UNIVERSITY OF PIRAEUS

DEPARTMENT OF INFORMATICS

Similarity Issues in Data Mining – Methodologies and Techniques

PhD thesis

Irene Ntoutsi

MSc, Dipl. Computer Engineering & Informatics

Piraeus, June 2008

Approved by the Examination Committee, June 2008

.....	.....	.....
Y. Theodoridis	G. Tsihrintzis	M. Vazirgiannis
Asst. Professor, U. Piraeus	Assoc. Professor, U. Piraeus	Assoc. Professor, AUEB
Supervisor	Member of Advisory Committee	Member of Advisory Committee

.....	.....	.....
D. Despotis	T. Sellis	Y. Siskos
Professor, U. Piraeus	Professor, NTUA	Professor, U. Piraeus
Member of Examination Committee	Member of Examination Committee	Member of Examination Committee

.....

G. Vassilacopoulos

Professor, U. Piraeus

Member of Examination Committee

Preface

The amount of patterns extracted nowadays from Knowledge Discovery and Data Mining (KDD) is rapidly growing, thus imposing new challenges regarding their management. One of the most important operations on the extracted pattern sets is that of dissimilarity assessment which raises a lot of fruitful research issues and results in a variety of important applications.

This dissertation studies several issues that arise during the pattern dissimilarity assessment process. At first, we propose a generic framework for the comparison of arbitrary complex patterns defined over raw data and over other patterns. Next, we study specific dissimilarity problems for the most popular pattern types, namely frequent itemsets, decision trees and clusters. More specifically, for the case of frequent itemset patterns, we study how the mining parameters affect the dissimilarity assessment process. For the case of decision tree patterns, we propose a framework that evaluates dissimilarity between both decision trees and classification datasets. Finally, for the case of clusters, we propose dissimilarity measures between clusters and clusterings, which we then employ for change detection in dynamic populations. All the above were studied under the consideration that patterns (of any type) are composed of a structure and a measure component, which opens a field towards a unified model for KDD results.

Irene Ntoutsi

Acknowledgements

He dies a slow death who quits a project before starting it, not asking
about what he doesn't know.
No slow death for me, Pablo Neruda

My very special thanks to my advisor, Asst. Prof. Yannis Theodoridis, for his support and guidance during all these years. His knowledge and experience has been very supportive all those times where specific problems seemed to be insuperable. His attitude inspired me in both scientific and human level.

Much of the work in this thesis is a result of collaboration. With Prof. Myra Spiliopoulou we worked together on cluster evolution issues and I would like to thank her not only for her contribution but also for her hospitality in Magdeburg. With Ilaria Bartolini, Marco Patella and Prof. Paolo Ciaccia, we worked together on the PANDA framework; this was my first external collaboration and I would like to thank them a lot for their confidence. With Alexandros Kalousis we collaborated by chance, starting from a square in Porto; I would like to thank him for his help with decision trees.

I would like to thank my labmates in the Database Group (Kostas, Evaggelos, Gerasimos, Elias, Nikos, Despoina, Nikos) for our cooperation but also for our nice and stress-free conversations.

Also, I would like to thank all the scientists I met all these years, since they affected my way of thinking about the PhD and scientific life. I would like to mention here the symposium in Mantaneia, the meetings for the PANDA project, the HDMS conferences and the summer school in Bolzano. Special thanks to Assoc. Prof. Michalis Vazirgiannis, for his fruitful comments.

My thanks go to all my friends for being there for me and understanding my up and down moods. Many thanks to Spyros for a lot of corrections in English. Special thanks to Nikos for substantially help me, supporting me and believing in me all these years.

Finally, I would like to thank my parents and my sisters for their endless love and care.

Irene Ntoutsi

Contents

List of Figures

List of Tables

Chapter 1

Pattern Management

In this chapter, we analyze the need for pattern management emphasizing on a specific aspect of the pattern management problem, the dissimilarity assessment issue.

The chapter is organized as follows: In Section 1.1, we explain why patterns are so popular nowadays. In Section 1.2, we overview the basics of the KDD process emphasizing on the Data Mining step. In Section 1.3, we present the motivation behind pattern management and overview the work performed so far in this area. In Section 1.4, we concentrate on the pattern dissimilarity assessment problem, where we justify its importance and present its challenges. In Section 1.5, we outline the organization of the thesis.

Index terms pattern management, Pattern Base Management Systems (PBMS), dissimilarity assessment problem.

1.1 The compulsory need for patterns

Due to the wide spread usage of computer devices nowadays, our ability to produce and collect data has impressively increased. As a result, *huge amounts of data* are collected from different application domains like business, science, telecommunication and health care systems. Also, the World Wide Web overwhelms us with information. According to a recent survey [60], the world produces between 1 and 2 exabytes of unique information per year, which is roughly 250 megabytes for every man, woman and child on earth.

Apart from their huge quantity, modern data are also characterized by *low level of abstraction*. For example, supermarkets collect transaction data of their customers consisting of the items purchased by a customer in a single transaction. Telecommunication compa-

nies collect calling data including details such as the time and duration of a call. Web site owners collect click-stream data from their visitors concerning information such as the sequence of pages visited during a session and the duration of browsing a single page. Tracking devices, e.g., GPS, periodically transmit location and time information to main servers.

Other characteristics of the data nowadays are their *high degree of diversity* (e.g., transactions of a supermarket, satellite images, music parts) and *complexity* (e.g., text, image, sound, video). Furthermore, data are not only produced in a *centralized* but also in a *distributed* way, which imposes new challenges regarding their management.

Due to the above mentioned reasons, it is impossible for humans to thoroughly investigate these data collections through a manual process. *Knowledge Discovery in Databases (KDD)* and *Data Mining (DM)* provide a solution to this problem by generating patterns from raw data. DM constitutes actually one of the steps of the KDD process; we overview the KDD steps in the next section concentrating mainly on the DM step.

1.2 The KDD process and the Data Mining step

Definition 1 *Knowledge Discovery in Databases (KDD) is a nontrivial procedure for the extraction of valid, previously unknown, potentially useful and easily understood patterns from data [29].*

This is a general definition of the KDD process. The term *data* corresponds to a set of instances stemming from the problem under investigation. It might be for example, a set of customer transactions for a supermarket application, or a set of call records for a telecommunication application. The term *pattern* refers to an expression in a specific language that describes some subset of the data, e.g., "*If income* $\leq 10K$*, then the loan request is rejected*". The discovered patterns should be *valid*, which means that to a certain degree they should also hold for new, previously unseen problem instances. Patterns should also be *useful* and facilitate the end user in the decision making process. And definitely, the end user should be able to *interpret* patterns in an *efficient* and *effective* way.

The term KDD refers to the whole process of discovering knowledge from data. It consists of five steps: i) *Selection* of data which are relevant to the analysis task, ii) *Preprocessing* of these data, including tasks like data cleaning and integration, iii) *Transformation* of the data into forms appropriate for mining, iv) *Data Mining*

for the extraction of patterns, and v) *Interpretation/Evaluation* of the generated patterns so as to identify those patterns that represent real knowledge, based on some interestingness measures. These steps are depicted in Figure 1.1.

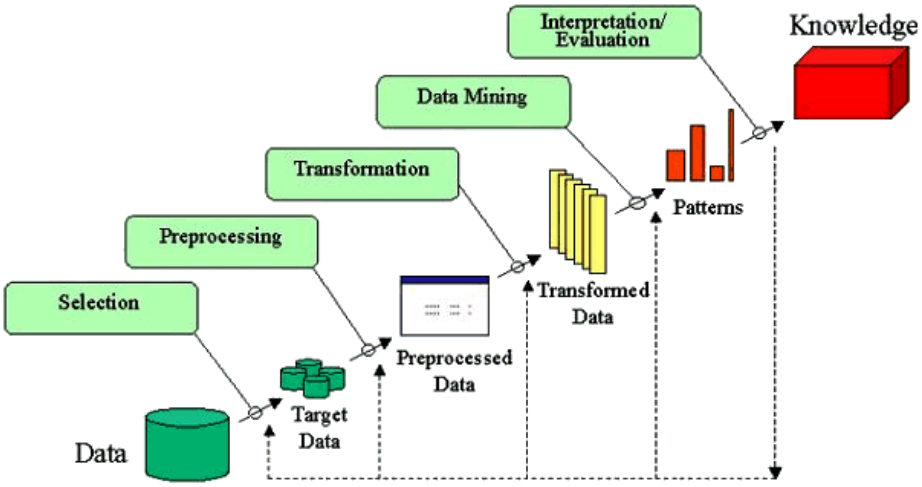


Figure 1.1: The KDD steps

The Data Mining step is in the core of the KDD process.

Definition 2 *Data Mining is a step in the KDD process that consists of applying data analysis and discovery algorithms that produce a particular enumeration of patterns (or models) over the data [29].*

Patterns can be described as compact and rich in semantics representations of raw data [85]: *compact* since they provide a high level description of the raw data characteristics and *rich in semantics* since they reveal new knowledge hidden in the mass of raw data.

The main goals of the Data Mining step are prediction and description. *Prediction* refers to the prediction of the value of a specific attribute and is applied over new, previously unseen problem instances, whereas *description* refers to the extraction of easily interpretable patterns from data.

The main Data Mining tasks are classification, clustering and association rules extraction. *Classification* aims at building a model that distinguishes data into classes; this model is used for predicting the class of objects with unknown class label. A typical classification problem is that of loan credit assignment in a bank, where the goal is to discover whether some loan application should be accepted or rejected. *Clustering* refers to the partitioning of data into groups so

as similar instances are placed in the same group. As a clustering example, consider the grouping of people according to their education and salary. *Association Rules Extraction* aims at discovering associations between attribute-value pairs that occur frequently together in the data. A well known application is that of market basket analysis, where the goal is to discover the products that people tend to buy together.

Among the most popular data mining models (or pattern types) are decision trees, association rules, clusters and sequences.

1.3 Pattern management

Due to the wide application of KDD and DM and as a result of data flood, the amount of patterns extracted nowadays from heterogeneous data sources is *huge* and, quite often, *non-manageable* by humans. Thus, there is a need for pattern management including issues like modeling, storage, retrieval and querying of patterns. Pattern management is not a trivial task. Apart from the huge amounts of the generated patterns, another reason is the large *variety of pattern types* as a result of the different application needs that each type tries to accomplish.

So far, the majority of the work in the data mining area mainly focuses on efficient mining, putting aside the pattern management problem. Recently, however, the need for pattern management has been recognized by both scientific and industrial parts resulting in a number of approaches towards efficient pattern management. The main difference between scientific and industrial approaches lies in the fact that scientific approaches try to deal with all the aspects of the pattern management problem including both representation and manipulation issues. On the other hand, industrial approaches mainly focus on pattern representation and storage issues, aiming at easy interchange of patterns between different vendor applications. In the next subsections, we quickly overview these approaches; further details can found in [76, 92].

1.3.1 Scientific approaches

Scientific approaches try to provide an overall solution to the pattern management problem, providing both representation, storage and retrieval capabilities. The basic approaches in this category are the so-called inductive databases, 3–Worlds model and PANDA model.

Inductive databases The inductive databases framework [43], introduced in 1996, is inspired by the idea that the data mining process should be supported by the database technology. This is the reason why inductive databases rely on an integrated architecture where data and patterns are stored in the same repository.

Within the inductive databases framework, the KDD process is considered to be a kind of extended query processing in which users can query both data and patterns. Towards this aim, a so–called *inductive query language* is used, which comprises an extension of a database query language allowing to: i) select, manipulate and query *data* as in standard queries, ii) select, manipulate and query *patterns* and, iii) execute *cross–over* queries over patterns, i.e., queries that associate patterns to raw data. Due to the importance of querying in inductive databases, several query languages-extensions to SQL have been proposed like DMQL [38], MINE–RULE [63] and MINE–SQL [44].

3-Worlds model The 3-Worlds model (3W) [49], introduced in 2000, provides a unified framework for pattern management and, to the best of our knowledge, it comprises the first attempt towards discrete pattern management.

The 3W model is based on a separated architecture consisting of three distinct worlds: i) the *intensional world* of patterns, ii) the *data world* of data, and iii) the *extensional world* that determines the mappings between data and patterns. The manipulation of each world can be performed through an algebra of choice. For the intensional world, the authors propose the *dimension algebra* which extends traditional relational algebra operators through new operators that add significant value to data mining and analysis. To facilitate moving in and out of the worlds, the authors propose the so called *bridge operators* namely populate, mine, lookup and refresh.

PBMS (Pattern Base Management Systems) The PBMS model [85], introduced in 2003 in the settings of the PANDA project [80], provides a unified framework for the representation of heterogeneous patterns relying on a separated architecture where data and patterns are stored in different repositories. The different pattern types are represented under a common schema, as quintuples of the form $pt = (n, ss, ds, ms, et)$: n is the *name* of the pattern type; ss is the *structure schema* that defines the pattern space; ds is the *source schema* that defines the source data space; ms is the *mea-*

sure schema that describes the measures that quantify the quality of the source data representation achieved by the patterns; *et* is the *expression template* that describes the relationship between the source space and the pattern space, thus carrying the semantics of the patterns. A *pattern* comprises an instantiation of a pattern type. Semantically related patterns are grouped into *classes*.

For the manipulation of patterns, a Pattern Manipulation Language (PML) and a Pattern Query Language (PQL) have been proposed that support queries over patterns and cross over queries between patterns and data. The basic model [85] has been extended in [20] by adding, in pattern definition, the notions of temporal validity, semantic validity and safety. Prototype implementations of the PANDA model appear in [19, 53, 52].

1.3.2 Industrial approaches

Industrial approaches mainly focus on pattern representation and storage aiming at the easy interchange of patterns between different vendor applications rather than the efficient management of these patterns. Several specifications and standards (PMML, CWM, ISO SQL/MM, JDMAPI) have been proposed in this category, some of which are described bellow. Also, extensions of existing commercial DBMS have been proposed like Oracle Data Mining [78], IBM DB2 Intelligent Miner [42] and Microsoft SQL Server Analysis Manager [67].

Predictive Model Markup Language (PMML) PMML [26] is an XML-based language providing a quick and easy way for companies to define data mining and statistical models using a vendor–independent method and share these models between PMML compliant applications. The structure of the models is described by an XML Schema.

ISO SQL/MM SQL/MM [93] comprises a specification for supporting data management of common data types (text, images, DM results, etc.) relevant to multimedia and other knowledge intensive applications. Part 6 of SQL/MM specification refers to Data Mining. SQL/MM defines first-class SQL types that can be accessed through SQL:1999 base syntax. In the case of DM models, every model has a corresponding SQL structured user-defined type.

Common Warehouse Model (CWM) CWM [22] is a specification that enables easy interchange of metadata between data warehousing tools and metadata repositories in distributed heterogeneous environments. It consists of a number of sub–metamodels representing common warehouse metadata in the areas of Data Resources, Data Analysis (including OLAP and Data Mining) and Warehouse Management.

Java Data Mining API (JDMAPI) Java Data Mining API [48] addresses the need for an independent (of the underlying data mining system) Java API that will support the creation, storage, access and maintenance of data and metadata. It provides a standardized access to data mining patterns represented in various formats, e.g., PMML [26].

1.4 Pattern dissimilarity assessment

So far, the importance of the pattern management problem has been demonstrated together with the different aspects that management involves (e.g., modeling, storage and manipulation of patterns) and the work performed so far in the area by both scientific and industrial parts.

Among the several interesting operators that might be defined over patterns, one of the most important is that of dissimilarity assessment, i.e., detecting how similar two patterns are to each other. In the following subsections, several application examples are presented, which denote the importance of the dissimilarity assessment problem and the challenges that it raises.

1.4.1 The importance of dissimilarity assessment

Defining similarity/distance[1] operators over patterns results in a variety of interesting applications. We briefly present some of them below:

- *Similarity queries*

  A straightforward application of a dissimilarity operator is to express similarity queries over a set of patterns (or pattern base) including *k-nearest neighbor queries* (i.e., find the k-most similar pattern(s) to a query pattern) and *range queries* (i.e.,

[1]Similarity and distance are complementary notions; the generic term dissimilarity refers to either distance or (the inverse of) similarity.

find the most similar pattern(s) to a given pattern within a given range). The efficient computation of dissimilarity is one of the core issues for a PBMS [85] with applications in pattern indexing and retrieval.

- *Monitoring and change detection*

  Another application of dissimilarity is monitoring and detecting changes in patterns, e.g., detect changes in customer behaviors over time. This is especially useful nowadays since data are highly dynamic and thus, the corresponding patterns are also volatile and associated to some temporal semantics. Furthermore, it is usually more helpful for the end user to know how the old patterns (which represent the current knowledge about the problem) have been changed rather than to deal with a new set of patterns (which represent the new knowledge about the problem).

  Change detection is also helpful for *KDD synchronization*, in order to keep patterns up to date with respect to the corresponding raw data (e.g., synchronizing patterns only when the corresponding raw data have significantly changed).

- *Dataset comparison*

  A common technique for the comparison of two datasets is to compare their corresponding pattern sets, e.g., compare two retail shops in terms of the buying behavior of their customers (which might be captured through a clustering model). The intuition behind utilizing pattern comparison for data comparison, is that, to some degree, patterns preserve the information contained in the original raw data, and therefore, dissimilarity in pattern space could be considered as a measure of dissimilarity in the original raw data space.

  Discovering a *mapping* (either exact or approximate) between the dissimilarity in data and pattern space is really useful. For example, we could avoid the hard task of comparing the original datasets whenever the corresponding pattern sets are available for comparison or even avoid the hard task of mining a dataset whenever it is found to be similar to another dataset for which the results of mining are already available.

- *Evaluating data mining algorithms*

  Currently, the experimental comparison between two algorithms (or the same algorithm with different criteria/parameters) is limited to the visual interpretation of their results. However,

such an evaluation could be carried out automatically by comparing their outcomes (Figure 1.2). Depending on the dataset characteristics (e.g., dense vs. sparse datasets) an algorithm might be more suitable than another. Therefore, one might need to experiment with many algorithms and evaluate their results in order to conclude which algorithm best fits her/his needs.

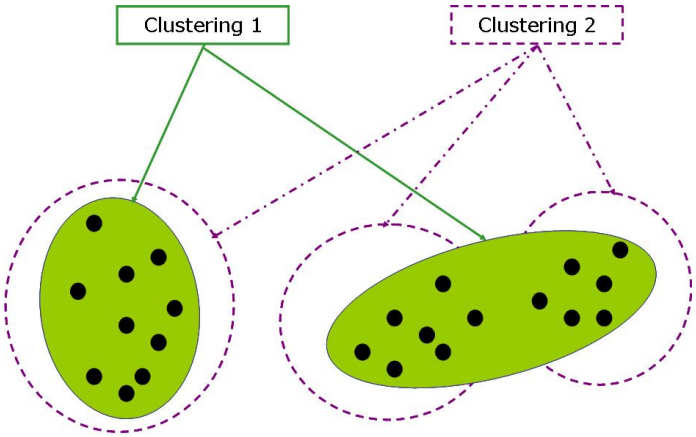


Figure 1.2: Comparing the outcome of different clusterings

- *Privacy aware data mining*

  There are cases where due to privacy reasons only patterns and not the actual raw data are available. Depending on the mining parameters, it is hard to recover the original raw data (for example, deciding whether there is a dataset compatible with a given set of frequent itemsets is NP-hard [64]). In this case, we should rely on the available pattern sets in order to compare the original datasets.

- *Mining from distributed data sources*

  In a distributed environment, the mining is not centralized since locally strong patterns should be also preserved. A common approach in this setting is to first cluster similar datasets into groups and then perform mining over each group independently [59]. The similarity between the datasets could be evaluated in terms of their corresponding pattern sets.

- *Discovering outlier or unexpected patterns*

  Discovering outliers or unexpected patterns constitutes another application of dissimilarity, which is usually of great importance

for the end user. A possible solution to this problem is to compare a discovered pattern with a target one, possibly provided by the user (Figure 1.3); patterns that significantly differ from the target pattern might be considered as outliers.

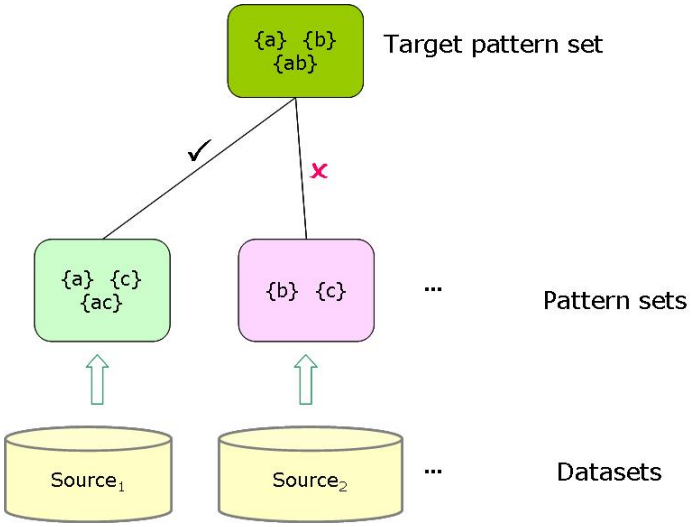


Figure 1.3: Comparing DM outcomes with respect to a target pattern set

1.4.2 The challenges of dissimilarity assessment

In the previous section, a variety of examples has been presented which indicate the importance of the pattern dissimilarity assessment problem and motivate its further investigation.

However, defining dissimilarity operators for patterns is not an easy task as it will be clear from the emerging challenges described below:

Challenge 1 *First of all, we should define* dissimilarity operators for the different pattern types*, e.g., clusters (Figure 1.4, left top), decision trees (Figure 1.4, right), association rules (Figure 1.4, left bottom), frequent itemsets etc.*

Challenge 2 *Except for patterns defined over raw data (hereafter called* simple patterns*), patterns defined over other patterns also exist, e.g., a clustering of clusters (Figure 1.5), an association rule over clusters, a forest of decision trees. For these patterns, hereafter called* complex patterns*, dissimilarity operators should also be defined.*

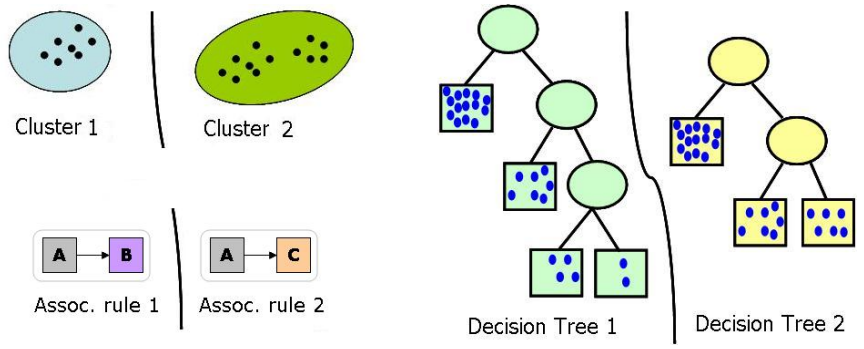


Figure 1.4: Comparing clusters (left top), association rules (left bottom) and decision trees (right)

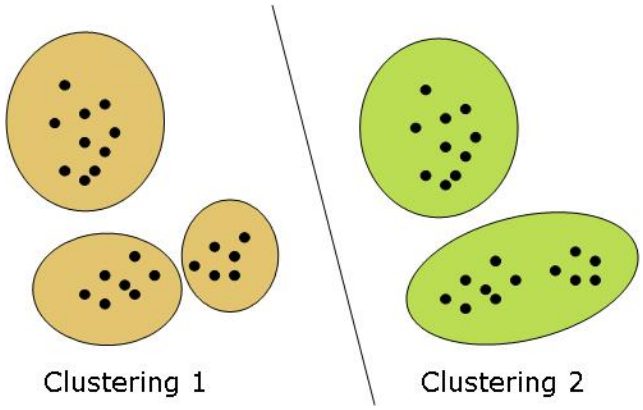


Figure 1.5: Comparing clusterings

Challenge 3 *Another interesting aspect of the dissimilarity assessment problem is the semantics of the dissimilarity operators with respect to the raw data from which patterns have been extracted. More specifically, it is really useful to investigate whether dissimilarity in pattern space depends on the dissimilarity in the original raw data space; we refer to this problem as the* dissimilarity reasoning problem *(Figure 1.6).*

In this thesis, we address the above mentioned challenges for some of the most popular pattern types, namely frequent itemsets, decision trees and clusters, whereas we also propose methods and techniques for similarity estimation between generic and arbitrary complex patterns like web sites and graphs.

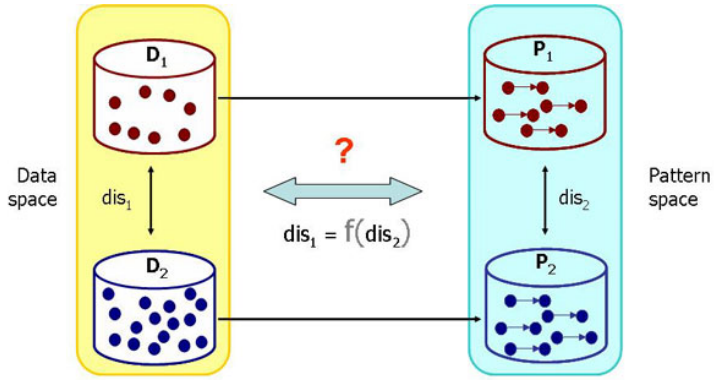


Figure 1.6: Dissimilarity reasoning

1.5 Outline of the thesis

The importance of the pattern management and, in particular, that of the dissimilarity assessment problem clearly arises through the discussion in this chapter. Although recently a lot of work has been done in the area of pattern management, this work mainly focuses on pattern representation, storage and querying aspects. A lot of interesting issues are to be explored, like dissimilarity assessment, indexing and visualization. In this thesis, we investigate different aspects of the pattern dissimilarity assessment problem for some of the most popular pattern types, namely frequent itemsets, decision trees and clusters. Early versions of this discussion appears in [73, 74, 53, 92, 76, 55, 54].

In Chapter 2, we make an introduction to the basic data mining pattern types studied in this thesis (namely, frequent itemsets, decision trees and clusters) so as to make the reader familiar with these concepts. In the same chapter, we also present a representation schema for patterns that utilizes both their extensional (i.e., in terms of data) and intensional (i.e., in terms of patterns) description. Early part of this work appears in [71, 75].

In Chapter 3, we present the PANDA framework for the assessment of dissimilarity between arbitrarily complex patterns. PANDA is capable of handling patterns defined over raw data (simple patterns) as well as patterns defined over other patterns (complex patterns). We instantiate PANDA for basic pattern types, including frequent itemsets, decision trees and clusters, and for more complex pattern types, like collections of documents and web sites. This part is in line with Challenge 2. An early version of our study has been published in [9, 55, 54], whereas an extended version has been submitted

in [10].

In Chapter 4, we concentrate on dissimilarity assessment between sets of frequent itemsets. More specifically, we study how the different mining parameters, namely the *minSupport* threshold used for the generation of itemsets and the adopted frequent itemset lattice representation (namely, frequent itemsets, closed frequent itemsets or maximal frequent itemsets), affect the derived dissimilarity score between two sets of itemsets. This part is in line with Challenge 1 and Challenge 3. An early version of this study appears in [69, 71], whereas an extended version has been submitted in [77].

In Chapter 5, we concentrate on dissimilarity assessment between decision trees and classification datasets. In particular, we present a general similarity assessment framework based on decision tree models, which includes as special cases i) the estimation of semantic similarity between decision trees and ii) various forms of similarity estimation on classification datasets, with respect to different probability distributions defined over the attribute-class space of the datasets. This part is in line with Chall enge 1 and Challenge 3. Results have been previously published in [70, 71].

In Chapter 6, we concentrate on dissimilarity assessment between clusters and clusterings, which are then used for change detection and monitoring in dynamic environments. More specifically, we provide a typification of the different transitions that a cluster might encompass, transition indicators for each transition type and a algorithm for their detection. Furthermore, we study how the evolution can be organized in an effective and efficient way so as to facilitate the end user to gain more insights on the population and to exploit the history of the population evolution. This part is in line with Challenge 1 and Challenge 3. Part of this work appears in [90, 89, 91, 88, 71, 54], whereas an extended version has been submitted in [72].

We conclude our thesis in Chapter 7, where we also discuss open issues and directions for future research.

Chapter 2

Preliminaries on Patterns

In this chapter, we overview three popular data mining pattern types relevant to our work, namely frequent itemsets (and their extensions, association rules), clusters (and their groupings, clusterings) and decision trees; our goal is to make the reader familiar with the notion of patterns in Data Mining.

After a short introduction (Section 2.1), we describe a representation schema for patterns based on both the extensional (i.e., in terms of data) and intensional (i.e., in terms of patterns) description of patterns (Section 2.2). Then we describe how the different pattern types (decision trees, frequent itemsets/assoociation rules and clusters/clusterings) can be expressed based on this schema (Section 2.3, 2.4 and 2.5, respectively). Section 2.6 summarizes this chapter.

Index terms data mining patterns, decision trees, frequent itemsets, association rules, clusters, clusterings.

2.1 Introduction

Knowledge Discovery in Databases (KDD) and Data Mining (DM) provide a solution to the information flood problem by extracting valid, novel, potentially useful, and ultimately understandable patterns from data [29]. *Patterns* constitute compact and rich in semantics representations of raw data [85]; compact by means that they summarize, to some degree, the amount of information contained in the original raw data, and rich in semantics by means that they reveal new knowledge hidden in the abundance of raw data.

Several *pattern types* exist in the literature, mainly due to the wide heterogeneity of data and data mining applications, as well

as due to the large variety of pattern extraction techniques as a result of the different goals that a mining process tries to accomplish (i.e., what are the data characteristics that the mining task tries to highlight). Different data mining tasks achieve different insights over the data: frequent itemsets capture the correlations between attribute-value pairs, clusters reveal natural groups in data, decision trees detect characteristics that predict (with respect to a given class attribute) the behavior of future records, and so on.

2.2 Pattern representation

Patterns summarize raw data in a compact and semantically rich way [94]. As such, the description of a pattern might be either *extensional*, i.e., in terms of the data members participated in its generation, or *intensional*, i.e., in terms of the meaning/concept represented by the pattern. The *extensional description of a pattern* is just an enumeration of its data members, thus it is common for all pattern types. The *intensional description of a pattern*, however, reveals information about the "shape" and the semantics of the pattern, thus it depends on the pattern type.

Regarding the intensional description of patterns, in [32] the authors propose the *2-component property of patterns*. According to this property, a broad class of pattern types can be described in terms of a structure and a measure component. The *structure component* describes the structure of the patterns which are instances of the particular pattern type; e.g., it is the "head" and the "body" in case of an association rule pattern. The *measure component* relates the patterns to the underlying raw data; e.g., it is the "support" and the "confidence" in case of the association rule pattern. In other words, the structural component describes the pattern space, whereas the measure component quantifies how well the pattern space describes the underlying raw data space.

The 2-component property of patterns has been extended in the settings of the PANDA approach [85], where a *unified model* for the representation of the different pattern types has been introduced. This model, besides the structure and measure components, also includes a source component that describes the dataset from which patterns have been extracted and an expression component that describes the relationship between the source data space and the pattern space (c.f. Section 1.3.1).

Similar ideas appear in [49], where the authors introduce the 3-Worlds model. In this model, patterns are described as constraints

over the attribute space, whereas their relationships to the original raw data space from which they have been extracted are also preserved (c.f. Section 1.3.1).

To summarize, throughout this work we adopt the *extensional-intensional description of patterns*. Regarding the extensional part, patterns are described in terms of the data subset that contributed in their generation (i.e., as an enumeration of the data instances). Regarding the intensional part, the 2-component property of patterns is adopted, that is, each pattern is described in terms of a structure and a measure component.

2.3 Decision trees

Decision Trees (DTs) are very popular *classification methods* due to their intuitive representation that render them easily understandable by humans. In this section, we provide some basic concepts on DTs [65].

Let $A = \{A_1, A_2, ..., A_m\}$ be the set of attributes on which classification will be based (*predictive attributes*), where attribute A_i has domain $dom(A_i)$. Let C be the *class attribute*, i.e., the attribute to be predicted, with domain $dom(C) = \{C_1, C_2, ..., C_k\}$ where k is the number of classes. The joint probability distribution of the predictive and the class attribute, $P = dom(A_1) \times dom(A_2) \times \cdots \times dom(A_m) \times dom(C)$, is called *problem distribution*. The probability distribution of the predictive attributes, i.e., $A = D(A_1) \times D(A_2) \times \cdots \times D(A_m)$, is called *attribute space distribution*.

The goal of a decision tree is to learn a predictor function $f : dom(A_1) \times dom(A_2) \times \cdots \times dom(A_m) \rightarrow dom(C)$. Towards this goal, a set of problem instances drawn from P is utilized; this is known as the *training set* D. A decision tree T constructed from D provides a classification of D instances into the C_j, $j = 1 \ldots k$, classes based on the values of the predictive attributes A_i, $i = 1 \ldots m$.

Predictive attributes might be either numerical, categorical or ordinal. The domain of a *numerical attribute* is an ordered set (e.g., age, income), the domain of a *categorical or nominal attribute* is a finite set without any natural ordering (e.g., colors, gender, marital status), whereas the domain of an *ordinal attribute* is a set of discrete values with an imposed order, but without any knowledge regarding the absolute differences between values (e.g., preference scale, severity of an injury). In practice, predictive attributes are usually numerical.

Regarding its structure, a decision tree consists of internal nodes and leaf nodes. An *internal node* has an associated test condition (*splitting predicate*) which specifies a test over some predictive attribute (e.g., "$Age \geq 20$"). Each branch descending from that node corresponds to one of the possible values for this attribute. Most common are binary predicates, i.e., predicates of the form "Yes" or "No". A *leaf node* provides the class label of the instances that follow the path from the root to this node. If the instances belong to more than one classes, this label might be the label of the majority class. In the general case, a leaf node might be associated to some degree to all problem classes; this degree/weight depends on the amount of instances that fall into the leaf and belong to the specific class.

In Figure 2.1 an example of a decision tree is depicted, which refers to the bank loan assignment problem. There are two predictive attributes: "Age" and "Income", whereas the class attribute C contains two values: $C = \{C_1, C_2\}$.

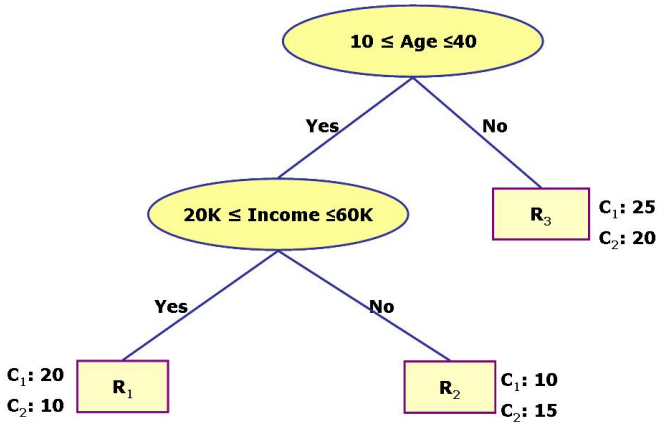


Figure 2.1: An example of a decision tree

A small part of the dataset used for the generation of the decision tree of Figure 2.1 is depicted in Table 2.1.

Instance	Age	Salary	Class
1	30	30K	C_1
2	35	10K	C_2
3	50	100K	C_1

Table 2.1: A sample of the training set for the DT of Figure 2.1

Decision tree evaluation As already stated, a DT is build upon a training set D of problem instances drawn from the joint prob-

ability distribution P of the predictive attributes and of the class attribute. A "fully developed tree" will perfectly fit the training set. However, a DT should not only fit the training set well, but it should also predict correctly the class labels of future, previously unseen problem instances; this property is known as the *generalization accuracy* of a DT. *Overfitting* the training set is a wrong property for a DT, since it might follow every idiosyncrasy of the training set, much of which are unlikely to occur in future problem instances. The generalization accuracy of a DT is evaluated through the *miss-classification error (ME)* measure, which is based on the number of instances wrongly predicted by the DT model.

Ideally, we would like to know the ME of the classifier f on the whole problem distribution P. However, since P is unknown (the only we know are some instances drawn from it, i.e., the training set), several techniques have been developed to estimate the miss-classification error of a classifier f over the problem distribution P, namely $ME(f, P)$. The most common technique is the *holdout test estimate*: the initial set of problem instances D is split into two disjoint sets: the training set and the test set. The *training set* is used to build the classifier, whereas the *test set* is used to evaluate its performance. Usually, 1/3 of instances is used for testing and 2/3 is used for training. Other popular techniques in this category are *re–substitution estimation* and *V –fold cross validation*.

A solution to the overffiting problem is *tree pruning*. Beginning from the bottom DT level, the child nodes are pruned away if the caused change in tree accuracy is less than a times the change in tree complexity. Due to pruning, the resulting tree might not perfectly predict its training set; this error is called *re-substitution error*. Thus, a good DT should minimize both the re–substitution error (with respect to the training set) and the miss–classification error (with respect to an independent test set).

Attribute Space Partitioning The DT growing process can be viewed as the process of partitioning the *attribute space*, i.e., the space defined by the predictive attributes: $D(A_1) \times D(A_2) \times \cdots \times D(A_m)$, into a set of disjoint regions until each region contains instances of the same class[1]. The border line between two neighboring regions of different classes is known as the *decision boundary*. Decision boundaries are parallel to the attribute axis since each test condition involves only a single attribute. Thus, decision regions are axis parallel hyper-rectangles, also called *isothetic* [49].

[1]This holds in the extended case, since after pruning, a region might contain instances from more than one problem classes.

Each leaf node of the tree corresponds to a region R. A *region* can be described *extensionally* as an enumeration of the data instances that are mapped to the corresponding leaf node. A region can be also described *intensionally* in terms of the corresponding tree path, which starts from the root of the tree and results in the corresponding leaf node. More specifically, the intensional description of a region consists of a structure and a measure component; the *structure component of a region* is the conjunction of the conditions across the corresponding tree path, whereas the *measure component of a region* is the distribution of region instances into the different problem classes.

The partitioning of DT presented in Figure 2.1 is depicted in Figure 2.2.

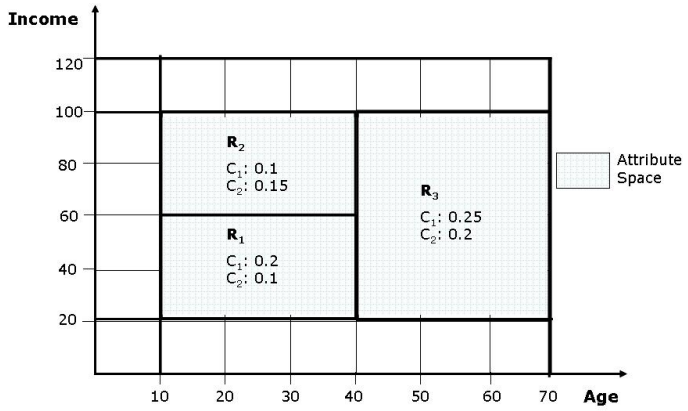


Figure 2.2: The attribute space partitioning achieved by the DT of Figure 2.1

Let us now see how the region R_1 of the DT (c.f. Figure 2.2) can be described in terms of the extensional-intensional representation schema. The structure component is: $R_1.struct = (10 \leq Age \leq 40) \cap (20K \leq Income \leq 60K)$, whereas the measure component is: $R_1.meas = \{(C_1 : 20\%), (C_2 : 10\%)\}$; these two measure components comprise the intensional description of R_1. The extensional description of R_1 contains the subset of the D instances that fall in R_1.

Note that the attribute space is determined by the predictive attributes of the problem, and as such it is common for all DTs referring to the specific classification problem. What actually differentiates the different DTs is the partitioning they perform over the attribute space, i.e., what are the resulting regions.

So far, we have concentrated on the representation of a specific DT region under the adopted extensional-intensional representation

schema. The whole DT consists of *a set of regions*, each region depicted under the extensional-intensional schema as described in this section.

Further detail on the DT partitioning can be found in Chapter 5.

2.4 Clusters and clusterings

Clustering is the unsupervised classification of data into natural groups (called *clusters*) so that data points within a cluster are more similar to each other than to data points in other clusters [47]. The term *unsupervised* stands for the fact that there is no a priori knowledge about the partitioning of the data. In a more formal definition, we can state that a clustering ζ is the partitioning of a dataset D into clusters $C_1, C_2, \ldots, C_k$ such that $C_i \cap C_j = \emptyset$ and $\cup_{j=1}^{k} C_j = D$. This definition stands for *hard clustering*, where an instance is assigned to exactly one cluster thus forming a crisp partitioning of the data set. A more "relaxed" definition of clustering is that of *soft clustering* where an instance is allowed to belong to different clusters based on some degree of membership.

Clustering algorithms are based on some *distance function* which determines the cluster an object should be assigned to. A commonly used distance function for numerical instances is the Euclidean distance. There is also an *evaluation function* that evaluates the goodness of the resulting clustering. Usually such a function aims at minimizing the distance of every data point from the mean of the cluster to which it has been assigned after the clustering process.

Clustering algorithms Due to its broad application areas, the clustering problem has been studied extensively in many contexts and disciplines, including Data Mining. As a result, a large number of clustering algorithms exists in the literature (see [47] for a thorough survey). Although the different algorithms use a variety of cluster definitions, they can be categorized into [39]:

1. *Partitioning methods* that partition the dataset into k groups/ clusters each one represented through a centroid like done in k-means algorithm [39] or through a medoid like done in k-medoids [39]. The number of clusters, k, is defined by the user. An example of k–means results is depicted in Figure 2.3.

2. *Hierarchical methods* that create a hierarchical decomposition of the dataset called dendrogram. Such a decomposition might

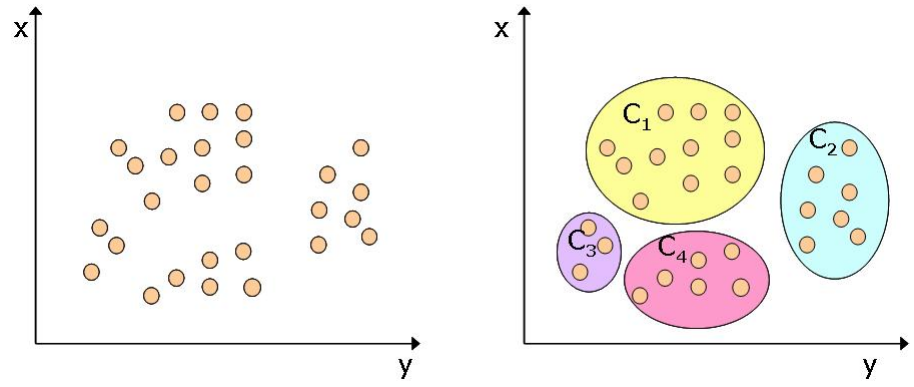


Figure 2.3: An example of a small dataset (left) and the resulting k-means clustering for $k = 4$ (right)

be formed either in a bottom-up or in a top-down fashion, resulting in *agglomerative hierarchical methods* or *divisive hierarchical methods*, respectively. In both cases, a distance function between clusters is required. Different distance functions can be used like single linkage, complete linkage, average linkage or centroids distance [39]. An example of a clustering dendrogram is depicted in Figure 2.4.

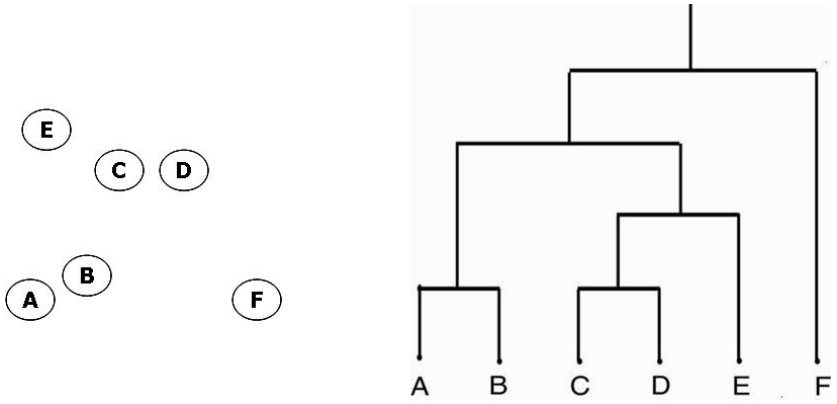


Figure 2.4: An example of a small dataset (left) and the resulting dendrogram (right)

3. *Density-based methods* that continue to grow a cluster as long as the density, i.e., the number of data points, in its "neighborhood" exceeds some threshold. In this category belongs the DBSCAN algorithm [27], some examples of which are depicted in Figure 2.5.

4. *Grid–based methods* that quantize the object space into a finite

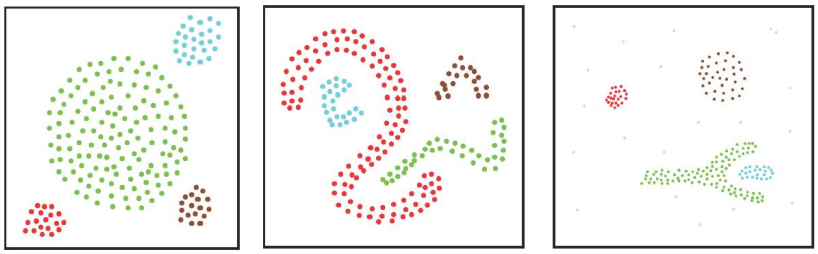


Figure 2.5: Examples of the DBScan algorithm

number of cells that form a grid structure. Algorithms of this category are STING [98] and CLIQUE [4].

5. *Model-based methods* that hypothesize a model for each of the clusters and find the best fit of the data to the given model. Statistical approaches, like the COBWEB algorithm [39], and neural network approaches [39] are the two major approaches of this category.

Cluster representation The *extensional description of a cluster* is straightforward; it simply consists of an explicit enumeration of the data instances that fall into the cluster boundary. Thus, the extensional description is common for the different cluster types.

The *intensional description of a cluster* though, depends on the specific cluster type, even on the specific characteristics of the clustering algorithm. For example, in case of a partitioning clustering, the cluster can be defined by its center, as in k-means, or by its centroid, as in k-medoids. In case of a density based clustering, the cluster can be described in terms of its probability density function. Note, however, that there are algorithms like the hierarchical ones, where some intensional description of the structure of the generated clusters is not available. In this case, only the extensional description of the clusters in terms of their data members is available.

Regarding the *measure component* of a cluster, there are several alternative measures that could be adopted like, for example, the cluster support (i.e., the percentage of dataset instances that fall into the cluster) or the intra-cluster distance (i.e., the average distance between cluster members) or the average distance of the cluster members from its centroid or medoid.

In order to display the extensional-intensional description of a cluster, we depict in Figure 2.6 an example of a k-means clustering. Let us consider the left cluster of this figure: Its extensional description consists of an enumeration of its data members, i.e.,

{(3,4), (3,8), (4,5), (4,7), (2,6)}, whereas its intensional description consists of the cluster centroid, $(3.2, 6)$, and, say, the number of object-members, 5.

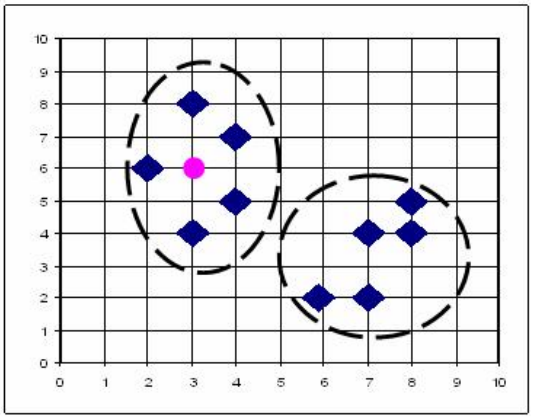


Figure 2.6: An example of extensional-intensional cluster description

2.5 Frequent itemsets and association rules

The *Frequent Itemset Mining (FIM)* problem is a core problem in many data mining tasks, such as association rules, correlations, sequences and episodes. The original motivation for frequent itemsets mining came from the need to analyze retail transaction data in order to find items that are frequently purchased together. Frequent itemsets mining and association rules mining are strongly related to each other since frequent itemsets mining is the first step towards association rules mining.

For the definition of the FIM problem, we follow the work by Agrawal et al. [5]: Let I be a finite set of distinct items and D be a database of transactions where each transaction T contains a set of items, $T \subseteq I$. An example of a transaction database is depicted in Table 2.2.

Transaction ID	**Transaction Items**
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Table 2.2: A sample transaction database D

An *itemset* X is a non-empty lexicographically ordered set of items, $X \subseteq I$. If X contains k items, it is called *k-itemset*. The frequency of X in D, equals to the number of transactions in D that contain X, i.e., $fr_D(X) = |\{T \in D : X \subseteq T\}|$. The percentage of transactions in D that contain X, is called the *support* of X in D, $supp_D(X) = \frac{fr_D(X)}{|D|}$ (Sometimes, the support of an itemset X is defined as the absolute number of transactions in D that contain X, i.e., $supp_D(X) = fr_D(X)$). A *frequent itemset* is an itemset with support greater than or equal to a user-specified minimum support threshold σ called *minSupport*, i.e., $supp_D(X) \geq \sigma$. Two itemsets belong to the same equivalence class F_k if they share a common $k-1$ length prefix. The FIM problem is defined as finding the set $F_\sigma(D)$ of all itemsets X in D that are frequent with respect to a given *minSupport* threshold σ. Let $F_\sigma(D)$ be the set of frequent itemsets extracted from D under *minSupport* threshold σ.

A set of frequent itemsets (FI) forms the itemset lattice L in which the lattice property holds: an itemset is frequent iff all of its subsets are frequent. Considering $minSupport = 2$ and the database example of Table 2.2, the corresponding frequent itemset lattice is depicted in Figure 2.7.

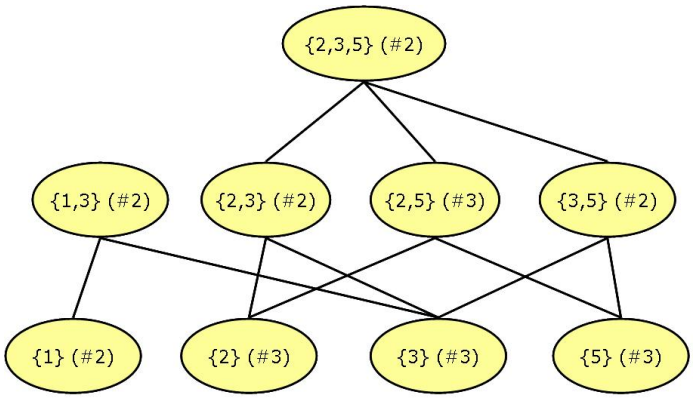


Figure 2.7: An example of a frequent itemset lattice

Itemset representation The *extensional description of an itemset* consists of an enumeration of the transactions that support the specific itemset. The *intensional description of an itemset* consists of the set of items that form it (this comprises the *structure component*) and of its support (this comprises the *measure component*).

As an example, let us consider the itemset $(\{1,3\}, \#2)$ of Figure 2.7. Its extensional description consists of the transactions that

support it, thus it is $\{100, 300\}$ (cf. Table 2.2). Regarding its intensional description, the structure component consists of the items $\{1, 3\}$, whereas the measure component, namely the support, equals to 2.

Association Rules Mining (ARM) The *Association Rules Mining problem* problem was first introduced by Agrawal and Swami [5] and was mainly motivated by the market basket analysis domain. It is defined as follows: Let D be a database of transactions, where each transaction consists of a set of distinct items I, called itemsets. An *association rule* is a implication of the form $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$ (X and Y are itemsets). X is called the head (or antecedent or left hand side) of the rule, whereas Y is called the body (or consequent or right hand side) of the rule. The rule is associated with a *support* s and a *confidence* c. A rule $X \rightarrow Y$ is said to have support s, if $s\%$ of the transactions in D contain $X \cup Y$, whereas it is said to have confidence c, if $c\%$ of the transactions in D that contain X also contain Y. A rule is *interesting* or *strong* if its support and threshold exceed some user specified thresholds.

The Association Rules Mining problem consists of two steps: First, the set of frequent itemsets is extracted, and used as input to the second step so as to extract the association rules. Association rules provide some additional information comparing to frequent itemsets, since they determine whether a set of itemsets implies another set of itemsets.

