Difference Analysis in Big Data : Exploration, Explanation, Evolution

Sofia Kleisarchaki

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

préparée dans le cadre d'une cotutelle entre **l'Université de Grenoble** et **l'Université de Créte**

Spécialité : **Informatique**

Arrêté ministérial : 7 aout 2006

Présentée par

Sofia Kleisarchaki

Thèse dirigée par **Sihem Amer-Yahia et Vassilis Christophides**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Analyse des Différences dans le Big Data: Exploration, Explication, Évolution

Thèse soutenue publiquement le **28 Novembre 2016**,
devant le jury composé de :

Mme, Claudia Roncancio
Professor, Grenoble INP, Président
Mr, Albert Bifet
Associate Professor, Telecom ParisTech, Rapporteur
Mr, Ingmar Weber
Principal Scientist, Qatar Computing Research Institute, Rapporteur
Mme, Angela Bonifati
Professor, Lyon 1 University, Examinateur
Mme, Anne Laurent
Professor, Montpellier 2 University, Examinateur
Mr, Ioannis Tsamardinos
Professor, University of Crete, Examinatrice
Mme, Sihem Amer-Yahia
Research Director, CNRS delegation Alpes, Directeur de thèse
Mr, Vassilis Christophides
Professor, University of Crete, Directeur de thèse

Difference Analysis in Big Data: Exploration, Explanation, Evolution

Sofia Kleisarchaki

University of Grenoble Alps & University of Crete

This dissertation is submitted for the degree of
Doctor of Philosophy

December 2016

I would like to dedicate this thesis to my precious treasures...
One given by life; my beloved Andreas and one given by my parents; my sister Foteini.

Résumé

La Variabilité dans le Big Data se réfère aux données dont la signification change de manière continue. Par exemple, les données des plateformes sociales et les données des applications de surveillance, présentent une grande variabilité. Cette variabilité est dûe aux différences dans la distribution de données sous-jacente comme l'opinion de populations d'utilisateurs ou les mesures des réseaux d'ordinateurs, etc. L'Analyse de Différences a comme objectif l'étude de la variabilité des Données Massives. Afin de réaliser cet objectif, les data scientists ont besoin (a) de mesures de comparaison de données pour différentes dimensions telles que l'âge pour les utilisateurs et le sujet pour le traffic réseau, et (b) d'algorithmes efficaces pour la détection de différences à grande échelle. Dans cette thèse, nous identifions et étudions trois nouvelles tâches analytiques : *L'Exploration des Différences, l'Explication des Différences* et *l'Evolution des Différences*.

L'Exploration des Différences s'attaque à l'extraction de l'opinion de différents segments d'utilisateurs (ex., sur un site de films). Nous proposons des mesures adaptées à la comparaison de distributions de notes attribuées par les utilisateurs, et des algorithmes efficaces qui permettent, à partir d'une opinion donnée, de trouver les segments qui sont d'accord ou pas avec cette opinion. L'Explication des Différences s'intéresse à fournir une explication succinte de la différence entre deux ensembles de données (ex., les habitudes d'achat de deux ensembles de clients). Nous proposons des fonctions de scoring permettant d'ordonner les explications, et des algorithmes qui guarantissent de fournir des explications à la fois concises et informatives. Enfin, l'Evolution des Différences suit l'évolution d'un ensemble de données dans le temps et résume cette évolution à différentes granularités de temps. Nous proposons une approche basée sur le requêtage qui utilise des mesures de similarité pour comparer des clusters consécutifs dans le temps. Nos index et algorithmes pour l'Evolution des Différences sont capables de traiter des données qui arrivent à différentes vitesses et des types de changements différents (ex., soudains, incrémentaux). L'utilité et le passage à l'échelle de tous nos algorithmes reposent sur l'exploitation de la hiérarchie dans les données (ex., temporelle, démographique).

Afin de valider l'utilité de nos tâches analytiques et le passage à l'échelle de nos algorithmes, nous réalisons un grand nombre d'expériences aussi bien sur des données synthétiques que réelles.

Nous montrons que l'Exploration des Différences guide les data scientists ainsi que les novices à découvrir l'opinion de plusieurs segments d'internautes à grande échelle. L'Explication des Différences révèle la nécessité de résumer les différences entre deux ensembles de donnes, de manière parcimonieuse et montre que la parcimonie peut être atteinte en exploitant les relations hiérarchiques dans les données. Enfin, notre étude sur l'Evolution des Différences fournit des preuves solides qu'une approche basée sur les requêtes est très adaptée à capturer des taux d'arrivée des données variés à plusieurs granularités de temps. De même, nous montrons que les approches de clustering sont adaptées à différents types de changement.

Abstract

Variability in Big Data refers to data whose meaning changes continuously. For instance, data derived from social platforms and from monitoring applications, exhibits great variability. This variability is essentially the result of changes in the underlying data distributions of attributes of interest, such as user opinions/ratings, computer network measurements, etc. *Difference Analysis* aims to study variability in Big Data. To achieve that goal, data scientists need: (a) measures to compare data in various dimensions such as age for users or topic for network traffic, and (b) efficient algorithms to detect changes in massive data. In this thesis, we identify and study three novel analytical tasks to capture data variability: *Difference Exploration, Difference Explanation* and *Difference Evolution*.

Difference Exploration is concerned with extracting the opinion of different user segments (e.g., on a movie rating website). We propose appropriate measures for comparing user opinions in the form of rating distributions, and efficient algorithms that, given an opinion of interest in the form of a rating histogram, discover agreeing and disagreeing populations. Difference Explanation tackles the question of providing a succinct explanation of differences between two datasets of interest (e.g., buying habits of two sets of customers). We propose scoring functions designed to rank explanations, and algorithms that guarantee explanation conciseness and informativeness. Finally, Difference Evolution tracks change in an input dataset over time and summarizes change at multiple time granularities. We propose a query-based approach that uses similarity measures to compare consecutive clusters over time. Our indexes and algorithms for Difference Evolution are designed to capture different data arrival rates (e.g., low, high) and different types of change (e.g., sudden, incremental). The utility and scalability of all our algorithms relies on hierarchies inherent in data (e.g., time, demographic).

We run extensive experiments on real and synthetic datasets to validate the usefulness of the three analytical tasks and the scalability of our algorithms. We show that Difference Exploration guides end-users and data scientists in uncovering the opinion of different user segments in a scalable way. Difference Explanation reveals the need to parsimoniously summarize differences between two datasets and shows that parsimony can be achieved by exploiting hierarchy in data. Finally, our study on Difference Evolution provides strong

evidence that a query-based approach is well-suited to tracking change in datasets with varying arrival rates and at multiple time granularities. Similarly, we show that different clustering approaches can be used to capture different types of change.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Context: Big Data

While the concept of ”Big Data” has been around for many years, its term was first articulated in the early 2000s. The term of Big Data refers to datasets that are so large or complex that traditional applications are inadequate to deal with them and thus devising new scalable structures and advanced methods is required. The emergence of social platforms decisively contributed to the generation of Big Data that are dynamic in nature, in terms of their volume, arrival rate and type. Beyond social platforms, several other means are automatically generating such content. Sales transactions, network traffic and news feed subscriptions are few examples of data deriving from real world monitoring applications.

The characteristics of big data are usually summarized by a 3-dimensional model, namely the ”3Vs”: the increasing *Volume* (amount of data), the unpredicted *Velocity* (speed of data) and the high *Variety* (range of data types and sources). Although the ”3Vs” give us an insight into the scale of data, it is only scratching the surface of the depth and criticality of Big Data. To this end, an augmented model has been proposed of four more “Vs”: *Validity* (correctness and accuracy of data), *Veracity* (noise and abnormality of data), *Volatility* (duration of validity) and *Variability* (continuously changing meaning of data).

In this thesis, we are particularly interested in *variability* of data. *Variability* refers to data whose meaning is continuously changing. It is usually confused with *variety*, but an intuitive example can easily distinguish them. Consider a coffee shop with five different blends of coffee (i.e., that is variety) but the same blend tastes different every day (i.e., that is variability). To express and summarize variability different dimensions (i.e., attributes) are considered. For instance, time and space attributes, but also user demographics (e.g., age, gender) or item attributes (e.g., movie title, actors) are used, depending on the context. In order to perform variability analysis of data, we are interested in (a) *exploring differences*

over different attributes, (b) *explaining* such differences according to some attributes of interest and (c) *tracking the evolution* of data and summarizing differences over time. We refer to those three analytical tasks as *Difference Analysis*.

There are several examples of analyzing the variability in data, which have attracted the attention of both scientists and market analysts over time. In this thesis, we introduce and focus on three analytical tasks derived from the aforementioned monitoring applications, indicating their potential impact in science and marketing. *Exploring* the opinions in movies of different rater populations can result in targeted population-oriented (e.g., geographic-, age-oriented) or item-oriented (e.g., movies of an actor) campaigns. *Explaining* the differences in consumption habits of various demographic groups across different stores can change the marketing strategies of different stores. *Monitoring the evolution* of users' preferences in news subscriptions can improve recommendation systems, and monitoring intrusions can improve network security.

Each one of the above tasks on analyzing differences pushes the borders of knowledge in the wider area of Difference Analysis. For each one of the analytical tasks, we encounter particular challenges and we envision novel algorithmic solutions.

1.2 Our Approach for Difference Analysis

In this thesis, we deal with three different analytical tasks studying the variability in data. Figure 1.1 depicts an abstract overview of the tasks, namely *Difference Exploration*, *Difference Explanation* and *Difference Evolution*.

A common denominator of all three tasks is (a) the detection of differences and (b) the exploitation of hierarchies (i.e., formed by attributes) inherent in data. First, difference detection is performed by contrasting and comparing items of interest (shown as input in Figure 1.1) in order to extract the attributes over which these items differ (shown as output in Figure 1.1). For the purpose of performing such a comparison, three objectives are significant: finding the *attributes* over which the two items will be contrasted (e.g., demographic), the *measure of interest* that summarizes them (e.g., rating distribution, number of purchases) and the *extent of their difference* given by a distance measure (e.g., difference in rating distributions, absolute difference in number of purchases). Second, exploiting hierarchies is performed by decomposing input data at different dimensions (i.e., attributes of hierarchy) and detecting those dimensions over which they differ. For the purpose of decomposition, time, demographic and item hierarchies are exploited, allowing multiple summarizations of input data.

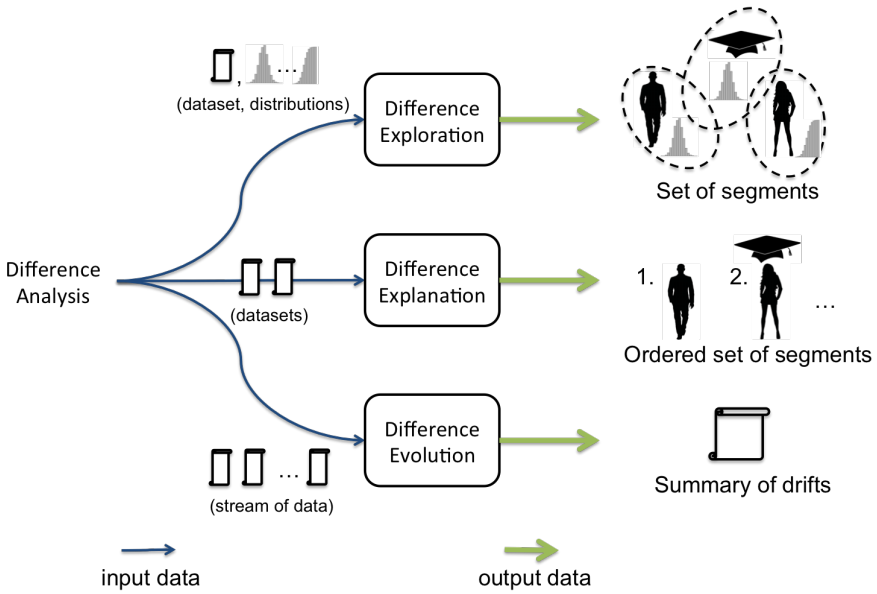


Fig. 1.1 Our approach for difference analysis: Exploration, Explanation, Evolution

In the following subsections, we give an overview of the three tasks, we summarize the encountered challenges and we briefly discuss how related work deals with the aforementioned objectives. Although other analytical tasks, such as *change prediction* [ZCB08], *user recommendations based on dynamic changes* [ACALAZ+15], would also be of interest they are beyond the scope of this thesis.

1.2.1 Difference Exploration

Motivation. We define *Difference Exploration* as the act of exploring differences in opinions of various population segments. Difference exploration admits as input a set of opinions and a dataset of interest and extracts a set of population segments agreeing (or disagreeing) with these opinions. It addresses questions such as *Which social groups oppose Donald Trump? Which user populations agree with the movies I like?* The aforementioned questions are only few examples of how opinions for the same topic may vary across different population segments and indicate the need for a further exploration.

Challenges. The problem of exploring the opinions of various population segments raises two main challenges. The first is *the choice of an appropriate distance measure for comparing an input opinion with the opinions of various population segments*. In this thesis, we define an opinion as a rating distribution, i.e. a histogram summarizing the ratings of an item, and thus a well-adapted measure should be able to compare such distributions. The second

challenge concerns *the design of efficient algorithms that dynamically detect population segments among all available ones*, which are in (dis-) agreement with the input opinions. The space of possible segments is exponential in the number of possible attribute values which renders this task challenging.

State-of-the-art. In the field of extracting meaningful demographic patterns some steps have been taken in exploring different population segments. For instance, authors in [DAyDY11] introduce a method for mining rated datasets in order to discover users with good, bad or polarized opinions about a topic (e.g., movie). Similarly, subgroup detection [DGD12] is concerned with finding agreeing or disagreeing groups by analyzing their discussions on online forums. Although these works succeed in exploring hidden demographic patterns, they exhibit two deficiencies. First, they lay their foundation on comparing rating average values. However, the use of averages can be confounding as a group might not contain a difference while its finer subgroups do. In that case, differences of finer groups are missed. Second, the analysis is bounded in discovering a limited number of opinions; groups either have a good (resp., agree), a bad (resp., disagree) or a polarized opinion.

1.2.2 Difference Explanation

Motivation. We define *Difference Explanation* as the act of constructing a parsimonious set of explanations for a detected difference. Difference explanation admits as input two datasets that we wish to contrast, and extracts a set of population segments that best explain their differences. It serves to address questions such as *Why does the number of sales of two stores significantly differ, although they have the same number of customers?* or *Which demographic groups have different consumption habits in two different stores?* These are common questions from market analysts, indicating how preferences of the same demographic group may vary across different stores.

Challenges. The problem of explaining differences encounters two main challenges. The first is *devising a scoring function which is able to sort the explanations, based on their ability to parsimoniously summarize differences, while at the same time describe as many differences as possible*. The second is to *design an algorithm utilizing the scoring function which guarantees near-optimal explanations, in terms of conciseness in the number of explanations and informativeness of the reported explanations*.

State-of-the-art. In the spirit of providing explanations of differences some effort has been done [ABG$^+$07, JBL09]. However, these works still lack the ability of minimizing

the number of explanations (i.e., conciseness) while maximizing the reported differences (i.e., informativeness). Either they restrict their work in providing explanations in a concise, a-priori given hierarchical structure by sacrificing information not included in the hierarchy [ABG+07] or they focus on being informative even if they produce some bits of redundancy [JBL09]. Moreover, these approaches do not fully exploit the structural properties of attributes, either because they are not capable of exploring multi-dimensional data segments [ABG+07] or because they ignore several inclusion relations existing in conjunctions of data segments [JBL09].

1.2.3 Difference Evolution

Motivation. We define *Difference Evolution* as the act of tracking differences at multiple time granularities (e.g., daily, weekly). Difference evolution admits as input a stream of data and extracts a number of change points (i.e., moments in time) at different time granularities where a difference is detected. It addresses questions such as *When did the preferences of users in news subscription change (week granularity)? When did an intrusion take place in a network (hour granularity)?* The previous questions investigate how a particular item (e.g., user preference, network traffic) evolves over time and for different time granularities.

Challenges. The problem of tracking the evolution of data exhibits two main challenges. The first concerns *the ability to detect differences, named drifts, arriving at an unpredicted arrival rate*. For this purpose, it is essential to explore different time granularities avoiding to make any assumption on the underlying distribution. Exploring different granularities is challenging as it requires an appropriate segmentation of the input stream in order not to detect very subtle drifts (low precision) but also not miss them (low recall). The second challenge deals with the ability to detect drifts of different types (e.g., sudden, incremental).

State-of-the-art. Several methods of *drift detection* have been proposed [BGP10, Ozo08, JMG95, KBDG04, VB09] segmenting the input stream and processing it into fixed time intervals. Although these approaches leverage the time dimension and use robust statistical comparisons, they suffer from the problem of adaptability to drift detection: they cannot dynamically adapt to different arrival rates (i.e., low or high) and types (i.e., sudden or incremental) of drifts. Although some effort is made in [WK96, Bif10, GMCR04] to address arrival rates and types, these works still exhibit the deficiency of querying differences over historical data and analyzing their precision and recall at different time granularities.

1.3 Contributions & Thesis Organization

1. **Difference Exploration.** In Chapter 2, we study difference exploration in the context of collaborative rating systems, such as *MovieLens* for rating movies and *BookCrossing* for rating books. Given a set of input opinions, in the form of a rating distribution, we are interested in finding population segments with significantly converging or diverging opinion from the input ones. [1] We make the following contributions:

   (a) We formalize our problem as building *rating maps* on demand that, not only facilitate, but also guide the exploration of opinions among different sub-populations. A *rating map* is a set of pairs of the form *(population segment, rating distribution)* that are dynamically built given desired input distributions. We prove that finding segments whose rating distribution is close to input ones is NP-Complete (Section 2.4.1), by a reduction from the Minimum Height Decision Tree Problem [T$^+$07].

   (b) We conduct a thorough study on several distance measures for comparing rating distributions and we show that Earth's Mover Distance (EMD) is well-adapted to our problem (Section 2.3).

   (c) We propose an efficient algorithm for building Partition Decision Trees (PDT) that satisfy different quality criteria: *coverage* of input records, *size* and *diversity* of resulting population segments. Moreover, we propose heuristics for combining the resulting partitions of a PDT to further improve the quality criteria (Section 2.4).

   (d) Our experimental evaluation (Section 2.5) on real and synthetic datasets validates the utility of rating maps for both analysts and end-users.

2. **Difference Explanation.** In Chapter 3, we study difference explanation in the context of monitoring applications, such as retail sales from *Intermarché* stores. Given a quantity of interest (e.g., number of sales) which aggregates values in two different datasets (e.g., sales of two stores), we are interested in finding a parsimonious explanation set of data segments (e.g., demographic groups) for which the values of that quantity significantly differ between the two datasets. [2] We make the following contributions:

   (a) We formalize the problem of finding a parsimonious set of explanations as an optimization problem of maximizing the overall strength of differences being explained (i.e., informativeness) constrained by the number or description length of data segments (i.e., conciseness). We show that finding a parsimonious set of

[1] Work under review: [AYKKK$^+$16]. [2] Work under review: [KCAY16].

explanations is NP-hard, by a reduction from the maximum dispersion problem [GoCBBRD77].

(b) We propose two scoring functions that decompose the differences in datasets w.r.t. the quality criteria. We prove two interesting properties of those functions: *sub-modularity* and *monotonicity* (Section 3.3.2).

(c) We design a greedy algorithm exploiting the scoring functions with provable near-optimal approximation guarantees (Section 3.4).

(d) Our experiments (Section 3.5) on real retail sales data validate the utility of parsimonious explanations and their high quality w.r.t. a simple baseline explanation containing the top-k differences.

3. **Difference Evolution.** In Chapter 4, we study difference evolution in the context of monitoring applications, such as users' subscriptions in news, and network traffic. Given a stream of data, we are interested in finding moments of significant divergence in data distributions between incoming data and historical data. [3] We make the following contributions:

(a) We introduce and formalize drift queries that provide flexibility in analyzing precision and recall of drift detection at different time granularities (Section 4.4).

(b) We propose a drift index, a structure that maintains clustering summaries of data at multiple time granularities and enables flexible drift queries (Section 4.3).

(c) We propose learning algorithms for adapting our drift and clustering parameters to the various rates and types of drifts (Section 4.3.4).

(d) We perform a thorough study of the performance of our algorithms on real-world and synthetic datasets with varying rates of change (Section 4.5).

[3] Relevant publications: [KCAYDC14, KAYDCC15].

Chapter 2

Difference Exploration

2.1 Introduction

Collaborative rating systems are routinely used by analysts to understand the preferences of different rater populations, and by end-users to make daily choices such as joining a book club or renting a movie. While many item-centric recommendation approaches have been proposed [BGLB15], very little has been done to contrast and compare the ratings of different population segments and enable the exploration of their opinions. In this work, we propose *rating maps*, a collection of disjoint *(population segment, rating distribution)* pairs, and study how to build them dynamically and their utility in the exploration of rated datasets.

Our input data is a set of rating records in the form of ⟨user, item, rating⟩ to which user demographics and item attributes are associated [MOV73]. Population segments such as *middle-aged people in the USA who rated J. K. Rowling's books* or *young artists who rated Sci-Fi movies*, can be constructed from those records. Figure 2.1 illustrates an example on IMDb [1] for the movie *The Social Network*. One can see that these *pre-computed segments* have similar average ratings and do not carry more information than the overall average. Figure 2.2, on the other hand, shows how population segments are built on-demand from rating records. For instance, the movie exploration scenario is a multi-step process where an analyst requests a rating map containing segments of raters who *like* or *dislike Sci-Fi* movies. The result of Step 1 is a rating map with four segments: two that like *Sci-Fi* movies by directors *Ridley Scott* and *Stanley Kubrick*, and two that dislike movies by directors *Sidney J. Furie* and *Roger Christian*. The analyst continues her exploration by selecting the rating records of *Ridley Scott*. As a result, a rating map containing two segments is obtained. The first segment contains the rating records of *Alien (1979)*, and the second contains the rating

[1] *http://www.imdb.com*

Fig. 2.1 Pre-computed segments on IMDb

records of movies starring *Russel Crowe*. Finally, Step 3 reveals that fans of *Alien (1979)* are *young artists* and a group of people living in *Washington*.

Similarly, Mary, an American end-user is looking to find an online book club to discuss author *Debbie Macomber* (Figure 2.2). Mary is shown a rating map containing two segments: readers who agree with her, i.e., *middle-aged* reviewers who do not like the book *204 Rosswood Lane*, and those who disagree with her, i.e., people who love the book *Changing Habits*. These examples show the utility of building rating maps.

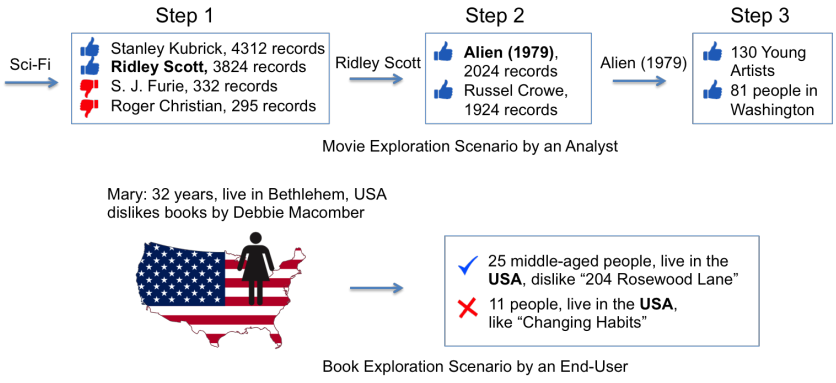


Fig. 2.2 Dataset Exploration with Rating Maps

As illustrated above, an analyst or an end-user, who are interested in exploring a rated dataset would benefit from the ability to find rating maps containing population segments *whose rating distributions are close to some input distributions of interest*. A number of challenges arise when building rating maps. First, the choice of which segments to include in the map must be *flexible*, i.e., determined by input rating distributions of interest. Second, the segments forming a map must *cover* as many input rating records as possible. Third, to

be informative, a segment should not contain *too few records*. Finally, segments descriptions must be *diverse* to show different facets of the rater population.

Contributions. Our first contribution is the choice of a measure for comparing the ratings of different populations (Section 2.3). We adopt the Earth Mover's Distance (`EMD`) [RTG00], a measure that captures the minimum amount of work required to transform one distribution into another. We show that average as well most sophisticated distance measures fail to discriminate between distributions that are intuitively quite different.

Our second contribution is to formalize building rating maps as a simple optimization problem that encompasses the challenges we want to tackle (Section 2.2). We propose to represent population segments using *a decision tree* [T$^+$07] where nodes split an arbitrary set of rating records along user and item attributes. We conjecture that segments with short descriptions are more likely to be large enough and cover more input rating records. Hence, our problem is formulated as *finding a (partial) partition of a set of rating records into a rating map such that each segment in the partition has the shortest description possible, and enjoys a rating distribution that has a low* `EMD` *with respect to some input distribution.*

Population segments need to be *dynamically discovered from an exponential search space*. Our third contribution (Section 2.4) is to show that our problem is NP-complete and propose `DTAlg`, an elegant linear time algorithm. `DTAlg` extends the classic decision tree algorithm [T$^+$07]: whereas classic decision trees are driven by gain functions like entropy[2] and Gini-index,[3] `DTAlg` is based on the `EMD` (Section 2.3.1) whose properties it leverages to speed up processing.

Splitting input records into segments does not guarantee that all records will belong to the resulting rating map. Thus, to improve coverage, we draw inspiration from the work of Breiman [Bre01] and propose `RF`, an approach based on *Random Forests*. `RF` runs several iterations of `DTAlg` on a random subset of user and item attributes. Unlike the `RF` approach used for classification or regression, we face the challenge of combining partitions obtained by different runs of `DTAlg` and obtain high quality rating maps. We develop novel heuristics tailored to address the challenges raised above, namely, coverage of input records, segment size, and segment description diversity.

We run comprehensive experiments on real and synthetic datasets and demonstrate the effectiveness of rating maps in exploring rated datasets (Section 2.5). In particular, we develop scenarios for both analysts and end-users (Section 2.5.3). We confirm the efficiency of `DTAlg` and identify the `RF` heuristic with the best compromise between the quality of generated maps and response time (Section 2.5.4). Section 4.6 summarizes and concludes the chapter.

[2] *http://en.wikipedia.org/wiki/Entropy* [3] *https://en.wikipedia.org/wiki/Gini_coefficient*

2.2 Data Model and Problem

A rated dataset consists of a set of users with schema $S_{\mathscr{U}}$, items with schema $S_{\mathscr{I}}$ and rating records with schema $S_{\mathscr{R}}$. For example, $S_{\mathscr{U}}$ = ⟨uid, age, gender, state, city⟩ and a user instance may be $\langle u1, young, male, NY, NYC\rangle$. Similarly, movies on IMDb[4] can be described with $S_{\mathscr{I}} = \langle$item_id, title, genre, director$\rangle$, and the movie *Titanic* as $\langle i2, Titanic, Romance, James\ Cameron\rangle$. The schema of rating records is $S_{\mathscr{R}} =$ $\langle$uid, item_id, rating$\rangle$. The domain of rating depends on the dataset, e.g., $\{1,...,5\}$ in MovieLens [MOV73], $\{1,...,10\}$ in BookCrossing.[5] As an example, the record $\langle u1, i2, 5\rangle$, essentially says that *a young male from NYC assigned 5 to the romance movie Titanic, directed by James Cameron.* An *instance* consists of relations $\mathscr{U}, \mathscr{I}$, and $\mathscr{R}$ over their respective schemas.

2.2.1 Population Segments and Rating Maps

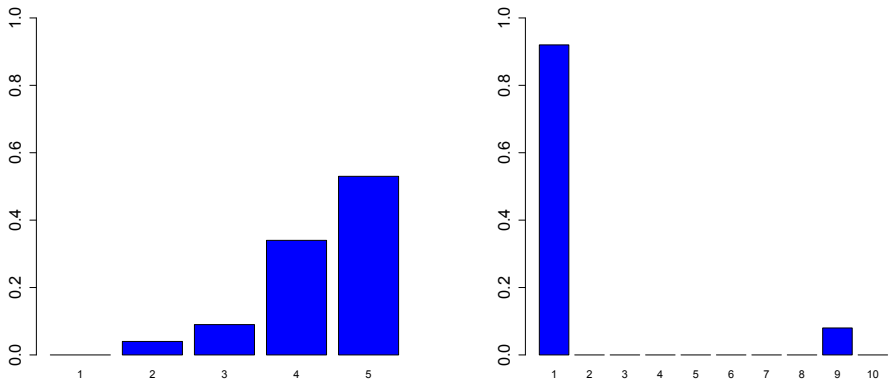
Population Segments. We adopt the formalism of [DAyDY11] whereby a rated dataset $\mathscr{R}$ is viewed as population segments that are *structurally describable* using a conjunction of predicates on user and item attributes of the form Attr = *val*. For a population segment g, we let $g.idesc$ (resp., $g.udesc$) denote the set of item (resp., user) predicates associated with g. We use $g.desc$ to refer to $g.idesc \wedge g.udesc$. E.g., for $g_1.desc = \{$genre $= Romance$, gender $=$ $male$, state $= NY\}$, $g_1.idesc$ refers to the first predicate and $g_1.udesc$ to the remaining ones. We abuse the notation and write $u \in g$ (resp., $i \in g$), to mean user u (resp., item i) satisfies all user (resp., item) predicates in $g.udesc$ (resp., $g.idesc$).

Rating Distributions. The set of all population segments that contributed ratings in a dataset $\mathscr{S} \subseteq \mathscr{R}$ is denoted $G^{\mathscr{S}}$. Given a segment $g \in G^{\mathscr{S}}$, we define $records(g, \mathscr{S}) = \{\langle u, i, r\rangle \in \mathscr{S} \mid u \in g \wedge i \in g\}$ as the set of rating records of all users in g on items in g, in the rated set $\mathscr{S}$. The rating distribution of g in $\mathscr{S}$ is defined as a probability distribution, $dist(g, \mathscr{S}) = [w_1, \ldots, w_M]$ where the rating scale is $\{1, \ldots, M\}$ and $w_j = \frac{|\{\langle u,i,r\rangle \in records(g,\mathscr{S}) | r=j\}|}{|records(g,\mathscr{S})|}$ is the fraction of ratings with value j in $records(g, \mathscr{S})$. We blur the distinction between g and $records(g, \mathscr{S})$ and speak of the records in g or the size $|g|$ of g.

Figure 2.3 contains two example rating distributions including a *high* distribution of young artists for *Alien (1979)* and Mary's *low* distribution for books by *Debbie Macomber*.

Comparing Rating Distributions. We assume a generic function ratComp that compares two rating distributions and returns a score to reflect how far apart they are. We will explore the choices for ratComp in Section 2.3.

[4] *http://www.imdb.com* [5] *http://www2.informatik.uni-freiburg.de/~cziegler/BX/*



(a) (b)

Fig. 2.3 (a) Young artists' distribution for Alien (1979) (b) Mary's distribution for Debbie Macomber

Rating Maps. Given a dataset $\mathscr{S}$ and its population segments $G^{\mathscr{S}}$, a rating map associated with $\mathscr{S}$ is a set of pairs $(g, dist(g,\mathscr{S}))$ where $g \in G^{\mathscr{S}}$. Multiple rating maps could be built for $\mathscr{S}$. The choice of which $g \in G^{\mathscr{S}}$ will belong to a rating map depends on the application. A rating map may contain population segments whose distributions are similar (using the function `ratComp`) to *unanimous* distributions $U_1, \ldots, U_M$. Here, U_i denotes the distribution where the mass is concentrated at rating value i: $U_i(j) = 1, j = i$ and $U_i(j) = 0, j \neq i$. For example, $U_1 = [1,0,0,0,0]$ in a rating scale of 5. Another example is a rating map containing *polarized* distributions $U_{1,M}$ where mass is concentrated on the extreme ratings 1 and M: e.g., $U_{1,M}(1) = U_{1,M}(M) = 0.5$ and $U_{1,M}(j) = 0, j \neq 1, M$.

In a first example, $\mathscr{S}$ contains all rating records for *Alien (1997)*. An analyst could be interested in a rating map for $\mathscr{S}$ whose distributions are in the vicinity of $U_{4,5}$, i.e., people who liked *Alien (1997)*. In that case, the young artists' distribution in Figure 2.3a, could be returned. In a second example, $\mathscr{S}$ is the set of all rating records for *Debbie Macomber*'s books. Mary, whose distribution for those books is given in Figure 2.3b, is interested in engaging in stimulating debates on books. She would find it useful to have a rating map containing segments whose distributions are close to hers, and segments whose distributions are far from hers.

