

DOTTORATO DI RICERCA IN
COMPUTER SCIENCE AND ENGINEERING

Ciclo XXX

Settore concorsuale di afferenza: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore scientifico disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Enabling Ubiquitous OLAP Analyses

Presentata da: Simone Graziani

Coordinatore Dottorato

Prof. Paolo Ciaccia

Supervisore

Prof. Stefano Rizzi

Co-supervisore

Prof. Matteo Golfarelli

Contents

1	Introduction	1
1.1	Service-Oriented Data Sources	2
1.1.1	On-Demand ETL	3
1.1.2	Cost Models for Web Services	3
1.2	Visualization of Multidimensional Data	4
1.2.1	Optimization Techniques for the Shrink Operator	4
1.2.2	Multidimensional Shrink	5
1.3	Multidimensional Modeling	5
1.3.1	Automatic Multidimensional Modeling for Data Vaults	5
1.3.2	Multidimensional Modeling Over Sensor Data	6
2	Background Concepts	7
2.1	OLAP Analysis	8
2.2	ETL	10
2.3	Formal Framework	11
3	Extracting Data from Service-Oriented Sources	13
3.1	Service-Oriented Data Sources and BI	14
3.2	On-Demand ETL from Non-Owned Data Sources	16
3.2.1	Motivating Example	16
3.2.2	Formal Background	18
3.2.3	Contribution and Outline	19
3.2.4	Related Literature	20

3.2.5	The QETL Approach	23
3.2.5.1	Query and Extraction Model	25
3.2.5.2	Dice Management	27
3.2.5.3	Dice Dropping Policies	29
3.2.5.4	Optimization	30
3.2.5.5	Filtering	33
3.2.5.6	Dice Size Estimates	33
3.2.5.7	Additional Issues	35
3.2.6	Experimental Results	36
3.2.6.1	Dropping Policy Analysis	40
3.2.6.2	Reuse Analysis	41
3.2.6.3	Comparison with Chunking	43
3.2.6.4	Efficiency Analysis	47
3.2.6.5	Cost Function Analysis	47
3.2.7	Wrapping up QETL	48
3.3	Building Adaptive Cost Models for Web Services	50
3.3.1	Related Literature	51
3.3.1.1	QoS Prediction	51
3.3.1.2	Cost Modeling with Machine Learning	51
3.3.1.3	Database Histograms	52
3.3.1.4	Active Learning	53
3.3.2	Approach Overview	53
3.3.3	Formal Framework	54
3.3.4	Cost Model Management	57
3.3.4.1	The SAIRT Algorithm	57
3.3.4.2	Extending SAIRT with Multiple Linear Regression	58
3.3.5	Active Learning	62
3.3.6	Experimental Results	66
3.3.6.1	Comparison with SAIRT	68

3.3.6.2	Active Learning Analysis	71
3.3.7	Wrapping up Tiresias	72
4	Compact Visualization of Multidimensional Data	73
4.1	Motivation and Outline	74
4.2	Related Work	76
4.3	Background on the Shrink Operator	78
4.4	Optimization Techniques for the Shrink Operator	82
4.4.1	Mathematical Formulation	83
4.4.2	A Dual Ascent	84
4.4.2.1	Parametric Relaxation	85
4.4.2.2	Lagrangian Relaxation	85
4.4.3	A Lagrangian Heuristic	89
4.4.4	An Exact Method	90
4.4.5	Computational Results	91
4.4.5.1	Dual Ascent procedure	92
4.4.5.2	Greedy and Lagrangian Heuristics	98
4.4.5.3	Exact Method	99
4.4.6	Wrapping up the new Shrink Implementations	104
4.5	Multidimensional Shrink	106
4.5.1	The Multidimensional Shrink Framework	107
4.5.1.1	The Shrink Approximation	112
4.5.1.2	The Reduction Problems	114
4.5.1.3	Problem Search Space	115
4.5.1.4	A Heuristic Approach	116
4.5.2	Lazy Shrink Computation	118
4.5.3	Eager Shrink Computation	121
4.5.4	Experimental Results	123
4.5.4.1	Effectiveness Analysis	124