Let us now return to the extensional – intensional description of association rules: the *extensional description of a rule* consists of an enumeration of the transactions that contribute to its generation. For example, the extensional description of the rule $2 \rightarrow 5$ is the set of instances $\{200, 400\}$ from Table 2.2. The *intensional description of a rule* consists of its head and body (structure component) and its support and confidence (measure component). In our example, $head$=$\{2\}$, $body$=$\{5\}$, $support = 50\%$ and $confidence = 100\%$.

2.6 Summary

In this chapter, we presented some of the most popular pattern types, namely decision trees, clusters/clusterings and frequent itemsets/association rules. For the description of patterns, we employed the extensional-intensional description schema. Based on this schema, a pattern can be described in terms of the data from which it has

been generated (extensional description). A pattern can also be described in terms of the pattern space characteristics (intensional description) through a structure and a measure component; the structure component describes the pattern space, whereas the measure component relates the pattern space to the underlying raw data space. The extensional description comprises an analytical description of the pattern in terms of its data members, whereas the intensional description describes the meaning/concept represented by the pattern and how important is this meaning/concept in the underlying raw data. Note that the definition of the extensional description of a pattern is common for all pattern types (i.e., all patterns can be described in terms of the data which support them), whereas the intensional description depends on the pattern type itself (e.g., the description of a decision tree differs from the description of a frequent itemset).

The definition of a pattern is complete if both it extensional and intensional descriptions are available. However this is not always feasible, due to e.g., privacy or efficiency considerations (recall Chapter 1). In this work, we undertake the problem of pattern comparison by mainly utilizing the intensional description of patterns. There are cases however, where we also utilize the extensional description of patterns, e.g., in monitoring clusters of arbitrary cluster types (hierarchical, partitioning, density based). Although the complete information of a pattern is contained in its extensional-intensional description, there are cases where the intensional description itself still carries much of the information represented by the pattern.

Chapter 3

Assessing Dissimilarity between Arbitrarily Complex Patterns - the PANDA Framework

In this chapter we present PANDA [1], a both generic and flexible framework for assessing dissimilarity between arbitrary complex patterns. PANDA handles both simple and complex patterns, defined over raw data and over other patterns, respectively.

The chapter is organized as follows: In Section 3.1, some motivating examples are presented, together with the requirements that a generic pattern dissimilarity assessment framework should fulfill. The adopted pattern representation model is described in Section 3.2, whereas the pattern dissimilarity assessment process is described in Section 3.3. In Section 3.4 we describe some implementation details, whereas in Section 3.5, we run PANDA for specific applications. Related work is presented in Section 3.6. In Section 3.7, the basics of the PANDA framework are summarized, whereas in Section 3.8, open issues and possible improvements are described.

Index terms pattern comparison, pattern comparison framework, simple patterns, complex patterns, dissimilarity decomposition.

[1] PANDA stands for Patterns for Next Generation Database Systems, an acronym used for IST-2001-33058 project for the European Union, which proposed and studied the PBMS (Pattern Base Management Systems) concept [79].

3.1 Motivation and requirements

In this section we provide some illustrative examples which, in combination with the examples/applications already presented in Section 1.4, motivate the need for a unified and flexible framework for the comparison of patterns. These examples are not fully supported by existing approaches, as it will be clarified in Section 3.6 where the related work is presented.

Example 1 *Consider a telecommunication company providing a package of new generation services with respect to different customer profiles. Let us also consider a decision maker of the company who requests a monthly report that depicts the usage of this package.*

□

Such a report would be far more translatable by the decision maker (e.g., for target marketing), if it was accompanied by the monthly comparison of the classification of the customer profiles that use these services; such a classification could be portrayed through decision tree models, for example.

Example 2 *Consider a manager of a chain of supermarkets who wants to analyze the trends of sales in the shops of the chain. In particular, the manager is willing to understand if there is any shop which sales differ significantly from the sales of the other shops.*

□

This analysis could be carried out in different ways: i) by looking at the sales of single products; ii) by considering sales transactions, i.e., shops baskets; iii) by looking at the products which characterize each customer segment.

Example 3 *A spatial data mining application analyzes the correlation between the density of the population in a town and the number of car accidents. Due to privacy considerations, the raw data are not available, rather only the population distribution and the car accidents distribution in the areas of the town can be used.*

□

Correlation is high if the (spatial) relationships between neighboring areas are taken into account, whereas a much lower correlation is estimated if the two distributions are compared on a per-area basis.

Example 4 *A developer of a copy detection system has to experiment with different techniques in order to select the most effective technique for comparing multimedia documents given a feature-based representation of the documents (e.g., a list of weighted keywords for the text, the color distribution of the images, etc.).*

□

She/he needs to setup a set of methods that consider all these features and return a score that evaluates how similar two documents are.

Example 5 *A portal web site is characterized by high diversity since it covers a wide variety of topics. These topics are usually predefined and the users can just post articles on them. In practise, topics might be close to each other or some post might refer to more than one topic. Consider for example the topics "Databases" and "Data Mining' and a post regarding "Data Mining in Large Databases". In this case, it is not so obvious for the user where her/his post better fits, so she/he might assign it randomly in one of the two topics or in both of them. As a result, the portal might consist of topics that overlap with respect to their contents.*

□

To better serve users needs (e.g., search, navigation), the portal owner should organize the material so as the topic labels are representative of their contents. As a first step, towards this direction, similar topics should be identified. Then, the owner should consider, for example, whether merging of similar topics into a more general and representative topic is meaningful. Another solution would be the redistribution of the posts into the topics so as the topics to be more homogenous with respect to their contents and more precise with respect to their title/label.

The above examples suggest that a framework for pattern comparison should satisfy the following basic requirements:

General applicability: The framework should be applicable to arbitrary types of patterns, such as association rules, histograms and graphs. At the same time, it should not limit the complexity of the considered patterns. Example 5 regarding the comparison of web sites raises such a requirement since a web site comprises a complex structure organized in terms of topics and articles.

Flexibility: The framework should allow for the definition of alternative dissimilarity functions, even for the same pattern type. Indeed, personal preferences and specific constraints regarding the dissimilarity assessment should be easily specifiable in the framework[2]. This requirement is portrayed in Example 4, where the user has to experiment with many configurations in order to decide the one that is most suitable to her/his needs.

Efficiency: It should be possible to define the dissimilarity between patterns without the need to access the underlying raw data. The intuition behind this requirement is that the connection to the raw data might not always be available due to efficiency considerations (like in Example 2) or privacy considerations (like in Example 3).

Simplicity: The framework should be built upon a few basic concepts, so as its application is clear for the end user. Such a requirement also ensures generality and extensibility for the framework.

3.2 Pattern representation in PANDA

Our approach to pattern representation builds upon the logical pattern-base model proposed in [85] in the context of the PANDA project [80]. However, it is only the parts of the model relevant to our purposes that are used.

Following this model, we consider a simple system for defining *pattern types* - the specific choice of types, however, does not influence our reasoning. The model assumes a set of *base types*, like `Int`, `Real`, `Boolean` and `String`, and a set of *type constructors*, like `list(`$<\ldots>$`)`, `set ({`$\ldots$`})`, `array ([`$\ldots$`])` and `tuple ((`$\ldots$`))`. Let us call $\mathcal{T}$ the *set of types* including all the base types and all the types that can be derived from them through repeated application of the type constructors. Types to which a (unique) name is assigned are called *named types*. Some simple examples of types are:

`{Int}`	(set of integers)
`XYPair = (x:Int,y:Int)`	(named tuple type with attributes `x` and `y`)
`<XYPair>`	(list of `XYPair`s)

Definition 3 (Pattern type) *A pattern type is a named pair, $PT = (SS, MS)$, where SS is the structure schema and MS is the measure*

[2]Note here that dissimilarity is intuitive by its nature, thus the end user should be able to adjust the dissimilarity assessment to her/his specific needs.

schema. Both SS and MS are types in $\mathcal{T}$. A pattern type PT is called `complex` *if its structure schema SS includes another pattern type, otherwise PT is called* `simple`*.*

The *structure schema SS* defines the pattern space by describing the structure of the patterns which are instances of the particular pattern type. The complexity of the pattern space depends on the expressiveness of the typing system $\mathcal{T}$. The *measure schema MS* describes measures that relate patterns to the underlying raw data, i.e., they quantify how well patterns represent these data. It is clear that the complexity of measures also depends exclusively on $\mathcal{T}$.

A pattern is an instance of a pattern type, thus it instantiates both the structure and the measure schemes of the corresponding pattern type. Assuming that each base type B is associated with a set of values $dom(B)$, it is immediate to define values for any type in $\mathcal{T}$. Depending on its pattern type, a pattern might be either simple or complex.

Definition 4 (Pattern) *Let $PT = (SS, MS)$ be a pattern type. A* pattern *p, instance of PT, is defined as $p = (s, m)$, where p is the pattern identifier, s (the* structure *of p, also denoted as p.s) is a value for type SS, and m (the* measure *of p, also denoted as p.m) is a value for type MS.*

In order to show the flexibility of this representation model, we provide three different representation schemes for clusters. We choose clusters for our displaying purposes, because different clustering algorithms result in a variety of cluster descriptions and thus, from a modeling view, clusters are of special interest. There are several alternative representations for clusters (c.f. Section 2.4): hierarchical clustering algorithms describe clusters as sets of objects, metric space algorithms, like k-means, describe clusters as geometrical shapes, density based algorithms, like EM, describe clusters through a probability density function and so on [39]. What is described below does not exhaust possible representations for cluster models. Rather, the examples below only aim at demonstrating the functionality of our PANDA framework and should not be considered as compulsory representation schemes for clusters.

We provide three candidate modeling schemes for: (a) clusters derived through some *metric space clustering algorithm* like k-means, (b) clusters derived through some *density based clustering algorithm* like EM and (c) clusters derived through some *hierarchical clustering algorithm*. In the first case, clusters are modeled as spheres (we call them "Euclidean clusters" because of the metric space), in the

second case clusters are described through some probability density function (we call them "Density-based clusters"), whereas in the third case clusters are modeled as sets of objects (we call them "Hierarchical clusters"). For each one of these types, we also describe an indicative complex pattern type.

Example 6 (Euclidean cluster and clustering types) *A Euclidean cluster in a D-dimensional space, such as the ones obtained from the k-means algorithm [39], can be modeled through a center and a radius, which form the* structure schema *of the cluster. For the* measure schema*, one could consider the cluster support, i.e., the fraction of objects that fall into the cluster, and the average intra-cluster distance. That is:*

$$\begin{aligned}&\texttt{EuclideanCluster} = \\ &\qquad (SS : \texttt{(center:[Real]}_1^D\texttt{,radius:Real)},\\ &\qquad MS : \texttt{(supp:Real,avgdist:Real)})\end{aligned}$$

Assuming $D = 3$, a possible instance of this type could be as follows:

$$\begin{aligned}&p407 = \\ &\quad (s : (\texttt{center} = [0.75, 1.25, 0.46], \texttt{radius} = 0.24),\\ &\quad m : (\texttt{supp} = 0.13, \texttt{avgdist} = 0.17))\end{aligned}$$

A PartitioningEuclideanClustering pattern can be defined as the composition of cluster patterns. In particular, for a hard partitioning clustering algorithm like k-means, a EuclideanClustering pattern can be simply modeled as a set of EuclideanCluster patterns with no measure:

$$\begin{aligned}&\texttt{PartitioningEuclideanClustering} = \\ &\qquad (SS : \{\texttt{EuclideanCluster}\},\\ &\qquad MS : \bot)\end{aligned}$$

where $\bot$ denotes the null type*.*

□

Example 7 (Density–based cluster and clustering types) *A density based clustering algorithm, like EM [39], produces clusters that can be described through a (say) Gaussian density function, thus they can be represented through a mean and a standard deviation in each dimension; these comprise the* structure schema *of the cluster.*

As for the measure schema*, the cluster support, i.e., the amount of the population covered by the cluster, could be considered. That is:*

$$\begin{aligned}&\texttt{DensityBasedCluster} = \\ &\qquad (SS : \texttt{(mean:[Real]}_1^D\texttt{,stdDev:[Real]}_1^D\texttt{)},\\ &\qquad MS : \texttt{(supp:Real)})\end{aligned}$$

Assuming $D = 2$, a possible instance of this type could be as follows:

$$\begin{aligned}&p111 = \\ &\quad (s : (\texttt{mean} = [15.5, 41.4], \texttt{stdDev} = [3.6, 4.7]),\\ &\quad m : (\texttt{supp} = 0.33)\end{aligned}$$

A DensityBasedClustering pattern can be modeled as a set of DensityBasedCluster component patterns with no measure:

$$\begin{aligned}&\texttt{PartitioningDensityBasedClustering} = \\ &\qquad (SS : \{\texttt{DensityBasedCluster}\},\\ &\qquad MS : \bot)\end{aligned}$$

where $\bot$ denotes the null type*.*

□

Example 8 (Hierarchical cluster and clustering types) *A cluster obtained by a hierarchical algorithm is usually described as the set of its object members (structure schema)*[3]*. For the measure schema, the cardinality of this set could be considered. That is:*

$$\begin{aligned}&\texttt{Point} = \\ &\quad (SS : \texttt{(coords:[Real]}_1^D\texttt{)},\\ &\quad MS : \bot)\end{aligned}$$

$$\begin{aligned}&\texttt{HierarchicalCluster} = \\ &\qquad (SS : \{\texttt{Point}\},\\ &\qquad MS : \texttt{(supp:Real)})\end{aligned}$$

A HierarchicalClustering pattern can be modeled as a set of HierarchicalCluster component patterns with no measure:

$$\begin{aligned}&\texttt{HierarchicalClustering} = \\ &\qquad (SS : \{\texttt{HierarchicalCluster}\},\\ &\qquad MS : \bot)\end{aligned}$$

[3]Note that such a description holds only for the cases where raw data access is available.

□

Note again that the distinction between simple and complex patterns relies on whether their structure schema is defined upon other patterns or not. In the former case, patterns are complex, e.g., a cluster of association rules. The simplest form of complex patterns contains a 2-level hierarchy, i.e., the complex pattern and its component simple patterns, like in the clustering examples presented above. The definition of complex patterns, however, allows for multiple nesting within the structure schema, thus hierarchies of any level are supported. An indicative example is that of a Web site: typically, the contents of a Web site are organized into categories, each category consists of a set of pages, and each page is described through a set of keywords. The Web site example could be placed within the simple-complex patterns rationale of PANDA as follows: A *Web site* represents a complex pattern composed of other patterns, namely the categories. A *category* also represents a complex pattern composed of other patterns, namely the Web pages. A *Web page* also represents a complex pattern which consists of other patterns, namely the keywords. A *keyword*, finally, represents a simple pattern, and could be described through a name (e.g., "computer") and a weight (e.g., 50%) that quantifies the importance or frequency of the keyword in the Web page.

As soon as some pattern dissimilarity assessment problem can be expressed in the rationale of simple-complex patterns, it can be directly handled by our PANDA framework.

Table 3.1 summarizes the symbols used throughout the chapter.

Symbol	**Description**
p	simple pattern
cp	complex pattern
dis_{struct}	evaluates the dissimilarity of the structure components
dis_{meas}	evaluates the dissimilarity of the measure components
Combiner	combines the dissimilarities of the structure and measure components
Matching type	establish how the component patterns can be matched
Aggregation logic	aggregates the scores of the matched component patterns

Table 3.1: List of symbols for Chapter 3

3.3 The PANDA framework for assessing dissimilarity between patterns

In this section, we describe the PANDA framework for assessing dissimilarity between two patterns p_1, p_2, instances of the same pattern type PT. From Section 3.2, it follows that the complexity of PT can widely vary and is only restricted by the adopted typing system $\mathcal{T}$.

The basic principles upon which we built our framework are as follows:

1. The dissimilarity between two patterns should yield a *score value*, normalized in the [0..1] range (the higher the score, the higher the dissimilarity).
2. The dissimilarity between two complex patterns should (recursively) depend on the dissimilarity of their component patterns.
3. The dissimilarity between two patterns should be evaluated by taking into account both the dissimilarity of their structures and the dissimilarity of their measures.

The first principle, normalization of the dissimilarity scores, offers a better and more intuitive interpretation of the results.

The second principle provides the whole flexibility of our PANDA framework. Note that for the case of complex patterns, one could devise arbitrary models for their comparison. However, it is useful and, at the same time, sufficient for practical purposes, to consider divide and conquer solutions that decompose the "difficult" problem of comparing complex patterns into simpler sub–problems like those of comparing simple patterns, and then "smartly" aggregate the so-obtained partial solutions into an overall score.

The third principle is a direct consequence of our approach that allows for arbitrarily complex structures in patterns. Since the structure of a complex pattern might include measures of its component patterns, neglecting the structural dissimilarity could easily result in misleading results, like comparing two semantically contradictory patterns. To deal with such cases, we introduce the notion of *combining function* that combines the structure and measure dissimilarity scores only if there exists some structural compatibility between the patterns. Another motivation underlying this principle arises from the need of building an *efficient* framework, which does not force accessing the underlying dataset(s) in order to determine the dissimilarity of two patterns, e.g., in terms of their common instances. To this end, we utilize all the parts of information that are

available in the pattern space, namely the structural description of the pattern space and the qualitative measures that associate the pattern space with the underlying raw data space.

Furthermore, for each pattern type of interest at least one *dissimilarity operator* should be defined. At the same time, however, it should be possible to define multiple operators for the same pattern type; this might be required in order to capture different aspects of the patterns (e.g., only the structure dissimilarity or both structure and measure dissimilarity might be considered) or to allow different evaluation of the same aspects (e.g., for the measure dissimilarity, one could choose between the absolute and the relative difference functions).

In the next subsections, we first describe how the above-stated principles can be applied to the basic case of simple patterns (Section 3.3.1) and then, we show how they can be generalized to the general case of complex patterns (Section 3.3.2).

3.3.1 Dissimilarity between simple patterns

The dissimilarity between two simple patterns p_1, p_2 of a same simple pattern type PT is based on three key ingredients:

- a *structure dissimilarity* function, dis_{struct}, that evaluates the dissimilarity of the structure components $p_1.s$ and $p_2.s$,
- a *measure dissimilarity* function, dis_{meas}, that evaluates the dissimilarity of the corresponding measure components $p_1.m$ and $p_2.m$, and
- a *structure & measure combining* function also called *Combiner*, $Comb$, that aggregates the structure and measure dissimilarity scores into an overall score that reflects the total dissimilarity between the compared (simple) patterns.

Consequently, the dissimilarity between two patterns is defined as:

$$dis(p_1, p_2) = Comb(dis_{struct}(p_1.s, p_2.s), dis_{meas}(p_1.m, p_2.m)) \quad (3.1)$$

The overall schema of the dissimilarity assessment process is depicted in Figure 3.1:

If patterns p_1 and p_2 share the same structure, then $dis_{struct}(p_1.s, p_2.s) = 0$ and their dissimilarity solely depends on the dissimilarity of their measures. In the general case, however, the patterns to be compared may have different structures and two alternatives exist:

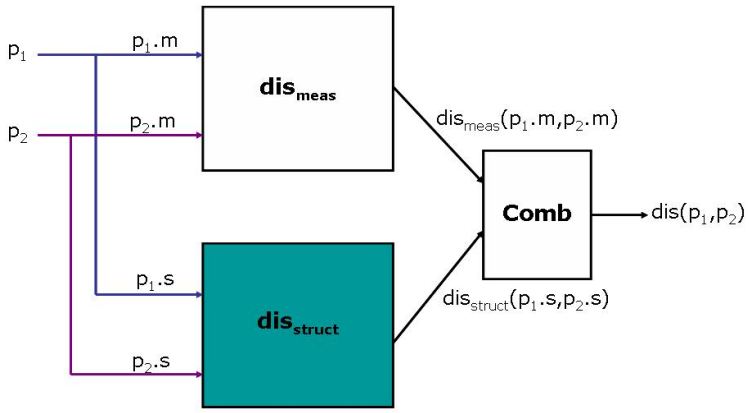


Figure 3.1: Assessment of dissimilarity between (simple) patterns

- The structure components are somewhat "compatible" and the structure dissimilarity score $dis_{struct}(p_1.s, p_2.s)$ is interpreted as the "additional dissimilarity" cost one wants to charge with respect to the case of identical structures.
- The structure components are totally unrelated (in a sense that depends on the application at hand), i.e., $dis_{struct}(p_1.s, p_2.s) = 1$. In this case, regardless of the measure dissimilarity score, we also require that the overall dissimilarity score should be the maximum one, i.e., $dis(p_1, p_2) = 1$. This restriction is enforced in order to prevent cases where two completely different patterns might be considered somewhat similar due to low differences in their measures.

Note that in PANDA the structure components play the key role during the pattern dissimilarity assessment process. This is reasonable, since in order to compare the "qualitative" information of two patterns (which is captured through their measure components) there should be some kind of compatibility between their structures, i.e., the patterns to be compared should be somewhat relevant.

Below we present two examples on simple pattern comparison: comparison between frequent itemsets (relevant to the motivating Example 2 of Section 3.1) and comparison between keywords (relevant to the motivating Example 4 of Section 3.1). At first, each pattern type is expressed in terms of a structure and a measure component and then, the PANDA blocks for dissimilarity assessment between simple patterns, namely dis_{struct}, dis_{meas} and $Comb$, are properly instantiated.

Example 9 (Itemsets) *Consider a dataset D described in terms of a relation $R = \{I_1, I_2, \ldots, I_m\}$. An itemset IS derived from D can be modeled as a simple pattern whose* structure *consists of a set of items from R, i.e., $IS.s = \{I_1, \ldots, I_m\}$, whereas its* measure *is the support of the itemset, i.e., the fraction of tuples in D that contain all the items in $IS.s$, thus $IS.m = (\texttt{supp:Real})$.*

Let us now consider the problem of comparing two itemsets IS_1, IS_2 derived from the above settings, where:

$$IS_1 = (\{\textit{bread, honey, milk}\},\ 0.1) \textit{ and } IS_2 = (\{\textit{butter, milk}\},\ 0.2)$$

- ***Structure dissimilarity:*** *For the structure dissimilarity function, we could employ the overlap of their structures, i.e., how many items the two itemsets have in common:*

$$dis_{struct} = 1 - \frac{IS_1.s \cap IS_2.s}{IS_1.s \cup IS_2.s} \tag{3.2}$$

  In our example, this equals to $1 - \frac{1}{4} = 0.75$.

- ***Measure dissimilarity:*** *For the measure dissimilarity function, we could consider the absolute difference of their measures:*

$$dis_{meas}(IS_1.m, IS_2.m) = |IS_1.m.supp - IS_2.m.supp| \tag{3.3}$$

  In our example, this equals to $|0.1 - 0.2| = 0.1$.

- ***Combiner:*** *Finally, for the combining function we could consider the average value of the structure and measure dissimilarity scores, as far as the itemsets structures are somewhat relevant, i.e.,*

$$Comb(dis_{struct}, dis_{meas}) = \begin{cases} 1 & , \textit{ if } dis_{struct} = 1 \\ \text{avg}(dis_{struct}, dis_{meas}) & , \textit{ otherwise} \end{cases} \tag{3.4}$$

  In our example, this equals to $(0.75 + 0.1)/2 = 0.425$.

  If, at an alternative scenario, the application expert considers that structure dissimilarity is more important than measure dissimilarity, e.g., in a 3:1 rate, the combining function would have weighted average form:

$$Comb(dis_{struct}, dis_{meas}) = \begin{cases} 1 & , \textit{ if } dis_{struct} = 1 \\ \frac{3}{4} * dis_{struct} + \frac{1}{4} * dis_{meas} & , \textit{ otherwise} \end{cases} \tag{3.5}$$

  In our example, this equals to $\frac{3}{4} \cdot 0.75 + \frac{1}{4} \cdot 0.1 = 0.5875$.

□

Example 10 (Weighted keywords) *A weighted keyword extracted from a document can be represented as a pair* $(s = t, m = w)$*, where* t *is the keyword (structure component) and* $w \in (0, 1]$ *is its normalized weight in the document (measure component). Consider two weighted keywords:* $k_1 = (s = t_1, m = w_1)$ *and* $k_2 = (s = t_2, m = w_2)$*. As with the previous example, in order to define the dissimilarity between two keywords, we should define how their structure and measure components are compared and how the resulting scores are combined.*

- ***Structure dissimilarity:*** *If the two keywords are identical then,* $dis_{struct}(k_1, k_2) = 0$*. When* $t_1 \neq t_2$*, two alternatives are conceivable: if some information about the semantics of the keywords is available, such as a thesaurus or a hierarchical hypernymy/hyponymy ontology, such as WordNet [99], then* $dis_{struct}(k_1, k_2)$ *could reflect the "semantic distance" between* t_1 *and* t_2*, e.g., by considering the normalized depth of their least common ancestor (LCA) [12]. If, on the other hand, no such information is available, one should conclude that* $dis_{struct}(k_1, k_2) = 1$*.*

- ***Measure dissimilarity:*** *An obvious choice for the measure dissimilarity function is the absolute measure difference, thus* $dis_{meas}(w_1, w_2) = |w_1 - w_2|$*.*

- ***Combiner:*** *Finally, a suitable combining function for this example might be the algebraic disjunction of the corresponding structure and measure dissimilarity scores:*

$$\begin{aligned} dis(k_1, k_2) &= dis_{struct}(k_1.s, k_2.s) + dis_{meas}(k_1.m, k_2.m) \\ &\quad - dis_{struct}(k_1.s, k_2.s) * dis_{meas}(k_1.m, k_2.m) \\ &= dis_{struct}(t_1, t_2) + dis_{meas}(w_1, w_2) - dis_{struct}(t_1, t_2) * dis_{meas}(w_1, w_2) \end{aligned} \tag{3.6}$$

where $0 \leq dis_{struct} \leq 1$ *and* $0 \leq dis_{meas} \leq 1$*. This equation correctly yields* $dis(k_1, k_2) = 1$ *when* $dis_{struct}(t_1, t_2) = 1$*, and* $dis(k_1, k_2) = dis_{meas}(w_1, w_2)$ *when* $dis_{struct}(t_1, t_2) = 0$*. Furthermore, for any fixed value of* $dis_{meas}(w_1, w_2)$*,* $dis(k_1, k_2)$ *increases monotonically with the semantic distance of* t_1 *and* t_2*, i.e.,* $dis_{struct}(t_1, t_2)$*.*

□

The above examples indicate the flexibility of the PANDA framework: both structure and measure dissimilarity functions, as well as the combining function are fully modular and can be easily adapted to specific user requirements. Due to the possibility of different instantiations of these components, several dissimilarity configurations might arise within PANDA facilitating the end user to choose the best configuration for her/his needs.

3.3.2 Dissimilarity between complex patterns

Although, in line of principle, one could define simple patterns with arbitrarily complicated structural components, this would necessarily force dissimilarity functions to be complex and hardly reusable. Among the requirements stated in Section 3.1, this "monolithic" approach would only comply with the efficiency requirement, failing to address the other ones. In PANDA we pursue a modular approach that, by definition, is better suited to guarantee flexibility, simplicity, and reusability. Moreover, as it will be discussed later, this approach does not rule out the possibility of efficient implementations.

Coherently with the PANDA pattern model described in Section 3.2, the dissimilarity of complex patterns is conceptually evaluated in a bottom-up fashion based on the dissimilarities of their component patterns. The structure of complex patterns plays here a major role, since the comparison process relies on the structure components of the patterns to be compared.

Without loss of generality, in what follows we assume that the component patterns, $p^1, p^2, \ldots, p^N$, of a complex pattern cp completely describe the structure of cp (no additional information is present in $cp.s$) and that they form a set ($cp.s = \{p^1, p^2, \ldots, p^N\}$).

The *structural dissimilarity* between two complex patterns ($cp_1 = \{p_1^1, p_1^2, \ldots, p_1^{N_1}\}$) and ($cp_2 = \{p_2^1, p_2^2, \ldots, p_2^{N_2}\}$) can be easily adapted to specific needs/constraints by acting on two fundamental abstractions, namely:

- the *matching type*, which is used to establish how the component patterns of cp_1, cp_2 can be matched, and

- the *aggregation logic*, which is used to combine the dissimilarity scores of the matched component patterns into a single value representing the total dissimilarity score between the structures of the complex patterns.

3.3.2.1 Matching type

As already stated, a complex pattern can be eventually decomposed into a number of component patterns. Thus, when comparing two complex patterns cp_1, cp_2, we need a way to associate their component patterns. To this end, we introduce the *coupling type*, which defines how the component patterns of cp_1 (cp_2) are matched to the component patterns of cp_2 (cp_1, respectively), taking into account specific user/application requirements.

A matching between the complex patterns cp_1, cp_2 can be represented through a *matrix* $\mathbf{X}_{N_1 \times N_2} = (x_{ij})$; each element $x_{ij} \in [0, 1]$ $(i = 1, \ldots, N_1;\ j = 1, \ldots, N_2)$ represents the (amount of) "matching" between the i-th component pattern of cp_1, p_1^i, and the j-th component pattern of cp_2, p_2^j. Such a matrix is depicted in Figure 3.2.

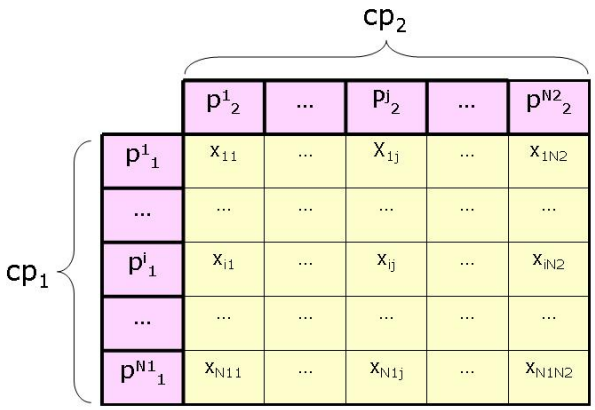
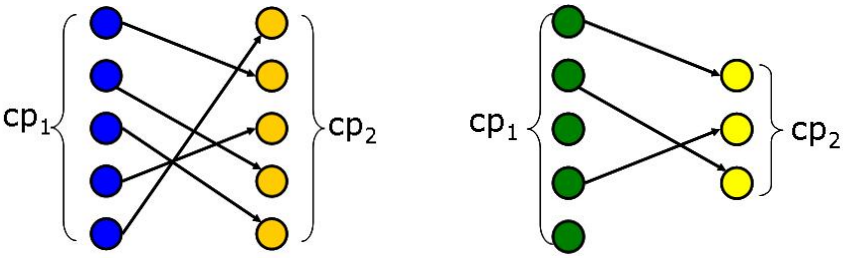


Figure 3.2: The matching matrix between complex patterns cp_1, cp_2

Essentially, a *matching type* is a set of constraints on the x_{ij} coefficients so that only some of all the possible matchings are valid. Below we describe some examples of matching types:

- **1–1 matching:** In this case, each component pattern of cp_1 can be matched to at most one component pattern of cp_2 and vice versa. If $N_1 = N_2$, a full matching exists between cp_1 and cp_2 like in Figure 3.3 (a). A partial matching might occur if $N_1 \neq N_2$ like in Figure 3.3 (b). Formally, the 1–1 matching corresponds to the following set of constraints over the elements



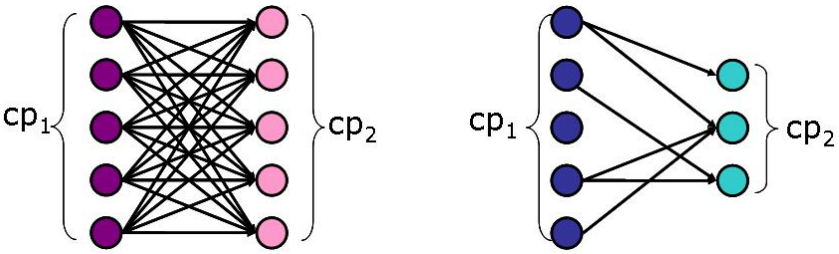
(a) Full matching (b) Partial matching

Figure 3.3: 1–1 Matching

$x_{i,j}$ $(i = 1, \ldots, N_1; j = 1, \ldots, N_2)$ of matrix $\mathbf{X}_{N_1 \times N_2}$:

$$\begin{aligned} x_{ij} &\in \{0, 1\} \\ \sum_{i=1}^{N_1} x_{ij} &\leq 1, \quad \forall j \\ \sum_{j=1}^{N_2} x_{ij} &\leq 1, \quad \forall i \\ \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} x_{ij} &\leq \min\{N_1, N_2\}, \quad \forall i, j \end{aligned} \tag{3.7}$$

- **N–M matching:** In this case, each component pattern of cp_1 can be matched to more than one component patterns of cp_2 and vice versa. In the extreme case, each component pattern of cp_1 is matched to every component pattern of cp_2 like in Figure 3.4 (a). In the general case, however, partial matching might occur like in Figure 3.4 (b). Formally, the N–M match-



(a) Full matching (b) Partial matching

Figure 3.4: N–M Matching

ing corresponds to the following set of constraints over the el-

ements $x_{i,j}$ $(i = 1, \ldots, N_1; j = 1, \ldots, N_2)$ of matrix $\mathbf{X}_{N_1 \times N_2}$:

$$\begin{aligned} x_{ij} &\in \{0, 1\} \\ \sum_{i=1}^{N_1} x_{ij} &\leq N_2, \quad \forall j \\ \sum_{j=1}^{N_2} x_{ij} &\leq N_1, \quad \forall i \\ \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} x_{ij} &\leq N_1 * N_2, \quad \forall i, j \end{aligned} \tag{3.8}$$

- **EMD matching:** This matching type, introduced to define the Earth Movers Distance (EMD) [86, 57], differs from the previous matching types in that each component pattern p is also associated with some weight w.

  As such, the complex patterns cp_1, cp_2 become of the form:

  $$cp_1 = \{(p_1^1, w_1^1), (p_1^2, w_1^2), \ldots, (p_1^{N_1}, w_1^{N_1})\}$$
  $$cp_2 = \{(p_2^1, w_2^1), (p_2^2, w_2^2), \ldots, (p_2^{N_2}, w_2^{N_2})\}$$

  and EMD is defined as:

  $$W(cp_1, cp_2, X) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} x_{ij} d_{ij}$$

  where d_{ij} is some measure of dissimilarity between p_1^i and p_2^j.

  Formally, EMD matching corresponds to the following set of constraints over the elements $x_{i,j}$ $(i = 1, \ldots, N_1; j = 1, \ldots, N_2)$ of matrix $\mathbf{X}_{N_1 \times N_2}$:

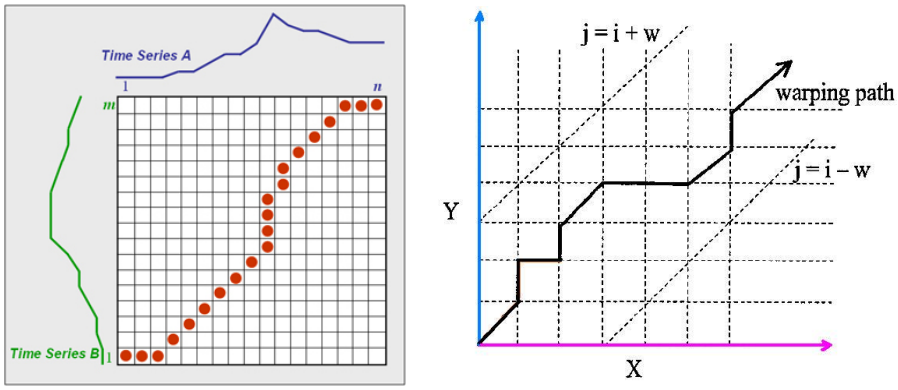
  $$\begin{aligned} x_{ij} &\geq 0 \\ \sum_{i=1}^{N_1} x_{ij} &\leq w_2^j, \quad \forall j \\ \sum_{j=1}^{N_2} x_{ij} &\leq w_1^i, \quad \forall i \\ \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} x_{ij} &= \min(\sum_{i=1}^{N_1} w_1^i, \sum_{j=1}^{N_2} w_2^j) \end{aligned}$$

- **DTW matching:** Dynamic Time Warping (DTW) is a widely used technique for time series similarity assessment [23]. Its peculiarity lies in the fact that it finds the best alignment/matching between two time series X, Y by allowing their local deformations (stretch or shrink) across the time axis; such an alignment is called warping path. The procedure for finding the best alignment involves finding all possible warping paths between X and Y and choosing the one that minimizes their overall distance. DTW is given by:

$$DTW(X,Y) = \min_{\forall X',Y' \ s.t. \ |X'|=|Y'|} L_1(X',Y') \tag{3.9}$$

  where X', Y' are the new sequences resulting from the original sequences X, Y by repeating their elements. A representative example is depicted in Figure 3.5 (a) [23].

  Sometimes, for efficiency issues, a bounded wapping window of size w is defined that controls the distance between a time point of X and its alignment in Y. In this case, for any coefficient $x_{i,j}$ on the warping path, it should hold that $|i-j| \leq w$. Such an example is depicted in Figure 3.5 (b) [23].



(a) Unbounded DTW (b) Bounded DTW

Figure 3.5: Dynamic Time Warping (DTW) Matching

Before we complete the discussion on alternative matching types, we should note that the dissimilarity functions, either explicitly or implicitly, rely on some kind of matching: For example, 1–1 matching is a core issue in graph comparison. Also, during distribution or histogram comparison a 1–1 matching is employed; for example, the Kullback-Leibler (KL) divergence measures the difference between two distributions by comparing, for each "bucket", its probability in

the first and the second distribution [45]. As another example, consider the hierarchical clustering technique: in the complete linkage algorithm a kind of N–M matching is applied between the cluster members so as to decide which clusters to merge/split at the next step of the algorithm [39].

3.3.2.2 Aggregation Logic

In order to compute the total dissimilarity between two complex patterns, the dissimilarity scores of their matched component patterns have to be *aggregated* so as to obtain a single overall dissimilarity score. In general, such an aggregation is achieved by means of an *Aggr* function, which takes as input the matrix $D = (dis(p_1^i, p_2^j))$ of the dissimilarities of the component patterns and a matching matrix X, i.e.,

$$Aggr(D, X)$$

Usually, the x_{ij} coefficients are used to weight the dissimilarities of the component patterns and thus, the above function usually takes the following form:

$$Aggr(dis(p_1^i, p_2^j) \times x_{ij})$$

Among all valid matchings (as specified by the matching type), the rationale is to pick the "best" one, i.e., the one that minimizes the overall dissimilarity score. Thus:

$$dis_{struct}(cp1.s, cp2.s) = \min_X \{Aggr(D, X)\} \tag{3.10}$$

Conceptually, the process followed by PANDA in order to compute the structural dissimilarity between two complex patterns is summarized in Figure 3.6 (it corresponds to the colorful box of Figure 3.1).

The idea is as follows:

- The *"Matcher"* block produces candidate matchings between the component patterns of the complex patterns to be compared; obviously, the constraints imposed by the matching type are taken into consideration.
- The *"Aggr"* block aggregates the scores of the matched component patterns, resulting in a total score for the specific matching.
- The *"Min"* block loops through candidate matchings, choosing the best one, i.e., the one that produces the lower dissimilarity score.

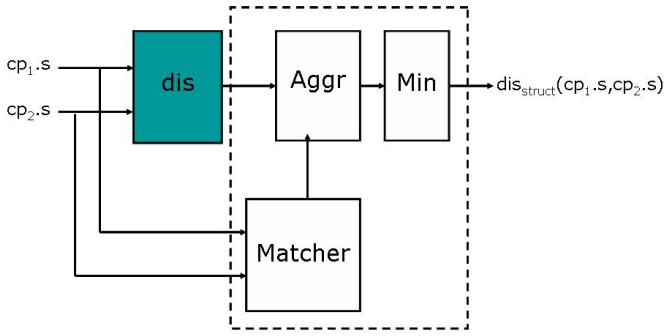


Figure 3.6: Assessment of structural dissimilarity between complex patterns

In case of multi-level aggregations, the dissimilarity block (the colorful box in Figure 3.6) might encompass the recursive computation of dissimilarity between the complex patterns.

Finally, the overall dissimilarity between cp_1, cp_2 is as in Equation 3.1, thus it follows the same rationale as with the simple patterns case.

It should be noted that Figure 3.6 only depicts the main components of our framework for a clearer exposition of our arguments. Actually, we do not require that the dissimilarity is computed in this way. In particular the "Matcher" block does not need to produce all valid matchings (such a solution would be quite inefficient in practice, since it has a prohibitive cost $O(n!)$). Indeed, efficient matching algorithms can be devised for the relevant matching types in order to quickly compute, given the constraints, the solution of the "best matching problem" (this is represented by the dashed box in Figure 3.6). For example, for the 1–1 coupling type, which coincides with the well-known assignment problem in graph theory, the optimal solution is given by the Hungarian algorithm [56], which implements both the "Aggr" and the "Min" block of Figure 3.6. For the same problem however, one could use a Greedy algorithm (which, at each step, makes the most profitable assignment between two component patterns) that does not result in the optimal solution and it only implements the "Aggr" block of Figure 3.6. Hungarian with complexity $O(n^3)$ is more expensive than Greedy with complexity $O(n^2)$. The Hungarian algorithm provides the optimal solution to the assignment problem in contrast to the less expensive Greedy algorithm, which, however, most probably finds a local optimum.

Below we present an example of comparison between complex patterns; in particular, we consider the problem of comparing web sites. As already stated, a web site is a complex pattern composed

of web pages, which in turn are also complex patterns represented as sets of keywords. The keywords finally, play the role of simple patterns. We start with the comparison of web pages and then proceed with the comparison of web sites. The comparison between keywords could be carried out as already described in Example 10.

Example 11 (Web pages) *A web page wp can be modeled as a complex pattern whose structure consists of a set of weighted keywords extracted from wp (c.f. Example 10),* $wp.s = \{(t^1, w^1), \ldots, (t^M, w^M)\}$. *The measure of wp might be its Google PageRank [17],* $wp.m = pr$. *The dissimilarity between two web pages,* wp_1 *and* wp_2*, might take into account only their corresponding keywords, or it might also consider the differences in their PageRank measures [17]. As for the combining function, one could consider the algebraic disjunction of the structure and measure dissimilarity scores. In particular, for the structure dissimilarity between* wp_1, wp_2 *one could employ 1–1 matching between the corresponding component keywords and use the average function to aggregate the scores of the matched component pairs.*

□

Example 12 (Web sites) *Continuing Example 11, a web site ws can be modeled as a complex pattern whose structure consists of a graph, i.e., a set of web pages* $\{wp^1, wp^2, \ldots, wp^N\}$ *that are connected through links of the form* $(wp^i \rightarrow wp^j)$*. The dissimilarity between two web sites,* ws_1 *and* ws_2*, can be computed by considering the subgraph matching (or subgraph isomorphism) problem [96], i.e., finding whether an isomorphism exists among (subgraphs) of* ws_1 *and* ws_2*.*

For instance, let us consider the two web sites in Figure 3.7, where pages of the same borderline (and color) are about the same topic.

If dissimilarity between web pages is computed as in Example 11 and the matched pages are the ones portrayed under the same borderline (and color) in Figure 3.7, where the structure and measure dissimilarity scores between the matched pages are depicted in the right part of this figure, and if also the aggregation function is a simple average, then the overall dissimilarity score between the two

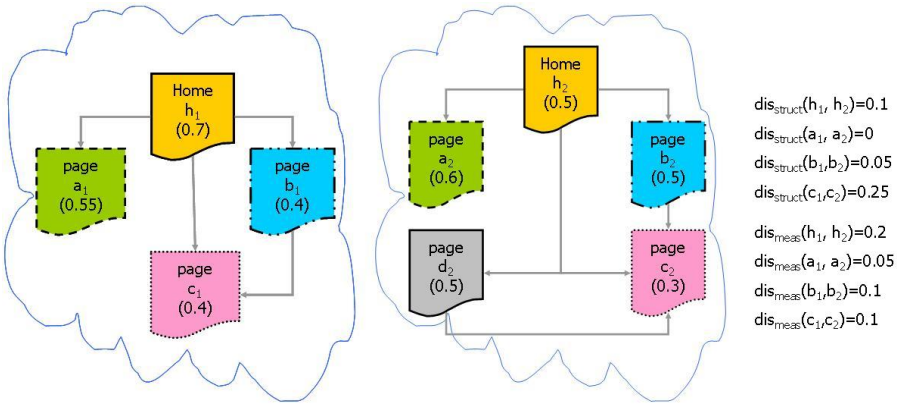


Figure 3.7: Two web sites to be matched (pages of the same borderline (and color) are about the same topic) and the structure and measure dissimilarity scores between the matched pages.

web sites is computed as follows:

$$dis(ws_1, ws_2) = dis_{struct}(ws_1.s, ws_2.s) = \frac{dis(h_1, h_2) + dis(a_1, a_2) + dis(b_1, b_2) + dis(c_1, c_2) + dis(\perp, d_2)}{5} = \frac{(0.1 + 0.2 - 0.1 * 0.2) + (0 + 0.05 - 0 * 0.05) + (0.05 + 0.1 - 0.05 * 0.1)}{5} + \frac{(0.25 + 0.1 - 0.25 * 0.1) + (1)}{5} = \frac{1.8}{5} = 0.36$$

□

A final remark on the design of PANDA frameworkis that, although the comparison between patterns yields a single dissimilarity score, PANDA can also provide detailed information on how the component patterns have been matched, as well as on how much important these matchings are with respect to the global dissimilarity score. Such a knowledge allows the end user to gain insights on the obtained score, e.g., to understand which are the component patterns that mostly contribute to the increase of the pattern dissimilarity score.

3.4 Implementation issues

The PANDA framework for assessing dissimilarity between patterns (either simple or complex) has been implemented in Java and several

sample applications for the comparison of different pattern types have been built around it[4]. In the following, we first provide an overview of the fundamental framework classes (Section 3.4.1) and then we briefly describe the application (Section 3.4.2).

3.4.1 Basic framework classes

The main classes are the generic *Pattern* class and its derivatives: *SimplePattern* class and *ComplexPattern* class.

The *Pattern* class: The core of the framework implementation is the abstract *Pattern* class (Figure 3.8). Each pattern has a unique identifier, *id*, and belongs to a specific pattern type, which can be returned through the *getType()* method.

The *getDissimilarityStructure()* method computes the structure dissimilarity between the specific pattern and another input pattern; both patterns should belong to the same pattern type. Correspondingly, there exists the *getDissimilarityMeasure()* method for assessing the measure dissimilarity between the specific pattern and a given pattern. The *getDissimilarity()* method implements the combining function, *Comb*, i.e., it combines the structure and measure dissimilarity scores into an overall dissimilarity score.

The methods *getType()*, *getDissimilarityStructure()*, *getDissimilarityMeasure()* and *getDissimilarity()* are abstract and should be specified for each class that extends the basic *Pattern* class. The *getDissimilarity()* method uses an object of the abstract class *Combiner* that corresponds to different combining functions.

The *SimplePattern* class: The *SimplePattern* class implements a pattern instance of the simple pattern type and extends the basic *Pattern* class (Figure 3.9).

The different structure dissimilarity functions available for each pattern type are stored in the static member *PossibleStructureDissimilarity*. Several structure dissimilarity functions have been implemented under the package *dissimilarity.structure* including *EuclideanDistance* and *JaccardDistance*.

The different measure dissimilarity functions available for each pattern type are stored in the static member *PossibleMeasureDissimilarity*. Several structure dissimilarity functions have been implemented under the package *dissimilarity.measure* including *AbsoluteDistance* and *RelativeDistance*.

[4]The code is freely available for non–commercial use. The interested users could download it from: http://195.251.230.17/panda/index.html

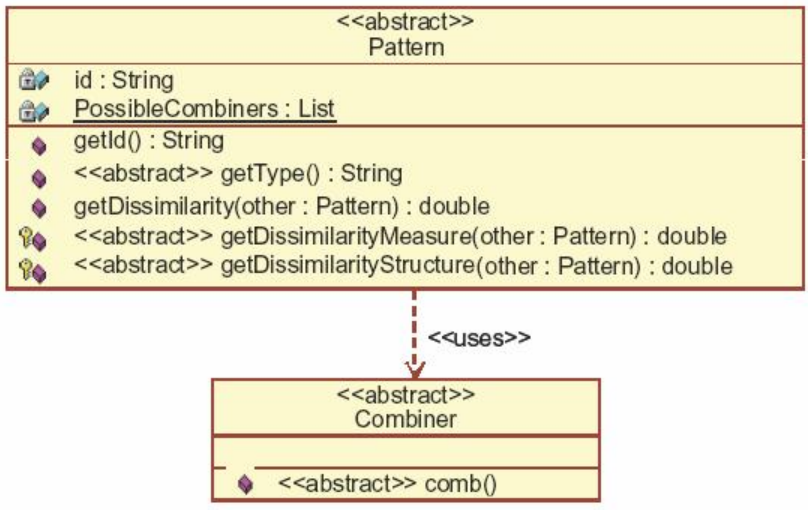


Figure 3.8: The *Pattern* class

The different combining functions available for each pattern type are stored in the static member *PossibleCombiners*. Several combining functions have been implemented under the package *dissimilarity.combiner* including *CombinerStructure* that only considers the structure dissimilarity score and *CombinerWeighted* that considers both structure and measure dissimilarity scores weighted according to user specified weights.

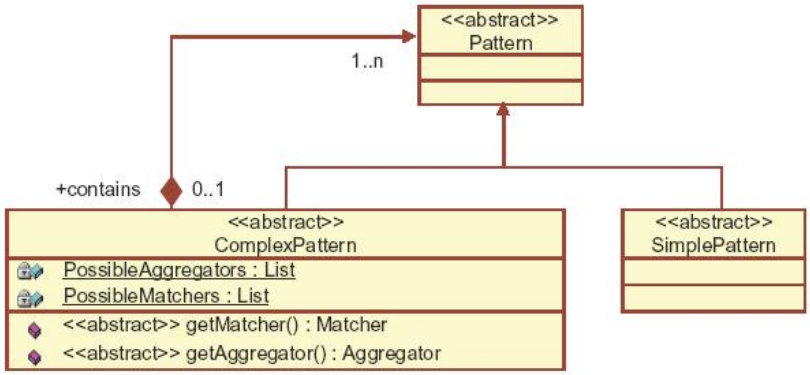


Figure 3.9: The *Pattern* class hierarcy

The *ComplexPattern* class: The *ComplexPattern* class implements a pattern instance of the complex pattern type and extends

the basic *Pattern class* (Figure 3.9). A complex pattern consists of a list of simple patterns, stored in the *PatternList* member.

The different matching types available for each pattern type are stored in the static member *PossibleMatchers*, which contains references to *Matcher* objects. Several matchers have been implemented under the package *dissimilarity.matcher* including *MatcherHungarian*, *MatcherGreedy* and *MatcherMN*.

The different aggregate functions available for each pattern type are stored in the static member *PossibleAggregators*, which contains references to *Aggregator* objects. Several aggregation functions have been implemented under the package *dissimilarity.aggregator* including *AggregatorSimpleAvg* that takes the average of the dissimilarities of the matched component patterns, *AggregatorMin* that takes the minimum of the dissimilarities of the matched component patterns and *AggregatorMax* that takes the maximum of the dissimilarities of the matched component patterns.

The relationships among complex patterns, matchers and aggregation functions are depicted in Figure 3.10, where it is illustrated that each pattern type uses its own *Matcher* and *Aggregator* objects, chosen from the *PossibleMatchers* and *PossibleAggregators* lists.

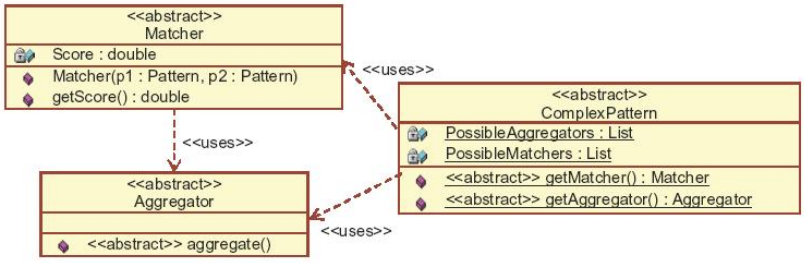


Figure 3.10: Complex patterns, matchers and aggregation functions

3.4.2 Implementing the application

Our sample application allows the comparison of different pattern types, namely frequent itemsets, clusters, time series, collections of documents etc. In the following, we detail in the implementation of PANDA for the comparison of monthly time series data obtained from the Italian MIB stock market [103]. We choose time series for illustrating purposes because the visualization of the results is easier comparing to, say, frequent itemsets.