2.2.2 Difference Exploration Problem

Since rating maps are for human consumption, be they analysts or end-users, the description of each segment they contain should be as short as possible. Another desirable property

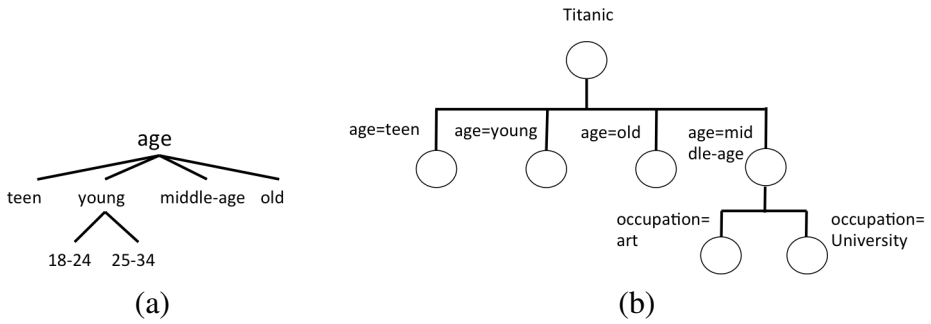


Fig. 2.4 (a) Hierarchy of attribute age (b) Categorical split, where each node is a segment

of rating maps is diversity, i.e., with little or no overlap between its segments descriptions. Finally, rating maps must cover as many records in $\mathscr{S}$ as possible. These goals form the basis of our problem.

We call $[g_1,\ldots,g_\ell]$ a partial partition of $\mathscr{S}$ if g_i's are pairwise disjoint and $\bigcup_i g_i \subseteq \mathscr{S}$. One way to organize a partition is using a *partition decision tree*, defined as follows.

Partition Decision Tree (PDT). Given a rated set $\mathscr{S}$, a *partition decision tree* (PDT) of $\mathscr{S}$ is a rooted tree T such that: (i) the root of T contains the set $\mathscr{S}$ and every node x of T contains a subset of $\mathscr{S}_x \subset \mathscr{S}$ and every edge is labeled with a predicate `Attr` op *val* where *op* is $=$; (ii) for a node x and its children $y_1,\ldots,y_l$, the collection $\{\mathscr{S}_{y_1},\ldots,\mathscr{S}_{y_l}\}$ forms a disjoint partial partition of $\mathscr{S}_x$; (iii) for parent x and child y with the edge (x,y) labeled by the predicate `Attr` op *val*, we have $\mathscr{S}_y = \{t \in \mathscr{S}_x \mid t$ satisfies `Attr` op *val*$\}$.

Attribute values can be organized in a hierarchy. Figure 2.4a shows a partial hierarchy of attribute age from MovieLens. In order to generate PDTs, we use *categorical splitting*. Given an attribute $\texttt{Attr}_i$ and a value v_j, this splitting results in n child segments, one for each value $v_j \in V$ where $V = \{v_1,\ldots,v_n\}$ is the active domain of $\texttt{Attr}_i$. Figure 2.4b shows an example of categorical splitting of the rating records of the movie *Titanic* using attribute age. This splitting results in four child segments each node containing the rating records of users whose age is labeled by the node. Our PDTs can represent both categorical and numerical attributes. We can bin numerical attributes. In MovieLens and BookCrossing, age and year are already binned. The height of a PDT is the length of the longest root-to-leaf path. The leaves of a PDT form a disjoint (partial) partition of the rated set $\mathscr{S}$ at the root, denoted $Part(T)$. Each block in $Part(T)$ naturally corresponds to a segment and is describable by definition: each of them is described by the conjunction of predicates labeling the edges on the path from the root to the leaf containing the block. We henceforth use the terms block and segment interchangeably.

Building Rating Maps Problem. Building rating maps with short segment descriptions corresponds to finding a PDT of small height. We can therefore state: *Given a rated dataset* $\mathscr{S} \subseteq \mathscr{R}$*, a rating proximity threshold* θ*, and a set of input distributions* $\{\rho_1, \ldots, \rho_p\}$*, find a PDT* T *of* $\mathscr{S}$ *such that the objective function is minimized:*

$$\underset{\forall g \in T, \exists \rho_j : ratComp(dist(g,S),\rho_j) \leq \theta}{\operatorname{argmin}} \quad height(T)$$

PDT T admits a minimum height such that $\texttt{ratComp}(dist(g,\mathscr{S}),\rho_j) \leq \theta$ for some $j \in [1,p]$. A typical value for p is 3, indicating low, high and polarized input distributions.

The expression $\texttt{ratComp}(dist(g,\mathscr{S}),\rho_j) \leq \theta$, finds population segments whose distributions are close to some input distribution. In order to find distributions that are far from input distributions, $\texttt{ratComp}(dist(g,\mathscr{S}),\rho_j) \geq \theta$ can be used without affecting our algorithms.

2.3 Rating Comparison Measures

A key ingredient in the problem we study is the choice of the function `ratComp` that quantifies the proximity between two rating distributions. In this section, we review the Earth Mover's Distance (`EMD`) and argue why it is the best choice among a variety of widely used comparison measures. Then, we provide a linear time algorithm for computing the EMD distance between two distributions.

EMD. Intuitively, `EMD` is the minimum amount of work done per unit mass in converting one distribution to another, where the distributions are viewed as piles of earth at various positions [RTG00]. The rating distributions we consider (Section 2.2.1) are normalized and are therefore probability distributions. On discrete domains, `EMD` computation is similar to the well-known transportation problem [RTG00].

Let $\rho_1 = [1 : p_1, .., i : p_i, .., M : p_M]$, $\rho_2 = [1 : q_1, .., i : q_i, .., M : q_M]$ represent two probability distributions over a discrete domain $D = \{1, 2, 3, \ldots, M\}$. The amount of work required to convert ρ_1 to ρ_2 is defined as:

$$\min_F \texttt{Work}(\rho_1, \rho_2, F) \; = \sum_{i=1}^{M} \sum_{j=1}^{M} d_{ij} f_{ij}$$

subject to the constraints: $f_{ij} \geq 0 \;\; 1 \leq i, j \leq M$; $\sum_{j=1}^{M} f_{ij} = p_i \;\; 1 \leq i \leq M$; and $\sum_{i=1}^{M} f_{ij} = q_j \;\; 1 \leq j \leq M$, where f_{ij} is the amount of mass moved from position i to j in the process of converting ρ_1 to ρ_2. $F = [f_{ij}]$ is the matrix representing the flows and d_{ij} is the ground distance from position i to j, which, for simplicity, is defined as the absolute difference in positions, $|i - j|$.

Measure	(ρ_1, ρ_2)	(ρ_1, ρ_3)
`Cosine`	0.058	0.058
`KL-Divergence`	3.13	3.13
`JS-Divergence`	0.53	0.53
`Euclidean distance`	1.24	1.24
`Hellinger Distance`	0.791	0.791
`Total Variation Distance`	0.875	0.875
`Renyi Entropy Distance (0.5 order)`	1.962	1.962
`Battacharya Distance`	0.981	0.981
`Distance correlation`	0.2500	0.2500
Signal Noise Ratio	2.0372	4.221
Lukaszyk-Karmowski Metric	1.1625	3.525
EMD	0.875	3.5

Table 2.1 Distance Measures Comparison

A flow F is optimal if the work done in the flow is minimum among all flows that convert ρ_1 to ρ_2. Therefore, the `EMD` is defined as:

$$\texttt{EMD}(\rho_1, \rho_2) = \frac{\min_F \texttt{Work}(\rho_1, \rho_2, F)}{\sum_{i=1}^{M} \sum_{j=1}^{M} f_{ij}}$$

`EMD` work is done per unit mass in an optimal flow. In our setting, the region D over which `EMD` is calculated is always the whole domain of the distribution, so the value of the denominator in the above equation is 1. Thus, we can ignore the denominator and speak of `EMD` as the work done itself.

EMD vs other measures. Table 2.1 shows various distance scores between two pairs of distributions.[6] We use:

$$\rho_1 = [0.9, 0.025, 0.025, 0.025, 0.025],$$

$$\rho_2 = [0.025, 0.9, 0.025, 0.025, 0.025],$$

$$\rho_3 = [0.025, 0.025, 0.025, 0.025, 0.9]$$

corresponding to ratings on three books i_1, i_2, i_3. Intuitively, distributions ρ_1 and ρ_2 are more in agreement with each other than ρ_1 and ρ_3: users have similar opinions about books i_1 and i_2 and different opinions about i_1 and i_3. KL-divergence, a well-known proximity measure for probability distributions, defined as $D_{KL}(\rho_1, \rho_2) = \sum_j \rho_1^j log(\frac{\rho_1^j}{\rho_2^j})$, and its symmetric counterpart, JS-divergence, defined as $D_{JS}(\rho_1, \rho_2) = \frac{1}{2}(D_{KL}(\rho_1, \rho_3) + D_{KL}(\rho_2, \rho_3))$, where

[6] *http://en.wikipedia.org/wiki/Statistical_distance*

$\rho_3 = \frac{1}{2}(\rho_1 + \rho_2)$, are two natural choices for us [KL51]. Or we could interpret rating distributions as vectors and use cosine or Euclidean distance.

Table 2.1 shows that only Signal Noise Ratio (SNR), Lukaszyk-Karmowski metric, and EMD distinguish between the two pairs. Lukaszyk-Karmowski has the undesirable property that the distance between a distribution and itself is not zero. While SNR works for the above example, consider:

$$\rho_1 = [0.0125, 0.0125, 0.0125, 0.0125, 0.95],$$

$$\rho_2 = [0.0025, 0.0025, 0.0025, 0.0025, 0.99],$$

$$\rho_3 = [0.95, 0.0125, 0.0125, 0.0125, 0.0125],$$

$SNR(\rho_1, \rho_2) = 10.11, SNR(\rho_1, \rho_3) = 6.25, SNR(\rho_2, \rho_3) = 16.37$

This places ρ_1 closer to ρ_3 than to ρ_2, which is counterintuitive. However, $\texttt{EMD}(\rho_1, \rho_2) = 0.01$, $\texttt{EMD}(\rho_1, \rho_3) = 3.75$, and $\texttt{EMD}(\rho_2, \rho_3) = 3.85$. So EMD finds ρ_1, ρ_2 closer to each other than either of them to ρ_3, with ρ_1 being a little closer.

One important property that we exploit in our algorithms is the *additive property* of EMD: $d_{ik} = d_{ij} + d_{jk}$, for $i < j < k \in [1, M]$. *Specifically, our algorithms and results hold for all definitions of distance d_{ij} between ratings i and j that satisfy the additive property.*

EMD with Differential Score Distances. Clearly, defining $d_{ij} := |i - j|$ satisfies the additive property. However, intuitively it can be argued that a change between 10 and 9 (top ratings) is more significant than between 2 and 1 (bottom ratings). The EMD framework is general enough to accommodate a *differential* treatment of score distances. As an example, we could define the ground distance between rating scores i and j as $d_{ij} = |e^{\alpha i} - e^{\alpha j}|$, where $\alpha \in (0, 1]$ is a tuning parameter. E.g., for $\alpha = 0.25$, $d_{9\,10} = 2.69 > d_{12} = 0.36$. This function satisfies the additive property used.

2.3.1 EMD Calculation

A key operation of our problem is computing the EMD between a segment and its closest input distribution. In this section, we present a single-pass algorithm to do that.

In general, the calculation of EMD between two distributions is done using the Hungarian algorithm and takes time $O(M^3 log M)$ where M is the domain size of the distribution [Kuh55]. However, in our setting, the distributions are probabilities over the same domain (rating scale), thus it is possible to compute their EMD in linear time. A similar observation was also exploited by [LLV07]. The key insight is to use a stack to manipulate the *flow* that corresponds to the amount of mass moved to transform one distribution to another. The stack

makes use of the additive property of EMD (defined in Section 2.3) and helps minimize the work needed to find the closest distribution from among a set of input ones. We remark that *even when differential ground distance functions such as the ones illustrated in Section 2.3 are used, our algorithms and results hold with minimal adaptation. For simplicity of exposition below, we assume the distance between two positions is computed as* $d_{ij} = |i - j|$.

Computing EMD of Two Distributions

For clarity, we explain our algorithm with an example. Suppose we want to measure the EMD between two rating distributions ρ_1 and ρ_2. We make one pass over ρ_1, ρ_2 starting from the left-most positions (1). A position i is an *excess* (resp., *deficit*, *equal*) position iff $\rho_1[i] > \rho_2[i]$ (resp., $\rho_1[i] < \rho_2[i]$, $\rho_1[i] = \rho_2[i]$). We keep track of each position as we scan it. We move mass such that ρ_1 converts to ρ_2, and keep track of the mass flow $F[i, j]$ from position i to j.

When we encounter an excess position, we store it on a stack which becomes an *excess state*. Future excess positions are pushed to the stack while deficit positions are processed by flowing mass out of the top excess position on the stack. The stack may transition to *deficit state* as deficit positions are seen. Thus the stack is always in a well-defined state – excess or deficit. It reaches an *equal state* when it is empty. For each position type, we perform the following actions.

Excess Position ($\rho_1[i] > \rho_2[i]$): Set the flow $F[i,i] = \rho_2[i]$. If the stack is in equal or excess state, push the entry $(i, \rho_1[i] - \rho_2[i])$ onto the stack. This is the excess mass available at i. If the stack is in deficit state, pop the top element, say (j, δ), i.e., there is a deficit of δ at j. If $\mu =_{def} \rho_1[i] - \rho_2[i] > \delta$, set $F[i, j] = \delta$ and decrement δ from μ. Repeat this for remaining deficit positions on stack until excess mass is left in μ. If μ becomes $< \delta$, set $F[i, j] = \mu$, and set $\delta = \delta - \mu$ and $\mu = 0$. In the end, the stack may remain in deficit state or move to equal state (i.e., become empty). If position i remains an excess position push its entry on stack and move the stack to excess state.

Deficit Position ($\rho_1[i] < \rho_2[i]$): It is the mirror analog of the above case and we omit the obvious detail.

Equal Position ($\rho_1[i] = \rho_2[i]$): We set the flow $F[i,i] = \rho_1[i]$ and simply move to the next position.

Example 1 Table 2.2 illustrates the algorithm on an example with $\rho_1 = [0.2, 0, 0.2, 0.3, 0.3]$ and $\rho_2 = [0, 0, 0.5, 0.5, 0]$. In Step 1, since there is excess mass of 0.2 at $\rho_1[1]$, it is pushed to the stack and the stack state changes from equal to excess. In Step 2, since $\rho_1[2]$ is an equal position, the algorithm simply moves to the next position. $\rho_1[3]$, is a deficit position, so we use excess positions on the stack to flow mass to $\rho_1[3]$. However, since there is not enough

Step	stack	state	work
0	$\{\ \phi\ \}$	`equal`	0
1	$\{\ (1,0.2)\ \}$	`excess`	0
2	$\{\ (1,0.2)\ \}$	`excess`	0
3	$\{\ (3,0.1)\ \}$	`deficit`	0.4
4	$\{\ (3,0.1),(4,0.2)\ \}$	`deficit`	0.4
5	$\{\ \phi\ \}$	`equal`	0.8

Table 2.2 Running example of EMD computation

mass, $\rho_1[3]$ is added to the stack with a deficit of 0.1 and the state of the stack is changed to deficit. Since there is a mass flow of 0.2 from position 1 to 3, 0.4 work is added. In the next Step, $\rho_1[4]$ is a deficit position and is pushed to the stack. Since $\rho_1[5]$ is an excess position, mass is moved from it to previous deficit positions. Figure 2.5 shows the corresponding flows. It is easy to see that $\texttt{EMD}(\rho_1,\rho_2)$ is 0.8.

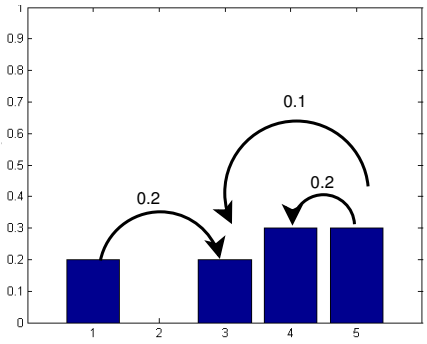


Fig. 2.5 Flows for example in Table 2.2

Algorithm 1 illustrates the steps of calculating the `EMD` between two distributions (ρ_1,ρ_2) of the same rating scale $[1,M]$. At every Step it keeps track of excess, deficit positions and moves the mass such that ρ_1 converts to ρ_2. Algorithm 2 illustrates the process of updating the stack. The complexity of our proposed `EMD` algorithm is $O(M)$.

Theorem 1 *Work computed by the algorithm to convert ρ_1 to ρ_2 is optimal and is equal to* $\texttt{EMD}(\rho_1,\rho_2)$.

The proof relies on the fact that the ground distance function d_{ij} between rating positions i and j satisfies the additive property (see Section 3). For a distribution ρ, let $\rho[1:k]$, $k \leq M$, denote the region of distribution ρ from positions 1 through k, where M is the length of the rating scale. Call region $\rho_1[1:k]$ excess (resp., deficit, or self-sufficient [SS for short]),

Algorithm 1 : EMD($\rho_1[1,M], \rho_2[1,M]$)

```
1: work = 0 // work done
2: Stack S = φ
3: state = equal
4: for i = 1 → M do
5:    if ρ1[i] > ρ2[i] then
6:       if state = excess or equal then
7:          S.push({i, (ρ1[i] − ρ2[i])})
8:       else if state = deficit then
9:          work + = updateStack(i)
10:      end if
11:   else if ρ1[i] < ρ2[i] then
12:      if state = deficit or equal then
13:         S.push({i, (ρ2[i] − ρ1[i])})
14:      else if state = excess then
15:         work + = updateStack(i)
16:      end if
17:   end if
18: end for
19: return work
```

Algorithm 2 : updateStack(i)

```
1: work = 0;
2: mass = |ρ1[i] − ρ2[i]|
3: while mass >0 and S.isEmpty = false do
4:    {k, kmass} = S.pop()
5:    work = work + (kmass)(|i − k|)
6:    updateFlow(kmass,i,k,state)
7:    if mass ≥ kmass then
8:       mass -= kmass
9:    else
10:      mass = 0;
11:      S.push({k, (kmass−mass)})
12:   end if
13: end while
14: if mass >0 then
15:    state = !state //invert
16:    S.push({i, mass})
17: end if
18: if S.isEmpty = true then
19:    state = equal
20: end if
21: return work
```

whenever the sum of the mass in the corresponding positions in the two distributions is related as follows: $\sum_{i=1}^{k} \rho_1[i] > \sum_{i=1}^{k} \rho_2[i]$ (resp., $\sum_{i=1}^{k} \rho_1[i] < \sum_{i=1}^{k} \rho_2[i]$,or $\sum_{i=1}^{k} \rho_1[i] = \sum_{i=1}^{k} \rho_2[i]$). We can show that whenever a region $\rho_1[1:k]$ is excess or SS, any optimal flow does not involve an inflow into the region. Similarly, whenever $\rho_1[1:k]$ is a deficit or SS region, any optimal flow does not involve an outflow from this region. These facts can be proved by direct contradiction, by showing that any flow which does involve such flows can be modified into a flow involving no such flows and which does strictly less work.

Let $\rho[1:k]$ be a minimal SS region. That is, $\sum_{i=1}^{k} \rho_1[i] = \sum_{i=1}^{k} \rho_2[i]$ and $\forall j < k$: the region $\rho_1[1:j]$ is not SS. It follows from the observations in the preceding paragraph that in an optimal flow, there is neither an inflow nor an outflow that involves the region $\rho_1[1:k]$. In general, k can be any number in the range $[1,M]$. Let us consider a general case for k. Let F^{opt} represent the matrix corresponding to the optimal flow and G denote the matrix corresponding to the flow determined by our (greedy) `EMD` algorithm. We denote by $F^{opt}[i:j][k:l]$ the submatrix of F^{opt} corresponding to rows i–j and columns k–l of F^{opt}. These entries capture the flows between corresponding pairs of positions in these ranges. It follows from the above observations that $F^{opt}[1:k][k+1:M] = 0_{k\times(M-k)}$ and $F^{opt}[k+1:M][1:k] = 0_{(M-k)\times k}$, where $0_{i\times j}$ denotes an $(i \times j)$ matrix all of whose entries are zeros.

It is easy to show from the property of our stack-based greedy algorithm that $G[1:k][k+1:n] = 0_{k\times(M-k)}$ and $G[k+1:M][1:k] = 0_{(M-k)\times k}$ as well. That is, the greedy flow coincides with the optimal flow between the positions indicated. It thus suffices to show that $G[1:k][1:k] = F^{opt}[1:k][1:k]$ and $G[k+1:M][k+1:M] = F^{opt}[k+1:M][k+1:M]$. We can show by induction on k that $G[1:k][1:k] = F^{opt}[1:k][1:k]$. The base case of $k = 1$ is trivial. Assume w.l.o.g. that $\rho_1[1]$ is an excess or SS position. If it is a deficit position, simply interchange ρ_1 and ρ_2, exploiting the fact that `EMD` is a symmetric measure. Now, move all the excess mass, if any, from $\rho_1[1]$ to $\rho_1[2]$. Create an instance of the `EMD` calculation problem for $\rho_1' := \rho_1[2:k]$ and $\rho_2' := \rho_2[2:k]$. By induction hypothesis, the greedy flow coincides with the optimal flow on the instance (ρ_1', ρ_2'). This flow can be edited into an optimal flow for the original instance $(\rho_1[1:k], \rho_2[1:k])$ by accounting for the move of excess mass from $\rho_1[1]$ to $\rho_1[2]$. It is easy to see that the total work done by this edited flow equals the work done by the optimal flow on the instance $(\rho_1[1:k], \rho_2[1:k])$. We just showed that $G[1:k][1:k] = F^{opt}[1:k][1:k]$. The claim $G[k+1:M][k+1:M] = F^{opt}[k+1:M][k+1:M]$ can be shown analogously.

2.4 Building Rating Maps

Our problem requires to develop algorithms for dynamically building rating maps driven by desired input distributions. We first discuss the inherent complexity of the problem and then we present our algorithms.

2.4.1 Problem Complexity

Theorem 2 *Given a rated dataset $\mathscr{S} \subseteq \mathscr{R}$, a set of input distributions, and an* EMD *threshold θ, finding a minimum height partition decision tree for $\mathscr{S}$, where each segment's* EMD *is at most θ from some input distribution, is NP-complete.*

Proof of Theorem 1. We show that the decision version of our problem defined in Section 2.2 is NP-hard by reduction from the classic Minimum Height Decision Tree problem [T$^+$07]. Given a set I of n m-bit vectors and a number k, the question is whether there is a binary decision tree with height $\leq k$ such that, each of its leaves is a unique bit vector in I and internal nodes are labeled by binary tests on some bit. We construct an instance J from I as follows. J has m binary attributes $\texttt{Attr}_1, ..., \texttt{Attr}_m$ and a categorical attribute $\texttt{Attr}_0$. For each bit vector s_i, J has two records t_i^+ and t_i^-, $i \in [1,n]$, with $t_i^+[j]$ and $t_i^-[j]$ set to the j-th bit of vector s_i, $j \in [1,m]$. Assign $t_i^+[\texttt{Attr}_0]$ and $t_i^-[\texttt{Attr}_0]$ to two distinct constants appearing nowhere else. Finally, the rating value for t_i^+ (resp., t_i^-) is 5 (resp., 1). Set the EMD threshold $\theta = 0$ and let the input distributions be $\{U_1, U_5\}$. E.g., if $I = \{011, 010, 100\}$ then J contains the records $(a_1,0,1,1,5), (b_1,0,1,1,1), (a_2,0,1,0,5), (b_2,0,1,0,1), (a_3,1,0,0,5), (b_3,1,0,0,1)$, the last value being the rating.

Claim. I admits a decision tree of height $\leq k$ iff J admits a partition decision tree of height $\leq k+1$ where each segment at its leaf is describable and exactly matches U_1 or U_5.

Only If: Given a decision tree T for I, by definition, it contains a unique bit vector $s_i \in I$ at each leaf. If we apply this tree to J, we will get a tree each of whose leaf corresponds to a segment containing exactly the records $\{t_i^+, t_i^-\}$, $i \in [1,m]$. These segments do not match either of U_1, U_5. Applying a split based on $\texttt{Attr}_0 = a_i$ versus $\texttt{Attr} = b_i$ divides this segment into two singleton segments $\{t_i^+\}$ and $\{t_i^-\}$ which match U_5 and U_1. The segments are describable. This tree has height one more than that of T.

If: Let T be a partition decision tree of height $\leq k+1$ for J. By definition, each leaf of T contains a unique record of J. Notice that none of the segments at the leaves can contain more than one record with the same rating value, as they are not describable (without disjunction or negation). T must apply the predicates on attribute $\texttt{Attr}_0$ to separate records t_i^+ and t_i^-. Suppose T applies these tests after all other tests. Then the node at which $\texttt{Attr}_0 = a_i$ vs.

$\texttt{Attr}_0 = b_i$ is applied must contain exactly the segment $\{t_i^+, t_i^-\}$. By replacing that segment with the corresponding bit vector s_i, we get a decision tree of height $\leq k$ for I. Suppose T applies one or more tests on $\texttt{Attr}_0$ before other attributes $\texttt{Attr}_i$, where $i > 0$, we can show that we can "push down" those tests on $\texttt{Attr}_0$ so they are applied at the parent of leaf nodes, without increasing the tree height.

Membership in NP is trivial: given a height threshold h and a tree T, we can easily check in polynomial time whether T is indeed a partition decision tree of $\mathscr{S}$, each block has an EMD distance at most δ from some input distribution, and whether the height of T is no more than h.

2.4.2 Algorithms for Difference Exploration

We now describe our algorithms for minimizing description length via finding a minimum height partition decision tree. Whereas classic decision trees [T$^+$07] are driven by gain functions like entropy[7] and gini-index,[8] a novelty in our case is that our decision trees are designed to discover segments whose distributions are close to input ones. Thereto, we leverage the properties of EMD as we will show in Section 2.3.1.

Minimizing Description Length with DTAlg

Our first algorithm, DTAlg, is based on partition decision trees. The classical decision tree algorithm splits an input set using the attribute that offers the best normalized information gain. It then adds the obtained segments as children and recurses. We use this idea in Algorithm 3 that takes as input a rating set $\mathscr{S}$ and divides it in a breadth-first manner, to find segments with short descriptions.

At each node, DTAlg checks if its segment has EMD $\leq \theta$ to some input distribution (lines 3-4). If the segment's EMD distance to the closest input distribution is $> \theta$ (line 5), DTAlg uses a gain function to choose a splitting attribute (line 6), and the segment is split into child segments which are retained (line 7); Finally, retained segments are checked and are either added to the output (line 11) or recursively processed further (line 13).

Splitting using a Gain Function Algorithm 3 relies on a gain function for choosing the attribute with maximum gain for splitting a segment (line 6). We use the *minimum average* EMD as our gain function. Suppose splitting a segment g using an attribute $\texttt{Attr}_i$ yields l children $y_1^i \ldots y_l^i$. The gain of $\texttt{Attr}_i$ is defined as the reciprocal of the average EMD of its

[7] *http://en.wikipedia.org/wiki/Entropy* [8] *http://en.wikipedia.org/wiki/Gini_index*

Algorithm 3 $\texttt{DTAlg}(\mathscr{S}, \{\rho_1, \ldots, \rho_j, \ldots, \rho_p\}, \theta)$

1: $parent = \mathscr{S}$
2: $Array\ children$
3: **if** $min_{j\in[p]}\texttt{EMD}(parent, \rho_j) \leq \theta$ **then**
4: Add $parent$ to $Output$
5: **else if** $min_{j\in[p]}\texttt{EMD}(parent, \rho_j) > \theta$ **then**
6: $Attribute\ \texttt{Attr} = \text{findBestAttribute}(parent)$
7: $children = \text{split}(parent, \texttt{Attr})$
8: **end if**
9: **for** $i = 1 \rightarrow No.\ of\ children$ **do**
10: **if** $min_{j\in[p]}\texttt{EMD}(children[i], \rho_j) \leq \theta$ **then**
11: Add $children[i]$ to $Output$
12: **else**
13: $\texttt{DTAlg}(children[i], \{\rho_1, \ldots, \rho_j, \ldots, \rho_p\}, \theta)$
14: **end if**
15: **end for**

children. If child segments have a zero EMD then the gain is infinity. More formally:

$$\texttt{Gain}(\texttt{Attr}_i) = \frac{l}{\sum_{j=1}^{l} min_{\rho\in\{\rho_1,\cdots,\rho_p\}}\texttt{EMD}(y_j^i, \rho)}$$

An attribute will not be useful for splitting a segment if all the rating records in the segment have the same value for that attribute. For example, if a segment contains rating records of one movie, *Titanic*, none of the movie attributes are useful for splitting the segment. Such attributes are discarded and not considered for further splitting.

Algorithm 3 takes time $O(k^2npM)$, where k is the number of distinct attribute-value pairs that are present in the rating set $\mathscr{S}$, n is the number of rating records in $\mathscr{S}$ and p is the number of input distributions with length of rating scale equals to M.

Improving Coverage with Random Forests

As discussed in Section 2.2, segments with shorter descriptions are expected to contain more records (and hence increase coverage of $\mathscr{S}$). However, splitting $\mathscr{S}$ into segments whose ratings are close to input distributions *does not necessarily guarantee that all records in $\mathscr{S}$ will belong to the resulting segments.* In this section, we focus on obtaining partitions with improved quality. In particular, we devise heuristics that are likely to increase coverage of input records. At a high level, our approach, referred to as RF, runs multiple iterations of DTAlg with different splitting attributes. It then uses one of RF-Cluster, RF-Desc, RF-Random, RF-Size, and RF-EMD, to combine resulting partitions.

We draw inspiration from the work of Breiman [Bre01] who proposed the approach of *Random Forests* to improve the performance of decision tree-based classifiers. The idea is that given d predictor attributes, a classical decision tree examines all d of them to pick the best split attribute and split point at each stage. The `RF` approach consists of two main Steps. In Step 1, the classical decision tree algorithm is run m times for some parameter m, where each run examines a random subset of $\hat{d}$ predictor attributes at each splitting node, a default value for $\hat{d}$ being $\sqrt{d}$. This generates m decision trees. In Step 2, those m trees are combined, e.g., by voting, to yield an ensemble classifier.

In our adaptation, Step 1 of `RF` remains the same: we run the `DTAlg` algorithm on a random subset of $\sqrt{d}$ available user/item attributes m times. For us, *Step 2 should produce a partition, not a classifier.*

It has been shown that using the McNemar test [9] to limit the number of trees generated in Step 1 is effective [LDD01]. In our work, we examine our datasets and determine the best value for m in each case based on improvemet in rating map quality (Section 2.5.1). Below, we examine different strategies for combining the m partitions from Step 1 into a single partition. Instead of improving classification accuracy, our goal is to improve coverage. It is important to note that `RF` heuristics are designed to improve the quality of produced rating maps. We will see in Section 2.5.4, that this improvement comes at a (computational) cost.

Combining Partitions. There are multiple ways of combining the m partitions $p_1, \ldots, p_m$ produced in Step 1 of `RF`. We propose five heuristics that give precedence to different segment quality dimensions, namely overlap, description, size, and `EMD` value.

1. `RF-Cluster`: Each partition p_i intuitively captures a (possibly partial) clustering. For each pair of rating records $r_i, r_j \in \mathscr{S}$, we can compute k_{ij}, the number of partitions to which they both belong. Then, we can define the similarity between r_i and r_j as $sim_{ij} = \frac{k_{ij}}{m}$. Now, we can use any standard clustering algorithm to obtain a clustering of $\mathscr{S}$ with this distance measure (e.g., hierarchical clustering). As the desired number of clusters, we chose the average number of segments in the partitions $p_1, \ldots, p_m$. Only good segments are retained. The resulting segments do not necessarily have a natural exact description. We hence adopt a pattern mining approach to solve this issue. Viewing each record as a transaction and each user and item attribute as an "item", we obtain maximal frequent patterns, by setting the support threshold to 90%. Any maximal frequent pattern serves as an approximate description of the segment, with an accuracy of at least 90%. Algorithm `RF-Cluster` takes $O(mk^2npM + mn^3)$ time

[9] *https://en.wikipedia.org/wiki/McNemar's_test*

where m is the number of decision trees built, n the number of rating records and k the number of distinct attribute-value pairs in the dataset.

2. `RF-Desc`: A second strategy favors a partition containing segments with diverse descriptions. We define the Jaccard distance between segment descriptions as $Jaccard(g_i, g_j) = \frac{|g_i.desc \cap g_j.desc|}{|g_i.desc \cup g_j.desc|}$. `RF-Desc` starts with an empty output partition and successively adds a segment whose total distance to segments in the output is the highest. The first segment is picked at random. This heuristic takes time $O(mk^2npM + (ml)^3)$, where l is the maximum number of blocks produced by any of the m runs of `DTAlg`.

The remaining strategies explore different ways of ordering segments from the various partitions $p_1, \ldots, p_m$ and adding them to the output one by one if they do not overlap with existing segments in the output.

3. `RF-Size`: This heuristic favors larger segments, which may help with coverage of input rating records. Its time complexity is $O(mk^2npM + m^2nl)$.

4. `RF-EMD`: This heuristic favors segments with the lowest `EMD` to their closest input distribution. Its time complexity is $O(mk^2npM + m^2nl)$.