4.5.4.2	Efficiency Analysis	128
4.5.5	Wrapping up Multidimensional Shrink	130

5 Modeling of Unconventional Data Sources 133

5.1	Automating Multidimensional Modeling from Data Vaults	134
5.1.1	Related Work	134
5.1.2	Data Vault Basics	136
5.1.3	Formal Background	137
5.1.4	The Starry Vault Approach	138
5.1.4.1	Hub-To-Hub FD Detection	138
5.1.4.2	Md-Schema Discovery and Ranking	141
5.1.4.3	Candidate Selection	142
5.1.4.4	Md-Schema Construction	142
5.1.4.5	Ranking	144
5.1.4.6	Md-Schema Enrichment	145
5.1.5	Wrapping up Starry Vault	147
5.2	Multidimensional Modeling Over Sensor Data	148
5.2.1	Related Literature	149
5.2.2	Reference Architecture and Domain Model	150
5.2.2.1	Functional Architecture of the Analytical System	150
5.2.2.2	Domain Model	153
5.2.3	Multidimensional Schemata	155
5.2.4	Case Studies	158
5.2.4.1	Air Quality Monitoring	158
5.2.4.2	Requirements for Air Quality Monitoring	159
5.2.4.3	Sensing Air Quality	160
5.2.4.4	Warehousing Air Quality Data	160
5.2.4.5	Landslides Risk Management	162
5.2.4.6	Requirements for Landslides Risk Management	162

5.2.4.7	Sensing Landslide Risk Data	163
5.2.4.8	Warehousing Landslides Risk Data	163
5.2.5	Wrapping up Sensor Data Modeling	164
6	Conclusions	165
	Bibliography	180

Acknowledgements

With the following few lines, I would like to thank all those people who helped and supported me during my PhD endeavours.

First, my sincere gratitude goes to Prof. Matteo Golfarelli and Prof. Stefano Rizzi, who not only taught me most of what I know about research, but also constantly motivated me to push forward.

In addition, I would like to thank Prof. Simon Dobson, who hosted me at the University of St Andrews, for all the valuable discussions and insights on sensor networks.

Moreover, I wish to thank all my colleagues and friends, with whom I shared many good laughs and who always have been there to help me.

Finally, above everyone else, I would like to thank my family, who patiently and steadily supported me, especially when I needed it the most. A big thank you to my two little nephews as well, who never failed to cheer me up.

Abstract

An OLAP analysis session is carried out as a sequence of OLAP operations applied to multidimensional cubes. At each step of a session, an operation is applied to the result of the previous step in an incremental fashion. Due to its simplicity and flexibility, OLAP is the most adopted paradigm used to explore the data stored in data warehouses. With the goal of expanding the fruition of OLAP analyses, in this thesis we touch several critical topics. We first present our contributions to deal with data extractions from service-oriented sources, which are nowadays used to provide access to many databases and analytic platforms. By addressing data extraction from these sources we make a step towards the integration of external databases into the data warehouse, thus providing richer data that can be analyzed through OLAP sessions. The second topic that we study is that of visualization of multidimensional data, which we exploit to enable OLAP on devices with limited screen and bandwidth capabilities (i.e., mobile devices). Finally, we propose solutions to obtain multidimensional schemata from unconventional sources (e.g., sensor networks), which are crucial to perform multidimensional analyses.

Chapter 1

Introduction

The term *Business Intelligence (BI)* refers to a set of processes and technologies that aim at gathering, transforming, and analyzing data with the end goal of obtaining useful insights for decision processes. Traditionally, at the core of a BI system lies a *Data Warehouse (DW)*, which is a repository of integrated and consistent data modeled in a multidimensional fashion [1]. In the multidimensional model data are represented as cubes whose cells and edges respectively stand for events and analysis dimensions. Moreover, each cell is described by a set of measures (e.g., total income) and on top of each dimension is built a hierarchy that defines different levels of aggregation of the data.

Among the many techniques available to analyze the data stored in a DW, the most widespread is *On-Line Analytical Processing (OLAP)*. An OLAP analysis session is carried out as a sequence of OLAP operations (i.e., roll-up, drill-down, slice & dice, and pivoting) applied to multidimensional cubes. More precisely, at each step of the session, an operation is applied to the result of the previous step. This incremental approach coupled with the intuitive multidimensional model enables users with very limited IT expertise to carry out both explorative analyses and reporting duties. Due to its simplicity and flexibility, OLAP has been a staple technology of BI since its inception and still exists alongside the more sophisticated approaches offered by data mining techniques.