In order for the PANDA framework to work for a specific pattern comparison problem, one has to represent the patterns to be

compared according to the simple-complex rationale. Then, for the simple patterns case, one has to define/choose the *getDissimilarityStructure()*, *getDissimilarityMeasure()* and *getDissimilarity()* methods, whereas for the complex patterns case, one has to define/choose the *Matcher* and *Aggregator* functions. In the following we describe all these steps for our sample application.

3.4.2.1 Defining the pattern types

A *StockValue* pattern type is a *simple* pattern type defined as follows:

$$\texttt{StockValue} = (SS : \texttt{(month:Integer, year:Integer)}, MS : (\texttt{[Real]}_1^N))$$

A *Stock* pattern type is a *complex* pattern type composed of *StockValue* objects:

$$\texttt{Stock} = (SS : \{\texttt{TimeSeries}\}, MS : \bot)$$

An example of two *Stock* patterns and their component *StockValue* patterns is depicted in Figure 3.11):

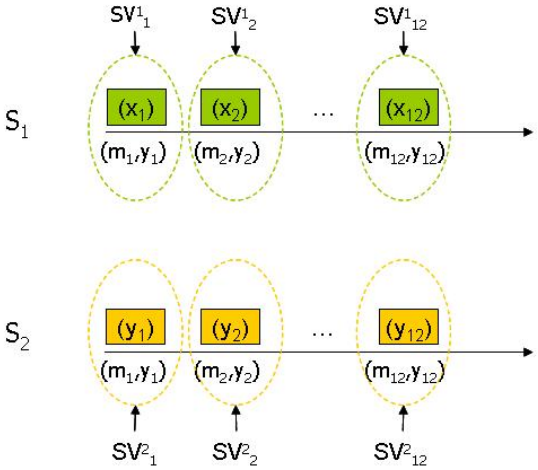


Figure 3.11: Two *Stock* patterns (denoted by S in the figure) and their component *StockValue* patterns (denoted by SV in the figure)

A *SetOfStocks* pattern type is a *complex* pattern type composed

of *Stock* objects:

$$\texttt{SetOfStocks} = (SS : \{\texttt{Stock}\}, MS : \perp)$$

Defining dissimilarity between (simple) *StockValue* patterns: The structure dissimilarity, dis_{struct}, equals to 1, if the structures are different, otherwise it equals to 0. The measure dissimilarity, dis_{meas}, equals to the absolute difference of their (normalized) measures. Structure and measure dissimilarity scores are combined ($Comb$ function) so that if $dis_{struct} = 1$, the overall distance dis to be also 1, otherwise $dis = dis_{meas}$.

Defining dissimilarity between (complex) *Stock* patterns: The overall dissimilarity between two *Stock* patterns is obtained by averaging dissimilarities between their component *StockValue* patterns corresponding to the same month. Thus, the *Matcher* block is instantiated to 1–1 matching, whereas the *Aggregation* block is instantiated to the average function.

Defining dissimilarity between (complex) *SetOfStocks* patterns: The overall dissimilarity between two *SetOfStocks* is obtained by averaging dissimilarities between the component *Stock* patterns. Again, 1–1 matching is chosen as the *Matcher* between the component *Stock* patterns and the average function is chosen to be the *Aggregator*.

Defining the matching type: As already stated in this sample application we employed 1–1 matching between the component patterns. We experimented with two different 1–1 matching algorithms: the optimal *HungarianMatcher* algorithm and the *GreedyMatcher* algorithm. The former finds the optimal 1–1 matching between the component patterns based on the Hungarian algorithm [56], whereas the second, following a Greedy approach, pairs together at each step the least dissimilar unmatched component patterns. The Greedy approach is faster than the Hungarian, however it might stuck in some local optimal, i.e., it does not guarantee the optimal solution.

Defining the aggregation logic: In this application we used average as the aggregation function.

The application interface: The main form of the application is illustrated in Figure 3.12. On the left side of the window, the user can choose, among the available pattern types, the (complex) pattern type that she/he wants to consider. After the selection of the pattern type, the list of the available matchers and aggregation functions for this type is displayed in the form and the user can choose the matcher and the aggregator for her/his application.

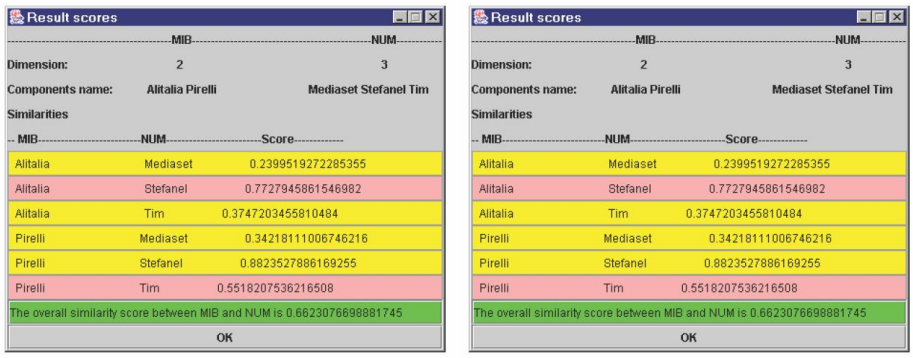
Figure 3.12: Main form of the application

In Figure 3.14 we show the result of the comparison of two sample *SetOfStocks* patterns, namely the "MIB" and the "NUM" patterns. The first set consists of 2 stocks ("Alitalia" and "Pirelli"), whereas the second set consists of 3 stocks ("Mediaset", "Stefanel", and "TIM"). The window shows the scores (here representing similarity values computed as $1 - dis$) for each pairing of stocks; the best-match couplings are highlighted in pink. Finally, the overall best-match score is shown in green in the bottom of the window. Results obtained through the *HungarianMatcher* are depicted in the left part of this figure, whereas those obtained through the *GreedyMatcher* are depicted in the right part of the figure. As it obvious from this figure, the dissimilarity score achieved by the Hungarian matcher is lower than the one achieved by the Greedy matcher, a consequence of the fact that Greedy might stuck in some local optimal, whereas Hungarian guaranties the optimal solution.

The matched time series can be also displayed in a graphical format (Figure 3.15), where the matched stocks are shown on the same row along with their similarity score, whereas non-matched stocks are displayed alone.

3.5 Running PANDA for different pattern types

We have applied PANDA to four application examples with the aim of demonstrating its usefulness to a wide range of applications.



(a) using the Hungarian matcher (b) using the Greedy matcher

Figure 3.13: Comparing two *SetOfStocks* patterns under 1–1 matching

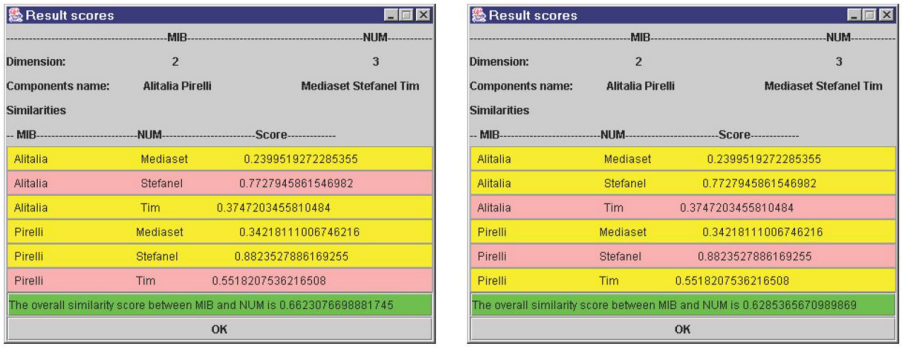


Figure 3.14: Comparing two *SetOfStocks* patterns using the optimal *HungarianMatcher* (left) and the sub-optimal GreedyMatcher(right)

3.5.1 Application to sets of itemsets

In this experiment, we used the synthetic dataset generator from the IBM Quest Data Mining group [6], which is assumed to mimic the transactions in a retailing environment to generate a dataset D of 1000 transactions with an average transaction length of 10 attributes. For the extraction of frequent itemsets, we used the MAFIA program [18].

In terms of the PANDA representation model, an *Itemset* can be modeled as a simple pattern $(s = S, m = supp)$ where S is the set of items that comprise the itemset (structure component) and $supp \in [0, 1]$ is the support of the itemset (measure component); recall also Example 9. A *SetOfItemsets* can be modeled as a complex

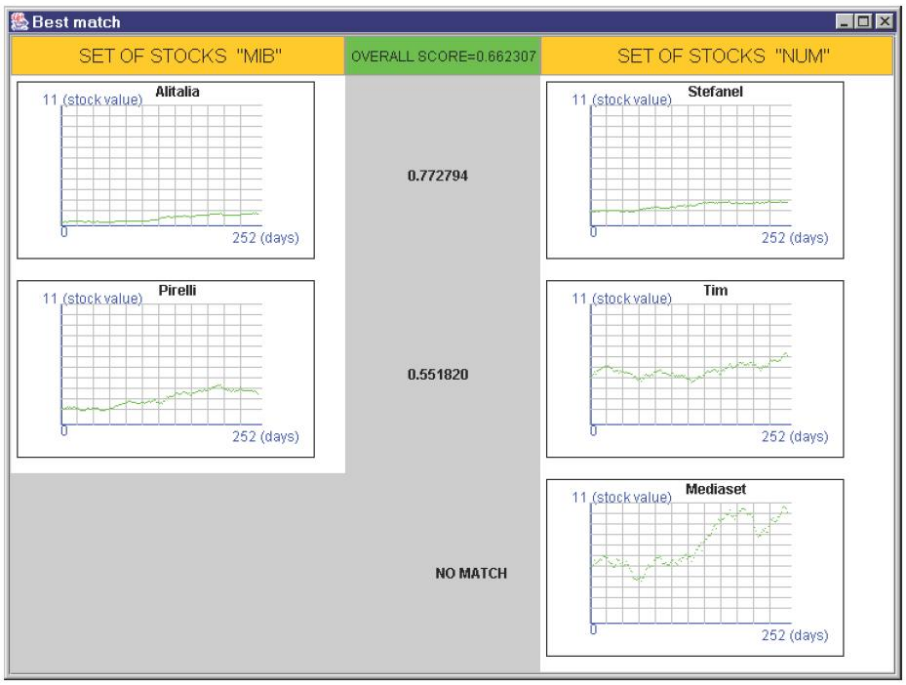


Figure 3.15: Visualization of matching between the two *SetOfStocks* patterns of Figure 3.14

pattern, composed of simple *Itemset* patterns:

$$\begin{aligned}\texttt{SetOfItemsets} = \\ &(SS : \{\texttt{Itemset}\},\\ &MS : \bot)\end{aligned}$$

In order to compare itemsets (simple patterns) we should define how their structures and measures will be compared (ie the structure and measure dissimilarity function, respectively) and how the resulting scores will be combined (i.e., the combiner function). For the measure dissimilarity function, we employed absolute difference:

$$dis_{meas}(p^i, p^j) = |p^i.m - p^j.m|$$

For the structure dissimilarity function, we used the overlap of their items:

$$dis_{struct}(p^i, p^j) = 1 - \frac{|p^i.s \cap p^j.s|}{|p^i.s \cup p^j.s|}$$

For the combiner function we employed the following function:

$$Comb(p^i, p^j) = dis_{struct}(p^i, p^j) + \big(1 - dis_{struct}(p^i, p^j)\big) * dis_{meas}(p^i, p^j)^2$$

The rationale behind this combiner is that the more similar the two itemsets are with respect to their structures, i.e., the lower the dis_{struct} score, the more the measure components will contribute to the final score, through the dis_{meas} score. This behavior is illustrated in Figure 3.16.

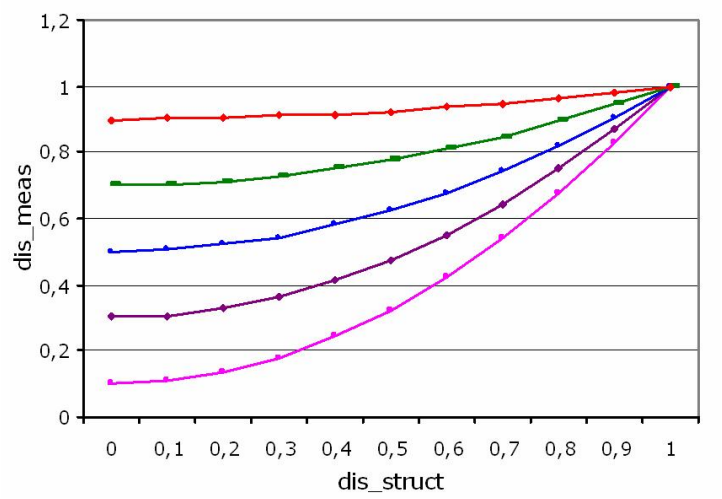


Figure 3.16: The behavior of the combiner with respect to dis_{struct} and dis_{meas}

In order to compare SetOfItemsets (complex patterns) we should define how their component patterns will be matched (i.e., matching type) and how the scores of the matched component patterns will be aggregated (i.e., aggregation logic). For the matching type, we have chosen 1–1 matching, whereas for the aggregation logic we employed the average function. In particular, for the 1–1 matching, we experimented with both the optimal Hungarian matcher [56] and the sub-optimal Greedy matcher (recall also Section 3.3.2)

Then, we performed a controlled experiment, by introducing noise to the initial dataset D_0. In detail, at each step we modified 10%, 20%, ..., 100% of the original transactions and, for each modified transaction, we changed a fixed percentage (in particular, 50%) of its items. From the result datasets (D_0, D_1, ..., D_{10}), the corresponding set of frequent itemsets ($P_0, P_1, \ldots, P_{10}$) have been extracted under a *minSupport* threshold of 20%. Then, we compared each "noisy" pattern set P_i ($i = 1, 2, \ldots, 10$), with the initial pattern set P_0. The goal was to investigate how the noise added to the data is propagated to the dissimilarity of the corresponding patterns.

As illustrated in Figure 3.17, the dissimilarity at the pattern level reflects the increased dissimilarity between the datasets when the

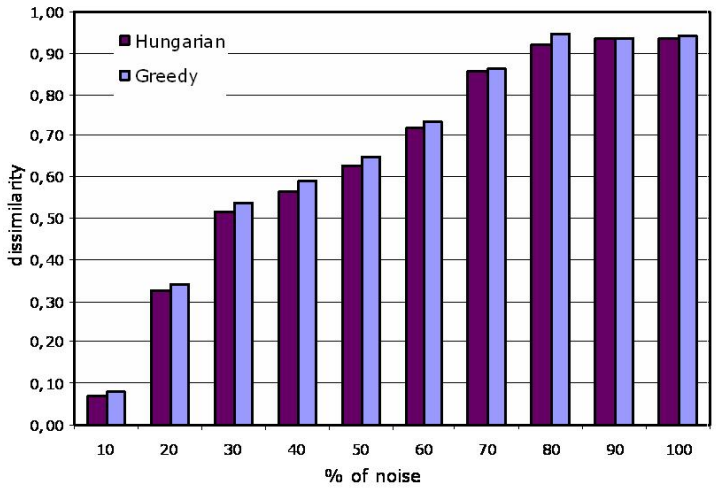


Figure 3.17: Impact of (dataset) noise on the dissimilarity of sets of itemsets for the Hungarian and the Greedy matchers

amount of noise grows. Starting from a dissimilarity of zero (when comparing P_0 with itself), the dissimilarity gradually increases as the percentage of noise added to the corresponding data set increases, reaching almost one when the initial dataset is full of noise.

When comparing the Greedy and Hungarian matchers, it is clear that the dissimilarity scores computed by the Hungarian matcher are slightly lower than those computed by the Greedy matcher, since the former provides the optimal solution to the 1–1 matching problem, while the latter only calculates an approximated solution. However, taking into account that the Greedy matcher is much faster than the Hungarian matcher (25 times, in our experiments), a practical conclusion of this case study is that the Greedy algorithm can be safely chosen as a matcher for the comparison of sets of frequent itemsets.

3.5.2 Application to decision trees

As already stated a decision tree partitions the attribute space into a set of non-overlapping regions, through its leaf nodes. A *Region* comprises a simple pattern. The structure component of a region, is composed of the conditions across the corresponding tree path. These conditions are usually numeric and thus they can be described in the form: $(ValueFrom \leq attribute \leq ValueTo)$, where *attribute* is some predictive attribute of the classification problem

and $ValueFrom, ValueTo$ are its value ranges in the specific region. Consequently, the *structure component of a region* can be described as a conjunction of such conditions. The *measure component of a region* contains the percentage of problem instances that results in this region for each of the problem classes, $\{(c, n_c)\}$, where $c \in C$ is a problem class and n_c is the percentage of problem instances that result in the specific region and belong to class c.

A DT can be modeled as a complex pattern:

$$\begin{aligned}&\texttt{DT} = \\ &(SS : \{\texttt{Region}\}, \\ &MS : \perp)\end{aligned}$$

The dissimilarity between two *regions* (simple patterns) is defined as the overlap of their corresponding hyper-rectangles. This is actually the structure dissimilarity between the regions. We do not consider the measure dissimilarity separately, rather we make the assumption that the problem instances are uniformly distributed within the attribute space and thus the volume of each region is representative of the importance of this region in the whole attribute space.

When comparing two *decision trees* (complex patterns), we employ a $N--M$ matching between the corresponding DT regions and then we aggregate the scores of the matched regions using the average function.

For our experiments, we selected the wine data set [14], which consists of the results of a chemical analysis of 178 wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each wine of the three cultivars. We carried out the following experiment: First, we induced the decision tree DT_0 from the original wine dataset. Then, we added noise to the paths/rules composing DT_0 by decreasing (increasing) of $10\%, 20\%, \ldots, 50\%$ the `ValueFrom` (respectively `ValueTo`) of each attribute of each rule. As a result, we induce a corresponding set of "noised" decision trees $(DT_{10}, DT_{20}, \ldots, DT_{50})$. We also randomly decreased or increased the support of each rule by the corresponding noise percentage. We studied the impact of the noise by comparing the "noised" trees $DT_i, i = \{10, 20, \ldots, 50\}$ with the initial "clean" tree DT_0. Intuitively, the more noise we add, the more dissimilar from the initial tree the new tree would be; Figure 3.18 asserts this perception.

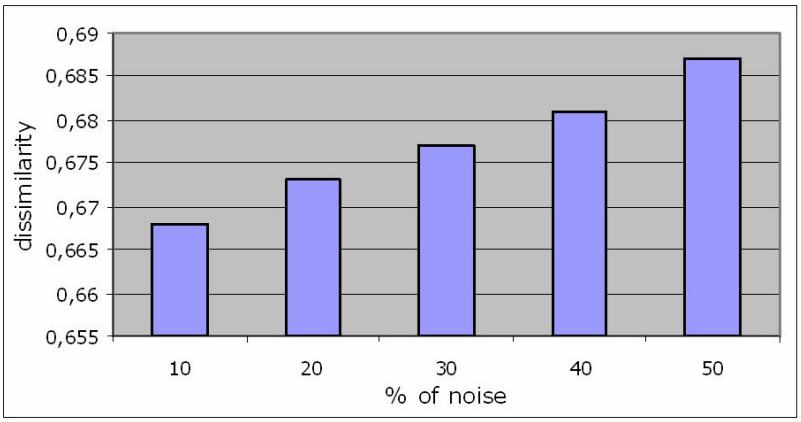


Figure 3.18: Impact of (pattern space) noise on DT dissimilarity

3.5.3 Application to collections of documents

Our final experiment refers to the comparison of collections of documents. In particular, we considered the DBLP database [58], which records, among others, the publications of a large collection of journals: each journal contains a set of articles referring to the same general topic (the one covered by the journal) and each article is described through a set of keywords. Keywords are compared as in Example 10, while the 1–1 matching type and the average function aggregator were employed for the comparison of the complex patterns.

The DBLP journals used for the experiments are listed in Table 3.2, whereas the results of the comparison are depicted in Figure 3.19.

DBLP Journal	Abbreviation
Computer Journal	Comp J
Artificial Intelligence	AI
Computer Networks	Comp N
Computers & Graphics	Comp G
Information Systems	Info Sys
Computer Networks and ISDN Systems	Comp Net & ISDN Sys
Computational Intelligence	Comp Intell
Computer Languages	Comp Lang
Distributed Computing	Dist Comp
Advances in Computers	Adv in Comp
Evolutionary Computation	Evol Comp
Computational Complexity	Comp Compl
Information Retr ieval	IR

Table 3.2: DBLP journals

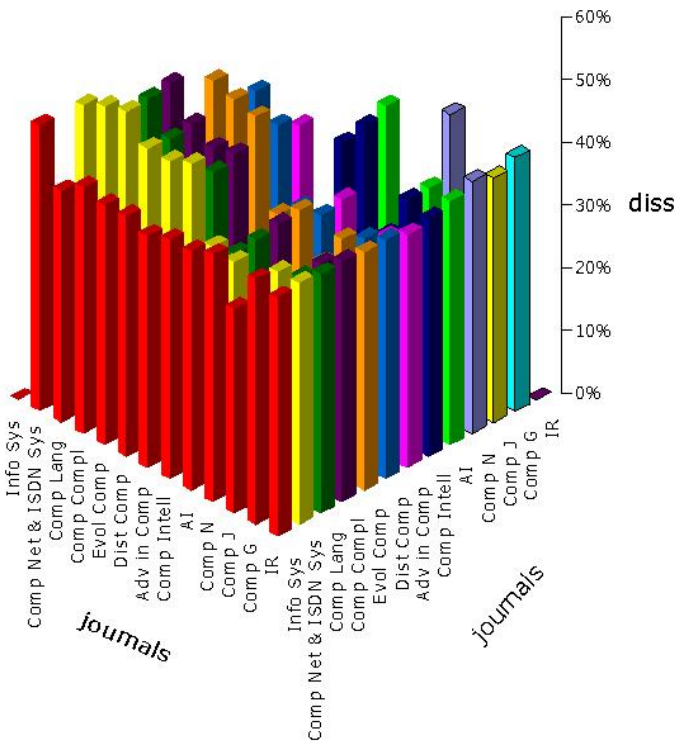


Figure 3.19: Comparing journals from the DBLP database

There are some interesting conclusions that can be drawn from this experiment:

- The *Computer Journal* appears to be quite similar to the other journals (with a dissimilarity score between $\backsim 0.2$ and $\backsim 0.3$), a result that is reasonable since this journal provides a wide overview of developments in the field of Computer Science.

- On the other hand, journals like *Evolutionary Computation*, *Computational Intelligence*, *Computer Networks*, *ISDN Systems*, and *Distributed Computing* are very dissimilar to each other, probably due to the fact that they cover specific and rather non-overlapping topics of Computer Science.

- The greatest distance calculated ($\backsim 0.5$) was the one between the *Evolutionary Computation* and the *Distributed Computing* journals, which seem to cover quite distinct topics.

3.6 Related work

In this section we overview approaches and techniques sharing some common aspects with our framework, pointing out their limitations with respect to PANDA.

FOCUS framework To the best of our knowledge the only generic framework for the comparison of patterns is FOCUS [32], which provides a principled approach for measuring the deviation between two datasets, in terms of the corresponding extracted pattern sets. The rationale behind using pattern comparison for data comparison is that patterns capture interesting characteristics of the data and thus they can be employed for dataset comparison purposes. Three popular pattern types, namely frequent itemsets, decision trees and non-overlapping clusterings, are supported by FOCUS.

The central idea of FOCUS, which is also shared by PANDA, is the modeling of patterns through a structure and a measure component (called the *2-components property* of patterns). According to FOCUS, a pattern p consists of a set of "regions" defined over the attribute space (the structure of p) and each region is associated to a set of measures (the measure of p). For instance, let as consider a decision tree DT built upon a dataset D that is described through the predictive attributes $A = \{A_1, \ldots, A_n\}$ and the class attribute $C = \{C_1, \ldots, C_m\}$. DT can be represented as a set of regions that partition the $n-$dimensional space; each leaf node of the DT corresponds to one such region. Each region is associated to m measures, one for each problem class C_j. The region measure for class C_j expresses the amount of the dataset instances that result in the specific region and belong to C_j.

Both structural and measure components are taken into account during similarity assessment. If two patterns share the same structure, then their dissimilarity is computed by aggregating (using a function g) over all regions the deviations between regions' measures (computed using a difference function f). The difference function f could be the absolute or relative difference, whereas the aggregation function g could be the sum or max function. If the structural components are not identical, then a preliminary step is required so as to reconcile them. This includes splitting regions until the two structures become identical; the result is called Greatest Common Refinement (GCR). For instance, the GCR of two decision trees is the decision tree whose regions are obtained by intersecting the sets of regions of the two corresponding decision trees. In general, this requires to compute the measures of the (new) regions in the GCR

from the underlying datasets. After that, what described for the basic case of equal structures can be also applied.

The comparison of FOCUS with PANDA reveals a number of limitations that prevent using the former as a general and flexible framework for pattern comparison, in the light of the requirements listed in Section 3.1:

- FOCUS only considers patterns whose structure consists of a two-level hierarchy, i.e., each pattern is composed of regions, while PANDA supports the recursive definition of arbitrarily complex patterns. A number of interesting pattern types are therefore left out by FOCUS, like the web sites comparison discussed in Example 5. On the other hand, all pattern types that are supported by FOCUS can also be managed by PANDA.
- Because of the use of GCR, FOCUS is not flexible concerning the matching of component patterns. This prevents comparing patterns based on some "holistic" criterion, since FOCUS only considers a simple aggregation of differences of measures on the corresponding regions. This kind of 1-1 matching is not general enough. Both examples 3 and 4 in Section 3.1 describe scenarios that cannot be captured by FOCUS; in Example 3, for instance, neighbor regions need to be considered, which precludes the usage of 1 - 1 matching between regions. On the other hand, PANDA is very flexible, since a user can freely define all the building blocks for the comparison of complex patterns, namely: (a) how to assess the dissimilarity between simple patterns, (b) how to match the component patterns, and (c) how to aggregate the scores of the matched component patterns so as to obtain the overall dissimilarity score.
- The usage of the GCR also prevents the applicability of FOCUS to a number of interesting patterns, where the notion of a common refinement cannot be defined, or where a more flexible definition of matching between the component regions is required.
- Since FOCUS is concerned with the comparison of datasets, it requires accessing the underlying data, whereas pattern comparison in PANDA is totally processed in the pattern space. Thus, in PANDA an improvement is obtained in efficiency (because costly accesses to the raw data are avoided), generality (because it can deal with cases where the raw data are not available), and privacy (there is now no need to access the raw data that may contain private information).

From all the above it follows that PANDA clearly overcomes FOCUS by dealing with arbitrarily complex patterns and allowing more flexibility during the dissimilarity assessment process.

Inductive databases Within the inductive databases framework [43] both data and patterns are stored together in order to be uniformly retrieved and manipulated (c.f. also Section 1.3.1). The patterns considered are mainly in the form of rules induced from these data. In fact, rule discovery is viewed as yet another, perhaps more expressive, type of querying and this is why a number of specialized inductive query languages have been proposed and implemented.

Most of these languages extend SQL with some primitives to support data mining. For example, DMQL [38] is a query language for several types of rules like association rules, generalized relations and characteristic rules. MINE RULE [63] proposes a model that enables a uniform description of the association rules discovery problem. MSQL [44] is a language for the generation and querying of association rules. In all these approaches, inductive queries specify patterns of interest using either syntactic or frequency constraints (or both of them).

It is obvious that, within the inductive databases framework, pattern comparison is limited to specific types of patterns, like association rules and string patterns. Furthermore, specific types of pattern matching are considered based on syntactic and frequency criteria. The emphasis here is on the retrieval of patterns rather than on their comparison. Also, the case of complex patterns is not considered at all.

Pattern monitoring A relevant line of research is that of pattern monitoring which aims at monitoring patterns across the time axis and at detecting their changes.

In this category the DEMON framework [33] belongs for mining systematically evolving data across the temporal dimension; "systematically" refers to data changes that are implied due to additions or deletions of blocks of records, i.e., sets of records simultaneously added to or deleted from the database. At first, the data blocks that have been changed are detected across the time dimension and then, they are processed by the miner in order to update the pattern base. Efficient model maintenance algorithms are described for the case of frequent itemsets and clusters models. However, the DEMON framework focuses on the efficient update of models (i.e., patterns), rather than on their comparison.

In the same context, the PAM framework [8] aims at the efficient maintenance of DM results, particularly of association rule patterns. Patterns are modeled as temporal, evolving objects that might exhibit changes either in their content, or in their statistical properties. A change detector mechanism is introduced for identifying changes in the statistics of the rules; statistical significance is used to assess the strength of these changes. The core of PAM implements the change detector and a series of heuristics so as to identify not only significant but also interesting rule changes which take different aspects of pattern reliability into account.

Indeed, pattern monitoring builds upon pattern comparison in order to detect significant changes across the temporal dimension. This indicates another applicability of the pattern dissimilarity assessment problem and of our PANDA framework, that of monitoring pattern changes in dynamic environments.

Kernel methods A relevant line of research is that of kernel methods, which become increasingly popular due to the fact that they can be utilized to perform Machine Learning tasks on complex high-dimensional data without the burden of running algorithms directly on these data [34]. All computations are done via kernel functions; for such functions it is known that there exists an embedding of the set in a linear space such that the kernel on the elements of the set corresponds to the inner product in this space. Some traditionally used kernels are: linear, normalized linear, polynomial, Gaussian and matching kernel.

More relevant to our work is the research on kernel methods for structured data. The best known kernel for the representation of spaces that are not mere attribute–value tuples is the convolution kernel proposed in [40]. The basic idea of convolution kernels is that the semantics of composite objects can often be captured by a relation between the object and its parts. The kernel on the object is then made up from kernels defined on different parts. Other examples of kernels on structured data include kernels on strings, trees and graphs.

The framework proposed in [35] allows for the application of kernel methods to different kinds of structured data. This approach is based on the idea of having a powerful knowledge representation that allows for modeling the underlying semantics of the data; this representation formalism is achieved by using a typed syntax and by representing individuals as terms. In [35], kernels are defined for basic terms, assuming that atomic kernels exist for all data con-

structors that are used. In addition, kernel modifiers are proposed so as to customize the kernel definition to the domain at hand. Semantically, this work is close to our work since both try to provide general frameworks by relying on the component parts of the data (structured data in this case, patterns in our case). The main difference is that PANDA framework focuses on comparing the results of the Data Mining process, whereas kernel methods constitute tools for Data Mining tasks, like classification and regression. However, the work on kernels might enhance PANDA especially in the issue of dissimilarity reasoning, i.e., associating dissimilarity in data space with the dissimilarity in pattern space.

Case-based reasoning The notion of similarity is a key concept in Case-Based Reasoning (CBR) where, given a problem to be solved, similar problems that have been solved in the past are retrieved so as to reuse/adapt their solution to the problem at hand [11]. Cases are usually represented as complex structures that are recursively built on top of simpler structures (using part-of relations). The similarity between aggregated structures is computed by combining similarities among base attributes [13, 25]. This is quite similar to our approach, since cases can be seen as complex patterns that are recursively built upon simple patterns (properties).

Similarity between simple attributes is usually computed by considering the difference among the values of the attributes; this can be considered as the distance of the measure components in our PANDA framework. The fact that differences in attributes can arise when attributes are chosen from a class hierarchy (due to different semantics of nodes in the hierarchy) [13] can be accounted as a kind of structure dissimilarity between the (base) patterns. Clearly, the problem of matching is not relevant here, since cases share the same structure, i.e., they have the same properties, thus only a (trivial) 1-1 matching is considered.

Ad-hoc solutions for particular cases Several ad-hoc solutions for comparing particular pattern types exists in the literature. We do not present these approaches in this section, since they are described in detail in the corresponding chapters of this thesis, namely in Chapter 4 for frequent itemsets, in Chapter 5 for decision trees and in Chapter 6 for clusters.

The interested reader can find in [71] an extensive survey on comparing data mining patterns, namely frequent itemsets and association rules, clusters and clustering and decision trees. Also, in [62] a

survey on comparing clusterings extracted from the same dataset either by different algorithms or by the same algorithm under different parameters is presented.

3.7 Summary

In this chapter we presented PANDA a both generic and flexible framework for assessing dissimilarity between arbitrary complex patterns. PANDA is *generic* since patterns of arbitrary complexity are supported and *flexible* since dissimilarity assessment can be easily adapted to specific user/application requirements. In PANDA patterns are modeled as entities composed of two parts: the *structure component* that identifies "interesting" regions in the attribute space, e.g., the head and the body of an association rule, and the *measure component* that describes how the patterns are related to the underlying raw data, e.g., the support and the confidence of the rule. When comparing two simple patterns, the dissimilarity of their structure components (*structure dissimilarity*) and the dissimilarity of their measure components (*measure dissimilarity*) are combined (through some *combining function*) in order to derive the total dissimilarity score. The problem of comparing complex patterns is reduced to the problem of comparing the corresponding sets (or lists, arrays etc.) of component (simple) patterns. Thus, component patterns are first paired (using a specific *matching type*) and their scores are then aggregated (through some *aggregation function*) so as to obtain the overall dissimilarity score. This recursive definition of the dissimilarity allows PANDA to handle patterns of arbitrary complexity.

The goal of the PANDA framework is to provide the end user with a powerful framework for dissimilarity assessment able to deal with specific user/application requirements. We do not consider here the problem of finding the "best" measure for every dissimilarity assessment problem, which of course would be utopian. What we provide are mechanisms for pattern comparison based on the concepts of simple/complex patterns. The end user can tune the building blocks of PANDA according to her/his actual needs and experiment with the results in order to decide which configuration is the most appropriate measure for her/his specific application.

An early version of the results presented in this chapter has been published in [9], whereas an extended version has been submitted in [10].

3.8 Open issues

PANDA is a fully modular framework and several improvements/ extensions can be performed over it. A straightforward extension is to enhance the different building blocks of PANDA (e.g., dis_{struct}, $Comb$, $Aggr$) with new evaluation functions so as to give more alternatives to the end user. A similar direction is to incorporate in PANDA new pattern types, like sequences, and thus to expand the set of the supported pattern types.

As already stated, by instantiating the different building blocks of PANDA different dissimilarity assessment configurations arise. The critical question is which of these configurations is suitable for some particular problem. Already proposed schemas could be incorporated into the PANDA framework, providing ready solutions to the end user.

So far, PANDA deals with the dissimilarity assessment between patterns of the same pattern type. It is interesting however to investigate how dissimilarity between different pattern types could be evaluated. A straightforward solution would be to convert one pattern type into the other (e.g., a decision tree into as a set of rules) and then apply the PANDA framework. This solution, however, introduces several limitations since such a mapping is not always possible. An alternative solution would be to define a similarity operator on a pattern super-type, whose sub-types are the two pattern types to be compared. More effective solutions could be also explored.

Chapter 4

Comparing Datasets using Frequent Itemsets: the Effect of Mining Parameters

In this chapter we investigate the problem of utilizing dissimilarity between pattern sets as a measure of dissimilarity between the original raw datasets. More specifically, we focus on frequent itemset patterns and investigate how the mining parameters used for their extraction affect the resulting dissimilarity score. We examine two mining parameters, namely the *minSupport* threshold and the adopted *lattice representation* among three alternatives frequent itemsets, closed frequent itemsets or maximal frequent itemsets).

The chapter is organized as follows: In Section 4.1, we make an introduction to the problem. In Section 4.2, we describe the basic concepts of the Frequent Itemset Mining (FIM) problem that are necessary for the understanding of this work. In Section 4.3, we discuss the related work and describe a general formula under which the different dissimilarity measures proposed in literature can be described. In Section 4.4, we present the FIM parameters that affect dissimilarity, namely the *minSupport* threshold and the compactness level of the lattice representation (frequent itemsets, closed frequent itemsets or maximal frequent itemsets). In Section 4.5, we verify the theoretical results by experimentally evaluating the effect of the different parameters on dissimilarity. In Section 4.6, we summarize the findings of this study, whereas in Section 4.7, we outline the emerged research issues.

Index terms frequent itemsets dissimilarity, *minSupport* effect, lattice compactness level effect.

4.1 Introduction

Detecting changes between datasets is an important problem nowadays due to the highly dynamic nature of data and the collection of data from different data sources. A common approach for comparing datasets is to utilize the pattern sets extracted from these datasets. The intuition behind this approach is that, to some degree, patterns condense the information contained in the original raw data. For example, in [32], the authors measure the deviation between two datasets in terms of the data mining models they induce, namely frequent itemsets, decision trees or clusters. This comparison is used as the basis for mining systematically evolving data [33], like the buying habits of customers in a supermarket. In [81], the authors measure the dissimilarity between distributed datasets (e.g., the different branches of a supermarket) using the corresponding sets of frequent itemsets. The same rationale is followed by the authors in [59].

Intuitively, it is reasonable to employ pattern comparison for data comparison. Indeed, patterns preserve part of the information contained in the original raw data (recall that by definition patterns are compact and rich in semantics representations of raw data), however the degree of preservation depends on the mining parameters used for their extraction. Thus, when comparing two datasets on the basis of their corresponding pattern sets, the resulting dissimilarity score is subjective, with respect to the mining parameters used for the extraction of these patterns. This is a kind of a dissimilarity reasoning problem (recall Challenge 3 in Section 1.4).

In this chapter, we investigate the dissimilarity reasoning problem for a very popular pattern type, namely the frequent itemset patterns and their variations, closed frequent itemsets and maximal frequent itemsets. Frequent Itemset Mining (FIM) is a well studied research area in DM due to its broad applications (e.g., association rules, correlations, sequential patterns and episodes [102]). Plenty of work has been carried out in this domain, mainly focusing on developing efficient algorithms for the FIM problem (see [41], [37] for an overview of the area). Furthermore, algorithms for discovering more compact representations of the frequent itemset lattice (FI), like maximal frequent itemsets (MFI) and closed frequent itemsets (CFI), have been also proposed in the literature, e.g., [36], [105].

To summarize, in this chapter, we investigate whether the dissimilarity score between two sets of frequent itemsets is affected by the mining parameters used for their extraction. The first parameter that we examine is the *minSupport threshold*, which constraints the

itemset lattice by eliminating those itemsets whose support is less than the given threshold value. The second parameter is the *lattice compactness level* (FI, CFI or MFI), which constraints the frequent itemsets lattice by eliminating redundant itemsets based on either the structure components (like in MFIs), or in both structure and measure components (like in CFIs). Thus, both parameters constraint the frequent itemsets lattice based on either the structure or the measure component of the itemsets or in both of them.

Our analysis shows that utilizing frequent itemsets comparison for dataset comparison is not as straightforward as related work has argued, a result which is verified through an experimental study and opens issues for further research in the KDD field. The contributions of this chapter are as follows:

- We describe the different dissimilarity measures proposed so far in the literature ([32, 59, 81]) through a general common dissimilarity schema according to the PANDA framework presented in Chapter 3.

- We provide a theoretical analysis that shows the dependency of dissimilarity in pattern space on the FIM settings. In particular, we prove two lemmas: Regarding the *minSupport* threshold, our analysis shows that the larger this threshold is, the higher the dissimilarity in pattern space is. Regarding the different lattice representations (FI, CFI or MFI), it turns out that the more compact the representation achieved by the itemset type is, the higher the dissimilarity in pattern space is.

- We verify the above theoretical results through an experimental study. The results indicate that utilizing pattern comparison for data comparison is not as straightforward as argued by related work and should only be carried out under certain assumptions (e.g., FIM settings).

4.2 Background on the FIM problem

In Section 2.5, we have already presented some basic concepts on the Frequent Itemsets Mining (FIM) problem. In this section, we provider further information necessary for understanding of this chapter.

As already stated, an itemset is frequent if its support exceeds a user defined *minSupport* threshold σ, i.e., $supp_D(X) \geq \sigma$. The set of frequent itemsets extracted from D under the *minSupport*

threshold σ is defined as:

$$F_\sigma(D) = \{X \subseteq I \mid supp_D(X) \geq \sigma\} \tag{4.1}$$

The set of frequent itemsets forms the itemset lattice L in which the Apriori property holds: an itemset is frequent iff all of its subsets are frequent. The Apriori property allows us to enumerate all frequent itemsets using more compact representations like closed frequent itemsets (CFI) and maximal frequent itemsets (MFI).

A frequent itemset X is called *closed* if there exists no frequent superset $Y \supset X$ with $supp_D(X) = supp_D(Y)$. The set of closed frequent itemsets extracted from D under σ is defined as:

$$C_\sigma(D) = \{X \in F_\sigma(D) : Y \supset X \Rightarrow supp_D(X) > supp_D(Y), Y \in F_\sigma(D)\} \tag{4.2}$$

$C_\sigma(D)$ is a subset of $F_\sigma(D)$ since every closed itemset is also frequent.

$$C_\sigma(D) = F_\sigma(D) - \{X \in F_\sigma(D) : Y \supset X \Rightarrow supp_D(X) = supp_D(Y), Y \in F_\sigma(D)\} \tag{4.3}$$

By definition, $C_\sigma(D)$ is a lossless representation of $F_\sigma(D)$ since both the lattice structure (i.e., frequent itemsets) and the lattice measure (i.e., itemset supports) can be derived from CFIs [105].

On the other hand, a frequent itemset is called *maximal* if it is not a subset of any other frequent itemset. The set of maximal frequent itemsets extracted from D under the *minSupport* threshold σ is defined as:

$$M_\sigma(D) = \{X \in F_\sigma(D) : Y \subset X \Rightarrow Y \notin F_\sigma(D), Y \subseteq I\} \tag{4.4}$$

$M_\sigma(D)$ is also a subset of $F_\sigma(D)$ since every maximal itemset is frequent.

$$M_\sigma(D) = F_\sigma(D) - \{X \in F_\sigma(D) : Y \supset X \Rightarrow Y \in F_\sigma(D)\} \tag{4.5}$$

Maximal frequent itemsets are also closed since by definition they cannot be extended by another frequent item to yield a frequent itemset [105]:

$$M_\sigma(D) = C_\sigma(D) - \{X \in C_\sigma(D) : Y \supset X \Rightarrow Y \in C_\sigma(D)\} \tag{4.6}$$

Unlike $C_\sigma(D)$, $M_\sigma(D)$ is a lossy representation of $F_\sigma(D)$ since it is only the lattice structure (i.e., frequent itemsets) that can be determined from MFIs whereas frequent itemset supports are lost [105].

By Equation 4.3, 4.5 and 4.6 it follows that:

$$M_\sigma(D) \subseteq C_\sigma(D) \subseteq F_\sigma(D) \tag{4.7}$$

In practice, CFIs can be orders of magnitude less than FIs, and MFIs can be orders of magnitude less than CFIs [105]; see also Figure 4.3.

For illustrating purposes, let us consider the two sets of frequent itemsets A, B depicted in Figure 4.1. Suppose that A and B were generated under the same *minSupport* threshold σ from the original datasets D and E, respectively. Each itemset is described as a pair <*structure, measure*> denoting the items forming the itemset (*structure component*) and the itemset support (*measure component*).

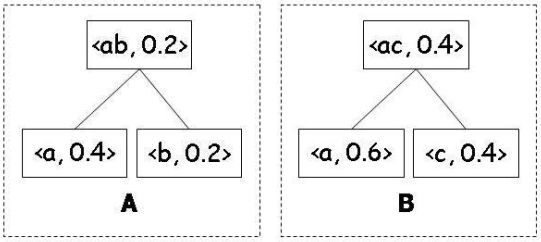


Figure 4.1: Two lattices of frequent itemsets: A (left), B (right).

The question is how similar to each other A and B are. There are many cases where the two sets might differ: An itemset, for example, might appear in both A and B sets but with different supports, like the itemset $< a >$ in Figure 4.1. Or, an itemset might appear in only one of the two sets, like the itemset $< b >$ which appears in A but not in B. In this case, two alternatives might occur: either it does not actually exist in the corresponding dataset E or $< b >$ has been pruned due to low support (lower than the *minSupport* threshold σ).

Since the generation of A, B sets depends on the FIM parameters namely, the *minSupport* threshold σ used for their generation and the adopted lattice representation (FI, CFI or MFI), we argue that the estimated dissimilarity score also depends on these parameters. Consequently, since dissimilarity in pattern space is often used as a measure of dissimilarity in raw data space, the above mentioned parameters also affect this correspondence.

Table 4.1 summarizes the symbols introduced in this section.

4.3 Comparing FI lattices

Related work ([32], [59], [81]) utilizes comparison between sets of frequent itemsets for dataset comparison, by considering both structure and measure components. In the following paragraphs we outline

Symbol	Description
D	a dataset
X	an itemset
$supp_D(X)$	the support of itemset X in D
σ	the *minSupport* threshold
$F_\sigma(D)$	the set of frequent itemsets in D under σ
$C_\sigma(D)$	the set of closed frequent itemsets in D under σ
$M_\sigma(D)$	the set of maximal frequent itemsets in D under σ
$dis(A, B)$	dissimilarity between two set of itemsets A, B

Table 4.1: List of symbols for Chapter 4

the above related work; we exclude [14], which leaves measure components out of consideration.

4.3.1 Parthasarathy-Ogihara approach

Parthasarathy and Ogihara [81] present a method for measuring the dissimilarity between two datasets D and E by using the corresponding sets of frequent itemsets (A and B, respectively). Their metric is defined as follows:

$$dis(A, B) = 1 - \frac{\sum_{X \in A \cap B} \max\{0, 1 - \theta * |supp_D(X) - supp_E(X)|\}}{|A \cup B|} \tag{4.8}$$

In the above equation, θ is a scaling parameter that is specified by the user and reflects how significant are the variations in the support values. For $\theta = 0$, the measure component carries no significance, whereas for $\theta = 1$, the measure component is of equal importance with the structure component.

This measure works with itemsets of identical structure, i.e., those appearing in $A \cap B$. Itemsets that only partially fit each other like $< ab >$ and $< ac >$ are considered totally dissimilar.

For our running example (c.f. Figure 4.1), it holds that $A \cap B = \{< a >\}$. Assuming $\theta = 1$, it follows that $dis(A, B) = 0.84$ according to Equation 4.8.

4.3.2 FOCUS approach

Ganti et al. [32] propose the FOCUS framework for quantifying the deviation between two datasets D and E in terms of the sets of frequent itemsets (A and B, respectively) they induce. The deviation is defined as the amount of work required to transform one set into the other. To this end, A and B are refined into their union, $A \cup B$, and the support of each itemset is computed with respect to both

datasets D and E. Next, the deviation is computed by summing up the deviations of the frequent itemsets in the union:

$$dis(A,B) = \frac{\sum_{X \in A \cup B} |supp_D(X) - supp_E(X)|}{\sum_{X \in A} supp_D(X) + \sum_{X \in B} supp_E(X)} \quad (4.9)$$

FOCUS measures the dissimilarity between two sets of frequent itemsets in terms of their union. Partial similarity is not considered within FOCUS; in fact FOCUS tries to find itemsets of identical structures. More specifically, if an itemset X appears in A with $supp_D(X)$ but not in B, then the corresponding data set E of B is queried so as to retrieve $supp_E(X)$.

An upper bound on dissimilarity is provided for the case of frequent itemsets [32], which involves only the induced models and avoids the expensive operation of querying the original raw data space. In this case, if an itemset X does not appear in B, it is considered to appear but with zero measure, i.e., $supp_E(X) = 0$. In the following, we consider the upper bound case, since we are interested on how patterns (i.e., frequent itemset sets) capture similarity features contained in the original raw datasets.

For our running example (c.f. Figure 4.1), it holds that $A \cup B = \{< a >, < b >, < c >, < ab >, < ac >\}$, resulting in $dis(A,B) = 0.64$ according to Equation 4.9.

4.3.3 Li-Ogihara-Zhou approach

Li et al [59] propose a dissimilarity measure between datasets based on the set of maximal frequent itemsets (MFIs) extracted from these datasets. Their metric is defined as follows: Let $A = \{X_i, supp_D(X_i)\}$ and $B = \{Y_j, supp_E(Y_j)\}$ where X_i, Y_j are the MFIs in D, E respectively. Then the dissimilarity between A and B is defined as:

$$dis(A,B) = 1 - \frac{2I_3}{I_1 + I_2} \quad (4.10)$$

where

$$I_1 = \sum_{X_i, X_j \in A} d(X_i, X_j), \quad I_2 = \sum_{Y_i, Y_j \in B} d(Y_i, Y_j), \quad I_3 = \sum_{X \in A, Y \in B} d(X,Y)$$

and

$$d(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} * log(1 + \frac{|X \cap Y|}{|X \cup Y|}) * \min(supp_D(X), supp_E(Y))$$

I_3 can be considered as a measure of "mutual information" between A and B sets, whereas the fraction $\frac{2}{I_1+I_2}$ serves as a normalization factor.

This measure works with the average dissimilarity between pairs of MFIs from A and B sets. Partial similarity is considered within this approach; that is, itemsets that have some common part in their structure are compared and their score is aggregated to the total dissimilarity score, $dis(A, B)$.

For our running example (c.f. Figure 4.1), $dis(A, B) = 0.58$ according to Equation 4.10.

4.3.4 Common background of the three approaches

All approaches express the dissimilarity between two sets of frequent itemsets as an aggregation of the dissimilarities of their component itemsets[1]:

$$dis(A, B) = \sum_{X \in A, Y \in B} dis(X, Y) \tag{4.11}$$

where $dis(X, Y)$ is the dissimilarity function between two simple frequent itemsets, defined in terms of their structure and measure components, as follows:

$$dis(X, Y) = \mathsf{f}(dis_{struct}(X, Y), dis_{meas}(X, Y)) \tag{4.12}$$

The function $dis_{struct}(X, Y)$ evaluates the dissimilarity between the structure components (i.e., frequent itemsets) of X, Y, whereas the function $dis_{meas}(X, Y)$ evaluates the dissimilarity between their measure components (i.e., supports). Finally, the function f aggregates the corresponding structure and measure dissimilarity scores by summing them up into a total dissimilarity score.

All approaches ([81], [32], [59]) follow the same rationale of Equation 4.11 and differentiate only on how $dis_{struct}(X, Y)$ and $dis_{meas}(X, Y)$ are evaluated and on how their scores are combined through the aggregation function f.

- In case of the Parthasarathy-Ogihara measure, Equation 4.12 can be written as:

  $$dis(X, Y) = \max\{0, 1 - dis_{struct}(X, Y) - \theta * dis_{meas}(X, Y)\}$$

  where

  $$dis_{struct}(X, Y) = \begin{cases} 0 & \text{, if } X = Y \\ 1 & \text{, otherwise} \end{cases}$$

  $$dis_{meas}(X, Y) = \begin{cases} |supp_D(X) - supp_E(Y)| & \text{, if } X = Y \\ 0 & \text{, otherwise} \end{cases}$$

[1] In this formula, we do not consider the normalization factor.

- In case of the FOCUS measure, Equation 4.12 can be written as:

$$dis(X,Y) = (1 - dis_{struct}(X,Y)) * dis_{meas}(X,Y)$$

where

$$dis_{struct}(X,Y) = \begin{cases} 0 & \text{, if } X = Y \\ 1 & \text{, otherwise} \end{cases}$$

$$dis_{meas}(X,Y) = \begin{cases} |supp_D(X) - supp_E(Y)| & \text{, if } X,Y \in A \cap B \text{ and } X = Y \\ supp_D(X) & \text{, if } X \in A - B \\ supp_E(Y) & \text{, if } Y \in B - A \end{cases}$$

- Finally, for the Li-Ogihara-Zhu measure, Equation 4.12 can be written as:

$$dis(X,Y) = dis_{struct}(X,Y) * \log(1 + dis_{struct}(X,Y)) * dis_{meas}(X,Y)$$

where

$$dis_{struct}(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

$$dis_{meas}(X,Y) = \min\{supp_D(X), supp_E(Y)\}$$

The fact that all approaches follow a general rationale is also confirmed by the experiments (Section 4.5), where, although the dissimilarity scores vary, a similar behavior is exhibited by all approaches.