5. `RF-Random`: This heuristic orders segments at random. It takes $O(mk^2npM + nm)$ time.

`RF-Cluster` is by far the most expensive algorithm used for combining partitions, since it requires finding the distance between all pairs of records in $\mathscr{S}$. Section 2.5.4 explores the tradeoff between the quality of segments forming a rating map and the time taken to compute them using these heuristics.

2.5 Difference Exploration Experiments

In this section, we design exploratory scenarios that show the utility of rating maps. That is followed by an evaluation of their quality over synthetic and real datasets. In particular, we study the robustness of our algorithms w.r.t. noisy datasets and demonstrate the quality of rating maps. We then study the scalability of `DTAlg` and `RF` approaches.

2.5.1 Experimental Setup

We use two real datasets, MovieLens (ML) [MOV73] and BookCrossing [10] (BC), summarized in Table 2.3, and a synthetic one, presented in Section 2.5.4. ML contains user attributes `gender, age, occupation` and `location`. We join this data with IMDb (via movie titles) to obtain attributes `title, actor, director, writer` for each movie. BC provides `location, age` for users and `title, author, year, publisher` for books. Some attributes have a hierarchy (e.g, $Country \rightarrow State \rightarrow City$ for `location`). This information is readily available for every user in BC. For ML, we queried Yahoo! Maps[11] to get this information. We manually created hierarchies for attributes `age, year` and `occupation`. Other attributes like `director`, `gender`, `author` have trivial hierarchies (i.e., height = 1).

	MovieLens (+IMDb)	BookCrossing
#Users	6,040	38,511
#Items	3,900	260
#Ratings	1,000,209 (million)	196,842
Rating Scale	1 to 5	1 to 10

Table 2.3 Summary of datasets

While our framework is general enough to admit any input distribution, we will mostly use 3 intuitive distributions *low, high* and *polarized*, to ease exposition. Particularly, for ML, we use $\{U_{1,2}, U_{4,5}, U_{1,5}\}$ and for BC, $\{U_{1,2,3}, U_{8,9,10}, U_{1,2,9,10}\}$. The quality of rating maps is evaluated using several measures: average coverage of input records, average segment description length, average segment size, and average `EMD`. We use `maxEMD` to denote the maximum value that `EMD` can take (4 for ML and 9 for BC).

In order to avoid producing segments with too few rating records, we set 50 as a lower bound threshold on segment size. One could also modify our problem statement in Section 4.2 to state a minimum segment size threshold without affecting the problem complexity. We chose to keep our problem statement general since this restriction on size is data-dependent and can be determined by examining the results of several runs of our algorithm.

Experiments were conducted on 2 GHz Intel Core i7, 8 GB RAM, MAC OS. Code is written in Java, JDK/JRE 1.6.

2.5.2 Summary of Results

In order to examine the utility of rating maps, we design real-world exploration scenarios for analysts and end-users. We give the analyst the ability to find population segments

[10] *http://www2.informatik.uni-freiburg.de/~cziegler/BX/* [11] *https://maps.yahoo.com*

of *low, high* and *polarized* opinions and the flexibility to further explore those segments based on her interests. We enable an end-user to find the most similar/dissimilar population segments to her and to further explore their demographics. The exploration scenarios in Section 2.5.3 demonstrate the utility of rating maps in uncovering the opinion of different raters and guiding users in their exploration.

Experiments on synthetic datasets, in Section 2.5.4, show that the accuracy of our algorithms in finding segments close to input distributions, is sensitive to the amount of noise in input records. However a proper tuning of the θ parameter on how close/far two distributions can be, results in robust performance on noisy data. Moreover, we validate our optimization objective by showing that minimizing description length has a positive impact on the quality of resulting maps.

Experiments on real datasets, in Section 2.5.4, reveal that `RF` heuristics do improve the quality of generated maps over `DTAlg`. They also verify that description length is a good choice to optimize for when building rating maps, as it affects positively other quality dimensions. `RF-Cluster` generates rating maps with the highest quality and `RF-Desc` improves description diversity. However, neither of them scales with larger input datasets. On the contrary, `RF-Size` and `RF-EMD` scale linearly. However, the quality of maps built by `RF-EMD` is sometimes worse than `RF-Random`. At the end, we show that `RF-Size` is the best option as it provides a good tradeoff between producing rating maps with good coverage and scaling linearly with the number of input records.

2.5.3 Exploration Scenarios

In this section, we design scenarios for analysts interested in exploring different population segments. We also study scenarios for end-users curious in discovering segments that share their opinion and segments that differ. Rating maps are built using `RF-Size`, which achieves a good compromise between the quality of resulting maps and response time, as it will be demonstrated in Section 2.5.4.

Analysts' Scenarios

We set θ, the `EMD` threshold to 10% of `maxEMD` i.e., 0.4 for ML and 0.9 for BC. For ML, we tested the quality of rating maps produced by `RF-Size` with 2 to 32 trees and empirically observed that generating 4 trees resulted in the best average `EMD`. We hence set that number to 4 for all `RF`.

ML Dataset. Our analyst is interested in finding rating maps containing a mix of *low, high* and *polarized* opinions about movies. Specifically, she wants to explore how

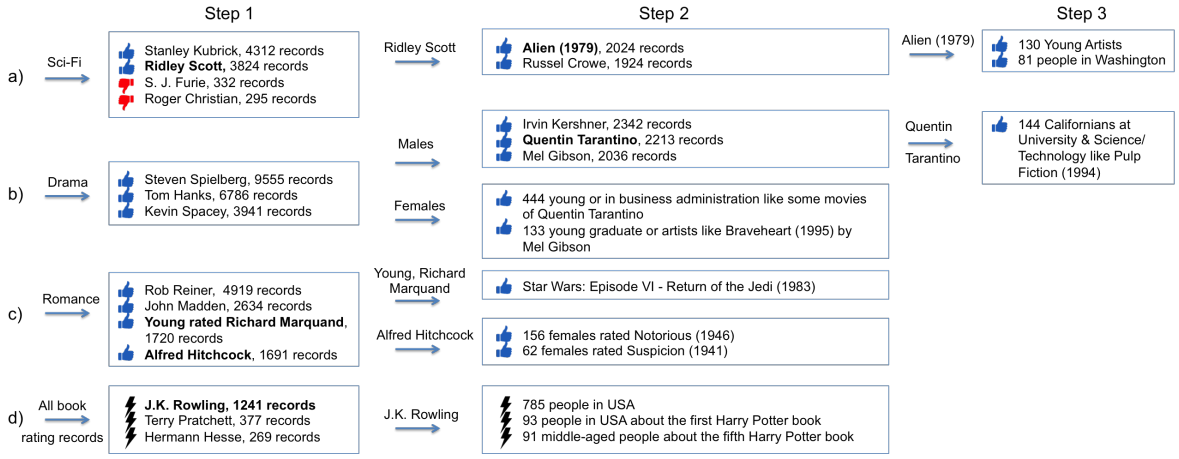


Fig. 2.6 Analyst's exploration scenarios

people's taste is related to the intrinsic characteristics of the movies (e.g., `genre, actors, directors`) or by their demographics (e.g., `age, gender, occupation`).

Figure 2.6a shows a scenario where the analyst wishes to explore the ratings of *Sci-Fi* movies. This exploration results in a first rating map containing high and low distributions that characterize different directors. In Step 2, the analyst requests to have a deeper look at director *Ridley Scott* and obtains a rating map containing two segments with a distribution in the vicinity of *high*: one for the movie *Alien (1979)*, and the other for movies where actor *Russel Crowe* starred. A further exploration of *Alien (1979)*, in Step 3, shows which population segments loved it the most (*Young artists* and residents of *Washington* state).

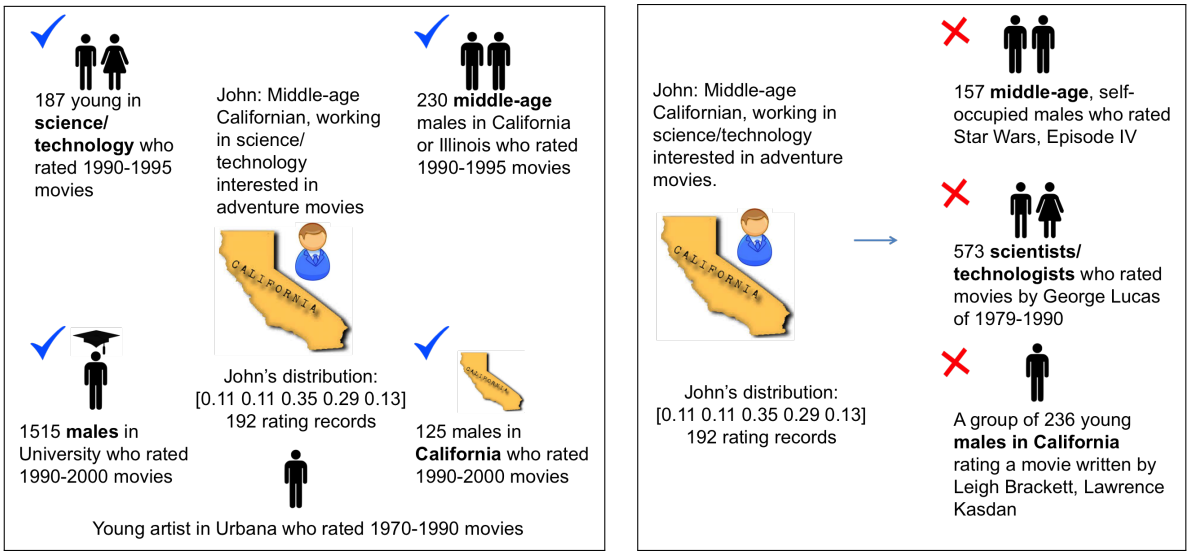
In Figure 2.6b our analyst wants to discover differences between genders for *Drama* movies. Step 1 shows that *males* and *females* agree that *Steven Spielberg*, *Tom Hanks* and *Kevin Spacey* directed the best *dramas*. However, when considered alone, *males* show a preference for directors *Irvin Kershner*, *Quentin Tarantino* and *Mel Gibson*. Particularly, a group of *Californian males* working at a *University* or in *science/technology* show a high preference for *Tarantino*'s *Pulp Fiction (1994)* (Step 3). Our analyst can hence conclude that *males'* taste in drama depends exclusively on movie attributes (e.g., `director`), while *females*' varies depending on `age` and `occupation`. For example, *young* women and women in *business administration* love some movies by *Quentin Tarantino*, while women who are *young graduate*s or *artists* love *Braveheart (1995)* by *Mel Gibson*.

In the last scenario, the analyst is interested in exploring highly rated romantic movies. The first map is shown in Figure 2.6c. Although the analyst was expecting the results of directors *Rob Reiner* and *John Madden*, she is surprised that *young* raters liked a romantic movie directed by *Richard Marquand* and that *Alfred Hitchcock* is also loved for his romantic movies. Thus, she decides to further explore these two results. She finds (Step 2) that *young*

people who rated *Star Wars: Return of the Jedi*, also classified as romantic, are the ones who love romantic movies by *Richard Marquand*. Furthermore, she discovers a group of female fans of *Alfred Hitchcock*'s *Suspicion (1941)* and *Notorious (1946)*.

BC Dataset. This time, our analyst repeats the same exploratory task for finding *polarized* opinions on books. Figure 2.6d shows some of the segments that form the resulting maps. One prominent observation is that people have polarized opinions on author *J. K. Rowling*. A further exploration in Step 2 results in a demographics breakdown for that author. The corresponding rating map helps the analyst understand who causes this polarization.

End-Users' Scenarios



(a) (b)

Fig. 2.7 (a) Similar segments (ML), (b) Different segments (ML)

ML Dataset. John, a *middle-aged Californian* working in *science/technology*, is interested in finding users like him and users different from him on *adventure* movies. Our input dataset consists of all rating records for *adventure* movies and our input distribution is John's computed from his *192* ratings for *adventure* movies, Figure 2.7a. The `EMD` threshold is ≤ 0.1 for similar (resp., ≥ 1.2 for dissimilar) in order to impose very close (resp., not-so-close) distributions to John's. The number of trees generated by `RF` approaches is 4.

John is active in rating *adventure* movies and he is interested in discovering population segments that share his passion. He decides to find segments with which he shares at least one demographic attribute (e.g., `gender, age`). He discovers that *young people* with the same `occupation` and people of the same `age` from *California* or *Illinois* "agree" with him on

adventure movies released in the period *1990-1995*. Moreover, John shares the same opinion with people of the same `gender` working in *universities* and with people of the same `gender` and `occupation` on *adventure* movies released in *1990-2000*. Finally, one intriguing finding for John is another reviewer perfectly matching his distribution on *adventure* movies, *a young male artist* living in *Urbana* and who rated *1970-1990* movies, 53 times. On the contrary, John disagrees, see Figure 2.7b, with a segment of the same `age`, a set of *self-occupied males* who rated *Star Wars, Episode IV*. Also, he disagrees with people with the same `occupation` who rated movies directed by *George Lucas* during *1979-1990*, and with *young Californian males* on a movie written by *Leigh Brackett* and *Lawrence Kasdan*.

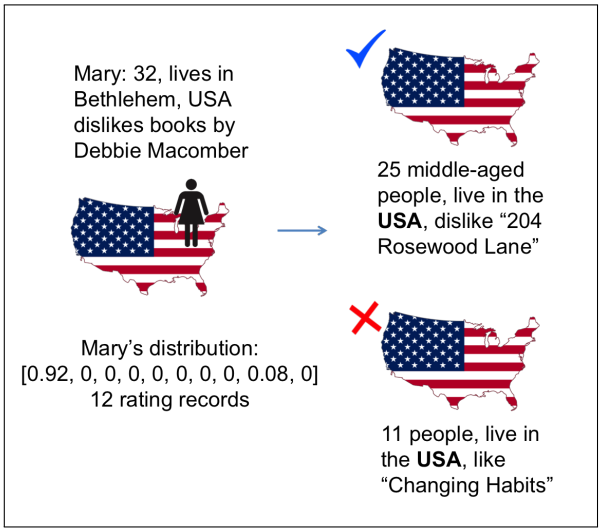


Fig. 2.8 Similar and different segments (BC)

BC Dataset. Mary, a *32 years old woman* living in *Bethlehem, Pennsylvania, USA*, dislikes books by *Debbie Macomber*. Mary is interested in finding population segments, also located in the *USA*, that have similar or dissimilar opinions as hers. Our input dataset consists of all ratings for books by *Debbie Macomber* and the input distribution is Mary's computed from her *12* ratings on books by *Debbie Macomber*, Figure 2.8. We set the `EMD` threshold to ≤ 0.3 for similar users (≥ 3 for dissimilar), the number of trees generated by `RF` to 4, and the minimum segment size to 5.

Figure 2.8 illustrates a group of 25 *middle-aged* people in the *USA* with a negative opinion on the book *204 Rosewood Lane* written by *Debbie Macomber*. On the contrary, a small segment of 11 people living in the *USA* "disagree" with Mary as they highly rated the book *Changing Habits*. Mary can get in touch with people in both segments to engage in online debates on the author.

2.5.4 Detailed Evaluation

We present a quality evaluation of the rating maps built by our algorithms, `DTAlg` and `RF`. Particularly, we use synthetic datasets to stress-test our algorithms over noisy data and show their performance in terms of accuracy (ability to identify the right segments) and rating map quality. We also explore scalability on the real datasets ML and BC.

Synthetic Data

We developed a synthetic data generator that provides the flexibility to produce datasets with different distributions and having different percentages of noise. Noise is defined as the mass distributed in different ratings than the ones where the largest mass is assigned. We produce 5 datasets each one consisting of 5,000 rating records and a percentage of noise, ranging from 10% to 50%. Particularly, our ground truth, GT, consists of rating records where each record is associated to a virtual user. Each virtual user has 3 attributes: `age` with values *Teen, Young, Middle, Old*, `occupation` with values *Lawyer, Doctor, Farmer, Sports, Student* and `location` with values *East, Central*, and *West*. For each segment g, a random distribution is assigned given a noise percentage. For example, for a segment described by $\{\texttt{age} = \textit{Teen}, \texttt{occupation} = \textit{Doctor}, \texttt{location} = \textit{East}\}$ and for a 10% noise percentage, a perturbed U_1 distribution, denoted as $\underline{U}_1$, is formed by assigning a large chunk of mass (0.9) to $U_1(1)$ and uniformly distributing the remaining mass (0.1) as noise over other positions $U_1(i), i \neq 1$. The introduction of noise makes it more difficult for our algorithms to identify segments close to input distributions. The `EMD` threshold θ is set as follows: 0.4 for 10% noise, 0.5 for 20%, 0.6 for 30%, 0.7 for 40%, and 0.8 for 50%. We test our algorithms using the input distributions $\{U_1, ..., U_5\}$.

Accuracy Evaluation. Since our algorithms generate a rating map by partitioning a rated dataset $\mathscr{S}$ into segments in $G^{\mathscr{S}}$, we borrow standard supervised clustering evaluation measures like `Precision`, `Recall` and `F-Measure`,[12] adapted to our context, in order to evaluate the quality of our partitions.

We refer to the ground truth as GT and to an output rating map as G. `Precision` captures the fraction of pure segments in G, where the purity of a segment g_i is defined as its similarity to its closest segment $g \in GT$. More precisely, it is the product of its description purity (based on Jaccard similarity) and its distribution purity (based on `EMD` distance). A similar remark holds for `Recall` except for the difference in the denominator (i.e., $|GT|$). `F-Measure` is

[12] *https://en.wikipedia.org/wiki/Precision_and_recall*

defined as the harmonic mean of `Precision` and `Recall`. The exact formulations are given below. Recall that `maxEMD` is equal to 4 for ML and to 9 for BC.

$$\texttt{Precision}(G, GT) = \frac{\sum_{g_i \in G} max_{g \in GT} \texttt{Purity}(g_i, g)}{|G|}$$

$$\texttt{Recall}(G, GT) = \frac{\sum_{g_i \in G} max_{g \in GT} Purity(g_i, g)}{|GT|}$$

$$\texttt{F}-\texttt{Measure}(G, GT) = \frac{2 * Precision(G, GT) * Recall(G, GT)}{Precision(G, GT) + Recall(G, GT)}$$

$$\texttt{Purity}(g_i, g) = \texttt{DescPurity}(g_i, g) * \texttt{DistPurity}(g_i, g)$$

$$\texttt{DescPurity}(g_i, g) = \frac{|g_i.desc \cap g.desc|}{|g_i.desc \cup g.desc|}$$

$$\texttt{DistPurity}(g_i, g) = \frac{\texttt{maxEMD} - \texttt{EMD}(g_i, g)}{\texttt{maxEMD}}$$

Robustness to Noise. We test the accuracy of all algorithms over the 5 synthetic datasets described earlier. Figure 2.9 depicts that all algorithms are sensitive to noise and shows a decrease in accuracy as noise increases. However, a proper tuning of θ, ranging from 0.4 to 0.8 results in increasing tolerance to noise. It is worth noting that all algorithms achieve a similar accuracy, which is always higher than 70%. Moreover, `RF-EMD` outperforms all other heuristics, as it is the only heuristic which maximizes, by its design, the distribution purity factor (`DistPurity`). However, it produces lower quality maps on other dimensions (Section 2.5.4).

We also studied the accuracy of our algorithms when noise is not uniformly distributed. E.g., for half of input data, the largest mass is assigned to the highest rating value ($U_{1,5}(5) = mass$) and the remaining mass is assigned as noise to the lowest rating ($U_{1,5}(1) = noise$). We observed no significant difference in accuracy. Thus, our algorithms are not affected by how the noise is distributed over rating values.

Description Length vs Rating Map Quality. We study how the description length of population segments affects the quality of the rating map they belong to. The aim of this experiment is to validate if our optimization objective, i.e., minimizing description length, is a good choice. Coverage indicates the proportion of rating records of a dataset $\mathscr{S}$ included in the resulting rating map G, and is defined as $\texttt{Coverage}(G, \mathscr{S}) = \frac{\sum_{g_i \in G} |records(g_i, \mathscr{S})|}{|\mathscr{S}|}$. The

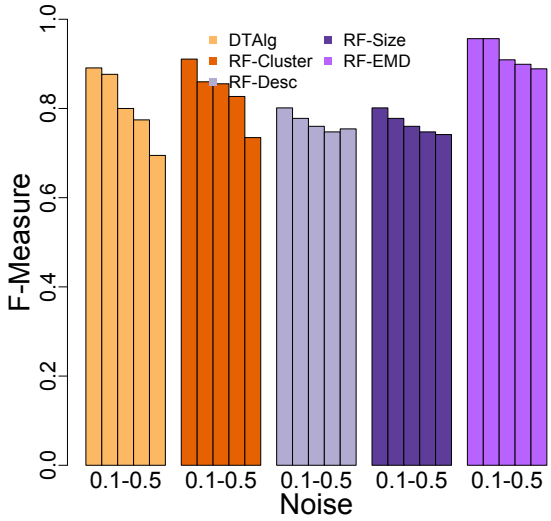


Fig. 2.9 Accuracy of algorithms Vs. Noise

diversity of a rating map G is defined as the average pairwise Jaccard distance between its segments' descriptions, $\texttt{Diversity}(G) = \frac{\sum_{(g_i,g_j)\in G^2, i\neq j} Jaccard(g_i.desc, g_j.desc)}{|\{(g_i,g_j)\in G^2, i\neq j\}|}$.

In order to control the description length, we vary the EMD threshold θ and generate trees of different heights. Table 2.4 summarizes the results of RF-Cluster over the first synthetic dataset (3 attributes, 10% noise). The results of the other algorithms are similar and are omitted. We notice that even a slight decrease of description length results in a relatively high increase of coverage, description diversity and segment size. The value "0" of description length is explained by an empty description with RF-Cluster since it is as generated by the pattern mining algorithm of 90% support threshold (Section 4.2.2.1).

	$\theta = 0.1, \lvert\mathscr{S}\rvert = 9$	$\theta = 0.5, \lvert\mathscr{S}\rvert = 13$	$\theta = 1, \lvert\mathscr{S}\rvert = 9$
Description Length	3	2.77	2.44
Coverage	14.34	61.74	96.64
Description Diversity	0	0.15	0.31
Segment Size	79.67	237.46	536.89
EMD	0.04	0.29	0.59

Table 2.4 Effect of segment Description Length on Rating Map quality for RF-Cluster

Real Data

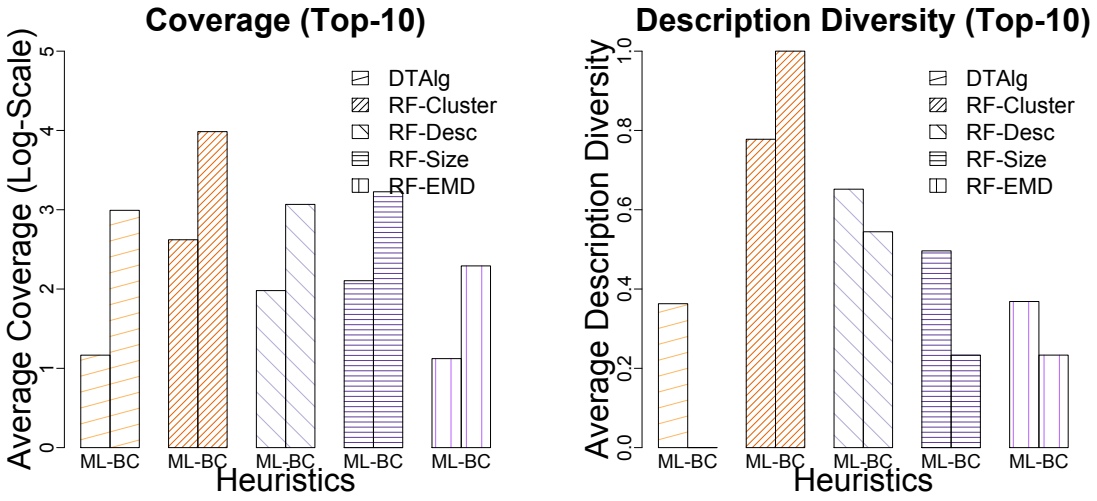
We study the performance of our algorithms on ML and BC. Given the lack of ground truth, we do not compute accuracy results, as in Section 2.5.4. Instead, we study the quality of our rating maps, i.e., average coverage of input records, average segment description length, average segment size, and average segment EMD. We find that RF-Cluster produces high quality rating maps but is not scalable. Therefore, in order to compare our other heuristics to RF-Cluster, we perform our quality experiments on a small sample containing about 1,000 rating records for both ML and BC. We set the EMD threshold to 0.2 for ML and 2 for BC. The number of trees generated by RF approaches is set to 20.

Quality of Heuristics. Figure 2.10 illustrates quality results. Each rating map contains the top-10 population segments, as ranked by the different heuristics. For DTAlg and RF-Cluster, where there is no ranking criterion, we randomly pick 10 segments. Overall, RF approaches achieve better results than DTAlg for all quality measures. This confirms the weakness of a single tree to capture the various non-intersecting segments and the benefit of using a forest of trees. We also show that RF-Cluster produces rating maps with the highest quality overall. The results of RF-Size confirm the assumption that coverage of input records, Figure 2.10b, is favored by large segments, shown in Figure 2.10c. RF-Desc returns segments with high description diversity, Figure 2.10b. RF-EMD achieves the best average EMD, Figure 2.10d, but it performs poorly on the other quality measures. We also observe that it has lower quality than RF-Random. Finally, all heuristics, except RF-EMD, appear to have similar average EMD values.

Table 2.5 shows that RF-Cluster produces population segments with the minimum description length. It is worth mentioning that the resulting segments of RF-Cluster do not have a natural exact description, but they are assigned a description as generated by the pattern mining algorithm (Section 4.2.2.1). Thus, setting the support threshold of frequent patterns to 90% can result in segments with an empty description and an average description length lower than 1.

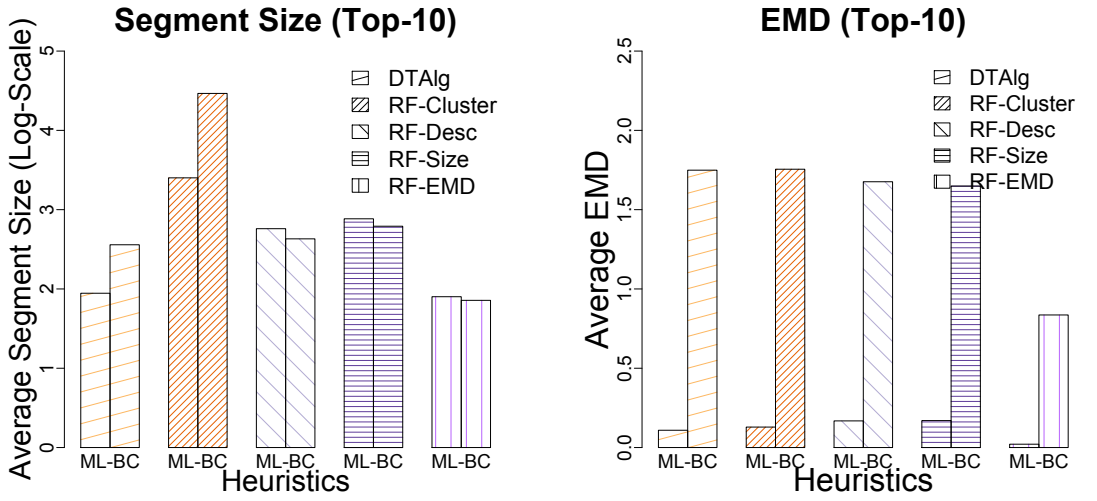
	DTAlg	RF-Cluster	RF-Desc	RF-Size	RF-EMD
ML	2.9	0.9	2	2.3	2.9
BC	1	0.25	1.1	1.3	1.7

Table 2.5 Average Description Length (Top-10)



(a)

(b)



(c)

(d)

Fig. 2.10 Evaluation of Rating Map quality for all RF heuristics

Scalability Evaluation. In order to study scalability, we gradually increase the size of input rating records $\mathscr{S}$ using the ML dataset. Figure 2.11 shows the running times of the various algorithms as the size of $\mathscr{S}$ grows. While we only report results with $250K$ input records, our experiments confirm the high scalability of `DTAlg` (around 4 minutes to process 1M records from ML). However, not all `RF` approaches scale as well. That is expected since they were primarily designed to improve the quality of the resulting maps. In fact, the time taken to create random trees is the same for all `RF` approaches and the difference between them is due to different strategies to combine partitions. Although `RF-Cluster` produces high quality maps (refer to Section 2.5.4, Quality of Heuristics), it scales poorly. On the contrary, `RF-Size` and `RF-EMD` scale linearly with the size of $\mathscr{S}$. However, `RF-Desc` does not scale because finding segments whose descriptions are far apart is expensive. In summary, `RF-Size` achieves a good compromise between the quality of rating maps and response time.

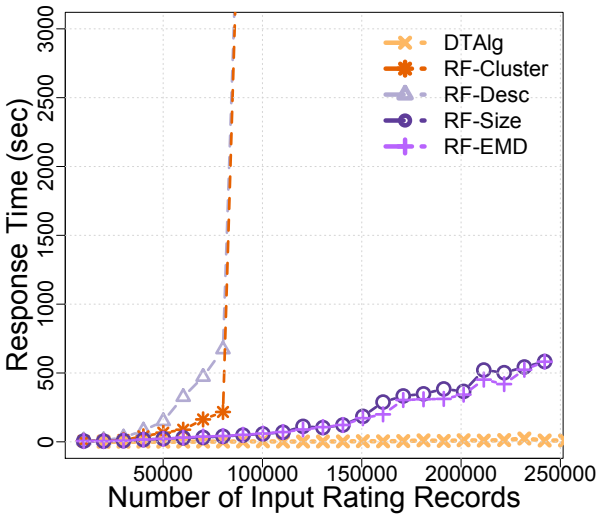


Fig. 2.11 Response time Vs. Input size on ML

2.6 Summary of Difference Exploration

To facilitate online exploration of rated datasets by analysts and end-users, we proposed rating maps consisting of sets of (population segment, rating distribution) pairs with segments that cover a large number of input records, have diverse descriptions, and are close to one of desired distributions. We formulated the problem of finding rating maps as finding partition

decision trees of minimum height and showed that the problem is NP-complete. We proposed a linear time algorithm to the number of input rating records for finding a basic PDT and heuristics based on random forests for improving their quality. Our extensive experiments show that PDTs with short descriptions (small height) achieve high coverage and that among our heuristics, the heuristic that selects the largest segments strikes the best balance between quality of rating maps found and running time.

Chapter 3

Difference Explanation

3.1 Introduction

Many of the changes are hidden in the masses of data collected during daily operations. Such changes may mirror, for instance, external influences such as customer and market trends, but also internal influences such as supply shortages or shifts in product quality. *Contrast mining* [DB12] quantifies and describes the difference between two datasets in terms of the models they induce or the patterns they contain. It provides knowledge on *what* has changed as well as the *extent* of it [Boe11].

Contrast mining is useful in several application contexts. For a *retail manager*, it is paramount to detect whether customer segments have significantly changed since some marketing and sales plans were decided. If he learns that, for instance, the largest customer group used to be single in their early twenties are now married middle agers, he could revise the original plans to match the characteristics of the latter customer group. Another example is a *marketing analyst* akin to know which products (or product categories) can better explain the difference in sales between two nearby stores. Such analysis can then be used to decide whether different marketing strategies are needed for each store. Based on the deviation between pairs of customer transactions, a set of stores can be grouped together and earmarked for the same marketing strategy. In social media marketing (e.g., for tourist or entertainment sectors), analysts would like to know which user demographic segment better explains the difference in online ratings between two products or services. Besides the business domain, several scientific applications benefit from contrast mining. For instance, if a patient's reactions to two competitive or related drugs are found to be statistically different, *biomedical researchers* seek to understand the differences with respect to some factors, such as blood pressure, age, etc.

A common denominator of these examples is the need to understand *if two datasets differ* in some measure of interest (e.g., total sales, rating, etc) and explain *how they differ* (in terms of the underlying demographic groups, product/service characteristics, etc). Given two datasets, D_1 and D_2, that one wishes to contrast, *contrast patterns* [DB12] aim to recognize patterns that best explain the differences between two datasets according to some dimensions (i.e., attributes) of interest [WBN03, BP01a]. Such patterns essentially represent different segments of the two datasets for which a significant difference in a measure of interest (e.g., an aggregated value) is observed. Although contrast mining techniques [ZW03] emphasize on providing interpretable and expressive *contrast patterns*, they fail to parsimoniously describe the set of differences between two datasets. An attempt to solve this problem by providing a minimal number of meaningful contrast patterns is presented in [ABG$^+$07, JBL09]. Both works aim to minimize the number of contrast patterns existing in a hierarchical structure, where the hierarchy is either domain specific [JBL09] (e.g., attribute location described by the hierarchy of state/city/zip_code) or dynamically built [ABG$^+$07]. The main deficiency of these works is that they lack the ability of maximizing the overall strength of reported differences, while constraining the number or granularity (i.e., description length) of data segments.

