0, 1]$  a frequency threshold,  $(\mathbf{E}, \mathbf{S})$  a  $(\sigma, \theta)$  explanation-property pair (with  $\mathbf{S}$  and  $\mathbf{E}$  disjoint sets). Moreover, let  $DB'$  be the database  $DB_{o[\mathbf{E}]}$ ,  $\mathbf{S}'$  denote a generic superset of  $\mathbf{S}$ , and  $\mathbf{E}' \supseteq \mathbf{E}$  be a generic explanation such that  $(\mathbf{E}', \mathbf{S}')$  is a  $(\sigma, \theta)$  explanation-property pair (with  $\mathbf{S}'$  and  $\mathbf{E}'$  disjoint sets).

Two basic questions naturally arise, that are:

- A Is it possible to compute an upper bound to the outlierness of the property  $\mathbf{S}'$  in  $o$  w.r.t.  $DB$  with explanation  $\mathbf{E}'$ ?
- B It is possible to compute an upper bound to the outlierness of the property  $\mathbf{S}'$  in  $o$  w.r.t.  $DB$  with explanation  $\mathbf{E}'$ ?

These upper bounds should be both accurate and fast to compute, in order to be effectively exploited for the sake of pruning our search space. The following Theorems 9 and 10 provide an answer to questions A and B, respectively.

First, Question A is analyzed. It is worth noticing that the *outlierness is not monotone w.r.t. property inclusion*. That is, fixed an explanation set  $\mathbf{E}$  and an object  $o$ , for any pair of sets of attributes  $\mathbf{S}$  and  $\mathbf{S}'$  such that  $\mathbf{S} \subseteq \mathbf{S}'$ ,

the outlierness  $\text{out}_{\mathbf{E},\mathbf{S}'}(o)$  can be the same, greater, or smaller than  $\text{out}_{\mathbf{E},\mathbf{S}}(o)$ . Nevertheless, a tight upper bound to the outlierness of any superset of  $\mathbf{S}$  can be provided, as showed in the following.

Let  $h$  be a frequency histogram, and let  $f$  be a frequency in  $h$ . In the rest of the section, for the sake of simplicity, the notation  $\text{out}^h(f)$  will be used to refer to the outlierness of a property  $\mathbf{S}$ , with  $h = \text{hist}_{\mathbf{S}}$ , associated with an object  $o$ , such that  $f = \text{freq}_{\mathbf{S}}(o)$ .

Before presenting the next main results, three technical lemmata are needed to be proved. These lemmata provide upper bounds to the value of the outlierness function  $\text{out}^{h^c}(f_c)$ , where  $c$  denotes a non-negative integer, while  $h^c$  and  $f_c$  denote certain families of histograms and frequency values, respectively, that are of interest for proving main results.

**Lemma 1.** *Let  $h$  be the frequency histogram*

$$h = \left\{ \frac{b}{n}, \frac{a_1}{n}, \dots, \frac{a_k}{n} \right\}.$$

*Consider the family of histograms*

$$h^c = \left\{ \frac{c}{n}, \frac{b-c}{n}, \frac{a_1}{n}, \dots, \frac{a_k}{n} \right\},$$

*with  $1 \leq c \leq b$  and  $n = b + \sum_i a_i$ . Then  $f(c) = \text{out}^{h^c}(\frac{c}{n})$  has its maximum in  $c = 1$ .*

Before presenting the proofs, it is needed to introduce an alternative formula for computing the outlierness which is derived from formula of Theorem 2 and that will be exploited in the proofs. Given a frequency histogram  $h = \left\{ \frac{a_1}{n}, \dots, \frac{a_k}{n} \right\}$  and a frequency  $\frac{a_p}{n}$  in  $h$ , by Theorem 2, the outlierness  $\text{out}^h(\frac{a_p}{n})$  is equal to

$$\sum_{a_i > a_p} \frac{a_i}{n} \left( \frac{a_i}{n} - \frac{a_p}{n} \right)$$

Starting from the above formula,  $\text{out}^h(\frac{a_p}{n})$  can also be formulated as follows

$$\begin{aligned} \text{out}^h \left( \frac{a_p}{n} \right) &= \sum_{a_i > a_p} \frac{a_i}{n} \frac{a_i}{n} - \sum_{a_i > a_p} \frac{a_i}{n} \frac{a_p}{n} = \\ &= \sum_{a_i > a_p} \frac{a_i}{n} \frac{a_i}{n} - \sum_{a_i > a_p} \frac{a_i}{n} \frac{a_p}{n} + \sum_{a_i \leq a_p} \frac{a_i}{n} \frac{a_p}{n} - \sum_{a_i \leq a_p} \frac{a_i}{n} \frac{a_p}{n} = \\ &= \sum_{a_i} \frac{a_i}{n} \frac{\max\{a_i, a_p\}}{n} - \frac{a_p}{n} \sum_{a_i} \frac{a_i}{n} = \\ &= \sum_{a_i} \frac{a_i}{n} \frac{\max\{a_i, a_p\}}{n} - \frac{a_p}{n} \cdot 1 = \\ &= \frac{(\sum_i a_i \max\{a_i, a_p\}) - a_p n}{n^2}. \end{aligned}$$



In particular, if  $a_p = 1$ , it holds that

$$\text{out}^h\left(\frac{1}{n}\right) = \frac{(\sum a_i^2) - (\sum a_i)}{(\sum a_i)^2}.$$

*Proof. of Lemma 1.* Next, it is proved that the function

$$\bar{f}(c) = \text{out}^{h^c}\left(\frac{c}{n}\right) = \frac{[(\sum_i a_i \max\{a_i, c\}) + c^2 + (b - c) \max\{b - c, c\}] - c(b + \sum_i a_i)}{(b + \sum_i a_i)^2}$$

is not increasing. This function is a discrete one, since  $c$  assumes only non-negative integer values.

Consider the function  $f(c)$ , defined as

$$f(c) = \frac{[(\sum_i a_i \max\{a_i, c\}) + c^2 + (b - c) \max\{b - c, c\}] - c(b + \sum_i a_i)}{(b + \sum_i a_i)^2}$$

with  $c$  now ranging in  $\mathbb{R}^+$ .

Clearly,  $f(c) = \bar{f}(c)$  whenever  $c$  assumes integer values. Moreover, if  $f(c)$  is not increasing then also  $\bar{f}(c)$  has to be not increasing.

In order to prove that  $f(c)$  is not increasing, the derivative of  $f(c)$  is computed:

$$[f(c)]' = \frac{2c - b + t(c) - (\sum_{a_i \geq c} a_i)}{n^2}$$

where  $t(c) = 2(c - b)$  for  $c \leq \lfloor \frac{b}{2} \rfloor$ , and  $t(c) = b - 2c$  for  $c \geq \lfloor \frac{b}{2} \rfloor$ . To complete the proof, it is sufficient to show that  $[f(c)]' \leq 0$ . For  $c \leq \lfloor \frac{b}{2} \rfloor$ ,  $f(c)$  is decreasing and the inequality is strict:

$$\left( \forall c \leq \left\lfloor \frac{b}{2} \right\rfloor \right), [f(c)]' < 0 \iff \sum_{a_i \geq c} a_i < 4c - 3b.$$

The above inequality is always verified, since  $(\sum_{a_i \geq c} a_i) \geq 0$  as the  $a_i$ s are positive integers and  $4c - 3b < 0$ . Indeed,  $4c \leq 4\lfloor \frac{b}{2} \rfloor \leq 2b < 3b$ . Furthermore, for  $c \geq \lfloor \frac{b}{2} \rfloor$ ,  $f(c)$  is not increasing:

$$\left( \forall c \geq \left\lfloor \frac{b}{2} \right\rfloor \right), [f(c)]' \leq 0 \iff \sum_{a_i \geq c} a_i \geq 0.$$

Note that the latter inequality is always satisfied. In particular  $f(c)$  is strictly decreasing between 1 and  $\max_i\{a_i\}$  and equals zero between  $\max_i\{a_i\}$  and  $b$ .  
□

**Lemma 2.** Let  $h$  be the histogram of frequencies

$$h = \left\{ \frac{1}{n+1}, \frac{a_1}{n+1}, \dots, \frac{a_k}{n+1} \right\}$$

with  $a_i$  ( $1 \leq i \leq k$ ) positive integers, and  $n = \sum_i a_i$ . Let  $c$  be an integer such that  $a_1 < c \leq n$ . Consider the histogram

$$h^c = \left\{ \frac{1}{n+1}, \frac{A-c}{n+1}, \frac{c}{n+1}, \frac{a_{j+1}}{n+1}, \dots, \frac{a_k}{n+1} \right\},$$

where  $j$  is the smallest integer such that  $A = \sum_{i \leq j} a_i$  is greater than or equal to  $c$ . Then  $\text{out}^h(\frac{1}{n+1}) > \text{out}^{h^c}(\frac{1}{n+1})$ .

**Proof. of lemma 2.** PROOF OF LEMMA 2. First of all, consider the two expressions:

$$\begin{aligned} \text{out}^h\left(\frac{1}{n+1}\right) &= \frac{(\sum_i a_i^2) - (\sum_i a_i)}{(n+1)^2}, \quad \text{and} \\ \text{out}^{h^c}\left(\frac{1}{n+1}\right) &= \frac{(\sum_i a_i^2) - (\sum_i a_i) - [(\sum_{i \leq j} a_i^2) - A] + [((A-c)^2 + c^2) - ((A-c) + c)]}{(n+1)^2}. \end{aligned}$$

Notice now that

$$\text{out}^{h^c}\left(\frac{1}{n+1}\right) - \text{out}^h\left(\frac{1}{n+1}\right) > 0 \iff (A-c)^2 + c^2 > \sum_{i \leq j} a_i^2$$

But,  $(A-c)^2 + c^2 \geq c^2 \geq (\sum_{i \leq j} a_i)^2 > \sum_{i \leq j} a_i^2$ , and, hence, the result follows.  $\square$

**Lemma 3.** Let  $h^c$  be the histogram of frequencies

$$\left\{ \frac{1}{n+c+1}, \frac{c}{n+c+1}, \frac{a_1}{n+c+1}, \dots, \frac{a_k}{n+c+1} \right\},$$

with (w.l.o.g.)  $1 \leq a_1 \leq \dots \leq a_k$  integers,  $a_k > 1$ ,  $n = \sum_i a_i$ , and where  $c$  is an integer number such that  $0 \leq c \leq a_k$ . Let  $g(c)$  denote the function  $\text{out}^{h^c}(\frac{1}{n+c+1})$ . Then  $g(c)$  is convex and it is maximized in  $c = 0$ .

**Proof. of lemma 3.** PROOF OF LEMMA 3. By using the formula of Theorem 2, the function  $g(c)$  can be expressed as follows:

$$g(c) = \frac{(\sum_i a_i^2) - (\sum_i a_i) + c(c-1)}{(\sum_i a_i + c + 1)^2}.$$

First of all, the following claim is proved.

*Claim.* For  $c \in [0, a_k]$  and  $c \in \mathbb{N}$ , the function  $g(c)$  is convex.

*Proof.* In order to prove that  $g(c)$  is convex, the function  $\bar{g}(c)$  defined as

$$\bar{g}(c) = \frac{(\sum_i a_i^2) - (\sum_i a_i) + c(c-1)}{(\sum_i a_i + c + 1)^2}$$

with  $c \in [0, a_k]$  and  $c \in \mathbb{R}$ , is considered.

Clearly, if  $\bar{g}(c)$  is convex then also  $g(c)$  is convex. Thus, in order to prove the claim,  $\bar{g}(c)$  is proved to be convex. To this end, it can be verified that the second derivative of  $\bar{g}(c)$  is greater than or equal to 0 for all  $c \in [0, a_k]$ .