We could express all these measures in terms of the PANDA framework terminology (c.f. Chapter 3): itemsets correspond to simple patterns whereas sets of itemsets correspond to complex patterns. The dissimilarity between two simple patterns (i.e., itemsets) for each approach is given as described in this section. For the comparison of complex patterns (i.e., sets of frequent itemsets) all approaches employ the *sum* function as the aggregation function. Regarding the matching type, Parthasarathy-Ogihara and FOCUS employ $1-1$ *matching* between the component itemsets, whereas Li-Ogihara-Zhou employs $M-N$ *matching* between the component itemsets of the two sets to be compared.

4.4 Effect of mining parameters on dissimilarity

In the following subsections, we investigate how the dissimilarity between two sets of frequent itemsets depends on the *minSupport* threshold σ used for their generation and on the adopted lattice representation (FI, CFI or MFI). We start with a single dataset from

which we extract patterns under different mining parameters and then we compare the resulting pattern sets with the initial pattern set (i.e., the one extracted from the initial dataset).

4.4.1 Effect of *minSupport* threshold on dissimilarity

Let σ, $\sigma + \delta$ $(0 < \sigma, \delta < \sigma + \delta \leq 1)$ be two *minSupport* thresholds applied over a data set D and let F_σ, $F_{\sigma+\delta}$ be the corresponding sets of frequent itemsets produced by (any) FIM algorithm. The difference set $F_\sigma - F_{\sigma+\delta}$ contains all those itemsets whose support lies between σ and $\sigma + \delta$:

$$Z \equiv F_\sigma - F_{\sigma+\delta} = \{X \subseteq I \mid \sigma \leq supp(X) < \sigma + \delta\} \qquad (4.13)$$

In Figure 4.2, an example is depicted which illustrates how the resulting lattice is affected by the increase δ in the *minSupport* threshold value. As it is shown in this figure, as δ increases the resulting lattice is reduced, which is an obvious behavior. Note however, that two values δ, δ' might result in the same lattice even if $\delta < \delta'$, that is, the pruning function is not monotonic. For example, the pruning effect on the lattice of Figure 4.2 for $\delta = 0.05$ and $\delta' = 0.1$ is the same.

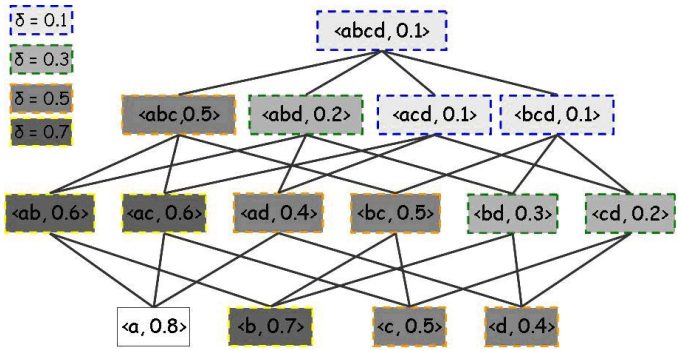


Figure 4.2: Effect of δ increase on the lattice structure ($\sigma = 0.1$)

Below, we show how each of the presented measures is affected by the increase δ in the *minSupport* value.

Lemma 1 *For either the Parthasarathy-Ogihara, or FOCUS or Li-Ogihara-Zhou approach, it holds that the larger the increase in the minSupport threshold value δ is, the larger the computed dissimilarity score, $dis(F_\sigma, F_{\sigma+\delta})$, is.*

Proof:

Parthasarathy-Ogihara approach: From Equation 4.8 and since $F_\sigma \cap F_{\sigma+\delta} = F_{\sigma+\delta}$, it holds that:

$$dis(F_\sigma, F_{\sigma+\delta}) = 1 - \frac{\sum_{X \in F_\sigma \cap F_{\sigma+\delta}} \max\{0, 1 - \theta * |supp_D(X) - supp_D(X)|\}}{|F_\sigma \cup F_{\sigma+\delta}|}$$
$$= 1 - \frac{\sum_{X \in F_{\sigma+\delta}} \max\{0, 1-0\}}{|F_\sigma|}$$

$$\Rightarrow dis(F_\sigma, F_{\sigma+\delta}) = 1 - \frac{|F_{\sigma+\delta}|}{|F_\sigma|} \tag{4.14}$$

From the above equation, it arises that the larger the increase in the *minSupport* threshold value δ is, the smaller the enumerator $|F_{\sigma+\delta}|$ is (c.f. Equation 4.13) and thus the greater the distance between the two sets is.

FOCUS approach: Recalling Equation 4.9 and Equation 4.13, it holds that[2]:

$$dis(F_\sigma, F_{\sigma+\delta}) = \frac{\sum_{X \in F_\sigma \cup F_{\sigma+\delta}} |supp_D(X) - supp_D(X)|}{\sum_{X \in F_\sigma} supp_D(X) + \sum_{X \in F_{\sigma+\delta}} supp_D(X)}$$
$$= \frac{\sum_{X: \sigma \leq supp_D(X) < \sigma+\delta} supp_D(X)}{2 * \sum_{X \in F_\sigma} supp_D(X) - \sum_{X: \sigma \leq supp_D(X) < \sigma+\delta} supp_D(X)} \tag{4.15}$$

For simplicity, let $C = \sum_{X: \sigma \leq supp_D(X) < \sigma+\delta} supp_D(X)$; that is C considers all those itemsets whose support values lie in-between σ and $\sigma + \delta$. Thus:

$$\Rightarrow dis(F_\sigma, F_{\sigma+\delta}) = \frac{C}{2 * \sum_{X \in F_\sigma} supp_D(X) - C} \tag{4.16}$$

In the above equation, if the value of δ increases, the numerator C will also increase whereas the denumerator will decrease (c.f. Equation 4.13 as well). Thus, as δ increases, the dissimilarity also increases.

[2]The notation in this equation might confuse the reader, so we provide some further explanation. In particular, the term $\sum_{X \in F_\sigma \cup F_{\sigma+\delta}} |supp_D(X) - supp_D(X)|$ corresponds to the sum of the absolute differences of the two supports of itemsets in D, due to thresholds σ and $\sigma + \delta$, respectively. In fact, it corresponds to the sum of the supports of all those itemsets which appear in the difference set $F_\sigma - F_{\sigma+\delta}$. As far, as this set is not empty, this term is > 0.

Li-Ogihara-Zhou approach: From Equation 4.10 and Equation 4.13, it holds that:

$$I_1 + I_2 = \sum_{X,Y \in F_\sigma} d(X,Y) + \sum_{X,Y \in F_{\sigma+\delta}} d(X,Y)$$
$$= 2 * \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{\substack{X:\sigma \leq supp(X) < \sigma+\delta \\ Y:\sigma \leq supp(Y) < \sigma+\delta}} d(X,Y)$$

$$I_3 = \sum_{\substack{X \in F_\sigma \\ Y \in F_{\sigma+\delta}}} d(X,Y) = \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{\substack{X:\sigma \leq supp(X) < \sigma+\delta \\ Y:\sigma \leq supp(Y) < \sigma+\delta}} d(X,Y)$$

For simplicity, let $G = \sum_{\substack{X:\sigma \leq supp(X) < \sigma+\delta \\ Y:\sigma \leq supp(X) < \sigma+\delta}} d(X,Y)$; that is G considers all those itemsets whose support values lie in-between σ and $\sigma + \delta$. Thus:

$$\Rightarrow dis(F_\sigma, F_{\sigma+\delta}) = 1 - \frac{2I_3}{I_1 + I_2} = 1 - \frac{2(I_1 - G)}{2I_1 - G} = \frac{G}{2I_1 - G} \quad (4.17)$$

From the above equation, it arises that as δ increases, the enumerator G also increases, whereas the denumerator $(2I_1 - G)$ decreases (c.f. Equation 4.13 as well). Thus, dissimilarity increases as δ increases.

To summarize, Equation 4.14, 4.16 and 4.17 prove Lemma 1.

4.4.2 Effect of lattice representation on dissimilarity

Let $F_\sigma(D)$, $C_\sigma(D)$, $M_\sigma(D)$ be the sets of frequent, closed frequent and maximal frequent itemsets, respectively, extracted from D under (fixed) *minSupport* threshold σ. In this subsection, we study how the adopted lattice representation affects dissimilarity.

In Figure 4.3, an example is depicted which illustrates the effect of the different representations (FI, CFI, MFI) in the lattice structure. These figures affirm Equation 4.7, which states that the greater the compactness level is, the "smaller" the resulting lattice will be.

Below, we show for each of the presented measures, how they are affected by the adopted lattice compactness level (FI, CFI, MFI).

Lemma 2 *For either the Parthasarathy-Ogihara, or FOCUS or Li-Ogihara-Zhou approach, it holds that the more compact the addopted lattice representation (MFIs vs CFIs vs FIs) is, the larger the computed distance is.*

Proof:

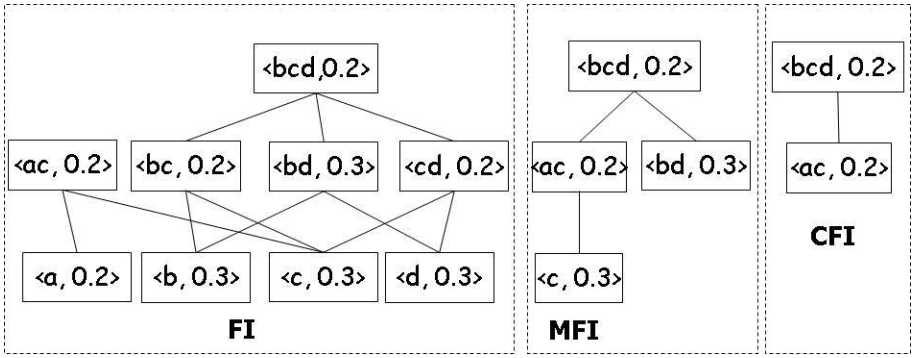


Figure 4.3: Effect of compactness level (FI, CFI, MFI) on the lattice structure ($\sigma = 0.1$)

Parthasarathy-Ogihara approach: From Equation 4.8, and since $F_\sigma \cap C_\sigma = C_\sigma, F_\sigma \cap M_\sigma = M_\sigma$, it holds that:

$$dis(F_\sigma, C_\sigma) = 1 - \frac{\sum_{X \in F_\sigma \cap C_\sigma} \max\{0, 1 - \theta * |supp_D(X) - supp_D(X)|\}}{|F_\sigma \cup C_\sigma|} = 1 - \frac{\sum_{X \in C_\sigma} \max\{0, 1 - 0\}}{|F_\sigma|} = 1 - \frac{|C_\sigma|}{|F_\sigma|} \quad (4.18)$$

also:

$$dis(F_\sigma, M_\sigma) = 1 - \frac{\sum_{X \in F_\sigma \cap M_\sigma} \max\{0, 1 - \theta * |supp_D(X) - supp_D(X)|\}}{|F_\sigma \cup M_\sigma|} = 1 - \frac{\sum_{X \in M_\sigma} \max\{0, 1 - 0\}}{|F_\sigma|} = 1 - \frac{|M_\sigma|}{|F_\sigma|} \quad (4.19)$$

From the above two equations, for the Parthasarathy-Ogihara measure it holds that:

$$dis(F_\sigma, C_\sigma) \leq dis(F_\sigma, M_\sigma) \quad (4.20)$$

FOCUS approach: Recalling Equation 4.9, and since $F_\sigma \cup C_\sigma = F_\sigma, F_\sigma \cup M_\sigma = F_\sigma$, it holds that:

$$dis(F_\sigma, C_\sigma) = \frac{\sum_{X \in F_\sigma \cup C_\sigma} |supp_D(X) - supp_D(X)|}{\sum_{X \in F_\sigma} supp_D(X) + \sum_{X \in C_\sigma} supp_D(X)} \quad (4.21)$$

$$= \frac{\sum_{X \in F_\sigma - C_\sigma} supp_D(X)}{2 * \sum_{X \in F_\sigma} supp_D(X) - \sum_{X \in F_\sigma - C_\sigma} supp_D(X)}$$

also:

$$dis(F_\sigma, M_\sigma) = \frac{\sum_{X \in F_\sigma \cup M_\sigma} |supp_D(X) - supp_D(X)|}{\sum_{X \in F_\sigma} supp_D(X) + \sum_{X \in M_\sigma} supp_D(X)} \quad (4.22)$$
$$= \frac{\sum_{X \in F_\sigma - M_\sigma} supp_D(X)}{2 * \sum_{X \in F_\sigma} supp_D(X) - \sum_{X \in F_\sigma - M_\sigma} supp_D(X)}$$

where $F_\sigma - C_\sigma$ is given by Equation 4.3 and $F_\sigma - M_\sigma$ is given by Equation 4.5.

From the above two equations, for the FOCUS measure it holds that:

$$dis(F_\sigma, C_\sigma) \leq dis(F_\sigma, M_\sigma) \quad (4.23)$$

Li-Ogihara-Zhou approach: From Equation 4.10, it holds that:

$$I_1 + I_2 = \sum_{X,Y \in F_\sigma} d(X,Y) + \sum_{X,Y \in C_\sigma} d(X,Y)$$
$$= 2 * \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{X,Y \in F_\sigma - C_\sigma} d(X,Y) = 2 * I_1 - \sum_{X,Y \in F_\sigma - C_\sigma} d(X,Y)$$

$$I_3 = \sum_{\substack{X \in F_\sigma \\ Y \in C_\sigma}} d(X,Y) = \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{X,Y \in F_\sigma - C_\sigma} d(X,Y)$$
$$= I_1 - \sum_{X,Y \in F_\sigma - C_\sigma} d(X,Y)$$

For simplicity, let $K = \sum_{X,Y \in F_\sigma - C_\sigma} d(X,Y)$. Thus:

$$dis(F_\sigma, C_\sigma) = 1 - \frac{2(I_1 - K)}{2I_1 - K} = \frac{K}{2I_1 - K} \quad (4.24)$$

Similarly for the FI-MFI comparison case, it holds that:

$$I_1 + I_2 = \sum_{X,Y \in F_\sigma} d(X,Y) + \sum_{X,Y \in M_\sigma} d(X,Y)$$
$$= 2 * \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{X,Y \in F_\sigma - M_\sigma} d(X,Y) = 2 * I_1 - \sum_{X,Y \in F_\sigma - M_\sigma} d(X,Y)$$

$$I_3 = \sum_{\substack{X \in F_\sigma \\ Y \in M_\sigma}} d(X,Y) = \sum_{X,Y \in F_\sigma} d(X,Y) - \sum_{X,Y \in F_\sigma - M_\sigma} d(X,Y)$$
$$= I_1 - \sum_{X,Y \in F_\sigma - M_\sigma} d(X,Y)$$

For simplicity, let $L = \sum_{X,Y \in F_\sigma - M_\sigma} d(X,Y)$. Thus:

$$dis(F_\sigma, M_\sigma) = 1 - \frac{2(I_1 - L)}{2I_1 - L} = \frac{L}{2I_1 - L} \tag{4.25}$$

From Equation 4.24 and Equation 4.25, for the Li-Ogihara-Zhou measure it holds that:

$$dis(F_\sigma, C_\sigma) \leq dis(F_\sigma, M_\sigma) \tag{4.26}$$

Equations 4.20, 4.23 and 4.26 prove Lemma 2.

Consequently, Sections 4.4.1 and 4.4.2 prove the dependency of the dissimilarity measures between sets of frequent itemsets, on the mining parameters used for their extraction, namely on the *minSupport* threshold and on the adopted lattice representation (FI, CFI or MFI).

4.5 Experimental evaluation

To evaluate the theoretical results presented in Sections 4.4.1 and 4.4.2, we experimented with the different dissimilarity measures (Parthasarathy-Ogihara, FOCUS, Li-Ogihara-Zhu) on datasets from the frequent itemset mining dataset repository [30].

For the experiments we used both synthetic and real datasets that are further characterized either as dense or as sparse [3]. The dataset characteristics are described in Table 4.2. For the extraction of the FI, CFI, MFI sets we used the MAFIA program [18].

dataset	**# trans.**	**# items**	**|avg. trans.|**	**type**	**type**
T10I4D100K	100,000	1,000	10	sparse	synthetic
chess	3,196	76	37	dense	real
connect	67,557	130	43	dense	real

Table 4.2: Dataset characteristics

For the FOCUS approach (c.f. Section 4.3.2), we used the upper bound of the dissimilarity measure proposed by the authors without re-querying the original raw data space. This decision, as already mentioned, is justified by the fact that we are interested on how patterns (i.e., frequent itemsets) capture the similarity features contained in the original raw data. For the Parthasarathy-Ogihara approach (c.f. Section 4.3.1), we used $\theta = 1$, considering that both structure and measure components contribute equally to the final dissimilarity score.

[3] Market basket data are usually sparse, in contrary to telecommunications and cencus data that are dense.

4.5.1 Comparing dissimilarity in data and FI spaces

The first set of experiments evaluates the argument that dissimilarity in pattern space can be adopted to discuss dissimilarity in data space. In particular, we select a popular representation, FI, and a specific *minSupport* threshold σ for their generation, while we modify the dataset by adding different proportions of noise. Then, we compare the dissimilarity measured in the FI space with respect to the dissimilarity enforced (by adding noise) in the original raw data space.

The scenario is the follows: Starting with an initial dataset D and for a specific *minSupport* threshold σ, we extracted $F_\sigma(D)$. Then, at each step, we modified an increased number $0\%, 5\%, \ldots, 50\%$ of the transactions of D. The selection of the transactions to be affected was performed in a random way, and for each selected transaction we modified a certain percentage (in particular, 50%) of its items. The modification we made was that the selected item values were reset to 0 (in a preprocessing step both datasets of the experiments where transformed into binary format). As such, the derived pattern sets $F_\sigma(D_{p\%})$ were subsets of the initial pattern sets $F_\sigma(D_{0\%})$. Then, we compared the noised pattern sets $F_\sigma(D_{p\%})$ with the initial "unnoised" pattern set $F_\sigma(D_{0\%})$.

Regarding the generation of the pattern sets, we used $\sigma = 80\%$ for the dataset $D = chess$ and $\sigma = 0.5\%$ for the dataset $D = T10I4D100K$. The choice of these values was based on the cardinality analysis presented in [102]. The results for the two datasets are illustrated in Figure 4.4.

As illustrated in these figures, for both the sparse and the dense datasets, as the dataset becomes noisier, the distance between the initial pattern set and the new ("noisy") pattern set becomes larger, for all the dissimilarity measures. Moreover, if we consider that the dissimilarity in data space is the *edit distance* between the original and the noisy dataset, we can see that none of the approaches captures in pattern space the dissimilarity that appears in the original raw data space.

A comparative study of these figures shows that the effect of noise is more destructive in the case of the dense datasets like *chess*. In Figure 4.4 (bottom), the dissimilarity increases quickly up to the upper bound (i.e., dissimilarity = 1). This can be explained by the fact that small changes in a dense dataset may cause critical changes in the produced FI lattice. On the other hand, this is not the case for a sparse datasets like *T10I4D100K*, which appears to be more robust in the increase of the dataset noise as it is shown in Figure 4.4

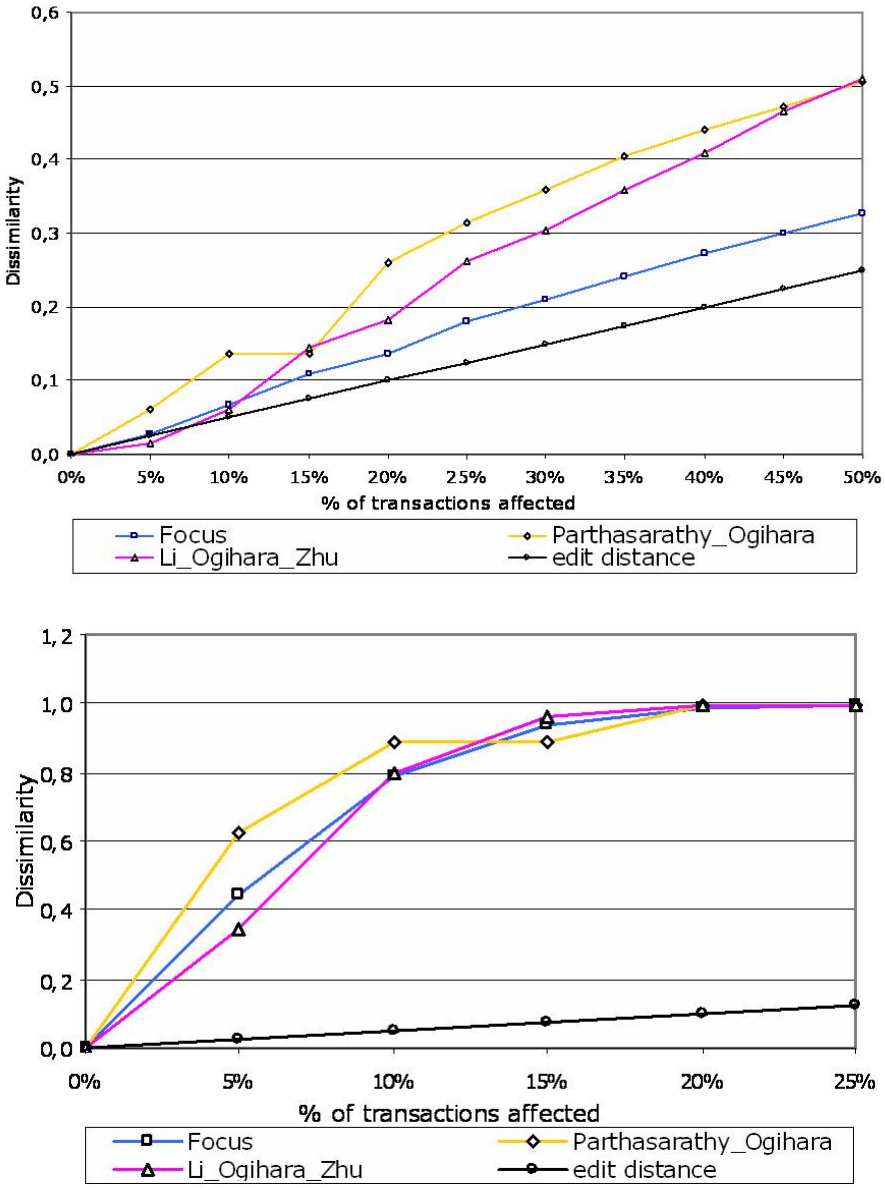


Figure 4.4: Impact of dataset noise increase on FI dissimilarity: $D = T10I4D100K$, $\sigma = 0.5\%$ (top), $D = chess$, $\sigma = 80\%$ (bottom).

(top).

4.5.2 Effect of *minSupport* threshold

In this section, we evaluate the effect of the *minSupport* threshold on the computed dissimilarity score.

The scenario is as follows: For each dataset D, we fixed the initial *minSupport* threshold σ and varied the increase δ in the *minSupport* values in the range $\delta_0, \delta_1, \ldots, \delta_n (\sigma + \delta_i \leq 1)$. Then, we generated the corresponding FIs for the different *minSupport* values, namely $\sigma+\delta_0, \sigma+\delta_1, \ldots, \sigma+\delta_n$. After that, we compared $FI_{\sigma+\delta_i}$ with the initial $FI_{\sigma+\delta_0}$. We chose the σ, δ values for the different datasets based on the cardinality analysis presented in [102]. Since our analysis does not depend on specific support values we chose parameters that yield a reasonable amount of patterns, that is: in case of the $D = T10I4D100K$ dataset, we choose $\sigma = 0,5\%$ for the initial support and the values $0\%, 0.5\%, \ldots, 4.5\%$ for the *minSupport* increase δ, whereas in case of the $D = chess$ dataset, we choose $\sigma = 90\%$ for the initial support and the values: $0\%, 1\%, \ldots, 9\%$ for the *minSupport* increase δ.

The results for the two datasets are illustrated in Figure 4.5. As it is shown in these charts, the larger the increase in the *minSupport* threshold value δ is, the larger the dissimilarity between the corresponding pattern sets is, and this holds for all dissimilarity measures.

More comments on the results: Both Parthasarathy-Ogihara and FOCUS are based on some kind of $1-1$ matching between (identical) itemsets of the two sets. They differ in the fact that Parthasarathy-Ogihara considers only those itemsets that appear in the set intersection $(A \cap B)$, whereas FOCUS considers also itemsets that appear in $(A - A \cap B)$ and $(B - A \cap B)$, that is, it considers itemsets in the set union $(A \cup B)$. If an itemset appears in one of the two sets, Parthasarathy-Ogihara will increase the total dissimilarity score by 1 (in absolute value), whereas FOCUS will increase the total score by the support value of the specific itemset. This is the reason why FOCUS results in lower dissimilarity scores comparing to Parthasarathy-Ogihara. On the other hand, the Li-Ogihara-Zhou approach follows a different rationale than Parthasarathy-Ogihara and FOCUS: It performs a M-N matching between the itemsets of the two sets and considers partial similarity between itemsets.

Note also the difference in the behavior depending on the dataset characteristics (dense, sparse). The *minSupport* effect in case of the dense datasets like *chess* (Figure 4.5, bottom) is smoother comparing to the sparse datasets like *T10I4D100K* ((Figure 4.5, top)). This is probably due to the fact that in a dense dataset the itemsets are spread more smoothly at the different support levels comparing to a sparse dataset. This is affirmed in Figure 4.6 where the distribution of the number of itemsets at the different support levels is depicted for each dataset.

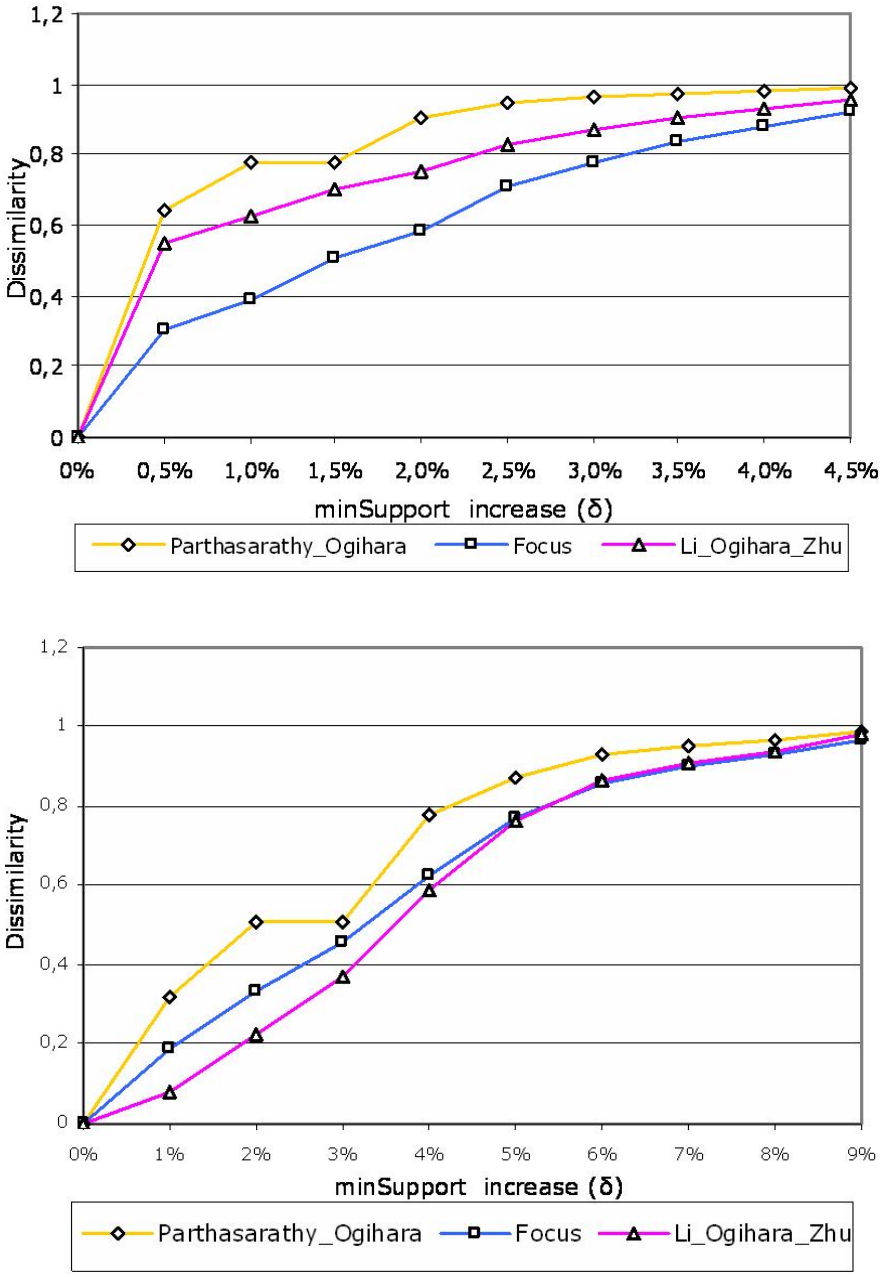
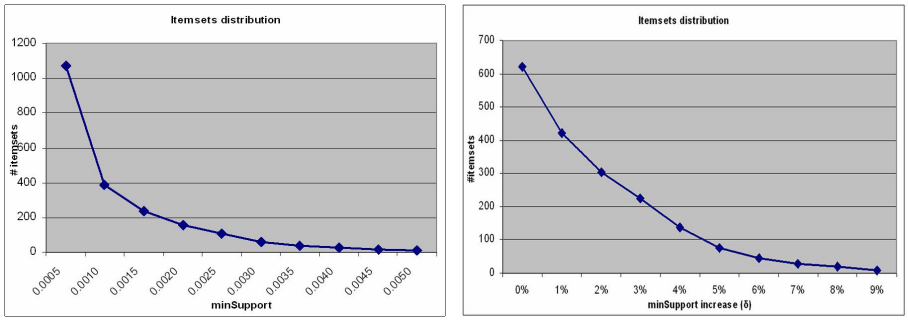


Figure 4.5: Impact of *minSupport* increase δ on FI dissimilarity: $D = T10I4D100K$, $\sigma = 0.5\%$ (top), $D = chess$, $\sigma = 90\%$ (bottom).

To summarize, experiments in this subsection have confirmed our theoretical analysis (Lemma 1, Section 4.4.1) regarding the dependency of the dissimilarity scores between two pattern sets of frequent



(a) (b)

Figure 4.6: Itemsets distribution at different support levels for (a) $D = T10I4D100K$, $\sigma = 0.5\%$ and (b) $D = chess$, $\sigma = 90\%$

itemsets on the *minSupport* threshold that was used for their generation. Indeed, as the *minSupport* becomes more "selective", the dissimilarity increases. Generalizing this result, we can state that the more "selective" the *minSupport* threshold is, the less informative the set of frequent itemsets becomes with respect to the original raw data space. Thus, when comparing two datasets in terms of the corresponding pattern sets, the dissimilarity score is subjective to the *minSupport* threshold used for the generation of frequent itemsets patterns.

4.5.3 Effect of lattice representation

In this section, we study the effect of the different lattice representations on dissimilarity. More specifically, we set the value of the *minSupport* threshold parameter σ to a fixed value and calculate the dissimilarity scores between the different lattice representations under the same noise level, i.e., $dis(F_\sigma(D_{p\%}), C_\sigma(D_{p\%}))$, $dis(F_\sigma(D_{p\%}), M_\sigma(D_{p\%}))$. The results for FI - CFI and FI - MFI dissimilarity cases are illustrated in Figure 4.7.

These charts point out the dependency of the dissimilarity scores on the adopted compactness level of the frequent itemsets lattice. As illustrated in these figures, CFIs can well capture the behavior of FIs whereas MFIs cannot; this is true for all the datasets.

However, the degree of difference between FI-CFI and FI-MFI dissimilarities scores seems to be subject to the dataset characteristics (sparse vs. dense). More specifically, in case of the sparse dataset $T10I4D100K$, CFIs manage to fully capture the behavior of FIs at every noise level while MFIs smoothly approximate FIs as the dataset becomes more noisy (c.f. Figure 4.7, top). On the

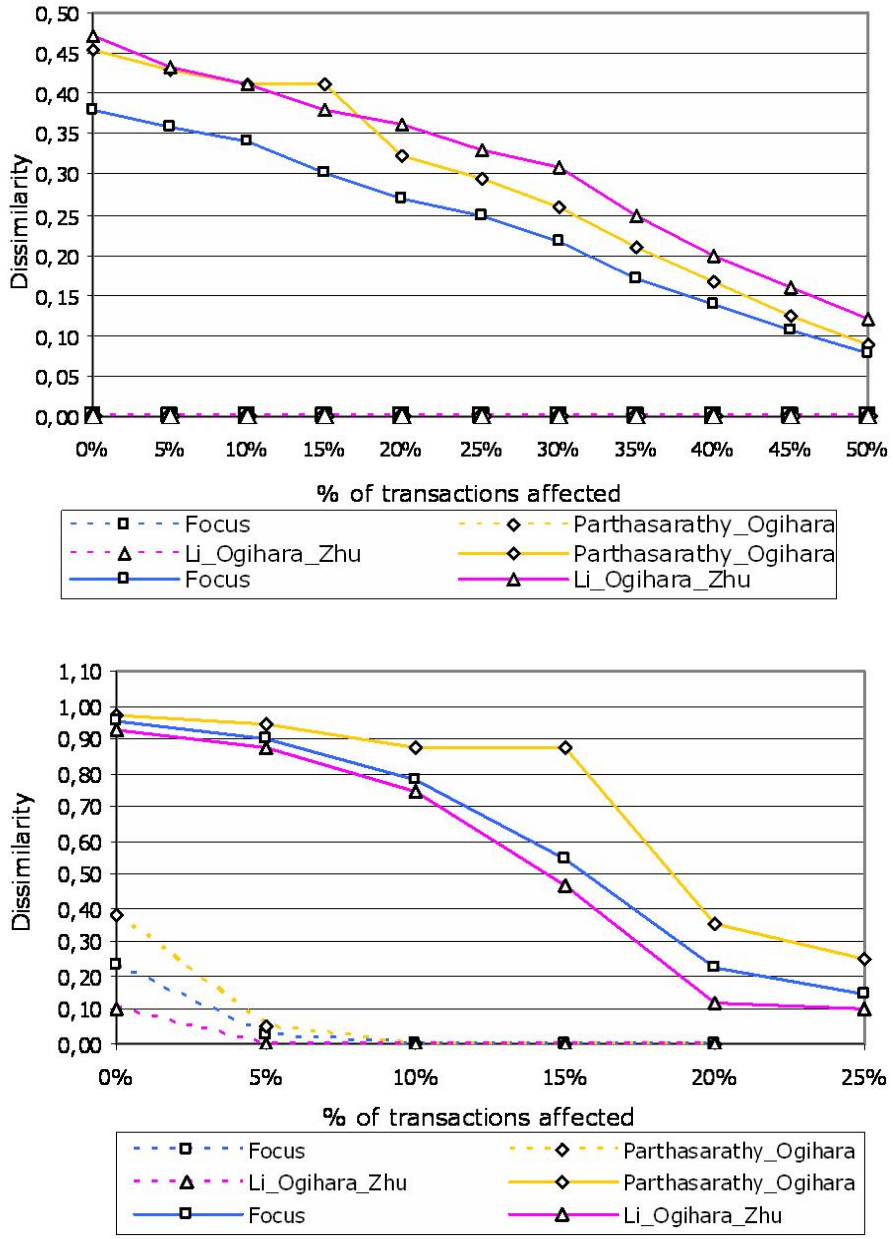


Figure 4.7: Impact of noise on dissimilarity for FI-CFI (dotted lines), FI-MFI (solid lines): $D = T10I4D100K$, $\sigma = 0.5\%$ (top), $D = chess$, $\sigma = 80\%$ (bottom).

other hand, in case of the dense dataset *chess*, CFIs are still close to FIs although they do not fully capture their behavior (c.f. Figure 4.7, bottom). As shown in these figures, the differences in FIs-CFIs, FIs-MFIs are more "strong" in case of the dense datasets.

To summarize, experiments in this subsection show that the adopted representation for the frequent itemsets lattice affects the derived dissimilarity scores, thus confirming the theoretical analysis regarding the dependency of dissimilarity on the lattice representation (Lemma 2, Section 4.4.2). In addition, it seems that CFIs are very good representatives for FIs, whereas this is not the case for MFIs. It turns out that the more compact the representation of the pattern space is, the less informative this space becomes with respect to the initial data space from which patterns have been extracted.

4.6 Summary

In this chapter, we investigated whether dissimilarity between sets of frequent itemsets could serve as a measure of dissimilarity between the original datasets from which these pattern sets have been extracted through some FIM algorithm. We presented the parameters that affect the problem, namely the *minSupport* threshold used for itemsets generation and the *compactness level of the lattice representation* (frequent itemsets, closed frequent itemsets and maximal frequent itemsets). Both theoretical analysis and experimental results confirmed that the more "restrictive" the mining parameters are, the larger the dissimilarity between the two sets is, for all the approaches proposed so far in the literature.

This result declares that utilizing pattern comparison for data comparison is not as straightforward as related work has argued, but is subjective to the mining parameters used for the extraction of patterns. A possible explanation for the similar behavior of the different measures w.r.t. the mining parameters is that they all follow the same generic schema (c.f. Section 4.3.4). More specifically, all measures rely either on the union or on the intersection of the sets of itemsets to be compared. However, the choice of the *minSupport* threshold value and of the compactness level (FI, CFI or MFI) strongly affect the resulting set of itemsets. Also, as shown in the experiments a small change in the dataset might result in very different sets of itemsets. All these facts indicate that, when utilizing pattern comparison for data comparison, one should also take into account the mining parameters used for the extraction of patterns. A reasonable choice when comparing two datasets would be to use those sets of itemsets extracted at the lowest *minSupport* threshold value and at the less compact lattice representation level (i.e., FIs); such sets are most probable to carry more of the information contained in the original raw datasets. Especially for the lat-

tice representation level, the behavior of CFIs turns out to be very close to that of FIs, thus one could employ CFIs for the dataset comparison. However, such a choice also depends on the dataset characteristics (dense, sparse); as it raises from the experiments, dense datasets are less robust to the mining parameters and dataset variations comparing to sparse datasets.

Early versions of this study appear in [69, 71], whereas an extended version has been submitted in [77]

4.7 Open issues

An open issue is the "discovery" of a dissimilarity measure that would be more robust to the mining parameters and would better preserve the original raw data space characteristics in the pattern space. From the current study, it seems that a M-N comparison between the itemsets of the two sets is more stable than a 1-1 comparison. Also, the results are more stable when one takes into account the itemsets that appear in the union of the two sets to be compared instead of those appearing in the set intersection. However, instead of looking at itemsets appearing in the union or in the intersection of the two sets, several other alternatives exist. For example, one could employ a spectrum-like approach by considering for the comparison the set of k-itemsets, e.g., all 2-itemsets, instead of all itemsets; this might be a more reasonable and unbiased reflection of the composition of the underlying datasets.

Chapter 5

Estimating Similarity between Decision Trees (and Classification Datasets) - A Semantic Comparison Framework

In this chapter, we present a general framework for similarity estimation between either decision trees or classification datasets. Our approach exploits the information provided by the partitioning that each decision tree performs over the attribute space of the dataset from which it has been induced. We demonstrate the usefulness and applicability of our framework in estimating the *semantic similarity* between decision tree models induced from different subsamples of classification datasets. We evaluate the performance of our semantic similarity measures with respect to the empirical semantic similarity measure, which is estimated on the basis of independent hold-out test sets.

The chapter is organized as follows: In Section 5.1 we make an introduction to the problem. In Section 5.2, we describe the basic concepts of the decision tree models that are necessary for the understanding of this work. In Section 5.3, we describe our similarity framework. In Section 5.4, we experimentally evaluate the proposed semantic similarity measures between decision tree models. In Section 5.5, we present the related work. In Section 5.6, we summarize the work, whereas in Section 5.7 we discuss the emerged research issues.

Index terms decision tree dissimilarity, semantic similarity, structural differences.

5.1 Introduction

Decision trees (DT) are among the most popular learning paradigms in the area of Data Mining thanks to a number of attractive properties, such as scalability to large datasets and relative easiness of interpretation by end users, provided of course that their size does not exceed certain limits.

On the other hand, decision tree models are also well known for their *instability*; small changes in the training dataset may result in completely different decision trees, which might contain different tests on the predictive attributes or even different predictive attributes. Note however, that two decision trees, though *structurally different*, may describe the same concept, i.e., they may be *semantically similar*. Intuitively, two decision trees should be semantically similar, if they have been learned from datasets that come from the same generating distribution.

Semantic similarity in the presence of structural differences might result for a variety of reasons, such as, superficially different tests on attributes which are in fact equivalent, different attributes that convey the same information due to attribute redundancy, or simply because, the same concept can be described in many different ways which are nevertheless semantically equivalent. To capture the degree of semantic similarity between decision trees we need a measure of the semantic similarity of the concepts that they describe.

There is a plethora of reasons for which the definition of semantic similarity measures between decision trees is required. By far, the most important is being able to report to the end users whether the differences observed in decision trees induced from different training sets (which however are thought to come from the same data generating distribution) are only structural and do not correspond to semantic differences, or whether the concepts described by the decision trees are indeed semantically different. In the latter case, a quantification of this semantic difference would be useful.

Moreover, the availability of a similarity measure on classification models makes it possible to apply a number of standard mining tasks on classification models rather than on raw data, resulting in what we could call *meta-analysis* or *meta-mining* tasks. For example, the semantic similarity measures can be used to cluster different sites into groups of similar behavior according to the decision tree models

learned locally on each of the sites; e.g., clustering the different branches of a bank according to the credit strategy they adopt (as this is expressed by the decision tree model extracted locally at each branch).

Also, in case of dynamic data like data streams, similarity could be employed in order to monitor the evolution of the induced decision tree models across the time axis. A crucial question in this case is whether the concept, which is captured in the decision tree models, remains (about) the same or there are concept shifts in the population. Based on such information, a telecommunication company, for example, could adapt its billing strategy on time.

Similarity could be also employed in order to study the effect of the decision tree learning parameters, like pruning level or induction algorithm, on the resulting decision tree models. As another application, consider the comparison of a decision tree model extracted from the data to a golden standard decision tree model (recall Figure 1.3), so as to know how far away is the extracted (real) model from the expected (possibly artificial) model.

In this chapter, we propose a general similarity estimation framework that includes as special cases: i) the estimation of semantic similarity between decision trees and, ii) the estimation of similarity between classification datasets based on the different probability distributions that rule these sets, namely the *marginal probability distribution of the attributes*, the *joint attribute-class probability distribution* and the *attribute-conditional class probability distribution*. The framework is based on the comparison of the partitionings that the decision trees define over a given attribute space considering the probability distribution of the data space over that partitioning. Similar ideas have been previously used for dataset comparison [32, 97, 82], however, to the best of our knowledge, this is the first time that the semantic similarity between decision trees, on which we focus in this work, is explored. Depending on the available information regarding the probability distribution that generated the data, we get different instantiations of the decision tree semantic similarity measure. To evaluate the proposed *decision tree semantic similarity measures* we compare them with the empirical semantic similarity, which is estimated by applying the decision trees on independent test sets.

5.2 Preliminaries on decision trees

In this section, we provide some basic concepts on decision tree models that are necessary for the understanding of this chapter. More general information on the DT models can be found in Section 2.3.

Consider a classification problem described through a vector of *predictive attributes* $A = (a_1, a_2, ..., a_m)$ and a *class attribute* C. Each attribute a_i has domain $d(a_i)$ and the domain of the class attribute is $d(C) = \{c_1, c_2, ..., c_k\}$, where k is the number of classes. Predictive attributes might be either numeric, categorical or ordinal; usually, numeric attributes are considered [65].

The cartesian product $S_A = d(a_1) \times d(a_2) \cdots \times d(a_m)$ describes the *attribute space*, whereas the cartesian product $S_{(A,C)} = S_A \times d(C)$ defines the *attribute-class space*.

The goal of a decision tree model, is to learn a *prediction function* $f : S(A) \rightarrow dom(C)$. Towards this goal, a dataset D of problem instances, known as *training set*, is provided as input to the decision tree induction algorithm. The training set D should be drawn from the joint distribution $P(A, C)$ of the predictive attributes A and the class attribute C, so as to be representative of the classification problem at hand. Definitely, the choice of D is crucial since it affects the generalization accuracy of the extracted decision tree over future, previously unseen problem instances.

Let $U(A)$ be the *uniform distribution* over the attribute space and $P(A) = \sum_C P(A, C)$ be the *marginal distribution of attributes*, also defined over the attribute space.

As a running example for the comparison problem, let us consider the decision trees illustrated in Figure 5.1. Both trees refer to the same classification problem, which is described through two predictive attributes and one class attribute. The predictive attributes are Age and $Income$ with domains $(10 \leq Age \leq 70)$ and $(20K \leq Income \leq 100K)$, respectively. The values for the class attribute C are, e.g., $C_1 =$ "$HighRisk$", $C_2 =$ "$LowRisk$".

Table 5.1 summarizes the symbols used throughout this chapter.

5.3 Measuring deviation using decision trees

A decision tree DT induced from a training set D partitions the attribute space into a set of non–overlapping regions $R_{DT} = \{r_i | i = 1 \ldots |R_{DT}|\}$, through its leaf nodes. The partitionings of the decision trees of our running example (c.f. Figure 5.1) are depicted in

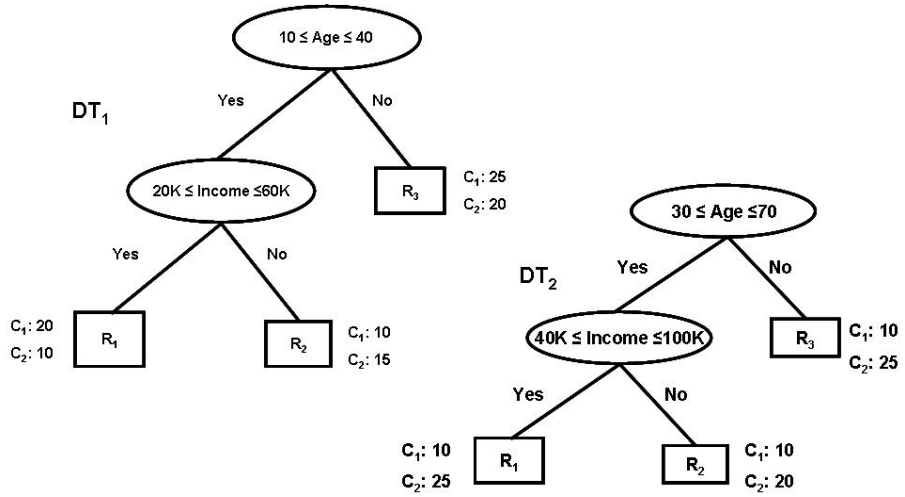


Figure 5.1: Two decision trees, DT_1, DT_2

Symbol	Description
DT	decision tree
R_{DT}	the partitioning of DT
A	predictive attributes
C	class attribute
S_A	attribute space
$S_{(A,C)}$	attribute - class space
$U(A)$	uniform distribution over S_A
$P(A)$	distribution of predictive attributes
$P(A,C)$	joint distribution of the predictive attributes and the class attribute
$P(C\|A)$	conditional distribution of the class attribute given the predictive attributes

Table 5.1: List of symbols for Chapter 5

Figure 5.2.

The partitioning R_{DT} defined by the decision tree DT can be considered as an approximation of the joint class-attribute distribution $P(A, C)$, in the form of a histogram (Section 5.3.1). Each bin of the histogram corresponds to a region of the partitioning, and respectively, to a leaf node of the decision tree. Different decision trees result in different partitionings; in Section 5.3.2, we show how to derive the overlay partitioning of two decision trees and how to estimate its statistics, for each dataset, depending on whether we have access to the original raw datasets or not. Based on the overlay partitioning, we define various similarity measures between decision trees and classification datasets (Section 5.3.3).

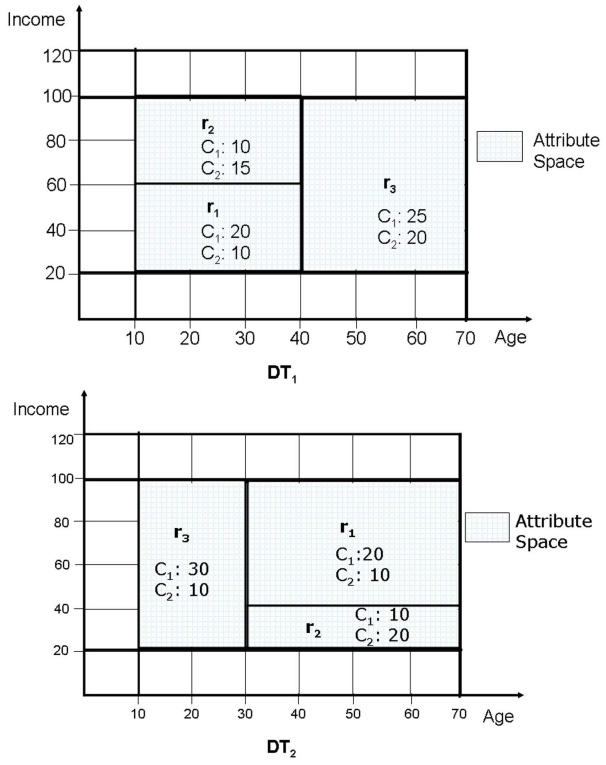


Figure 5.2: The partitioning of DT_1 (top), DT_2 (bottom)

5.3.1 Decision tree partitioning information

Each region $r \in R_{DT}$ is characterized by a structure and a measure component that are directly derived from the decision tree.

Structure component The *structure component* of the region is defined as the conjunction of the test conditions on the attributes along the corresponding tree path from the root to the leaf node associated with that region:

$$r.s = \{\wedge t_i(a_i), i = 1...m\}$$

Test conditions are usually numeric and can be expressed in the form: $t(a) = \min_a(r) \leq a \leq \max_a(r)$ denoting the min and max values of attribute a in region r.

Let us also define the length of a test condition on an attribute a as: $|t(a)| = \max_a(r) - \min_a(r)$ and the length of the domain of a as: $|dom(a)| = \max_a - \min_a$. Note here that if an attribute a is not included in the structure component $r.s$ of a leaf node, i.e., no test on that attribute has been included in the path from the root

to the leaf node during the training phase, then the test condition on that attribute is: $t(a) = \min_a \leq a \leq \max_a$, that is, the attribute a can take any value from its domain within the region r. Thus, the structure component of a region contains test conditions over all (i.e. m) predictive attributes of the problem.

Measure component The *measure component* of a region is defined as the number of training set instances that fall into this region for each of the problem classes and, it depends on the training set D used for the induction of the decision tree:

$$r.m_D = [n_{c_1}, n_{c_2}, \ldots, n_{c_k}]$$

where $n_{c_i}, i = 1 \ldots k$ is the number of instances that fall into region r and belong to class c_i.

The total number of instances in region r, can be easily obtained by summing up the number of instances in r for each of the problem classes; thus, the size of the measure component is:

$$|r.m_D| = \sum_{1 \leq i \leq k} n_{c_i}$$

The class label assigned to r, is the label of the majority class:

$$r.cl = \arg\max_{c_i} r.m_D$$

For example, the region r_1 of DT_1 (c.f. Figure 5.2) can be described as:

$$r.s = \{(10 \leq Age \leq 40) \wedge (20 \leq Income \leq 60)\}$$
$$r.m_D = [n_{c_1} = 20, n_{c_2} = 10]$$

As it also arises from this example, both $r.s$ and $r.m_D$ are directly derivable from the decision tree, without any further post–processing.

Region probability, P(r): A region describes a decision area of the problem. The *probability of a region* represents the probability that some instance x will follow the corresponding decision tree path. Formally, this probability is given by:

$$P(r) \equiv Pr(x \in r) = \int_r P(A)dA$$

where $P(A)$ is the probability density function of the instances. However, since we do not have access to the exact form of $P(A)$, we should use the data to make an estimation of it.

More specifically, if we consider the training set D used for the induction of the decision tree DT, we can make a *training set dependent estimate* of $P(r)$ as follows[1]:

$$\mathbf{P_D}(\mathbf{r}) = \frac{|r.m_D|}{N_D} \tag{5.1}$$

This is simply the percentage of the training set instances (N_D) that fall in r ($|r.m_D|$).