In our work, we are interested in exploiting the structural relations among multi-dimensional data segments to assess the utility of the contrast patterns included in a parsimonious explanation. To the best of our knowledge this is the first work that formalizes parsimonious explanations as an optimization problem of maximizing the overall strength of differences being explained (i.e., informativeness) constrained by the number or granularity of data segments (i.e., conciseness). By a reduction from the maximum dispersion problem, we show that computing such summaries is an NP-hard problem and thus we propose a nearly optimal greedy algorithm for solving it.

This chapter is organized as follows. Section 3.2 illustrates intuitive examples, motivating the need for parsimonious explanations of differences. Section 3.3 gives background definitions and formalizes our optimization problem. Section 3.3.2 provides proofs of properties of our optimization problem, which guarantee near-optimal approximation solutions for our greedy algorithm. Section 3.4 discusses a greedy approach, as well as a baseline algorithm. Our experimental study and findings are given in Section 3.5. We conclude in Section 3.6.

3.2 Motivating Example

We are given two datasets, D_1 and D_2, that contain retail sales for the same time period of stores s_1 and s_2 respectively. Both datasets are described by the same schema, as shown

in Table 3.1. Each dataset consists of four entities: `Sale`, `Store`, `Product` and `Customer`. Each entity is described by a set of attributes, shown in the second column of Table 3.1.

Entities	**Attributes,** $Attr$
`Sale`	$\langle storeid, productid, customerid, time\rangle$
`Store`	$\langle storeid, latitude, longitude, postcode, location\rangle$
`Product`	$\langle productid, name, category\rangle$
`Customer`	$\langle customerid, date, name, age, gender, profession, region, children, status\rangle$

Table 3.1 Schema of datasets D_1, D_2

Given D_1 and D_2, analysts are usually interested in *detecting whether a significant difference exists between a quantity of interest*, e.g., sales, defined as aggregate values in both datasets and *understanding to which customer segments of the datasets this difference is mainly due*. For example, the aggregated sales of a store s can be defined as $count(\sigma_{storeid=s}(Sales))$ while demographic attributes like *Customer.age* and *Customer.status* can be used to describe customer segments that significantly diverge in sales when contrasting the two stores. Each segment is essentially described by conjunctive predicates over the attributes of the schema of the two datasets: e.g., the segment `age`=*Young* and `status`=*Single* could result in significant differences between the aggregated sales of the two stores. Figure 3.1 illustrates the space of all possible segments, forming a (semi-)lattice (the root corresponds to the trivial predicate *true*) in which edges represent inclusion relationships between the data segments. For instance, segments (`age`=*Young*$\wedge$`status`=*Single*) and (`age`=*Young*), connected with an edge, exhibit an inclusion relation as the former is a subset of the latter. Clearly the conjunction length determines the granularity of the underlying segments, e.g., (`age`=*Young*$\wedge$`status`=*Single*) belongs to the finest granularity, while (`age`=*Young*) determines a coarser one.

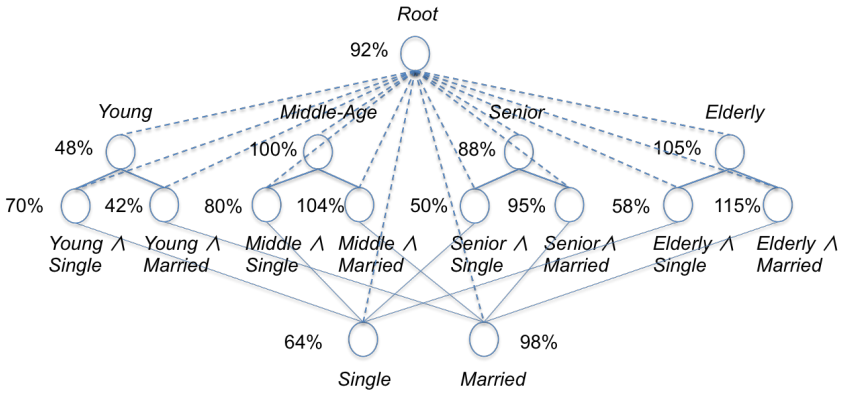


Fig. 3.1 Two dimensional space of data segments

Each segment contributes to the aggregated value of interest and is potentially responsible for the differences observed between the two datasets. Table 3.2 illustrates examples of data segments at various granularities (shown in columns) along with their aggregated values. For the purpose of our example we use the `count` aggregate function, which counts the number of sales in a particular segment and store (other additive *aggregate* functions such as `sum` may be also used). To assess the divergence between the aggregated values computed for each store we rely on a *difference measure*. It can be either absolute, quantifying the actual difference in sales or relative, indicating their percentage difference[1], $PD(s_1, s_2)$, shown in the last row of Table 3.2. Each segment in Figure 3.1 is annotated with the relative percentage difference of its data.

	Root	*'Young'*	*'Middle-Age'*	*'Middle-Age'*$\wedge$*'Married'*
s_1	7,675,342	650,772	2,657,143	2,303,353
s_2	2,803,984	396,701	878,017	727,092
$PD(s_1, s_2)$	92%	48%	100%	104%

Table 3.2 Aggregate values of various data segments

As we can observe in Table 3.2, the percentage difference of aggregated values at coarse granularity levels is at least as much as the percentage difference at finer granularity levels. For instance, the root segment exhibits a difference of 92%, but when its data are further refined with (`age`=*Middle-Age*) segment the percentage difference increases to 100%. The percentage difference is increased even more, reaching 104%, when exploring more fine-grained data segments, e.g., (`age`=*Middle-Age*$\wedge$`status`=*Married*). It should be stressed that descendant segments may exhibit a greater percentage difference than the children of a segment (e.g., `age`=*Middle-Age* vs `age`=*Middle-Age*$\wedge$`status`=*Married*). However, the exploration of more refined granularities does not always guarantee higher and more significant differences. For instance, (`age`=*Young*) which also refines the data of root has a lower difference, 48%. In essence, *percentage difference is not monotonic* with respect to the granularity (i.e., description length) of data segments.

The question that naturally arises in this context is *which and how many data segments exhibiting a significant percentage difference* should be included in the parsimonious summarisation given to an analyst. A simple approach consists in selecting the top-k data segments with the greater percentage difference. However, this solution is prone to producing verbose and redundant information. Consider for instance the top-5 data segments of Figure 3.1, i.e., $S =$ {(`age`=*Elderly*$\wedge$`status`=*Married*), (`age`=*Elderly*), (`age`=*Middle-Age*$\wedge$`status`=*Married*), (`age`=*Middle-Age*), (`status`=*Married*)}. This set

[1] https://en.wikipedia.org/wiki/Relative_change_and_difference

consists of overlapping segments where the percentage difference of a descendant (e.g., age=*Middle-Age*∧status=*Married*) is higher than the percentage difference of its ancestors (e.g., age=*Middle-Age*, status=*Married*). Clearly, there is no need to consider ancestor segments in the summarization, when their descendant already indicate a higher relative difference. Similarly, differences can be summarized by considering only ancestors (e.g., status=*Married*) which exhibit a higher relative difference than their descendants (e.g., status=*Married*∧age=*Young* or status=*Married*∧age=*Senior*). Justifying if descendants or ancestors provide a better summary is related to the distribution of differences among segments at different granularities. In this paper, we are interested in summarizing data segments which exhibit a significant relative difference w.r.t. to their ancestor or descendant segments not included in the summary. Compared to related work [ABG+07, JBL09], this is the first work that accounts for the structural relations among data segments in order to maximize the overall strength of reported differences, while constraining the number or granularity of data segments.

In a nutshell, we make the following contributions:

1. We formalize parsimonious explanations of differences between two datasets as an *optimization problem* of maximizing the overall strength of differences being explained (i.e., informativeness) constrained by the number or granularity of data segments (i.e., conciseness). By reduction from the maximum dispersion problem [GoCBBRD77], we show that finding parsimonious explanations is NP-hard.

2. We propose two scoring functions that capture the overall strength of differences of a parsimonious explanation and exploit the structural relations between the data segments. Then we prove two interesting properties, namely *sub-modularity* and *monotonicity* of scoring functions. These properties allow us to design a greedy algorithm for computing parsimonious explanations with provable near-optimal approximation guarantees.

3. We experimentally evaluate the performance of our algorithm with real datasets containing retail sales of French stores and compare its utility with a baseline algorithm returning only the top-k most differing data segments. We show that gender and profession are those demographic attributes to which significant differences can be attributed and that consumption habits are changing by region (e.g., urban or agricultural area).

3.3 Formal Model for Difference Explanation

We are given two input datasets D_1, D_2 with the same schema $S(a_1, \cdots, a_n)$, where $a_i \in \mathscr{A}$ and $\mathscr{A}$ is the set of all attributes.

Definition 1 (Segment) *A segment $s \in \mathscr{S}$ is described by a set of conjunctive predicates, $s.pred = (a_1 = v_1 \wedge \cdots \wedge a_i = v_j)$, where $a_i \in \mathscr{A}$. We refer to $s.pred$ as the label of s and to the length of $s.pred$ as its number of predicates $|s.pred|$.*

Definition 2 (Segment Score) *A segment $s \in \mathscr{S}$ is assigned a pair of aggregate values, $s.N_1$, $s.N_2$, corresponding to the number of sales for D_1, D_2 respectively. A segment score, $Score(s)$, is defined over the aggregate values, indicating their percentage difference.*

$$Score(s) = \frac{|s.N_1 - s.N_2|}{(s.N_1 + s.N_2)/2} * 100$$

Percentage difference equals the absolute value of the change in $s.N_1, s.N_2$, divided by the average of these two numbers, all multiplied by 100. Given this definition, $Score(s)$ ranges in $[0, 200]$.

Definition 3 (Segment Relation) *Two segments s_i, $s_j \in \mathscr{S}$ described by the conjunctive predicates $(a_{i_1} = v_{i_1} \wedge \cdots \wedge a_{i_k} = v_{i_k})$ and $(a_{j_1} = v_{j_1} \wedge \cdots \wedge a_{j_k} = v_{j_m})$ respectively, exhibit one of the following structural relations:*

- *Inclusion: $s_i \subseteq s_j$, iff $\forall (a_{i_k} = v_{i_k}) | \exists (a_{j_m} = v_{j_m}) : a_{i_k} = a_{j_m}$ and $v_{i_k} = v_{j_m}$*
- *Disjointness: $s_i \cap s_j = \emptyset$, iff $\forall (a_{i_k} = v_{i_k}) | \nexists (a_{j_m} = v_{j_m}) : a_{i_k} = a_{j_m}$ and $v_{i_k} = v_{j_m}$*
- *Overlap: $s_i \cap s_j \neq \emptyset$, iff $\exists (a_{i_k} = v_{i_k}) | \exists (a_{j_m} = v_{j_m}) : a_{i_k} = a_{j_m}$ and $v_{i_k} = v_{j_m}$*

Definition 4 (Segment Distance) *$Dist(s_i, s_j)$ assigns a non-negative value between any pair of segments $(s_i, s_j) \in \mathscr{S}$.*

$$Dist(s_i, s_j) = \left\{ \begin{array}{c} Score(s_j) - Score(s_i), \text{ if } s_j \subseteq s_i, Score(s_j) > Score(s_i) \\ 0, \text{ otherwise} \end{array} \right\}$$

For two segments s_i, s_j such that $s_j \subseteq s_i$ and $Score(s_j) > Score(s_i)$, their distance is defined as the difference of their segment scores. Figure 3.2 illustrates two properties of our distance measure. The first property states that $Dist(s_i, s_j) = 0$ does not necessarily mean that the two segments are identical. This property is referred to as the *identity of indiscernibles* and states that no two distinct segments, e.g., s_i and s_j, exactly resemble

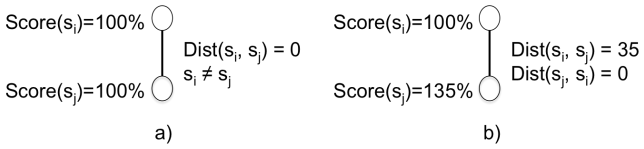


Fig. 3.2 Distance measure properties: (a) Identity of indiscernibles, (b) Asymmetry

each other. The second property mentions that *the distance measure is asymmetric*, hence $Dist(s_i, s_j) \neq Dist(s_j, s_i)$.

The underlying assumption of the above distance measure is to favor segments that maximize their distance from their ancestors. The design of this measure is inspired by the Simpson's paradox [DF05]: a difference appears in descendant segments but disappears or diminishes when these segments are combined to higher granularities of ancestors. In Figure 3.3a the descendant (*Young*∧*Single*) summarizes two of its *ancestors* (*Young*, *Single*). However, a different approach of detecting segments that maximizes their distance from their descendants is also of interest. For this purpose, we simply change the direction of subset relation in the first condition of $Dist(s_i, s_j)$, i.e., if $s_j \supseteq s_i, Score(s_j) > Score(s_i)$. The design of this measure is inspired by cases where differences appear at high granularities of ancestors but decrease in their descendants. In Figure 3.3b the root summarizes all of its *descendants*. Justifying which summary is preferred depends on how differences in data are distributed at different granularities.

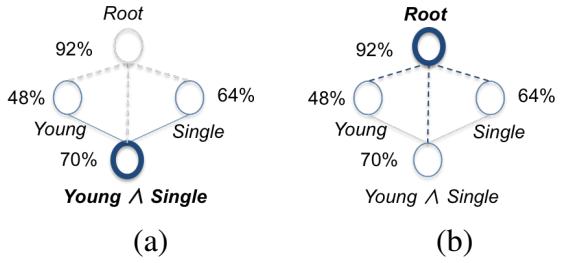


Fig. 3.3 Summarizing (a) Ancestors, (b) Descendants

Definition 5 (Objective Function) *The objective function $f(S)$ assigns a value to a set $S \subseteq \mathscr{S}$, indicating how far segments $s \in S$ are, in terms of the percentage differences they describe.*

$$g_s(S) = \max_{s' \in S} Dist(s, s')$$

$$f(S) = \sum_{s \in \mathscr{S}} g_s(S)$$

For $s \in \mathscr{S}$, the function $g_s(S)$ measures the distance of segment s to the selected set S, as the maximum score of s from any node $s' \in S$. Then, function f measures the distances of all segments $s \in \mathscr{S}$ to the set S, as the sum of their distances. Hereby, we set $f(\emptyset) = 0$.

3.3.1 Difference Explanation Problem

Our problem admits a set of segments $\mathscr{S}$ for two datasets D_1 and D_2 and returns a subset $S \subseteq \mathscr{S}$ of segments such that the percentage difference described by segments in S is as informative as the entire space of segments $\mathscr{S}$. More formally, given a set $\mathscr{S}$ of segments and the distance $Dist(s_i, s_j)$ among segments $s_i, s_j \in \mathscr{S}$, our optimization problem consists of finding a subset S of $\mathscr{S}$, such that the objective function is maximized.

$$\arg\max_{S \subseteq \mathscr{S}} f(S) \textit{ subject to some constraints on } S$$

Our maximization problem is subject to some constraints on S. Two such constraints are:

1. *Cardinality constraint* where we require that $|S| \leq k$, for some given k. Hence, we wish to identify at most k segments that maximize $f(S)$. Note that if k is high, all segments of $g_s(S) > 0, s \in \mathscr{S}$ are included in S.

2. *Granularity constraint* where we require that $|u.pred| \leq p$ (resp., $|u.pred| \geq p$). Hence, we wish to identify segments with at most (resp., at least) p-length label. Note that the highest value of p is the number of attributes $|Attr|$.

Such search problems are reduced from the general category of maximum cut problems. In particular, they are referred to as *dispersion* problems [GoCBBRD77] and they are useful in the context of multi-objective decision making [Osi89]. In multi-objective decision making where the number of non-dominated solutions is high, the decision maker is interested in selecting a smaller number of solutions which are as informative as possible with respect to the values of an objective function.

3.3.2 Sub-modularity and Monotonicity

Sub-modularity has attained an important role in theoretical computer science, due to its beneficial properties in the design of optimization algorithms. The sub-modularity property of a set function can be intuitively interpreted as follows: additional items have less and less value, as the set we possess grows. This property of diminishing returns is useful in several real world problems where diminishing returns naturally arise, like feature selection [LWK+13], facility location [KLG+08] etc. We define the sub-modularity and monotonicity properties and prove them for our objective functions.

Definition 6 (Sub-modularity) *A function* $f : 2^{\mathscr{S}} \rightarrow R$ *is* submodular *if for every* $A \subseteq B \subseteq \mathscr{S}$ *and* $n \in \mathscr{S} \setminus B$ *it holds that*

$$f(B \cup \{n\}) - f(B) \leq f(A \cup \{n\}) - f(A)$$

Definition 7 (Monotonicity) *A function* $f : 2^{\mathscr{S}} \rightarrow \mathbb{R}^+$ *is monotone if for every* $A \subseteq B \subseteq \mathscr{S}$, $f(A) \leq f(B)$.

Proofs

We prove that our objective function (see Definition 5) is sub-modular and monotone.

For $A \subseteq B$ and $n \in \mathscr{S} \setminus B$ we have

$$\max_{v \in B \cup \{n\}} Score(u,v) - \max_{v \in B} Score(u,v) =$$

$$max(0, Score(u,n) - \max_{v \in B} Score(u,v)) \leq max(0, Score(u,n) - \max_{v \in A} Score(u,v)) =$$

$$\max_{v \in A \cup \{n\}} Score(u,v) - \max_{v \in A} Score(u,v)$$

Summing over u we get:

$$f(B \cup \{n\}) - f(B) \leq f(A \cup \{n\}) - f(A)$$

which, by definition, states that function f is submodular. Moreover,

$$f(B \cup \{n\}) - f(B) \geq 0$$

which states that function f is monotone (non decreasing).

3.4 Algorithms for Difference Explanation

In this section we present our greedy algorithm for summarizing differences in data. We provide the pseudo-code of our algorithm and discuss two different approaches for summarizing either ancestors or descendants. Then, we present the `Top-k` algorithm, along with two variations of it properly adapted to summarize ancestors or descendants. These variations serve us as a baseline in the experimental section. For both algorithms we provide a time complexity analysis.

3.4.1 Greedy Algorithm

The `k-Greedy` algorithm (see Algorithm 4) takes as input the set of segments S, the k constraint and returns a subset S of k segments optimizing the objective function. It starts with the empty set S_0 (line 1), and in iteration j, adds the segment $s \in \mathscr{S} \setminus S_{j-1}$ maximizing the marginal gain (or discrete derivative) $f(S_{i-1} \cup \{s\}) - f(S_{i-1})$ (line 3-4).

Algorithm 4 `k-Greedy`

Input: Set of all segments $\mathscr{S}$, cardinality constraint k
Output: Difference summarization S
1: $S_0 \leftarrow \emptyset$
2: **for** $j = 1$ to k **do**
3: $\quad s_i \leftarrow \arg\max_{s \in \mathscr{S} \setminus S_{j-1}} f(S_{j-1} \cup \{s\}) - f(S_{j-1})$
4: $\quad S_j \leftarrow S_{j-1} \cup \{s_i\}$
5: **end for**
6: **return** S_j

In order for our greedy approach to satisfy the *granularity constraint p* of our problem statement, one more condition should be added when inserting a segment to the resulting set. To this end, line 3 changes as follow:

$$s_i \leftarrow \arg\max_{s \in \mathscr{S} \setminus S_{j-1} \wedge |s.pred| \leq p} f(S_{j-1} \cup \{s\}) - f(S_{j-1})$$

Furthermore, the objective function f can maximize any of the score functions, summarizing either ancestors or descendants. According to which segments the algorithm summarizes at each time, we call the algorithm `k-Greedy(Ancestors)` or `k-Greedy(Descendants)`.

Finding efficient policies for this general class of optimization problems is hard. However, when the objective function is monotone and sub-modular, a simple greedy policy attains a $(1 - \frac{1}{e})$-approximation ratio in terms of expected utility [KG12].

Complexity Analysis

At each step, the `Greedy` algorithm looks for the segment s that maximizes the objective function (line 3). This step requires the comparison of all $s \in \mathscr{S} \setminus S_{j-1}$ with all $s' \in S_{j-1}$. Given that the comparison has a constant complexity, this step requires $O(|\mathscr{S}| * |S_{j-1}|)$ time. Since this step is performed k times for $|\mathscr{S}|$ number of segments and the size of each set $|S_{j-1}|$ is at most k, the complexity of the `Greedy` algorithm is:

$$O(k^2 * |\mathscr{S}|^2)$$

3.4.2 Top-k Algorithm

`Top-k` is a well-known, practical and simple algorithm, able to obtain *the k-most important segments, as indicated by our score function*. We use this algorithm as our baseline and compare its results with `k-Greedy`.

As previously, we provide two variations of `Top-k`, namely `Top-k(Ancestors)` and `Top-k(Descendants)`. Both variations rank the results using the same scoring function and thus they report the same set of k most important segments. However, for each variation we provide a different interpretation of whether their their segments summarize ancestors or descendants, respectively. Hence, we are able to detect whether `Top-k` approaches provide explanations of finer (i.e., explaining ancestors) or broader (i.e., explaining descendants) granularities. We measure the aforementioned ability using our objective function $f(S)$ (according to Definition 5).

Complexity Analysis

At each step, `Top-k` algorithm compares two segments in order to provide their between ranking. The total number of comparisons for the entire ranking process is:

$$|\mathscr{S}| * log(|\mathscr{S}|)$$

where $|\mathscr{S}|$ is the total number of segments in the search space.

3.5 Difference Explanation Experiments

In this section, we investigate the performance and effectiveness of our greedy algorithms, `k-Greedy(Ancestors)` and `k-Greedy(Descendants)` in summarizing differences. On the quality front, we perform a thorough evaluation of different datasets and get an insight of how their intrinsic characteristics, such as dispersion or granularity level of differences, affect the algorithms. Then, we perform a more in-depth analysis in order to test `k-Greedy` approaches, by comparing them with `Top-k` approaches, in terms of average segments score, average summarized segments, $f(S)$ value and average explanation length. On the scalability front, we study their time results as we increase the search space and the values of k-constraint. Finally, we present real examples of summaries of differences, extracted from a real retail sales dataset.

3.5.1 Dataset Preparation

Our data contains retail sales derived from the *Intermarché* French stores during the period of *May 2012* to *September 2014*. Each dataset, namely $D_{storeid}$, corresponds to a particular store described by a unique id *storeid*. Particularly, we are interested in contrasting datasets which exhibit the following counter-intuitive property: although the sales of one store (e.g., D_{84803}) are greater than the sales of an other (e.g., D_{5407}), the number of distinct customers is smaller between the former and the latter. Motivated by this observation, we are interested in finding which data segments best explain the difference in sales.

Datasets	Location	Sales	Distinct Customers
D_{58102}	Chelles, France	10.174.746	17.563
D_{83906}	Nogent-le-Rotrou, France	8.725.648	26.990
D_{83202}	Bernay, France	8.676.482	27.809
D_{84803}	Versailles, France	7.675.342	13.842
D_{49506}	Champigny-sur-Marne, France	7.607.802	14.251
D_{18906}	Crécy-la-Chapelle, France	7.413.875	13.395
D_{5407}	Louviers, France	2.803.984	15.842

Table 3.3 Summary of Datasets

More precisely, we are interested in contrasting the aggregated value of sales of the two stores and explain to which customer segment this difference is mainly due. In this experiment, our analysis aims to identify the demographic attributes to which significant differences on customer purchases can be attributed to the two stores.

Our input datasets are generated by joining `Customer` and `Sale` tables, shown in the schema of Table 3.1, using store ids. Differences are explained based on five demographic attributes in the `Customer` table, i.e., `age`, `gender`, `profession`, `children` and `status`. Note that we exclude from our analysis the customer attributes, `customerid`, `name` in order to ensure customers privacy. The five demographic attributes, participating in the analysis, take between two to 21 different values. For instance, although `status` may be *single* or *in relationship*, `profession` takes 21 different options (e.g., *student*, *retired*). All possible combinations of (attribute, value) pairs form a search space of 16.170 demographic segments. The size of the search space is common for any contrasting pair of datasets, as we consider that all pairs are described by the same set of attributes and each attribute exhibits the same domain for any dataset pair.

Experiments were conducted on 2 GHz Intel Core i7, 8 GB RAM, MAC OS.

3.5.2 Summary of Results

We validate the utility of our algorithms by explaining differences in real data. Our experiments in Section 3.5.3, reveal that `gender` and `profession` are those demographic attributes to which significant differences can be attributed. Particularly, *women*, *unemployed* (e.g., *homemakers*, *inactive*, *retired*, *students*) and *self-employed* (e.g., *entrepreneurs*, *liberal profession*) are those groups impacting the most store sales. Moreover, we show that the consumption habits and purchases of customers are affected by the location of each store. Indeed, urban areas are more likely to attract *students* and *singles*, while agricultural regions have increased sales to *farmers*.

Our experimental study in Section 3.5.4 verifies our initial intuition that there is a need to parsimoniously explain differences in real data. Particularly, our experiments reveal that differences dispersed in the structural relations of data can result in qualitative summaries when these relations are taken into consideration. Moreover, when the dispersion in differences exists at fine (resp., coarse) granularities, `k-Greedy(Ancestors)` (resp., `k-Greedy(Descendants)`) is a good option for parsimoniously explaining those differences. `k-Greedy` approaches achieve a good tradeoff between how much difference their demographic segments exhibit and how many other segments they are able to summarize. For instance, `k-Greedy(Descendants)` has a coverage of at least 75% when compared to `Top-k`, by sacrificing just a 25% of average segments score. Moreover, `k-Greedy` approaches, unlike `Top-k`, can adapt to the characteristics of the contrasted data while they can be easily tuned in providing short or long labels. `k-Greedy` approaches also achieve a tradeoff between the quality of their summaries and the granularity (in terms of label length) of their segments. (Section 3.5.5). Finally, the high quality results of `k-Greedy(Descendants)` comes with a computational cost (Section 3.5.6). However, the execution cost remains reasonable (4 minutes) even for large spaces (16.170 segments).

3.5.3 Examples of Difference Explanation

In this section, we explore the utility of our `k-Greedy` algorithms by investigating some real scenarios provided by *Intermarché* analysts. Particularly, we are interested in understanding why stores with similar number of customers, such as stores in *Louviers, France* or *Crécy-la-Chapelle, France*, have significant difference in their product sales. Moreover, we are interested in going beyond this analysis and study whether stores, that overall have similar number of sales, e.g., *Versailles, France* and *Champygny-sur-Marne, France*, do exhibit differences in their sales when considering individual demographic groups.

In an attempt to address the aforementioned analytical questions and extract meaningful summaries, we apply `k-Greedy` over different contrasting stores. Table 3.4 summarizes the results of our greedy approach, when contrasting stores of high difference in their overall sales (higher than 90%). Particularly, we present the results of `k-Greedy(Ancestors)` and `k-Greedy(Descendants)`, as well as the results of the latter for p=2. Each result summarizes differences between the two stores, consisting of five demographic segments (k=5). Similarly, Table 3.5 illustrates summaries of differences when contrasting datasets with a similar number of overall sales, i.e., where their difference is lower than 16%.

Our experiments indicate that `gender` and `profession` are those demographic attributes to which significant differences can be attributed. Even when considering long, short or constrained length labels, these two attributes are very often involved in the parsimonious explanations. Thus, in general we can state that the gender of a customer and what job she does for a living affect her consumption habits and the volume of purchases. To this end, we notice that product sales of stores are positively or negatively affected by the habits of these demographic segments. In particular, the demographic segment, that most parsimoniously explain differences in sales between the various pairs of stores, is the group of *women*. This observation signifies that groups of women are those that cause the difference in sales between the contrasted stores. Taking into consideration that grocery aisles are traditionally considered the domain of women shoppers, observing women in our summary of explanations may not be surprising. Even when the summary contains a demographic segment of *men*, it is worth mentioning that these men are either *old* and *retired*, *single* or *homemaker*. These groups of men are probably managing the house and do the necessary purchases. Moreover, we notice that among the 21 available professions, *unemployed people* (e.g., *homemakers*, *inactive*, *retired*, *students*) and *self-employed* (e.g., *entrepreneurs*, *liberal profession*) are those impacting sales the most. Another significant factor that impacts the consumption habits and purchases of customers is the location of each store. To this end, urban areas are more likely to attract *students* and *singles*, while agricultural regions sell more to *farmers*.

Having a closer look at the first dataset pair, we notice that the store located in *Louviers, France* (D_{5407}) has an impressive difference in its number of sales compared with the store located in *Crécy-la-Chapelle, France* (D_{18906}). Although the former store has 15% more registered customers than the latter, it shows a 3-fold decrease in sales. Thus, we expect that most of the differences in sales between these two stores are due to the higher number of sales in *Crécy-la-Chapelle, France*. Indeed, `k-Greedy(Ancestors)` confirms this assumption as it discovers that *young, single, women with one child being workers* and *young, married, homemaker men with 3 children* contribute to the sales of the second store. However, another less expected explanation of differences is given by the same approach. Particularly, it states

Louviers VS *Crécy-la-Chapelle*			
`k-Greedy(Ancestors)`	<35 ∧ Mme ∧ Worker ∧ 1 child ∧ Single, 193% <35 ∧ Mr ∧ Homemaker ∧ 3 children ∧ In relationship, 200% 50-64 ∧ Mme ∧ Inactive, Other ∧ 5 children ∧ Single, 200% <35 ∧ Mme ∧ Entrepreneur ∧ 2 children ∧ In relationship, 200% >65 ∧ Mr ∧ Worker ∧ 2 children ∧ Single, 200%		
`k-Greedy (Descendants)`	Root No children Cadre Mlle >65	Mme ∧ In relationship, 95% Mlle ∧ In relationship, 130% >65 ∧ In relationship, 121% Mr ∧ In relationship, 79% Mr ∧ Single, 81%	*p*=2
Louviers VS *Versailles*			
`k-Greedy(Ancestors)`	<35 ∧ Mr ∧ Homemaker ∧ 1 child ∧ Single, 200.0% >65 ∧ Mme ∧ Inactive, Other ∧ 3 children ∧ Single, 200.0% 50-64 ∧ Mr ∧ Retired ∧ 5 children ∧ In relationship, 197% 35-49 ∧ Mlle ∧ Entrepreneur ∧ 2 children ∧ In relationship, 195% 50-64 ∧ Mme ∧ Worker ∧ 4 children ∧ Single, 196%		
`k-Greedy (Descendants)`	Root, 92% No children, 147% Cadre, 175% Student, 152% 35-49, 108%	Mr ∧ In relationship, 108% Mme ∧ In relationship, 94% Mlle ∧ In relationship, 110% <35 ∧ Single, 85% >65 ∧ Mme, 99%	*p*=2

Table 3.4 Difference summarization of `k-Greedy` when contrasting stores of high difference in their overall sales

that the last three demographic segments made the most of their purchases in the less popular (i.e., in terms of sales) store of *Louviers, France* causing a difference of at least 190%. Although this store has less registered customers, it is preferred by some demographic groups. Particularly, we discovered that this store has 20% more registered *parents* and 57% more *entrepreneurs* and the percentage of registered entrepreneurs reaches 67% when considering only women.

Moving the scope of our analysis to broader granularities for the same dataset pair (D_{5407}, D_{18906}), generated by `k-Greedy(Descendants)`, provides us with an overview of the most impacting segments while omitting details of very refined granularities. We notice that for unconstrained length labels the root segment is included in the summary. This is not surprising as we are contrasting stores with significant difference in their overall (i.e., root) sales. However, when a constraint is applied (p=2) we notice that the most impacting demographic segments are *women and senior (>65) people in relationship* as well as *men*.

An interesting observation is that although the store in *Crécy-la-Chapelle, France* has 16% less registered *women in relationship*, these women show a tendency to regularly visit the store exhibiting a higher number of purchases than the same demographic group in *Louviers, France*. Similar observations are drawn for registered *men*; although the store in *Crécy-la-Chapelle, France* has 15% fewer registered men, these men make on average more purchases per capita.

The second contrasting pair verifies some of our previous findings. For instance, the first four segments generated by `k-Greedy(Ancestors)`, indicates again that *entrepreneurs* and *parents* prefer *Louviers, France* (D_{5407}) store to the store in *Versailles, France* (D_{84803}). However, we make another interesting remark when examining also the results of `k-Greedy(Descendants)`. The store in *Versailles, France* has a high number of purchases from *students*, *middle-age (35-49)*, *young (<35)* singles and *senior (>65) women*, implying the better services (e.g., university, employment, hospital) provided by urban areas to sensitive social groups. Moreover, the purchasing and consumption habits are subject to the spending capacity and lifestyle in urban areas. Thus, the increased income as well as the more intense pace of life in *Versailles, France* explain the difference in sales between the two stores and the tendency of some demographic segments, such as *middle-age* or *young (<35) singles* to consume more.