The first order derivative of  $\bar{g}(c)$  is:

$$\begin{aligned}\bar{g}'(c) &= \frac{(2c-1)((\sum_i a_i + 1) + c) - 2((\sum_i a_i^2 - \sum_i a_i) + c^2 - c)}{((\sum_i a_i + 1) + c)^3} = \\ &= \frac{2(\sum_i a_i + 1)c + c - (\sum_i a_i + 1) + 2(\sum_i a_i^2 - \sum_i a_i)}{((\sum_i a_i + 1) + c)^3};\end{aligned}$$

Computing the derivative of  $\bar{g}'(c)$ , it follows:

$$\bar{g}''(c) = \frac{(2(\sum_i a_i + 1) + 1)((\sum_i a_i + 1) + c) - 3(2(\sum_i a_i + 1)c + c - (\sum_i a_i + 1) - 2(\sum_i a_i^2 - \sum_i a_i))}{((\sum_i a_i + 1) + c)^4}$$

Now, the range of  $c$  in which the second derivative  $\bar{g}''(c) \geq 0$  has to be computed, and, then, the values of  $c$  such that:

$$(2(\sum_i a_i + 1) + 1)((\sum_i a_i + 1) + c) - 3(2(\sum_i a_i + 1)c + c - (\sum_i a_i + 1) - 2(\sum_i a_i^2 - \sum_i a_i)) \geq 0$$

that is equivalent to

$$(\sum_i a_i)^2 + 2\sum_i a_i + 1 + 2\sum_i a_i + 2 + 3\sum_i a_i^2 - 3\sum_i a_i \geq c(3 + 2\sum_i a_i)$$

Then  $\bar{g}(c)$  is convex for:

$$c \leq \frac{(\sum_i a_i)^2 + \sum_i a_i + 3\sum_i a_i^2 + 3}{3 + 2\sum_i a_i}$$

For concluding the proof, it must be shown that

$$a_k \leq \frac{(\sum_i a_i)^2 + \sum_i a_i + 3\sum_i a_i^2 + 3}{3 + 2\sum_i a_i}$$

The latter inequality can be rewritten as:

$$\begin{aligned}a_k(3 + 2\sum_i a_i) &\leq (\sum_i a_i)^2 + \sum_i a_i + 3\sum_i a_i^2 + 3 \\ \Rightarrow 3a_k + 2a_k \sum_i a_i &\leq (\sum_i a_i)^2 + \sum_i a_i + 3\sum_i a_i^2 + 3 \\ \Rightarrow 3a_k &\leq \left(\sum_i^{k-1} a_i\right)^2 + \sum_i a_i + 3\sum_i^{k-1} a_i^2 + 3 + 2a_k^2 \\ \Rightarrow 2a_k^2 - 3a_k + 3 &+ \left(\sum_i^{k-1} a_i\right)^2 + \sum_i a_i + 3\sum_i^{k-1} a_i^2 \geq 0\end{aligned}$$

Since

$$\left(\sum_i^{k-1} a_i\right)^2 + \sum_i a_i + 3 \sum_i^{k-1} a_i^2 \geq 0$$

and

$$2a_k^2 - 3a_k + 3 \geq 0, \forall a_k \in \mathbb{R}$$

it can be concluded that  $\bar{g}(c)$  is convex for all  $c \leq a_k$ .

Now, the main proof of Lemma 3 can be resumed.

Since  $c \leq a_k$ , the following inequality holds:

$$g(c) \leq g^*(c) = \frac{(\sum_i a_i^2) - (\sum_i a_i) + a_k(a_k - 1)}{(\sum_i a_i + c + 1)^2}.$$

To determine the range of values of  $c$  for which  $g^*(c)$  is increasing, the function  $\bar{g}^*(c)$  is considered. It is defined as

$$\bar{g}^*(c) = \frac{(\sum_i a_i^2) - (\sum_i a_i) + a_k(a_k - 1)}{(\sum_i a_i + c + 1)^2}$$

with  $c \in \mathbb{R}^+$  and  $c \leq a_k$ .

The derivative of  $\bar{g}^*(c)$  is:

$$[\bar{g}^*(c)]' = \frac{-2(\sum_i a_i^2 - \sum_i a_i + a_k(a_k - 1))}{(\sum_i a_i + c + 1)^3} \geq 0.$$

Since  $\sum_i a_i^2 > \sum_i a_i, \forall a_i \in \mathbb{N}$  and  $a_k > 1$ , the numerator of  $[\bar{g}^*(c)]'$  is always less than 0, whereas the denominator is always greater than 0. It follows that  $\bar{g}^*$ , and clearly also  $g^*$ , are strictly decreasing for  $c \geq 0$  and they must have their maximum value in  $c = 0$ . To conclude, since (i)  $g^*(0) = g(0)$ , (ii)  $g^*$  has its maximum in  $c = 0$ , (iii)  $g^*$  is monotonically decreasing for  $c \geq 0$ , and (iv)  $g^*(c) \geq g(c)$ , then it is the case that also  $g$  has its maximum in  $c = 0$ .  $\square$

Now, question A can be considered and it is answered by the following theorem.

**Theorem 9.** Let  $h = \text{hist}_{\mathbf{S}}^{DB'} = \{f_1, \dots, f_k\}$  be the frequency histogram of  $DB'$  w.r.t.  $\mathbf{S}$ , let  $f$  be the frequency of  $o$  in  $DB'$  w.r.t.  $\mathbf{S}$ , and let  $n$  be the number of objects in  $DB'$ . Then

$$\text{out}_{\mathbf{E}, \mathbf{S}'}(o) \leq \left(\sum_i f_i^2\right) - \frac{(2f + 1)n - 2}{n^2}.$$

the term on the right of the inequality is called  $\text{oub}_{\mathbf{E}, \mathbf{S}}(o)$ ; it denotes an upper bound on the outlierness of  $\mathbf{S}'$  in  $o$  with explanation  $\mathbf{E}$ .

*Proof.* The proof uses Lemmas 1 and 2. Let  $f_i = \frac{a_i}{n}$ , for  $i = 1, \dots, n$ . First of all, note that a frequency histogram  $\bar{h}$  of  $DB'$  w.r.t. any superset  $\mathbf{S}'$  of  $\mathbf{S}$  is of the form

$$\bar{h} = \bigcup_{i=1}^k \left\{ \frac{a_{i,1}}{n}, \dots, \frac{a_{i,k_i}}{n} \right\}$$

with  $a_{i,1} + \dots + a_{i,k_i} = a_i$ , for each  $i = 1, \dots, k$ .

Let  $f = f_p$ , and  $h^*$  be the histogram

$$\left( h - \left\{ \frac{a_p}{n} \right\} \right) \cup \left\{ \frac{1}{n}, \frac{a_p - 1}{n} \right\} = \left\{ \frac{1}{n}, \frac{a_1}{n}, \dots, \frac{a_{p-1}}{n}, \frac{a_p - 1}{n}, \frac{a_{p+1}}{n}, \dots, \frac{a_k}{n} \right\}.$$

Next, it will be shown that  $\text{out}_{\mathbf{E}, \mathbf{S}'}(o) \leq \text{out}^{h^*}(\frac{1}{n})$ . By contradiction, assume that there exists an histogram  $\bar{h}$  of the form depicted above such that the frequency of  $o$  in  $\bar{h}$  is  $\frac{a_{p,1}}{n}$  and  $\text{out}^{\bar{h}}(\frac{a_{p,1}}{n}) > \text{out}^{h^*}(\frac{1}{n})$ . Consider the histogram  $\bar{h}^* = ((\bar{h} - \{\frac{a_{p,1}}{n}\}) \cup \{\frac{1}{n}, \frac{a_{p,1}-1}{n}\})$ . By Lemma 1 it follows that  $\text{out}^{\bar{h}^*}(\frac{a_{p,1}}{n}) \leq \text{out}^{\bar{h}}(\frac{a_{p,1}}{n})$ . Furthermore, by Lemma 2 it can be concluded that  $\text{out}^{\bar{h}^*}(\frac{1}{n}) \leq \text{out}^{h^*}(\frac{1}{n})$  since  $h^*$  can be obtained from  $\bar{h}^*$  by grouping together the frequencies  $\frac{a_{i,1}}{n}, \dots, \frac{a_{i,k_i}}{n}$  to obtain the frequency  $\frac{a_i}{n}$ , for each  $i = 1, \dots, p-1, p+1, \dots, k$ , and the frequencies  $\frac{a_{p,1}-1}{n}, \frac{a_{p,2}}{n}, \dots, \frac{a_{p,k_p}}{n}$  to obtain the frequency  $\frac{a_p-1}{n}$ . Hence, a contradiction is obtained.

Thus, the upper bound is given by the outlierness  $\text{out}^{h^*}(\frac{1}{n})$ , where  $h^* = \{f_1, \dots, f_{p-1}, f_{p+1}, \dots, f_k, \frac{1}{n}, \frac{nf_p-1}{n}\}$ . To conclude the proof, note that:

$$\begin{aligned} \text{out}^{h^*} \left( \frac{1}{n} \right) &= \left[ \left( \sum_{i \neq p} f_i^2 \right) + \left( \frac{1}{n} \right)^2 + \left( \frac{nf_p - 1}{n} \right)^2 \right] - \frac{1}{n} = \\ &= \left( \sum_{i \neq p} f_i^2 \right) + f_p^2 + \frac{1 - 2nf_p + 1 - n}{n^2} = \\ &= \left( \sum_i f_i^2 \right) - \frac{(2f + 1)n - 2}{n^2}. \end{aligned}$$

Theorem 9 above states that the outlierness of the test object  $o$  is maximized, w.r.t. any superset  $\mathbf{S}'$  of  $\mathbf{S}$ , when  $\mathbf{S}'$  preserves the actual histogram of frequencies, except for the case of making  $o$  different from any other object. For example, consider the portion of the database  $DB^{ex}$  depicted in Figure 5.9(a). Let  $o$  be  $c_5$ , let  $\mathbf{S} = \{\text{Tyres}\}$ , and let  $\mathbf{E} = \emptyset$ . If the set  $\mathbf{S}$  is augmented with the virtual column  $\mathbf{S}' - \mathbf{S}$ , then the maximum possible achievable outlierness value by  $c_5$  on any superset of  $\mathbf{S}$  is obtained. Notice that, while  $\text{out}_{\mathbf{S}}(o) = 0$ ,  $\text{out}_{\mathbf{S}'}(o) \simeq 0.35$ .

Having answered Question A, let us consider Question B. A strict upper bound based on the histogram of frequencies of  $DB'$  is provided next by the following theorem.

<i>Car ID</i>	<i>Tyres</i> ( <b>S</b> )	<b>S'</b> – <b>S</b>	<i>Car ID</i>	<i>Tyres</i> ( <b>S</b> )	<b>S'</b> – <b>S</b>	<b>E'</b>	<i>Car ID</i>	<i>Tyres</i> ( <b>S</b> )	<b>S'</b> – <b>S</b>	<b>E'</b>
c <sub>1</sub>	B	n	c <sub>1</sub>	B	n	n	c <sub>1</sub>	B	n	n
c <sub>2</sub>	B	n	c <sub>2</sub>	B	n	n	c <sub>2</sub>	B	n	n
c <sub>3</sub>	B	n	c <sub>3</sub>	B	n	n	c <sub>3</sub>	B	n	y
c <sub>4</sub>	B	n	c <sub>4</sub>	B	n	n	c <sub>4</sub>	B	n	y
c <sub>5</sub>	B	y	c <sub>5</sub>	B	y	y	c <sub>5</sub>	B	y	y
c <sub>6</sub>	M	n	c <sub>6</sub>	M	n	y	c <sub>6</sub>	M	n	y
c <sub>7</sub>	M	n	c <sub>7</sub>	M	n	y	c <sub>7</sub>	M	n	y
c <sub>8</sub>	M	n	c <sub>8</sub>	M	n	y	c <sub>8</sub>	M	n	y
c <sub>9</sub>	M	n	c <sub>9</sub>	M	n	y	c <sub>9</sub>	M	n	y
c <sub>10</sub>	M	n	c <sub>10</sub>	M	n	y	c <sub>10</sub>	M	n	y
c <sub>11</sub>	M	n	c <sub>11</sub>	M	n	y	c <sub>11</sub>	M	n	y

(a)
(b)
(c)

**Fig. 5.9.** Example of upper bounds obtained on the database  $DB^{ex}$ .

**Theorem 10.** Let  $h = \{\frac{a_1}{n}, \dots, \frac{a_k}{n}\}$  be the frequency histogram of  $DB'$  w.r.t.  $\mathbf{S}$ , let  $\frac{a_p}{n}$  be the frequency of the object  $o$  in  $DB'$ , and let  $m$  denote the integer number  $\lceil \sigma n \rceil$ . Let  $b_0 = 1$ , let  $b_1, \dots, b_k$  denote the integers  $a_1, \dots, a_{p-1}, a_p - 1, a_{p+1}, \dots, a_k$  sorted in non increasing order, that is  $b_1 \geq b_2 \geq \dots \geq b_k$ , and let  $j$  ( $0 \leq j \leq k$ ) be the greatest integer such that  $s = (\sum_{i \leq j} b_i) \leq m$ . Let

$$h^A = \left\{ \frac{1}{m}, \frac{b_1}{m}, \dots, \frac{b_j}{m}, \frac{m-s}{m} \right\}, \quad \text{and} \quad h^B = \left\{ \frac{1}{s+b_{j+1}}, \frac{b_1}{s+b_{j+1}}, \dots, \frac{b_{j+1}}{s+b_{j+1}} \right\},$$

and let  $A = \text{out}^{h^A}(\frac{1}{m})$  and  $B = \text{out}^{h^B}(\frac{1}{s+b_{j+1}})$ . Then

$$\text{out}_{\mathbf{E}', \mathbf{S}'}(o) \leq \max\{A, B\}.$$

the upper bound term appearing on the right of the inequality is called  $\text{eub}_{\mathbf{E}, \mathbf{S}, \sigma}$ .