Although OLAP itself has not undergone significant changes, over the years the scope of the analyses in which it is used has been drastically expanding. Indeed, decision makers have started to incorporate more and more data that are not typically stored in corporate databases. For instance this is the case for social business intelligence [2], in which relevant data are fetched from the web in the form of user-generated content made available in forums, blogs, social networks, and the like; or it is the case for scientific applications where huge datasets (e.g., containing genomic data [3]) are shared worldwide and publicly available for research purposes. Besides, the fruition of BI is no more limited to desktop computers and it is now possible to carry out sophisticated analyses from mobile devices. While on the one hand these devices widen the fruition of BI technologies,

on the other they have some specific limitations, such as screen size and data bandwidth, that need to be taken into account by analysis tools. These limitations can bring new research opportunities and spur the creation novel approaches specifically tailored for mobile devices.

This work takes on the challenges and opportunities described above by focusing on bringing OLAP analyses to data sources and devices that traditionally do not support it. Specifically, the main issues tackled in this work that are critical to enable ubiquitous OLAP analyses are: (i) data extraction from service-oriented sources; (ii) compact visualization of multidimensional data; (iii) multidimensional modeling over unconventional data sources. The first issue is particularly relevant to extending the scope of traditional BI tools; indeed, many modern data sources are hidden behind service-based interfaces, which often have more restricted querying capabilities than traditional databases. The second issue is instead related to the fruition of OLAP analyses on mobile devices by allowing the users to tailor the visualization of multidimensional data based on their needs. Finally, the last issue has a similar goal to the first one (i.e., extending OLAP to new data sources), but instead of focusing on data extraction it tackles multidimensional modeling, which is crucial to enabling OLAP analyses.

In the following, the aforementioned research thematic will be briefly introduced emphasizing our envisioned solutions and novel contributions. Specifically, Section 1.1 introduces the approaches designed to deal with service-based data sources, Section 1.2 presents our novel techniques for data visualization and, finally, Section 1.3 shows the results achieved related to multidimensional modeling. The rest of the thesis is structured as follows: Chapter 2 introduces the required background concepts; Chapters 3, 4, and 5 present in detail our novel contributions (introduced below); lastly, Chapter 6 draws the conclusions and sets the direction of potential future work.

1.1 Service-Oriented Data Sources

The contribution related to service-oriented data sources is twofold: the first part is an on-demand ETL framework especially tailored for non-owned data sources, while the second one is a technique to automatically build adaptive cost models for web services. Noticeably, the second contribution can sit on-top of the first one to enhance the optimization of data extractions. Indeed, each time an extraction is required, the proposed on-demand ETL approach has to find the best set of queries to issue to the data source that minimizes the cost of the operation (e.g., time). One of the key issues in this optimization problem is building a cost model that allows to predict the cost of a given query.

1.1.1 On-Demand ETL

In traditional OLAP systems, the ETL process loads all available data in the data warehouse before users start querying them. In some cases, this may be either inconvenient (because data are supplied from a provider for a fee) or unfeasible (because of their size); on the other hand, directly launching each analysis query on source data would not enable data reuse, leading to poor performance and high costs. The alternative investigated here is that of fetching and storing data on-demand, i.e., as they are needed during the analysis process. In this direction we propose the Query-Extract-Transform-Load (QETL) paradigm to feed a multidimensional cube; the idea is to fetch facts from the source data provider, load them into the cube only when they are needed to answer some OLAP query, and drop them when some free space is needed to load other facts. Remarkably, QETL includes an optimization step to cheaply extract the required data based on the specific features of the data provider.

In greater detail, our novel contributions related to on-demand ETL are:

- We introduce an abstraction called *dice* for compactly representing the facts available in a cube at each time, and we show how dice can be used to efficiently determine the facts missing to answer an OLAP query.
- We present a heuristic algorithm that, given the missing facts and considering the features of the source data provider, finds the cheapest set of extractions that the ETL can carry out to fetch the data required.
- We discuss the result of a set of experimental tests, performed using a ROLAP architecture, aimed at evaluating QETL from both points of view of efficiency and effectiveness and at comparing it with a previous approach in the literature.

1.1.2 Cost Models for Web Services

Delivering accurate estimates of query costs in web services is important in different contexts, e.g., to measure their Quality of Service. However, building a reliable cost model is difficult as (i) a web service is a black box often hiding a complex computation, (ii) a call to the same service can yield completely different costs by simply changing a parameter value, and (iii) execution costs can drift with time. We propose Tiresias, an approach that, given a web service exposing an interface with a fixed number of parameters, initializes and actively adapts a model to accurately predict query costs. The cost model is represented by a regression tree trained through two interleaved querying cycles: a passive one, where the costs measured for user-generated queries are used to update the tree, and an active one, where the service is probed through system-generated queries to cope with drifts in the cost function.