Table 3.5 summarizes the differences when contrasting stores with a difference in their sales less than 16%. Simply by looking at this aggregate value, which is always less than 16%, we can conclude that overall there is not a high difference in sales between these stores. However, an in-depth exploration of their different demographic segments results in the opposite conclusion. While there is not a high difference in sales between the stores located in *Chelles, France* (D_{58102}) and in *Bernay, France* (D_{83202}) their sales to particular `age` or `profession` groups are significantly different. In particular, the explanation generated by `k-Greedy(Descendants)` shows that the percentage difference in sales of *old (50-64)* and *young (<35)* people is 59% and 41% respectively. This percentage is increased to 63% when considering *old people in relationship* and 66% for *young ladies*. Moreover, we notice that store in *Bernay, France*, an area known for the diversity and abundance of its agricultural products, has 82% more registered farmers than *Chelles, France* causing 131% more sales in that store.

The second contrasting pair in Table 3.5 concerns stores located in *Versailles, France* (D_{84803}) and in *Champigny-sur-Marne, France* (D_{49506}). The two stores exhibit a difference in their sales which is just 4.7%. However, when more refined segments are explored, such as *parents* or *singles*, the difference is always higher than 30%. Particularly, the store in *Versailles, France* achieves 74% and 60% more sales to *parents of 4 and 5 children*

Chelles VS *Bernay*			
`k-Greedy(Ancestors)`	<35 ∧ Mme ∧ Entrepreneurs ∧ 1 child ∧ Single, 198% 35-49 ∧ Mlle ∧ Cadre ∧ 3 children ∧ In relationship, 200.0% >65 ∧ Mr ∧ Liberal Profession ∧ No children ∧ Single, 200.0% <35 ∧ Mr ∧ Homemaker ∧ 4 children ∧ In relationship, 200.0% 35-49 ∧ Mlle ∧ Employee ∧ 4 children ∧ Single, 200.0%		
`k-Greedy (Descendants)`	Intermédiaire, 88% 50-64, 59% Farmer, 131% Worker, 93% <35, 41%	Intermédiaire ∧ In relationship, 96% 50-64 ∧ In relationship, 63% Intermédiaire ∧ 1 child, 131% <35 ∧ Mlle, 66% Mme ∧ Intermédiaire, 91%	*p=2*
Versailles VS *Champigny-sur-Marne*			
`k-Greedy(Ancestors)`	<35 ∧ Mme ∧ Intermédiaire ∧ No children ∧ Single, 200.0% 50-64 ∧ Mr ∧ Entrepreneurs ∧ 2 children ∧ Single, 199% <35 ∧ Mr ∧ Liberal Profession ∧ 3 children ∧ In relationship, 200.0% 35-49 ∧ Mlle ∧ Student ∧ No children ∧ In relationship, 200.0% 35-49 ∧ Mr ∧ Employee ∧ 4 children ∧ Single, 200.0%		
`k-Greedy (Descendants)`	4 children, 74% Single, 34% 5 children, 60% 1 child ∧ Single, 70% Student, 48%	1 child ∧ Single, 70% 4 children ∧ In relationship, 80% >65 ∧ Single, 70% Inactive, Other ∧ Single, 113% 35-49 ∧ Student, 90%	*p=2*

Table 3.5 Difference summarization of `k-Greedy` when contrasting stores of low difference in their overall sales

respectively than store in *Champigny-sur-Marne, France*. The aforementioned observations indicate that two stores exhibiting similar overall number of sales may differ in their sales to particular segments. This phenomenon, where a difference appears in segments of data but disappears or reverses when these segments are combined, is known as the Simpson's paradox [DF05]. Thus, instead of reporting differences at the highest granularity level that might hide or obfuscate patterns in the underlying data segments, we show that there is a need to further explore the segments in order to reveal those patterns.

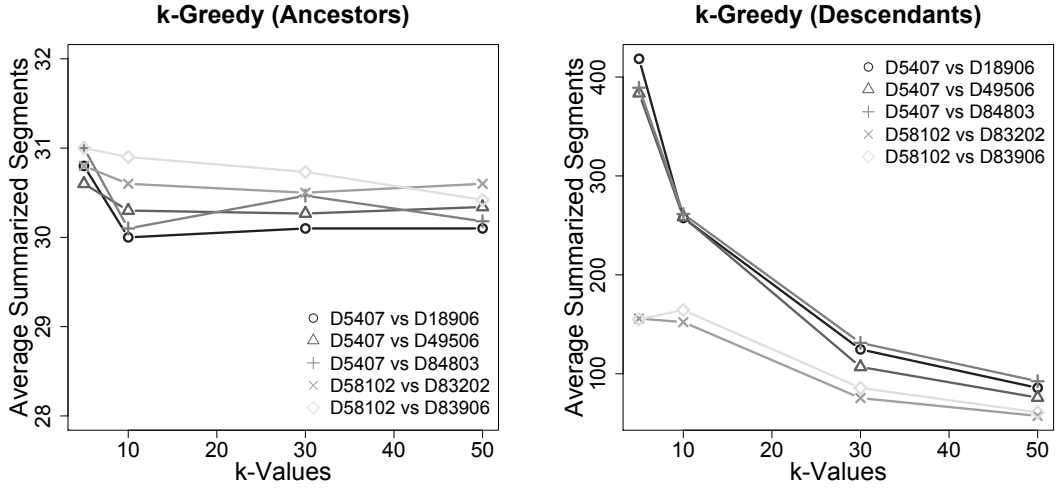
3.5.4 Segment Difference Characterization

Our first set of experiments aims to answer two questions; *does the need to summarize differences in real datasets exist?* and *how the intrinsic characteristics of contrasted datasets (e.g., dispersion, granularity level of difference) impact the quality of summaries?* To study the first question, we perform an analysis on various contrasting datasets and study if the

policy of summarizing ancestors (resp., descendants) can be beneficial for parsimoniously explaining differences in real data. To study the second question, we investigate how the dispersion of differences in data segments at various granularity levels affects the produced summaries. The relative differences are considered to be dispersed when the scores of the corresponding data segments significantly differ as their structural relationships vary. We measure the quality of our algorithms using $f(S)$.

Segment Difference Explanation

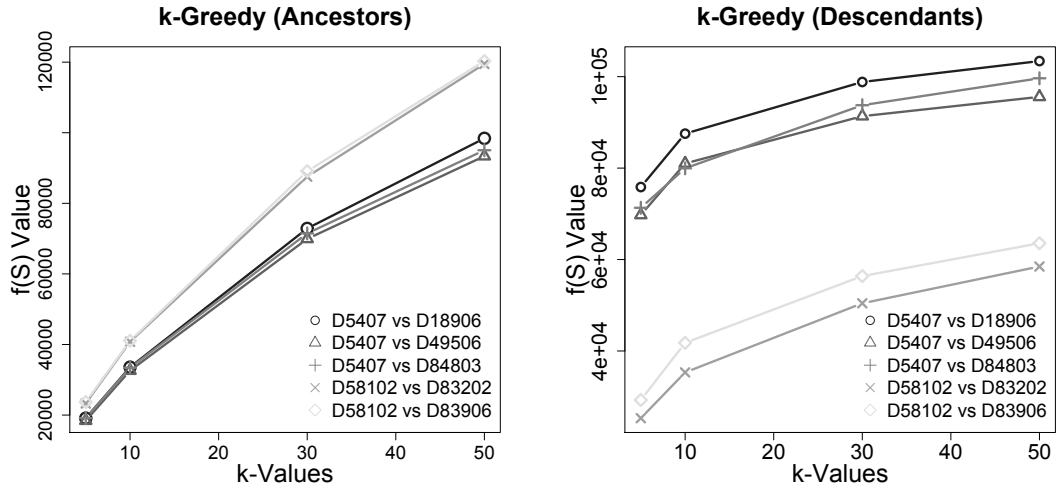
Figure 3.4a depicts the average number of summarized segments (y-axis) generated by `k-Greedy(Ancestors)` for various contrasting datasets described by at most k segments (x-axis). The average number of summarized ancestor segments constantly remains greater than 30 for any k value or pair of datasets. Thus, `k-Greedy(Ancestors)` summarizes 30 ancestors to only five segments, leading to 83% less reported differences. Similarly, Figure 3.4b shows the average number of summarized segments generated by `k-Greedy(Descendants)`. Once more, the average number is significantly high. Less than 10 segments can summarize more than 150 descendants, while in the case of (D_{5407}, D_{18906}), the number of summarized segments reaches 400 descendants. *Figure 3.4 verifies our initial intuition that there is a need to summarize differences and this summarization can be achieved by exploiting the structural relations in data.*



(a) (b)

Fig. 3.4 Average number of summarized segments of various datasets for (a) `k-Greedy (Ancestors)`, (b) `k-Greedy (Descendants)`

Segment Difference Dispersion and Granularity

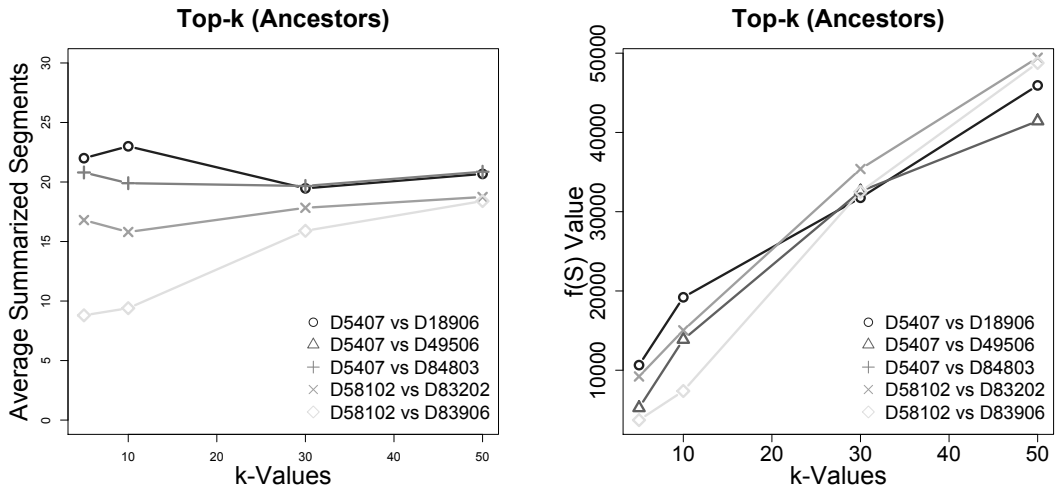


(a) (b)

Fig. 3.5 $f(S)$ value of various datasets for (a) `k-Greedy (Ancestors)`, (b) `k-Greedy (Descendants)`

Figure 3.5a shows the quality of a summary S in terms of $f(S)$ values (y-axis) of `k-Greedy(Ancestors)` for increasing k (x-axis). Overall, we notice that our algorithm's performance increases for any dataset when k increases. This trend is expected since our objective function $f(S)$ is non-negative and non-decreasing. Similar observations are drawn from Figure 3.5b, illustrating the $f(S)$ performance of `k-Greedy(Descendants)`. It is worth mentioning that $f(S)$ increases until it reaches a maximum bound. The maximum bound indicates that the addition of any other segment in S does not offer any further summarization, as all differences have already been summarized by the segments in S (recall Property 1). Finally, we notice that for both greedy algorithms, the higher the average number of summarized segments (Figure 3.4) the better their performance (Figure 3.5). We conclude that, *differences dispersed in the structural relations of data can result in qualitative summaries when these relations are taken into consideration.*

In Figure 3.5a we notice that the last two pairs, (D_{58102}, D_{83202}) and (D_{58102}, D_{83906}), exhibit at least 17% better performance. Their increased performance is explained by the fact that the most informative segments (i.e., segments able to summarize other segments) of these pairs exist at fine-grained granularities. `k-Greedy(Ancestors)` tends to prefer those fine-grained segments, leading to a high performance. Similar observations are drawn from Figure 3.5b. We observe that when descendants are used to summarize differences in (D_{58102}, D_{83906}), the worst $f(S)$ is achieved and this $f(S)$ is decreased by 50% (compared with



(a) (b)

Fig. 3.6 (a) Average summarized segments (b) $f(S)$ value of `Top-k(Ancestors)` for various datasets

`k-Greedy(Ancestors)`) for $k = 50$. On the contrary, for the pair (D_{5407}, D_{18906}), where the most informative differences are met in coarse granularities, `k-Greedy(Descendants)` achieves at least 55% better $f(S)$ than `k-Greedy(Ancestors)`. Therefore we conclude that, *when informative segments exist at fine-grained (resp., coarse-grained) granularities,* `k-Greedy(Ancestors)` *(resp.,* `k-Greedy(Descendants)`*) is a good option for summarizing their differences.*

We perform the same analysis for `Top-k` approaches over the various dataset pairs. Both approaches of `Top-k` result in the same set of top-k segments, as they both perform a ranking based on segments' score. It is worth noticing that in the case where there are several segments sharing the same rank, `Top-k` picks a segment among them by chance. However, the probability to pick a very fine-grained segment (e.g., leaf) is higher than the probability of selecting a very coarse-grained one (e..g., root), since the available number of the former is higher than the latter. To this end, since there are not many coarse-grained segments in top-k set, `Top-k(Descendants)` fails to provide explanations and thus we present only the results of `Top-k(Ancestors)`.

Figure 3.6a shows the average number of summarized segments (y-axis) generated by `Top-k(Ancestors)` for various contrasting datasets with different k constraints (x-axis). The number of summarized segments is relatively high ranging within $[10, 25]$ for any pair of datasets. It is evident that `Top-k(Ancestors)` is not designed to parsimoniously summarize differences and thus it is not able to summarize more ancestors than `k-Greedy(Ancestors)`.

Although its performance is lower than `k-Greedy(Ancestors)`, the necessity to parsimoniously explain differences, by considering hierarchical relations, is again verified.

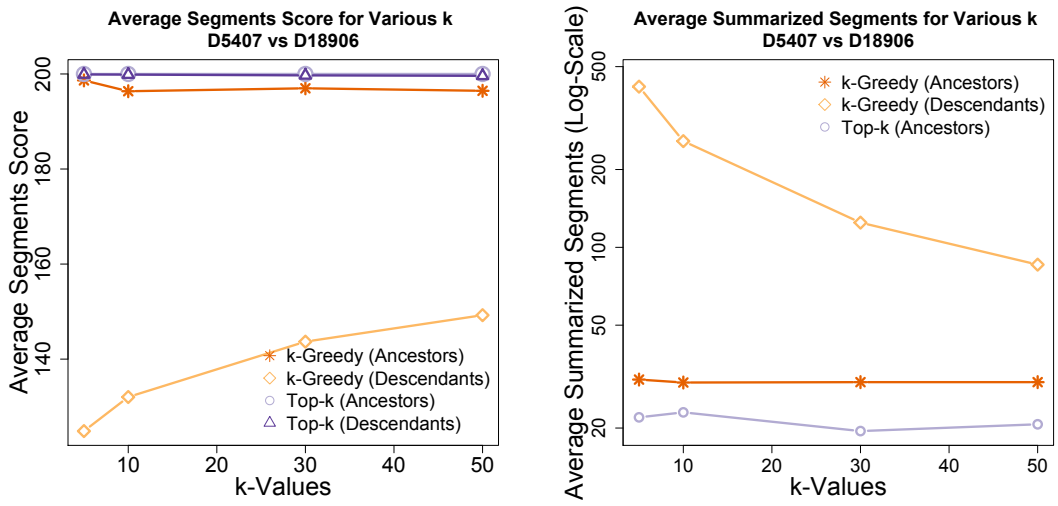
Figure 3.6b illustrates the $f(S)$ (y-axis) of `Top-k(Ancestors)` for various contrasting datasets of different k constraints (x-axis). We verify again that the various dataset pairs have the same trend, when k increases. Moreover, for higher k the performance of `Top-k(Ancestors)` improves. The addition of more segments (higher k) in the resulting summary increases the chance of the algorithm to summarize more ancestors (higher $f(S)$), regardless of the dataset pair. Finally, it is worth mentioning that differences in performance between datasets cannot be observed. The difference in performance is caused by dispersed segments existing at different granularities, summarizing either many ancestors or descendants. Since `Top-k` algorithms do not summarize ancestors or descendants, these differences cannot be observed and the corresponding $f(S)$ value of all datasets is lower than `k-Greedy(Ancestors)`.

3.5.5 Difference Explanation Evaluation

The second set of our experiments aims to perform an in-depth investigation of a single pair of datasets and attempts to answer the question regarding *the quality of summaries w.r.t. a simple summary containing the top-k contrasting data segments*. In this respect we consider two additional quality measures: (a) how significant differences (i.e., average segments score) are summarized by the returned data segments and (b) what is the granularity (i.e., average label length) of the returned segments. The first indicates the *degree to which the segments of a summary describe significant differences in data*. The second captures the *ability of the returned segments to provide refined or broad summaries of the differences in data*. We perform our analysis by contrasting the pair (D_{5407}, D_{18906}). This pair achieves a relatively good $f(S)$ for all algorithms.

Impact of Segments' Score

Figure 3.7a shows the average segment score (y-axis) of a summary S, as generated by each approach, for different values of k (x-axis). `Top-k` gets the highest possible score value (i.e, 200%) which remains stable for all k values, indicating the significance of all segments in S. Similarly, `k-Greedy(Ancestors)` performs well, for any k, describing significant differences which exhibit at least 196% average segment score. On the contrary, `k-Greedy(Descendants)` has a slightly worse performance. It achieves an average segment score of 146% for low k values and this percentage increases to 163% for $k = 50$. It is worth mentioning that `k-Greedy` algorithms do not only aim to optimize the segment score



(a) (b)

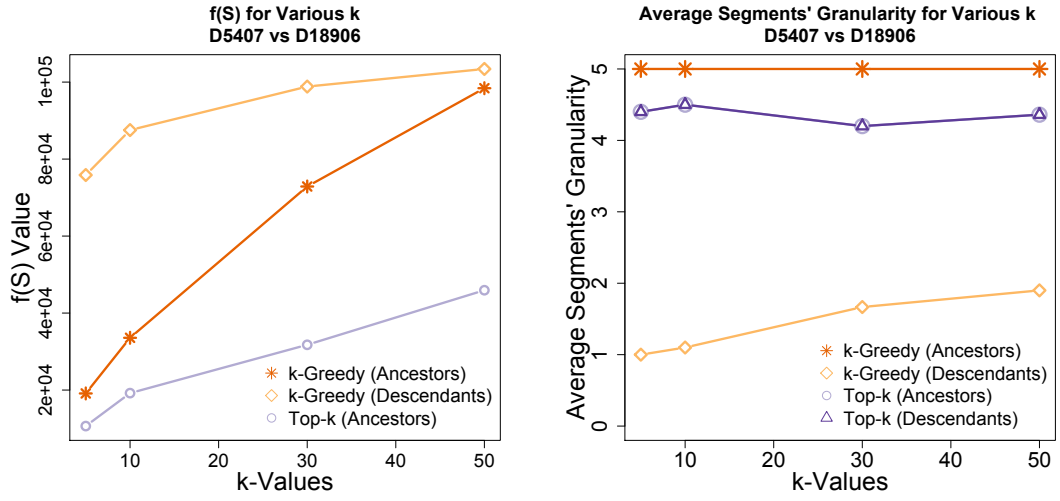
Fig. 3.7 (a) Average segment score of explanation S, (b) Average number of segments summarized by S

(as `Top-k` does), but also the number of other segments being summarized by S. However, `k-Greedy` algorithms perform relatively well, even when considering solely the segments score objective.

Figure 3.7b illustrates the average number of segments being summarized by S (y-axis) for different values of k (x-axis). `Top-k` results in segments with a poor ability in summarizing other segments (`Top-k(Descendants)` cannot even provide any results). On the contrary, `k-Greedy(Descendants)` achieves at least 65% better performance than the others. This result is mainly due to the root segment, which is assigned to S in this particular dataset. The root happens to have the desired property to describe a relatively high difference (90%) and at the same time to summarize many descendants. In particular, since the root has all segments as descendants, each descendant with lower segment score is being summarized by the root and thus the average summarized segments increase. Finally, we notice that *there is a tradeoff between how much difference (i.e., average score) each segment exhibits (Figure 3.7a) and how many other segments are being summarized (Figure 3.7b).* `k-Greedy(Descendants)` *exhibits a good tradeoff, as it generates at least 75% more summarized segments than* `Top-k`*, by sacrificing just a 25% of average segment score.*

Impact of Segments' Granularity

To enrich the quality analysis of our algorithms, we study two more aspects. First, we explore the values of our objective function $f(S)$, as well as the impact that different k-values have



(a) (b)

Fig. 3.8 (a) $f(S)$ for explanation S, (b) Average granularity level of segments in S

on $f(S)$. Then, we study the average segment granularity (i.e., in terms of segment label length) and how it is being affected by the different approaches.

Figure 3.8a depicts the $f(S)$ value (y-axis) for all approaches, when increasing the value of k (x-axis). Overall, we notice that `k-Greedy` approaches achieve a better performance than `Top-k`, regardless of k. `k-Greedy(Descendants)` outperforms the others, since the most informative segments of the pair (D_{5407}, D_{18906}) exist at fine-grained granularities (see Section 3.5.4). It achieves 75% better performance than `Top-k` for $k = 5$, while at the same time maintaining a high $f(S)$ for any k. However, the difference in performance between the various approaches decreases as k increases; since more and more segments are being added to the summary, the different approaches tend to provide similar summaries and thus their performance converges. Figure 3.8a can be used to devise a methodology for selecting the best k, as the k where an elbow in line exists (e.g., $k = 30$ for `k-Greedy(Ancestors)`).

Figure 3.8b illustrates the granularity level of summaries w.r.t. their average label length (y-axis) when varying the values of k (x-axis). Since our approaches are designed to summarize ancestors or descendants, they tend to prefer long and short labels respectively. To this end, `k-Greedy(Ancestors)` provides refined summaries (i.e., maximum label length of five) while `k-Greedy(Descendants)` provides broad summaries (i.e., short label length) for any k. On the contrary, `Top-k` approaches tend to prefer only refined summaries for any k. As there might exist several segments with equal score in the first place, but with different length, `Top-k` is more probable to select a long than a short label (i.e., refined segments of long labels are more numerous than broad segments of short labels). In short, we can state

that `k-Greedy` approaches are flexible and can be easily tuned in providing short or long labels. On the contrary, `Top-k` approaches are not flexible as they tend to prefer longest labels for any k. Finally, *Figure 3.8 highlights a tradeoff between the quality of summaries (in terms of $f(S)$) and the granularity (in terms of label length) of their segments. The more we increase the segments' granularity, the lower the quality of the summary S becomes.*

Impact of p Constraint

Figure 3.9 shows $f(S)$ (y-axis) of our greedy approaches ($k = 30$) for various p values (x-axis). We notice that increasing p has a negative impact on `k-Greedy(Descendants)`, as the summary becomes less informative; longer labels (at least p-length) summarize fewer descendants. In the extreme case, where $p = 5$, this approach returns segments of maximum label length summarizing no descendants and thus having a zero $f(S)$. On the contrary, for `k-Greedy(Ancestors)`, the higher the value of p, the better its performance. It is obvious that allowing the algorithm to reward segments of longer labels, ends up summarizing more ancestors and thus achieving higher $f(S)$. In the extreme case, where $p = 1$, the approach returns segments of minimum label length summarizing no ancestors and thus having a zero $f(S)$. Note that the best performance of `k-Greedy(Ancestors)` ($p = 5$) is 26% lower than the best performance of `k-Greedy(Descendants)` ($p = 1$). This difference is due to the performance boost given by the root.

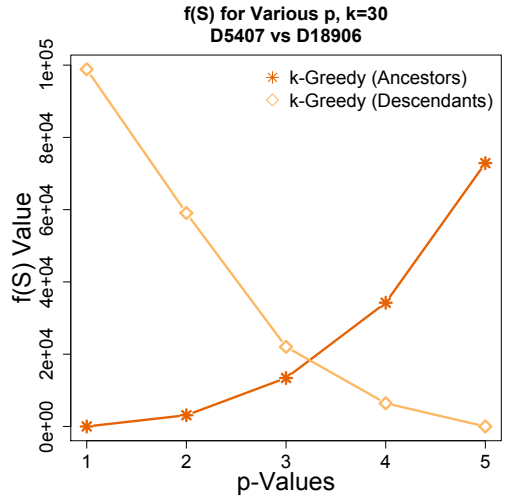
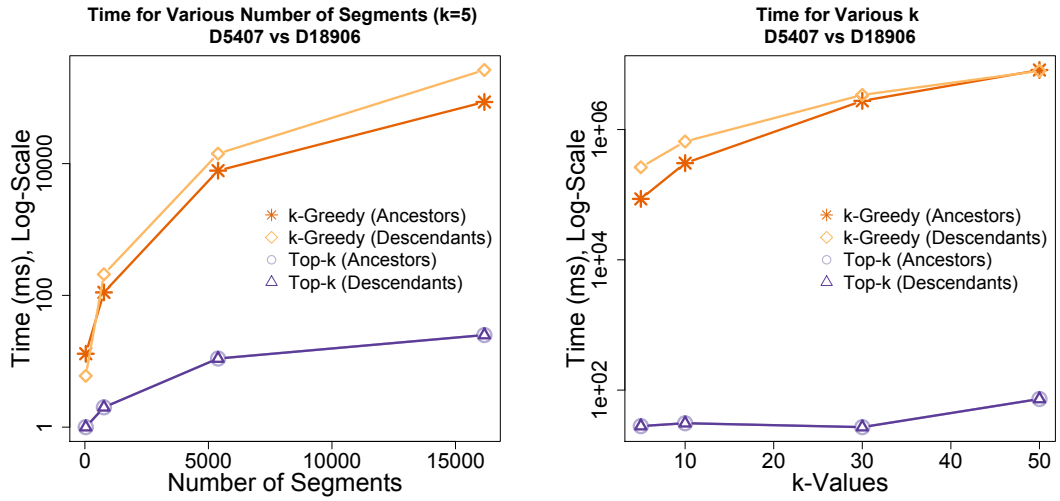


Fig. 3.9 Values of $f(S)$ for various p constraints

3.5.6 Scalability Evaluation



(a) (b)

Fig. 3.10 Response time for various (a) Segments, (b) k-values

In order to study the scalability of our algorithms, we discuss all parameters that may affect the execution time of producing parsimonious explanations. To this end, we show how the size of the search space (i.e., number of data segments) and the values of k impact scalability. Beyond the time to generate the resulting explanations, some time is required to produce the search space. This time is common for all algorithms and increases as the number of segments in the search space increase. We indicatively mention that the time for creating the search space for pair (D_{5407}, D_{18906}) is less than 90 seconds. However, studying or optimizing the time to build the search space is beyond the scope of this work.

An interesting remark is that the running time to generate explanations, required by any algorithm, is not dependent on the input size of the contrasting datasets. All algorithms iterate over the search space, where its building block is a data segment. Thus, all comparisons and decisions of whether a segment belongs to a parsimonious explanation are made over data segments and not over individual tuples.

Figure 3.10a illustrates the running time in logarithmic scale of milliseconds (y-axis) of all approaches (for k=5) when varying the size of the search space (x-axis). We notice that increasing the search space causes an increase of time, for any approach. As more data segments are available and thus more comparisons are required, `k-Greedy` approaches become more time demanding. `k-Greedy(Descendants)` requires more time than `k-Greedy(Ancestors)` to produce a parsimonious explanation. In order to explain the difference in time between the two approaches, we need to make two remarks. First, a compar-

ison between two segments is performed only if they exhibit an inclusion relation. Thus, the more relations segments in an explanation exhibit with segments in the search space, the more costly an approach becomes. Second, we remind that `k-Greedy(Descendants)` includes the root segment in the parsimonious explanation. The root exhibits the maximum possible inclusion relations with segments in the search space. Thus `k-Greedy(Descendants)` needs to perform many comparisons explaining its time requirements. On the contrary, `Top-k` approaches are more efficient than `k-Greedy`, as they require time only for ranking segments. The ranking process becomes more demanding as we increase the search space.

Figure 3.10b shows the running time in logarithmic scale of milliseconds (y-axis) of all approaches for the largest search space when varying the k-constraint (y-axis). Increasing the values of k has a negative impact on the scalability of `k-Greedy` approaches. For instance, when k is lower than 30, our approaches require few minutes in order to provide a parsimonious explanation. However, when k increases to more than 30, they require at least two hours. On the contrary, `Top-k` approaches are more efficient exhibiting an almost stable time requirement when k increases. We can understand this behavior if we consider that `Top-k` needs time to perform two steps. At the first step it ranks the search space based on segments score and at the second step it selects the k-segments with the highest score. The time required for the first step is common for any k. The time required for the second step slightly increases as k increases, explaining the general tendency of `Top-k`. Finally, we notice that although `k-Greedy(Descendants)` has the best quality, in terms of $f(S)$, it requires at least three orders of magnitude more time to produce the results. *Thus, there is a tradeoff between obtaining segments of high score and the time required for building them.*

3.6 Summary of Difference Explanation

In this work we propose a novel framework for summarizing differences between two multi-dimensional datasets that exploits the structural relationships of the various data segments. We formulate our problem as maximizing the overall strength of differences being explained (i.e., informativeness) constrained by the number or granularity of data segments (i.e., conciseness). We propose a simple greedy algorithm for computing summaries of differences with a theoretical guarantee of $(1-\frac{1}{e})$-approximation. We show that there is a tradeoff between the low cost of obtaining the top-k differing segments and the analytical value of summarizing differences by our greedy algorithm. Our experiments indicate that `gender` and `profession` are those demographic attributes to which significant differences can be attributed and that consumption habits are changing by region (e.g., urban or agricultural area). Given the quadratic complexity of the algorithm in the number of segments (and thus

of the data dimensions), we are currently studying heuristics that effectively prune our search space of segments (local search) without severely penalizing the approximation guarantees.

Chapter 4

Difference Evolution

4.1 Introduction

Monitoring streaming content is a challenging big data analytics problem, given that very large datasets are rarely (if ever) stationary. In several real world monitoring applications (e.g., newsgroup discussions, network connections) we need to detect significant change points in the underlying data distribution (e.g., frequency of words, sessions) and track the evolution of those changes over time. These change points, depending on the research community, are referred to as *temporal evolution*, *non stationarity*, or *concept drift* and provide valuable insights on real world events (e.g. a discussion topic, an intrusion) to take a timely action. In this work, we adopt *a query-based approach to drift detection* and address the question of processing *drift queries* over very large datasets. To the best of our knowledge, our work is the first to formalize flexible drift queries on streaming datasets with varying change rates.

In the problem of drift detection, given a number of m drifts ordered in time, we need no less than $m+1$ intervals to detect them. Thus, without any assumption on the underlying distribution, we are interested in exploring how to segment the input stream in order to find a reasonable tradeoff between true positives and false negatives. Existing methods rely on segmenting the input stream, mostly into smaller fixed length intervals [BGP10, Ozo08, JMG95, KBDG04, VB09]. Although some works exist on partitioning the same stream into intervals of different granularities [Bif10, GMCR04, WK96], they either adopt an offline analysis or they lack the ability of querying historical drifts in streams at multiple granularities.

A granularity in this case is an interval of time (e.g., every hour) or a number of observed data points (e.g., every 200 points). A drift is then defined as a significant difference in data distributions between two consecutive intervals at the same granularity. To detect drifts either

statistical tests are directly applied on the data of two intervals [DR09, KBDG04] or on their summaries, as for instance provided by a clustering algorithm [AHWY03, BGP10, CEQZ]. To this end, two parameters impact the accuracy and efficiency of drift detection: the granularity of the intervals at which the original data items are clustered and the drift significance threshold used to assess whether or not there is a drift between two consecutive clusterings. In fact, fine-grained intervals can be used to capture the evolution of frequently changing streams. However, they may induce computation overhead for slowly changing ones. In addition, they may cause false positives, i.e., detecting drifts that are too sudden and noisy, hence hurting precision. While a coarser granularity will improve precision, since more data is clustered in each interval, it may incur missing a drift that occurred at a finer granularity. Those misses will negatively affect recall. Moreover, the rate of change of a given dataset may vary over time thereby requiring to consider different clustering granularities and drift thresholds for the same dataset.

Understanding the tradeoff between precision (at higher segmentation granularities) and recall (at lower segmentation granularities), and the choice of thresholds to determine what constitutes a drift between two consecutive intervals of the same granularity, are the main objectives of this work. We adopt an analytics approach in which we formalize drift queries over both fresh and historical data of arbitrary time granularities, in order to provide flexibility in tracking and analyzing drifts in evolving datasets. For this reason, we propose a flexible drift index to organize past data (or more precisely their summaries) at several granularities. Furthermore. we explore different creation strategies for this index relying on two common clustering approaches, namely independent [EKSX96, OMA+01, ZRL96] and cumulative [AHWY03, CEQZ]. In independent clustering, data points belonging to a given interval are considered equally important and clustered independently. In cumulative clustering, data points in a given interval are clustered with all previously occurring points and fresher data is more important than older data. Moreover, we propose different materialization strategies in order to explore the tradeoff between index storage and query response time.