*Proof.* First of all, note that every histogram  $\bar{h}$  of  $DB_{o[\mathbf{E}']}$  w.r.t.  $\mathbf{S}'$  has the form

$$\bar{h} = \bigcup_{i=1}^k \left\{ \frac{a_{i,1}}{M}, \dots, \frac{a_{i,k_i}}{M} \right\}$$

where  $a_{i,1} + \dots + a_{i,k_i} \leq a_i$ , for each  $i = 1, \dots, k$ , with  $k_p \geq 1$  and  $m \leq M = \sum_i \sum_j a_{i,j}$ .

Let  $\mathbf{E} \subseteq \mathbf{E}^*$  and  $\mathbf{S} \subseteq \mathbf{S}^*$  be the explanation and the property maximizing the outlierness  $\text{out}_{\mathbf{E}', \mathbf{S}'}(o)$ , respectively, and let  $h^*$  be the histogram of  $DB_{o[\mathbf{E}^*]}$  w.r.t.  $\mathbf{S}^*$ . From Theorem 9 it follows that the outlier object in the optimal histogram  $h^*$  must have frequency  $\frac{1}{M}$ , where  $M$  is the number of objects selected by the optimal explanation  $\mathbf{E}^*$ .

Now, assume that exactly  $m$  objects are selected by the explanation  $\mathbf{E}^*$ . By repeatedly applying Lemma 2 and starting from a generic histogram  $\bar{h}$ , it follows that the optimal histogram  $h^A$ , when the number  $m$  of objects is held fixed, is  $\{\frac{1}{m}, \frac{b_1}{m}, \dots, \frac{b_j}{m}, \frac{m-s}{m}\}$ . The histogram  $h^A$  is obtained by selecting the object  $o$  as a singleton, plus the  $j$  most frequent groups of objects, plus exactly  $m - s$  objects of the  $(j + 1)$ -th most frequent group.

Consider now the family of the histograms  $h^c = \{\frac{1}{s+c}, \frac{c}{s+c}, \frac{b_1}{s+c}, \dots, \frac{b_j}{s+c}\}$ , with  $(m-s) \leq c \leq b_{j+1}$ , obtained by selecting some of the remaining objects of the  $(j+1)$ -th most frequent group (note that  $h^{(m-s)} = h^A$ ).

Since  $g(c) = \text{out}^{h^c}(\frac{1}{s+c})$  is convex (see Lemma 3), it is the case that the maximum of  $g(c)$  is in one of the extreme points of its domain, i.e. either in  $c = (m-s)$  (and in this case the maximum is  $A = g(m-s)$ ) or in  $c = b_{j+1}$  (and in this case the maximum is  $B = g(b_{j+1})$ ).

Assume now, by contradiction, that there exists an histogram  $\bar{h}$ , relative to a subset of  $M \geq m$  objects, such that  $\bar{g} = \text{out}^{\bar{h}}(\frac{1}{M})$  is strictly greater than  $\max\{A, B\}$ .

By exploiting Lemma 2 in conjunction with Lemma 3, it can be shown that there exists an histogram  $h^{\bar{c}}$ , belonging to the family  $h^c$  defined above, such that  $\bar{g} \leq g(\bar{c})$ . To obtain  $h^{\bar{c}}$  from  $\bar{h}$ , first, the  $j$  most frequent groups of objects plus  $\bar{c}$  objects of the  $(j+1)$ -th most frequent one are introduced by exploiting the transformation described in Lemma 2, and second, all the other remaining groups of objects, except for the outlier object, are removed by exploiting Lemma 3. Indeed, having the function  $g(c)$  its maximum in  $c = 0$ , Lemma 3 can be interpreted also as follows: if a group of objects, which are not the most frequent, is removed from the database, then the outlierness of  $o$  does not decrease.

To conclude, since  $\bar{g} \leq g(\bar{c}) \leq \max\{A, B\}$ , it has been shown that, given  $DB_{o[\mathbf{E}]}[\mathbf{S}]$ , for each possible database  $DB_{o[\mathbf{E}']}[\mathbf{S}']$  including  $M \geq m$  objects, where  $\mathbf{E}' \supseteq \mathbf{E}$  and  $\mathbf{S}' \supseteq \mathbf{S}$ , the associated histogram  $\bar{h}$  is such that  $\text{out}^{\bar{h}}(\frac{1}{M}) \leq \max\{A, B\}$ . Hence,  $\max\{A, B\}$  is the upper bound  $\text{eub}_{\mathbf{E}, \mathbf{S}, \sigma}(o)$ .

Before leaving the proof, two notable cases of the statement are to be mentioned.

The first case is when the most frequent group of objects of  $DB[\mathbf{E}]$ , call  $\frac{b_k}{n}$  the frequency of these objects, together with the outlier  $o$  are sufficient to reach the frequency threshold  $\sigma$ , i.e. when  $(b_k + 1) \geq m$ . In this case the optimal histogram is always  $h^* = \{\frac{1}{b_k+1}, \frac{b_k}{b_k+1}\}$ .

The second case is when  $m = s$ , i.e. when no object of the  $(j+1)$ -th most frequent group is used to form the histogram  $h^A$ . In this case, by Lemma 3,  $h^* = h^A$  is always the optimal histogram.

The property stated in Theorem 10 tells that the outlierness of the object  $o$  is maximized, w.r.t. any superset  $\mathbf{S}'$  of  $\mathbf{S}$  and any superset  $\mathbf{E}'$  of  $\mathbf{E}$ , when  $\mathbf{S}'$  preserves the actual histogram of frequencies, except for making  $o$  different from any other object, and  $\mathbf{E}'$  is such that  $DB_{o[\mathbf{E}']}$  contains both  $o$  and the most frequent objects w.r.t.  $\mathbf{S}'$ . Nevertheless, if those objects are less than  $\lceil \sigma |DB| \rceil$  in number, then they do not form a  $\sigma$ -explanation. In such a case, the second most frequent objects w.r.t.  $\mathbf{S}'$  are to be taken into account, and so on, till at least  $\lceil \sigma |DB| \rceil$  objects are included in the database  $DB_{o[\mathbf{E}']}$ .

As an example, consider the portion of the database  $DB^{ex}$  depicted in Figure 5.9(b), let  $\mathbf{S} = \{Tyres\}$ , let  $\mathbf{E} = \emptyset$ , let  $o = c_5$ , and let  $\sigma = 0.5$ . The most frequent objects w.r.t.  $\mathbf{S}'$ , which is the optimal superset of  $\mathbf{S}$  in the sense

explained above, are those assuming the value “ $M$ ” on the attribute *Tyres*. Then, the column named  $\mathbf{E}'$  represents the optimal explanation, i.e., that maximizing the outlierness of  $o$ , among all the supersets of  $\mathbf{E}$ . Notice that the frequency  $\text{freq}_{\mathbf{E}'}(o)$  is  $0.64 > \sigma$ , and, hence, no further objects are needed. In this case,  $\text{eub}_{\mathbf{E},\mathbf{S},0.5}(o) = 0.612$ .

Assume, conversely, the threshold  $\sigma$  to be set to 0.8. Then, the best explanation must select at least nine objects. (the portion of data set above is not large enough). The second most frequent group of objects in  $\mathbf{S}'$  is that consisting of the objects having identifiers  $ID = \{c_1, c_2, c_3, c_4\}$ . Since the most frequent objects w.r.t.  $\mathbf{S}'$  together with the object  $o$  are seven in number, then the best explanation is either that selecting also two objects with identifiers in  $ID$ , say  $\mathbf{E}'$ , or that of selecting in addition all the four objects with identifiers in  $ID$ , say  $\mathbf{E}''$ . Since  $A = \text{out}_{\mathbf{E}',\mathbf{S}'}(o) = 0.375$ , while  $B = \text{out}_{\mathbf{E}'',\mathbf{S}'}(o) = 0.0496$ , then the upper bound  $\text{eub}_{\mathbf{E},\mathbf{S},0.8}(o)$  is  $\max\{A, B\} = 0.375$ .

## 5.8 Algorithms

By exploiting the results presented above in Section 5.7, in this paragraph the algorithms for mining the top  $k$  explanation–outlying properties pairs is presented. The algorithms are designed with two main goals in mind. First, they have to compute the optimal solution. Second, in order to make the method as practical as possible, they should have only linear limit space requirements. In particular, the algorithm core is a complete depth-first search enhanced with pruning of unfruitful subspaces on the basis of the upper bound properties described above.

For clarity, the presentation of the algorithms is organized in two sections. First, Section 5.8.1 describes the algorithm *FindOutlyingProperties*, taking care of the special case in which explanations are not considered ( $\sigma = 1$ ). This algorithm then is exploited as a subroutine by the algorithm *FindLocalOutlyingProperties*, described in Section 5.8.2, which conversely solve the general version of the OPD problem (any  $\sigma \in [0, 1]$ ).

### 5.8.1 Global outlying properties

Figure 5.10 shows the algorithm *FindOutlyingProperties*, which computes the top  $k$  *global* outlying properties. The algorithm receives in input the database  $DB$ , the set of attributes  $\mathbf{A}$ , the outlier object  $o$  and an outlierness threshold  $\theta$ , representing the minimum value of outlierness of a property the user is interested in. Moreover, a further parameter  $\mathbf{S}$  is employed, denoting a subset  $\mathbf{S}$  of the attributes  $\mathbf{A}$ , which is initially set to the empty set, and then exploited through recursive calls. The algorithm returns the set  $\mathbf{T}_k$  of the top  $k$  global outlying properties of the database.

The search space of the problem, consisting in the  $2^n$  subsets of the set  $\mathbf{A}$ , is visited by exploiting a *set enumeration tree* [Rymon, 1992]. Enumeration of



```

FindOutlyingProperties( $DB, \mathbf{A}, o, \theta, \mathbf{S}, \mathbf{T}_k$ )
   $out := Outlierness(DB, \mathbf{S}, o)$ ;
  if  $out > \theta$  then  $UpdateTopOutliers(\mathbf{T}_k, \mathbf{S}, out)$ ;
   $\theta^* := MinOutlierness(\mathbf{T}_k)$ ;
   $upperBound := OutliernessUB(DB, \mathbf{S}, o)$ ;
  if  $upperBound > \max\{\theta^*, \theta\}$  then
     $\mathbf{A}_O := SortAttributes(\mathbf{A}, DB, \mathbf{S}, o)$ ;
    for  $a := FirstAttribute(\mathbf{A}_O)$  to  $LastAttribute(\mathbf{A}_O)$  do
       $\mathbf{A} := \mathbf{A} - \{a\}$ ;
       $FindOutlyingProperties(DB, \mathbf{A}, o, \theta, \mathbf{S} \cup \{a\}, \mathbf{T}_k)$ ;

```

**Fig. 5.10.** The algorithm *FindOutliers*.

the sets is dynamic since, at each node of the tree, remaining attributes are reordered, as explained in the following.