The vector:

$$\mathbf{P_D}(\mathbf{A}) = [\mathbf{P_D}(\mathbf{r_i})|r_i \in R_{DT}] \tag{5.2}$$

is an approximation of $P(A)$, from the dataset D. One can think of this vector as a histogram, where the bins are defined by the regions of R_{DT}.

Joint region-class probability, $\mathbf{P}(\mathbf{r}, \mathbf{c})$: Apart from $P(A)$, we can also approximate $P(A, C)$ by exploiting the measure component of the regions, which describe the distribution of the training set instances within the different problem classes.

The *joint region-class probability* of observing a region r under a class c is given by:

$$\mathbf{P_D}(\mathbf{r}, \mathbf{c}) = \frac{r.n_c}{N_D} \tag{5.3}$$

This is simply the percentage of the training set instances (N_D) that fall into region r and belong to class c ($r.n_c$); this estimation also depends on the training set D.

The vector:

$$\mathbf{P_D}(\mathbf{r}, \mathbf{C}) = [\mathbf{P_D}(\mathbf{r}, \mathbf{c_j}), j = 1 \ldots k]$$

describes the joint region-class probability distribution in r. One can think of this vector as a one dimensional histogram with k bins, one for each class.

The matrix:

$$\mathbf{P_D}(\mathbf{A}, \mathbf{C}) = [\mathbf{P_D}(\mathbf{r_i}, \mathbf{c_j})|r_i \in R_{DT}, c_j \in C] \tag{5.4}$$

in which each row corresponds to a region $r_i \in R_{DT}$ and each column to a class $c_j \in C$, is an approximation of the joint distribution $P(A, C)$ from the dataset D in the form of a multidimensional histogram; the bins of the histogram are defined by $R_{DT} \times C$.

[1] We denote the actual distribution by P and its estimation by $\mathbf{P}$.

Conditional class-region probability, $\mathbf{P(c|r)}$: Furthermore, the measure component can provide us with an estimate of the conditional probability of a class c given the region r:

$$\mathbf{P_D(c|r)} = \frac{r.n_c}{|r.m_D|}$$

This is simply the percentage of the region instances ($|r.m_D|$) that belong to class c ($r.n_c$).

The vector:

$$P_D(C|r) = [\mathbf{P_D(c_j|r)}, j = 1 \dots k]$$

describes the class conditional distribution in region r. One can think of this vector as a one dimensional histogram with k bins, one for each problem class.

The matrix:

$$\mathbf{P_D(C|A)} = [\mathbf{P_D(c_j|r_i)}|r_i \in R_{DT}, c_j \in C] \tag{5.5}$$

in which each row corresponds to a region $r_i \in R_{DT}$ and each column to a class $c_j \in C$, is an approximation of $P(C|A)$ from the dataset D in the form of a multidimensional histogram; the bins of the histogram are defined by $R_{DT} \times C$.

5.3.2 Decision tree overlay partitioning information

Let R_{DT_1} and R_{DT_2} be the partitionings defined by the decision trees DT_1 and DT_2, respectively. Overlaying the two partitionings, a finer partitioning $R_{DT_1 \times DT_2}$ arises, where each region r in it is the result of overlaying some region $r_i \in R_{DT_1}$ with some region $r_j \in R_{DT_2}$, that is $r = r_i \cap r_j$. Figure 5.3 displays the overlay partitioning of the DT_1, DT_2 partitionings illustrated in Figure 5.2.

The goal is to estimate the region probability $P(r)$, the joint region-class probability $P(r, c)$ and the class conditional probability $P(c|r)$ for each region $r \in R_{DT_1 \times DT_2}$ and each class $c \in C$. To this end, we rely on the observation that each region r in the overlay is also a hyper-rectangle and thus it can be also described through a structure and a measure component.

In Figure 5.4, we appose the result of overlaying for the regions $R_1 \in DT_1$ and $R_3 \in DT_2$.

Structure component of the overlay regions The *structure component* of the overlay region $r_i \cap r_j$ is easily defined through the intersections of the DT regions that participate in its formation:

$$r_i \cap r_j.s := \{\wedge t_i(a_i), i = 1 \dots m\}$$

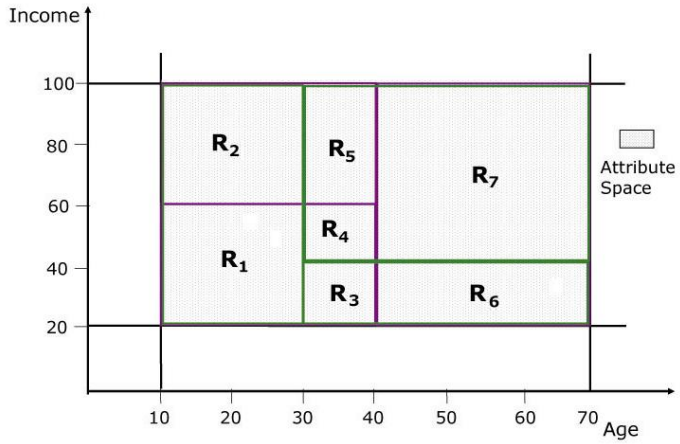


Figure 5.3: The overlay partitioning $R_{DT_1 \times DT_2}$

$$t(a) := \min_a(r_i \cap r_j) \leq a \leq \max_a(r_i \cap r_j)$$
$$\min_a(r_i \cap r_j) := \max(\min_a(r_i), \min_a(r_j))$$
$$\max_a(r_i \cap r_j) := \min(\max_a(r_i), \max_a(r_j))$$

If $\max_a(r_i \cap r_j) \leq \min_a(r_i \cap r_j)$, the overlay region $r_i \cap r_j$ is not defined since the regions are disjoint.

Measure component of the overlay regions On the other hand, the estimation of the measure component of $r_i \cap r_j$ is dataset dependent; the obvious choices for the dataset are D_1, D_2 and $D_1 \cup D_2$. However, even if we do not have anymore access to any of these datasets, we can still estimate the measure component of the overlay regions based on the measure components of the regions of the original partitionings R_{DT_1} and R_{DT_2}. We refer to the first case as the data dependent probability estimation and to the second case as the pattern dependent probability estimation; we analyze both cases below.

- **Data dependent probability estimation:** If we have access to the original raw datasets, we can get the *exact measure* of the overlay regions by simply projecting each dataset $D \in \{D_1, D_2, D_1 \cup D_2\}$ on $R_{DT_1 \times DT_2}$.

  Thus, by projecting D_1 on $R_{DT_1 \times DT_2}$ we can have the exact measure $r_i \cap r_j.m_{D_1}$:

$$r_i \cap r_j.m_{D_1} = \frac{|t \in r_i \cap r_j, t \in D_1|}{N_{D_1}} \tag{5.6}$$

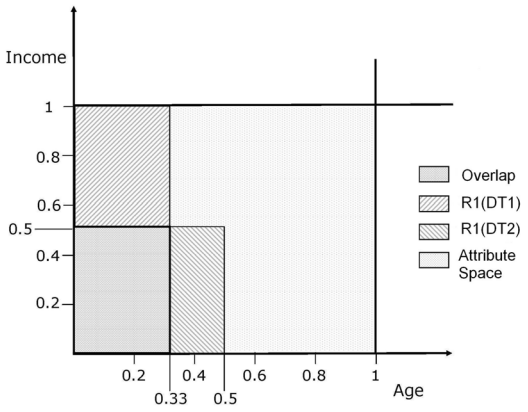


Figure 5.4: Overlaying regions $R_1 \in DT_1$ and $R_3 \in DT_2$ (instance values have been normalized)

Similarly, by projecting D_2 on $R_{DT_1 \times DT_2}$ we can have the exact measure $r_i \cap r_j.m_{D_2}$:

$$r_i \cap r_j.m_{D_2} = \frac{|t \in r_i \cap r_j, t \in D_2|}{N_{D_2}} \tag{5.7}$$

Finally, by projecting $D_1 \cup D_2$ on $R_{DT_1 \times DT_2}$ we can have the exact measure $r_i \cap r_j.m_{D_1 \cup D_2}$:

$$r_i \cap r_j.m_{D_1 \cup D_2} = \frac{|t \in r_i \cap r_j, t \in D_1 \cup D_2|}{N_{D_1} + N_{D_2}} \tag{5.8}$$

- **Pattern dependent probability estimation:** Even if we do not have access to the original raw datasets, we can still make an estimation of the expected measure for each region $r_i \cap r_j \in R_{DT_1 \times DT_2}$ using the measure components of the original regions $r_i \in R_{DT_1}$ and $r_j \in R_{DT_2}$. The expected measure of $r_i \cap r_j$ according to D_1 is:

$$r_i \cap r_j.m_{D_1} = r_i.m_{D_1} \frac{V(r_i \cap r_j)}{V(r_i)} \tag{5.9}$$

where the term $\frac{V(r_i \cap r_j)}{V(r_i)}$ stands for the relative volume of the intersection region $r_i \cap r_j$ with respect to the volume of the region r_i. Since the regions established by a decision tree are axis parallel hyper-rectangles it holds that:

$$V(r) = \prod_{a_i} \frac{|t(a_i)|}{|dom(a_i)|}$$

The term $\frac{|t(a_i)|}{|dom(a_i)|}$ represents the relative importance of attribute a_i in region r. If we assume a uniform distribution $U(A)$ of the instances over the attribute space then $V(r) = P(r)$. In Equation 5.9, though, we adopt a middle assumption, namely that the D_1 instances are uniformly distributed within the region r_i of R_{DT_1}, instead of being uniformly distributed within the whole attribute space.

Following the same rationale as in Equation 5.9, the expected measure of $r_i \cap r_j$ according to the dataset D_2 is:

$$r_i \cap r_j.m_{D_2} = r_j.m_{D_2} \frac{V(r_i \cap r_j)}{V(r_j)} \tag{5.10}$$

Note that this estimation also assumes that the D_2 instances are uniformly distributed within the region r_j of DT_2.

Finally, if we assume that the two datasets come from the same distribution $P(A)$, we can get the expected measure of $r_i \cap r_j$ according to the union dataset, $D_1 \cup D_2$:

$$r_i \cap r_j.m_{D_1 \cup D_2} \quad = \quad r_i \cap r_j.m_{D_1} + r_i \cap r_j.m_{D_2}$$

So far, we have shown how we can estimate the probabilities of the overlay regions depending on whether we have access to the original raw datasets or not. As with the single decision tree partitioning case (c.f. Section 5.3.1), we can use these estimations to approximate the distributions $P(A), P(A, C), P(C|A)$. Depending on which dataset, $D \in \{D_1, D_2, D_1 \cup D_2\}$, we use to calculate the measures of the overlay regions, we get the corresponding estimations of $\mathbf{P_D}(\mathbf{A}), \mathbf{P_D}(\mathbf{A}, \mathbf{C})$ and $\mathbf{P_D}(\mathbf{C}|\mathbf{A})$ under the $R_{DT_1 \times DT_2}$ partition. To distinguish between the case where the measure components are computed by accessing the original raw datasets (Equations 5.6, 5.7) or under the uniform region distribution assumption (Equations 5.9, 5.10), we use the superscripts ^{Q} and ^{U} respectively.

5.3.3 Dissimilarity measures for decision trees and classification datasets

In the previous section we described methods for the estimation of $\mathbf{P_D}(\mathbf{A})$, $\mathbf{P_D}(\mathbf{A}, \mathbf{C})$ and $\mathbf{P_D}(\mathbf{C}|\mathbf{A})$ under the $R_{DT_1 \times DT_2}$ partition and for the different datasets $D \in \{D_1, D_2, D_1 \cup D_2\}$. These estimations can be used to compute similarities between either decision trees or classification datasets.

Before we proceed with the definition of the actual similarity measures, we first provide a similarity function between histograms,

since all our estimations come in the form of histograms. Let P, Q be the probability density estimations for a random variable X from two different populations. Let P, Q come in the form of histograms and both histograms are defined over the same bins.

The *affinity coefficient* between P and Q is given by:

$$s(P,Q) = \sum_i \sqrt{P_i Q_i}$$

The resulting score is in the $[0-1]$ range.

Based on the affinity coefficient and the different estimations of the overlay partition statistics, we can now define a number of similarity measures between decision trees and datasets:

Case a: We can measure the *similarity of two datasets* D_1, D_2 with respect to their attribute space probability distributions $P_{D_1}(A)$, $P_{D_2}(A)$ by directly computing their affinity coefficient:

$$s(\mathbf{P_{D_1}}(\mathbf{A}), \mathbf{P_{D_2}}(\mathbf{A})) \tag{5.11}$$

This similarity measure can be used to determine if the two datasets were generated from the same distribution $P(A)$. The estimations $\mathbf{P_{D_i}}(\mathbf{A}), i = \{1,2\}$ can be either $\mathbf{P^Q_{D_i}}(\mathbf{A})$ or $\mathbf{P^U_{D_i}}(\mathbf{A})$ depending on whether raw data access is allowed or not.

Case b: We can measure the *similarity of two decision trees* DT_1, DT_2 with respect to their predictions. This is a measure of their *semantic similarity*, i.e., how similar are the concepts they describe, and corresponds to the percentage of times that both DTs produce the same predictions on instances drawn from a given attribute space probability distribution.

We first define the vector:

$$\mathbf{I}(\mathbf{C}|\mathbf{A}) = [I(r_i.cl, r_j.cl) | r_i \cap r_j \in R_{DT_1 \times DT_2}]$$

which indicates whether the two DTs agree or disagree in their predictions over the regions of the overlay partition $R_{DT_1 \times DT_2}$. $I(r_i.cl, r_j.cl)$ returns 1 if the predictions of the two DTs regarding the region $r_i \cap r_j$ are the same, i.e., if $r_i.cl = r_j.cl$, otherwise it returns 0. The inner product [2]:

$$S(DT_1, DT_2) = \mathbf{I}(\mathbf{C}|\mathbf{A})' \mathbf{P}(\mathbf{A}) \tag{5.12}$$

computes the similarity in the predictions of DT_1, DT_2 under the $P(A)$ distribution. The similarity score equals to the sum

[2] We denote by X' the inverse of matrix X.

of the probabilities of the $r_i \cap r_j$ regions for which the trees agree in their predictions.

One issue that arises here is which estimation of $P(A)$ we should employ. Possible choices include:

- *the uniform distribution,* $U(A)$. In this case the agreement will be examined over all possible input worlds. Under this assumption, the probability of a region $r_i \cup r_j$ is given by its hyper-volume. Thus, the similarity between two DTs equals to the total volume of the regions in which the two decision trees agree in their predictions. In this case, Equation 5.12 gives the semantic similarity between the two decision trees as it was defined by Turney [95]. Note however that, for this estimation, we *do not require* the generation of an artificial test set drawn from $U(A)$, as required in [95].
- *a dataset dependent distribution* $\mathbf{P_D}(\mathbf{A})$, where D can be one of the D_1, D_2 and $D_1 \cup D2$ datasets. In this case, instances are assumed to follow the distribution of the dataset $D \in \{D_1, D_2, D_1 \cup D_2\}$. The union, $D_1 \cup D2$, is the most appropriate choice if the trees are generated from datasets following the same distribution and we are interested in evaluating their similarity under that distribution.
- finally, $P(A)$ might be a distribution that is *different* from the distributions that rule the training sets.

Case c: We can also measure the similarity of two datasets with respect to the attribute conditional probability distribution of the class attribute $P(C|A)$ that the decision trees, which were induced from these datasets, impose over the attribute space. We first define the vector:

$$\mathbf{S}(\mathbf{C}|\mathbf{A}) = [s(\mathbf{P_{D_1}}(\mathbf{C}|\mathbf{A})[r_i,], \mathbf{P_{D_2}}(\mathbf{C}|\mathbf{A})[r_j,])| \; r_i \cap r_j \in R_{DT_1 \times DT_2}]$$

$\mathbf{S}(\mathbf{C}|\mathbf{A})$ has the same structure as $\mathbf{I}(\mathbf{C}|\mathbf{A})$, but the 0/1 similarity function $I(.,.)$ has been replaced by $s(.,.)$, which computes the similarity of the attribute conditional class distributions of the region $r_i \cap r_j$ in the datasets D_1 and D_2. The inner product:

$$S(D_1, D_2) = \mathbf{S}(\mathbf{C}|\mathbf{A})' \mathbf{P}(\mathbf{A}) \tag{5.13}$$

provides a measure of the similarity of the two datasets with respect to their attribute conditional class distributions under an attribute space that follows the $P(A)$ distribution.

Note here that this measure is similar to the measure employed in [82] in order to rank the changing regions between two datasets. In fact, the approach followed in [82] is equivalent to introducing a distance measure of the form:

$$D(D_1, D_2) = \mathbf{D}(\mathbf{C}|\mathbf{A})'\mathbf{P}(\mathbf{A})$$

where $D(C|A)$ has the same structure as $S(C|A)$ but the similarity function is replaced by the Euclidean distance and $P(A)$ is approximated by:

$$\mathbf{P}(\mathbf{A}) = \frac{1}{2}(\mathbf{P_{D_1}}(\mathbf{A}) + \mathbf{P_{D_2}}(\mathbf{A}))$$

However, the authors in [82] not go so far as to define the $D(D_1, D_2)$. They rather define the product of $D(C|A)$ and $P(A)$, i.e., the vector consisting of the pairwise products of the coordinates of the two vectors, and use that in order to rank regions according to their level of change from one dataset to the other.

Case d: Finally, we can measure the similarity of the joint attribute-class probability distribution of the two datasets $P_{D_1}(A, C)$, $P_{D_2}(A, C)$ by simply applying the affinity coefficient:

$$s(\mathbf{P_{D_1}}(\mathbf{A}, \mathbf{C}), \mathbf{P_{D_2}}(\mathbf{A}, \mathbf{C})) \tag{5.14}$$

$\mathbf{P_{D_i}}(\mathbf{A}, \mathbf{C})$ is the estimation of $P_{D_i}(A, C)$ under the overlay partition. Note here that if the two datasets came from the same $P(A)$ distribution then it can be easily shown that this measure is equivalent to $S(D_1, D_2)$ given in Equation 5.13. In fact this is the approach that was followed by FOCUS [32] for measuring dataset deviation. The difference lies in the fact that instead of the affinity coefficient, FOCUS employs a difference function f (e.g. absolute or relative difference) to compute the measure similarity within each region and an aggregation function g (e.g. sum or max) to aggregate the scores of the overlay regions into an overall score.

In this section we presented a general framework for similarity estimation between either decision trees or classification datasets. Under this framework, we can estimate the similarities of classification datasets with respect to a number of probability distributions:

i) the attribute space distribution $P(A)$ (Equation 5.11), ii) the joint attribute-class distributions $P(A, C)$ (Equation 5.14) and iii) the class attribute conditional distribution $P(C|A)$ (Equation 5.13). We can also use this framework in order to estimate the semantic similarity between decision trees (Equation 5.12) under different assumptions for the attribute space probability distribution; It is this direction that we are going to explore and evaluate in more detail in the next experimental section.

5.4 Experimental evaluation

The semantic similarity of any two classification models M_1, M_2 is defined as the fraction of times that the two models produce the same predictions over instances generated from a given attribute space probability distribution $P(A)$. Turney [95] defined a semantic similarity measure for classification models, called *agreement*, as the probability that they will produce the same predictions over all possible instances drawn from the uniform distribution on the attribute space, $U(A)$. Turney estimates the agreement between two classification models empirically, by applying both of them on a test set D_H of instances drawn from the $U(A)$ distribution, and computing the percentage of times that they produce the same predictions. The argument for employing $U(A)$, instead of the distribution $P(A)$ that generated the data, was that the agreement of two concepts should be examined over all possible input worlds.

We believe that in a real world application what is more important is not the similarity of the DTs in all possible worlds, but rather their similarity in the world the data come from. So, unlike [95], in order to estimate the semantic similarity, we draw the D_H dataset from $P(A)$, which is the distribution that rules the attribute space. We denote by $S_H(DT_1, DT_2)$ the semantic similarity between DT_1 and DT_2; this similarity is empirically estimated on the D_H dataset by applying the two decision trees on D_H and computing the number of times that they produce the same predictions. $S_H(DT_1, DT_2)$ provides the *ground truth* to which we will compare the proposed DT semantic similarity measures.

5.4.1 Experimental settings

We need a systematic way to generate decision trees that exhibit varying degrees of semantic similarity. To this end, we randomly divide a given dataset D in two parts, a *training set* D_T used during

Dataset	# of instances	# of attributes	# of classes
mfeat-factors	2,000	21	10
mfeat-fourier	2,000	76	10
mfeat-karhunen	2,000	64	10
mfeat-zernike	2,000	47	10
segment-challenge	2310	19	7
waveform-5000	5,000	40	3

Table 5.2: Description of datasets

the model construction phase, and a *test set* D_H used as the hold out set for the computation of S_H ($|D_H| = \frac{1}{3}|D|$). Then, we create random sub-samples of the D_T of size p ($p = 5\% \ldots 95\%$) with a step of 5%. On each sub-sample DT_p, a decision tree is trained and compared to the decision tree that was created on the complete training set, DT_{100}. Then, we compute the semantic similarity between the complete DT and the sampled one, i.e., $S_H(DT_p, DT_{100})$, on the hold out set D_H.

We experimented with six different datasets, a short description of which is given in Table 5.2. The different *mfeat* datasets are versions of the same pattern recognition problem in which the goal is to classify handwritten numerals. The versions correspond to different features used to describe the numerals: in *mfeat-factors*, attributes are profile correlations, in *mfeat-zernike* zernike moments, in *mfeat-karhunen* Karnhunen-Love coefficients and in *mfeat-fourier* fourier coefficients of the character shapes [46]. *Waveform-5000* is an artificial dataset where classes correspond to different types of waves [16]. In the *segment-challenge* dataset [14], features are high level descriptors of regions of images and the goal is to classify each region to the correct class, e.g., sky, grass.

First of all, we should verify that the procedure we employed for the generation of the different decision trees DT_p indeed results in trees that exhibit varying levels of semantic similarity with respect to DT_{100}. We expect $S_H(DT_p, DT_{100})$ to increase as p increases and approaches 100%, since the training set D_p used in the construction of DT_p becomes more and more similar to the training set D_{100} used in the construction of DT_{100}. This is indeed the case as one can see in Figure 5.5, where we plot S_H as a function of the sampling size p; there is a smooth increase in the values of S_H as p increases towards 100%. The goal of the experimental evaluation that we present in this section is to examine how the different semantic similarity measures that we propose correlate with S_H.

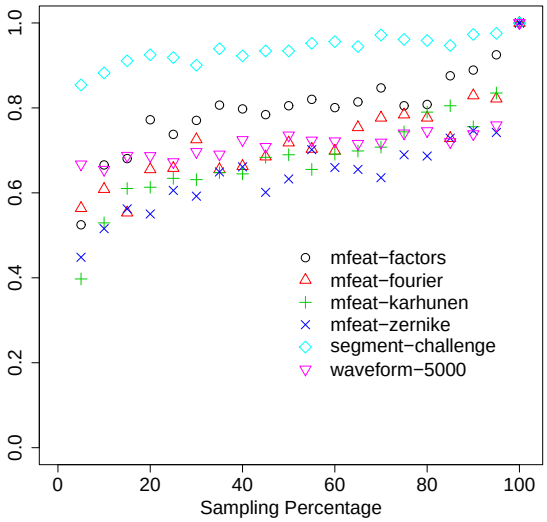


Figure 5.5: Evolution of $S_H(DT_p, DT_100)$ with sampling size p

5.4.2 Qualitative evaluation of the proposed semantic similarity measure

The decision tree semantic similarity measure $S(DT_1, DT_2)$ that we propose (Equation 5.12) depends on the estimation of the $P(A)$ distribution that rules the attribute. In fact, the computation of similarity makes sense for a given world, in which a specific distribution $P(A)$ holds for the attribute space. Then, $S(DT_1, DT_2)$ is simply the sum of the probability densities, under the chosen $P(A)$, of the $r_i \cap r_j$ regions in which the two decision trees agree. As already mentioned, under the uniform distribution assumption this sum equals to the sum of the hypervolumes of these regions. Moreover, under that assumption $S(DT_1, DT_2)$ provides the semantic similarity of Turney [95] *without* having to apply the learned models on the hold out set. We will not further examine the uniform assumption as a possible estimation for $P(A)$. Instead, we will experiment with three different instantiations of $S(DT_1, DT2)$ that differ with respect to the estimation of $P(A)$ they employ. In particular, we will investigate the following estimations for $P(A)$:

- $P^U_{D_1 \cup D_2}$: this is the estimation of $P(A)$ that we get when the measure components are computed under the uniform region distribution assumption, as in Equations 5.9, 5.10.
- $P^Q_{D_1 \cup D_2}$: this is the estimation of $P(A)$ that we get when the

measure components are computed from the direct application of the overlay partition $R_{DT_p \times DT_{100}}$ on the D_p and D_{100} datasets.

- P_H^Q: this is the estimation of $P(A)$ that we get from the direct application of the overlay partition $R_{DT_p \times DT_{100}}$ on the hold out set D_H.

Each of these estimations, P_X^Y, of P(A) results in a different instantiation of $S(DT_1, DT_2)$ which we denote by $S_{P_X^Y}(DT_1, DT_2)$. We should note that the order in which the different P_X^Y are listed reflects an increasing amount of knowledge about the $P(A)$ distribution that rules the computation of the semantic similarity S_H which we use in order to evaluate the proposed similarity measures. $P_{D_1 \cup D_2}^U$ assumes the less knowledge about $P(A)$; to estimate the measure components of the overlay tree, it only relies on the analysis of the structures of the respective decision trees, under the assumption of a uniform within region distribution. $P_{D_1 \cup D_2}^Q$ requires *querying* D_1 and D_2 in order to estimate the measure components of the overlay tree; as a result, its estimation of $P(A)$ is more precise than the one provided by $P_{D_1 \cup D_2}^U$. Finally, P_H^Q has complete knowledge of $P(A)$, as this knowledge underlies in the D_H dataset, since we derive it by *querying* D_H. As a result, $S_{P_H^Q}(DT_1, DT_2)$ will correlate perfectly with S_H. In that sense, $S_{P_H^Q}$ represents the ideal behavior that we get when we have knowledge of the true $P(A)$.

For each $S_{P_X^Y}(DT_p, DT_{100})$ we show how its value varies with respect to the sample size p, in the first column of Figures 5.6, 5.7. All the measures exhibit a similar pattern; similarity increase as p increases. More particular, $S_{P_{D_1 \cup D_2}^Q}$ and $S_{P_H^Q}$ have a very regular behavior, with an almost steady increase of values and small fluctuations. In case of the $S_{P_{D_1 \cup D_2}^U}$ similarity, the trend is also increasing but here the fluctuations can be considerably larger, as it happens in the *mfeat-zernike*, *mfeat-factors*, *segment-challenge*, *mfeat-fourier* datasets. $S_{P_{D_1 \cup D_2}^Q}$ is constantly overestimating decision tree similarity compared to $S_{P_H^Q}$, while $S_{P_{D_1 \cup D_2}^U}$ considerably underestimates it; recall here that $S_{P_H^Q}$ reflects the ideal behavior.

In the second column of Figures 5.6, 5.7, we see how the three different versions of $S_{P_X^Y}(DT_p, DT_{100})$ correlate with the actual evaluation measure $S_H(DT_p, DT_{100})$. As it was expected, $S_{P_H^Q}$ correlates perfectly since its estimation of $P(A)$ is taken from the D_H dataset on which $S_H(DT_p, DT_{100})$ is computed. Consequently, $S_{P_{D_1 \cup D_2}^Q}$ is constantly overestimating $S_H(DT_p, DT_{100})$, while $S_{P_{D_1 \cup D_2}^U}$ is consid-

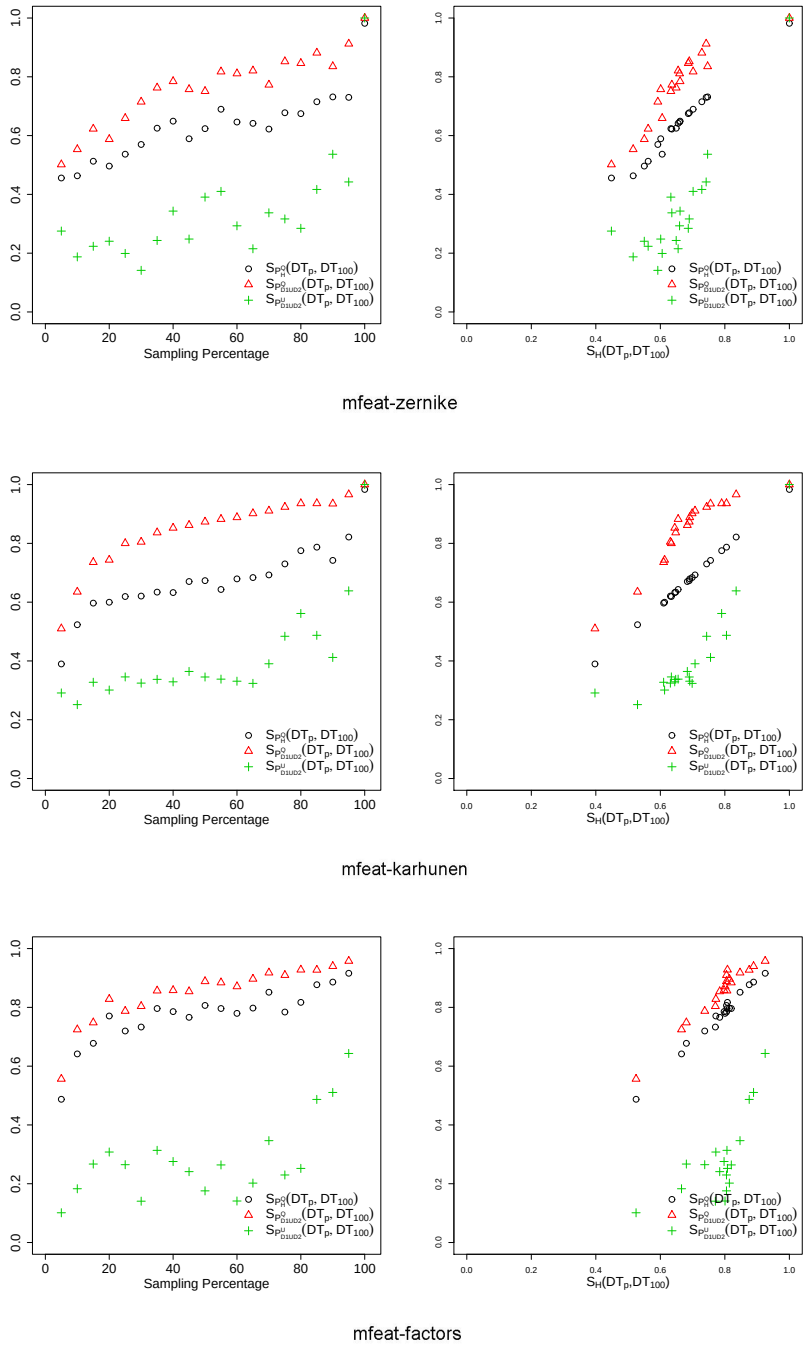
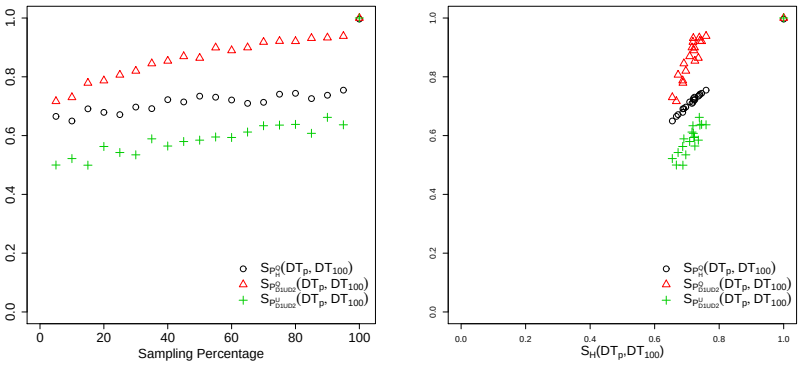
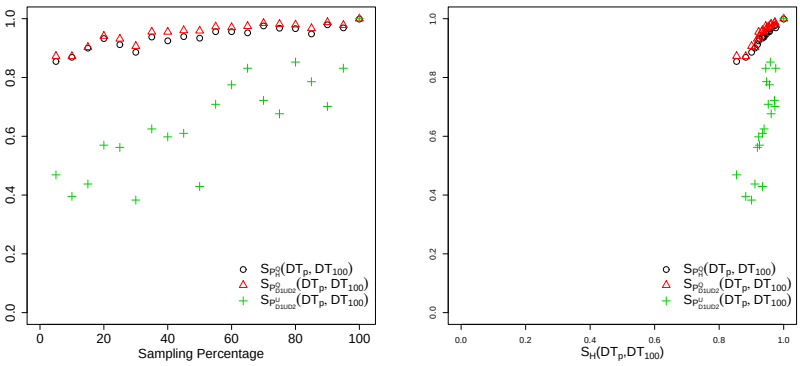


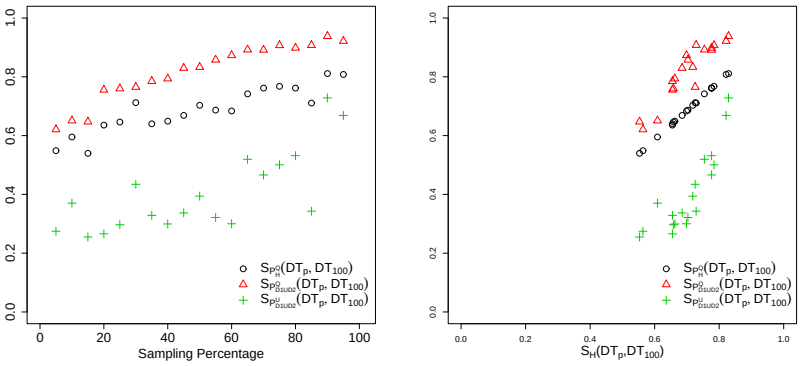
Figure 5.6: Evolution of the decision trees semantic similarity measures with the sampling rate (first column) and with S_H (second column)



waveform-5000



segment-challenge



mfeat-fourier

Figure 5.7: Evolution of the decision trees semantic similarity measures with the sampling rate (first column) and with S_H (second column).

erably underestimating it. The performance of $S_{P^Q_{D_1 \cup D_2}}$ is quite close to the ideal performance of $S_{P^Q_H}$ with the most notable cases being *segment-challenge* and *mfeat-factors*, while the highest discrepancy appears in the case of *mfeat-karhunen*. Note here that datasets D_p, D_T and D_H are all drawn from the same $P(A)$ distribution. The discrepancy between the behavior of $S_{P^Q_{D_1 \cup D_2}}$ and $S_{P^Q_H}$ can be explained by the inaccuracy in the sampling procedure. As the number of instances increases, the behaviors of $S_{P^Q_{D_1 \cup D_2}}$ and $S_{P^Q_H}$ will converge since the estimations of $P(A)$ that the two methods employ will also converge. Alternatively, if we use repeated sampling over the D_H, D_T and D_p datasets and subsequently average over the different samples, the two measures would also converge. On the other hand, the behavior of $S_{P^U_{D_1 \cup D_2}}$ will be similar to that of $S_{P^Q_H}$ only to the level that the assumption of a within region uniform distribution is a valid assumption for the $P(A)$ that rules D_H; nevertheless, as it is apparent for the datasets we have considered here, this is far from being a valid assumption.

5.4.3 Quantative evaluation of the proposed semantic similarity measure

In order to quantify the behavior of each of the $S_{P^Y_X}(DT_p, DT_{100})$ we computed their Pearson correlation coefficient with $S_H(DT_p, DT_{100})$. The results are depicted in Table 5.3, where it seems that $S_{P^Q_{D_1 \cup D_2}}$ exhibits a very strong correlation with $S_H(DT_p, DT_{100})$. For most of the datasets, the correlation is higher than 0.9, with the notable exception of *waveform-5000* for which a low correlation coefficient is recorded. $S_{P^U_{D_1 \cup D_2}}$ has also a strong correlation with $S_H(DT_p, DT_{100})$ although not as strong as $S_{P^Q_{D_1 \cup D_2}}$, again with the remarkable exception of *waveform-5000* for which it exhibits its highest correlation value.

The Pearson correlation coefficient is an estimate of the linear correlation of two values, nevertheless it does not indicate how good predictor one variable is for the other. This is especially true in our case, since the pattern of linear correlation of any given $S_{P^Y_X}(DT_p, DT_{100})$ with $S_H(DT_p, DT_{100})$ changes from dataset to dataset as it is obvious from Figures 5.6, 5.7. In order to estimate the predictive value of the various $S_{P^Y_X}(DT_p, DT_{100})$ with respect to $S_H(DT_p, DT_{100})$, we compute their Mean Absolute Deviation (MAD). The MAD of two variables a and b for which we have N paired ob-

dataset	$S_{P^U_{D_1 \cup D_2}}$	$S_{P^Q_{D_1 \cup D_2}}$	$S_{P^Q_H}$
mfeat-factors	0.692	0.971	0.993
mfeat-fourier	0.852	0.927	0.999
mfeat-karhunen	0.858	0.910	0.999
mfeat-zernike	0.869	0.911	0.987
segment-challenge	0.831	0.951	0.986
waveform-5000	0.969	0.712	0.998

Table 5.3: Correlation coefficient of $S_{P^Y_X}(DT_p, DT_{100})$ with $S_H(DT_p, DT_{100})$

dataset	$S_{P^U_{D_1 \cup D_2}}$	$S_{P^Q_{D_1 \cup D_2}}$	$S_{P^Q_H}$
mfeat-factors	0.504	0.063	0.014
mfeat-fourier	0.301	0.114	0.015
mfeat-karhunen	0.279	0.158	0.013
mfeat-zernike	0.316	0.108	0.022
segment-challenge	0.289	0.016	0.005
waveform-5000	0.120	0.140	0.003
Average	0.302	0.1	0.012

Table 5.4: Mean absolute deviation of $S_{P^Y_X}(DT_p, DT_{100})$ with $S_H(DT_p, DT_{100})$

servations is given by:

$$MAD(a, b) = \sum_{i}^{N} \frac{|a_i - b_i|}{N},$$

The MAD results are given in Table 5.4. These results indicate the good predictive performance of $S_{P^Q_{D_1 \cup D_2}}$, its average error (MAD) in predicting $S_H(DT_p, DT_{100})$ is 0.1. The performance of $S_{P^U_{D_1 \cup D_2}}$ is considerably worse, its average MAD is roughly 0.3.

The goal of the current section was to compare and evaluate a number of different instantiations for a decision tree semantic similarity measure. The different instantiations are the result of different assumptions or different ways of estimating the attribute space distribution under which the semantic similarity computation will take place. In fact, the semantic similarity computation of two decision trees makes sense if we can assume a specific probability distribution $P(A)$ ruling the attribute space. The overlayed tree provides a partition of the full attribute space, the agreement or disagreement of the two decision trees in a given segment of that partition is more or less important depending on the density of that region under $P(A)$. If, for example, the two decision trees disagree on a given region

this is not going to affect their similarity, even if the volume of the region is large, as far as the probability density of that region under $P(A)$ is zero. Alternatively, if we do not want to assume a specific attribute space distribution and we want to compute similarity under all possible worlds we should make the assumption of a uniform distribution on the attribute space, a case that is also covered by our framework.

In order to evaluate our semantic similarity measures, we used the semantic similarity empirically estimated on a separate hold-out set. The performance of the different instantiations of the semantic similarity measures depends on how different was the estimation of $P(A)$ used in them from the $P(A)$ that rules the hold-out set, on which the semantic similarity was computed. In fact, the choice of the appropriate $P(A)$ should be done based on the knowledge of the application domain. If we know that our learning problem is ruled by a specific $P(A)$, then it is that $P(A)$ that should be "plugged" in the decision tree similarity measure. Alternatively, if no such knowledge exists, we can estimate $P(A)$ from the datasets from which the decision trees were constructed, as it was done in the $S^{Q}_{P_{D_1 \cup D_2}}$ semantic similarity measure.

5.5 Related work

Although decision tree induction is an extensively studied research area, limited work has been carried out on the problem of decision tree comparison and more precisely on the computation of semantic similarity between decision trees; the only notable exception is [95]. Recently, several approaches have been proposed that utilize decision tree models for dataset comparison, e.g., [32, 82].

Turney [95], presented a framework for evaluating the *stability* of a classification algorithm, namely the degree to which it generates repeatable results, when trained on different datasets drawn, though, from the same probability distribution. The intuition is that, since the datasets come from the same distribution, the algorithm should induce approximately the same concepts from both datasets. To quantify stability, Turney introduces a semantic measure of similarity called *agreement*. The agreement of two classifiers is defined as the probability that they will produce the same prediction for some random instance drawn from the probability distribution that generated the datasets of the classifiers. Turney estimates the agreement of two classifiers empirically, by applying both of them on artificial test sets of instances drawn from $U(A)$, the uni-

form distribution over the attribute space. Note that, according to Turney, agreement is measured over instances drawn from $U(A)$ and not from $P(A, C)$, the joint attribute-class distribution; the underlying reason is that agreement should be examined over all possible input worlds. As justified in [95], by using $U(A)$ any statistical relationships between attributes that are implicit in the distribution $P(A, C)$ are eliminated. Recall that the agreement, as examined by Turney [95], requires the evaluation of the decision trees with respect to instances drawn from the distribution $U(A)$.

Recently, several change detection methods have been proposed that utilize decision tree models for *dataset comparison*. The intuition behind these approaches is that decision tree models capture interesting characteristics of the datasets and thus, they can be employed in order to compare the original datasets. All the methods in this category follow the same rationale: they use the decision tree models extracted from the datasets to be compared in order to induce a "finer" model and then, they compare the distributions of the two datasets over this (common) "finer" structure. Below, we describe some representative approaches in this category.

In [32], Ganti et al. propose the FOCUS framework (already described in Section 3.6). Decision trees are among the models considered within this framework. Initially, a finer structure, the so-called Greatest Common Refinement (GCR), is obtained by overlaying the partitionings that the two decision trees define over the attribute space. Next, the GCR is populated with instances from the D_1, D_2 datasets and thus, the number of instances that fall in each region of the GCR is computed, for each dataset; this requires querying the original datasets D_1, D_2. Then, the deviation between the two datasets is computed by aggregating, for each region in the GCR, the difference in the number of tuples that result in this region for each dataset. As it arises from the description, FOCUS requires access to the original datasets in order to estimate their deviation.

In [97], Wang and Pei consider the problem of quantifying changes between two datasets D_1, D_2 with class labels. The idea is to assign a *signature* to each dataset and then to compare the datasets on the basis of their corresponding signatures. The challenge is to find a good signature that would be used as the (common) basis for the comparison. The straightforward solution is to use as signature the partitioning that one of the two decision trees, which are extracted from these datasets (e.g., the partitiong of the decision tree DT_1 that is extracted from D_1), defines over the attribute space. However, as the authors justify, this approach is flawed, since the distribution that generated the D_1 dataset, from which DT_1 is extracted,

might be quite different from the distribution that generated the D_2 dataset. Thus, such a comparison would be subjective with respect to the dataset D_1 used for the extraction of the decision tree signature over which the comparison would take place. For this reason, the authors propose to use as signature an *arbitrary tree structure* that partitions the multidimensional problem space into a number of bins. Indeed, instead of a single partitioning the authors propose to use multiple partitionings, each of which is independently and randomly partitioned. They propose two ways to create random partitionings: random forests and random histograms. The authors suggest using *random histograms* as the common structure (i.e., signature) for the comparison, since they are most diverse comparing to random forests; as a measure of diversity, the authors use the number of the different attribute combinations. Diversity is a desired property since it guaranties larger exploration of the problem space and thus it ensures that the signature can "fit" any dataset. The next step, after the construction of the (common) signature over which the comparison would take place, is the population of this signature with instances from both datasets. Then, the distance between the two datasets is computed by aggregating their distances over each one of the N-random histograms. For the comparison of two random histograms, the Manhattan distance is employed. Note that this method requires access to the original datasets in order to estimate their deviation.

Recently, Pekerskaya et al. [82] propose a method for mining *changing regions* from access-constrained data sets. A region is characterized as changing if it appears under different class labels in the two datasets. The goal is to find such regions and to order them according to their degree of change, without, however, accessing the original raw datasets; this requirement might be imposed due to privacy concerns or due to the un-availability of the original raw datasets (e.g., in data streams, old data are forgotten after some time points). The authors justify that the partitioning that a decision tree defines over the attribute space does not consist a good approximation for the dataset distribution. To this end, they extend the traditional decision tree model by further splitting each leaf node of the decision tree into a set of clusters, through some clustering algorithm. The new model, called *cluster-embedded decision tree*, provides a better approximation of the dataset distribution comparing to the approximation of the (simple) decision tree model. After extracting the cluster-embedded decision tree structure for each dataset, the overlay of the two structures is computed and its statistics are estimated without, however, re-querying the original

raw datasets, as in FOCUS [32]. Rather, the authors approximate the measure components of the overlay regions for each dataset by employing the statistics of the corresponding cluster-embedded decision trees. The assumption that takes place during the estimation is that instances are uniformly distributed within each cluster of the cluster-embedded decision tree structure. Note that this method requires the generation of the cluster-embedded decision tree structures instead of the (simpler) traditional decision tree models.

There is a considerable amount of work on comparing tree structures based on the *edit distance*, e.g., [106]. These approaches are based on counting the number and the cost of edit operations (insert, delete, update) that are required in order to convert one tree into the other. However, they work with symbolic trees where the nodes are labeled with symbols from a given alphabet. In decision trees, the nodes are more complex since they include conditions over the symbols-attributes and furthermore, each decision tree path is assigned an importance factor, which is determined by the number of instances that follow this path.

In this work, we present a general framework for similarity estimation that includes as special cases the estimation of semantic similarity between decision trees, as well as various forms of similarity estimation on classification datasets with respect to different probability distributions defined over the attribute-class space of the datasets. The similarity estimation is based on the partitions induced by the decision trees on the attribute space of the datasets. We use the proposed framework in order to estimate the semantic similarity of decision trees induced from different subsamples of classification datasets; we evaluate its performance with respect to the empirical semantic similarity, which we estimate on the basis of independent hold-out test sets.

5.6 Summary

In this chapter, we presented a general framework for the estimation of similarity within a classification problem setting. More specifically, we employ decision tree models in order to estimate either their similarity or the similarity of the datasets that were used for their induction. The *decision tree similarity* is computed in terms of the agreement of the class predictions they return over the attribute space and, it corresponds to the decision tree semantic similarity [95]. The computation of *dataset similarity* can be carried out on the basis of their attribute space probability distribution $P(A)$,

their attribute-class joint probability distribution $P(A, C)$ or their attribute conditional class probability distribution $P(C|A)$. All the above comprise special cases of our similarity estimation framework.

Through this work, we mainly concentrated on the estimation of the semantic similarity between decision trees, i.e., the degree to which the decision trees agree in their predictions over the attribute space. The critical point in the computation of the decision tree similarity is the selection of an appropriate attribute space probability distribution $P(A)$ under which the similarity estimation would take place. The choice of $P(A)$ reflects our belief about the real world on which the decision trees would be applied. If no prior knowledge exists, the uniform distribution $U(A)$ could be selected for $P(A)$ thus examining the decision tree semantic similarity over all possible input worlds.

We experimented with different ways of estimating the attribute space probability distribution $P(A)$ and we compared the resulting instantiations of the decision tree semantic similarity measure with the actual semantic similarity, as this was established by the application of the decision trees on an independent hold-out test set. Depending on the knowledge we have about the $P(A)$ distribution that rules the independent hold-out test set, the computed decision tree semantic similarity is a more or less good predictor of the actual semantic similarity. More specifically, when $P(A)$ is computed by querying the actual datasets, the corresponding decision tree similarity $S_{P^Q_{D_1 \cup D_2}}$ is a very good predictor of the true semantic similarity. Actually, we expect the value of $S_{P^Q_{D_1 \cup D_2}}$ to converge to the real value of the semantic similarity as the size of the datasets increases, since the estimated $\mathbf{P(A)}$ will converge to the true $P(A)$.

We believe that the greatest contribution of a decision tree semantic similarity measure is the potential that it offers to determine whether the observed differences between two decision trees are simply superficial structural differences or they reflect real semantic differences on the described concepts, and moreover, to quantify these differences. This is a problem that deplores decision trees due to their high sensitivity to changes in the underlying training dataset.

Early versions of this study has been previously published in [70, 71].

5.7 Open issues

The availability of a semantic similarity measure for classification models, here decision trees, allows us to perform a number of stan-

dard data mining tasks, not anymore over raw data, but rather over the classification models extracted from these data. This is a kind of *meta-mining* or *meta-analysis*. Below we present some relevant research directions.

Clustering of decision tree models Using the semantic measure of similarity we could group a set of decision trees into a number of clusters and find the *representative decision tree* for each cluster. Decision trees within a cluster should be semantically similar, that is (to some degree) they should agree in their predictions regarding the classification problem they describe. A typical application of this scenario is *distributed data mining* (recall the bank example in Section 5.1).

As with the traditional clustering of raw data, clustering of decision trees involves the following steps: i) the definition of a dissimilarity measure between decision trees, ii) the choice of a clustering algorithm and iii) the definition of some quality criteria for the evaluation of the resulting clusterings. For the i) step, one could employ the semantic similarity measure proposed in this work.

Monitoring decision tree evolution in dynamic environments Since nowadays data are mainly dynamic (e.g., data streams, sensor data) except for extracting some model from a static dataset, there is also the need to monitor how the models extracted from a dynamic dataset change over time.

In the case of classification models, like decision trees, the goal is to find how the decisions represented by the induced decision tree models change over time: Do old decisions hold in the new data? Or there is some concept drift in the decision making process? Also, such questions might focus on a specific class or decision area of the problem: Are there any regions whose decisions (i.e., class label) remain (about) the same across the evolution axis? Are there any regions that change enduringly from time to time?

Indeed, monitoring involves some distance function to estimate the similarity between successive models. However, further issues should be considered, like efficiency in terms of decision trees storage and comparison, effective presentation to the user etc.

Simplification of decision tree ensembles A *decision tree ensemble* (or *decision tree forest*) is a collection of decision trees referring to the same classification problem. In an ensemble, the decision trees are grown independently; when a new instance is coming, the

predictions of the individual decision trees of the ensemble are combined so as to predict its class label.

Using ensembles, instead of simple decision tree models, is more accurate since in an ensemble there are better possibilities to explore the whole problem space. Thus, the generalization accuracy of the ensemble classifier over future, previously unseen problem instances is higher comparing to the accuracy of a simple decision tree. However, an ensemble model is more complex comparing to the simple decision tree model. Consequently, it is more difficult for the user to apprehend the decisions of an ensemble, since she/he has to consider all the component decision tree models. The idea is to simplify the ensemble by reducing the number of decision trees that the user has to investigate through some clustering procedure.

Another alternative to the simplification of decision tree ensembles, is the construction of the overlayed tree from all the component decision trees of the ensemble. Each region of the overlayed tree will be labeled according to the labels of the corresponding regions of the original trees. The overlayed tree will have the same predictive power as the ensemble, since it will make exactly the same predictions, however its partitions will be much finer than the partitions of the original trees thus having a larger complexity than its constituents. Nevertheless, it is possible to simplify the overlayed decision tree by applying standard pruning techniques.

The apparent advantage of having a single decision tree, or a small set of decision trees, instead of the full ensemble is the much easier interpretation of the learned model.

Effect of mining parameters One interesting application is studying the effect of the different mining parameters on the resulting decision tree models. For example:

- How similar, with respect to their decisions, are two decision trees extracted from the same dataset but under *different decision tree induction algorithms*? If, for example, two algorithms give similar results, then one could use the less expensive (or more user friendly) algorithm for her/his purposes.

- Or, how *a specific algorithm parameter* affects the resulting decision tree models? If the parameter does not affect the decisions of the trees, one could use for this parameter the lowest value (or the larger value, depending on the parameter) so as to gain in performance, e.g., in running time.

- Or, how the *pruning level*, which is applied in order to avoid

over-fitting of the decision tree to the training data, affects the agreement of the resulting decision tree models? If, for example, a pruned (and thus smaller) tree makes the same decisions with a larger tree, one should use the first since it is simpler and thus, it is much easier for the end user to understand it.