Unlike existing approaches [BGP10, Ozo08, JMG95, KBDG04, VB09] comparing only the last most recent intervals, we exploit this index in order to identify drifts at different granularities. In particular, we formalize three kinds of queries: unary, refinement and synthesis aiming to detect drifts against historical data. A unary query is used to extract all drifts detected at a given granularity. A refinement query explores drifts from a source granularity (e.g., 5,000 points) to a finer target granularity (e.g., 500 points), iteratively. Such a query is useful to provide a more detailed description of drifts that have been detected in a high granularity, resulting in better recall. Synthesis queries, on the other hand, start from a relatively low granularity and summarize them into coarser ones. In this case, some of the

particular details might be missed (low recall) in order to get drifts with higher precision. This flexibility in querying drifts allows us to explore, in a declarative fashion, precision and recall tradeoffs at different granularities. Also, it addresses a long standing concern in detecting and tracking drifts in streaming content, namely adaptability of drift detection to different drift arrival rates and types.

The evaluation of declarative drift queries relies on traversing the index of historical data summaries and, at each granularity, comparing its nodes pairwise to identify points where clusterings dissimilarity exceeds a threshold θ. Rather than setting drift thresholds a-priori [Ozo08, SG07], we learn a θ-value for each dataset and at each granularity level in the index.

In summary, this chapter makes the following contributions:

1. We introduce and formalize drift queries that provide high flexibility in analyzing precision and recall of drift detection at different time granularities.

2. We propose a drift index, a graph structure that captures change at different granularities and explore different materializations of the index that lead to the design of various index maintenance and query evaluation algorithms.

3. We propose learning algorithms for learning drift and clustering thresholds adaptively for different granularities and rates of change.

4. We perform a thorough study of proposed queries and indices using two real datasets, KDD Cup'99 [1] and Usenet [KTV06], and a synthetically generated dataset. On the effectiveness front, our study confirms the need for our refinement and synthesis queries, as demonstrated by the very good precision/recall results they attain. On the scalability front, it validates the need for different materializations of the drift index in order to achieve a tradeoff between storage and query response time for datasets of varying change rates.

The chapter is organized as follows. Section 4.2 defines our data model and queries. Section 4.3 describes the drift index, the online index maintenance algorithms and threshold learning. Section 4.4 is dedicated to query evaluation algorithms. Section 4.5 contains a description of our experiments and findings. We conclude in Section 4.6.

4.2 Data Model and Queries

We are given a stream of data points $D = \{d_1, \ldots, d_i, \ldots\}$, $d_i = (tc_i, ts_i)$ where tc_i is an r-dimensional vector of attributes describing d_i and ts_i is the timestamp at which tc_i arrived.

[1] *https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data*

For example, on Usenet each attribute represents a term appearing in news feeds, while on KDD Cup'99 an attribute can be any feature (e.g., transmitted bytes, duration of connection) describing a connection record.

4.2.1 Clustering and Drifts

In this section we give some basic concepts and definitions to be used in this work. We discuss the differences between *time-based* and *point-based* granularities and intervals and we use the notion of *clustering granularity* in order to define the *clustering dissimilarity* measure.

Definition 8 Time-Based and Point-Based Granularities. *A time-based granularity g is an interval of time. For example, g could be hourly, daily, bi-daily, or weekly. A point-based granularity g is an interval containing a fixed number of consecutive data points. For example, g could be 500 points or 1000 points.*

We assume a total order between granularities and use $g \prec\prec g'$ to denote that g' follows g. We also say that g' is coarser than g (and g is finer than g'). We write $g \prec g'$ to denote that g' immediately follows g when there does not exist a granularity between g and g'. In both cases, $g <> g'$.

Definition 9 Time-Based and Point-Based Intervals. *Granularities are used to segment data points in D. A segmentation of a dataset D using a time-based or a point-based granularity g, results in a list of consecutives intervals denoted I_1^g, I_2^g, ... where I_i^g is the $i-th$ interval (time-based or point-based) of granularity g.*

For a daily granularity g applied to segment a week starting on Sunday, I_1^g corresponds to the time interval of Sunday. Respectively, I_1^g corresponds to the interval containing the first 500 data points when a point granularity $g = 500$ is used to segment incoming data.

The choice of point-based or time-based intervals to segment a dataset depends on the rate of arrival of data points. The main advantage of point-based intervals is the processing of data in fixed-size batches (in terms of number of points) although resulting intervals may have different lengths (in terms of time). Time-based intervals on the other hand, give the ability to tune the time granularity of the analysis (e.g., hour, day) resulting in fixed-length intervals (in terms of time) and varying in size (in terms of number of data points). Consequently, in order to generate intervals of comparable size, datasets that exhibit a high changing rate should be segmented with point-based intervals while more stable datasets can be segmented using time-based intervals.

Definition 10 Granularity Clustering. *A granularity clustering $C^g(D)$ is a partitioning of all data points $d_i \in D$ into a set of clusterings $\{C_i^g, C_{i+1}^g, \ldots\}$ corresponding to consecutive, non-overlapping time intervals $\{I_i^g, I_{i+1}^g, \ldots\}$ at granularity g. A data point $d_i = (tc_i, ts_i)$ will belong to one interval I_j^g s.t. $ts_i \in I_j^g$. Here, tc_i is a vector of r-entries with the j-th entry corresponding to the weight of the j-th attribute (e.g., word). Each cluster $c \in C_i^g$ has a centroid, $center(c)$ which is itself an r-dimensional vector, where the j-th entry is the mean over the j-th entries of all data points in the cluster.*

Definition 11 Clustering Dissimilarity. *Given two clusterings C_i^g and C_j^g, we define, $d^2(c,c')$, the dissimilarity between a cluster $c \in C_i^g$ and a cluster $c' \in C_j^g$ as the Euclidean distance between their centroids:*

$$\|center(c) - center(c')\|_2 \tag{4.1}$$

The dissimilarity between cluster $c \in C_i^g$ and a clustering C_j^g, $cdis(c, C_j^g)$, is defined as the closest cluster to c in C_j^g:

$$\arg\min_{c' \in C_j^g} d^2(c,c') \tag{4.2}$$

The dissimilarity between two clusterings, $dis(C_i^g, C_j^g)$, is defined as:

$$\frac{1}{|C_i^g|} \sum_{c \in C_i^g} cdis(c, C_j^g) + \frac{1}{|C_j^g|} \sum_{c' \in C_j^g} cdis(c', C_i^g) \tag{4.3}$$

where $|C_i^g|$ is the number of data points belonging to C_i^g.

Definition 12 Drift. *For a dataset D, a granularity g, a threshold θ, we say that there is a drift between two consecutive intervals I_i^g and I_{i+1}^g, iff their associated clusterings C_i^g and C_{i+1}^g, satisfy $dis(C_i^g, C_{i+1}^g) \geq \theta$. We use $x_i^g = (I_i^g, I_{i+1}^g)$ to denote the pair of consecutive intervals for which there exists a drift and $X^g = \{x_1^g, x_2^g, \ldots\}$ for the set of all drifts detected at granularity g.*

4.2.2 Drift Queries in Difference Evolution

The goal of drift queries is to compare drifts at different granularities and provide analysts with the ability to explore drift precison and recall across granularities. We study two kinds of queries, *refinement* and *synthesis*. Both kinds rely on a simpler *unary query* defined as follows.

Definition 13 Unary Query. *A unary query $UQ(D,g)$ returns the set of all drifts X^g detected at granularity g for a dataset D.*

Definition 14 Refinement Query. *A refinement query $RQ(D,g_s,g_t)$ admits a source granularity g_s and a target one g_t s.t. $g_t \prec\prec g_s$, and returns a set of pairs $(x_i^{g_s},x_j^g)$ where each drift $x_i^{g_s} \in X^{g_s}$ at g_s is associated to the finest corresponding drift $x_j^g \in X^g$ at a granularity g no finer than g_t as follows:*

$$\{x_j^g \in X^g, g_t \prec\prec g \prec\prec g_s \vee g = g_t \mid$$
$$\exists x_i^{g_s} \in X^{g_s}, I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s}),$$
$$\nexists x_k^{g'} \in X^{g'}, g_t \prec\prec g' \prec\prec g \vee g' = g_t, I_k^{g'} \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})\}$$

where $I_i^{g_s} \cup I_{i+1}^{g_s} = [\min_{ts_j \in I_i^{g_s}}(ts_j), \max_{ts_k \in I_{i+1}^{g_s}}(ts_k)]$ and
$I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ if $\min_{ts_j \in (I_i^{g_s} \cup I_{i+1}^{g_s})}(ts_j) \leq \min_{ts_k \in I_i^{g_s}}(ts_k)$ and $\max_{ts_k \in I_i^{g_s}}(ts_k) \leq \max_{ts_j \in (I_i^{g_s} \cup I_{i+1}^{g_s})}(ts_j)$

Refinement queries provide a detailed analysis of drifts iteratively. For instance, for a source granularity $g_s = 1000$ connections on KDD Cup'99, selecting a granularity $g_t = 500$ might result in missing a more insightful analysis occurring at granularity $g_t = 100$. On the other hand, selecting $g_t = 100$ may result in retrieving false positives which could be avoided at $g_t = 500$. Therefore, the analyst will use the refinement query $RQ(D,1000,100)$ to obtain details of each drift at $g_s = 1000$ with a tradeoff between false negatives and false positives.

Definition 15 Synthesis Query. *A synthesis query $SQ(D,g_s,g_t)$ admits a source granularity g_s and a target one g_t s.t. $g_s \prec\prec g_t$, and returns a set of pairs $(x_i^{g_s},x_j^g)$ where each drift $x_i^{g_s} \in X^{g_s}$ at granularity g_s is associated to the coarsest corresponding drift $x_j^g \in X^g$ at a granularity g no coarser than g_t as follows:*

$$\{x_j^g \in X^g, g_s \prec\prec g \prec\prec g_t \vee g = g_s \mid$$
$$\exists x_i^{g_s} \in X^{g_s}, I_i^{g_s} \subseteq (I_j^g \cup I_{j+1}^g),$$
$$\nexists x_k^{g'} \in X^{g'}, g \prec\prec g' \prec\prec g_t \vee g' = g_t, I_i^{g_s} \subseteq (I_k^{g'} \cup I_{k+1}^{g'})\}$$

Synthesis queries provide a summary analysis of drifts iteratively. For instance, for a source granularity $g_s = 100$ connections on KDD Cup'99, selecting a granularity $g_t = 1000$ might result in missing a more precise synthesis occurring at $g_t = 2000$. On the other hand, selecting $g_t = 2000$ can result in missing a summary of a drift, which could be obtained at $g_t = 1000$. Therefore, the analyst can use the synthesis query $SQ(D,100,2000)$ to obtain a summary of each drift at $g_s = 100$ with a tradeoff between false negatives and false positives.

4.3 Drift Index

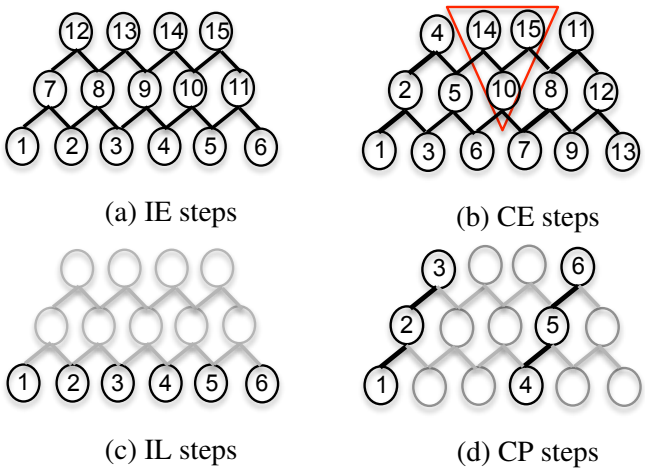
The flexibility of querying drifts at different granularities requires the design of appropriate data structures able to capture clusterings at different granularities in such a way that queries are evaluated efficiently. In this section, we describe the *drift index*, an efficient graph data structure that is used to store and compute clusterings at different granularities. We first formalize the index and then study several materializations and develop algorithms for incremental index maintenance as data points continue to arrive.

Definition 16 Drift Index. *The drift index is an undirected graph $G = (V, E)$ where each node contains a clustering C_i^g of points in D during interval I_i^g of granularity g. Given two different granularities g and g' s.t. $g \prec g'$, and two intervals I_i^g of granularity g and $I_j^{g'}$ of granularity g', there exists an edge in G between nodes C_i^g and $C_j^{g'}$ if $I_i^g \subseteq I_j^{g'}$.*

Definition 16 does not necessarily impose an edge between nodes C_i^g and $C_j^{g'}$ every time $I_i^g \subseteq I_j^{g'}$ is satisfied. Indeed, different *materializations* of the index may be explored. The choice of which nodes to materialize affects three parameters: (i) the index size and hence the time it takes to build and maintain it as new data points arrive, (ii) the query response time, and (iii) the accuracy of query results. Since our approach is to serve queries for any time period, and not only the latest period at which data points arrived, the index continuously grows in size. Therefore, the key question we address in designing the index is: what are possible index materialization strategies, how much space they consume and how do they affect query evaluation (response time and accuracy)? In this section, we study index materialization alternatives. The impact of each index on query evaluation will be discussed in Section 4.4. In all our indices, the smallest granularity, g_{min}, is used to generate leaf-level nodes. In Section 4.5, we experiment with different values of g_{min}.

4.3.1 Full Index Materialization

When fully materialized, the drift index is a hierarchical structure where each level contains clusterings of data points inside intervals of the same granularity. Nodes corresponding to the finest granularity are leaves in the graph and each node, except nodes at the coarsest granularity, has one or two parents. For example, a node containing a clustering of data points for a 1-day granularity, e.g., Monday, will have two parent nodes each of which corresponds to a two-day granularity, in this case, Sun-Mon and Mon-Tue. Similarly, a node containing 1000 data points will have two parents, one containing it with the previous 1000 points and another containing it with the following 1000 points. More formally, given two granularities



(a) IE steps (b) CE steps

(c) IL steps (d) CP steps

Fig. 4.1 Drift Index variants

g and g' s.t. $g \prec g'$, and two intervals I_i^g and $I_j^{g'}$, there exists an edge from node C_i^g to node $C_j^{g'}$ iff $I_i^g \subseteq I_j^{g'}$.

Each node of the index contains a clustering of data points of a given granularity. Thus, an important aspect of index materialization is the selection of a clustering strategy and algorithm to generate the nodes. According to the clustering literature, timestamped datasets can be clustered in one of two ways. The first one, referred to as *the independent strategy*, encompasses a family of algorithms built upon the idea of visiting consecutive batches of data points by considering them as independent (e.g., one batch for Mon and another for Tue) and equally important in terms of arrival time (e.g., older data points are not penalized against fresher ones) [OMA+01, EKSX96, ZRL96]. The second approach, referred to as *the cumulative strategy*, parses data in a cumulative, single-pass fashion (e.g., data of Tue are clustered with those of Mon), aging older data points in a such a way that fresher data points are given more importance [AHWY03, CEQZ]. Our exhaustive index is designed to work with any of the two clustering strategies which gives rise to two indices: Independent-Exhaustive (IE) and Cumulative-Exhaustive (CE).

Independent Exhaustive Index

IE is generated using the independent strategy where nodes of the same granularity, e.g., Sun, Mon, Tue, are produced using data points of consecutive, non-overlapping intervals. Nodes at the finest granularity level are produced by clustering the arriving data points, while nodes at coarser granularities are produced by summarizing the centroids of clusterings associated with lower granularities.

Figure 4.1a illustrates an instance of IE with nodes numbered in the order they are created. Algorithm 5 summarizes the different steps for building and maintaining the index. The algorithm takes as input the drift index G (empty at the beginning, non-empty in the case of index maintenance), a data stream D, a maximum granularity g_{max}, and an interval $I^{g_{min}}$ of minimum granularity g_{min}. For each batch of data point inside $I^{g_{min}}$, a clustering is produced (lines 3-6) using an independent clustering algorithm (e.g., DBScan [EKSX96] or k-means [HW79]). Nodes 1 to 6 of Figure 4.1a are produced by this step. Then, nodes at coarser granularities (e.g., nodes 7-15) are generated by applying the same algorithm over the centroids of clusters at lower granularity (lines 7-12).

Algorithm 5 IE Creation & Maintenance

Input: Drift index G, Stream of data points D, Max granularity g_{max}, Interval $I^{g_{min}}$ of min granularity g_{min}
Output: Updated drift index G'

1: $G' \leftarrow G$
2: $\{I_1^{g_{min}}, I_2^{g_{min}}, ...\} \leftarrow$ consecutive intervals at g_{min}
3: **for all** $I_i^{g_{min}}$ **do**
4: $\quad C_i^{g_{min}} \leftarrow$ clustering in $I_i^{g_{min}}$ (e.g., DBScan, k-means)
5: $\quad$ Store $C_i^{g_{min}}$ in G' at granularity g_{min}
6: **end for**
7: **for all** $g_{min} \prec g \prec\prec g_{max}$ **do**
8: $\quad$ **for all** $C_i^g \in C^g$ **do**
9: $\quad\quad C \leftarrow$ clustering of the centroids of C_i^g and C_{i+1}^g
10: $\quad\quad$ Store C in G' as the right parent of C_i^g
11: $\quad$ **end for**
12: **end for**
13: return G'

Cumulative Exhaustive Index

CE is generated using the cumulative strategy where nodes of the same granularity, e.g., Sun-Mon and Mon-Tue, are produced using data points from overlapping intervals. As a result, data points belonging to a given interval are clustered with previously occurring data points.

Figure 4.1b shows an instance of CE with its nodes numbered in the order they appear. Algorithm 6 summarizes steps of building and maintaining CE. The algorithm takes as input a drift index G (empty at the beginning, non-empty in the case of index maintenance), a data stream D, as well as the maximum granularity g_{max}, and the minimum granularity interval $I^{g_{min}}$ at g_{min}. After the initialization steps (lines 2-5), each data point is assigned to a cluster

Algorithm 6 CE Creation & Maintenance

Input: Drift index G, Stream of data points D, Max granularity g_{max}, Interval $I^{g_{min}}$ of min granularity g_{min}
Output: Updated drift index G'

1: $G' \leftarrow G$
2: **if** G' is empty **then**
3: $C \leftarrow$ clustering of D (e.g., DBScan, k-means)
4: store C in G'
5: **end if**
6: $g \leftarrow g_{min}$
7: **for all** $d_i \in D$ **do**
8: $C \leftarrow$ clustering of d_i (e.g., CluStream)
9: **if** $(ts_i - ts_1)$ % $I^{g_{min}} == 0$ **then**
10: **if** $g \prec\prec g_{max}$ **then**
11: store C in G' at g-th granularity
12: **if** $\cup_{i=1,2..} I_i^g > g_{max}$ **then**
13: Build inverse triangle at g, by merging each node's left child with right-most child at g_{min}
14: **end if**
15: Move g to immediately following granularity
16: **else**
17: $C' \leftarrow C - C_j^{g_{max}}$ %Initializes path by subtracting the last stored $C_j^{g_{max}}$ from current clustering C
18: Store C' in G' at granularity g_{min}
19: Move g to granularity immediately following g_{min}
20: **end if**
21: Create right sub-path of node C
22: **end if**
23: **end for**
24: return G'

(line 7-8). Then every $I^{g_{min}}$ number of points (line 9), a corresponding clustering is generated and its centroid is stored in the index (line 11) at granularity g, which is incremented until g_{max}. Nodes 1, 2 and 4 of Figure 4.1b are generated by this step. When the maximum granularity g_{max} is reached a new path is initialized starting from the smallest granularity (line 16-21) and the process is repeated. Node 7 initializes this new path. In addition, for every node added in the index, its right sub-path to g_{min} is also produced (line 21). For instance, after the addition of node 4 in Figure 4.1b, its right sub-path consisting of nodes 5 and 6 is also added. These nodes are produced by applying the subtractive property [AHWY03] of clusters (i.e., subtracting clusters centroids). Finally, there is a set of nodes that do not belong to any right sub-path (e.g., nodes 10, 14, 15), forming the inverse triangle of Figure

4.1b. Each one of these nodes is generated after the addition of its right sibling and its sibling's sub-path. Lines 12-14 of Algorithm 6 illustrate this process using the additive property [AHWY03] of clusters (i.e., adding clusters centroids).

To handle infinite streams, several deletion strategies can be provided. A naive approach is to remove x intervals every X data points, including all corresponding nodes. However, this approach misses valuable historical data. For this reason, we consider an alternative deletion policy in which for every X data points, the oldest x intervals are deleted and only their highest available node in the index is kept.

4.3.2 Partial Index Materialization

Since fully materialized versions of the drift index are expected to consume a lot of space, we propose Independent-Leaf (IL) and Cumulative-Path (CP) two partial index materializations, where fewer nodes are materialized thereby resulting in indices that are smaller in size.

The main idea in IL is to build nodes at the lowest granularity only (black nodes in Figure 4.1c), corresponding to lines 3-6 of Algorithm 5. All other nodes of higher granularities can be extracted from the leaf nodes at query time, if necessary. Respectively, the main idea in CP is to build paths containing all the nodes at higher granularities that include a given leaf node (black nodes in Figure 4.1d). The algorithm that builds and maintains CP is a modification of Algorithm 6, by ignoring lines 12-14 that build nodes of the inverse triangle and line 21 that builds the right sub-paths. From these nodes, built in partial materialization, all the remaining nodes (gray nodes in Figure 4.1d) may be generated at query time, if necessary (more details in Section 4.4).

4.3.3 Time & Space Complexity

The worst-case time complexity of IE and IL is dictated by DBScan, which needs $O(logn)$ time to find the neighbors for each of the n data points within an interval. Thus, the time complexity is $O(m*n*logn)$, where m is the number of nodes in the index. Furthermore, each cluster is represented by the statistics $(CF1;CF2;n)$ where $CF1$ and $CF2$ are r-dimensional vectors. Particularly, $CF1$ (resp, $CF2$) maintains, for each dimension, the sum of data values (resp, sum of the squares of data values). Thus, each cluster maintains $2r+1$ values and the space complexity is $O(K*(2r+1))$, where K is the number of clusters for all nodes.

The worst-case time complexity of CE and CP is specified by k-means, $O(n*k*r*i)$, where n is the number of r-dimensional data points forming k clusters at each interval and i the number of iterations. Furthermore, the space complexity is $O(m*k*(2*r+3))$, where $2r+3$ values are maintained for each of the k-clusters of all m clustering nodes. These values

contain the statistics described for IE and two extra values (details in [AHWY03]); the sum and the sum of the squares of the timestamps of input data.

Finally, the total number of nodes maintained in IE and CE is $m = \frac{1}{2} * L * (2 * |C^{gmin}| - L + 1)$, where $|C^{g_{min}}|$ is the number of clustering nodes at g_{min} and L is the number of index levels. The total number of nodes maintained in IL and CP is $m = \frac{N}{g_{min}}$, where N the total number of points.

4.3.4 θ and ε Learning

According to Definition 12, when the dissimilarity between two clusterings exceeds a threshold θ, a drift is detected. Since fixed threshold values are not always appropriate for data with varying drift rates, we are interested in learning θ experimentally and do so for each granularity of our index.

During the learning phase, a training dataset is used in order to estimate the drift parameter, θ. The training region is independent from the testing dataset over which queries are to be evaluated. Furthermore, the estimation of θ is automated and without any a-priori knowledge of the arrival rates of drifts. However, in order to be well-estimated, it should be learned on a long-enough time period to ensure capturing the occurrence of several drifts.

Algorithm 7 summarizes the learning process of θ_g values per granularity level g. The algorithm takes as input a drift index G, as well as minimum g_{min} and maximum g_{max} granularities for which θ_g values need to be estimated. For each granularity g within g_{min} and g_{max}, it extracts the distribution of dissimilarities X, based on Definition 11, between each pair of consecutive clusterings at g (line 3). Then, it performs DBScan (i.e., any other algorithm could be used, like k-means) over X, given as ε the average pairwise similarity of the 3-nearest neighbors in X. DBScan is performed 10 times, in order to select the clustering C that optimizes the *DunnIndex* criterion. The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster to maximal intra-cluster distance. The inter-cluster distance is defined as the average distance between the centroids of the clusters. Similarly, the intra-cluster distance is defined as the average distance of any pair of points inside each cluster. Finally, the value of θ_g is extracted by calculating the average Euclidean distances between clusters in C (line 10).

The precision of θ estimation could be challenged when the similarity between two consecutive clusterings (line 3) varies significantly (i.e., bimodal distribution). This is due to the fact that the clustering distance is estimated (line 10) by using the mean of clusters distribution and assuming a low and constant standard deviation over time. Applying a Z-Score statistical test over all training and testing intervals we observed that the variation of

Algorithm 7 Learning of θ-parameter

Input: Drift index G, Minimum granularity g_{min}, Maximum granularity g_{max}
Output: Parameter θ_g for each $g_{min} \prec\prec g \prec\prec g_{max}$

```
1: minPts ← number of data dimensions
2: for all g_min ≺≺ g ≺≺ g_max do
3:    X ← dissimilarities distribution between consecutive clusterings of g
4:    ε ← avg pairwise similarity of 3 NN in X
5:    for all i ∈ [1, 10] do
6:       C ← DBScan(X, ε, minPts)
7:       dunnIndex ← DunnIndex(C)
8:       Pick C that maximizes dunnIndex
9:    end for
10:   θ_g ← average between clusters similarity of C
11: end for
```

the majority of intervals (at least 95%) are less than two times the standard deviation from the mean for all granularities and real datasets.

We also propose a training phase to learn the ε parameter used by DBScan for building IE. Parameter ε defines a maximum ε-neighborhood for each cluster. In literature, a common way to choose its value is by plotting all distances to the nearest neighbors and selecting the value where the plot shows a strong bend. A similar approach is followed by our learning process, adapted to each granularity level. The ε parameter can be estimated without any condition on the period's length. Thus, we propose to estimate both parameters (θ, ε) within the same wide-enough period.

It is worth noticing that the estimation of clustering parameters can quickly become outdated, particularly when dealing with rapidly evolving data distributions. In such cases, parameters re-estimation (e.g., every X intervals) may be useful to periodically adapt their values to data changes.

4.4 Query Evaluation Algorithms for Difference Evolution

This section presents our query evaluation algorithms using our proposed indices. Refinement and synthesis queries rely on unary queries that return a set of drifts X^g for any g. This is done by comparing the statistics between each pair of consecutive, non-overlapping clusterings at g using threshold θ_g. When a partial index is used (IL or CP), some index nodes (depicted in gray in Figures 4.1c, 4.1d) need to be generated on the fly possibly incurring computation overhead. The unary query algorithm is straightforward and is omitted for brevity. The performance of unary queries will be studied in detail in Section 4.5.

Algorithm 8 illustrates the steps for evaluating a refinement query. It takes as input any of the four materialized drift indices G and a range of granularities between g_s and g_t. Initially, it applies a unary query to detect all drifts $x_i^{g_s} \in X^{g_s}$ at g_s (line 2). Then, for each of these drifts, it detects all corresponding drifts $x_j^g \in X^g$ at finer granularities g, that are no finer than g_t (lines 3-11) using the condition $I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ (line 8) for all drifts at g.

Algorithm 8 RefinementQuery

Input: Drift index G, Granularity range $[g_s, g_t]$ ($g_t \prec\prec g_s$)
Output: A set S of drift pairs $(x_i^{g_s}, x_j^g)$, $g \prec\prec g_s$

1: $S, prev, curr \leftarrow \emptyset$
2: $X^{g_s} \leftarrow UQ(D, g_s)$
3: **for all** $x_i^{g_s} \in X^{g_s}$ **do**
4: **for all** $g_s \prec\prec g \prec\prec g_t$ **do**
5: $prev \leftarrow curr$; $curr \leftarrow \emptyset$
6: $X^g \leftarrow UQ(D, g)$
7: **for all** $x_j^g \in X^g$ **do**
8: **if** $I_j^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s}) \wedge I_{j+1}^g \subseteq (I_i^{g_s} \cup I_{i+1}^{g_s})$ **then**
9: $curr += x_j^g$
10: **end if**
11: **end for**
12: **if** $curr == \emptyset$ **then**
13: $S += (x_i^{g_s}, prev)$
14: break
15: **end if**
16: **if** $g == g_t$ **then**
17: $S += (x_i^{g_s}, curr)$
18: **end if**
19: **end for**
20: **end for**
21: return S

A synthesis query is evaluated over a drift index G and a granularity range between g_s and g_t, where $g_s \prec\prec g_t$. The steps of the algorithm are equivalent to Algorithm 8, by simply replacing the condition in line 8 with $I_i^{g_s} \subseteq (I_j^g \cup I_{j+1}^g)$. Thus, for any observed drift $x_i^{g_s}$ at g_s, a corresponding drift x_j^g at a coarser granularity g, no coarser than g_t is returned. The time interval $I_j^g \cup I_{j+1}^g$ of the corresponding drift x_j^g should take place during $I_i^{g_s}$ where $x_i^{g_s}$ was observed.

4.5 Difference Evolution Experiments

In this section, we provide a thorough investigation of our queries, both from the accuracy and the scalability perspectives. All experiments were conducted on a 2 GHz Intel Core i7 processor with 8 GB memory, which runs MAC operating system. Our accuracy results are the average of 5 consecutive runs. We learn ε and θ on a training dataset covering approximately 30% of the input data. Also, unless mentioned otherwise, we set the k parameter of CE to the average number of clusters produced by IE. Finally, we refer to each granularity level using incremental numbers (e.g., level 1 for the lowest granularity, then 2 etc). For both clustering algorithms (Clustream [AHWY03] and DBScan [EKSX96]), the implementations provided in MOA [BHKP10] are used. Some necessary extensions are applied in the implementation of CluStream, in order to provide additive and subtractive properties [AHWY03].

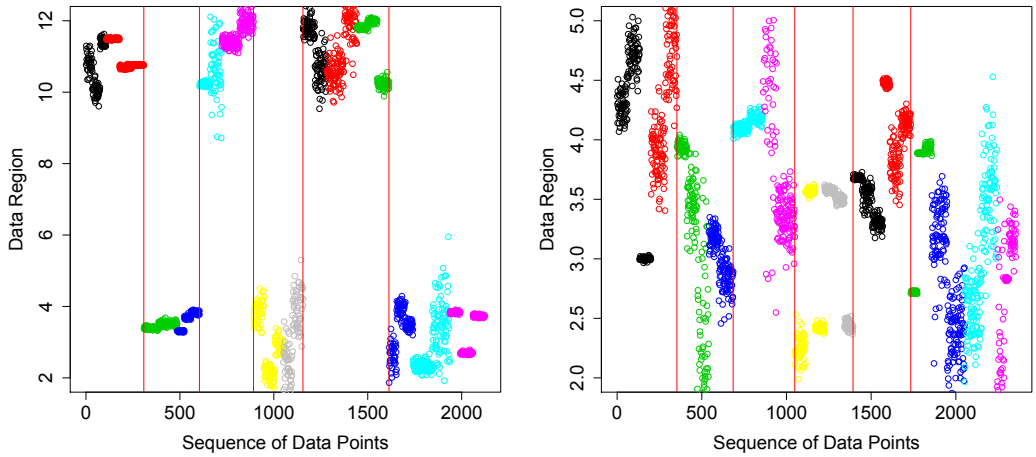
4.5.1 Dataset Preparation

Synthetic Datasets

We developed a synthetic data generator that provides the flexibility to produce datasets deriving from different distributions (i.e., well-separated, overlapping) and rates of change (i.e., sudden, incremental). Furthermore, the parameters of clustering are also tuned, including the number and size of clusters, as well as their density.

Specifically, synthetic datasets are produced with data points deriving from two distributions in a low and a high region. Each distribution consists of a number of close clusterings, that derive from the same region (i.e., low, high). Furthermore, each distribution contains a given number of data points and each clustering is described by k clusters. The total size of synthetic data is generated randomly and the number of drifts is also parameterized.

Three data sets are generated, in order to simulate different types of drifts. Figure 4.2 illustrates two examples. Specifically, Figure 4.2a depicts sudden drifts, marked with vertical lines, that occur each time the distribution of data oscillates between low and high region. The distribution of data in low region has values within [2, 4], while the distribution of high within [10, 12] forming well-separated regions. On the contrary, Figure 4.2b illustrates incremental drifts, occurring between low [2,4] and high [3, 5] regions of overlapping values. Finally, a third dataset is generated containing incremental drifts of consecutive regions, with values of low region within [2,4] and high within [4, 6].



(a) Well-separated regions (b) Overlapping regions

Fig. 4.2 Different Data Distributions

For the purpose of building the index, the dataset is split into intervals of 200 data points each forming a leaf node at g_{min}. The next granularity consists of 400 points, the third of 600 and the 10th contains intervals of 2000 data points.