Initially  $\mathbf{T}_k$  is set to  $\emptyset$ . First of all, the outlierness  $out$  of  $\mathbf{S}$  in  $o$  w.r.t.  $DB$  is computed and the set  $\mathbf{T}_k$  is updated, provided that  $out$  is above the threshold  $\theta$ . The function *UpdateTopOutliers* inserts  $\mathbf{S}$  in  $\mathbf{T}_k$  only if either  $|\mathbf{T}_k| < k$  or  $out$  is greater than the value associated with the  $k$ th outlying property in  $\mathbf{T}_k$ . In such a case, if  $\mathbf{T}_k$  already contains  $k$  properties then, before inserting  $\mathbf{S}$ , the  $k$ th outlying one is removed from  $\mathbf{T}_k$ .

The function *MinOutlierness* returns the outlierness  $\theta^*$  associated to the  $k$ th outlying property in the set  $\mathbf{T}_k$ . If  $\mathbf{T}_k$  contains less than  $k$  elements, then this value is set equal to  $-\infty$ . An upper bound  $upperBound$  to the outlierness of any superset of  $\mathbf{S}$  is returned by the function *OutliernessUB*, which exploits Theorem 9 in order to compute the bound.

If  $upperBound$  is greater than the maximum between  $\theta^*$  and  $\theta$ , then the algorithm continues exploring the subsets of  $\mathbf{A}$  containing  $\mathbf{S}$  (in the enumeration tree); otherwise the search space is pruned since the children of the current node are associated to properties whose outlierness cannot be better than those of the current top- $k$  properties. It is worth noting that the larger the bound given by the maximum between  $\theta^*$  and  $\theta$  is, the more the property stated in Theorem 9 shall prove itself effective in pruning the search space. Such a bound is initially set equal to the value  $\theta$ , and begins to increase, and to be equal to  $\theta^*$ , only after that the algorithm has detected at least  $k$  properties with an outlierness value greater than  $\theta$ .

As anticipated above, before starting the visit of the subtree rooted in the current node, its children are ordered. Ordering is done in decreasing order of outlierness, that is, let  $a_i, a_{i+1} \in \mathbf{A}$ , then  $a_i$  precedes  $a_{i+1}$  if  $out_{\mathbf{S} \cup \{a_i\}}(o) \geq out_{\mathbf{S} \cup \{a_{i+1}\}}(o)$ . This strategy allows us to visit the most promising subsets of attributes earlier in order to hopefully accelerate the convergence of the threshold  $\theta^*$  towards the outlierness of the  $k$ th top global outlier, thus improving the pruning power of the method.

```

FindLocalOutlyingProperties( $DB, \mathbf{A}, o, \sigma, \theta, e, e_A, \mathbf{T}_k$ )
  FindOutlyingProperties( $DB_{o[e]}, \mathbf{A}, o, \theta, \emptyset, \mathbf{T}_k$ );
   $\mathbf{u} := \text{OutliernessWithExplanationUB}(DB, \mathbf{A}, o)$ ;
   $upperBound := \max\{\mathbf{u}_a \mid a \in \mathbf{A}\}$ ;
   $\theta^* := \text{MinOutlierness}(\mathbf{T}_k)$ ;
  if  $upperBound > \max\{\theta^*, \theta\}$  then
     $e_O := \text{SortAttributes}(e_A, DB, e, o)$ ;
    for  $a := \text{FirstAttribute}(e_O)$  to  $\text{LastAttribute}(e_O)$  do
      if  $\text{freq}_{e \cup \{a\}}(o) \geq \sigma$  then
         $e_A := e_A - \{a\}$ ;
         $\theta^* := \text{MinOutlierness}(\mathbf{T}_k)$ ;
         $\mathbf{A}' := \{a' \in \mathbf{A} \mid \mathbf{u}_{a'} > \max\{\theta^*, \theta\}\}$ ;
        if  $\mathbf{A}' - \{a\} \neq \emptyset$  then
          FindLocalOutlyingProperties( $DB, \mathbf{A}' - \{a\}, o, \sigma, \theta, e \cup \{a\}, e_A, \mathbf{T}_k$ );

```

**Fig. 5.11.** The algorithm *FindLocalOutliers*.

Completeness of the method is guaranteed by the use of a set enumeration tree in conjunction with the property enunciated in Theorem 9. Furthermore, since the outliers stored in  $\mathbf{T}_k$  are exactly those scoring the largest outlierness among the set of attributes visited during the search, the following result holds.

**Theorem 11.** *Algorithm FindOutlyingProperties computes the top  $k$  global outlying properties of the test object w.r.t. the input database.*

*Proof.* Without the pruning rule, the algorithm *FindOutlyingProperties* exhaustively visits the whole search space including all the subsets of the attribute set  $\mathbf{A}$  by exploiting a set enumeration tree. During the visit, it maintains the top  $k$  outlying properties in  $\mathbf{T}_k$ . By Theorem 9, the value *upperBound* returned by the function *OutliernessUB* on the set  $\mathbf{S}$ , is an upper bound to the outlierness of any superset of  $\mathbf{S}$ . Hence, if *upperBound* is not greater than the outlierness of the best  $k$ -th outlying property encountered till now, then the subtree rooted at  $\mathbf{S}$  can be pruned without affecting completeness. This closes the proof.

### 5.8.2 Local Outlying Properties

Figure 5.11 shows the algorithm *FindLocalOutlyingProperties*. The input parameters of the method are: the database  $DB$ , its set of attributes  $\mathbf{A}$ , the test object  $o$ , the frequency threshold  $\sigma$ , the outlierness threshold  $\theta$ , the explanation set  $e$ , and the set  $e_A$  of attributes that can be used to augment the current explanation  $e$ . Initially, the parameter  $e$  is set to  $\emptyset$ , whereas  $e_A$  is set to  $\mathbf{A}$ . The output of the algorithm is the set  $\mathbf{T}_k$  containing the top  $k$  explanation–outlying properties  $(\sigma, \theta)$ –pairs for the input problem.

First of all, the algorithm calls the method *FindOutlyingProperties* in order to update the set  $\mathbf{T}_k$  with the global outlying properties of  $o$  w.r.t. the subset  $DB_{o[e]}$  of the database  $DB$ .

Then the function *OutliernessWithExplanationUB* returns the vector  $\mathbf{u}$  including  $|\mathbf{A}|$  rational numbers. Each element  $\mathbf{u}_a$  of  $\mathbf{u}$  is associated with a distinct attribute  $a$  of  $\mathbf{A}$ , and represents an upper bound to the outlierness of any explanation–property pair  $(e', \mathbf{S}')$  such that  $e \subseteq e'$  and  $a \in \mathbf{S}'$ . These upper bounds are computed as explained in Theorem 10. The maximum upper bound included in  $\mathbf{u}$  is stored in the variable *upperBound*.

If *upperBound* is greater than the maximum between the threshold  $\theta$  and the outlierness of the current  $k$ th top outlying property, then the search proceeds on the supersets of  $e$ . Also in this case, explanation sets are visited by exploiting a set enumeration tree. The attributes in the set  $\mathbf{E}_A$  to be added to the current set  $\mathbf{E}$  are ordered by increasing frequency, that is, let  $a_i, a_{i+1} \in \mathbf{E}_A$ , then  $a_i$  precedes  $a_{i+1}$  if  $\text{freq}_{\mathbf{E} \cup \{a_i\}}(o) \geq \text{freq}_{\mathbf{E} \cup \{a_{i+1}\}}(o)$ , and then visited according to this ordering.

Before visiting the children of the current node, the set  $\mathbf{A}'$ , containing the attributes that are candidates to form a top  $k$  outlying property, is determined. This set includes the attributes  $a'$  of  $\mathbf{A}$  such that the upper bound  $\mathbf{u}_{a'}$  referred to above is greater than the outlierness of the  $k$ th outlying property. Note that, after visiting a subtree, the solution set  $\mathbf{T}_k$  is updated and, hence, the set  $\mathbf{A}'$  may get smaller while visiting the children of the current explanation. If  $\mathbf{A}'$  becomes empty, then this corresponds to pruning a portion of the search space, since for the subtrees rooted at the remaining attributes of  $e_A$ , the attributes employable as property have an upper bound to their outlierness that is smaller than the outlierness of  $k$ -th current top outlying property.

Completeness of the method is guaranteed by the use of a set enumeration tree in conjunction with properties enunciated in the previous section. Furthermore, since the outliers stored in  $\mathbf{T}_k$  are exactly those scoring the largest outlierness among the set of attributes visited during the search, the following result can be derived.

**Theorem 12.** *Algorithm *FindLocalOutlyingProperties* computes the top  $k$  local outlying properties of the test object w.r.t. the input database together with the associated explanations.*

*Proof.* Without the pruning rule, the algorithm *FindLocalOutlyingProperties* exhaustively visits the whole search space including all the pairs of disjoint subsets of the attribute set  $\mathbf{A}$ , by exploiting two nested set enumeration trees. The outer set enumeration tree concerns explanations and is directly visited by the procedure *FindLocalOutlyingProperties*, while the inner set enumeration tree concerns outlying properties and is visited by the procedure *FindOutlyingProperties* described in previous section. In particular, each node of the outer tree is associated with an explanation  $\mathbf{E}$ , while each node of the inner tree is associated with an outlying property  $\mathbf{S}$  which has  $\mathbf{E}$  as explana-

tion. During the visit the algorithm maintains the top  $k$  explanation-outlying property pairs in  $\mathbf{T}_k$ .

Consider the set  $\{\mathbf{u}_a \mid a \in \mathbf{A}\}$  returned by the function *OutliernessWithExplanationUB* on the set  $\mathbf{A}$ . By Theorem 10, for each  $a \in \mathbf{A}$ ,  $\mathbf{u}_a$  is an upper bound to the outlierness for any property  $\mathbf{S}$  having as explanation a superset of  $\mathbf{E}$  and such that  $a \in \mathbf{S}$ . The value  $\max\{\mathbf{u}_a \mid a \in \mathbf{A}\}$  is an upper bound to the outlierness for any property  $\mathbf{S}$  having as explanation a superset of  $\mathbf{E}$ . Hence, if *upperBound* is not greater than the outlierness of the best  $k$ th explanation-outlying property pair encountered till now, then the subtree in the outer set enumeration tree rooted at  $\mathbf{E}$  can be pruned without affecting completeness.

Note that, the inner tree rooted at  $\mathbf{E}$  does not include in its visit the attributes  $a'$  whose associated upper bound  $\mathbf{u}_{a'}$  is less than the outlierness associated with the current top  $k$ th pair. This pruning rule does not affect completeness, since it cannot exist a top pair  $(\mathbf{E}', \mathbf{S}')$  with  $\mathbf{E}' \supseteq \mathbf{E}$  and  $a' \in \mathbf{S}'$ . This closes the proof.

## 5.9 Algorithm implementation details, time and spatial cost

In this paragraph implementation details of the algorithms are provided, and their concrete time and space requirements are discussed. In the following,  $n$  denotes the size of  $DB$  and  $m$  denotes the size of  $\mathbf{A}$ . The section is organized as follows. First, the data structures employed by the algorithms are described. Then, both their temporal and spatial costs are discussed.

### 5.9.1 Data structures

First, the data structures, exploited in implementing the algorithms, are described.

#### Database

The input database is preprocessed as to encode each distinct attribute value in the same domain with a distinct integer number. The transformed database is then stored in an ad-hoc data structure, called  $db$  in the following, which serves the purpose of efficiently computing both the outlierness of a generic object and any projection of the database. The structure  $db$  consists of  $m$  arrays  $db_a$ , each associated with a distinct attribute  $a$  of  $\mathbf{A}$ . Every entry  $db_{a,v}$  of the array  $db_a$  is associated with a distinct value  $v$  in the active domain of the attribute  $a$  and it stores the identifiers, sorted in increasing order, of the objects of  $DB$  assuming the value  $v$  on the attribute  $a$ .