In all these cases, some measure of similarity should be employed in order to quantify the similarity between the decisions of the two models.

Error estimation The idea of a representative decision tree for a set of decision trees could also be useful in a classification error estimation scenario. Typically, in error estimation a re-sampling technique is applied resulting in a number of different models, the final result is an estimation of the classification performance of the algorithm and not that of a single tree. The question is which model to choose among the different models that were produced; one solution would be to choose the median model, i.e., the one that abstains the smaller distance from all the other models.

Chapter 6

Estimating Similarity between Clusters (and Clusterings) - Change Detection and Exploitation in Dynamic Environments

In this chapter, we employ comparison between either clusters or clusterings, as a means for monitoring and detecting changes in a dynamic environment.

The chapter is organized as follows: Problem settings and motivation are presented in Section 6.1. The adopted model of evolving data is described in Section 6.2. In Section 6.3, we present the MONIC framework for modeling and tracking of cluster transitions, upon clusters defined independently of the cluster type (i.e., hierarchical, partitioning, density based). MONIC is extended into MONIC+ (Section 6.4), which further exploits the special characteristics of each cluster type. Cluster transitions are maintained in an Evolution Graph (Section 6.5), where nodes represent clusters that are observed at different timepoints, while edges denote cluster transitions. In Section 6.6, cluster evolution is summarized through the FINGERPRINT framework that suppresses less informative cluster transitions subject to preciseness and compactness criteria. Experimental study on MONIC , MONIC+ and FINGERPRINT is presented in Section 6.7.2. Related work is presented in Section 6.8. We conclude our study in Section 6.9, whereas in Section 6.10 we describe further improvements and open issues.

Index terms cluster transitions, cluster monitoring, change detection, cluster summarization, dynamic environments.

6.1 Introduction

Data streams are used in many modern applications and impose new challenges for data management because of their size and high degree of variability. One of these challenges is the efficient detection and monitoring of changes in the underlying population. For example, changes in the patterns known to a network intrusion detection system may indicate that intruders test new attacks and abandon old, already known (and blocked) intrusion patterns. In general, monitoring of change is essential for applications demanding long–term prediction and pro-action.

Cluster models are commonly used as a tool for studying the dynamics of a population. In recent years actually, due to the dynamic nature of data, it has been recognized that clusters upon the data of many real applications are affected by changes in the underlying population of customer transactions, user activities, network accesses or documents. A lot of research has been devoted in *adapting* the clusters to the changed population. Recently, research has expanded to encompass *tracing and understanding* of the changes themselves, as means of gaining insights on the population and supporting strategic decisions. Consider, for example, a business analyst who studies customer profiles; understanding how such profiles change over time would allow for a long-term proactive portfolio design instead of reactive portfolio adaptation.

For the categorization and tracing of cluster changes upon accumulating datasets we propose the MONIC (for monitoring clusterings) framework. MONIC takes as input an accumulating data collection, the records of which are subject to ageing, as is typical in data stream applications. The records are clustered at consecutive data points and their evolution is monitored. To this purpose, we first define a set of *cluster transitions*, like survival, split and absorption, and then we propose *transition indicators* incorporated in a transition detection algorithm for their detection. MONIC differentiates from existing approaches for pattern change detection (e.g., [1, 33, 61]), since it is independent of the clustering algorithm; in MONIC , transitions rely on the cluster members. To exploit the special features of each cluster type, we extend MONIC into MONIC+ that also considers the cluster type (hierarchical, partitioning, density based) and thus, is cluster dependent.

Then, we model changing clusters in an Evolution Graph, consisting of sequences of transitions from the first time a cluster was detected to the time it was dissolved. Such a graph contains a wealth of information regarding the evolution of the underlying population. There are several exploitation capabilities upon this graph like querying or studying the stability of the underlying population by observing the lifetime of clusters and clusterings.

However, since evolution is a permanent characteristic of data streaming, the long-term perusal requires a representation that highlights remarkable changes while suppressing trivial pattern perturbations so as to help the end user to understand population evolution at a glance. To this end, we present the FINGERPRINT framework that condenses this graph into a "fingerprint", a structure in which similar clusters are summarized, subject to preciseness and compactness criteria.

The maintenance and summarization of cluster changes (pattern changes, in general) upon a stream is a new problem. Summarization of data (rather than patterns), however, has been studied extensively. Popular summarization methods include histograms and wavelets; there is much work on the efficient maintenance of these structures and on the adaptation of their contents when data change (e.g., [87]). But, these methods do not show how the data change nor do they maintain the changes themselves. There is also research on storing, modifying and querying patterns in inductive or conventional databases (e.g., [9, 19]). These approaches, though, have not been designed for patterns over streams and, although there is provision for modifying patterns when new data arrive, there are no solution for the maintenance of their changes over time. However, since in a dynamic environment patterns accumulate as the period of monitoring increases, the requirements for the efficient maintenance and exploitation of their evolution history becomes crucial.

To summarize, we consider a dynamic environment of evolving data and we use cluster models as a mean of analyzing these data. Our proposal is threefold:

- First, the MONIC /MONIC+ frameworks detect transitions between clusters of consecutive time points.
- Then, both clusters and their transitions are organized into a graph structure, named Evolution Graph, that contains the whole history of the population evolution.
- Finally, the FINGERPRINT framework provides the summarization of these transitions in some condensed and informative

way.

These three components comprise the necessary infrastructure for studying the evolution of a population in a dynamic environment. The architecture of our approach is illustrated in Figure 6.1. As it is shown in this figure, the end user can access each of these components getting different kind of information over the evolving population.

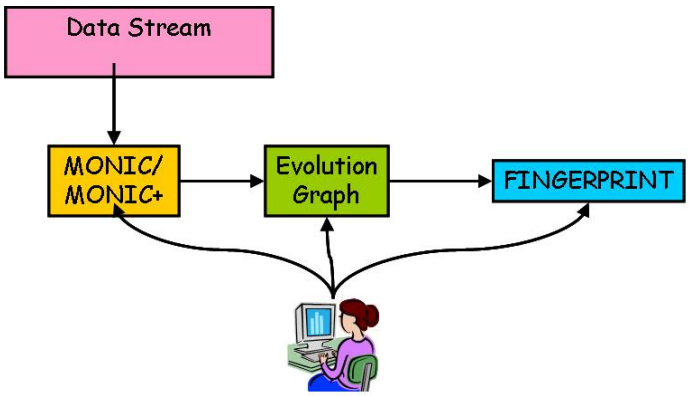


Figure 6.1: The system architecture

6.2 Monitoring dynamic environments

It is assumed that the data stream consists of a set of data records $d_1, \ldots, d_n$ arriving at time stamps $t_1, \ldots, t_n$, where d_i $(i = 1 \ldots n)$ is the substream of data records arrived at the interval $(t_{i-1}, t_i]$. Old data may be subject to a data ageing function that assigns lower weights to all or some of the past data records, as is often the case for topic detection and tracking methods [7].

Definition 5 (Data ageing function) Let $t_1, \ldots, t_n$ be the sequence of timepoints under observation and let $d_i, i = 2, \ldots, n$ be the set of data records accumulated from t_{i-1} until t_i, while d_1 is the initial dataset, so that $d_i \cap d_j = \emptyset$ for $i \neq j$. A *data ageing function* assigns a weight $age(x, t_i) \in [0, 1]$ to data record x at t_i, for each $x \in \cup_{l=1}^{i} d_l$ and for each t_i.

$$age : \cup_{l=1}^{i} d_l \times \{t_1 \ldots, t_n\} \rightarrow [0, 1] \tag{6.1}$$

This function covers sliding windows (the weights of records outside the window are zero) but also more elaborate schemes, e.g., [68]

which considers re-appearances of each record and assigns higher weights to recurring records.

Let $D_1, D_2, \ldots, D_n$ be the actual datasets seen at $t_1, \ldots, t_n$ timepoints; note that, depending on the ageing function, D_i might contain, except for the d_i data records, data records from the past. The weights assigned by the ageing function determine the impact of each record upon clustering $\zeta_i \equiv \zeta_i(\cup_{l=1}^{i} d_l, age, t_i)$. We are interested in studying the population evolution across the time axis. To this end, we rely on the clustering models $\zeta_1, \zeta_2, \ldots, \zeta_n$ extracted from the corresponding $D_1, D_2, \ldots, D_n$ datasets. In Figure 6.2, the considered model of the dynamic environment is depicted.

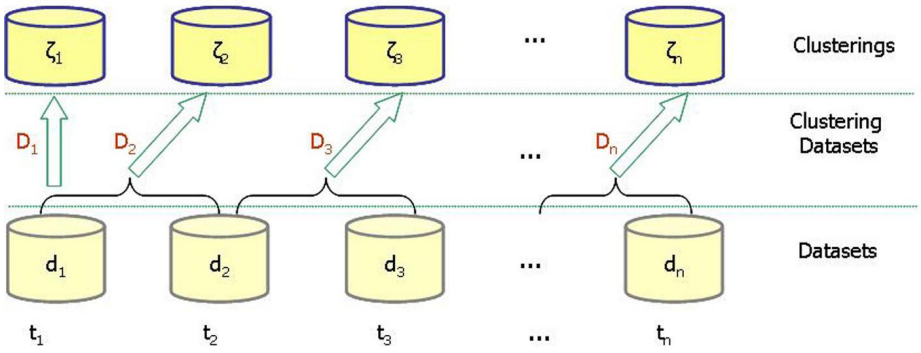


Figure 6.2: Monitoring dynamic environments (window size = 2)

The set of features used for clustering may also change during the period of observation, thus allowing for the inclusion of new features and the removal of obsolete ones. To handle such *dynamic feature spaces*, as well, we assume re–clustering rather than cluster adaptation at each timepoint, so that transitions can be detected even when the underlying feature space changes, i.e., when cluster adaptation is not possible. Moreover, this approach allows both changes in existing clusters and new clusters to be monitored.

A clustering over a dataset can be observed as a partitioning of the dataset into homogeneous groups. We concentrate on *hard clustering*, in which each object belongs to exactly one cluster, as opposed to soft clustering, where an object is associated with each cluster by a probability or possibility value.

Our goal is to trace/monitor a cluster found at some timepoint t_i among the clusters of the next timepoint t_j. Since this depends on the notion of cluster itself, we should first introduce a typification of clusters.

Clustering algorithms use a variety of cluster definitions [39]. We propose the following typification that facilitates the study of clusters as *changing objects*:

Definition 6 (Type A clusters:) *Clusters are discovered upon a* dataset - independent *metric space. A cluster is a geometric object, e.g. a sphere like in K-means. Cluster changes are observed over the static metric space as geometric transformations.*

Definition 7 (Type B1 clusters:) *There is no metric space or it depends on the contents of the dataset at each timepoint. A cluster is defined* extensionally *as a set of data records. Hierarchical algorithms which build dendrograms and express clusters as sets of proximal data points belong to this type. These algorithms use a metric space to derive a clustering on a dataset, but this space is* data-dependent*, in the sense that the addition of a new record might change the border of a cluster, even if this record does not belong to the cluster at all.*

Definition 8 (Type B2 clusters:) *A cluster is defined* intensionally *as a distribution. For a cluster X of type B2, we denote its cardinality as $card(X)$, its mean as $\mu(X)$ and its standard deviation as $\sigma(X)$. The Expectation-Maximization (EM) algorithm belongs to this category.*

Several combinations of the base types are possible, e.g., when both the dataset and its statistics are used (types B1+B2). Note also, that each cluster can be described as a set of objects (i.e., type B1); this is a generic definition that holds for every cluster type.

Table 6.1 summarizes the symbols used throughout the chapter.

6.3 The MONIC framework for detecting cluster transitions

In Figure 6.3, we visualize the challenge of understanding cluster changes in a dynamic environment: We depict clusters at two timepoints, denoting with a + (red) color the records that have been added at each timepoint. Old records are forgotten, using a time window of size 2. The clusters at each timepoint can be easily seen. It is also apparent that changes have occurred. However, it is much more challenging to find the same cluster again and to *categorize and trace* the changes upon it: "Did some clusters disappear? Or were they rather absorbed by others? When is a cluster the same

Symbol	*Description*
$age()$	the age function (Def. 5)
d_i	the dataset seen during $(t_{i-1}, t_i]$
D_i	the dataset on t_i considering function $age()$
ζ_i	the clustering at t_i based on D_i
$\|A\|$	cardinality of set A
$overlap(X, Y)$	the overlap between clusters X and Y (Def. 9)
$\tau \equiv \tau_{match}$	the match threshold ($0.5 < \tau \leq 1$) (Def. 10)
τ_{split}	the split threshold ($\tau_{split} < \tau$) (Sec. 6.3.2.1)
$\widehat{c}$	the label of a cluster c (Section. 6.5.1.1)
$T \equiv trace(c)$	the trace of an emerging cluster c (Def. 17)
$lifetime(c)$	the lifetime of cluster c (Def. 18)
$lifetime(\zeta)$	the lifetime of clustering ζ (Def. 19)
$survivalRatio(\zeta)$	the survival ratio for clustering ζ (Eq. 6.9)
$absorptionRatio(\zeta)$	the absorption ratio for clustering ζ (Eq. 6.9)
$passforwardRatio(\zeta)$	the passforward ratio for clustering ζ (Eq. 6.9)
$\widehat{X}$	the virtual center of a subtrace X (Def. 23)
$ILoss\_trace(T, S)$	information loss due to the replacement of a trace T by a summary S (Equation 6.10).
$CGain\_trace(T, S)$	compactness gain due to the replacement of a trace T by a summary S (Equation 6.11).

Table 6.1: List of symbols for Chapter 6

and when does it mutate?" To detect such transitions, we propose the MONIC framework for the i) *categorization* and ii) *tracing* of cluster changes upon accumulating datasets.

The MONIC framework takes as input two consecutive clusterings extracted from the (evolving) population and outputs the transitions between their component clusters. The first step in detecting such transitions is to trace a cluster found at some timepoint among the clusters of the next timepoint; to this end, we employ the notions of cluster overlap and cluster match (Section 6.3.1). Then, we present the different transitions that a discovered cluster might encompass; we distinguish between external transitions that refer to changes in the whole clustering (Section 6.3.2.1) and internal transitions that refer to changes in a specific cluster (Section 6.3.2.2). In Section 6.3.2.1, we present an algorithm for the detection of these transitions.

Our aim with MONIC is to provide a framework for cluster transition detection that is independent of the clustering algorithm. To this end, we adopt the definition of clusters as sets of objects (Definition 7), which, as already stated, holds for the different clustering methods. Later on, however, we extend MONIC into MONIC+ (Section 6.4) that also considers the special characteristics of each cluster

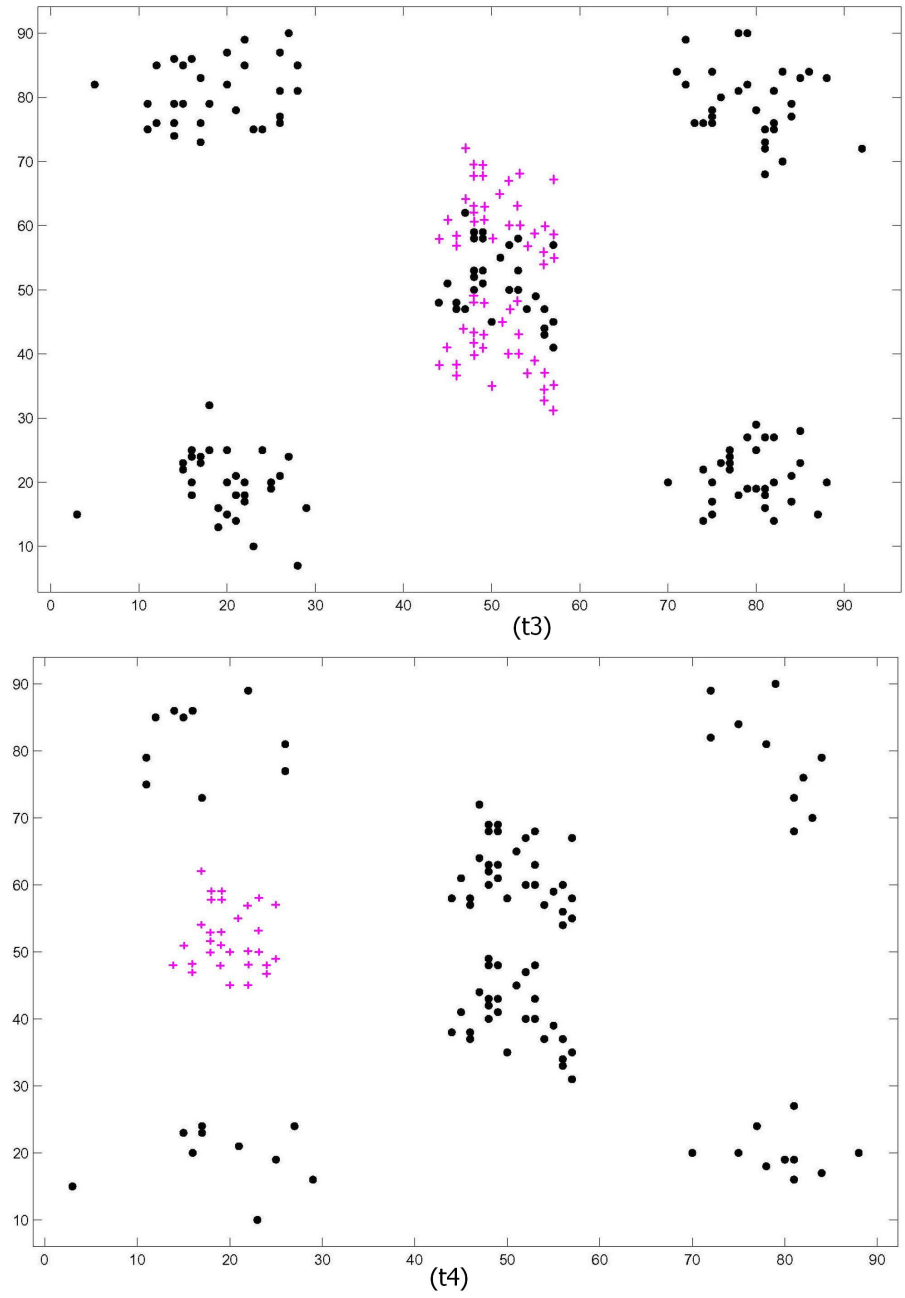


Figure 6.3: A dynamic population at two timepoints t_1 (top), t_2 (bottom)

type and thus, is cluster type dependent.

6.3.1 Cluster matching

Consider a cluster X discovered at a timepoint t_i as part of the corresponding clustering ζ_i. A *cluster transition* is a change effected upon this cluster, when we observe it at a later timepoint t_j ($t_j > t_i$). The first step in identifying such a transition is the detection of the cluster X in the corresponding clustering ζ_j – if it is still existent. Hence, we define the notion of (non-symmetric) *overlap* and of (best) *match* for a cluster, before we proceed with the presentation of cluster transitions.

Definition 9 (Cluster overlap) *Let ζ_i, ζ_j ($i \neq j$) be two clusterings derived at the timepoints t_i, t_j, respectively, and let $X \in \zeta_i, Y \in \zeta_j$ be two clusters. The overlap of X to Y is the normalized sum of the weights of their common records:*

$$overlap(X, Y) = \frac{\sum_{a \in X \cap Y} age(a, t_j) \times weight(a)}{\sum_{x \in X} age(x, t_j) \times weight(x)} \qquad (6.2)$$

That is, the overlap of X to Y depends on the amount of the survived in t_j X members that belong to cluster Y.

We can now define for each cluster found at a timepoint t_i, its best match at a later timepoint t_j.

Definition 10 (Cluster match) *Let X be a cluster in the clustering ζ_i at timepoint t_i and Y be a cluster in the clustering ζ_j at timepoint $t_j > t_i$. Further, let $\tau \equiv \tau_{match} \in (0.5, 1]$ be a threshold value. Y is* a match for X in ζ_j subject to τ, *i.e., $Y = match_\tau(X, \zeta_j)$ if and only if:*

1. *Y has the maximum overlap to X among all clusters in ζ_j, i.e., $overlap(X, Y) = \max_{Y' \in \zeta_j}\{overlap(X, Y')\}$ and*

2. *$overlap(X, Y) \geq \tau$.*

If there is no such $Y \in \zeta_j$, then $match_\tau(X, \zeta_j) = \emptyset$.

By Definition 10, ζ_j can contain at most one match for each cluster in ζ_i, although the same cluster in ζ_j can be the match of more than one clusters in ζ_i. We restrict the threshold τ to the interval $(0.5, 1]$ to stress that a cluster is a match only if it contains more than a half of the pivot cluster members (e.g., more than a half of its members, if the members are weighted equally).

6.3.2 Cluster transitions in MONIC

In MONIC, a *cluster transition* at a given timepoint is a change experienced by a cluster that has been discovered at an earlier timepoint. Such a transition may concern its relationship to the rest of the clustering, i.e., be *external* to it, or it may concern the content and form of the cluster, i.e., be an *internal* transition. We first define these types of transitions and then introduce heuristics for their detection.

6.3.2.1 Detecting external transitions

The external transitions of cluster $X \in \zeta_i$ with respect to ζ_j at $t_j > t_i$ are defined in Table 6.2: A cluster may disappear, be split into multiple clusters, be absorbed by another larger cluster or survive, whereupon internal transitions may occur.

A cluster $X \in \zeta_i$ *survives* in ζ_j if (a) there is a match for it in ζ_j subject to τ and (b) this match does not cover any further cluster of ζ_i. If the match covers at least one further cluster in ζ_i, then X has been *absorbed*. If no match exists, then a *split* may have occurred: The contents of X are in more than one clusters of ζ_j. Then, the overlaps must be no less than τ_{split} (obviously: $\tau_{split} < \tau$), to prevent degenerate cases. Moreover, all those clusters together must form a match for X. If none of these cases occur, then X has *disappeared*. *Emerging* clusters are detected after tracing all external transitions for each cluster in ζ_i: They are the clusters in ζ_j that are not the result of external transitions.

Transition	***Notation***	***Indicator***
the cluster survives	$X \rightarrow Y$	$Y = match_\tau(X, \zeta_j)$ AND $\nexists Z \in \zeta_i \setminus \{X\} : Y = match_\tau(Z, \zeta_j)$
the cluster is split into multiple clusters	$X \stackrel{\subseteq}{\rightarrow} \{Y_1, \ldots, Y_p\}$	$\forall u = 1 \ldots p : overlap(X, Y_u) \geq \tau_{split}$ AND $overlap(X, \cap_{u=1}^{p} Y_u) \geq \tau$ AND $(\nexists Y \in \zeta_j \setminus \{Y_1 \ldots, Y_p\} : overlap(X, Y) \geq \tau_{split})$
the cluster is absorbed	$X \stackrel{\subseteq}{\rightarrow} Y$	$Y = match_\tau(X, \zeta_j)$ AND $\exists Z \in \zeta_i \setminus \{X\} : Y = match_\tau(Z, \zeta_j)$
the cluster disappears	$X \rightarrow \odot$	none of the other cases holds
a new cluster has emerged	$\odot \rightarrow X$	

Table 6.2: External transitions of a cluster

In Figure 6.4 we present our transition detector algorithm, named $detectTransitions()$ that discovers the external transitions of clusters discovered at t_i (clustering $\zeta\_i \equiv \zeta_i$ in the figure) with respect

to the clusters discovered at a consecutive time point t_j ($t_j > t_i$) (clustering $\zeta\_j \equiv \zeta_j$ in the figure).

```
detectTransitions()
Input: ζ_i, ζ_j
Output: cluster transitions between ζ_i to ζ_j
BEGIN
1  overlap = computeOverlaps(ζ_i, ζ_j)                          //Matrix of overlaps
2  FOR X ∈ ζ_i
3       splitCandidates = splitUnion = deadList = splitList = absorptionList
        =absorptionSurvivals = absorptionCandidates = ∅;
4       survivalCandidate = NULL;
5       FOR Y ∈ ζ_j
6            Mcell = overlap(X,Y);
7            IF Mcell ≥ τ_match THEN
8                 IF g(X,Y) > g(X,survivalCandidate) THEN
9                      survivalCandidate = Y;
10                ENDIF
11           ELSEIF Mcell ≥ τ_split THEN
12                splitCandidates += Y;
13                splitUnion = splitUnion ∪Y ;
14           ENDIF
15      ENDFOR
16      IF survivalCandidate == NULL OR splitCandidates == ∅
17           THEN deadList += X;                                  //X → ⊙
18      ELSEIF splitCandidates ≠ ∅ THEN
19           IF overlap(X,splitUnion) ≥ τ_match THEN
20                FOR Y ∈ splitCandidates
21                     splitList += (X,Y);
22                ENDFOR                                  //X ⊆→ splitCandidates
23           ELSE deadList += X;                                  //X → ⊙
24           ENDIF
25      ELSE absorptionSurvivals += (X,survivalCandidate);
26      ENDIF
27 ENDFOR
28 FOR Y ∈ ζ_j
29      absorptionCandidates = makeList(absorptionSurvivals,Y);
30      IF cardinality(absorptionCandidates) > 1 THEN
31           FOR X ∈ absorptionCandidates
32             absorptionList +=(X,Y);                            //X ⊆→ Y
33             absorptionSurvivals -= (X,Y);
34           ENDFOR
35      ELSEIF absorptionCandidates == X THEN
36           survivalList +=(X,Y);                                //X → Y
37           absorptionSurvivals -= (X,Y);
38      ENDIF
39 ENDFOR
END
```

Figure 6.4: Detecting cluster transitions

The algorithm first computes a matrix of overlaps between the clusters of the two clusterings (line 1). Since this is an expensive computation we perform it once in advance and whenever the overlap between two clusters is needed we retrieve the appropriate cell of the overlap matrix. Then, for each cluster $X \in \zeta_i$, the detector performs some initializations (the variables are explained later) and retrieves its overlap to each cluster of ζ_j (line 6). The detector looks first for clusters in ζ_j that match X (lines 7–10). In line 8, the best match for X is selected. So, each cluster in ζ_i has at most one survival candidate. If X has none, clusters overlapping with it for more than $\tau_{split} < \tau_{match}$ are found (lines 11–14). If neither exist, then X is marked as disappeared (lines 16–17).

The case of cluster split detection involves building a list of split candidate clusters (line 12). As specified in Table 6.2, these clusters, when taken together, must form a match for cluster X. The operation of "taking the clusters together" (line 13) refers currently to simple set union of the records, i.e., weights are not taken into account at that point. However, weights are still considered in the overlap test performed at line 19. If this test succeeds, cluster X is marked as split (line 21), otherwise it is marked as disappeared (line 23).

The cases of absorption and survival are initially treated together: ζ_i clusters and their survival candidates are added to a list of absorptions and survivals (line 25). When all ζ_i clusters are processed, this list is completed (line 27). Then, for each ζ_j cluster Y, the detector extracts from this list all ζ_i clusters for which Y is a survival candidate (line 28). If this sublist contains more than one clusters, then these have been absorbed by Y: They are marked as such (lines 31–32) and removed from the original list (line 33). Otherwise, the single member of the sublist is a cluster X that has survived as Y (line 36). Again, the original list is updated (line 37).

An optimization is performed on the split detection test (line 19): Since, due to the hard clustering assumption considered in our problem settings, the clusters in ζ_j cannot have common members, it holds that:

$$\sum_{a \in X \cap (\cup_{u=1}^{p} Y_u)} age(a, t_j) = \sum_{u=1}^{p} \sum_{a \in X \cap Y_u} age(a, t_j)$$

This allows us to perform the split test from the individual intersection values in the overlap matrix and without any further com-

putations, that is:

$$overlap(X, \cup_{u=1}^{p} Y_u) = \sum_{u=1}^{p} overlap(X, Y_u)$$

Algorithm complexity The complexity of Algorithm 6.4 for detecting cluster transitions includes the cost of computing the overlap matrix between the component clusters of the two clusterings (which, as already stated, takes place once at the beginning of the algorithm) and the cost of the actual transition detection process.

Regarding the transition detection process, as it turns out from the discussion in the previous section, it requires $\mathcal{O}(|\zeta_i| * |\zeta_j|)$ complexity. Regarding the construction of the overlap matrix, the cost is $O(|D_i| * |D_j|)$, where $D_i(D_j)$ is the clustering dataset at $t_i(t_j)$. Thus, the total cost of the algorithm is $\mathcal{O}(|\zeta_i| * |\zeta_j| + |D_i| * |D_j|)$. Practically it turns out that the algorithm is quadratic to the cardinality of the datasets involved in the clustering process.

6.3.2.2 Detecting internal transitions

Survived clusters may undergo internal changes, e.g., size shrink or expand. In Table 6.3, we have grouped the internal transitions as changes in size, compactness and location. The transitions inside a group are mutually exclusive, but transitions of different groups can be combined. For example, a cluster $X \in \zeta_i$ survived at a cluster $Y \in \zeta_j$ can become larger and more compact at the same time. It is important to point out here, that there are transitions, like the size transition, that can be detected based directly on the cluster-members, whereas there are other transitions, like compactness and location, that require the computation of statistics over the data members.

The two indicators for the detection of *size transitions* compare the datasets of X and Y, rather than computing their intersection. The weights of the individual cluster members are thereby taken into account. However, while the weights used to compute the cluster overlap are those computed for timepoint t_j, the size transition indicators consider the weights of the members of X at the original timepoint t_i. This is reasonable because the size transition should consider the importance of the individual cluster members at t_i vs. t_j.

The *compactness transitions* cannot be traced by observing the data records directly, so we resort to studying derivative values over

Group	*Transition*		*Notation*	*Indicators*
1.	*Size transition*			
	1a.	the cluster shrinks	$X \searrow Y$	$\sum_{x \in X} f(x, t_i) > \sum_{y \in Y} f(y, t_j) + \varepsilon$
	1b.	the cluster expands	$X \nearrow Y$	$\sum_{y \in Y} f(y, t_j) > \sum_{x \in X} f(x, t_i) + \varepsilon$
2.	*Compactness transition*			
	2a.	the cluster becomes more compact	$X \xrightarrow{\bullet} Y$	$\sigma(Y) < \sigma(X) - \delta$
	2b.	the cluster becomes less compact (more diffuse)	$X \xrightarrow{\star} Y$	$\sigma(Y) > \sigma(X) + \delta$
3.	*Location transition (cluster shift)*		$X \cdots \rightarrow Y$	I1. $\lvert\mu(X) - \mu(Y)\rvert > \tau_1$ I2. $\lvert\gamma(X) - \gamma(Y)\rvert > \tau_2$ (c.f. Eq. 6.5 below)
4.	*no change*		$X \leftrightarrow Y$	

Table 6.3: Internal transitions of a cluster

the data distribution. The indicator σ appearing in Table 6.3 is the standard deviation :

$$\sigma(X) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu(X))^2} \tag{6.3}$$

where $\mu(X)$ is the mean value of X: if the standard deviation has decreased by more than some small value δ, then the cluster has become more compact; if it has increased by more than δ, the cluster has become more diffuse. The threshold δ is intended to prevent insignificant changes to be taken as compactness transitions. Other aggregate values over the distribution can be used instead of the standard deviation, like kurtosis:

$$kurtosis(X) = \frac{\frac{1}{card(X)} \sum_{x \in X} (x - \mu(X))^4}{\left(\frac{1}{card(X)} \sum_{x \in X} (x - \mu(X))^2 \right)^2} - 3 \tag{6.4}$$

while a significance test can be used instead of the threshold δ. In Figure 6.5, we show an example of two distributions with different kurtosis values; the distribution on the right has higher kurtosis comparing to the distribution on the left [83].

In the special case of a static metric space, the transitions in Table 6.3 can be detected by studying the topological properties of the cluster. In the metric space, a cluster can also *shift inside this space*. In the absence of a metric space, it is still possible to detect location transitions as *shifts in the distribution*: indicator I1

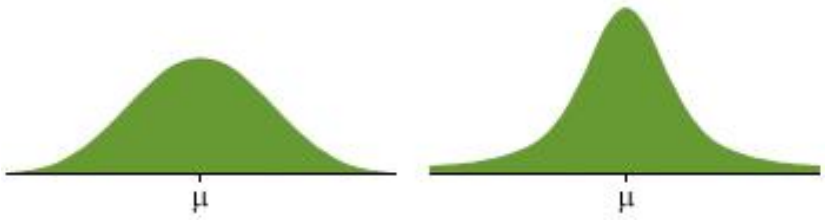


Figure 6.5: Kurtosis example

detects shifts of the mean $\mu(X)$ (within half a standard deviation, c.f. Definition 10), while I2 traces changes in the skewness $\gamma(X)$:

$$\gamma(X) = \frac{\frac{1}{card(X)} \sum_{x \in X} (x - \mu(X))^3}{\left(\frac{1}{card(X)} \sum_{x \in X} (x - \mu(X))^2 \right)^{\frac{3}{2}}} \tag{6.5}$$

The skewness indicator becomes interesting for clusters where the mean has not changed but the distribution exhibits a longer or shorter tail on either side of it. In Figure 6.6, we show an example of two distributions with different skewness values; the distribution on the left has negative skewness, whereas the distribution on the right has positive skewness [84].

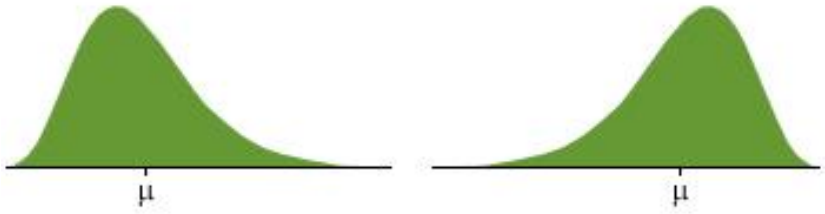


Figure 6.6: Skewness example

6.4 The MONIC+ framework for tracing cluster transitions for different cluster types

The MONIC framework (Section 6.3) detects transitions on clusters represented as sets of objects (Definition 7). In this section, we extend MONIC to MONIC+ that furthermore covers the special characteristics associated with the different cluster types, thus allowing us to capture cluster–type–specific transitions.

We first define the notion of cluster overlap/similarity for different cluster types (Section 6.4.1). Then (Section 6.4.2), we describe the different transition that each cluster type supports, as well as indicators for their detection.

6.4.1 Cluster matching for different cluster types

In Section 6.3.1, we introduced the notions of cluster overlap and cluster match in order to detect whether a cluster found at some timepoint t_i also exists at a next timepoint t_j. However, the definition of cluster overlap in that section refers to clusters described as sets of objects, i.e., Type B1 clusters (Definition 7). Here, we define the notion of cluster overlap for different cluster types. In particular, we define a generic cluster overlap function, which we then instantiate for different cluster types.

Definition 11 (Cluster overlap) *Let ζ_i be the clustering discovered at timepoint t_i and ζ_j the one discovered at $t_j, j \neq i$. We define a function overlap() that computes the similarity or overlap of a cluster $X \in \zeta_i$ towards a cluster $Y \in \zeta_j$ as a value in $[0, 1]$ such that (i) the value 1 indicates maximum overlap, while 0 stands for no overlap and (ii) it holds that $\sum_{Y \in \zeta_j} overlap(X, Y) \leq 1$.*

Cluster overlap is defined asymmetrically. After this generic definition of the overlap function, we specify *overlap*() for each cluster type.

Definition 12 (Overlap for Type A Clusters) *Let ζ_i, ζ_j be two clusterings of Type A clusters, derived at $t_i < t_j$ respectively. For two clusters $X \in \zeta_i$ and $Y \in \zeta_j$, the overlap of X to Y is the normalized intersection of their areas:*

$$overlap(X, Y) = \frac{area(X) \cap area(Y)}{area(X)} \tag{6.6}$$

Definition 13 (Overlap for Type B1 Clusters)

As in Definition 9.

Definition 14 (Overlap for Type B2 Clusters) *Let ζ_i, ζ_j be two clusterings of Type B2 clusters, derived at $t_i < t_j$ respectively. For two clusters $X \in \zeta_i$ and $Y \in \zeta_j$, the overlap of X to Y is defined in*

terms of the proximity of their means:

$$overlap(X,Y) = \begin{cases} 1 - \frac{|\mu(X)-\mu(Y)|}{\sigma(X)} & , \quad |\mu(X) - \mu(Y)| \leq \sigma(X) \\ 0 & , \quad otherwise \end{cases} \tag{6.7}$$

For each cluster found at a timepoint t_i, we can find its best match at a later timepoint t_j using the *Cluster match function* (c.f. Definition 10).

6.4.2 Type-dependent detection of transitions

The detection of external transitions in MONIC+ is as in MONIC (c.f. Section 6.3), but some steps must be implemented differently depending on the cluster type.

The observable transitions for each cluster type are depicted in Table 6.4. All external and internal transitions can be detected for clusters in a metric space (Type A). For clusters defined extensionally (Type B1), compactness and location transitions cannot be observed directly, because concepts like proximity and movement are not defined. However, when one derives the intensional definition of a cluster, both transitions become observable as changes in the cluster density function; we refer to this as Type B1+B2. Conversely, the intensional definition of a cluster (Type B2) does not allow for the detection of splits and absorptions, which in turn can be found by studying the cluster members (Type B1+B2).

	Transitions			
	External	***Internal transitions***		
Cluster type		*Size*	*Compactness*	*Location*
`A.` metric space	Yes	Yes	Yes	Yes
no metric space				
`B1.` extensional	Yes	Yes	No	No
`B2.` intensional	survival	Yes	Yes	Yes
`B1+B2.`	Yes	Yes	Yes	Yes

Table 6.4: Observable transitions for each cluster type

Transition Indicators for Type A Clusters. Let ζ_i, ζ_j be the clusterings at timepoints $t_i < t_j$ and let $X \in \zeta_i$ be the cluster under observation. The transition indicators proposed in Table 6.5 use the type-specific definition of cluster overlap (Definition 12) and the derived definition of cluster match (Definition 10).

External cluster transitions are detected by computing the area overlap between cluster X and each candidate in ζ_j. To detect a split, we customize the split test of the Algorithm 6.4. More specifically, we compute the overlap between the area of X and that of all split candidates $Y_1, Y_2, \ldots, Y_p$. Since these candidates cannot overlap, we use the following equation to perform the split test:

$$area(X) \cap area(\cup_{u=1}^{p} Y_u) = \sum_{u=1}^{p} area(X) \cap area(Y_u) \qquad (6.8)$$

Step	***Transition***	***Indicator***
1	Survival or Absorption	$\exists Y \in \zeta_j : \frac{area(X) \cap area(Y)}{area(X)} \geq \tau$
2	$X \overset{\subseteq}{\rightarrow} Y$	$\exists Z \in \zeta_i \setminus \{X\} : \frac{area(Z) \cap area(Y)}{area(Z)} \geq \tau$
3	$X \rightarrow Y$	$\nexists Z \in \zeta_i \setminus \{X\} : \frac{area(Z) \cap area(Y)}{area(Z)} \geq \tau$
4	$X \overset{\subseteq}{\rightarrow} \{Y_1, \ldots, Y_p\}$	$\exists Y_1, \ldots, Y_p \in \zeta_1 :$ $(\forall Y_u : \frac{area(X) \cap area(Y_u)}{area(X)} \geq \tau_{split}) \wedge \frac{area(X) \cap area(\cup_{u=1}^{p} Y_u)}{area(X)} \geq \tau$
5	$X \rightarrow \odot$	derived from the above
6	$X \nearrow \searrow Y$	B1 indicators & B2 indicators
7	$X \overset{\bullet\star}{\rightarrow} Y$	geometry-dependent & B2 indicators
8	$X \cdots \rightarrow Y$	geometry-dependent & B2 indicators

Table 6.5: Indicators for Type A cluster transitions

The detection of internal transitions translates into tracing the movements of a cluster in a static metric space. In Table 6.6, we propose indicators for spherical clusters, as produced by e.g., K-Means and K-Medoids algorithms. We can further use indicators for Type B1 and B2 clusters.

Transition	***Indicator***
$X \cdots \rightarrow Y$	$\frac{d(center(X), center(Y))}{min\{radius(X), radius(Y)\}} \geq \tau_{location}$
$X \overset{\bullet}{\rightarrow} Y$	$avg_{x \in X}(d(x, center(X)) > avg_{y \in Y}(d(y, center(Y)) + \varepsilon$
$X \overset{\star}{\rightarrow} Y$	$avg_{y \in Y}(d(y, center(Y)) > avg_{x \in X}(d(x, center(X)) + \varepsilon$

Table 6.6: Indicators for spherical clusters

The first heuristic in Table 6.6 detects location transitions by checking whether the distance between the cluster centers exceeds a threshold $\tau_{location}$; we normalize this distance on the size of the smallest radius. The second heuristic states that a cluster has become more compact if the average distance from the cluster center was larger in the old cluster than in the new one – subject to a

small threshold value ε. The third heuristic for clusters becoming less compact is the reverse of the second one.

Transition Indicators for Type B1 Clusters. Transition indicators for type B1 clusters have been already described in Section 6.3, where the MONIC framework has been presented. So, we do not further elaborate here on this cluster type. Just to note here that all external transitions can be detected for clusters of type B1, whereas, regarding the internal transitions, only the size transition can be detected directly from the cluster-members.

Transition Indicators for Type B2 Clusters. We consider again a cluster $X \in \zeta_i$. To detect size transitions, we used the heuristic for Type B1 clusters (c.f. Table 6.3). For the other obervable transitions (c.f. Table 6.4), we use the indicators in Table 6.7. The first one states that a cluster survives if there is a match for it, subject to a $\tau \in (0.5, 1]$ (c.f. Definition 10): The indicator demands that $\mu(X)$ and $\mu(Y)$ are closer than half a standard deviation. Since clusters of the same clustering do not overlap, we expect that no more than one cluster of ζ_j satisfies this condition.

Step	***Transition***	***Indicator***
1	$X \rightarrow Y$	$\exists Y \in \zeta_j : 1 - \frac{\lvert\mu(X)-\mu(Y)\rvert}{\sigma(X)} \geq \tau$
2	$X \rightarrow \odot$	negation of the above
3	$X \nearrow \searrow Y$	B1 indicators in Table 6.3
4	$X \cdots \rightarrow Y$	h1. $\lvert\mu(X) - \mu(Y)\rvert > \tau_{h1}$ h2. $\lvert\gamma(X) - \gamma(Y)\rvert > \tau_{h2}$ (c.f. Equation 6.5)
5	Compactness	
	$X \xrightarrow{\bullet} Y$	$\sigma(Y) < \sigma(X) + \varepsilon$ (c.f. Equation 6.3)
	$X \xrightarrow{\star} Y$	$\sigma(X) < \sigma(Y) + \varepsilon$ (c.f. Equation 6.3)

Table 6.7: Indicators for Type B2 cluster transitions

An absorption transition for $X \in \zeta_i$ implies finding a $Y \in \zeta_j$ that contains $X, Z \in \zeta_i$. Similarly, a split transition corresponds to finding clusters that contain subsets of X. However, this implies treating the clusters as datasets (Type B1). So, we only consider survival and disappearance for B2-clusters.

To detect compactness transitions, we use the difference of the standard deviations of the clusters X, Y. For location transitions, we use two heuristics that reflect different types of cluster shift: h1 detects shifts of the mean (within half a standard deviation, c.f.

Definition 10), while h2 traces changes in the skewness $\gamma()$ (c.f. Equation 6.5). Heuristic h2 becomes interesting for clusters where the mean has not changed but the distribution exhibits a longer or shorter tail.

6.5 The Evolution Graph history

The MONIC framework (and consequently, MONIC+), detects transitions between clusterings discovered at consecutive timepoints. As the period of observation increases, more and more clusters and transitions between these clusters are accumulated, thus building the history of the population evolution.

We model this history of population evolution in a graph structure, the so-called Evolution Graph $EG \equiv G(V, E)$, that spans the whole period of population monitoring (n timepoints). The set of nodes V corresponds to the set of clusters seen during this time period: $V = \{\zeta_1, \ldots, \zeta_n\}$, where each clustering $\zeta_i \in V$ contains the clusters discovered at the corresponding timepoint t_i, i.e., $\zeta_i = \{c_1, c_2, \ldots, c_{|\zeta_i|}\}$. The set of edges E contains the cluster transitions: $\forall e = (c, c') \in E$ there is a timepoint $t_i, 1 \leq i < n$ such that $c \in \zeta_i$ and $c' \in \zeta_{i+1}$. By this specification of the Evolution Graph, the edges connect nodes/clusters found at adjacent timepoints.

An example of an Evolution Graph is depicted in Figure 6.7, where the drawing of the edges indicates the semantics of the transitions. A *dotted (green)* edge denotes a split of the source cluster to multiple target clusters (e.g., $c_{12} \xrightarrow{\subseteq} \{c_{22}, c_{23}\}$). A *dashed (orange)* edge describes an absorption; the source clusters are contained in the target cluster (e.g., $\{c_{23}, c_{24}\} \xrightarrow{\subseteq} c_{33}$). A *solid (blue)* edge indicates a survival; the source cluster has survived into the target cluster (e.g., $c_{11} \rightarrow c_{21}$). Nodes without any incoming edge correspond to new appearing clusters (e.g., cluster c_{11}), whereas nodes without any outgoing edge correspond to disappeared clusters (e.g., cluster c_{43}).

6.5.1 Evolution Graph model

We first describe the semantics of the graph nodes (Section 6.5.1.1) and then the semantics of the graph edges (Section 6.5.1.2) based on which we deploy the Evolution Graph model.

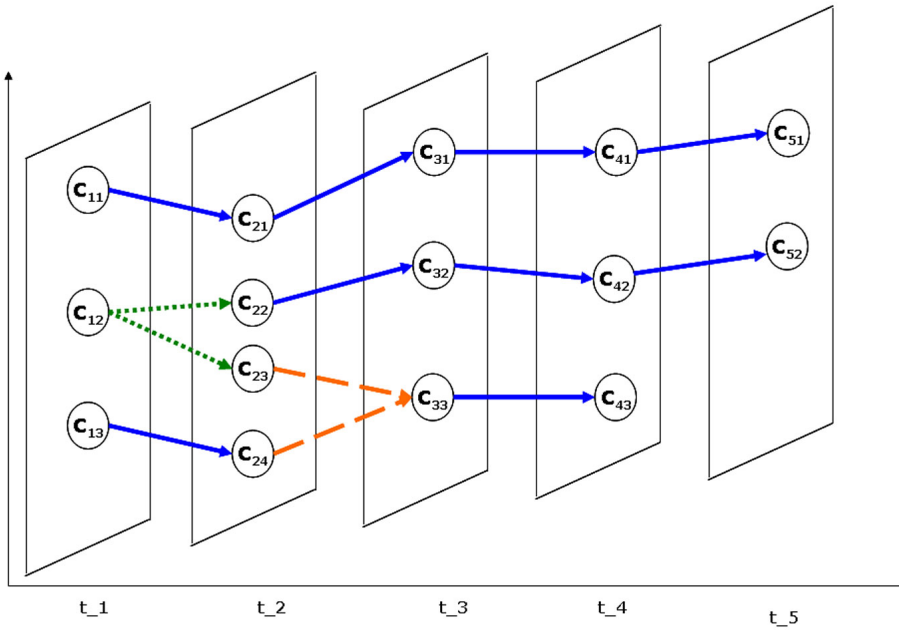


Figure 6.7: Example of an EVOLUTION GRAPH(EG)

6.5.1.1 Semantics of the graph nodes

A node $c \in V$ represents a cluster found at timepoint t_i, i.e., belonging to clustering ζ_i. A node in the evolution graph is adorned with a label $\widehat{c}$ that summarizes its members in some intensional form. Many elaborate summarized representations exists in literature, for example the cluster summaries of [31], the micro–clusters of [2] and the droplets over text data proposed in [3]. We choose two simple representations: the cluster centroids for clusters over arbitrary numerical data and the cluster topics for clusters over text data.

Definition 15 (Centroid as label) Let c be a cluster in an m-dimensional space of numerical properties. Its centroid is the vector of the mean values $\widehat{c} :=< \mu_1 \dots \mu_m >$, where μ_l is the average of the data records values across the l^{th}-dimension, $1 \leq l \leq m$.

For the case of centroid–based labels, we assume that the data values have been normalized, thus all mean values are in the [0..1] range.

For clusters over texts, we rather model a cluster label as the set of the most frequent keywords in it. Stopwords and common words are removed at a preprocessing step, thus the remaining keywords are those that actually characterize the cluster content.

Definition 16 (Keyword-based label) Let c be a cluster of text documents, where each document $d_i \in c$ is a vector in the feature space of the keywords $\{k_1, \ldots, k_m\}$. The cluster label is defined as $\widehat{c} :=< w_{k_1}, \ldots, w_{k_m} >$, where w_{k_l} is (a) the frequency of the l^{th}-keyword within c, if this frequency exceeds a boundary b, or (b) zero, otherwise.

6.5.1.2 Semantics of the graph edges

An edge $e = (c, c') \in E$ denotes that a cluster $c \in \zeta_i$ has been evolved into a cluster $c' \in \zeta_{i+1}$ of the next timepoint. Evolvement means that among the clusters of ζ_{i+1}, the cluster c' is the one most similar to the cluster c. The succession encompasses different cases, e.g., c' might be much larger than c or even might have absorbed c and more clusters from ζ_i, or c might have been split into multiple smaller clusters in ζ_{i+1} or the cluster centroid might have shifted to a different location from t_i to t_{i+1}. We design the semantics of cluster evolvement according to our MONIC framework (c.f. Section 6.3).

More specifically, an edge is drawn from c to c' in our Evolution Graph if the evolvement of c into c' was due to some survival, split or absorption transition. Each edge $e = (c, c') \in E$ is adorned with a label $e.ExtTransition$ that describes the (external) transition type between $c \in \zeta_i$ and $c' \in \zeta_{i+1}$ as one of $\{survival, split, absorption\}$. If a cluster in ζ_i has no outgoing edge, it has *disappeared*. If a cluster in ζ_{i+1} has no ingoing edge, it is a *new cluster*. In the case of internal transitions, e is also adorned with a label $e.IntTransition$ that describes the internal transitions that the cluster has encompassed and its values, e.g., {(size shrink, 20%) (more compact, 10%)}.

6.5.2 Evolution Graph construction

The Evolution Graph is built incrementally as the clustering results arrive at $t_1, \ldots, t_n$. When a new clustering ζ_i arrives at $t_i, i > 1$, MONIC (or MONIC+)is applied on the previous clustering ζ_{i-1} and the current one ζ_i: transitions between clusters of ζ_{i-1}, ζ_i are detected and an edge is added to the Evolution Graph for each detected transition, adorned with relative information.

The Evolution Graph building algorithm is depicted in Figure 6.8.

The algorithm adds the nodes/clusters of ζ_i (denoted as ζ_i in this figure) into the Evolution Graph and assigns labels to them (line 2). If this is the first clustering added to the Evolution Graph, no further actions are required. Otherwise (lines 3–7), the

```
buildEG()
Output: EG = G(V, E)
BEGIN
1. WHILE a new clustering ζ_i arrives at t_i BEGIN
2.    EG.addNodes(ζ_i.nodes);                          //add ζ_i clusters in EG
3.    IF (i > 1) THEN
        j = i − 1;                                                  //for notation
4.      E_ji=detectTransitions(ζ_j, ζ_i);                    //detect transitions
5.      EG.addEdges(E_ji);                                   //add transition edges
6.      EG.updateNodes(ζ_j.nodes);          //remove redundant information from ζ_j
7.    ENDIF;
8. END;
9. return EG;
END
```

Figure 6.8: EVOLUTION GRAPH building algorithm

transitions of clustering ζ_i with respect to the previous clustering ζ_{i-1}, already in EG, are detected and added to the graph. For the detection of transitions the MONIC transition detection algorithm (c.f. Algorithm 6.4) is invoked (line 4). If MONIC finds survivals, splits or absorptions in ζ_i, the corresponding edges are added and adorned with information on the transition type (line 5). If some of the external transitions are survivals, the corresponding edges in EG are also adorned with information regarding the (possible) internal transitions (size, compactness, location).

MONIC uses the cluster contents for transition detection. Hence, we retain this information until the next timepoint only. The nodes inserted at t_{i-1} are updated (line 6): The label of each node is computed and stored, while the data themselves are not retained.