Real Datasets

Query accuracy was evaluated over two datasets derived from two different application domains. The first collection consists of 5,931 Usenet articles from the 20 Newsgroup collection where each article belongs to one of 6 news feeds (e.g., sports, science). A user can subscribe to any of these feeds, showing his interest in receiving relevant articles, or unsubscribe at any time. Each article is represented with a binary vector of 658 attributes, where each attribute indicates the absence or presence of a word. Another attribute indicates whether the user is interested in an article or not. Thus, the clustering procedure will result in clusters containing articles that are likely to derive from the same feed (e.g., sports) and interest the user. A drift is the moment where a user decides to unsubscribe from some feeds and subscribe to others and can be computed on the whole data. The ground truth hence is known and encompasses five drifts. Experiments on Usenet are performed using a small point-based interval of 100 data points, a maximum index depth of 6 and the number of clusters for CE is $k = 3$. The training set consists of 2,400 data points, containing 2 drifts.

The second dataset of KDD Cup'99 is a Network Intrusion detection stream of 494,020 normal TCP connections and cyber attacks. It contains a variety of intrusions that fall into 4 categories: DOS, R2L, U2R and PROBING. Most of the connections in the dataset are

normal but occasionally bursts of attacks appear. Thus, we are interested in detecting drifts where realtime attacks occur. Each connection is described by 42 categorical (e.g., type of protocol) or continuous (e.g., bytes transmitted) attributes. For our analysis, we use the 34 continuous attributes. In order to create a ground truth for evaluating query accuracy, we consider as drifts the time moments where at least $minAttacks = 30$ consecutive malicious connections appear. The ground truth hence encompasses 45 drifts on the whole dataset. We set the smallest granularity to 500 points, the index depth to 10, corresponding to an interval length of 5,000 points and $k = 4$. The training dataset contains 20,500 points.

The smallest granularity is a critical parameter, indicated by the magnitude and arrival rate of drifts. To this end, the parameter settings used in [KTV06] are applied for our experiments in Usenet, where 5 drifts exist within 5,931 points. On the contrary, wider intervals are selected for KDD Cup'99 of lower arrival rate with 45 drifts within 494,020 points.

4.5.2 Summary of Results

Our experiments show that unary queries can reach a 79% accuracy on real datasets. They also show that independent clustering attains a significantly better accuracy than cumulative for incremental changes of overlapping data distributions. They also confirm the usefulness of refinement and synthesis queries, by demonstrating their ability to explore the tradeoff between precision/recall. For instance, using CE, while $UQ(\text{KDD}, 1)$ and $UQ(\text{KDD}, 10)$ attain 52% and 13% accuracy respectively, $SQ(\text{KDD}, 1, 10)$ attains 74%. Moreover, the scalability evaluation of our indices show a tradeoff between full and partial materializations, in terms of index size and query response time. Fully materialized indices are at least an order of magnitude faster in query response time than partial. On the contrary, fully materialized indices require at least 4 times more space than partial.

4.5.3 Accuracy of Drift Detection

Query accuracy varies between full and partial index materializations. This variation is caused by the random partitioning of data points during DBScan and k-means clustering. However, this variation is minimized with multiple executions and is not statistically significant. Hence we provide accuracy results for exhaustive indices only (IE, CE), assuming a not significantly different performance of partial indices (IL, CP). The accuracy is evaluated by using the traditional F-measure, which is the harmonic mean of precision and recall. A detected drift is considered a true positive if the corresponding real drift is within the compared intervals in the ground truth. This evaluation strategy is also used in [KBDG04]. For instance, a detected drift at point 800 extracted by comparing the point-based intervals [400, 800) and [800, 1200)

will correspond to a real drift within the region [400, 1200). This drift can, for example, take place at point 1000. However, a drift at point 1000 might also be detected by comparing intervals [800, 1200) and [1200, 1600). Thus, in case of multiple detections of the same drift at a given granularity, we ignore its subsequent detections.

Unary Queries

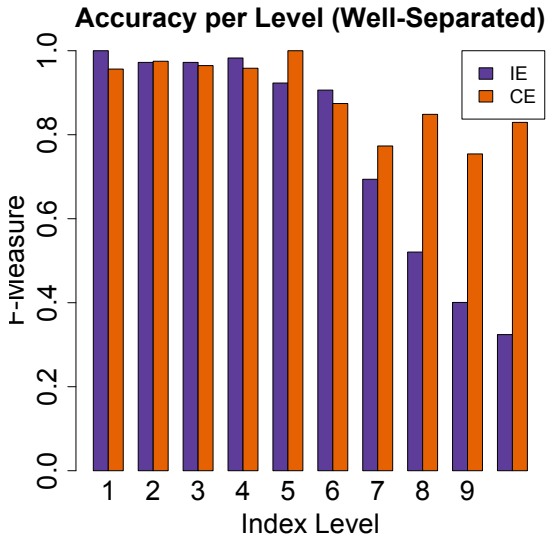
Synthetic Data. The goal of synthetic data evaluation is to understand how different time granularities, as well as clustering strategies (independent, cumulative) affect query accuracy. To this end, we perform unary queries over different granularity levels for both CE and IE.

Figure 4.3a illustrates the accuracy (y-axis) of unary queries for different granularities (x-axis) over the synthetic dataset with sudden drifts (Figure 4.2a). It shows that accuracy is very good for low granularities. However, recall worsens at higher granularities. Consequently, increasing the resolution of analysis decreases the ability of the algorithm to observe drifts occurring at finer granularities. However, precision remains greater than 0.9 at all levels.

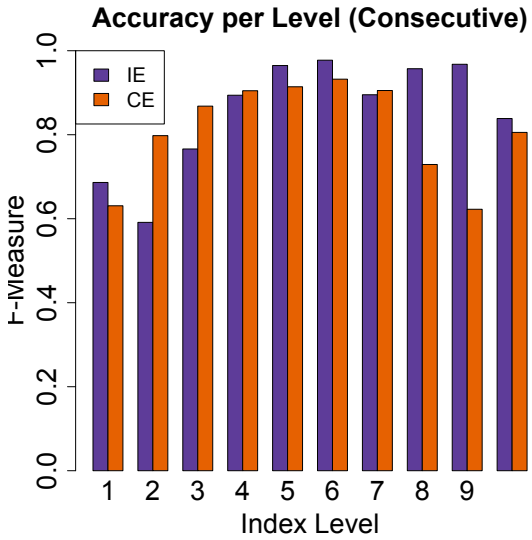
Figure 4.3b depicts the unary queries behavior when applied on incremental drifts of consecutive data regions. We observe that the algorithm performs badly for very small or very large intervals. Very small intervals are sensitive to subtle changes causing a large number of false positives. Thus, those granularities suffer from low precision but exhibit high recall. Similarly, very wide intervals are susceptible to false negatives as they may miss drifts existing within them. On the contrary, intermediate granularities provide intervals that fit data better and improve accuracy.

The last dataset containing incremental drifts of overlapping regions (Figure 4.2b) reveals a statistically significant difference in accuracy (Figure 4.3c) between IE and CE for all levels greater than 2. The observed difference is due to the design of each index. For instance, CE tends to add input data into existing clusters. This addition causes cluster centroids to shift over time and absorb any change, considering it as non-significant. To alleviate that, we ran an experiment varying the number of clusters, k. Although not shown here, we observed no significant improvement in performance. Thus, the online and one-pass design of the algorithm causes the absorption of incremental changes. On the contrary, IE forms clusters by independently visiting data points in different intervals. The algorithm detects data regions of high density and is independent from previously computed clusters. Therefore, IE outperforms CE for overlapping data regions.

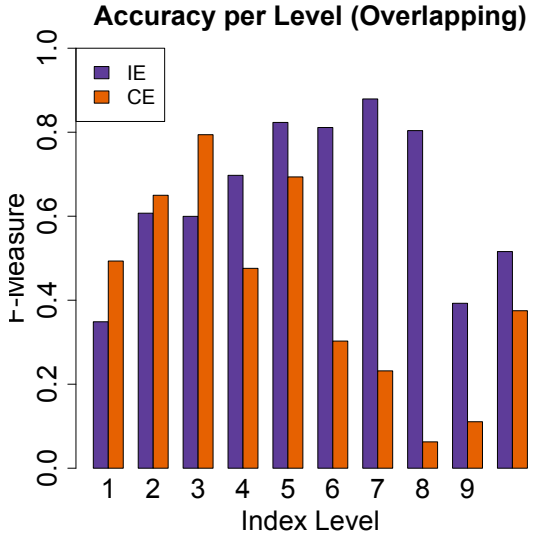
Although accuracy tends to decrease at higher index levels, there are some oscillations between levels. These fluctuations can be explained if we consider the statistical error introduced by adding and subtracting clusters' statistics as in [AHWY03]. A typical example of this error is illustrated in Figures 4.3a to 4.3c regarding the accuracy of CE at levels 9 and



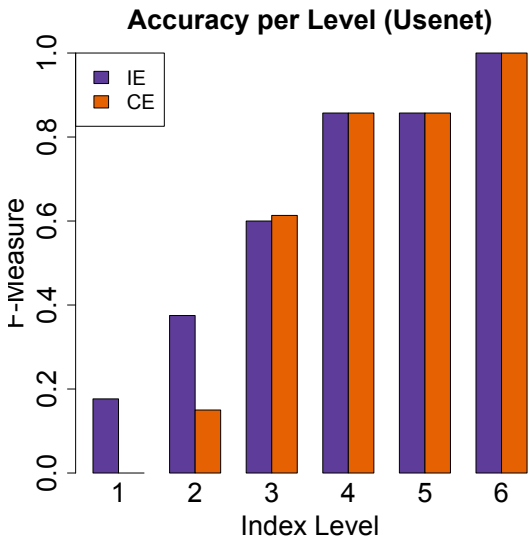
(a) Sudden drifts



(b) Incremental (Consecutive)



(c) Incremental (Overlapping)



(d) Usenet dataset

Fig. 4.3 Unary Queries Accuracy. A tradeoff between precision and recall is observed over different granularity levels.

10. Several nodes at level 9 are produced by the additive and subtractive property. On the contrary, none of the nodes at level 10 are generated by these properties. Thus, level 10 has a lower statistical error than level 9 and shows better accuracy despite its wider intervals.

Finally, we provide a comparison of our drift index accuracy with a state-of-the-art drift detection algorithm, named CUSUM [Pag54]. CUSUM calculates the cumulative sum which detects a drift when the mean of the input data is significantly different from zero. Results show that our drift index outperforms CUSUM for each granularity and dataset. Specifically, CUSUM reaches a 98% of accuracy for the dataset of sudden drifts, while the accuracy drops in 24% and 16% for consecutive and overlapping datasets respectively.

Real Data. Figure 4.3d shows the F-Measure results of unary queries (y-axis) on Usenet for each level in the index (x-axis). The main trend observed is an increase in accuracy as the granularity increases. This is not surprising, as the frequency of drifts in this dataset occurs at least every 700 data points. Thus, when the interval length increases (especially to 600 points at g_{max}) the algorithm performs very well, as there are enough available points to detect the drift. In fact, the unary query at g_{max} reports three different drifts. A user who initially subscribed to electronics and crypt news changed her interests into hockey and sales and then subscribed to motorcycles and space news. It is worth mentioning that the inappropriateness of interval length at the leaf level induces CE to report no drifts.

Tables 4.1 and 4.2 illustrate the accuracy (last line) of unary queries for KDD Cup'99 for each granularity level of IE and CE respectively. Similarly to synthetic data, we observe that the finest and coarsest granularities cause a decrease in accuracy for both indices. Indeed, a low precision and a high recall characterize the leaf level and the inverse is observed at the coarsest granularity.

Refinement & Synthesis Queries

We designed refinement and synthesis queries to explore precision and recall tradeoffs. Tables 4.1 and 4.2 present the accuracy results for KDD Cup'99. Each row (resp. column) corresponds to g_s (resp. g_t) which are given as input to the queries. The tables contain every possible combination of levels in an attempt to discuss the impact of each query on accuracy. For example, a refinement query is evaluated over a small range of levels (e.g., $RQ(D,2,1)$), as well as on the entire index (e.g., $RQ(D,10,1)$). The upper-half of the tables contains the F-Measure of synthesis queries and the lower-half concerns refinement queries. The best accuracy results for each table are mentioned in circles, along with the corresponding unary queries results.

The question we attempt to answer is whether refinement and synthesis queries can attain a tradeoff between accuracy of unary queries at g_s or g_t. All values mentioned in bold

		g_t										
		Unary	1	2	3	4	5	6	7	8	9	10
g_s	1	0.50	-	**0.65**	**0.76**	**0.75**	**0.79**	**0.79**	**0.77**	**0.80**	**0.80**	**0.78**
	2	0.62	0.49	-	**0.74**	**0.74**	**0.78**	**0.76**	**0.76**	**0.78**	**0.78**	**0.75**
	3	0.73	0.48	0.60	-	**0.73**	**0.78**	**0.76**	**0.75**	**0.77**	**0.76**	**0.73**
	4	0.73	0.49	0.61	**0.73**	-	**0.79**	**0.77**	**0.75**	**0.77**	**0.76**	**0.73**
	5	0.79	**0.50**	**0.62**	**0.74**	**0.73**	-	**0.77**	**0.75**	**0.76**	**0.75**	**0.70**
	6	0.77	0.49	**0.62**	**0.73**	**0.73**	**0.79**	-	**0.75**	**0.77**	**0.76**	**0.71**
	7	0.75	**0.50**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	-	**0.76**	**0.75**	**0.72**
	8	0.76	**0.50**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	**0.75**	-	**0.75**	**0.68**
	9	0.75	**0.51**	**0.63**	**0.74**	**0.74**	**0.79**	**0.77**	**0.75**	**0.76**	-	**0.66**
	10	0.44	0.36	0.41	**0.44**	**0.46**	**0.49**	**0.47**	**0.48**	**0.52**	**0.51**	-
Unary			0.50	0.62	0.73	0.73	0.79	0.77	0.75	0.76	0.75	0.44

Table 4.1 F-Measure for refinement (lower matrix) and synthesis (upper matrix) queries over **IE**, KDD Cup'99

		g_t										
		Unary	1	2	3	4	5	6	7	8	9	10
g_s	1	0.52	-	**0.63**	**0.72**	**0.73**	**0.69**	**0.68**	**0.68**	**0.68**	**0.71**	**0.74**
	2	0.61	0.50	-	**0.73**	**0.72**	**0.70**	**0.71**	**0.70**	**0.74**	**0.75**	**0.78**
	3	0.73	**0.52**	**0.61**	-	**0.73**	**0.71**	**0.73**	**0.74**	**0.72**	**0.73**	**0.77**
	4	0.70	0.50	0.60	**0.70**	-	0.68	**0.68**	**0.68**	**0.72**	**0.72**	**0.75**
	5	0.69	0.51	0.60	**0.71**	**0.72**	-	**0.66**	**0.68**	**0.66**	**0.68**	**0.76**
	6	0.62	0.50	0.57	**0.68**	**0.67**	**0.63**	-	0.57	**0.63**	**0.58**	**0.65**
	7	0.61	0.49	0.53	**0.65**	**0.62**	**0.64**	0.58	-	**0.56**	**0.58**	**0.67**
	8	0.56	0.48	0.54	**0.65**	**0.64**	**0.62**	**0.56**	0.55	-	**0.48**	**0.56**
	9	0.39	**0.44**	**0.48**	**0.55**	**0.49**	**0.48**	**0.45**	**0.43**	0.36	-	**0.37**
	10	0.13	0.25	**0.20**	**0.25**	**0.19**	**0.22**	0.12	0.12	**0.16**	0.09	-
Unary			0.52	0.61	0.73	0.70	0.69	0.62	0.61	0.56	0.39	0.13

Table 4.2 F-Measure for refinement (lower matrix) and synthesis (upper matrix) queries over **CE**, KDD Cup'99

indicate those cases. Thus, in the majority of the queries, the analyst will get a summary of drifts that exploits the tradeoff of precision and recall between g_s and g_t. However, note that there are few cases where the F-Measure is worse than any of the two levels. These cases are observed especially when refining queries at very low granularities (i.e., first two columns of tables). Low levels have a high rate of false positives, which negatively affects accuracy even when refining only the most precise drifts of higher levels.

4.5.4 Scalability Study of All Indices

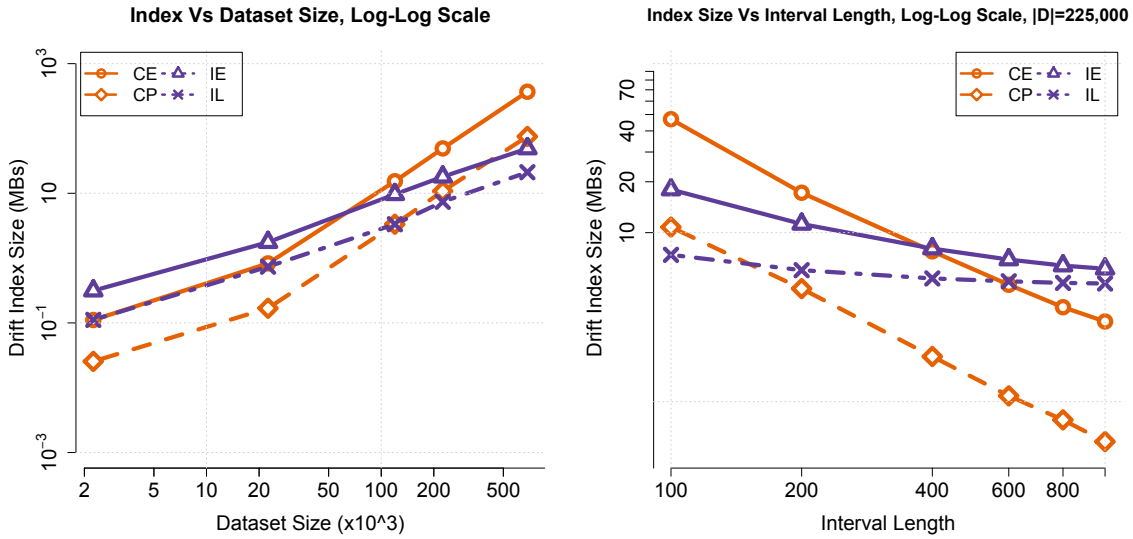
We perform an index scalability experiment to study index size and build time and query response time. For this purpose, synthetic datasets of different sizes are produced.

Index Size

Index size depends exclusively on the number and size of maintained nodes. Therefore, increasing the number of index nodes, for instance by increasing the index depth or the volume of input data or by decreasing the interval length, will inevitably lead to an increase of the index size. Moreover, increasing the size of each node, for instance by increasing the number of clusters (k), will negatively impact index size. We fix the depth to 5 and we test the impact of the other parameters on the scalability of our indices.

Varying Dataset Size. Figure 4.4a illustrates the drift index size calculated in MBs (y-axis) as a function of input data size calculated in number of points (x-axis). We set the minimum interval size at g_{min} to 100 points for all datasets. The chart is in a log-log scale and we observe a linear trend of all indices as data size increases. We also observe that the lines of cumulative indices have a steeper slope. This is explained by the extra overhead paid for maintaining information about a cluster's origin. This information describes if a cluster is derived from others and is more likely to increase as the dataset increases. Finally, as expected, partially materialized indices consume at least 5 times less space.

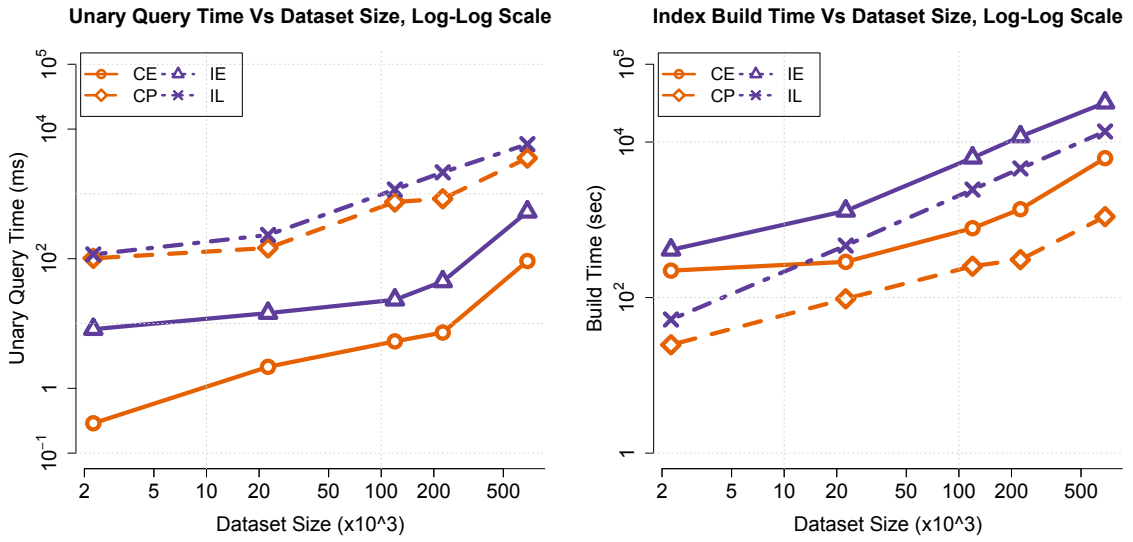
Varying Interval Length. The interval length selected for each granularity has also a critical impact on index scalability. Figure 4.4b shows the result of varying the interval length within $[100, 1000]$ for a fixed dataset size of 225,000 points. Figure 4.4b reports a decreasing index size trend for all indices with wider intervals. When the size of an interval increases, the size of the clustering inside that interval is not affected. That is because the same clustering statistics are maintained. Thus, wider intervals reduce the number of nodes maintained by the index without affecting the size of each node. This explains the decreasing trend observed.



(a) Index size Vs dataset size

(b) Index size Vs interval length

Fig. 4.4 Index Size



(a) Unary time Vs dataset size

(b) Index build time

Fig. 4.5 Unary Query Response and Index Build Time

Time Results

The most frequent and time consuming operation of drift detection is the computation of clustering dissimilarities for every pair of nodes at each granularity. The cost of this operation naturally increases with the number of index nodes as shown in Figure 4.5a. Figure 4.5a illustrates the response time of all unary queries applied at each granularity for different dataset sizes. It is evident that IL and CP need at least an order of magnitude more time to detect drifts, as they produce missing nodes at query time.

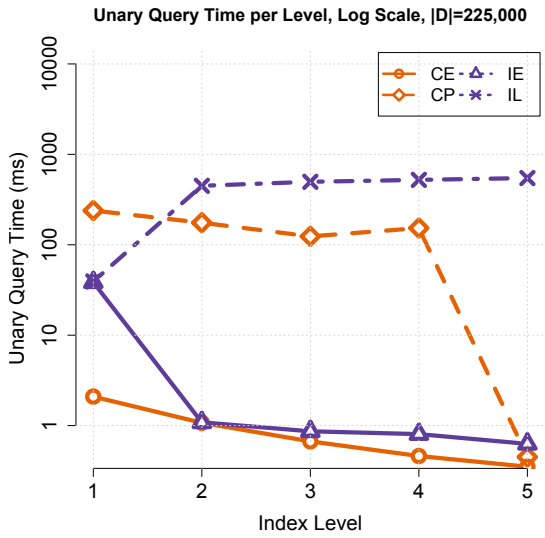
Figure 4.5b illustrates the build time of each index, providing evidence for the trade-off between index building time and query response time. Figures 4.5a and 4.5b indeed show that the more time we spend building the index, the less time we need during query evaluation. Furthermore, Figure 4.5b illustrates that both independent indices, IE and IL, need more build time than cumulative ones, CE and CP. Despite the fact that IL is a partial index, it needs a higher build time than CE. This could be explained by the fact that IL iteratively visits data points in order to form clusters.

Figure 4.6a depicts unary query response time (y-axis) per granularity (x-axis) for a synthetic dataset of almost 225K data points. IL shows a sharp increase in drift detection as levels increase, due to generating missing nodes on the fly. On the contrary, IE's response time decreases at higher levels, due to fewer nodes. Thanks to the hierarchical structure of our indices less nodes exist at higher granularities, explaining the almost constant response time after level one for both indices. That also explains the sudden drop in response time for CP.

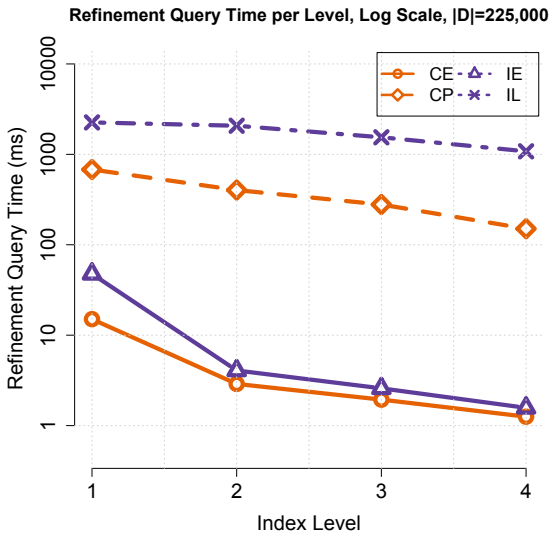
The performance of refinement queries is measured starting from level $g_s = 5$ until all finer levels of g_t, shown in the x-axis of Figure 4.6b. We notice that the longer the path from g_s to g_t, the more time it takes for the query to respond (for all indices). The same behavior is observed on synthesis queries, as shown in Figure 4.6c, where $g_s = 1$.

4.6 Summary of Difference Evolution

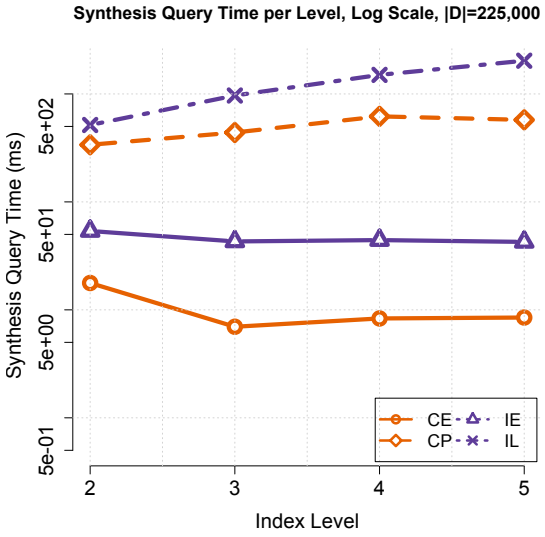
Our work introduces a novel form of analytical queries for detecting drifts in streaming content. Their evaluation relies on drift indices allowing us to explore, in a declarative fashion, precision and recall tradeoffs introduced by data segmentation at different granularities. We believe that our work lays the foundation for a series of new contributions addressing a long standing concern, namely adaptability of drift detection to different drift arrival rates and types. For fast arrival rates of drifts the detection accuracy could be excelled utilizing fine-grained intervals maintained in the index. On the contrary, for slow rates, coarse-grained intervals are preferred, compromising the computation overhead. Furthermore, sudden drifts



(a) Unary query time response



(b) Refinement query time response



(c) Synthesis query time response

Fig. 4.6 Query Time. Less nodes at higher levels of hierarchical index reduce the corresponding query time response.

can be easily detected by several state-of-the-art clustering techniques, while incremental drift detection is favoured by independent clustering.

Chapter 5

Related Work on Difference Analysis

5.1 Difference Exploration

Exploring users' preferences became an essential analytical task since the arising of social platforms. The plurality of users' opinions gave birth to the curiosity of further exploring them and gained the attention of scientists and market analysts [TP12]. While we are not aware of work that solves our problem, i.e., detecting user segments with an opinion close to input opinions, there are several bodies of work related to ours. For instance, researchers in *Databases* propose structures that capture the variety of user segments and criteria for meaningfully interpreting their opinion. In *Data Mining*, some work has addressed the high dimensionality of the search space, while in the field of *Social Data Analysis*, opinion convergence in the blogosphere has been studied. This section reviews the closest contributions to ours in three areas: Databases, Data Mining and Social Data Analysis.

5.1.1 Databases

In the database community, [DAyDY11] introduced mining rated datasets (i.e., datasets derived by social platforms) with the goal of extracting meaningful demographic patterns. Each demographic pattern describes users with distributions of the form $U_1, \ldots, U_M$ or with polarized opinions. The proposed problem statements maximize coverage of input rating records, as a way to detect users' segments with meaningful descriptions. Since their problem is shown to be NP-hard, they propose hill climbing algorithms for solving it, performed over a lattice structure that represents multiple users' segments. Although this work makes some effort in exploring the opinions of various users' segments, it exhibits two limitations. First, it restricts the number of input opinions to a limited set (e.g., low, high, polarized). Second, the opinion of each population is computed as the average of all its rating records.

However, comparing rating averages is less accurate than comparing based on the entire rating distribution.

In our work, we aim to explore the opinions of different users' segments by minimizing description length, which is shown to produce segments with high coverage. Our input distributions could have any shape including the ones handled in [DAyDY11], thereby generalizing them. Moreover, the comparison of opinions between population segments is performed over distributions (and not over averages) using `EMD`.

5.1.2 Data Mining

In data mining, subgroup discovery has been concerned with finding data regions where the distribution of a given target variable is substantially different from its distribution in the whole database [Klo02, FF99]. Subgroup detection [DGD12] is a related area that is concerned with finding agreeing or disagreeing groups by analyzing their discussions on online forums. Clustering is used to identify such groups based on characterizing each user with a feature vector extracted from discussions. Exceptional model mining [DFK16] is an extension of traditional subgroup discovery where the goal is to find regions of the input space that are substantially different from the entire database. In most cases, a subgroup discovery algorithm performs a top-down traversal of a search lattice. Our approach is more flexible since it can find user groups close to some input distribution and it leverages the additive property of `EMD` for efficient processing.

Subspace clustering has been used extensively for data exploration [KKZ09, PHL04]. CLIQUE [AGGR05] relies on a global notion of density, i.e., the percentage of the overall dataset that falls within a particular subspace. ENCLUS [CFZ99] uses information entropy as the clustering objective. CLTree [LXY00] uses a decision-tree approach to identify high-density regions, while Cell-Based Clustering [CJ02] improves scalability with data partitioning. *The ability to take into account input distributions to find relevant population segments would require substantial modifications to subspace clustering.*

5.1.3 Social Data Analysis

In social media, authors of [DCSJS08] examined opinion biases in the blogosphere, using entropy as an indicator of diversity in opinions. Alternatively, authors of [VVP08] proposed clustering accuracy as an indicator of the blogosphere opinion convergence. A different perspective where user segments are extracted from opinions, instead of the opposite, has also been studied. In particular, demographic characteristics of users have been extracted

in different contexts, by studying authored documents [AKFS03], blog pages [SKAP06] or online search behavior [WC10].

In our work, we propose a framework which is quite general, as it handles different input distributions. However, it does not directly support the online detection of diverging opinions or the extraction of demographics based on the expressed opinions of users.

5.2 Difference Explanation

The problem of comparing and explaining differences in data is a fundamental data mining problem, met early in literature. The first approaches in comparing multiple datasets were often very simple so that they could be performed only by observation of the underlying distributions. *Online Analytical Processing (OLAP)* tools gave a great boost to the analysis, enabling users to easily and selectively view and compare data from different points of view. However, the query-based nature of those approaches poses some limitations, as it is difficult to automatically discover specific patterns of interest. Thus, valuable insight may be missed either because patterns may be hidden in data or because their identification is dependent on the analyst's ability to make targeted explorations. *Contrast data mining* is a focused area of data mining that contributes with concepts and algorithms in overcoming those limitations. Although this area has gained a significant attention over time, fewer works can be found on identifying a minimal set of contrast patterns that best explain differences between datasets. The focused field of reporting parsimonious explanations is ubiquitously important for a variety of analytical queries and applications. This section summarizes related work focusing on the following three areas: Online Analytical Processing, Contrast Data Mining and Parsimonious Explanations.

5.2.1 Online Analytical Processing

Online Analytical Processing (OLAP) [CCS93] is a design paradigm, a way to seek information out of the physical data store. OLAP systems are aggregating information from multiple sources and store them in multi-dimensional databases [AGS97], where each data attribute (e.g., product, region, time period) is considered as a separate dimension. OLAP tools [1] can locate the intersection of dimensions (e.g., products sold in France during summer) and summarize them. To this end, they enable users to apply analytical operations and navigate through data either by rolling-up to aggregated views of data or drilling-down in their details. As a positive effect of this flexible exploration, analysts acquire the ability to

[1] https://en.wikipedia.org/wiki/Online_analytical_processing

explain differences in data by comparing different aggregates. However, explanation needs to be manually guided by the analyst through the huge space of all possible aggregations. Thus, the detection of particular patterns or patterns that are hidden in different granularity levels can be either inefficient or impossible. OLAP systems focus on providing flexible exploratory operations, rather than a dynamic detection of the most significant differences in data along with their explanations.

An attempt to solve the problem of looking a large number of values to spot a difference is done in [SAM98]. In this work, the authors bridge the gab between a hypothesis-driven exploration, where analysts navigate unaided through the search space, to a discovery-driven exploration. Their approach mines data containing differences (i.e., namely anomalies) and summarizes them in advance at an appropriate granularity level. In that way, analysts are guided by precomputed indicators of differences increasing their chance to identify abnormal patterns.

In our work, we suggest an approach which helps analysts in understanding differences in data. Thus, instead of navigating through the multi-dimensional search space a summary of explanations is provided.