Figure 5.12 shows an example of the data structure  $db$ .

Using the structure  $db$ , the projections of the database can be computed efficiently. Indeed, assume that the identifiers of the objects in  $DB_{o[\mathbf{E}]}$  are available; then, the identifiers of the objects in  $DB_{o[\mathbf{E} \cup \{a\}]}$  can be obtained by intersecting the former set of identifiers with the identifiers of the objects of  $DB$  assuming the value  $o[a]$  on the attribute  $a$ . The latter identifiers are immediately available in the entry  $db_{a,o[a]}$  of  $db$ . Since both set of identifiers are ordered, the cost of the above operation is linear in the size of the longest of the two lists, that is  $\mathcal{O}(n)$ . Subsequent Section 5.9.3 will provide details on how the list of the identifiers associated with the objects of the database  $DB_{o[\mathbf{E}]}$  is maintained.

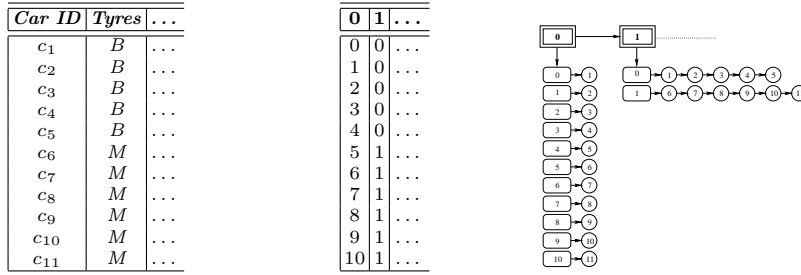


Fig. 5.12. Example of data structure  $db$ .

**Frequency histogram**

The frequency histogram of  $DB$  w.r.t.  $\mathbf{S}$  is represented as an array of object identifiers. The identifiers are sorted w.r.t. the values they assume on the set of attributes  $\mathbf{S}$ . Thus, the array can be viewed as partitioned in blocks of adjacent object identifiers. Each block is associated with a distinct value of  $\mathbf{S}$ . It is important to note that, within each single block, it is not required that object identifiers are sorted. However, blocks starting indexes are stored in an auxiliary array, called *index*.

This kind of representation has the advantage that if the identifiers of the objects of  $DB$  are sorted w.r.t. the values they assume on the attributes  $\mathbf{S} \cup \{a\}$ , then they are also sorted w.r.t. the values they assume on the attributes  $\mathbf{S}$ .

Thus, while visiting the set enumeration tree associated with outlying properties, only a single frequency histogram has to be stored, since saving

the *index* list in the father node is sufficient to recover its frequency histogram when returning from visiting the child node.

Furthermore, it is interesting to point out that the *index* list associated with each node of the current branch of the properties tree can be obtained from the *index* list associated with the leaf of the current branch by simply augmenting each entry of *index* with a positive integer number, say *level*. This number represents the smallest level of the tree in which the entry has been introduced. Indeed, the *index* list associated with a generic node of the current branch includes precisely the entries of the *index* list associated with the leaf node having a value for the field *level* which is less than or equal to the current level of the node.

Thus, while visiting the properties tree, only one single *index* list has to be stored and maintained.

Figure 5.13(b) shows two examples of a histogram and an *index* list (the first item of each entry denotes the first element in the histogram associated with a block of identical objects, while the second entry is the field *level*) obtained on the portion of database of Figure 5.13(a). On the left of Figure 5.13(b) the histogram (array) of the attribute set  $S_1 = \{Pit-Stops\}$  together with the associated *index* list are reported. On the right it is shown the histogram of the attribute set  $S_2 = \{Pit-Stops, Cylinders\}$ . Note that both the histogram and the *index* list on the left can be obtained from the histogram and the *index* list on the right by simply removing the entries of *index* having *level* > 1. Indeed, the only difference between the two histogram arrays is that the identifiers of the objects belonging to the same group are reordered, a condition which, as already stated (see the top of this paragraph), leaves unchanged the histogram represented by the array and its associated *index* list.

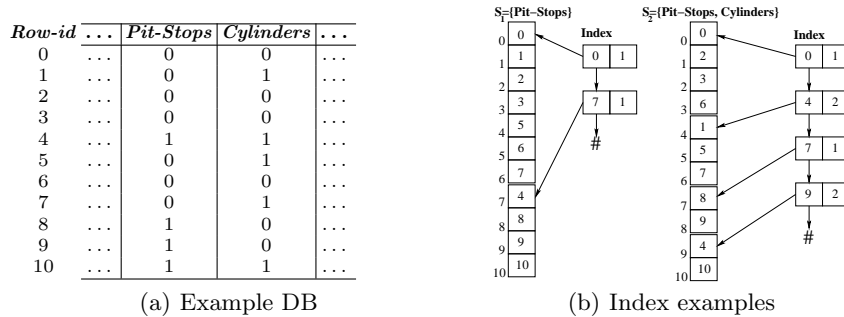


Fig. 5.13. Index structure

Building an histogram of frequencies can be performed in time linear in  $n$ , since it requires only a single scan of an entry of  $db$ , having size  $n$ . Indeed,

if  $\mathbf{S} = \{a\}$  is a singleton set, then the frequency histogram is the intersection of the entries of the array  $db_a$  of  $db$  with the list of identifiers of the objects of  $DB_{o[\mathbf{E}]}$ . Furthermore, if the frequency histogram of  $\mathbf{S}$  is available, then the frequency histogram of  $\mathbf{S} \cup \{a\}$  can be obtained with a single scan of  $db_a$ , as follows. For each value  $v$  in the active domain of  $a$ , the entry  $db_{a,v}$  of  $db_a$  is visited: *each set of identifiers stored in the entry and belonging to the same block in the old frequency histogram yields a novel block in the new frequency histogram*. The rest of this subsection takes care of describing how this operation is accomplished in our implementation.

*Frequency histogram update.* Consider an array that stores, for each object in the database, the block it belongs to in the current frequency histogram (see Figure 5.14; each block is identified by a distinct natural number), or a special value if the object does not belong to the current database  $DB_{o[\mathbf{E}]}$ . In the following this array shall be denoted by  $B$ . Consider, now, a second array, whose size is equal to the number of these blocks. In the following this array will be denoted by  $LB$ .

The new frequency histogram is built by scanning the entry  $db_a$  of  $db$ . Remember that the entries of  $db_a$  are partitioned into blocks. Assume a generic block  $b^*$  of  $db_a$  is being scanned and let  $ID^*$  be the current identifier in  $db_{a,b^*}$ :  $B[ID^*]$  is the block identifier of  $ID^*$  in the original frequency histogram, and  $b^*$  is the block identifier for  $ID^*$  in  $db_a$ .

For each block  $b$  of the original frequency histogram,  $LB[b]$  stores the block identifier in  $db_a$  which the last object in  $b$  already inserted in the new frequency histogram belongs to.

In the new frequency histogram, each block of the original frequency histogram may be kept as it is, or split in two or more blocks. To this end, the values stored in the array  $LB$  are exploited. In particular, if the last object  $ID$  already inserted in the new frequency histogram and belonging to the block  $B[ID^*]$  belongs to the same block  $b^*$  in  $db_a$  as the object  $ID^*$  currently under examination, then that  $ID^*$  assumes the same value of  $ID$  in  $\mathbf{S} \cup \{a\}$ , and thus they must belong to the same block in the new frequency histogram. Otherwise a new block is created.

Summarizing, if  $LB[B[ID^*]] = b^*$  then  $ID^*$  is inserted in the new frequency histogram without creating a new block; otherwise, a new block is created (that is, a new item is added to the *index* list). In any case, the value  $LB[B[ID^*]]$  is updated to  $b^*$ .

*Example 6.* Consider Figure 5.14. The frequency histogram  $h$  associated with  $\mathbf{S} = \{Pit-Stops\}$  consists of two blocks, with identifiers 0 and 1, respectively. Hence, the array  $LB$  will consist of two entries. The array  $B$  stores, for each object, the block in  $h$  which that object belongs to. The frequency histogram  $h'$  associated with  $\mathbf{S}' = \mathbf{S} \cup \{Engine-Failures\}$  has to be built. A block starting from 0 and a block starting from 7 will certainly occur in  $h'$ , since these are the starting points of the two blocks of  $h$ . All the objects in the first

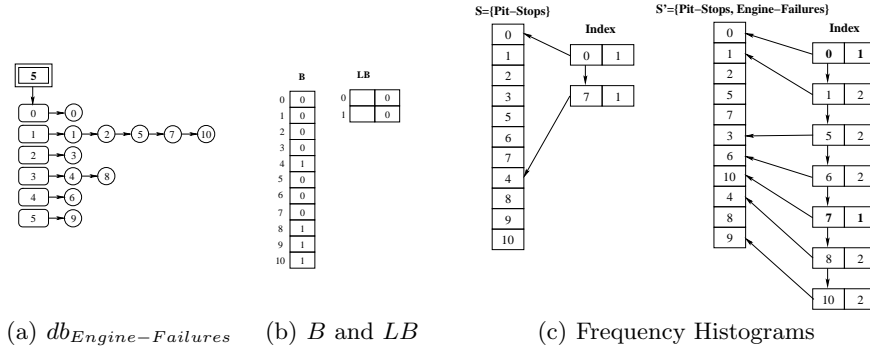


Fig. 5.14. Example of frequency histogram management.

block of  $h$  will take in  $h'$  a position in the interval  $[0 \dots 6]$ , whereas all other objects in  $h'$  will take a position in the interval  $[7 \dots 10]$ .

In order to build the new frequency histogram  $h'$ , the structure  $db_{Engine-Failures}$  reported in Figure 5.14(a) is analyzed. The first object has identifier 0 and its block in  $h$  is 0 ( $B[0] = 0$ ).

Each entry of  $LB$  is built as composed by two fields. The first is used to store a block identifier, as explained above, while the second is used to maintain the number of objects of the block of the original frequency histogram associated with entries from  $LB$  that have been already examined.

Since the first entry of  $LB[0]$  is empty, 0 is the first object encountered in the block 0 of  $h$ . Therefore, 0 is added in  $h'$  as the first object in the interval  $[0 \dots 6]$  (the position in which to insert 0 is given by the second field of  $LB[0]$ ), the first field of the entry  $LB[0]$  is set to 0, and the second field is increased by one.

The second object encountered while scanning  $db_{Engine-Failures}$  is 1, its block in  $h$  is 0. The first field of  $LB[0]$  is equal to 0, while the block in  $db_{Engine-Failures}$  of 1 is 1. Then, 1 is put in  $h'$  in the interval  $[0 \dots 6]$  (since  $B[1] = 0$ ), but a new block must be created (the entry 1 is added to  $Index$ ), and the first field of the entry  $LB[0]$  will thus be set to 1 (notice that Figure 5.14 just reports the initial state of the arrays).

Continuing with the analysis, the third object encountered is 2, its block in  $h$  is 0, the first field of  $LB[0]$  is 1, and also the block in  $db_{Engine-Failures}$  of 2 is 1. Then 2 is inserted in  $h'$  without creating a new block.

The objects with identifiers 5 and 7 are managed analogously as object having identifier 2. As for the object 10, since the first field of  $LB[B[10]]$  is empty, it is inserted in  $h'$  as first object in the interval  $[7 \dots 10]$ .

The remaining objects are handled analogously. Figure 5.14(c) (on the right) reports the final frequency histogram obtained.

Concluding, by adopting the above described strategy, the frequency histogram of  $DB[S \cup \{a\}]$  can be obtained from the frequency histogram of  $DB[S]$



by performing a single sequential scan of the entry  $db_a$ . Thus, the temporal cost of this operation is  $\mathcal{O}(n)$ , linear in the number of objects of the database  $DB$ . Also, the spatial cost of this operation is linear, since the auxiliary data structures  $B$  and  $LB$  are arrays of size at most  $n$ .