Overall, the main contributions are:

- An architectural framework for deriving cost models of web services using their public interfaces only.
- An extension of the SAIRT algorithm [4] that incorporates multiple linear regression models with the result of improving the overall accuracy while keeping training costs compatible with the requirements demanded by streaming applications.
- An active learning algorithm that initializes the cost model and dynamically adjusts it in case of function drift.
- A set of experimental tests performed on both real and synthetic datasets to evaluate Tiresias in terms of efficiency and effectiveness.

1.2 Visualization of Multidimensional Data

To cope with the problem of information flooding and with the limitations of mobile devices (i.e., screen's size and data bandwidth), Golfarelli et al. [5] presented the *shrink* operator, which is aimed at balancing precision and size when visualizing multidimensional cubes via pivot tables. This operator can be applied during an OLAP session to the cube resulting from a query to decrease its size while controlling the approximation introduced. The idea is to fuse similar facts together and replace them with a single representative fact (computed as their average), respecting the bounds posed by dimension hierarchies. However, the implementation of *shrink* presented in [5] can operate on a single dimension at time. Furthermore, the algorithms presented, while correct, still have margin of improvement. Going in the direction of enhancing the *shrink* operator, in this work we present new algorithms for the mono-dimensional (both heuristic and exact) and a multidimensional generalization.

1.2.1 Optimization Techniques for the Shrink Operator

We propose a model to optimize the implementation of the *shrink* operation, which considers two possible problem types. The first type minimizes the loss of precision ensuring that the resulting data do not exceed the maximum size allowed. The second one minimizes the size of the resulting data ensuring that the loss of precision does not exceed a given maximum value. We model both problems as a set partitioning with a side constraint, which we solve with:

- An original formulation of the problem as a set partitioning problem with side constraints.

- An heuristic method based on dual ascent procedure that exploit pricing and Lagrangian relaxation.
- An exact method which solves the problem starting from the dual solution found by the dual ascent procedure.

1.2.2 Multidimensional Shrink

Since the original shrink operation [5] can only be applied to one dimension (i.e., reduces a cube along one dimension), to improve its efficacy, we propose a multi-dimensional generalization where facts are fused along multiple dimensions. Multi-dimensional shrink comes in two flavors: *lazy* and *eager*, where the bounds posed by hierarchies are respectively weaker and stricter. Greedy algorithms based on agglomerative clustering are presented for both lazy and eager shrink, and experimentally evaluated in terms of efficiency and effectiveness.

The proposed contributions are the following ones:

- The formalization of the shrink framework, including the computation of the shrink approximation, is generalized from the mono- to the multi-dimensional case.
- Two different forms of the hierarchy compliance constraints are defined (*lazy* and *eager*).
- The size of the search space for computing both lazy and eager shrink is characterized, and greedy algorithms are proposed.
- The approach is evaluated in terms of efficiency and effectiveness, also in comparison to those achieved by traditional roll-up and mono-dimensional shrink.

1.3 Multidimensional Modeling

Multidimensional modeling is required to enable OLAP analyses, however it is often a non-trivial task to obtain a proper schema for a given domain. For this reason we present: an automatic approach to obtain multidimensional schemata specifically tailored for data vaults [6]; and a set of manually designed schemata for sensor data.

1.3.1 Automatic Multidimensional Modeling for Data Vaults

The data vault model natively supports data and schema evolution, so it is often adopted to create operational data stores. However, it can hardly be directly used for OLAP

querying. We propose an approach called *Starry Vault* for finding a multidimensional structure in data vaults. Starry Vault builds on the specific features of the data vault model to automate multidimensional modeling, and uses approximate functional dependencies to discover out of data the information necessary to infer the structure of multidimensional hierarchies. The manual intervention by the user is limited to some editing of the resulting multidimensional schemata, which makes the overall process simple and quick enough to be compatible with the situational analysis needs of a data scientist.

1.3.2 Multidimensional Modeling Over Sensor Data

Due to the rapid growth of the Internet of Things [7], sensors are more and more ubiquitous and are quickly becoming one of the major sources of data. While these data are usually well exploited for real-time monitoring tasks, the knowledge to properly handle them in an integrated and long-term fashion is still missing. To fill this gap we propose the following contributions:

- We present a functional architecture to support both real-time and off-line analyses.
- We introduce a set of multidimensional schemata covering the main requirements typical of systems that deal with sensor data.
- We show how the schemata can be used in two different real scenarios.