Algorithm Complexity We turn now our attention to the complexity of the above presented algorithm. Adding a new clustering ζ_i into the EVOLUTION GRAPH imposes the following costs: i) the computation of centroid labels for its cluster-members and ii) the computation of transitions between this clustering (ζ_i) and the clustering of the previous timepoint (ζ_j). Let $cost(c)$ be the cost of computing the centroid label of a cluster c. Then, the centroid labels computation for clustering ζ_i costs: $|\zeta_i| * cost(c)$. The cost of detecting transitions between ζ_i, ζ_j is the MONIC cost: $O(|\zeta_i| * |\zeta_j| + |D_i| * |D_j|)$ (recall Section 6.3.2.1). Thus, the computational cost for adding clustering ζ_i into the evolution graph becomes:

$$addCost = O(|\zeta_i| * |\zeta_j| + |D_i| * |D_j| + |\zeta_i| * cost(c))$$

Thus, if we consider an observation period of n timepoints, the

building cost becomes:

$$EGbuildCost = (n-1) * addCost$$

There is also the evolution graph *storage cost*, which refers to the space requirements for storing cluster labels, i.e., centroids, and their transitions. For a timepoint i, we should store the centroids of the clusters of the corresponding clustering ζ_i as well as the cluster transitions between this clustering and the previous timepoint clustering, t_j. If $|centroid|$ is the storage cost of a cluster centroid summary, then the storage cost for the ζ_i cluster centroid summaries is $|\zeta_i| * |centroid|$. If $|edge|$ is the storage cost for an edge, then the storage cost for the transitions between ζ_i and ζ_j is: $|\zeta_i| * |\zeta_j| * |edge|$.

Considering the n-timepoints observation period, the storage cost becomes:

$$EGstorageCost = \sum_{i=1...n} |\zeta_i| \times |centroid| + \sum_{i=1...n-1, j=i+1} |\zeta_i| \times |\zeta_j| \times |edge|$$

In the above formula, the first term corresponds to the cost of storing the cluster centroid labels, whereas the second term corresponds to the cost of storing the cluster transitions.

Note that during Evolution Graphconstruction, we should also store the contests of the most recent clustering, ζ_j. This is because, in order to find transitions of the next incoming clustering ζ_i with respect to clustering ζ_j, the contents of ζ_j should be available. This cost is $O(|D_j|)$, where D_j are the data members of ζ_j.

6.5.3 The traceset of the Evolution Graph

Let $t_i, 1 \leq i \leq n$, be a timepoint of the evolution period. There might be clusters "shaped" for the first time at this timepoint, that is, they do not comprise survivals from the previous timepoint t_{i-1}; we call them *emerged clusters*. By this definition, emerged clusters might be either new appearing clusters at t_i or the results of split or absorption transitions from t_{i-1}.

For an emerged cluster, we form its *cluster trace* as follows:

Definition 17 (Cluster Trace) *Let c be an emerged cluster at t_i. The sequence $\prec c_i \cdot c_{i+1} \cdots c_k \succ$ of clusters discovered at $t_i, t_{i+1}, \ldots, t_k$, $1 < k \leq n$ is the cluster trace of c, trace(c), if $c_i \equiv c$ and for each $c_j, j > 1$ there is an edge $e = (c_{j-1}, c_j)$ such that e.transition = survival. Furthermore, the last cluster of this sequence, namely c_k does not survive at the next time point t_{k+1}.*

That is, the trace of c consists of consecutive cluster survivals of c from the first time it was shaped, i.e., t_i, until the time it was resolved, i.e., t_m. Note that, since c is an emerged cluster at t_i, it might be either the result of some cluster split or cluster absorption transition from t_{i-1} or it might correspond to a new appearing cluster at t_i. Also note that, since c_m does not survive at the next timepoint, it might be split, absorbed or disappeared at the next timepoint.

The notion of cluster trace allows us to detect the path of a cluster within the evolving population from the first time it was "shaped" till the time it was resolved. We denote by $\mathcal{T}_{EG}$ the *traceset of EG*. The traceset $\mathcal{T}_{EG}$ of our sample EG (depicted in Figure 6.7) consists of the following sequences:

- trace $\prec c_{11}c_{21}c_{31}c_{41}c_{51} \succ$, indicating that the cluster c_{11} first appearing in t_1 has survived across all five timepoints,
- trace $\prec c_{22}c_{32}c_{42}c_{52} \succ$ of the cluster c_{22}, one of the clusters to which c_{12} has been split and
- two-node traces $\prec c_{13}c_{24} \succ$ and $\prec c_{33}c_{43} \succ$.

The other clusters c_{12}, c_{23}, c_{24} only existed for a single timepoint and therefore built no traces. Thus:

$$\mathcal{T}_{EG} = \{\prec c_{11}c_{21}c_{31}c_{41}c_{51} \succ, \prec c_{22}c_{32}c_{42}c_{52} \succ, \prec c_{13}c_{24} \succ, \prec c_{33}c_{43} \succ\}$$

To extract the traceset of the EVOLUTION GRAPH one can start scanning the graph from the beginning of the observation period and build cluster traces according to Definition 17.

6.5.4 EVOLUTION GRAPH exploitation

The EVOLUTION GRAPH contains the whole history of cluster change, thus is can be exploited for detecting and understanding changes in the underlying stream population. Below we present two such exploitation capabilities: the first one allows the user to gain insights on the population stability along the period of observation by studying the lifetime of clusters and clusterings (Section 6.5.4.1), whereas the second provides the user with query capabilities over the history of population evolution (Section 6.5.4.2).

6.5.4.1 Lifetime of clusters and clusterings

Cluster transition detection delivers insights on both the evolution of individual clusters and on the overall conformance of a clustering

with the underlying population. Intuitively, if most clusters in a clustering survive from one period to the next, then the population is rather stationary and the clustering is rather short–term. If other cluster transitions (e.g., splits) are frequent though, this signals that the population is volatile and that the clustering does not describe it well.

We define (a) the lifetime of a cluster and (b) the lifetime of a clustering so as to get clues about the stability of the clusters and the clusterings on the evolving data stream. Consequently, those values allow us to study the stability across the time axis of the population under observation.

Definition 18 (Lifetime of a cluster) *Let C be a cluster and t_i be the first timepoint where it emerged (as part of clustering ζ_i). The lifetime of C is the number of timepoints, in which C has survived. We define (i) a* strict lifetime *$lifetimeS$ as the number of consecutive survivals without internal transition, (ii) a* lifetime under internal transitions *$lifetimeI$ for which all survivals are counted and (iii) a* lifetime with absorptions *$lifetimeA$ that further counts absorptions of C.*

By the above definition, the lifetime of a cluster is at least 1, referring to the clustering where it first appeared. We compute cluster lifetime in a backward fashion: We start with ζ_n and set the lifetime of its clusters to 1. At an earlier timepoint t_i, the strict lifetime of cluster X is 1 if X did not survive in t_{i+1}. If there is a $Y \in \zeta_{i+1}$ with $X \leftrightarrow Y$, then $lifetimeS(X) = lifetimeS(Y) + 1$. If there is a $Y \in \zeta_{i+1}$ with $X \rightarrow Y$, then the lifetime of X under internal transitions is $lifetimeI(X) = lifetimeI(Y) + 1$. If there is a $Y \in \zeta_{i+1}$ with either $X \rightarrow Y$ or $X \stackrel{\subseteq}{\rightarrow} Y$, then the lifetime of X with absorptions is $lifetimeA(X) = lifetimeA(Y) + 1$.

Note also that $lifetimeI(c) = |trace(c)|$, that is the lifetime under internal transitions for a cluster c equals to the length of its trace (c.f. also Definition 17).

The counterpart of cluster lifetime for clusterings is an aggregation over the lifetimes of all clusters in the same clustering.

Definition 19 (Clustering Lifetime) *Let ζ be a clustering. Its lifetime $L(\zeta)$ is the median of the lifetime with absorption values among the clusters in it:*

$$L(\zeta) = median_{C \in \zeta}\{lifetimeA(C)\}$$

In this definition, we use the weakest definition of cluster lifetime, allowing for internal transitions and absorptions. To prevent the

dominance of a few short-lived or a few long-lived clusters, we use the median lifetime instead of the average.

The clustering lifetime is a *long-term* property. However, in most cases, clusterings are short-lived, even if some of their clusters survive over several timepoints. We define therefore a *short-term* version of clustering lifetime, based on the ratio of clusters in it that survive in the next timepoint.

Definition 20 (Survival Ratio) *Let ζ_i be the clustering at timepoint t_i for $i = 1, \ldots, n-1$. The* survival ratio *is the portion of clusters in ζ_i that survived (possibly with internal transitions) in ζ_{i+1}:*

$$survivalRatio(\zeta_i) = \frac{|\{X \in \zeta_i | \exists Y \in \zeta_{i+1} : X \rightarrow Y\}|}{|\zeta_i|}$$

The survival ratio only considers those clusters that have been survived at the next timepoint. We relax the notion of survival by also considering the clusters that have been absorbed by some cluster at the next timepoint, thus we define the notion of absorption ratio.

Definition 21 (Absorption Ratio) *The* absorption ratio *of ζ_i is the portion of its clusters that became absorbed by clusters of ζ_{i+1}:*

$$absorptionRatio(\zeta_i) = \frac{|\{X \in \zeta_i | \exists Y \in \zeta_{i+1} : X \overset{\subseteq}{\rightarrow} Y\}|}{|\zeta_i|}$$

Considering both survival and absorption ratios, the passforward ratio arises, which can be consider as a measure of the persistence of the population at the next timepoint.

Definition 22 (Passforward Ratio) *The* passforward ratio *of ζ_i is the portion of its clusters that survived or became absorbed by clusters of ζ_{i+1}, i.e., the sum of the survival ratio and the absorption ratio.*

$$passforwardRatio(\zeta_i) = survivalRatio(\zeta_i) + absorptionRatio(\zeta_i)$$

The passforward ratio indicates the extent to which a clustering describes the accumulated data of the next timepoint. If the passforward ratio is low, then the clustering lifetime is also low, although some clusters in it may have survived for several further timepoints.

6.5.4.2 Evolution Graph querying

The Evolution Graph contains a wealth of information regarding the evolution of the underlying population. Different queries might be imposed over the Evolution Graph so as to facilitate the end user to gain insights in the population and its evolution.

Some indicative queries are as follows:

- **Forward History Queries:** *How does X evolves after t_i?*
  Answer sketch: Start from X and follow its outgoing edge(s) until its descendant(s) disappears.
- **Backward History Queries:** *How did X emerge?*
  Answer sketch: Start from X and follow its ingoing edge(s) until its ancestor(s) appears for the first time.
- **Comparison Queries:** *What other clusters have a similar forward history of transitions or backward history, or both as X?*
  Answer sketch: Use the history of X as the query object and check whether the sequence of transitions that X has encompassed agrees with the sequence of transitions encompassed by some other cluster.
- **Impact Queries:** *Which clusters and at which timepoints have most influenced X into its current shape and content?*
  Answer sketch: Assign an importance factor to each cluster Y participating in the history of X based on factors like: the overlap of Y with respect to its outgoing cluster/node on the history of X, the distance of Y (in terms of the number of the in–between edges) between Y and X etc. Order clusters with respect to this importance factor.

6.6 The FINGERPRINT framework for summarizing cluster evolution

The Evolution Graph captures the whole history of the population under observation and allows the study of cluster transitions and the inspection of cluster interrelationships. However, as the period of observation increases the graph grows significantly and its manual exploitation by the end user becomes difficult. Who could comprehend for example a graph containing 1000's of nodes/clusters and their transitions? Definitely, such a vast of information can be hardly understood by the end user and thus there is a need for summarizing the graph in a compact yet informative way.

To this end, we exploit the fact that the graph is redundant. More specifically, the graph contains information about each change and also information about clusters that did not change at all or change slightly. Hence, we summarize the EVOLUTION GRAPH in such a way so that cluster transitions are reflected but redundancies are omitted. To this end we summarize traces, i.e., sequences of cluster survivals, into some condensed form which we call *fingerprint*. The summaries of all traces in the Evolution Graph constitute the FINGERPRINT of the EVOLUTION GRAPH.

We choose to summarize traces, i.e., survival transitions, because a survival indicates that the source cluster is somehow similar to the target cluster. On the contrary, some split, absorption or disappearance transition, indicates that some important change has occurred in the source cluster population; such a change should be announced to the end user instead of being suppressed. The same holds for the new appearing clusters; they should be reported to the end user since they reveal the formation of new groups in the population.

We first model the notion of summary for a trace and measure the loss in the accuracy of the trace due to the summarization process, as well as the compactness of the summary with respect to the original trace (Section 6.6.1). Then we propose different summarization techniques for summarizing traces into fingerprints taking into account both information loss and compactness gain criteria.

6.6.1 The Notion of Summary for a Trace

The summarization process is applied over cluster traces, as defined in Definition 17. Each trace T is traversed and the "removable" nodes are identified: These are the nodes that can be replaced by a smaller number of derived nodes, which are called "virtual centers" and are defined below.

Definition 23 (Virtual Center) *Let $\prec c_1 \dots c_m \succ$ be the trace of an emerged cluster c, trace(c) and let $X = \prec c_j \dots c_{j+k} \succ$ be a subtrace of this trace, i.e. a subsequence of adjacent nodes in the trace ($k \leq m-1$, $j \geq 1$). We define the "virtual center" of X, $\widehat{X}$ as a derived node composed of the averages of the labels of the nodes in X:*

$$\widehat{X}[i] = \frac{1}{|X|} \sum_{c_i \in X} \widehat{c}[i]$$

where $\cdot[i]$ is the i^{th} dimension and $\widehat{c}$ denotes the label of cluster c. We use the notation $c \mapsto \widehat{X}$ to indicate that cluster $c \in X$ has been "mapped to" the virtual center $\widehat{X}$.

If labels are centroid-based (Definition 15), $\widehat{X}$ is the center of the centroids of the clusters in X. If labels are keyword-based (Definition 16), $\widehat{X}$ contains the average frequencies of all frequent keywords in the clusters of X.

After introducing the virtual center as the summary of a subtrace, we define the summary of a trace: It consists of a sequence of nodes, each node being either an original cluster or a virtual center that summarized a subtrace.

Definition 24 (Trace Summary) *Let $T = \prec c_1 \dots c_m \succ$ be a trace. A sequence $S = \prec a_1 \dots a_k \succ$ is a "summary" of T if and only if (a) $k \leq m$ and (b) for each $c_i \in T$ there is an $a_j \in S$ such that either $c_i = a_j$ or $c_i \mapsto a_j$, i.e. c_i belongs to a subtrace that was summarized to the virtual center a_j.*

There are several possible summarizations of a trace, each one corresponding to a different partitioning of the trace into subtraces and consequently producing different virtual centers. We are interested in summarizations that achieve high compactness gain while keeping information loss minimal. We generalize these objectives into functions measuring "compactness gain" and "information loss", as explained below.

The replacement of a subtrace X by its virtual center $\widehat{X}$ results in *compactness gain*, since less nodes are stored, but also in *loss of information*, since the original clusters are replaced by a "virtual center". We model the information loss of each original cluster $c \in X$ as its distance from the virtual center $\widehat{X}$ to which it has been assigned after the summarization:

$$ILoss\_cluster(c, \widehat{X}) = dist(\widehat{c}, \widehat{X}) \tag{6.9}$$

where $dist(\widehat{c}, \widehat{X})$ is the distance between the label of the original cluster $\widehat{c}$ and that of the virtual center $\widehat{X}$.

The information loss for a cluster/node is now aggregated at the level of the trace, to which the node belonged. The compactness gain is also defined for traces, as follows:

Definition 25 *Let T be a trace and S be a summary of this trace. The "information loss" of T towards S is:*

$$ILoss\_trace(T, S) = \sum_{c \in T} ILoss\_cluster(c, a_c) \tag{6.10}$$

where $a_c \in S$ corresponds to either the virtual center to which c is mapped after the summarization or to the cluster c itself. In the latter case, $ILoss\_cluster(c, a_c) = 0$.

The "compactness gain" of T towards S is the decrease in the number of nodes and edges that need to be stored:

$$\begin{aligned} CGain\_trace(T,S) &= \frac{(|T|-|S|)+(|T|-1-(|S|-1))}{|T|+|T|-1} \\ &= \frac{2\times(|T|-|S|)}{2\times|T|-1)} \approx \frac{|T|-|S|}{|T|} \end{aligned} \tag{6.11}$$

where $|T|$ is the number of nodes in T and $|T|-1$ the number of edges among its nodes (similarly for S).

Next, we define the "fingerprint" of a trace as a summary, the virtual centers of which are proximal to the original cluster labels, subject to a distance upper boundary δ, so that the information loss effected through the replacement of a cluster by a virtual center is kept low.

Definition 26 (Fingerprint for a Trace) *Let T be a trace and S be a summary of T. S is a "fingerprint" of T if and only if:*

(C1) *For each node $c \in X$ that is replaced by a virtual center $a \in S$ it holds that $dist(\widehat{c}, a) \leq \delta$ and*

(C2) *for each (sub)trace $\prec c_1 \cdot \ldots \cdot c_k \succ$ of T that has been summarized into a single virtual center a it holds that $\forall i = 1l, dots, k-1$: $dist(\widehat{c_i}, \widehat{c_{i+1}}) \leq \delta$.*

By this definition, S is a fingerprint of T if it has partitioned T into subtraces of clusters that are similar to each other (condition *C2*) and each such subtrace has a virtual center that is close to all its original nodes (condition *C1*).

Once traces are summarized to fingerprints, the Evolution Graph can also be summarized, resulting in compactness gain and information loss at the graph level.

Definition 27 *Let EG be an* Evolution Graph *and $\mathcal{T}_{EG}$ be its traceset. For each trace $T \in \mathcal{T}_{EG}$, let S_T be its fingerprint (Def. 26), subject to a threshold δ on the distance among centroids. The set $\mathcal{S}_{EG} := \{S_T | T \in \mathcal{T}_{EG}\}$ is the "fingerprint of the* Evolution Graph*". It effects a compactness gain $CG(EG, \mathcal{S}_{EG})$ and an information loss $IL(EG, \mathcal{S}_{EG})$:*

$$CG(EG, \mathcal{S}_{EG}) = \sum_T CGain\_trace(T, S_T) \tag{6.12}$$

$$IL(EG, \mathcal{S}_{EG}) = \sum_T ILoss\_trace(T, S_T) \tag{6.13}$$

We next present the algorithm `BatchFINGERPRINT` that creates the fingerprint of an EVOLUTION GRAPHby partitioning traces in such a way that their fingerprints can be built. This algorithm requires that the EVOLUTION GRAPHis first constructed and stored as a whole. Then, we present an online algorithm, `IncrementalFINGERPRINT`, that builds the fingerprints of the traces incrementally as new cluster transitions are detected. In this case, the fingerprint of the evolution is built directly, without requiring the construction of the EVOLUTION GRAPHfirst.

6.6.2 Batch Summarization of the EVOLUTION GRAPH

The algorithm `BatchFINGERPRINT` summarizes an EVOLUTION GRAPH EG by identifying its traces, building a fingerprint for each trace and substituting the traces in EG with their fingerprints. `BatchFINGERPRINT` satisfies the two conditions of Definition 26 by applying two heuristics on each (sub)trace T:

- *Heuristic A:* If T contains adjacent nodes that are in larger distance from each other than δ, then the pair of adjacent nodes c, c' with the maximum distance is detected and T is then partitioned into T_1, T_2 so that c is the last node of T_1 and c' is the first node of T_2.

- *Heuristic B:* If T satisfies condition *(C2)* but contains nodes that are in larger distance from the virtual center than δ, then T is split as follows: The node c that has the maximum distance from the virtual center of T, $vcenter(T)$, is detected and T is partitioned into T_1, T_2 so that c is the last node of T_1 and its successor c' is the first node of T_2.

Heuristic A deals with violations of condition *(C2)* and *Heuristic B* deals with violations of condition *(C1)* for (sub)traces that already satisfy *(C2)*. We show the algorithm in Figure 6.9.

`BatchFINGERPRINT` creates a fingerprint of the EVOLUTION GRAPH by traversing the graph, extracting its traces (line 1, condition *C2*) and summarizing each of them (line 4). The "produced" fingerprints of the traces are added to the fingerprint graph FEG (line 5). This operation encapsulates the attachment of a summarized trace to the graph by redirecting the ingoing/outgoing edges of the original trace towards the ends of the summarized trace.

`BatchFINGERPRINT` invokes *summarize_HeuristicA* which recursively splits the trace into subtraces according to *Heuristic A* until *(C2)* is satisfied. If the trace consists of only one node, then this

`BatchFINGERPRINT`(EG)
Input: the EVOLUTION GRAPH~EG
Output: FEG, a fingerprint of EG
1. traverse the EG and extract its traces into $\mathcal{T}$;
2. $FEG = \emptyset$;
3. for each trace $T \in \mathcal{T}$ do
4. $FT = summarize\_HeuristicA(T)$;
5. $FEG.addTrace(FT)$;;
6. end-for
7. return FEG;

$summarize\_HeuristicA(T)$
Input: a trace T
Output: a fingerprint of the trace
1. if $|T| == 1$ then return T;
2. if C_b is not satisfied then
3. find $c \in T$ such that
$\forall(y,z) \in T\_1 : dist(y,z) < dist(c, c\_next)$ and
$\forall(y,z) \in T\_2 : dist(y,z) < dist(c, c\_next)$;
//(c, c_next) is the most dissimilar pair of consecutive nodes in T
4. split T into $T\_1 =\prec c\_1, \ldots, c \succ$ and $T\_2 =\prec c\_next, \ldots, c\_k \succ$;
5. $FT\_1 = summarize\_HeuristicA(T\_1)$;
6. $FT\_2 = summarize\_HeuristicA(T\_2)$;
7. return $\prec FT\_1 \cdot FT\_2 \succ$;
8. else return $summarize\_HeuristicB(T)$;
9. endif

$summarize\_HeuristicB(T)$
Input: a trace T
Output: a fingerprint of the trace
1. $v = vcenter(T)$;
2. if $\forall y \in T : dist(y,v) < \delta$ then //Condition $C1$
3. return v;
4. else
5. find $c \in T$ such that $dist(c,v) = \max\{dist(y,v) | y \in T\}$;
6. split T into $T\_1 =\prec c\_1, \ldots, c \succ$ and $T\_2 =\prec c\_next, \ldots, c\_k \succ$;
7. $FT\_1 = summarize\_HeuristicB(T\_1)$;
8. $FT\_2 = summarize\_HeuristicB(T\_2)$;
9. return $\prec FT\_1 \cdot FT\_2 \succ$;
10. endif

Figure 6.9: `BatchFINGERPRINT` for offline summarization of the EVOLUTION GRAPH

node is returned (line 1). Otherwise, we test whether the trace contains nodes whose labels are further off each other than the threshold δ (line 2). If *(C2)* is satisfied, then *summarize_HeuristicB* is invoked (line 8): It checks for condition *(C1)* and returns the fingerprint of the (sub)trace input to it. If *(C2)* is violated, the trace is partitioned according to *Heuristic A* (lines 3,4) and *summarize_HeuristicA* is invoked for each partition (lines 5, 6). Finally, the summarized

(sub)traces are concatenated (line 7) and returned. This concatenation operation restores or redirects the edges across which the split (line 4) was performed.

The recursive function *summarize_HeuristicB* operates similarly. It takes as input a (sub)trace T that has more than one nodes and satisfies condition *(C2)*. It builds the virtual center for T according to Definition 23. It then checks condition *(C1)* by comparing the distance of the virtual center from each node to δ (line 2). If δ is not exceeded, the virtual center is returned (line 3). Otherwise, T is split at the node that has the highest distance from the virtual center, according to *Heuristic B* (lines 5, 6). The *summarize_HeuristicB* is invoked for each partition (lines 7, 8). The returned fingerprints are concatenated into the fingerprint of T.

6.6.3 Incremental Summarization of the Evolution Graph

The batch summarization algorithm of Figure 6.9 requires as input the complete Evolution Graph, before building its fingerprint. This is resource-intensive, since the graph is growing continuously. We have therefore designed `IncrementalFINGERPRINT`, an algorithm that summarizes the traces incrementally and does not require the a priori construction of the Evolution Graph. We show `IncrementalFINGERPRINT` in Figure 6.10.

`IncrementalFINGERPRINT`(FEG) Input: FEG // the fingerprint built so far
ζ // the most recent clustering, build at timepoint $t\_i - 1$
ξ // the current clustering, build at the current timepoint t_i
Output: FEG // the updated fingerprint

1. E_i=MONIC(ζ, ξ);
2. for each edge$e = (x, y) \in E_i$ do
3. if $e.extTrans \neq$ "survival" then
4. $FEG.addNode(y)$;
5. $FEG.addEdge(e)$;
6. else if $dist(x.label, \widehat{y}) \geq \tau$ then // $C2$ is violated
7. $FEG.addNode(y)$;
8. $FEG.addEdge(e)$;
9. else
10. $v = vcenter(x, y)$;
11. $FEG.replaceNode(x, v)$;
12. endif
13. end-for
14. return FEG;

Figure 6.10: `IncrementalFINGERPRINT` for online construction and summarization of the Evolution Graph

`IncrementalFINGERPRINT` invokes MONIC (line 1), which compares the current clustering ξ (timepoint t_i) to the most recent one ζ (timepoint t_{i-1}), identifies the cluster transitions and returns them as a set E_i of labeled edges, according to Section 6.4. The source of each edge corresponds to a node that is already in the fingerprint graph FEG. It is stressed that MONIC operates on the clusterings rather than the cluster labels retained in the nodes of the fingerprint graph. So, from line 2 on, `IncrementalFINGERPRINT` transfers information about the detected transitions in the FEG, summarizing survivals wherever possible. The result is an already summarized version of the Evolution Graph.

For each edge $e = (x, y)$, `IncrementalFINGERPRINT` examines whether e is a survival transition (line 3), i.e. whether e is part of a trace. If not, FEG is expanded by adding the cluster y and the edge e (lines 4, 5). We do not add the whole cluster; we only retain its label (cf. Subsection 6.5.1.1).

If $e = (x, y)$ does belong to a trace, `IncrementalFINGERPRINT` checks whether the labels of x and y are similar to each other, according to condition *(C2)* of Definition 26 (line 6). Since cluster x has already been added to FEG, we access its label $x.label$ directly, while the label of cluster y must be computed as $\widehat{y}$. If condition *(C2)* is not satisfied, the FEG is expanded by y and e as before. If finally, *C2* is satisfied, then y and e do not need to be added to FEG. Instead, x and y are summarized into their virtual center v (line 10) and the node x is replaced by v (line 11). This means that all edges pointing to x are redirected to v.

`IncrementalFINGERPRINT` does not need to check for condition *(C1)* of Definition 26, because the distance of the virtual center of *two* nodes is less than the distance between the two nodes as a whole; the latter is less than τ by virtue of condition *(C2)*. This incremental algorithm operates locally, treating pairs of adjacent nodes only, instead of whole traces. However, it has the advantage of not requiring the a priori construction of the Evolution Graph.

6.7 Experimental study

We have applied MONIC+ on a synthetic, automatically generated dataset (Section 6.7.1). Also, we have applied MONIC on a real document collection, the ACM Library section H.2.8 on "database applications" from 1997 until 2004 (Section 6.7.2). The goal of the experiments with the synthetic dataset was to demonstrate the potentiality of our framework and to display the kind of transitions

that it can detect; thus, we used a 2D dataset, which can be easily visualized. The goal of our experiments with the real data was to acquire insights on cluster evolution and change in the clusterings, and to study the impact of the parameters on the transition discovery process.

We have also tested FINGERPRINT summarization algorithms on three public datasets and measured the compactness gain and information loss for different values of the centroid similarity threshold δ (cf. Definition 26) that governs the summarization process (Section 6.7.3).

6.7.1 Experiments on MONIC+ over synthetic data

We have tested MONIC+ on a synthetic stream of data records, in which we have imputed cluster transitions (Section 6.7.1.1). We report here on results for type B1 (Section 6.7.1.2) and type A clusters (Section 6.7.1.3).

6.7.1.1 Generation of an accumulating dataset

We used a data generator that takes as input the number of data points M, the number of clusters K, as well as the mean and standard deviation of the anticipated members of each cluster. The records were generated around the mean and subject to the standard deviation, following a Gaussian distribution. We fixed the standard deviation to 5 and used a 100×100 workspace for two-dimensional datapoint. The stream was built according to the scenario below (c.f. Figure 6.3).

- t_1**:** Dataset d_1 consists of points around the $K_1 = 5$ centers (20,20), (20, 80), (80, 20), (80, 80), (50, 50).
- t_2**:** Dataset d_2 consists of 40 datapoints, distributed equally across the four corner-groups of d_1 data.
- t_3**:** Dataset d_3 consists of 30 points around location (50,40) and 30 points around (50,60).
- $t_4, \ldots$**:** At each of t_4, t_5, t_6 timepoints we added 30 points around t_4 :(20,50), t_5 :(20,30) and t_6 :(20,40).

For data ageing, we used a sliding window of size $w = 2$. Hence, at each timepoint $t_i, i > 1$, the dataset under observation was $D_i = d_i \cup d_{i-1}$.

MONIC+ traces cluster transitions upon clusterings already found by the algorithm. However, some algorithms can discover

the optimal number of clusters, while others require it as input. In the latter case, the discovery of the external transitions is influenced. Since our objective is to show the functionality of our framework rather than to evaluate the algorithms, we gave as input to K-means [100] the optimal number of clusters as suggested by EM [100]. Hierarchical algorithms require a similar tuning: We had to specify the number of clusters or the layer of the dendrogram to be returned. Since the optimization criteria of the Type B1 clustering algorithm are not relevant for the post-clustering monitoring we perform, we have rather used the clusters delivered by EM as sets.

For a real application, we recognize that algorithms demanding the number of clusters as input will indeed imped the detection of external transitions. Nonetheless, internal transitions may indicate the existence of external ones. For example, if a cluster increases in size, becomes more diffuse and its center moves, then it may be worth testing whether it now covers the area of two clusters. In such a case, the expert should launch the algorithm with alternative values for the expected number of clusters.

6.7.1.2 Clustering and transition detection for type B1 clusters

We have generated and monitored B1 clusters upon the datasets $D_1, \ldots, D_6$ using the indicators introduced in the MONIC framework (Section 6.3). We have set $\tau \equiv \tau_{match} = 0.5$, $\tau_{split} = 0.2$ and $\varepsilon = 0.003$.

The clusterings results $\zeta_i, i = 1 \ldots 6$ are depicted in Fig. 6.11. This figure depicts the clusters at each timepoint but delivers little information about the impact of new data and of data ageing. In Table 6.8, the changes in the population are reflected in the discovered transitions.

	Cluster Transitions				
t_2	$C_{11} \nearrow C_{21}$	$C_{12} \nearrow C_{22}$	$C_{13} \nearrow C_{23}$	$C_{14} \nearrow C_{24}$	$C_{15} \rightarrow C_{25}$
t_3	$C_{21} \rightarrow C_{31}$	$C_{22} \rightarrow C_{32}$	$C_{23} \rightarrow C_{33}$	$C_{24} \rightarrow C_{34}$	$C_{25} \nearrow C_{35}$
t_4	$C_{31} \rightarrow \odot$ $\odot \rightarrow C_{41}$	$C_{32} \rightarrow \odot$ $\odot \rightarrow C_{42}$	$C_{33} \rightarrow \odot$ $\odot \rightarrow C_{43}$	$C_{34} \rightarrow \odot$ $\odot \rightarrow C_{44}$	$C_{35} \overset{\subseteq}{\rightarrow} \{C_{45}, C_{46}\}$ $\odot \rightarrow C_{47}$
t_5	$C_{41} \rightarrow \odot$	$C_{42} \rightarrow \odot$	$C_{43} \rightarrow \odot$ $\odot \rightarrow C_{51}$	$C_{44} \rightarrow \odot$ $C_{46} \rightarrow C_{54}$	$C_{45} \rightarrow C_{53}$ $C_{47} \rightarrow C_{52}$
t_6	$C_{51} \overset{\subseteq}{\rightarrow} C_{61}$	$C_{52} \overset{\subseteq}{\rightarrow} C_{61}$	$C_{53} \rightarrow \odot$	$C_{54} \rightarrow \odot$	

Table 6.8: Transitions for Type B1 clusters

As one can see, there exist both external and internal transitions. By juxtaposing the clusters in this table with the visualization in

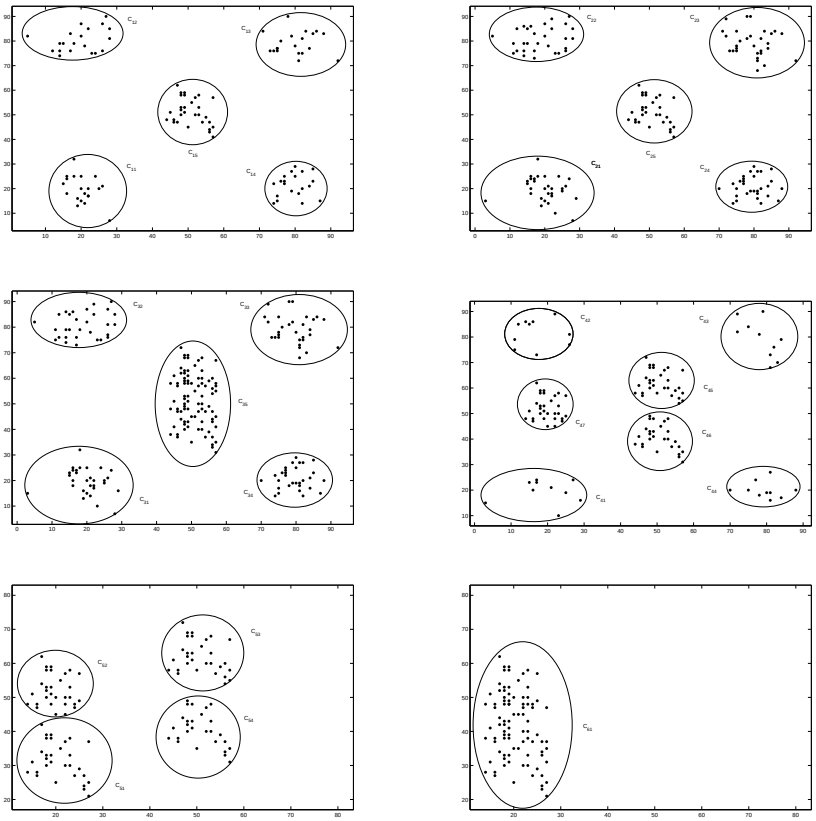


Figure 6.11: Type B1 clusters (found with EM) at timepoints t_1, t_2 (top); t_3, t_4 (middle) and t_5, t_6 (down)

Fig. 6.11, we can see that MONIC has correctly mapped the old clusters to the new ones, identifying survivals with or without internal transitions, absorptions and splits. Some clusters have experienced multiple internal transitions, e.g. C_{12} has expanded and shifted into C_{22}, which furthermore, is more compact than its predecessor. There are also new clusters found at t_4 and t_5.

6.7.1.3 Transitions of Type A Clusters

We have generated and monitored type A clusters upon the datasets $D_1, \ldots, D_6$ using as K the optimal number of clusters found by the EM algorithm.

The clusterings results $\zeta_i, i = 1, \ldots, 6$ are depicted in Fig. 6.12; they are different from the EM clusters, thus implying also different cluster transitions. We have used the indicators in Table 6.5, setting $\tau = 0.5$ and $\tau_{split} = 0.2$. For the size transition, we have used the

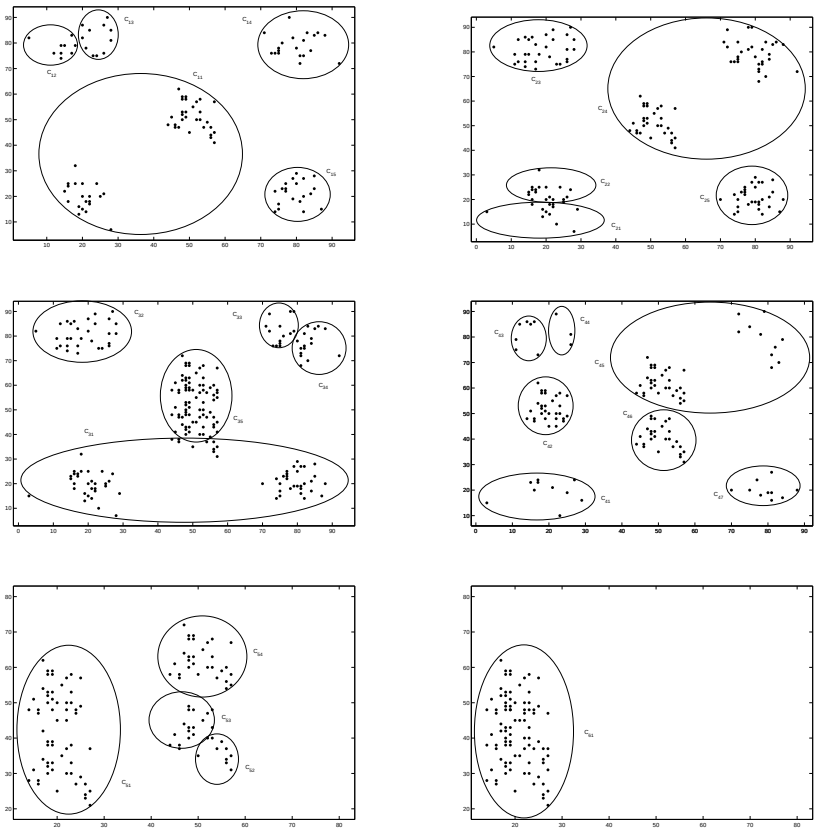


Figure 6.12: Type A clusters (found with K-means) at timepoints t_1, t_2; t_3, t_4 and t_5, t_6

B1 indicator in Table 6.3 with $\varepsilon = 0.003$. For the other internal transitions, we have used the indicators for spheres in Table 6.6 with $\tau_{location} = 0.1$ (location transitions) and $\varepsilon = 0.001$ (compactness transitions).

The transitions found by MONIC+ are shown in Table 6.9 and reveal that most clusters are unstable, experiencing all types of internal transitions, or they disappear, giving place to new (unstable) clusters. Even in the absence of a visualization (which might be difficult for a real dataset in a multi-dimensional feature space), these transitions indicate the cluster instability and the need for closer inspection of the individual clusters.

Comparing the transitions of clusters extracted through K-means (Figure 6.12) and EM (Figure 6.11), one can observe that K-means clusters experience more transitions, which is justified by the fact that K-means results in less stable clusters.

	Cluster Transitions				
t_2	$C_{11} \to \odot$	$C_{12} \stackrel{\subseteq}{\to} C_{23}$	$C_{13} \stackrel{\subseteq}{\to} C_{23}$ $\odot \to C_{21}$	$C_{14} \to \odot$ $\odot \to C_{22}$	$C_{15} \cdots \stackrel{\bullet}{\to} \nearrow C_{25}$ $\odot \to C_{24}$
t_3	$C_{21} \to \odot$	$C_{22} \to \odot$ $\odot \to C_{31}$	$C_{23} \to C_{32}$ $\odot \to C_{33}$	$C_{24} \to \odot$ $\odot \to C_{34}$	$C_{25} \to \odot$ $\odot \to C_{35}$
t_4	$\odot \to C_{41}$	$\odot \to C_{47}$	$C_{33} \to \odot$	$C_{34} \to \odot$ $\odot \to C_{42}$	$C_{35} \cdots \stackrel{\star}{\to} \searrow C_{45}$ $C_{32} \stackrel{\subseteq}{\to} \{C_{43}, C_{44}\}$ $C_{31} \cdots \stackrel{\bullet}{\to} \searrow C_{46}$
t_5	$C_{41} \to \odot$	$C_{47} \to \odot$	$C_{43} \to \odot$	$C_{44} \to \odot$	$C_{45} \cdots \stackrel{\bullet}{\to} \searrow C_{54}$ $C_{46} \stackrel{\subseteq}{\to} \{C_{52}, C_{53}\}$ $C_{42} \cdots \stackrel{\star}{\to} \nearrow C_{51}$
t_6		$C_{52} \to \odot$	$C_{53} \to \odot$	$C_{54} \to \odot$	$C_{51} \stackrel{\bullet}{\to} \nearrow C_{61}$

Table 6.9: Transitions for Type A clusters

6.7.2 Experiments on MONIC over the ACM H.2.8 document collection

We first describe the ACM H2.8 dataset (Section 6.7.2.1) and then we present our results (Section 6.7.2.2).

6.7.2.1 Section H2.8 of the ACM digital library

ACM library section H2.8 "Database applications" contains publications on *(1)* data mining, *(2)* spatial databases, *(3)* image databases, *(4)* statistical databases, *(5)* scientific databases categorized in the corresponding classes. It further contains *(6)* uncategorized documents, i.e., those assigned in the parent class "database applications" only, as well as those documents from other parts of the ACM library, which have one of the subarchive's classes as secondary class. In the latter case, the documents are treated identically to those have one of the first five classes as primary class.

We have considered those documents from 1997 to 2004 that have a primary or a secondary class in H2.8, i.e., one of the six classes above. For each document, we have considered the title and the list of keywords; we omitted the abstracts because they are only available for late periods; also, the keywords themselves should be adequate for class separation.

Before proceeding with the experiments, some remarks on the H.2.8 collection are due. This collection is unbalanced with respect to the six classes; the class "Data Mining" is larger than all the others together. Many clustering algorithms have difficulties with such data distributions. Moreover, this class grows faster than the others, while some of the smallest classes stagnate.

We have designed several alternative feature spaces, ranging from the whole set of words to a small list of frequent words. We have also considered alternative weighting schemes, including the embedded mechanism of CLUTO [51] and the entropy-based feature weighting function proposed in [15]. Best results were acquired for a feature space consisting of the 30 most frequent words with TFxIDF term weighting and for the method of [15]. We have opted for the former, computationally simpler approach.

For clustering, we have experimented with Expectation–Maximization [101], with a hierarchical clusterer using single linkage, with CLUTO and with bisecting K-means. Best results were obtained with bisecting K-means for K=10 (rather than 6), so our experiments were performed with this setting.

Document import, vectorization and clustering was done with the DIAsDEM Workbench open source text mining software [24]. To deal with data ageing, we applied a sliding window of size 2, i.e., documents older than two time periods acquired zero weight. The cluster transitions found by MONIC are presented in the next subsubsection.

6.7.2.2 Cluster transitions and impact of thresholds

We have first varied the threshold τ_{match} from 0.45 (rather than 0.50) to 0.7 in steps of 0.05 and depicted the number of clusters that experienced internal or external transitions. For values of τ_{match} larger than 0.7, there were hardly cluster survivals, so we omit these values. For cluster splitting, we have set $\tau_{split} = 0.1$. The results are illustrated in Fig. 6.13.

In Fig. 6.13(a) we can see that the number of surviving clusters drops as τ_{match} becomes more restrictive. The number of splits and disappearing clusters in Fig. 6.13(b), respectively (c) increases accordingly. At the same time, all surviving clusters experience changes in size. Furthermore, we have not detected any absorption transitions. Hence, the passforward ratio is equal to the survival ratio for all clusterings.

A comparison of the numbers for each timepoint, Fig. 6.13(a), (b), (c), reveals that the clusters in early clusterings tend to disappear and be replaced by new ones (more disappearances than splits), while the trend reverses in late clusterings: The clusters in recent timepoints are rather split than scattered. This might be explained by the increasing volume of the document collection: The number of documents inserted at each timepoint increases rapidly at the late timepoints, so that unstable clusters may be split by the clusterer

in large chunks instead of being dissolved and rebuilt.

To check this hypothesis, we analyzed the influence of the threshold τ_{split} upon the number of splits and disappearances, as shown in Fig. 6.14. We have varied τ_{split} from 0.1 to 0.35 with a step of 0.05, setting $\tau_{match} = 0.5$. As expected, large values of τ_{split} result in a higher number of disappearing clusters. However, the numbers of splits at late timepoints indicate that splits are only possible if the value of τ_{split} is small. Hence, the clusters are not split into large chunks, they are indeed dissolved and rebuilt. For example, the dominant class "Data Mining" grows substantially in the recent timepoints but is not homogeneous enough to produce clusters with a long lifetime.

6.7.2.3 Lifetime of clusters and clusterings

At the next step, we studied the persistence of the clusterings over time. We first computed the passforward ratios (c.f. Definition 22). The results are listed in clusters for simplicity; since $K = 10$, the relative numbers are trivial to compute. It is apparent from this table that the clusterings at some timepoints show a high, respectively low, passforward ratio, independently of the τ_{match} values: The low passforward ratio at timepoint 2002 indicates a drastic change in the documents between 2001 and 2002 (window size = 2), that has been preceded by a rather stable period of two years; the clusterings of 2000 and 2001 have quite high passforward ratios.

τ	*1999*	*2000*	*2001*	*2002*	*2003*	*2004*
0.45	4	7	7	1	5	4
0.50	4	5	7	1	3	4
0.55	3	3	3	0	2	3
0.60	3	2	3	0	1	1
0.65	3	0	1	0	0	1
0.70	2	0	1	0	0	0

Table 6.10: Passforward ratios for different values of survival threshold

The rather low passforward ratio of the clusterings is reflected in the lifetimes of the individual clusters. We have studied cluster lifetime according to . To this purpose, we have set $\tau_{match} = 0.5$ and $\tau_{split} = 0.1$ and have computed the lifetime with internal transitions for the clusters, *lifetimeI*: Since all survived clusters experience internal transitions, the strict lifetime is 1 for all of them. Since no absorptions have occurred, the lifetime with absorptions is equal to lifetimeI for all clusters. The results are presented in the table below. The second column shows the lifetimes of all clusters in each

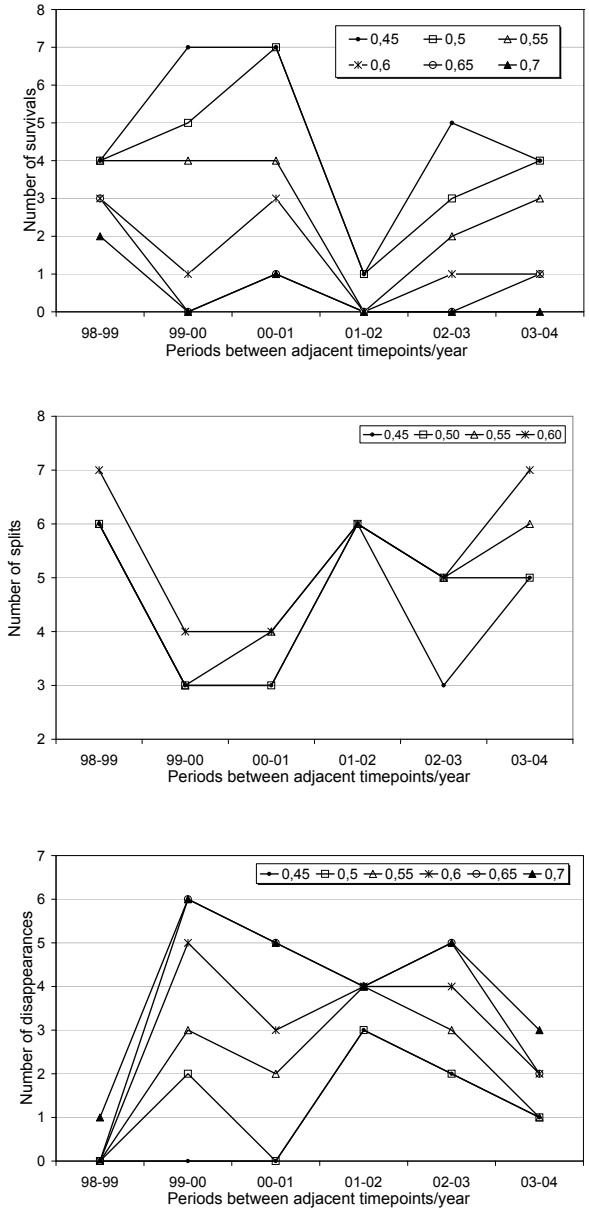


Figure 6.13: Cluster transitions for different values of τ_{match}: (a) Survived clusters, (b) split clusters and (c) disappeared clusters

clustering.

The third column in Table6.11 is the lifetime of the clusterings according to . It is obvious that all timepoints are characterized by short-lived clusterings, although some of them contain rather stable

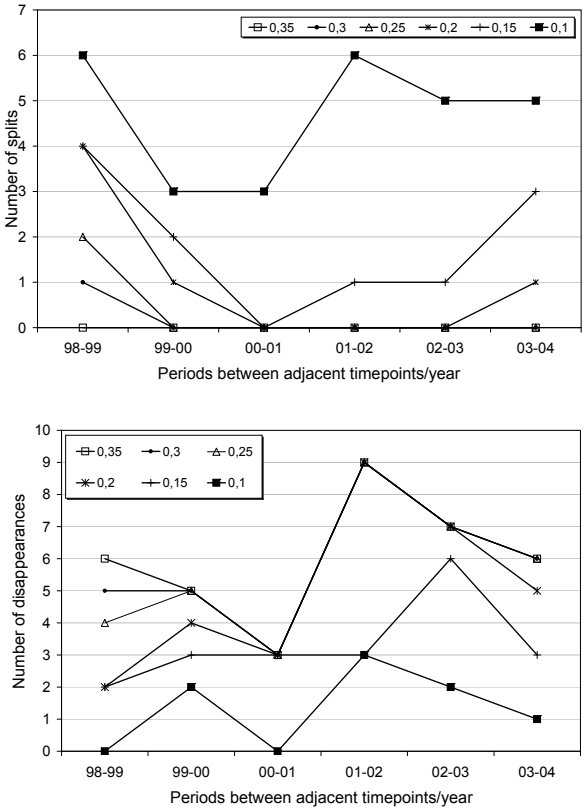


Figure 6.14: Cluster transitions for different values of τ_{split}: (a) Split clusters and (b) disappeared clusters

Timepoint	*Lifetimes of clusters*	*Lifetime L*
1998	{4,1,4,1,4,1,2,1,1,1}	1
1999	{4,1,4,1,1,4,1,2,1,3}	1
2000	{4,1,4,1,3,2,4,3,2,1}	2
2001	{4,1,4,2,4,2,3,2,1,1}	2
2002	{1,1,3,1,1,1,1,1,1,1}	1
2003	{3,1,1,1,1,1,1,2,3,1}	1
2004	{3,3,1,2,1,1,2,1,1,1}	1

Table: Lifetime of clusterings

individual clusters.

6.7.2.4 Clusters vs Classes of the ACM Library

So far, we have studied evolution of the clusters in H.2.8 without considering the real ACM classes. To this purpose, we juxtapose the transitions and cluster lifetimes found by MONICo the "real",

observable evolution of H.2.8. To do so, we have labeled each cluster with its two most frequent words and mapped these labels/"topics" to the ACM classes. For cluster transition detection, we have set $\tau = 0.5$ and $\tau_{split} = 0.1$ and concentrated on splits, disappearances and cluster lifetime with internal transitions (lifetimeI), since there were no strict survivals and no absorptions. On this basis, we have checked whether cluster transitions correspond to comprehensible topic evolutions. The results are as follows:

- There is always one cluster without a label, hereafter denoted as "cluster 0". The clusterer places in this cluster all records that cannot be accommodated elsewhere. By nature, this uninformative cluster has a high lifetime of 4 timepoints. However, it does not survive the population shift at timepoint 2002; at this timepoint it is dissolved and rebuilt.

- Each clustering contains two or three clusters on data mining, the dominant class. In the first 4 timepoints, we find a growing cluster on "association rules". In 2002, it is split into a smaller cluster with the same label and an unlabeled noisy cluster (other than cluster 0):

$$C_{1998_4} \nearrow C_{1999_9} \nearrow C_{2000_6} \nearrow C_{2001_4} \stackrel{\subset}{\rightarrow} \{C_{2002_7}, C_{2002_9}\}$$

  where denote as C_{y_w} the identifier of the "association rules" cluster in year y, $w = 1 \ldots 9$ [1].
  The small cluster C_{2002_7} disappears in 2003 ($C_{2002_7} \rightarrow \odot$). One of the emerging clusters of 2004 ($\odot \rightarrow C_{2004_3}$) has again the label "association rules".