### 5.9.2 Temporal cost

Given a frequency histogram, the cost of computing the outlierness is linear in the size of the histogram (see Theorem 2), that is,  $\mathcal{O}(n)$ . The same cost must be spent also for computing the upper bound  $\text{oub}_{\mathbf{E},\mathbf{S}}(o)$  as stated in Theorem 9. The upper bound  $\text{eub}_{\mathbf{E},\mathbf{S},\sigma}(o)$ , as stated in Theorem 10 can be obtained after sorting the elements belonging to the frequency histogram. Since in this case the range of values to be sorted is known in advance, this operation can be performed in linear time and space by using the counting sort algorithm [Knuth, 1973] or similar algorithms.

### Find Outlying Properties

Next, the cost of visiting a node of the tree explored by the algorithm *FindOutlyingProperties* shall be stated. Computing the outlierness and the upper bound  $\text{oub}_{\mathbf{E},\mathbf{S}}(o)$  is feasible in time  $\mathcal{O}(n)$ , as seen above. Updating the set  $\mathbf{T}_k$  can be done in time  $\mathcal{O}(\log k)$ . The procedure *SortAttributes* consists in computing the outliernesses  $\text{out}_{\mathbf{S} \cup \{a\}}(o)$ , for each  $a \in \mathbf{A}$ , and then sorting them. Thus, its cost is upper bounded by  $\mathcal{O}(mn + m \log m)$ . Notice that attributes whose  $\text{oub}_{\mathbf{E},\mathbf{S}}(o)$  is lower than the threshold  $\theta$  are not further considered while visiting the subtree rooted at the current node.

Summarizing, the cost paid in *FindOutlyingProperties* for visiting each node is  $\mathcal{O}(nm + m \log m + \log k)$ , where  $m$  decreases with the level of the node in the tree. Notice that the number of nodes of the tree is  $2^m$ . The  $m$  nodes of the first level of the property tree require  $\mathcal{O}(nm + m \log m + \log k) \leq \mathcal{O}(nm)$  time, assuming that  $nm \geq m \log m$  and  $nm \geq \log k$ .

### Find Local Outlying Properties

As far as the *FindLocalOutlyingProperties* algorithm is concerned, the cost of visiting a single node is determined as follows (in this cost analysis the procedure *FindOutlyingProperties* shall not be taken in account). The cost of computing  $DB_{o[\mathbf{E}]}$  is  $\mathcal{O}(n)$  (see above Section 5.9.1). The vector  $\mathbf{u}$  of upper bounds, returned by the procedure *OutliernessWithExplanationUB*, is computed by exploiting the property stated in Theorem 10, for each  $\mathbf{S} \cup \{a\}$ . Since, in our implementation, these values, together with their maximum, are computed by the procedure *FindOutlyingProperties* when visiting the first level of the outlying properties tree, then this procedure can be assumed to have no cost. The procedure *SortAttributes* computes the databases  $DB_{o[\mathbf{E} \cup \{a\}]}$ ,

for each  $a \in e_A$ , and then sorts the attributes  $a$  in order to decrease the size of the database. Its cost is thus  $\mathcal{O}(mn + m \log m)$ . Thus, the cost of visiting a node of the explanations tree is  $\mathcal{O}(n + nm + m \log m) \leq \mathcal{O}(nm)$ , where  $n$  decreases with the level of the node in the tree, and can be reduced to  $\mathcal{O}(n)$  if a static ordering of the attributes is employed.

Summarizing, if  $N$  is the total number of nodes explored by the algorithm *FindLocalOutlyingProperties*, both in the explanation and in the property trees, then the overall cost of the method is  $\mathcal{O}(Nnm)$ .<sup>2</sup> Notice, conversely, that the total size of the search space, i.e. the number of possible explanation-outlier pairs, is much larger, being equal to  $N_{\max} = \sum_{i=0}^m \binom{m}{i} 2^{m-i}$ . Hence, the efficiency of the algorithm relies in its ability of keeping  $N$  much smaller than  $N_{\max}$  by effectively pruning the search space.

### 5.9.3 Spatial cost

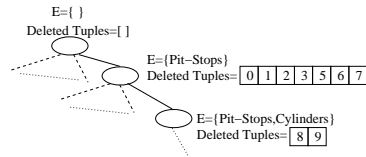
Next, the algorithm is shown to have linear space complexity w.r.t. the dimension  $\mathcal{O}(mn)$  of the input. The number  $k$  of top pairs to find is assumed to be fixed, so that the space required to store the set  $\mathbf{T}_k$  is  $\mathcal{O}(m)$ .

The input database  $DB$  is stored using the structure  $db$  that includes  $nm$  integers.

Since the adopted strategy is of the depth-first kind, the total required space is the sum of that occupied by the nodes along a branch of the explanation and property trees. While visiting a branch of the properties tree, whose depth is upper bounded by  $m$ , each node has to store the ordering of the  $m$  attributes w.r.t. their outlierness. Then, a total  $\mathcal{O}(m^2)$  spatial cost is implied along the current branch of the tree. Furthermore, *FindOutlyingProperties* stores a single *index* structure, whose size is upper bounded by  $n$ .

Row-id	...	Pit-Stops	Cylinders	...
	...	<b>3</b>	<b>4</b>	...
0	...	0	0	...
1	...	0	1	...
2	...	0	0	...
3	...	0	0	...
4	...	<b>1</b>	<b>1</b>	...
5	...	0	1	...
6	...	0	0	...
7	...	0	1	...
8	...	1	0	...
9	...	1	0	...
10	...	1	1	...

(a) Example database



(b) Explanation tree

**Fig. 5.15.** Explanation tree example.

<sup>2</sup> It is worth to point out that the *actual* cost to be paid in each node  $(\mathbf{E}', \mathbf{S}')$  is  $\mathcal{O}(n_0 m_0)$ , where  $n_0$  is the size of  $DB[\mathbf{E}']$ , and  $m_0$  is the size of  $\mathbf{S}'$ .

While visiting a branch of the explanation tree, the algorithm *Find-LocalOutlyingProperties* should store, for each node, the identifiers of the database objects selected by the explanation as associated with that node. Nonetheless, a more efficient strategy can be pursued. Indeed, a unique global array can be used to maintain the identifiers of the objects selected by the current explanation. Identifiers of the objects of  $DB$  not included in this global array are stored in the nodes of the current branch. Specifically, each node stores the identifiers of the objects selected by the explanation associated with its father but not selected by the explanation associated with it. Thus, maintaining the identifiers selected by the current explanation along a branch of the tree requires space  $\mathcal{O}(n)$ .

Figure 5.15 shows an example of the strategy adopted to store the identifiers associated with the current database  $DB_{o[\mathbf{E}]}$ . The object  $o$  is that having Row-id 4. The array "Deleted Tuples" associated with each node  $\mathbf{E} \cup \{a\}$  of the explanation tree stores the identifiers of the objects which are in  $DB_{o[\mathbf{E}]}$  but not in  $DB_{o[\mathbf{E} \cup \{a\}]}$ .

Furthermore, each node of the explanation tree has to store the upper bounds  $\mathbf{u}$  of the objects in  $\mathbf{A}$  together with the ordering of the attributes in  $\mathbf{E}_A$ . Notice that  $\mathbf{A}$  and  $\mathbf{E}_A$  are disjoint, and this requires space  $\mathcal{O}(m)$ . Since the explanations tree has depth at most  $m$ , the total space required along a branch of the tree is  $\mathcal{O}(m^2)$ .

Summarizing, the algorithm requires space  $\mathcal{O}(mn)$  to maintain the database, space  $\mathcal{O}(n + m^2)$  to visit the explanation tree and space  $\mathcal{O}(n + m^2)$  to visit the property tree. Assuming, as is reasonable, that  $m \ll n$ , then the cost simplifies to  $\mathcal{O}(mn)$ , which is linear in the input size.

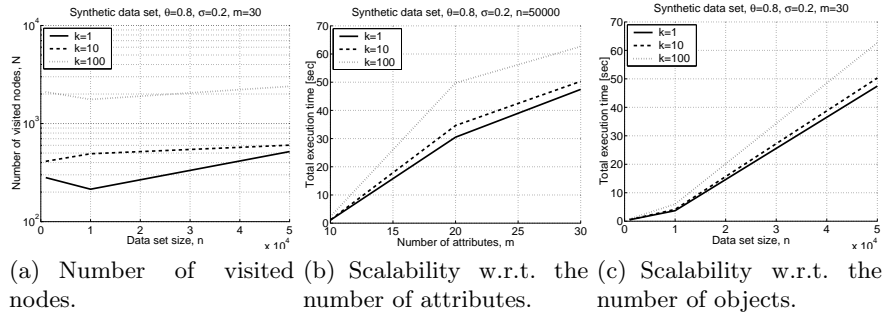
## 5.10 Experimental results

In this paragraph, results of experiments conducted by using proposed algorithms are presented.

Experiments are organized as follows. First of all, the scalability of the method is studied (Section 5.10.1). Then, the sensitivity to the parameters is investigated (Section 5.10.2). Examples of knowledge mined by the algorithms are illustrated in Section 5.10.3. Finally, the behavior of the method on random data is investigated to verify that it does not report supposedly "interesting" knowledge when no meaningful information actually occurs in the input data collection (Section 5.10.4).

### 5.10.1 Scalability

In order to study the scalability of the method, a family of synthetically generated data sets is considered. Each data set of the family consists of  $n$  objects having  $m$  attributes. Attribute values are obtained as follows. Initially,  $n-1$  values are generated according to a normal distribution and sorted. Then,



**Fig. 5.16.** Experimental results on the synthetic data set family.

these values are grouped into ten equally spaced bins and replaced with the center of the bin they belong to.

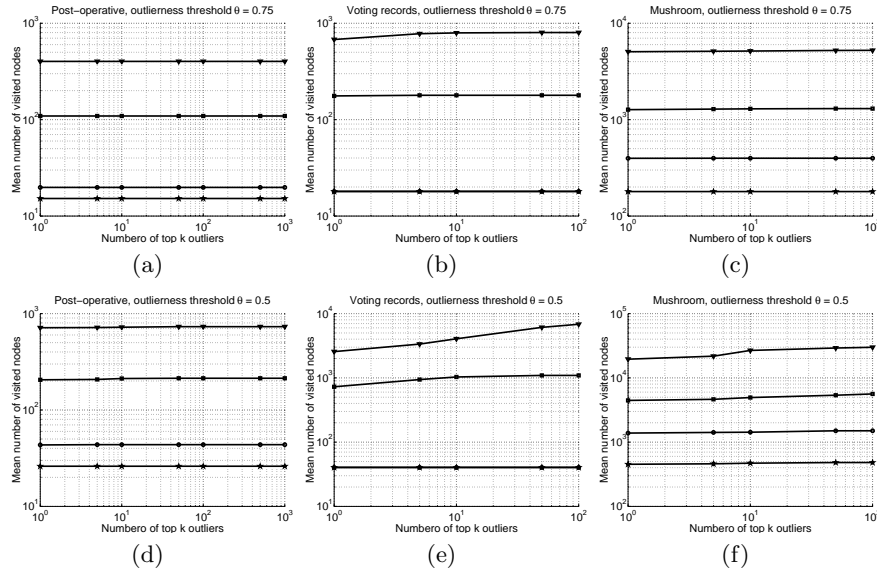
Finally, a single outlier object  $o$ , that is going to be used as input in the experiments, is generated as follows: 50% of its attribute values are set to the center of the most populated bin (call  $\mathcal{E}$  these attribute set), 25% of them are set to the center of the less populated bin (call  $\mathcal{S}$  these attribute set), and the remaining 25% are set according to randomly chosen bin centers. By construction, the attributes in  $\mathcal{S}$  represent exceptional properties for the object  $o$ , while the attributes in  $\mathcal{E}$  select a large portion of the data set and, then, denote meaningful explanations.

In all experiments the parameter  $\sigma$  was set to 0.2 and the parameter  $\theta$  to 0.8. The outlierness associated with outlying properties for  $o$  was always greater than 0.98. Figure 5.16 reports the experimental results on this family of data sets.