Chapter 2

Background Concepts

In this chapter we define some of the basic concepts used throughout the rest of this work. We start by giving an informal presentation of OLAP analysis and ETL, respectively in Section 2.1 and 2.2. We then close with Section 2.3, which defines the shared formal framework that will be used to support our novel contributions. Please, notice that both Section 2.1 and Section 2.2 are not meant to be in-depth presentations, indeed, they are meant to serve as an introduction for the novice reader.

2.1 OLAP Analysis

In the context of BI the main paradigm used for modeling data is the multidimensional one. As shown in Figure 2.1, in a multidimensional cube events to be analyzed (e.g., census outcomes) are associated with multidimensional cube cells, while cube edges stand for analysis dimensions (e.g., RESIDENCE, TIME, OCCUPATION). For each cube cell is given a value for each measure describing the event (e.g., citizen incomes, number of children). On top of each dimension is built a hierarchy that defines groupings of its values. An example of cube is shown in Figure 2.1, while an example of hierarchies associated to dimensions are shown in Figure 2.2.

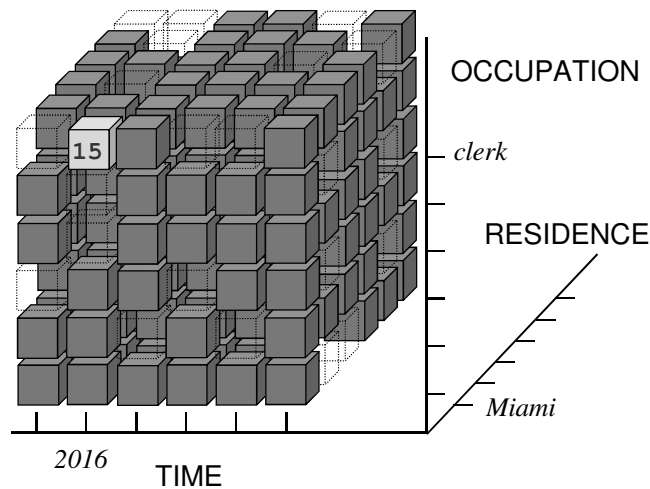


Figure 2.1: An example of a three dimensional cube

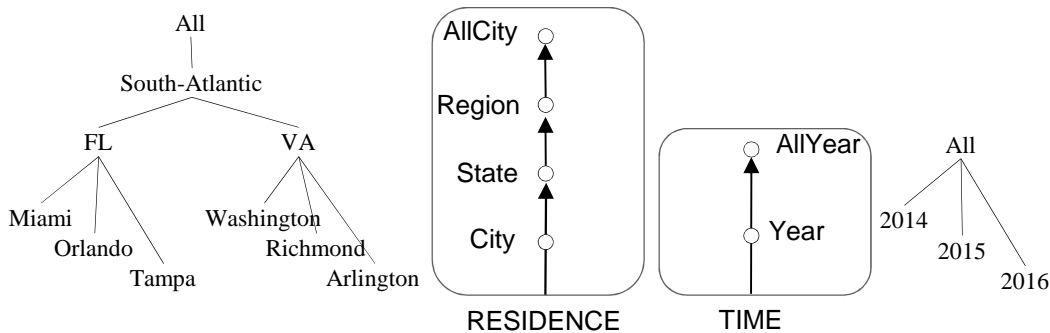


Figure 2.2: Roll-up orders and functions for two hierarchies in the CENSUS schema

Multidimensional cubes are queried through OLAP (*On-Line Analytical Processing*) queries, which allow users to interactively navigate the data without requiring advanced IT skills. An OLAP session is a navigation path composed by a sequence of queries. At each step of a session, the user can apply an OLAP operator to the result obtained from the previous