- The other clusters on data mining have less specific labels, such as "knowledge discovery" or "data mining". Their lifetime does not exceed 3 timepoints, during which they experience splits and size transitions.

- At the early timepoints, there are clusters labeled "spatial" and "image" (later: "image retrieval"). The labels appear in several periods but are associated with different clusters, so the cluster lifetime is low. Clusters associated to classes other than "Data Mining" appear only until 2002.

- The number of clusters with a label is large in the clusterings of the early timepoints and decreases in the clusterings of the

[1] Cluster identifiers are generated by the clustering algorithm at each timepoint. They do not indicate transitions.

late timepoints. The labels of the late timepoints are shorter and less informative ("model", "data"). Clusters that can be associated to classes other than "Data Mining" appear only at the early timepoints.

Hence, MONICetected a remarkable shift in the accumulating H2.8 section between 2001 and 2002, signaled by an increased number of cluster splits and disappearances. The history of H.2.8 contains at least one event that may explain this shift: Starting with KDD'2001, the proceedings of the conference and of some adjoint workshops are being uploaded in the ACM Digital Library, enriching the H.2.8 section with a lot of documents on many subtopics of data mining.

The results are indicative of the types of transitions that may be caused by a population shift in an unlabeled dataset and of the potential of detecting and understanding shifts through the monitoring of cluster transitions.

6.7.3 Experiments on FINGERPRINT

We first describe the datasets used in the experiments and provide some example traces and their fingerprints. Then, we discuss the findings for each dataset.

6.7.3.1 Datasets

We experimented with two numerical datasets, the *Network Intrusion dataset* and the *Charitable Donation dataset*, used also in the stream experiments of [2], and with the documentset *ACM H2.8* used in the experiments of MONIC(c.f.ection6.7.2.1). The first dataset is rapidly evolving, the second one is relatively stable, while the third one evolves in an unbalanced way – one of the classes grows faster than the others.

The *Network Intrusion dataset* (KDD Cup'99) contains TCP connection logs from two weeks of LAN network traffic (424,021 records). Each record corresponds to a normal connection or an attack. The attacks fall into four main categories: DOS (i.e., denial-of-service), R2L (i.e., unauthorized access from a remote machine), U2R (i.e., unauthorized access to local superuser privileges), and PROBING (i.e., surveillance and other probing). So, we set the number of clusters to 5, including the class of normal connections. We used all 34 continuous attributes for clustering and removed one outlier point, as in [2]. We turned the dataset into a stream by sorting on the data input order. We assumed a uniform flow in speed

of 2000 instances per time period. For data ageing, we assumed a sliding window of 2 time periods/timepoints.

The *Charitable Donation dataset* (KDD Cup'98) contains information (95,412 records) on people who have made charitable donations in response to direct mailings. Clustering identifies groups of donors with similar donation behavior. Similar to [28], we used 56 out of the 481 fields and set the number of clusters to 10. As with the previous dataset, we used the data input order for streaming and assumed a uniform flow with 200 instances per time period. For data ageing, we used a sliding window of size 2.

The *ACM H2.8 subarchive* has been already described in Section6.7.2.1, we use here the same settings. Just recall that it evolves in an unbalanced way: The category (1) is larger than all the others together and grows faster than the others.

6.7.3.2 Example traces and fingerprints

To highlight the behavior of the summarization algorithms on real data, we depict here some traces from the *ACM H2.8 dataset* and their fingerprints, as produced by our summarization algorithms.

In 1998, we observe a new cluster with the label "information systems". Its trace is $trace(c_{1998_2}) =\prec c_{1998_2} c_{1999_6} c_{2000_3} \succ$, where the notation c_{y_i} refers to the i^{th} cluster of year y, with $i = 1 \ldots 9$ (cluster 0 is the garbage cluster) [2]. The cluster centroids contain the terms "information" and "system" with the following frequencies:
$\widehat{c_{1998_2}} =< information(0.96), system(0.61) >$,
$\widehat{c_{1999_6}} =< information(0.88), system(0.74) >$ and
$\widehat{c_{2000_3}} =< information(0.76), system(0.78) >$.
Both summarization algorithms condense this trace into a single virtual center. The batch algorithm creates this new node v in one step $\widehat{v} =< information(0.87), system(0.71) >$,
while the incremental first summarizes c_{1998_2} and c_{1999_6} into a virtual center $\widehat{v_0} =< information(0.92), system(0.68) >$,
and then summarizes v_0 and c_{2000_3} into
$\widehat{v'} =< information(0.84), system(0.73) >$.

A further cluster that emerged in 1998 had the one-term label $< analysis(1.0) >$. In 1999, it was split into two clusters, one labeled $< mining(1.0), datum(0.74) >$ and one cluster with no label (garbage cluster). The former survived for two periods, thus resulting in the trace $\prec c_{1999_8} c_{2000_4} c_{2001_6} \succ$. The information delivered without summarization is the sequence $c_{1998_9} \xrightarrow{\subseteq} \{c_{1999_4}, \prec$

[2] Cluster identifiers are generated by the clustering algorithm at each timepoint

$c_{1999_8} c_{2000_4} c_{2001_6} \succ\}$; the summarization delivers the fingerprint $c_{1998_9} \xrightarrow{\subseteq} \{c_{1999_4}, v\}$ instead.

6.7.3.3 Compactness Gain and Information Loss

In Figures6.15, 6.16 and 6.17, we show the compactness gain achieved by the batch and the incremental summarization methods for each of the three datasets, considering different values of the centroid distance threshold δ. The values for the different dimensions of the centroids are between 0 and 1, so we vary δ in this range as well.

We can see in the figures that the two algorithms achieve similar compactness gain, although `IncrementalFINGERPRINT`hows slightly lower values for most values of δ in the *Network Intrusion* dataset. Obviously, the compactness gain increases for larger values of δ, because less proximal centroids can be merged.

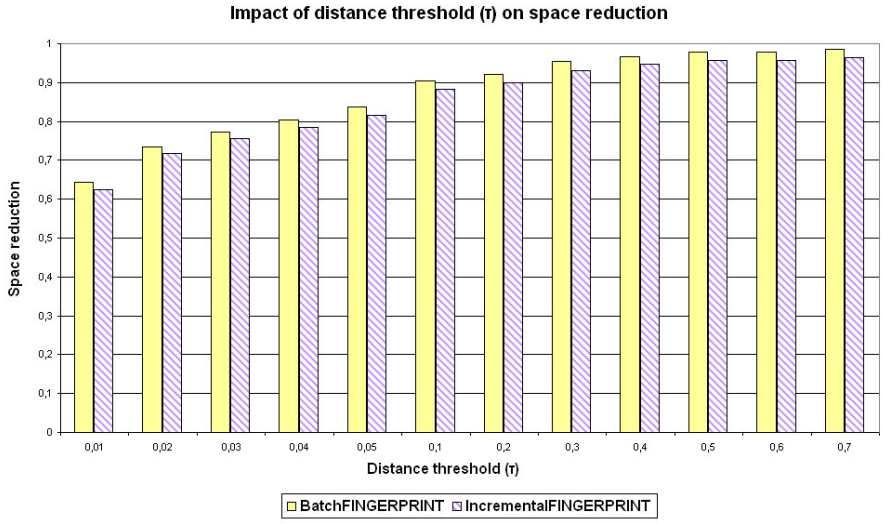


Figure: Network Intrusion dataset: Impact of threshold δ on compactness gain

The total compactness gain for each dataset depend of course on the number of survivals per se: Among the total of 1,195 clusters/nodes generated for the *Network Intrusion* dataset, 400 nodes participate in traces; the compactness gain achieved by both algorithms are in the range $[0.21, 0.33]$ of the total size of the Evolution Graph. The Evolution Graph of the *Charitable Donation* dataset contained 4,770 clusters, of which 614 were involved in traces; the compactness gain over the whole graph were thus no more than 7%. For the *ACM H2.8* subarchive, 24 out of 70 nodes were involved in

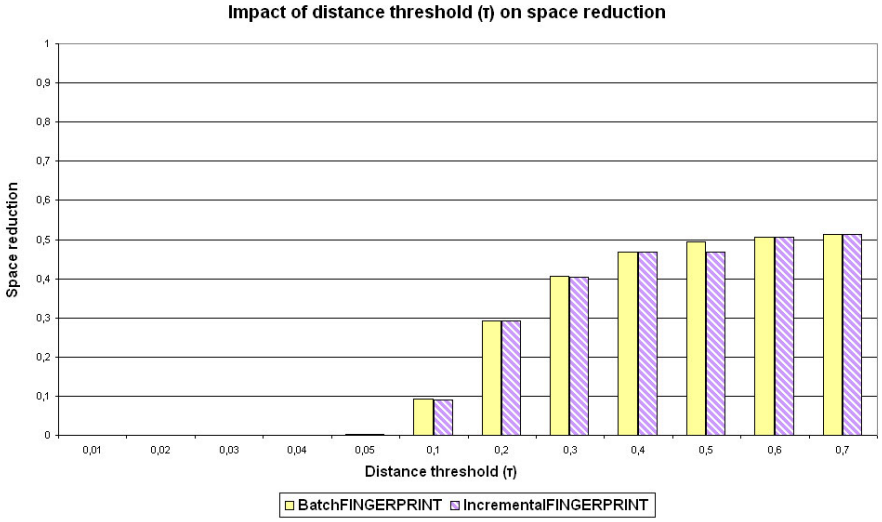


Figure: Charitable Donation dataset: Impact of threshold δ on compactness gain

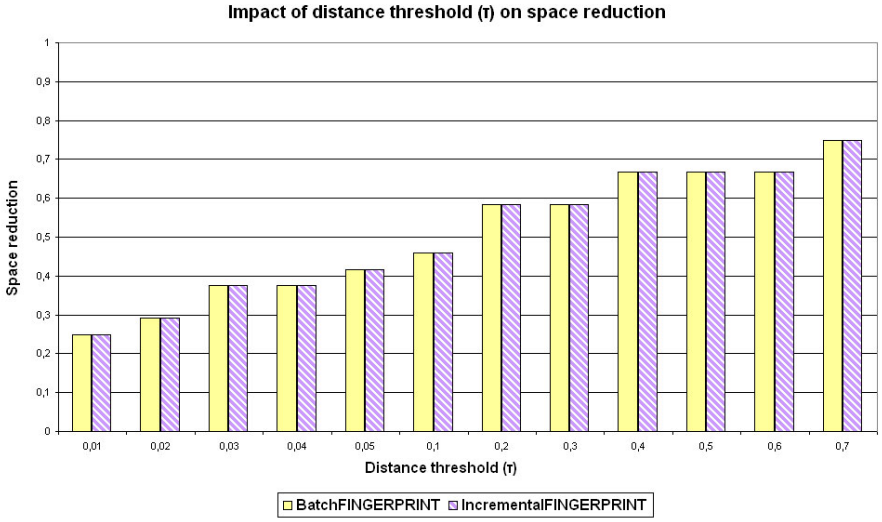


Figure: ACM H.2.8 dataset: Impact of threshold δ on compactness gain

traces, so that the compactness gain over the whole graph ranged between 9% and 33%.

In Figures6.18, 6.19 and 6.20, we depict the information loss effected upon the three datasets when summarizing incrementally versus in batch. For the *Charitable Donation* and the *ACM H2.8*

datasets, the information loss incurred by the incremental algorithm is slightly higher than for `BatchFINGERPRINT`ut follows the same curve for different values of δ. For the *Network Intrusion* dataset, the performance difference is dramatic: While `BatchFINGERPRINT`chieves a very low information loss (lower than for the other datasets), the incremental algorithm performs very poorly.

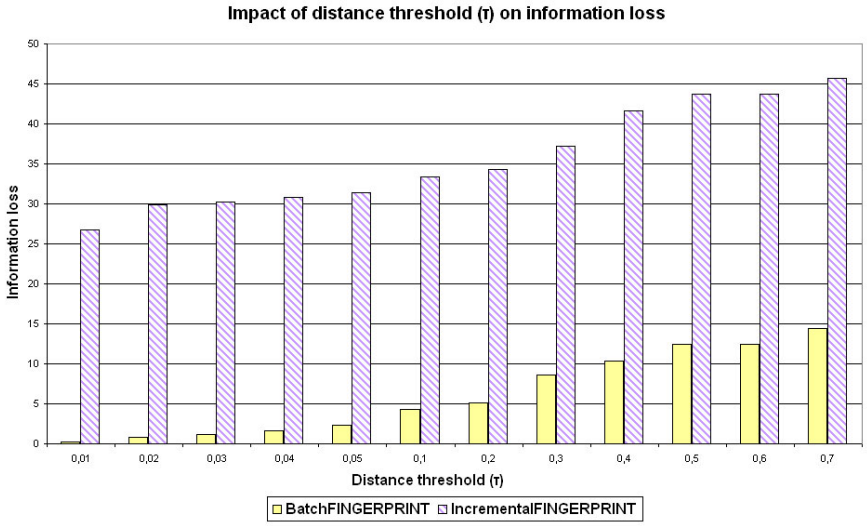


Figure: Network Intrusion dataset: Impact of δ on information loss

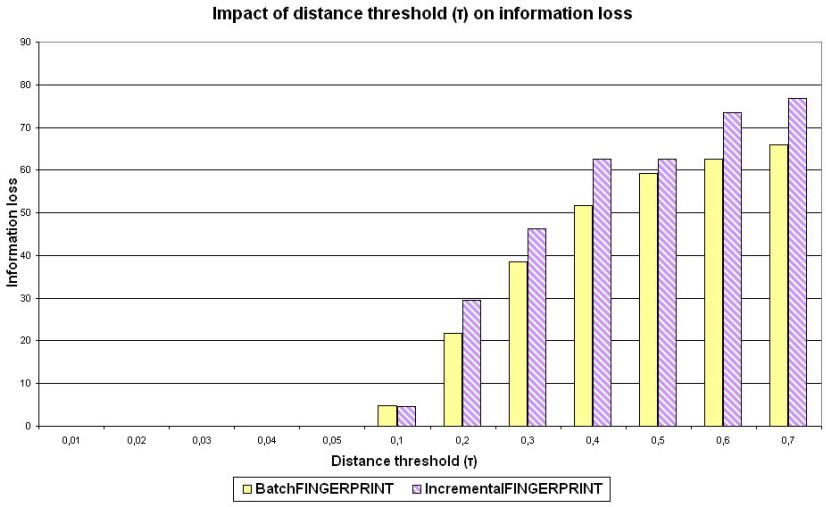


Figure: Charitable Donation dataset: Impact of δ on information loss

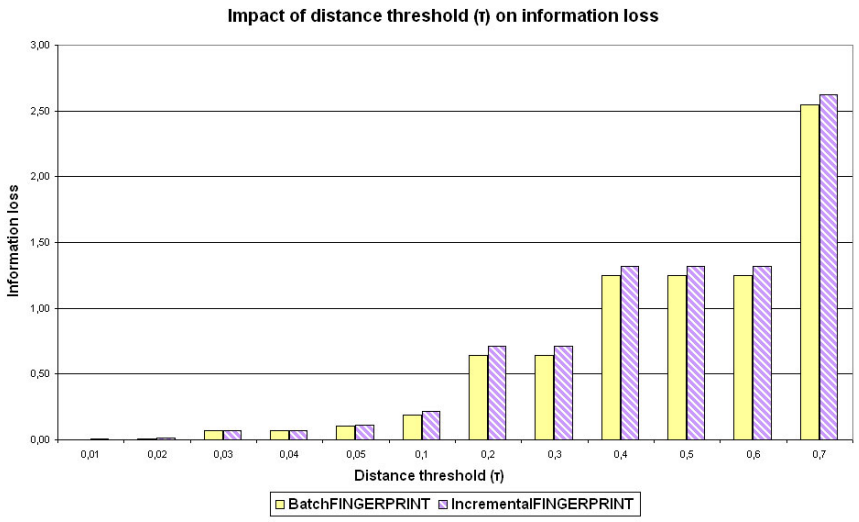


Figure: ACM H.2.8 dataset: Impact of δ on information loss

Comparing the compactness gain and information loss values for the datasets *Charitable Donation* and *ACM H2.8*, we can say that both algorithms summarize the traces in a similar way, although the batch algorithm manages to summarize slightly more nodes (higher compactness gain) and more similar ones (lower information loss). This is also reflected in Figure6.22 and Figure6.23, where we depict the joint curves of information loss and compactness gain: The curves of the two algorithms are almost identical.

The performance difference between the two algorithms is made more clear in the joint curves shown in Figure6.21. One explanation for the performance of `IncrementalFINGERPRINT`ight be the volatility of the dataset: The number of survivals is relatively low and it is likely that the survived clusters were unstable and not very similar to each other. Hence, `IncrementalFINGERPRINT`roduced virtual centers that were not very close to the original pairs of centroids, while `BatchFINGERPRINT`anaged to build better virtual centers among multiple adjacent centroids. More experimentation is needed here, though. In particular, it is necessary to highlight the quality of the original clusters and its impact on the compactness gain and information loss.

6.8 Related work

Research relevant to our work can be categorized into cluster monitoring methods and cluster summarization methods.

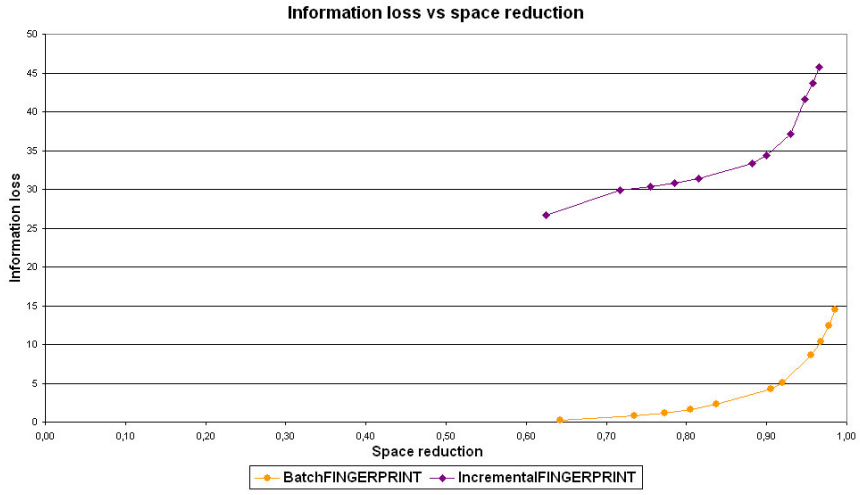


Figure: Network Intrusion dataset: Correlation between information loss and compactness gain

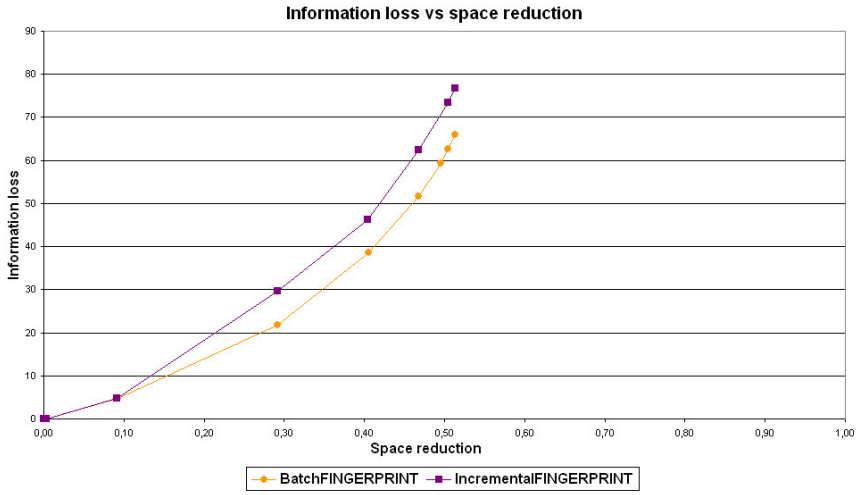


Figure: Charitable Donation dataset: Correlation between information loss and compactness gain

Monitoring methods Relevant to our work is the research on methods for cluster change detection and spatiotemporal clustering. We also discuss methods on the specific subject of topic evolution [61,,,66].

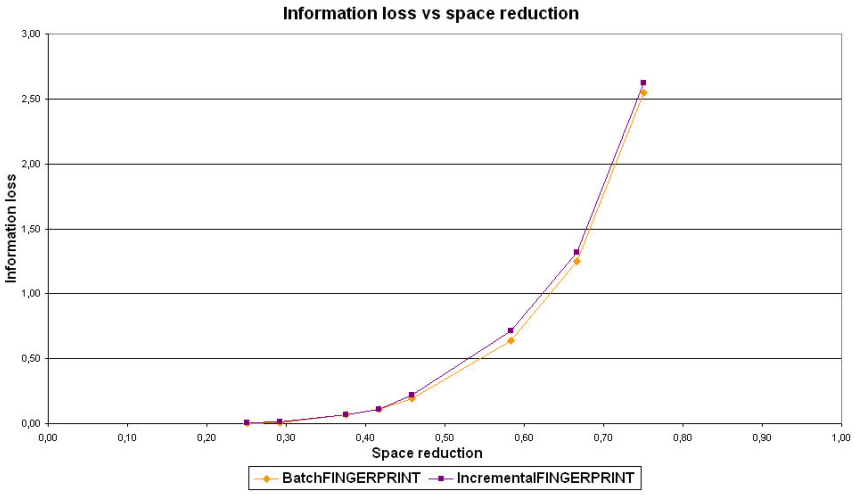


Figure: ACM H.2.8 dataset: Correlation between information loss and compactness gain

Cluster change detection The FOCUS change detection framework [32] compares two datasets and computes a deviation measure between them, based on the data mining models they induce. Clusters compose a special case of models, described as non-overlapping regions described through a set of attributes (structure component) and corresponding to a set of raw data (measure component). However, the emphasis in this work is on comparing datasets, not in understanding how a cluster has evolved inside a new clustering.

In [62], Meila provides an overview of the related work on comparing different clustering results produced from the same dataset under different mining parameters, e.g.,ifferent algorithms or different parameters over the same algorithm. The comparison process relies on criteria based on i) counting pairs, ii) cluster matching and iii) variation of information. The counting pairs criteria are based on counting the pairs of points on which the two clusterings agree (i.e.,lace them in the same cluster). The cluster matching criteria are based on finding for each cluster of the first clustering its best match at the second clustering, where the best match is evaluated based on the number of common points between the two clusters. Recently, Zhou et al [107], proposed a new measure in this category that also considers the distance between cluster centroids in order to yield more intuitive results. The variation of information criteria measure the amount of information that is lost or gained when mov-

ing from one clustering to the other. Note however that all these methods refer to the comparison of (different) clusterings extracted though from the same dataset and thus cannot be applied in the general case of different datasets [71].

Spatiotemporal clustering In spatiotemporal clustering, a cluster is a densification in a time-invariant trajectory.

Aggarwal[1] models clusters through kernel functions and changes as kernel density changes at each spatial location of the trajectory. The emphasis is on computing change velocity and finding the locations with the highest velocity–the epicenters. Three different types of change are considered in this work: i) data coagulation that corresponds to connected regions in the data which have velocity density larger than a user defined threshold, ii) data dissolution that correspond to connected regions whose velocity density is smaller than a user defined threshold and iii) data shift to other locations.

Yang et al[104] detect formation and dissipation events upon clusters of spatial scientific data. Their framework supports four types of spatial object association patterns (SOAP), namely Star, Clique, Sequence, and minLink, which are used to model different interactions among spatial objects. Such methods however, assume that the feature space does not change. Thus, they cannot be used for dynamic feature spaces, e.g.n text stream mining, where features are usually frequent words. Furthermore, hierarchical clustering algorithms cannot be coupled with such a method.

Kalnis et al[50] propose a special type of cluster change, the moving cluster, whose contents may change while its density function remains the same during its lifetime. They find moving clusters by tracing common data records between clusters of consecutive timepoints. Our MONICramework for transition detection is more general, since it encompasses several cluster transition types, allows for the ageing of old objects and does not require that the density function of a moving cluster is invariant.

Topic evolution Cluster change detection is also relevant to topic evolution in text streams, as dealt with in [66,,,61], where a topic is a cluster label, consisting of the dominant words inside the cluster.

In [66], Moringa and Yamanishi propose a topic analysis system that fulfills three relative tasks, namely i) topic structure identification that identifies what kind of main topics exist and how important they are, ii) topic emergence detection that detects the emergence of a new topic and iii) topic characterization that identifies the char-

acteristics of each main topic. All tasks are formalized in terms of a finite mixture model.

In[61], Mei and Zhai propose a method for discovering and summarizing the evolutionary patterns of themes in a text stream. The authors detect the themes at each period and use the KL divergence measure to find coherent themes over time, i.e.,hemes with similar labels. This way a topic evolution graph is built that can be used to trace theme transitions and to analyze the theme life cycles.

These methods are applicable whenever a human–understandable cluster label can be extracted and traced. Cluster labeling is not feasible for all applications though. For this reason, the proposed framework MONICetects cluster transitions rather than cluster label transitions.

Summarization methods The summarization of observed cluster transitions over a stream of data has not been adequately addressed in the literature, since research results on capturing cluster changes have only emerged in the last years. Relevant to our work is the research on the summarization of streams of data records (as opposed to streams of values) and on the identification, characterization and representation of pattern changes.

Summarization for a set of transactions with categorical attributes is studied by Chandola and Kumar[21]. In one of their methods, the derive summaries by clustering the transactions, extracting the feature/value pairs supported by all transactions in a cluster and using them as cluster summaries. They do not address the issue of cluster change upon a stream, but propose two metrics that characterize the output of the summarization algorithm, “compaction gain” and “information loss”. Quite naturally, our metrics are similarly motivated and have similar names. However, they summarize static data using clusters, while we summarize evolving clusters upon a stream.

Summarization and change are considered by Ipeirotis et al [45], who study changes of database content summaries. They define as “content summary” for a database a set of keywords, weighted on their importance within the database. Meta-search services use such summaries to select appropriate databases, towards which they issue keyword-based queries. The reliability of such a summary deteriorates as the contents of the database change over time. So, the authors propose methods to quantify and detect summary changes. This study addresses both the issue of summarization over the evolving database and the discovery of changes. However, the maintenance of the summaries themselves in a condensed form is beyond

the scope of their work. On the other hand, the proposed FINGERPRINTframework emphasizes on the summarization of the discovered population transitions.

The discovery and representation of cluster changes for text streams are studied by Mei and Zhai [61]. They apply soft clustering with mixture models at each time period, extract the representative keyword-list (the "theme") for each cluster, and then monitor the evolution of these lists by tracing divergences between a current keyword list and past ones. Theme transitions are maintained on a "theme evolution graph", which is then used to extract the life cycle of themes (by means of Hidden Markov Models).

Agrawal et al [2] propose the CluStream framework for clustering of evolving data streams. The clustering process is split into an online and an offline task: The online component stores summary statistics (the so-called *micro-clusters*) periodically and the offline component uses them for the formation of the actual clusters (the *macro-clusters*) over a user-specific time horizon. Micro-clusters can be observed as cluster summaries and are indeed designed to reduce space demand. Nonetheless, CluStream focuses on combining them into clusters rather than in summarizing them. Also, the information loss effected through summarization is not discussed.

Cluster summarization and evolution are addressed in CACTUS [31] and DEMON [33]. CACTUS clusters categorical data and derives cluster summaries, while DEMON is responsible for data evolution and monitoring in time, detecting and highlighting systematic data changes. With these two components, it is possible to study the evolution of data summaries, similarly to the evolution of text summaries studied in [61]. However, the modeling and maintenance of the observed changes are not studied in either component.

6.9 Summary

In this chapter we studied change detection and monitoring issues in a dynamic environment based on the the clustering models extracted from the environments through some DM algorithm. More specifically, we have proposed the MONICand MONIC+frameworks for change detection and monitoring in dynamic environmnets in terms of the extracted clustering models, the Evolution Graphmodel for the organization of the evolution history and the FINGERPRINTframework for the summarization of change in some condensed yet informative way.

The MONICframework is designed for arbitrary types of clusters.

It encompasses a cluster transition model, a transition detection algorithm and accompanying heuristics, operating upon a very generic model of clusterings over an accumulating dataset. We have applied MONICon a subarchive of the ACM library and have shown how cluster transitions give insights to the underlying data population and its shifts.

We have extended MONICinto the MONIC+framework which employs heuristics that exploit the particular characteristics of different cluster types, such as topological properties for clusters over a metric space and descriptors of data distribution for clusters defined as distributions. Our experiments show that our framework can provide useful insights on the evolution of the observing population and also, that our transition model and transition heuristics can reveal different forms of cluster evolution.

We have organized cluster changes in an Evolution Graphwhich models the whole history of population evolution. We have presented two exploitation capabilities over such a graph: querying of the evolution and studying of the stability of the population based on the lifetime of clusters and clusterings.

We have also studied the effective summarization of cluster changes over a stream of data records. We have proposed two algorithms for the summarization of the Evolution Graphinto FINGERPRINT a condensed representation in which less informative cluster transitions are suppressed. We derived functions that measure the effected information loss and compactness gain and we presented heuristics that drive the summarization process. We have implemented FINGERPRINTin two versions: The batch algorithm summarizes the Evolution Graphas a whole, while the online version creates the graphs fingerprint incrementally, during the process of cluster transition discovery. We have run experiments on three real datasets and have seen that the incremental algorithm achieves similar compactness gain as `BatchFINGERPRINT`, but the information loss may be much higher.

Part of this work appears in [90],[89],[91],[88],[71], whereas an extended version has been submitted in [72].

6.10 Open issues

A straightforward extension is to enrich MONIC(consequently, MONIC+) framework by incorporating more transition types and transition indicators.

Our experiments have shown that cluster monitoring can deliver

valuable insights on the evolution of the underlying data. The low passforward ratio at a specific timepoint can act as an alert for the end user and require careful inspection of the participant clusterings. So far, the overhead of this inspection is left to the user, however it would be useful to facilitate her/him by detecting, for example, the dimensions/features that are highly responsible for these transitions.

The evaluation of a framework like MONIC(or MONIC+) upon real data is a major challenge. Although datasets for the evaluation of data stream clustering algorithms emerge, a gold standard of data with clusters in transition is still missing. It is interesting for someone to focus on methods generating synthetic datasets for cluster transition detection and on criteria for the evaluation of cluster transition detectors upon them.

The experiment with K-means has shown that the transitions found by MONIC+can be used as alerts for cluster instability. This functionality should be further investigated.

Regarding the FINGERPRINTframework, so far we have concentrated on the summarization of cluster survivals. However, there are also other transitions that a cluster might encompass like absorption and split. It would be interesting to extend the summarization to include other cluster transitions as well, so that only the most informative changes are delivered to the end user.

Chapter 7

Conclusions and Outlook

In this chapter we summarize the contents of this thesis, and suggest future work arising from its findings.

7.1 Summary of contributions

Due to the wide application of Knowledge Discovery in Databases (KDD) and as a result of data flood that appears nowadays, the amount of patterns extracted from heterogeneous data sources (e.g., business, science, telecommunications, Web) is huge and, quite often, non-manageable by humans. Thus, there is a need for efficient pattern management including issues like modeling, storage, retrieval and querying of patterns.

An important issue that relates to several aspects of the pattern management problem, is that of dissimilarity assessment, i.e.,valuating how similar to each other two patterns are. Dissimilarity assessment comprises a complicated problem. First of all, there exist many different pattern types for which we should define dissimilarity measures. Furthermore, there exist patterns defined over raw data, the so called simple patterns as well as patterns defined over other patterns, the so called complex patterns. Moreover, patterns preserve the semantics of the raw data from which they have been extracted, however the degree of preservation depends on the mining parameters. All these parameters comprise the dissimilarity problem a difficult yet challenging problem.

We rely on the information contained in each pattern in order to evaluate the dissimilarity between two patterns. Such an information is either *intensional*, i.e., description of the concept represented by the pattern, or *extensional*, i.e.,n enumeration of the data members participating in the pattern formation.

Through this thesis, we dealt with different aspects of the pattern dissimilarity assessment problem, in particular:

- We proposed the PANDAframework for comparing both *simple and complex patterns*, defined over raw data and over other patterns, respectively. In PANDA, the problem of dissimilarity assessment between complex patterns is decomposed to the problem of comparing their component simple patterns. Thus, patterns of any complexity can be easily handled as far as they can be expressed in the simple-complex rationale of PANDA.

- We investigated the *dissimilarity reasoning problem*, i.e.,hether dissimilarity between two pattern sets can serve as a measure of dissimilarity between their corresponding datasets. In particular, we focused on the case of frequent itemset patterns and studied the effect of the *minSuppport* threshold and of the itemset lattice representation, namely frequent itemsets, closed frequent itemsets and maximal frequent itemsets, on such an association. Our theoretical and experimental findings suggest that such a correspondence is subjective to the mining parameters used for the extraction of patterns.

- We exploited the information provided by the decision tree patterns in order to compare either decision trees or classification datasets. In particular, our approach exploits the information provided by the partitioning that each decision tree performs over the attribute space of the dataset from which it has been induced. We demonstrated the usefulness and applicability of our framework in estimating the semantic similarity between decision tree models induced from different subsamples of classification datasets.

- Finally, we compared clusters and clustering and showed another application of similarity assessment, that of cluster monitoring. We studied cluster monitoring as a means of monitoring the evolution of a dynamic population across the time axis. Towards this aim we proposed the MONICnd MONIC+rameworks for the categorization and detection of cluster transitions. We organized clusters and their transitions into a graph structure, called EVOLUTION GRAPH, that contains the whole history of the population evolution. Since, as the observation period increases this graph becomes huge and thus un-manageable by the end user, we proposed the FINGERPRINTramework for an effective summarization of these transitions. Together, MONIC MONIC+nd FINGERPRINTrameworks provide the necessary

infrastructure for monitoring and maintaining changes in a dynamic environment.

7.2 Discussion on future work

Several research issues arise from this thesis, as outlined below.

- Our PANDAframework for the comparison of arbitrary complex patterns provides the necessary infrastructure for dissimilarity assessment by instantiating the appropriate building blocks for the specific case at hand. However, such an instantiation is not an easy task. In general, discovering the best dissimilarity function for a specific problem is not an easy task and is actually subjective. Thus, it would be useful and at the same time challenging to discover best configurations of PANDAfor specific dissimilarity assessment problems. One solution towards this direction is to incorporate in PANDAalready proposed in the literature best solutions for particular cases.
- In Chapter 4 we saw the dependency of dissimilarity on the mining parameters and we conclude that associating dissimilarity in pattern space with the dissimilarity in the corresponding raw data space is not so straightforward; rather, it depends on the mining parameters. Thus, an open issue is the "discovery" of a dissimilarity measure that would be more robust to the mining parameters and would better preserve the original raw data space characteristics in the pattern space.
- In Chapter 6 we show the application of pattern dissimilarity measures on monitoring, for the case of cluster and clustering patterns. An open direction is monitoring in terms of other pattern types like frequent itemsets and decision tree models.
- The availability of dissimilarity measures between patterns, allows us to perform a number of standard data mining tasks, not anymore over raw data, but rather over the patterns extracted from these data, i.e.,eta-mining tasks. As an example, let us consider clustering of decision tree models in order to discover groups of decision trees with similar characteristics. It would be interesting to apply our dissimilarity measures for such tasks.
- The methods proposed in this thesis refer to the comparison of patterns of the same pattern type. It is interesting to investigate how dissimilarity between patterns of different pattern

types can be evaluated. A straightforward solution towards this direction is to convert one type into the other and perform the comparison over this (common) type. More elegant solutions however, could be investigated. In general, finding a type independent dissimilarity assessment schema between patterns is useful from both research and application purposes.

Index

Bibliography

[1] C.C. Aggarwal. On change diagnosis in evolving data streams. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 17(5):587–600, 2005.

[2] C.C. Aggarwal, J.Han, J.Wang, and P.Yu. A framework for clustering evolving data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 81–92. VLDB Endowment, 2003.

[3] C.C. Aggarwal and P.S. Yu. A framework for clustering massive text and categorical data streams. In *SIAM International Conference on Data Mining (SDM)*. SIAM, 2006.

[4] R.Agrawal, J.Gehrke, D.Gunopulos, and P.Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Records*, 27(2):94–105, 1998.

[5] R.Agrawal, T.Imielinksi, and A.Swami. Mining association rules between sets of items ins large databases. In *ACM SIGMOD International Conference on Management of Data(SIGMOD)*, pages 207–216. ACM, 1993.

[6] R.Agrawal, M.Mehta, J.C. Shafer, R.Srikant, A.Arning, and T.Bollinger. The Quest data mining system. In *ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 244–249. AAAI Press, 1996.

[7] J.Allan, editor. *Topic Detection and Tracking: Event-based Information Organization*. Kluwer Academic Publishers, 2002.

[8] S.Baron, M.Spiliopoulou, and O.Günther. Efficient monitoring of patterns in data mining environments. In *East-European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 253–265. Springer, 2003.

[9] I.Bartolini, P.Ciaccia, I.Ntoutsi, M.Patella, and Y.Theodoridis. A unified and flexible framework for

comparing simple and complex patterns. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 496–499. Springer-Verlag, 2004.

[10] I.Bartolini, P.Ciaccia, I.Ntoutsi, M.Patella, and Y.Theodoridis. The PANDA framework for comparing arbitrary complex patterns. Submitted, May 2008.

[11] B.Bartsch-Spörl, M.Lenz, and A.Hübner. Case-based reasoning: Survey and future directions. In *Biannual German Conference on Knowledge-Based Systems*, pages 67–89. Springer-Verlag, 1999.

[12] M.A. Bender and M.Farach-Colton. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer-Verlag, 2000.

[13] R.Bergmann and A.Stahl. Similarity measures for object-oriented case representations. In *European Workshop on Advances in Case-Based Reasoning (EWCBR)*, pages 25–36. Springer-Verlag, 1998.

[14] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases. http://www.ics.uci.edu/mlearn/MLRepository.html (valid as of May 2008).

[15] C.Borgelt and A.Nürnberger. Experiments in Document Clustering using Cluster Specific Term Weights. In *Workshop Machine Learning and Interaction for Text-based Information Retrieval (TIR)*, pages 55–68, 2004.

[16] L.Breiman, J.Friedman, R.Olshen, and C.Stone. *Classification and Regression Trees*. Wadsworth International, 1984.

[17] S.Brin and L.Page. The anatomy of a large-scale hypertextual Web search engine. volume30, pages 107–117. Elsevier Science Publishers B. V., 1998.

[18] D.Burdick, M.Calimlim, and J.Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *International Conference on Data Engineering (ICDE)*, pages 443–452. IEEE Computer Society, 2001.

[19] B.Catania, A.Maddalena, and M.Mazza. Psycho: A prototype system for pattern management. In *International Conference on Very Large Data Bases (VLDB)*, pages 1346–1349. VLDB Endowment, 2005.

[20] B.Catania, A.Maffalena, A.Mazza, E.Bertino, and S.Rizzi. A framework for data mining pattern management. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 87–98. Springer-Verlag New York, Inc., 2004.

[21] V.Chandola and V.Kumar. Summarization - compressing data into an informative representation. In *IEEE International Conference on Data Mining (ICDM)*, pages 98–105, Houston, Texas, USA, 2005. IEEE Computer Society.

[22] CWM. Common Warehouse Metamodel (CWM). http://www.omg.org/cwm, (valid as of May 2008).

[23] G.Das and D.Gunopulos. *Time Series Similarity and Indexing*, chapter11, pages 279–302. Handbook of Massive Data Sets. Kluwer Academic Publishers, 2002.

[24] DIAsDEM. DIAsDEM. http://sourceforge.net/projects/hypknowsys, (valid as of May 2008).

[25] B.Díaz-Agudo and P.A. González-Calero. A declarative similarity framework for knowledge intensive cbr. In *International Conference on Case-Based Reasoning (ICCBR)*, pages 158–172. Springer-Verlag, 2001.

[26] DMG. Predictive Model Markup Language (PMML). http://www.dmg.org/pmml-v3-0.html., (valid as of May 2008).

[27] M.Ester, H.-P. Kriegel, J.Sander, and X.Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[28] F.Farnstrom, J.Lewis, and C.Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51–57, 2000.

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

[30] FIMI. Frequent itemsets mining data set repository. http://fimi.cs.helsinki.fi/data/, (valid as of May 2008).

[31] V.Ganti, J.Gehrke, and R.Ramakrishnan. CACTUS: Clustering categorical data using summaries. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 73–83. ACM, 1999.

[32] V.Ganti, J.Gehrke, and R.Ramakrishnan. A framework for measuring changes in data characteristics. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 126–137. ACM Press, 1999.

[33] V.Ganti, J.Gehrke, and R.Ramakrishnan. Demon: Mining and monitoring evolving data. In *International Conference on Data Engineering (ICDE)*, pages 439–448. IEEE Computer Society, 2000.

[34] T.Gärtner. A survey of kernels for structured data. *SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

[35] T.Gärtner, J.W. Lloyd, and P.A. Flach. Kernels for structured data. In *12the International Conference on Inductive Logic Programming (ILP)*, pages 66–83, 2002.

[36] K.Gouda and M.Zaki. Efficiently mining maximal frequent itemsets. In *IEEE International Conference on Data Mining (ICDM)*, pages 163–170. IEEE Computer Society, 2001.

[37] J.Han, H.Cheng, D.Xin, and X.Yan. Frequent pattern mining: current status and future directions. *Data Mining Knowledge Discovery (DMKD)*, 15(1):55–86, 2007.

[38] J.Han, Y.Fu, K.Koperski, W.Wang, and O.Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 1996.

[39] J.Han and M.Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., 2000.

[40] D.Haussler. Convolution kernels on discrete structures, ucsc-crl-99-10. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[41] J.Hipp, U.Guntzer, and G.Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *ACM SIGKDD Explorations*, 2(1):58–64, 2000.

[42] IBM. IBM DB2 Intelligent Miner. http://www-306.ibm.com/software/data/iminer, (valid as of May 2008).

[43] T.Imielinski and H.Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[44] T.Imielinski and A.Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.

[45] P.G. Ipeirotis, A.Ntoulas, J.Cho, and L.Gravano. Modeling and managing content changes in text databases. In *International Conference on Data Engineering (ICDE)*, pages 606 – 617. IEEE Computer Society, 2005.

[46] A.Jain, R.Duin, and J.Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[47] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computer Surveys*, 31:264–323, 1999.

[48] JDMAPI. Java Data Mining API. http://www.jcp.org/jsr/detail/73.prt, (valid as of May 2008).

[49] T.Johnson, L.Lashmanan, and T.Ravmond. The 3W Model and Algebra for Unified Data Mining. In *International Conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 2000.

[50] P.Kalnis, N.Mamoulis, and S.Bakiras. On discovering moving clusters in spatio-temporal data. In *International Symposium on Spatial and Temporal Databases (SSTD)*, pages 364–381. Springer, 2005.

[51] G.Karypis. CLUTO. http://glaros.dtc.umn.edu/gkhome/views/cluto/, (valid as of May 2008).

[52] E.Kotsifakos, G.Marketos, and Y.Theodoridis. A framework for integrating ontologies and pattern-bases. In H.Nigro, S.Cisaro, and D.Xodo, editors, *Data Mining with Ontologies: Implementations, Findings, and Frameworks*. Information Science Reference, 2008.

[53] E.Kotsifakos, I.Ntoutsi, and Y.Theodoridis. Database support for data mining patterns. In *Panhellenic Conference on Informatics (PCI)*. Springer-Verlag, 2005.

[54] E.Kotsifakos, I.Ntoutsi, Y.Vrahoritis, and Y.Theodoridis. Monitoring patterns through an integrated management and mining tool. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*. Springer-Verlag, 2008.

[55] E.Kotsifakos, I.Ntoutsi, Y.Vrahoritis, and Y.Theodoridis. Pattern-miner: Integrated management and mining over data mining models. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM press, 2008.

[56] H.Kuhn. Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[57] E.Levina and P.Bickel. The Earth Mover's Distance is the Mallow's distance: Some insights from statistics. In *International Conference on Computer Vision (ICCV)*, pages 251–256, 2001.

[58] M.Ley. DBLP The digital bibliography and library project. http://www.informatik.uni-trier.de/ley/db/ (valid as of May 2008).

[59] T.Li, M.Ogihara, and S.Zhu. Association-based similarity testing and its applications. *Intelligent Data Analysis*, 7:209–232, 2003.

[60] P.Lyman and H.R. Varian. How Much Information? http://www2.sims.berkeley.edu/research/projects/how-much-info/, 2003 (valid as of May 2008).

[61] Q.Mei and C.Zhai. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 198–207. ACM, 2005.

[62] M.Meila. Comparing clusterings. Technical report, Department of Statistics, University of Washington, 2002.

[63] R.Meo, G.Psaila, and S.Ceri. A New SQL–like Operator for Mining Association Rules. In *International Conference on Very Large Data Bases (VLDB)*, pages 122–133. VLDB Endowment, 1996.

[64] T.Mielikainen. On inverse frequent set mining. In *Workshop on Privacy Preserving Data Mining (PPDM)*, pages 18–23, 2003.

[65] T.Mitchell. *Machine Learning*. Kluwer Academic Publishers, 1997.

[66] S.Morinaga and K.Yamanishi. Tracking dynamics of topic trends using a finite mixture model. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 811–816, 2004.

[67] MSSQL. Microsoft SQL Server. http://www.microsoft.com/sql/2005/, (valid as of May 2008).

[68] O.Nasraoui, C.Cardona-Uribe, and C.Rojas-Coronel. Tecnostreams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *IEEE International Conference on Data Mining (ICDM)*, page 235, 2003.

[69] I.Ntoutsi. The notion of similarity in data and pattern space. In *Workshop on Pattern Representation and Management (ParMa) in conjunction with the Int. Conference on Extending Database Technology (EDBT)*. CEUR-WS.org, 2004.

[70] I.Ntoutsi, A.Kalousis, and Y.Theodoridis. A general framework for estimating similarity of datasets and decision trees: exploring semantic similarity of decision trees. In *SIAM International Conference on Data Mining (SDM)*. SIAM, 2008.

[71] I.Ntoutsi, N.Pelekis, and Y.Theodoridis. Pattern Comparison in Data Mining: a survey. In D.Taniar, editor, *Research and Trends in Data Mining Technologies and Applications (Advances in Data Warehousing and Mining)*, pages 86 – 120. Idea Group Publishing, 2007.

[72] I.Ntoutsi, M.Spiliopoulou, and Y.Theodoridis. Tracing cluster transitions for different cluster types. February Submitted.

[73] I.Ntoutsi and Y.Theodoridis. Current issues in modeling data mining processes and results. In *PANDA Workshop on Pattern-Base Management Systems*, 2003.

[74] I.Ntoutsi and Y.Theodoridis. Measuring and evaluating dissimilarity in data and pattern spaces. In *International Conference on Very Large Data Bases (VLDB) Phd Workshop*, 2005.

[75] I.Ntoutsi and Y.Theodoridis. The notion of patterns in data mining. Technical Report TR-2007-02, Information Systems

Laboratory, Department of Informatics, University of Piraeus, Greece, 2007.

[76] I.Ntoutsi and Y.Theodoridis. Pattern Management. Technical Report TR-2007-01, Information Systems Laboratory, Department of Informatics, University of Piraeus, Greece, 2007.

[77] I.Ntoutsi and Y.Theodoridis. Comparing datasets using frequent itemsets: Dependency on the mining parameters. In *Hellenic Conference on Artificial Intelligence (SETN)*. Springer-Verlag, 2008.

[78] ORACLE. Oracle 10g Data Mining Concepts. http://download-uk.oracle.com/docs/cd/B19306-01/datamine.102/b14339/toc.htm, (valid as of May 2008).

[79] PANDA. The PANDA Project. http://dke.cti.gr/PANDA/, (valid as of May 2008).

[80] PANDA. PAtterns for Next generation DAtabase systems - IST project 2001–2003, (valid as of May 2008).

[81] S.Parthasarathy and M.Ogihara. Clustering distributed homogeneous datasets. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 566–574. Springer, 2000.

[82] I.Pekerskaya, J.Pei, and K.Wang. Mining changing regions from access-constrained snapshots: A cluster-embedded decision tree approach. *Intelligent Information Systems (Special Issue on Mining Spatio-Temporal Data)*, 27(3):215–242, 2006.

[83] riskglossary.com. Kurtosis. http://www.riskglossary.com/link/kurtosis.htm, (valid as of May 2008).

[84] riskglossary.com. skewness. http://www.riskglossary.com/link/skewness.htm, (valid as of May 2008).

[85] S.Rizzi, E.Bertino, B.Catania, M.Golfarelli, M.Halkidi, M.Terrovitis, P.Vassiliadis, M.Vazirgiannis, and E.Vrachnos. Towards a Logical Model for Patterns. In *ER*, pages 77–90, 2003.

[86] Y.Rubner, C.Tomasi, and L.Guibas. A metric for distributions with applications to image databases. In *IEEE International Conference on Computer Vision*, pages 56–66. IEEE Computer Society, 1998.

[87] J.Spiegel and N.Polyzotis. Graph-based synopses for relational selectivity estimation. In *ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 205–216, 2006.

[88] M.Spiliopoulou, I.Ntoutsi, and Y.Theodoridis. Tracing cluster transitions for different cluster types. Technical Report TR-2007-03, Information Systems Laboratory, Department of Informatics, University of Piraeus, Greece, 2007.

[89] M.Spiliopoulou, I.Ntoutsi, Y.Theodoridis, and R.Schult. The MONIC framework for cluster transition detection. In *Hellenic Data Management Symposium (HDMS)*, 2006.

[90] M.Spiliopoulou, I.Ntoutsi, Y.Theodoridis, and R.Schult. MONIC: modeling and monitoring cluster transitions. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 706–711. ACM Press, 2006.

[91] M.Spiliopoulou, I.Ntoutsi, Y.Theodoridis, and R.Schult. Tracing cluster transitions for different cluster types. In *Workshop on Data Mining and Knowledge Discovery (ADMKD) in conjunction with the East-European Conference on Advances in Databases and Information Systems (ADBIS)*, 2007.

[92] M.Spiliopoulou, Y.Theodoridis, and I.Ntoutsi. Mining the Volatile Web - tutorial. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*. Springer-Verlag, 2005.

[93] SQL/MM. ISO SQL/MM Part 6. http://www.sql-99.org/SC32/WG4/Progression-Documents/FCD/fcd-datamining-2001-05.pdf, (valid as of May 2008).

[94] Y.Theodoridis, M.Vazirgiannis, P.Vassiliadis, B.Catania, and S.Rizzi. A Manifesto for Pattern Bases. PANDA TR-2003-03. Technical report, Research Academic Computer Technology Institute (RACTI), 2003.

[95] P.Turney. Technical note: Bias and the quantification of stability. *Machine Learning*, 20:23–33, 1995.

[96] J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):83–97, 1976.

[97] H.Wang and J.Pei. A random method for quantifying changing distributions in data streams. In *European Conference on*

Principles and Practice of Knowledge Discovery in Databases (PKDD), pages 684–691. Springer, 2005.

[98] W.Wang, J.Yang, and R.R. Muntz. STING: A statistical information grid approach to spatial data mining. In *International Conference on Very Large Data Bases (VLDB)*, pages 186–195. Morgan Kaufmann Publishers Inc., 1997.

[99] Wikipedia. WordNet, Retrieved from http://en.wikipedia.org/wiki/WordNet (valid as on May 2008).

[100] I.H. Witten and E.Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers Inc., 2000.

[101] I.H. Witten and E.Frank. *Data Mining: Practical Machine Learning tools and techniques*. Morgan Kaufmann, 2005.

[102] D.Xin, J.Han, X.Yan, and H.Cheng. Mining compressed frequent-pattern sets. In *International Conference on Very Large Data Bases (VLDB)*, pages 709–720. VLDB Endowment, 2005.

[103] Yahoo. Italian MIB stock. http://it.finance.yahoo.com/, (valid as of May 2008).

[104] H.Yang, S.Parthasarathy, and S.Mehta. A generalized framework for mining spatio-temporal patterns in scientific data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 716–721. ACM Press, 2005.

[105] M.Zaki and C.-J. Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(4):462–478, 2005.

[106] K.Zhang and D.Shasha. Fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computation*, 18(6):1245–1262, 1989.

[107] D.Zhou, J.Li, and H.Zha. A new Mallows distance based metric for comparing clusterings. In *International Conference on Machine Learning (ICML)*, pages 1028–1035. ACM, 2005.