Figure 5.16(a) shows the number of nodes visited by the algorithm *Find-LocalOutlyingProperties* w.r.t. the data set size when the number of attributes is held fixed to thirty. The number of analyzed nodes remained almost constant with the size of the data set. This can be explained since the number of attributes is constant over all the executions. Moreover, the experiment clarifies that pruning rules are almost insensitive to the data set size whenever the underlying distribution does not change.

Figure 5.16(b) shows the total execution time of the algorithm w.r.t. the number of attributes, for increasing values of  $k$ . As expected, the execution time increases with the number of attributes, but, interestingly, its growth is slow, confirming the effectiveness of the pruning rules.

Finally, Figure 5.16(c) shows the total execution time w.r.t. the data set size  $n$ , for increasing values of  $k$ . Since, as discussed above, the number of visited nodes is almost constant with  $n$ , and since  $m$  is held fixed, as expected by results about the temporal cost stated in Section 5.9.2, the total execution time increases almost linearly with  $n$ .



**Fig. 5.17.** Mean number of node visited (starting from the top of each figure the curves are for  $\sigma = 0.25, 0.5, 0.75$ , and  $1.0$ ). Notice that, on the Voting records database, curves for  $\sigma = 0.75$  and  $\sigma = 1.0$  overlap.

### 5.10.2 Sensitivity to parameters

In these experiments, three real-life data sets, namely the *Post-operative*, *Voting records*, and *Mushroom* data sets [Newman et al., 1998], whose characteristics are briefly described next, are considered. The *Post-operative* data set has 9 categorical attributes and includes 90 objects, the *Voting records* data set consists in 435 objects each having 17 binary attributes, while the *Mushroom* data set has 22 attributes, all categorical, and includes 8,124 objects. The total number of explanation–outlier pairs is 19,683 for the *Post-operative* data set, about 129 millions for the *Voting records* data set, and about 31 thousand millions for the *Mushroom* data set.

For different values of  $k$ , the number of nodes visited by the algorithm *FindLocalOutlyingProperties* on a random sample containing the 1% of the database objects is measured, and different curves, each associated to a different combination of the parameters  $\theta$  (0.5 and 0.75) and  $\sigma$  (0.25, 0.5, 0.75, and 1.0) are obtained. Figure 5.17 reports the mean number of visited nodes.

It is worth to note that, for many combinations of the parameters  $\theta$  and  $\sigma$ , the curves are insensitive to the parameter  $k$ . Furthermore, the pruning power of the method improves by increasing the parameters  $\theta$  or  $\sigma$ . Moreover, it must be pointed out that the number of visited nodes always represents a negligible fraction of the overall search space.

As for the execution times, Figure 5.18 reports a table showing the processing time per node. Interestingly, it clearly results from this table that the time spent to process a single node gets smaller while  $\sigma$  is decreasing. Since, by decreasing  $\sigma$ , the number of visited nodes increases, the above behavior has the desirable effect of mitigating the increase of the overall execution time.

### 5.10.3 About mined knowledge

In the sequel the meaningfulness of the knowledge mined using our technique is discussed by illustrating some exceptional properties found.

#### Mushroom data

This data set, having 22 categorical attributes and including 8,124 objects, includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.

The algorithm *FindLocalOutlyingProperty*, when run on this data set for the object of row 4,977, returned an explanation *habitat*=“grasses” for the abnormal property consisting of the attributes *stalk-color-above-ring*=“buff”, *stalk-color-below-ring*=“buff”. Indeed, among the 2,148 objects selected by the explanation, 1,716 assume value “white” on both the two attributes, while only 48 are such that the two attributes are “buff”.

#### Post-operative data

This database, having 9 categorical attributes and including 90 objects, has been used in order to determine where patients in a postoperative recovery area should be sent to next. Because hypothermia is a significant concern after surgery, the attributes correspond roughly to body temperature measurements, except for the attribute *COMF*, representing patient’s perceived comfort at discharge, and *DECS*, representing discharge decision, that is “I” (patient sent to Intensive Care Unit), “S” (patient prepared to go home), and “A” (patient sent to general hospital floor). The database is shown below after the description of the experiments.

The top explanation–outlier pair returned by the algorithm *FindLocalOutlyingProperty* for the object of row 59 has as explanation the attributes *L-BP*=“high”, *SURF-STBL*=“stable”, and *BP-STBL*=“stable”, and as property the attribute *L-CORE*=“low”, where *L-BP* represents the last measurement of blood pressure, *SURF-STBL* represents the stability of patient’s surface temperature, *BP-STBL* represents the stability of patient’s blood pressure, and *L-CORE* represents the patient’s internal temperature. The portion of database selected by this explanation includes the 10% of the objects. On this portion, all the objects assume the value *L-CORE*=“mid” except for the test object, which conversely assumes the value *low*.

Threshold $\theta$	0.5				0.75			
Threshold $\sigma$	0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00
<i>Post-operative</i>	0.191	0.349	0.662	0.687	0.260	0.479	0.965	0.892
<i>Voting records</i>	0.223	0.359	0.918	0.851	0.299	0.572	1.350	1.336
<i>Mushroom</i>	2.172	3.210	3.976	4.131	1.913	3.025	3.728	3.718

**Fig. 5.18.** Mean execution times per node (milliseconds).

On the object of row 4 the algorithm singled out the property *CORE-STBL* with explanation *SURF-STBL*=“*stable*”. The attribute *CORE-STBL* denotes stability of patient’s core temperature. Indeed, among the forty-five objects (50% of the database) selected by this explanation, only this object is such that *CORE-STBL*=“*unstable*”, while all the others assume the value “*stable*”.

Finally, consider the object of row 15. It resulted in singling out an “*unstable*” core temperature (attribute *CORE-STBL*) as outlying property with explanation *L-O2*=“*good*” and *BP-STBL*=“*stable*”. Indeed, among all the patients having a “*good*” oxygen saturation (attribute *L-O2*) and a “*stable*” blood pressure (attribute *BP-STBL*), the object of row 15 is the only one having unstable core temperature, whereas all the other have a stable core temperature.

The following table shows the *Post-Operative* database. For the sake of readability the database objects were reordered, so that the object of row 15 is the first one, and the 25 objects selected by the explanation (*L-O2*=“*good*”, *BP-STBL*=“*stable*”) are reported at the top of the table.

	L-CORE	L-SURF	L-O2	L-BP	SURF-STBL	CORE-STBL	BP-STBL	COMF	DECS
15	mid	low	good	high	unstable	unstable	stable	15	S
9	mid	high	good	mid	stable	stable	stable	10	S
11	mid	mid	good	mid	stable	stable	stable	15	A
18	mid	low	good	high	unstable	stable	stable	10	A
20	mid	mid	good	mid	stable	stable	stable	10	A
21	low	high	good	mid	unstable	stable	stable	15	A
24	mid	mid	good	mid	unstable	stable	stable	10	A
26	low	mid	good	mid	unstable	stable	stable	10	A
31	mid	mid	good	high	unstable	stable	stable	10	A
34	mid	low	good	mid	stable	stable	stable	10	A
36	mid	mid	good	mid	stable	stable	stable	10	A
39	low	low	good	mid	stable	stable	stable	7	S
41	low	low	good	mid	unstable	stable	stable	10	A
42	low	mid	good	mid	stable	stable	stable	15	S
43	high	high	good	high	unstable	stable	stable	15	S
44	mid	mid	good	mid	stable	stable	stable	10	S
46	low	mid	good	mid	unstable	stable	stable	10	S
47	low	mid	good	high	unstable	stable	stable	?	I
50	mid	high	good	low	unstable	stable	stable	10	A
56	mid	mid	good	mid	unstable	stable	stable	15	A
59	low	low	good	high	stable	stable	stable	10	A
66	mid	mid	good	high	stable	stable	stable	10	S
80	mid	mid	good	high	stable	stable	stable	10	A
88	mid	mid	good	mid	unstable	stable	stable	15	A
90	mid	mid	good	mid	unstable	stable	stable	15	S
1	mid	low	excellent	mid	stable	stable	stable	15	A
2	mid	high	excellent	high	stable	stable	stable	10	S
3	high	low	excellent	high	stable	stable	mod-stable	10	A
4	mid	low	good	high	stable	unstable	mod-stable	15	A
5	mid	mid	excellent	high	stable	stable	stable	10	A
6	high	low	good	mid	stable	stable	unstable	15	S
7	mid	low	excellent	high	stable	stable	mod-stable	5	S
8	high	mid	excellent	mid	unstable	unstable	stable	10	S
10	mid	low	excellent	mid	unstable	stable	mod-stable	10	S
12	mid	low	good	high	stable	stable	mod-stable	10	A
13	high	high	excellent	high	unstable	stable	unstable	15	A
14	mid	high	good	mid	unstable	stable	mod-stable	10	A
16	high	high	excellent	high	unstable	stable	unstable	10	A
17	low	high	good	high	unstable	stable	mod-stable	15	A
19	mid	high	good	mid	unstable	stable	unstable	15	A
22	low	mid	excellent	high	unstable	stable	unstable	10	S
23	mid	mid	good	mid	unstable	stable	unstable	15	A
25	high	high	good	mid	stable	stable	mod-stable	10	A
27	high	mid	good	low	stable	stable	mod-stable	10	A
28	low	mid	excellent	high	stable	stable	mod-stable	10	A
29	mid	mid	excellent	mid	stable	stable	unstable	15	A
30	mid	mid	good	mid	unstable	stable	unstable	10	S
32	low	low	good	mid	unstable	stable	unstable	10	A
33	mid	mid	excellent	high	unstable	stable	mod-stable	10	A
35	low	mid	excellent	high	stable	stable	mod-stable	10	A
37	low	mid	excellent	mid	stable	stable	stable	10	S
38	low	low	good	mid	unstable	stable	unstable	10	S
40	mid	mid	good	high	unstable	stable	mod-stable	10	A
45	low	low	excellent	mid	stable	stable	stable	10	A
48	mid	mid	excellent	mid	unstable	stable	stable	10	A
49	high	high	excellent	high	stable	stable	unstable	?	A
51	mid	high	good	mid	unstable	mod-stable	mod-stable	10	A
52	low	high	excellent	mid	unstable	stable	stable	10	A
53	mid	low	excellent	high	unstable	stable	unstable	10	A
54	mid	mid	good	mid	unstable	stable	mod-stable	10	S
55	high	high	excellent	mid	unstable	stable	mod-stable	10	A
57	high	mid	good	high	stable	stable	unstable	15	A
58	mid	low	good	high	unstable	stable	mod-stable	10	A
60	mid	high	good	mid	stable	stable	mod-stable	10	A
61	mid	high	good	mid	unstable	stable	unstable	10	A
62	mid	low	excellent	high	stable	stable	stable	10	A
63	mid	mid	good	mid	stable	stable	unstable	10	A
64	mid	low	excellent	mid	stable	stable	unstable	10	S
65	high	mid	excellent	mid	unstable	unstable	unstable	10	A
67	mid	low	excellent	mid	unstable	stable	stable	10	A
68	mid	mid	excellent	mid	unstable	stable	stable	10	A
69	mid	mid	excellent	high	stable	stable	stable	10	A
70	mid	mid	excellent	low	stable	stable	stable	10	A
71	mid	low	excellent	mid	unstable	unstable	unstable	?	A
72	low	low	excellent	mid	stable	stable	stable	10	A
73	mid	mid	excellent	mid	stable	stable	mod-stable	10	S
74	mid	mid	excellent	high	stable	stable	stable	10	A
75	mid	low	excellent	high	stable	stable	mod-stable	10	A
76	low	mid	good	mid	stable	stable	unstable	10	A
77	mid	mid	excellent	mid	stable	stable	mod-stable	10	A
78	mid	mid	excellent	mid	stable	stable	unstable	10	A
79	mid	mid	excellent	mid	unstable	unstable	stable	10	S
81	mid	mid	excellent	mid	stable	stable	stable	15	A
82	mid	mid	excellent	mid	stable	stable	stable	10	S
83	mid	low	good	mid	stable	stable	unstable	10	I
84	high	mid	excellent	mid	unstable	stable	unstable	5	A
85	mid	mid	excellent	mid	stable	stable	unstable	10	A
86	mid	mid	excellent	mid	unstable	stable	stable	10	A
87	mid	mid	excellent	mid	unstable	stable	stable	15	S
89	mid	mid	excellent	mid	unstable	stable	stable	10	A



### Breast cancer data

This breast cancer domain was obtained from the UCI repository [Newman et al., 1998] and was originally provided to UCI by M. Zwitter and M. Soklic of the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. The data set includes 286 instances. The instances are described by 9 categorical attributes, shown in the Table 5.2.