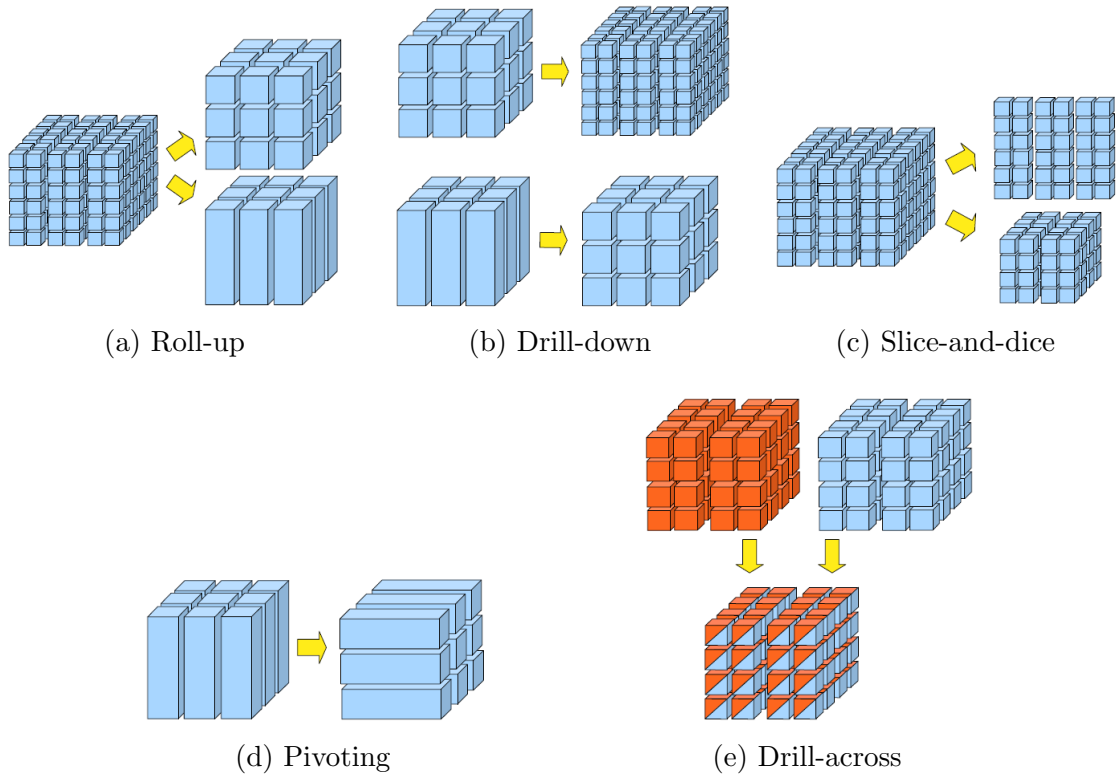


Figure 2.3: Visual representation of the OLAP operators

one. The most common OLAP operators are *roll-up*, *drill-down*, *slice-and-dice*, *pivoting*, *drill-across*, and *drill-through*.

- The *roll-up* operator aggregates data, thus reducing the level of detail (see Figure 2.3a). For instance, when applying a roll-up from **City** to **State**, the result will be a set of facts aggregated by **State** (e.g., the average income for each state).
- As shown in Figure 2.3b, the *drill-down* operator yields the opposite result of roll-up, hence it increases the level of detail of the data (e.g., from **State** to **City**).
- *Slice-and-dice* (see Figure 2.3c) reduces the number of visualized values by means of filtering (e.g., **City** = Bologna).
- *Pivoting* (see Figure 2.3d) causes a change in layouts, aiming at analyzing a group of data from a different viewpoint.
- Through the *drill-across* operator (see Figure 2.3b) it is possible to link the cells of different cubes to get a broader view of the data in the DW.
- The *drill-through* operator switches from multidimensional aggregated data to the data stored in operational data sources. This operator is rarely implemented by BI tools; indeed, the drill-through function of commercial tools is often a simple view of the data in the DW at the finest granularity.

		Year		
		2010	2011	2012
City	<i>Miami</i>	47	45	50
	<i>Orlando</i>	44	43	52
	<i>Tampa</i>	39	50	41
	<i>Washington</i>	47	45	51
	<i>Richmond</i>	43	46	49
	<i>Arlington</i>	—	47	52

Figure 2.4: A simple pivot table showing data per City and Year

The results obtained during an OLAP analysis are usually visualized either through charts (e.g., bar charts) or through pivot tables (as shown in Figure 2.4). While charts are more effective at summarizing big amounts of data, they might become too complex to understand when dealing with more than two dimensions. On the other hand, while pivot tables are simple and more flexible than charts, their readability quickly declines as the quantity of data to visualize rises.