5.2.2 Contrast Data Mining

Contrast data mining is the first effort to dynamically discover *contrast patterns* that describe significant differences between two datasets. These differences are described by combining various contrasting attributes, such as time, location, or demographics.

Most of the works in contrast mining focus on providing interpretable and expressive representations of contrasts, while at the same time they try to reduce the number of patterns being contrasted. For instance, [BP01b] introduces a search algorithm for mining contrast patterns among demographic groups with drastically pruning the search space to gain computational efficiency. In [WJ11], a large query log from a U.S. search engine is analyzed in order to explain differences in users search behavior. Users with similar search behavior are clustered together and differences are explained in terms of their demographic attributes. Some works take a further step than simply detecting patterns; they also summarize them in order to limit the number of patterns presented to a user. For this purpose, they select an interesting subset to present. While the definition of interesting is subjective, several approaches have been proposed [KMR$^+$94, NLHP98]. Subjective measures of interestingness can mainly be categorized into *unexpectedness*, *actionability* and *novelty*. Patterns are defined as unexpected if they diverge when compared with an explicit list of beliefs that the user already knows [PT98, ST96] or with a list of what the user has already seen [BP01b]. Patterns are defined as actionable [HXD05], if their knowledge permit users to do an action to

their advantage (e.g., maximize profit). Finally, patterns are considered novel [AHBK04], if they contribute to new knowledge, with respect to the previous discovered patterns. Although these works perform an effective pruning of discovered contrast patterns, their filtering criteria have a subjective nature and their usefulness depends on user's interest.

In our work, we are interested in objective filtering criteria such as data-driven informativeness *and* conciseness *of detected patterns rather than users' preferences.*

5.2.3 Parsimonious Explanations

Although contrast mining techniques focus on an interpretable and expressive representation of contrasts, only few works seek to provide a minimal number of explanations that describe the differences between datasets. Thus, they might result in a great number of patterns representing local differences, particularly when the underlying data distributions are highly distinctive. Related work [ABG$^+$07, JBL09] is in the spirit of reducing the number of identified differences. These works focus on providing parsimonious explanations of the most significant differences in data.

The authors of [ABG$^+$07] are interested in contrasting two datasets over a measured value of interest (e.g., total number of sales) and extract data segments with significant divergences. Those segments are described according to a known attribute of interest with a hierarchical domain (e.g., location attribute is described by a hierarchy: state/city/zip_code). Then, a parsimonious explanation is constructed by including those data segments that exhibit significant differences in the measured value for a specific aggregation level in that hierarchy. The main limitation of this work lies in the uni-dimensional data segments included in the different explanations. In many applications, differences in a measured value of interest may due to several attributes of data segments that are not known in advance and not necessarily dispose a hierarchical domain. In this respect, this method is not capable of exploring the structural relations among multi-dimensional data segments when summarizing their differences.

[JBL09] addresses parsimonious explanation of the differences between two datasets based on a hierarchical structure that is dynamically built over several non-hierarchical attributes. A greedy algorithm is proposed to minimize the number of data segments included in the difference explanation using an appropriate two-sample statistical test. Each node of the hierarchical explanation captures the local difference of the data segments it represents, as well as the differences of all nodes across the path to the root. However, the data segments included in such hierarchy provide redundant information to their ancestors in the same path to the root, while no formal properties are defined for the finally constructed hierarchy.

In our work, we formalize the quality of hierarchical explanations via objective interestingness criteria, such as conciseness in the number or description length of explanations and informativeness of the reported differences.

5.3 Difference Evolution

Drift Detection, the task of detecting and monitoring evolution of differences, became a core analytical task of data mining research with the rise of emerging applications, such as networks traffic monitoring, sensor data analysis, social networking sites. A main characteristic of such temporal data is that the generated distribution changes over time, producing significant evolutionary moments in the application level. The detection of network intrusions or users' shift of interest are indicative examples of different applications detecting changes over time. These changes are referred to as temporal evolution, covariate shift, non stationarity or concept drift. The aforementioned terms are used to describe unforeseen changes of the stream's underlying data distribution occurring over time.

Detecting different *types of drifts* (e.g., sudden, incremental) fundamentally relies on splitting the input data into *proper intervals* in time, enabling the comparison of the past data with the present. Specifically, when dealing with large-scale streaming data the segmentation of the input into temporal intervals is a common approach, providing solution to computational and memory restrictions. A drift is then detected under the hypothesis that there exists an evolving distribution and thus those two intervals appear significantly different. The significance is measured by the use of *similarity metrics*, applied either on the data points or on the models induced by the data points of each interval. Several interval policies have been proposed, in order to capture the drifts of the underlying distributions and different similarity metrics have been applied.

This section provides a brief summary of the different types of drifts, interval policies and similarity functions used in literature.

5.3.1 Types of Drift

Several categorizations of drifts have been proposed [Brz10, GMCR04, Tsy04] based on their intrinsic characteristics, varying in the described level of abstraction. However, a consolidated distinction is the *abrupt/sudden* and *incremental* drifts. The first type of drift describes the state where the distribution variables instantly and irreversible change from one class to another. A real life example of such a drift is a traffic light switching from the state of pass to the state of stop. The second category includes drifts where variables

are increasing (or decreasing) their values slowly and continuously over time. The exact start of an incremental drift is difficult to be specified. The progressively increasing prices of an economy due to inflation is considered a typical example of this category. A more fine-grained distinction of incremental and gradual drifts can be found in [Brz10] with the former drift to occur in the variables space and the latter to involve the class labels.

Several works have been proposed detecting sudden changes in the underlying data distributions. The field of anomaly or outlier detection boasts works [CBK09] on the task of finding patterns that do not conform to the expected behavior. However, these works lack the ability to detect changes that appear slowly and incrementally in data. An attempt to solve this problem has been made in [KBDG04, Brz10]. However, these works fail to provide some insights on the long-lasting question of what methods are appropriate for each type of drift.

In our work, we study the two types of drift by using real and synthetic datasets and suggest different clustering approaches for detecting them.

5.3.2 Interval Policies

For the task of segmenting the input data, two main perspectives are met, considering either *fixed* or *variable-length* intervals. However, a detailed study on formalizing the different interval types can be found in [Adä13], where the starting position and interval length are further discussed. Intervals provide a way to control the size of processed data and thus give memory guarantees. Moreover, eliminating the data points arriving from old concepts can reveal the underlying drifts.

Most of the existing drift detection methods rely on segmenting the input stream into smaller fixed length intervals [BGP10, Ozo08, JMG95, KBDG04, VB09]. However, segmenting input into equal, fixed length intervals induces a potential loss of drifts. For instance, a large interval is susceptible to miss a drift (low recall) as it absorbs the statistical changes inside its interval, while a small interval may cause false alarms (low precision) for frequently changing data. Thus, the selection of a proper interval length is critical for the effectiveness of drift detection. However, fixed length intervals do not detect varying rates of drifts (e.g., hourly, daily). Hence, the interval length needs to change dynamically as data evolves in order to capture drifts in different time granularities (e.g., hour, day, week). To this end, it becomes clear that a main drawback of the aforementioned works derive from the lack of dynamically adapting to the varying change rates. Furthermore, they suffer from the problem of single granularity, due to fixed length intervals, resulting in low precision/recall.

An attempt to solve the problem of single granularity is made in [MKT09] by providing multi-partition techniques, but it also remains in an offline context. On the contrary, an

online drift detection approach that aims to detect the most recent drifts, by comparing the last two intervals, is presented by FLORA2 [WK96]. FLORA2 dynamically learns interval length. A heuristic method shrinks the interval, by forgetting old data points, each time a drift occurs; otherwise, the interval grows. Also, some other approaches exist [Bif10, GMCR04] calculating statistics over sliding and growing intervals in order to detect a drift. However, all these works lack the flexibility of querying historical and fresh data for detecting drifts at different granularities.

In our work, we take a step towards the detection of drifts at multiple time granularities of varying interval length. Moreover, we propose a query-based approach for analyzing the precision and recall in drift detection when contrasting historical and fresh data.

5.3.3 Similarity Measures

A straight forward approach for comparing time intervals is by directly measuring the similarity of their data points. Although this approach achieves the best accuracy, it has a high computational and memory cost. On the other hand, more compact models can be learned from the intervals (past or present), applying the similarity comparisons of their underlying distributions. These comparisons are mainly performed using two techniques [GŽB$^+$13]. The first concerns statistical tests [KKTC13, KBDG04, GFR06] under the null hypothesis of equal distributions. Rejecting the hypothesis reveals a significant statistical difference of the two distributions and thus the existence of a drift. The second explores similarity or probabilistic measures that quantify the (probability) distribution inequality between two different intervals. For instance, the Kullback-Leibler divergence [Kul87] is utilized for this concept [Ozo08, SG07]. Moreover, an entropy based metric [VB09] can be exploited, reporting the drifts in distributions as the value of entropy decreases. Then, the change is detected based on a user defined threshold. If the similarity of the two intervals is below the given threshold, a drift is detected. A drawback of these works derive from the lack of dynamically learning and adapting thresholds to the peculiarities of input data (e.g., type of drifts).

In our work, we propose learning algorithms for adapting our thresholds to the various rates and types of drifts. Moreover, it is worth noticing that related work is simplifying the problem into one dimensional data. In our work, we deal with the problem of multi-dimensional drift detection, by applying appropriate clustering techniques and suggesting suitable clustering similarity measures.

Chapter 6

Conclusion

6.1 Research Summary

Data produced by social platforms and monitoring applications are rarely stationary. They exhibit significant variations in the underlying data distributions across multiple data dimensions of interest that are not always known in advance. In this thesis, we are address the challenging problem of *Difference Analysis* and study three analytical tasks for serving a wide range of applications: *Difference Exploration*, *Difference Explanation* and *Difference Evolution*.

We defined *Difference Exploration* as the act of exploring differences over multiple dimensions. Within the context of social platforms, we explored differences in users' opinions over various demographic (e.g., age) and item (e.g., movie's attributes) dimensions. Our proposed framework employs an appropriate distance measure for contrasting opinions, where each opinion is in the form of a rating distribution. Moreover, it encompasses an efficient algorithm and heuristics for solving the NP-Complete problem of finding population segments whose opinion is close to an input one.

We defined *Difference Explanation* as the act of explaining differences according to some dimensions of interest. Within the context of monitoring applications, we provided a parsimonious set of explanations for a difference in sales of two stores over various demographic dimensions. Our proposed scoring function ranks the potential sets of explanations, according to their conciseness and informativeness ability. Our proposed greedy algorithm solves the NP-hard problem of finding a parsimonious set of explanations by attaining $(1-\frac{1}{e})$-approximation guarantees.

We defined *Difference Evolution* as the act of monitoring the evolution of data and summarizing differences over the time dimension. Within the context of monitoring applications, we summarized differences in network traffic and differences in users' subscriptions at different

time granularities. Our proposed dissimilarity measure is applied over clustering summaries of varying time intervals and detects significant differences of the underlying summaries' distributions. Our query-based algorithms provide high flexibility in analyzing precision and recall of differences at multiple time granularities. Those algorithms are evaluated over a scalable index that maintains clustering summaries of varying time intervals.

Our main contribution through this thesis is the formulation of three analytical tasks for Difference Analysis and the definition of novel measures, scalable structures and efficient algorithms for addressing them. Our experiments, conducted over synthetic and a wide variety of real data, validate the usefulness of our analytical tasks, the scalability of our proposed data structures and the efficiency of our algorithmic solutions. Difference Exploration uncovers the opinion of different user segments and provides guidance to end-users and analysts in exploring them. Difference Explanation exploits the hierarchical relations that exist among multi-dimensional data segments to construct a concise and informative summary of the differences between two datasets. Finally, Difference Evolution provides a query-based approach for analyzing differences in datasets with varying arrival rates and at multiple time granularities. Similarly, we provide different clustering approaches to capture different types of change.

6.2 Perspectives

In this section, we identify potentials for improvement for each analytical task. Then, we discuss new perspectives along with their challenges for achieving that improvement.

6.2.1 Difference Exploration

In our work on difference exploration we have proposed an algorithm that is linear in the number of input records. Our algorithm finds a basic Partition Decision Tree (PDT) maintaining population segments close to some input opinion and heuristics for improving the quality of the PDT. However, in Section 2.5 we noticed that some of our heuristics, i.e. `RF-Cluster` and `RF-Desc`, do not scale. The scalability of these heuristics depends on the size of the dimensional space (i.e., number of attribute-value pairs in the dataset), the number of input rating records and the number of decision trees being generated. To this end, an interesting perspective would be to study ways for improving our algorithms' complexity.

The exploration scenarios that we developed in our user study in Section 2.5.3 taught us several lessons. An analyst often needs multiple exploration steps to discover user segments of interest. Indeed, since analysts are not aware of the full extent of their data, they need

to start seeing some examples to determine what else they want to explore. This calls for a multi-step data-driven exploration where each step is guided by the discoveries made in the previous step. The second lesson is related to the expressivity provided to the analyst at each step. The ability of an analyst to select a group, a rating distribution, or a set of groups or rating distributions, will change the whole experience of exploration, as well as the efficiency of the underlying group discovery algorithms. To this end, an interesting perspective would be to broaden the interaction actions of an analyst in order to enable higher flexibility in Difference Exploration.

Perspective. We discuss one perspective for scalability improvement that can benefit all of our algorithms. Our algorithms for finding partitions, `DTAlg` and (the first Step of) `RF`, involve multiple evaluations of the following request: *given a distribution ρ (corresponding to a segment) and a set of input distributions $\{\rho_1, \ldots, \rho_p\}$, what is the* `EMD` *of ρ to its closest input distribution?* We anticipate the application of our algorithm for any rated dataset where the rating scale may be large (e.g., Yahoo!Music has a rating scale of 1–100) and where the number of input distributions p may be large (e.g., the user may have seen several distributions while exploring items and may want to know which segments exhibit similar distributions on a given class of items). To make this possible, it is essential to provide efficient support for the above request. A naïve approach is to evaluate $\texttt{EMD}(\rho, \rho_i), i \in [1, p]$ and pick the smallest `EMD`. But this is inefficient. Thus, we suggest to investigate a pruning strategy that relies on maintaining a lower-bound ℓ_i and an upper-bound up_i on the possible values of $\texttt{EMD}(\rho, \rho_i), \forall i$. Then whenever $\ell_i > min\{up_j \mid j \in [1, p]\}$, we can safely discard ρ_i as a candidate for being the closest input distribution to ρ. We provide more details on this pruning strategy in the Appendix, as a first step towards improving the efficiency of our algorithms.

The second perspective is to provide a higher expressivity in the interaction actions [OTAYT15] that an analyst can make at each exploration step. For instance, merging distributions of different segments or digging into the opinions of sub-populations of a given segment, would provide a greater understanding of how opinions are formed. Other actions such as clustering together users of similar preferences (i.e., in movies or books) can enhance the expressivity of analysts. This new perspective would lead to a more interactive and efficient Difference Exploration.

Challenge. To show the usefulness of the aforementioned pruning strategy, theoretical and experimental results are essential. While we have taken the first step of showing theoretically the computational benefit of the pruning strategy, we leave as future work the implementation and further experimentation of the strategy.

The first challenge in providing higher expressivity in Difference Exploration is to define data-driven and task-specific actions and design efficient algorithms that allow an interactive exploration of users' opinion. Another challenge derives from the need for a principled methodology to evaluate (a) the expressivity of exploration actions and (b) the need for an interactive multi-step exploration.

6.2.2 Difference Explanation

In our work on difference explanation we have proposed a greedy algorithm for finding parsimonious sets of explanations. However, the complexity of this algorithm is quadratic in the number of segments in the search space (Section 3.4) raising scalability concerns. The higher the number of dimensions and their values in the search space, the more time consuming the algorithm becomes. Thus, the need for efficient heuristics becomes imperative. Moreover, other approaches from the field of statistical modeling can be promising in Difference Explanation.

Perspective. A perspective in designing efficient heuristics is given by the following key observation. Instead of iterating over the entire search space in order to find segments with the best score, more relaxed iterations can be performed exploring fewer segments in order to detect segments close to, but not necessarily with, the best score. This relaxation will enhance the performance of the algorithm, but will inevitably sacrifice some of the quality of the resulting explanation set. For the sacrificed quality to be minimized, the relaxed iterations should once more exploit the hierarchical relations inherent in data. To this end, starting from a randomly selected set of segments S two alternatives can be explored: (a) a top-down approach, where each iteration explores only descendants of segments in S and finds the one with the highest score to add in S, (b) a bottom-up approach, where each iteration explores only ancestors of segments in S and finds the one with the highest score to add in S. Both approaches should minimize the number of comparisons per iteration and thus the total complexity of the algorithm.

Another perspective in explaining differences can be given by adopting a statistical model. Particularly, regression analysis can be promising in explaining differences w.r.t. different dimensions. Regression analysis can be used to understand which among the independent variables (e.g., age, gender) are related to the dependent variable (e.g., total number of sales) and to explore the forms of these relationships (i.e., causation or correlation). Many techniques for carrying out regression analysis have been developed and nonparametric[1] may be of interest for Difference Explanation.

[1] https://en.wikipedia.org/wiki/Nonparametric_regression

Challenge. The aforementioned two approaches require the design of an appropriate index, which improves the speed of segment retrieval operation. For instance, a potential option for the bottom-up approach would be to permit indexing of most refined segments, i.e. segments of the longest conjunctive description, while all other ancestors are retrieved by decomposing the descriptions of indexed segments. A benefit of this solution is that it does not require the a-priori knowledge of all space dimensions.

6.2.3 Difference Evolution

In our work on difference evolution we have considered queries that summarize differences at various time granularities. In practice, however, most differences occur not merely over time but also over other dimensions. For instance, users exhibit different opinions at different geographic locations. To this end, querying differences over different time and space granularities becomes a non trivial problem.

Perspective. Spatio-temporal queries raise new perspectives in analyzing difference evolution. Particularly, they are interested in (a) querying the time dimension: *How the opinion of a given geographic population changes over time?* (b) querying the space dimension: *How the opinion of different geographic populations varies for a particular time period?* (c) querying both dimensions: *How the opinion of different geographic populations evolves over time?*

Challenge. To address such spatio-temporal queries, particularly when referring to multiple time (e.g., hourly, daily) and space (e.g., city, country) granularities, three challenges need to be solved. First, a spatio-temporal index needs to be designed, able to accommodate and incrementally maintain incoming data. In addition, the index should support distributed/parallel implementations in order to allow efficient evaluation of Big spatio-temporal queries. Second, an appropriate measure that captures the dissimilarity between two spatio-temporal clusterings should be defined. While our current measure quantifies the difference between two clusterings of consecutive time intervals, it ignores their proximity over the spatial dimension. Finally, a new design of *unary*, *refinement* and *synthesis* queries, capable of analyzing the precision and recall of spatio-temporal drifts, is needed. While our query evaluation algorithms provide a way to refine and synthesize the temporal space, they lack the ability of iterating over different geographic resolutions efficiently.

References

[ABG+07] Deepak Agarwal, Dhiman Barman, Dimitrios Gunopulos, Neal E. Young, Flip Korn, and Divesh Srivastava. Efficient and effective explanation of change in hierarchical summaries. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 6–15, New York, NY, USA, 2007. ACM.

[ACALAZ+15] Jiujun A Cheng, Yingbo A Liu, Huiting A Zhang, Xiao A Wu, and Fuzhen A Chen. A new recommendation algorithm based on user's dynamic information in complex social network. In *Journal Mathematical Problems in Engineering*, 2015.

[Adä13] Iris Adä, Michael R. Berthold. Eve: a framework for event detection. *Evolving Systems*, 4:61–70, March 2013.

[AGGR05] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11(1):5–33, 2005.

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, pages 232–243, Washington, DC, USA, 1997. IEEE Computer Society.

[AHBK04] Ahmed Sultan Al-Hegami, Vasudha Bhatnagar, and Naveen Kumar. *Novelty Framework for Knowledge Discovery in Databases*, pages 48–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 81–92. VLDB Endowment, 2003.

[AKFS03] Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. Gender, genre, and writing style in formal written texts. *TEXT*, 23:321–346, 2003.

[AYKKK+16] Sihem Amer-Yahia, Sofia Kleisarchaki, Naresh Kumar Kolloju, Laks V.S. Laskhmanan, and Ruben H. Zamar. Exploring Rated Datasets with Rating Maps. Paper under review, September 2016.

[BGLB15] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, pages 1–34, 2015.

[BGP10] Alexis Bondu, Benoît Grossin, and Marie-Luce Picard. Density estimation on data streams : an application to change detection. In *EGC'10*, pages 229–240, 2010.

[BHKP10] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[Bif10] Albert Bifet. Adaptive stream mining: Pattern learning and mining from evolving data streams. In *Proceedings of the 2010 Conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, pages 1–212, Amsterdam, The Netherlands,, 2010. IOS Press.

[Boe11] Mirko Boettcher. Contrast and change mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):215–230, 2011.

[BP01a] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Min. Knowl. Discov.*, 5(3):213–246, July 2001.

[BP01b] Stephen D. Bay and Michael J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, 2001.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 10.1023/A:1010933404324.

[Brz10] Dariusz Brzezinski. Mining data streams with concept drift. Master's thesis, 2010.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[CCS93] E.F. Codd, S.B. Codd, and C.T. Salley. *Providing OLAP (On-line Analytical Processing) to User-analysts: An IT Mandate*. Codd & Associates, 1993.

[CEQZ] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM CDM'06*, pages 328–339.

[CFZ99] Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 84–93, New York, NY, USA, 1999. ACM.

[CJ02] Jae-Woo Chang and Du-Seok Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, SAC '02, pages 503–507, New York, NY, USA, 2002. ACM.

[DAyDY11] Mahashweta Das, Sihem Amer-yahia, Gautam Das, and Cong Yu. Mri: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11):1063–1074, 2011.

[DB12] Guozhu Dong and James Bailey. *Contrast Data Mining: Concepts, Algorithms, and Applications*. Chapman & Hall/CRC, 1st edition, 2012.

[DCSJS08] Munmun De Choudhury, Hari Sundaram, Ajita John, and Dorée Duncan Seligmann. Multi-scale characterization of social network dynamics in the blogosphere. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 1515–1516, New York, NY, USA, 2008. ACM.

[DF05] Roger Purves David Freedman, Robert Pisani. *Statistics, 4th Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[DFK16] Wouter Duivesteijn, Ad J. Feelders, and Arno Knobbe. Exceptional model mining. *Data Mining and Knowledge Discovery*, 30(1):47–98, 2016.

[DGD12] Pradeep Dasigi, Weiwei Guo, and Mona Diab. Genre independent subgroup detection in online discussion threads: A pilot study of implicit attitude using latent textual semantics. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 65–69, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[DR09] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2(5-6):311–327, 2009.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[F+03] Ronald Fagin et al. Optimal aggregation algorithms for middleware. *J. Comp. Syst. Sci.*, 66(4):614–656, 2003.

[FF99] Jerome H. Friedman and Nicholas I. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.

[GFR06] João Gama, Ricardo Fernandes, and Ricardo Rocha. Decision trees for mining data streams. *Intell. Data Anal.*, 10(1):23–45, January 2006.

[GMCR04] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in AI – SBIA'04*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004.

[GoCBBRD77] F. Glover and University of Colorado Boulder. Business Research Division. *Selecting Subsets of Maximum Diversity*. Management science/information science report series. Business Research Division, Graduate School of Business Administration, University of Colorado, 1977.

[GŽB+13] Joao Gama, Indre Žliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, in press, 2013.

[HW79] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[HXD05] Zengyou He, Xiaofei Xu, and Shengchun Deng. Data mining for actionable knowledge: A survey. *CoRR*, abs/cs/0501079, 2005.

[JBL09] R. Jin, Y. Breitbart, and R. Li. A tree-based framework for difference summarization. In *2009 Ninth IEEE International Conference on Data Mining*, pages 209–218, Dec 2009.

[JMG95] N. Japkowicz, C. Myers, and M. Gluck. A novelty detection approach to classification. In *Proc. of AI*, pages 518–523, 1995.

[KAYDCC15] Sofia Kleisarchaki, Sihem Amer-Yahia, Ahlame Douzal-Chouakria, and Vassilis Christophides. Querying temporal drifts at multiple granularities. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1531–1540, New York, NY, USA, 2015. ACM.

[KBDG04] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 180–191. VLDB Endowment, 2004.

[KCAY16] Sofia Kleisarchaki, Vassilis Christophides, and Sihem Amer-Yahia. Summarizing Differences in Multidimensional Datasets. Paper under review, September 2016.

[KCAYDC14] Sofia Kleisarchaki, Vassilis Christophides, Sihem Amer-Yahia, and Ahlame Douzal-Chouakria. Online Detection of Topic Change in Social Posts. In *Big'2014*, pages 1–4, Seoul, Corée, North Korea, April 2014.

[KG12] Andreas Krause and Daniel Golovin. Submodular function maximization, 2012.

[KKTC13] Sophia Kleisarchaki, Dimitris Kotzinos, Ioannis Tsamardinos, and Vassilis Christophides. A methodological framework for statistical analysis of social text streams. In Yuzuru Tanaka, Nicolas Spyratos, Tetsuya Yoshida, and Carlo Meghini, editors, *Information Search, Integration and Personalization*, volume 146 of *Communications in Computer and Information Science*, pages 101–110. Springer Berlin Heidelberg, 2013.

[KKZ09] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, March 2009.

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[KLG+08] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, November 2008. (Draft; full version available here).

[Klo02] W. Klosgen. *Handbook of Data Min. Knowl. Discov, ch. 16.3: Subgroup Discovery*. Oxford Univ., NY, 2002.

[KMR+94] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, pages 401–407, New York, NY, USA, 1994. ACM.

[KTV06] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual data streams. In *in ECML/PKDD-2006*, page 107. Springer Verlag, 2006.

[Kuh55] Harold W. Kuhn. The hungarian method for the assignment problem. *NRL Quarterly*, 2:83–97, 1955.

[Kul87] S Kullback. The kullback-leibler distance. *Am Stat*, 41:340–341, 1987.

[LDD01] Patrice Latinne, Olivier Debeir, and Christine Decaestecker. Limiting the number of trees in random forests. In *MCS'01 Cambridge, UK, July 2-4, Proc.*, pages 178–187, 2001.

[LLV07] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115, April 2007.

[LWK+13] Y. Liu, K. Wei, K. Kirchhoff, Y. Song, and J. Bilmes. Submodular feature selection for high-dimensional acoustic score spaces. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7184–7188, May 2013.

[LXY00] Bing Liu, Yiyuan Xia, and Philip S. Yu. Clustering through decision tree construction. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, pages 20–29, New York, NY, USA, 2000. ACM.

[MKT09] Mohammad M. Masud, Latifur Khan, and Bhavani Thuraisingham. A multi-partition multi-chunk ensemble technique to classify concept-drifting data streams, ser. In *Advances in Knowledge Discovery and Data Mining*. Springer, 2009.

[MOV73] *MovieLens*, as of 2003, www.grouplens.org/node/73.

[NLHP98] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 13–24, New York, NY, USA, 1998. ACM.

[OMA+01] L. O'Callaghan, N. Mishra, Meyerson A., S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering, 2001.

[Osi89] Andrzej J. Osiadacz. Multiple criteria optimization; theory, computation, and application, ralph e. steuer, wiley series in probability and mathematical statistics - applied, wiley, 1986, no. of pages 546. *Optimal Control Applications and Methods*, 10(1):89–90, 1989.

[OTAYT15] Behrooz Omidvar-Tehrani, Sihem Amer-Yahia, and Alexandre Termier. Interactive user group analysis. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 403–412, New York, NY, USA, 2015. ACM.

[Ozo08] K. Ozonat. An information-theoretic approach to detecting performance anomalies and changes for large-scale distributed web services. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 522–531, June 2008.

[Pag54] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):pp. 100–115, 1954.

[PHL04] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explor. Newsl.*, 6(1):90–105, June 2004.

[PT98] Balaji Padmanabhan and Alexander Tuzhilin. A belief-driven method for discovering unexpected patterns. pages 94–100. AAAI Press, 1998.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. *Discovery-driven exploration of OLAP data cubes*, pages 168–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[SG07] Raquel Sebastião and João Gama. *Change Detection in Learning Histograms from Data Streams*, pages 112–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[SKAP06] J. Schler, M. Koppel, S. Argamon, and J. Pennebaker. Effects of Age and Gender on Blogging. In *Proc. of AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs*, March 2006.

[ST96] Avi Silberschatz and Alexander Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowl. and Data Eng.*, 8(6):970–974, December 1996.

[T+07] Pang-Ning Tan et al. *Introduction to Data Mining, (First Edition)*. W. W. Norton & Company, 2007.

[TP12] Mikalai Tsytsarau and Themis Palpanas. Survey on mining subjective data on the web. *Data Min. Knowl. Discov.*, 24(3):478–514, May 2012.

[Tsy04] Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical report, 2004.

[VB09] Peter Vorburger and Abraham Bernstein. Entropy-based concept shift detection. In *ICDM'06*, pages 1113–1118. IEEE Computer Society, 2007-01-09.

[VVP08] I. Varlamis, V. Vassalos, and A. Palaios. Monitoring the evolution of interests in the blogosphere. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 513–518, April 2008.

[WBN03] Geoffrey I. Webb, Shane M. Butler, and Douglas Newlands. On detecting differences between groups, 2003.

[WC10] Ingmar Weber and Carlos Castillo. The demographics of web search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 523–530, New York, NY, USA, 2010. ACM.

[WJ11] Ingmar Weber and Alejandro Jaimes. Who uses web search for what: And how. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 15–24, New York, NY, USA, 2011. ACM.

[WK96] Gerhard Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. In *ML*, pages 69–101, 1996.

[ZCB08] Jun Zhou, Li Cheng, and Walter F. Bischof. Prediction and change detection in sequential data for interactive applications. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 805–810. AAAI Press, 2008.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25:103–114, 1996.

[ZW03] Mohammed J. Zaki and Limsoon Wong. Data mining techniques, 2003.

Appendix A

EMD Pruning Strategy

In this section, we develop a pruning strategy that relies on maintaining a lower-bound ℓ_i and an upper-bound up_i on the possible values of $\texttt{EMD}(\rho,\rho_i),\forall i$. Then whenever $\ell_i > min\{up_j \mid j \in [1,k]\}$, we can safely discard ρ_i as a candidate for being the closest input distribution to ρ. Our bounds are used in the spirit of the well-known NRA algorithm in top-k query processing [F$^+$03]. Our algorithm maintains a list L of promising candidates to be the closest EMD-neighbors of the given distribution ρ (equiv., segment). Similar to our earlier algorithm (Algorithm 2) for calculating the EMD for a pair of distributions, this algorithm makes one concurrent pass over ρ and all the input distributions ρ_i. It maintains excess/deficit positions on a stack as before. We assume there are M rating values in the rating scale.

Our strategy for finding the closest input distribution to that of a given segment is based on maintaining lower and upper bounds on possible EMD to various input distributions, allowing us to prone the processing of certain distributions. The following lemma characterizes those bounds.

Lemma 1 *Let ρ be the distribution of a given segment, after the algorithm has examined the first i positions, let δ be the total mass on the stack, j,k be positions on the top and bottom of the stack. Let $\gamma = \sum_{t=1}^{i} \rho_p[t]$ be the cumulative mass of distribution $\rho_p \in Q$. Let Δ be the work done so far for converting ρ into ρ_p. Then $\ell_p = \Delta + \delta \times (i+1-j) \leq EMD(\rho,\rho_p) \leq up_p = \Delta + \delta \times (M-k) + (1-\delta-\gamma) \times (M-i-1)$.*

Proof Sketch of Lemma 1: The key intuition is that after examining i positions, in the best scenario, all the mass (say excess) on the stack just needs to move to the next ($(i+1)$-th) position. Thus, the EMD cannot be smaller than the current work done plus this amount. The upper bound corresponds to the worst case where all the mass on stack (say excess) has to move the furthest, i.e., from the bottom position to the very last (M-th) position. Additionally, it is possible that there is excess mass left over at the last position M, which

needs to move to position $i+1$. The maximum possible value of this excess mass is $1-\delta-\gamma$.

We illustrate the pruning algorithm with a simple example. Consider a given distribution $\rho = [0.8, 0.2, 0, 0, 0]$ and the distributions $\{\rho_1 = [1,0,0,0,0], \rho_2 = [0,1,0,0,0], \rho_3 = [0,0,1,0,0], \rho_4 = [0,0,0,1,0], \rho_5 = [0,0,0,0,1]\}$. After examining the first two positions, the current work done for the various distributions is 0.2, 0.8, 0, 0, 0. The bounds are: $\ell_1 = up_1 = 0.2$; $\ell_2 = up_2 = 0.8$; $\ell_3 = \ell_4 = \ell_5 = 1$; $up_3 = up_4 = up_5 = 4$. At this point, the lower bounds for $\rho_2, \ldots, \rho_5$ exceed the upper bound for ρ_1 and we can drop $\rho_2, \ldots, \rho_5$ from further consideration.