	Attribute	Values
1	<i>class</i>	no-recurrence-events, recurrence-events.
2	<i>age</i>	10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
3	<i>menopause</i>	lt40, ge40, premeno.
4	<i>tumor-size</i>	0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.
5	<i>inv-nodes</i>	0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
6	<i>node-caps</i>	yes, no.
7	<i>deg-malig</i>	1, 2, 3.
8	<i>breast</i>	left, right.
9	<i>breast-quad</i>	left-up, left-low, right-up, right-low, central.
10	<i>irradiat</i>	yes, no.

**Table 5.2.** Breast cancer data: attribute domains.

The algorithm is executed on some interesting objects, with parameters  $\theta = 0.7$  and  $\sigma = 0.3$ . Next three objects that showed a clear outlieriness are discussed about.

The object of row 129 scores outlieriness 0.960 with explanation  $\{ irradiat, inv-nodes, no-recurrence-events \}$  and outlier property  $\{ node-caps \}$ . This property is exceptional since among the objects with *irradiat*="no", *inv-nodes*="9-11", *class*="no-recurrence-events", very few objects are such that *node-caps*="yes".

The object of row 274 scores outlieriness 0.906 with explanation  $\{ inv-nodes, menopause \}$  and outlier property  $\{ node-caps \}$ . This property is exceptional since among the objects with *inv-nodes*="0-2", *menopause*="ge40", very few objects are such that *node-caps*="yes".

Finally, the object of row 286 scores outlieriness 0.904 with explanation  $\{ node-caps, irradiat, breast \}$  and outlier property  $\{ inv-nodes \}$ . This property is exceptional since among the objects with *node-caps*=no, *irradiat*="no", *breast*="left", very few objects are such that *inv-nodes*="3-5".

### Water supply data

The table of water supplies of an Italian town is analyzed. The table consists of supplies of a single month of the year 2001. It includes 38 attributes and 31,084 rows, each associated with a single customer. Attributes specify some customer information like, e.g., consumed water quantity, amounts due, and others. The town administrator provided us with an exceptional individual who complained about a bill to pay relative to the period which

Attrs / Vals		10	100	1000
50	mean	0.029976	0.002234	0.000640
	max	0.041120	0.003511	0.000902
100	mean	0.033693	0.002364	0.000067
	max	0.045251	0.003741	0.000090

**Table 5.3.** Random dataset family: experimental results.

the table refers to. The object under examination is found to have an exceptional property *AMOUNT-DUE*=“*very high*” with explanation *WATER-QUANTITY*=“*very low*” and *TYPE*=“*normal*”. This property has an outlierness 0.999537 in that it is the only object with *AMOUNT-DUE*=“*very high*” among 28,044 selected by the explanation, of which 28,038 are such that *AMOUNT-DUE*=“*very low*”, four such that *AMOUNT-DUE*=“*low*”, and an other such that *AMOUNT-DUE*=“*medium*”.

Notice that, even in this case, the method succeeded in singling out a significant justification for the notified abnormality, which evidently corresponds to an error made with the computation of the amount due for the water supply at hand.

#### 5.10.4 Random data

To verify if the method finds meaningful outlying properties also for non-exceptional objects, how it performs on uniformly distributed data is studied. Six random datasets, each composed by ten thousands objects, consisting of 50 or 100 attributes whose domain includes ten, or one hundred, or one thousand distinct and uniformly distributed values, have been considered. For  $\sigma = 0.1$  and  $\theta = 0.0$ , the mean and the maximum outlierness of the top outlier–explanation pair associated with one hundred randomly selected objects, is computed. Table 5.3 shows the result of this experiment. From this table, it is clear that, on random data, the top outlying property has a negligible score. Also, it is worth noting that, even though the outlierness threshold is set to zero and the explanation threshold is set to the ten percent, in all the executions the algorithm explored only the first level of the explanation tree and only the first two levels of the outlier tree.

This proves that the algorithms do not found exceptional properties if no meaningful one is present, and that the algorithms are able to recognize very quickly the absence of outlying properties.

---

## Conclusions

In this thesis an outstanding data mining task as the anomaly detection is tackled. Its wide practical applications are well known, and a lot of efforts have been made in these recent years, to find efficient and effective techniques to accomplish this task. Among the many facets of the anomaly detection, this thesis deals mainly with three of them: *outlier detection in data*, *outlier detection in data streams* and *outlier property detection*.

The first one is the classical problem for anomaly detection. It consists of finding objects that are significantly different with respect to the population they belong to. In this context, the main contribution of the thesis is the presentation of a novel algorithm. By means of it, it is shown that distance-based outlier detection is possible on very large disk resident datasets with both near linear CPU and I/O cost and simultaneously gaining efficiency by exploiting database indexing technology. The algorithm DOLPHIN, here described, outperforms existing methods by at least one order of magnitude. The proposed technique can be interestingly extended in several ways. DOLPHIN has very low space requirements that, for meaningful values of parameters, amounts to a little percentage of the input data set; however, in some applications, the mining tasks must be accomplished using only a limited amount of memory. Then, designing a variant of DOLPHIN which uses only a fixed amount of main memory may arouse a great interest. Furthermore, a deeper theoretical analysis may be conducted, in order to statistically bound the spatial cost of the algorithm, and to study the expected behavior of the algorithm on data set with a known distribution.

As for the second aspect this thesis deals with, the problem of detecting distance-based outliers in streams of data has been addressed. The novel *data stream outlier query* task was proposed and motivated, and both an exact and an approximate algorithm to solve it were presented. Also, bounds on the accuracy of the estimation accomplished by the approximated method are precisely stated. Finally, experiments conducted on both synthetic and real data sets showed that the proposed methods are efficient in terms processing time, and the approximate one is effective in terms of precision and recall of

the solution. This approach may be further improved by extending the statistical analysis and by studying the theoretical behavior of STORM when the data stream distribution is a known one. Moreover, a more extensive experimental campaign may be conducted, especially on real cases of study, and the embedding of STORM in real applicative system may be accomplished.

Finally, the third problem tackled in this thesis is a relatively new and till now less considered task of finding exceptional properties of exceptional objects. The problem consists of singling out subsets of attributes that best justify a given abnormal individual for its exceptionality with respect to a reference data population. This problem is somehow orthogonal to that of individuating outliers in a database. In this latter case, indeed, given a characteristics to be analyzed, the aim is to find those individuals that are significantly dissimilar from the other individuals of the data population they belong to as far as that characteristics is concerned. In the context analyzed here, conversely, the abnormal individual is given in advance, and it is of interest singling out the characteristics that best distinguish this individual from the rest of the data population.

From a practical viewpoint, the problem dealt with appears to be significant: think of singling out the peculiarities of exceptional performances in sports, or those of sick individuals in a context of otherwise healthy people or, finally, those characterizing people committing financial criminal acts.

A precise framework for the problem at hand has been first defined, and then its formal characterization has been developed. As a result, both an analysis of the computational complexity of involved problems and several formal tools, which have been exploited in designing as efficient as possible algorithms for computing the most relevant justification for the abnormality of the given individual, are obtained. This way, a fully automated support is provided to decode those properties determining the abnormality of the given object within the given data context. This work might be further extended along several directions. First of all, it might be interesting to devise methods to single out outlying properties conjunctively associated with a *set* of abnormal individuals. Second, it would be sensible to exploit additional information possibly available in specific application contexts in order to speed-up the search using heuristic functions. Third, a more extensive experimental campaign would be useful to further assess the behavior of the algorithms over actual data domains.

---

## References

- C. C. Aggarwal. On abnormality detection in spuriously populated data streams. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2005.
- C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 37–46, 2001.
- R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.
- F. Angiulli and C. Pizzuti. Fast outlier detection in large high-dimensional data sets. In *Proceedings of the International Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 15–26, 2002.
- F. Angiulli and C. Pizzuti. Outlier mining in large high-dimensional data sets. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pages 203–215, 2005.
- A. Arning, C. Aggarwal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 164–169, 1996.
- B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.
- V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994.
- S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 29–38, 2003.
- N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The  $r^*$ -tree: An efficient and robust access method for points and rectangles. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 322–331, 1990.
- J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, pages 509–517, 1975.

- S. Berchtold, D. A. Keim, and H. P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the International Conference on Very Large Data Bases System (VLDB)*, pages 28–39, 1996.
- C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, pages 322–373, 2001.
- M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 93–104, 2000.
- S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, pages 65–74, 1997.
- E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computer Surveys*, pages 273–321, 2001.
- Z. Chen and S. Toda. The complexity of selecting maximal solutions. *Information and Computation*, pages 231–239, 1995.
- Defense Advanced Research Projects Agency DARPA. Intrusion detection evaluation. In [www.ll.mit.edu/IST/ideval/index.html](http://www.ll.mit.edu/IST/ideval/index.html), 1998.
- E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection : Detecting intrusions in unlabeled data. In *Applications of Data Mining in Computer Security*, 2002.
- M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- A. Ghoting, M. E. Otey, and S. Parthasarathy. Loaded: Link-based outlier and anomaly detection in evolving data sets. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 387–390, 2004.
- A. Ghoting, S. Parthasarathy, and M. E. Otey. Fast mining of distance-based outliers in high-dimensional datasets. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2006.
- L. Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, pages 5–14, 2003.
- D. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.
- W. Jin, A. K. H. Tung, and J. Han. Mining top-n local outliers in large databases. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 293–298, 2001.
- R. M. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85–103. New York: Plenum, 1972.
- W. Klösgen. Explora: a multipattern and multistrategy discovery assistant. *Advances in knowledge discovery and data mining*, pages 249–271, 1996.
- E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 392–403, 1998.
- E. Knorr, R. Ng, and V. Tucakov. Distance-based outlier: algorithms and applications. *VLDB Journal*, pages 237–253, 2000.
- E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 211–222, 1999.
- D. E. Knuth. *The Art of Computer Programming, Vol.3 – Sorting and Searching*. Addison-Wesley (Reading MA), 1973.

- M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences (JCSS)*, pages 490–509, 1988.
- A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2003.
- D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998. URL [www.ics.uci.edu/~mllearn/MLRepository.html](http://www.ics.uci.edu/~mllearn/MLRepository.html).
- C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 315–326, 2003.
- S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 427–438, 2000.
- R. Rymon. Search through systematic set enumeration. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 539–550, 1992.
- S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 168–182, 1998.
- A. L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences (JCSS)*, pages 357–381, 1994.
- S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 187–198, 2006.
- P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
- Y. Tao, X. Xiao, and S. Zhou. Mining distance-based outliers from large databases in any metric space. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 394–403, 2006.
- J. D. Ullman. *Principles of Database Systems, 2nd Edition*. Computer Science Press, 1982.
- O. Watanabe. Simple sampling techniques for discovery science. *IEICE Transactions on Communications, Electronics, Information and Systems (TIEICE)*, 2000.
- L. Wei, W. Qian, A. Zhou, W. Jin, and J.X. Yu. Hot: Hypergraph-based outlier test for categorical data. In *Proceedings of the International Conference on Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 399–410, 2003.
- M. Wu and C. Jermaine. Outlier detection by sampling with accuracy guarantees. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 767–772, 2006.
- K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne. On-line unsupervised learning outlier detection using finite mixtures with discounting learning algorithms. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 250–254, 2000.

- J. Zhang and H. Wang. Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowledge and Information Systems (KAIS)*, pages 333–355, 2006.
- J. Zhang, Q. Gao, and H. Wang. A novel method for detecting outlying subspaces in high-dimensional databases using genetic algorithm. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 731–740, 2006.
- C. Zhu, H. Kitagawa, and C. Faloutsos. Example-based robust outlier detection in high dimensional datasets. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 829–832, 2005.