To close this brief introduction on OLAP analysis, we remark that commercial analysis tools (e.g. Tableau ¹) are nowadays trying to seamlessly include the OLAP exploration style of analysis with more advanced techniques (e.g., clustering, regression analysis, etc.) Furthermore, modern tools often mask the implementation of these operators behind clever interfaces that allow to execute complex operations with minimal effort.

2.2 ETL

The *Extract, Transform, and Load (ETL)* process is in charge of extracting data from operational data sources, enforcing quality and consistency standards (so that data coming from different sources can be integrated) and, finally, feeding the DW. Traditionally, this process is executed periodically (e.g., daily) in a batch fashion. The frequency of the ETL must be carefully chosen: a more frequent ETL allows for up-to-date data but at the cost of a heavier burden placed on operational sources; on the other hand, a less frequent ETL might result in stale data but has a smaller impact on data sources.

In the following we describe the main phases of the ETL process.

- *Extraction*: it includes the extraction of data from the operational sources. The extraction can be either *static* or *incremental*. A static extraction is used to populate

¹<https://www.tableau.com>

the DW from scratch; an incremental extraction is instead used to update the DW depending on the changes occurred in the operational data.

- *Transformation*: this phase is in charge of cleaning and preparing the data to be fed to the DW. The cleansing part of this phase includes all those operations that aim at improving the data quality and consistency. For instance, the operational data might contain duplicate values, missing data, impossible values, etc. After all these issues have been addressed, the data need to be integrated and transformed into a format that is more suitable for analytic queries; for example, denormalizations of the schemata are often introduced to improve performances at query time.
- *Loading*: it is the last step of the ETL process. Similarly to the extraction phase, it can be carried out in two ways: *refresh* and *update*. In the former case, the DW is completely rewritten; in the latter one, only those changes applied to source data are added to the DW.

While traditionally the ETL process is periodically executed in a batch fashion, nowadays business users demand approaches that are more agile and flexible. As discussed in literature [8], ETL processes and DW architectures need to evolve to manage the growing data heterogeneity (e.g., sensor data) and strict freshness requirements (e.g., near real-time scenarios). Section 3.2 will treat this topic in greater detail and present our novel approach to on-demand ETL.

2.3 Formal Framework

In this section we present the main definitions shared by the approaches presented in the following chapters.

Definition 1 (Multidimensional Schema) *An n -dimensional schema (or, briefly, a schema) \mathcal{M} is a couple of*

- *a finite set of hierarchies, $H = \{h_1, \dots, h_n\}$, each characterized by a set L_i of levels and a roll-up total order $<_i$ of L_i . Each level l is defined over a categorical domain of members, $Dom(l)$. The domain of the top level of each hierarchy has a single All member. Apices will be used to denote the hierarchy each level belongs to; besides, the indexes of levels will be ordered according to their roll-up order: $l_1^i <_i l_2^i <_i \dots$*
- *a family of roll-up functions including, for each pair of adjacent levels l_j^i and l_{j+1}^i , a function $RU_{l_j^i}^{l_{j+1}^i} : Dom(l_j^i) \rightarrow Dom(l_{j+1}^i)$ that associates each member in $Dom(l_j^i)$ to one member in the next level.*

Let two levels $l_j^i, l_k^i \in L_i$ with $j \leq k$ be given. Given member $m \in \text{Dom}(l_j^i)$, we denote with $\text{Roll}^k(m)$ the member of l_k^i recursively defined as follows:

$$\text{Roll}^k(m) = \begin{cases} m & , \text{ if } k = j \\ \text{RU}_{l_{k-1}^i}^{l_k^i}(\text{Roll}^{k-1}(m)) & , \text{ if } k > j \end{cases}$$

We say that m rolls-up to $\text{Roll}^k(m)$. Conversely, given a set of members $M \subseteq \text{Dom}(l_k^i)$, we denote with $\text{Drill}_j(M)$ the set of members of l_j^i that roll-up to a member in M .

Example 1 *IPUMS is a public database storing census data [9]. As a working example we will use a simplified form of its CENSUS multidimensional schema based on two hierarchies, namely RESIDENCE and TIME. Within RESIDENCE it is City $\prec_{\text{RESIDENCE}}$ State, and Miami $\in \text{Dom}(\text{City})$ rolls-up to FL $\in \text{Dom}(\text{State})$ and to South – Atlantic $\in \text{Dom}(\text{Region})$ (roll-up orders and functions are shown in Figure 2.2).*

The possible ways to aggregate data are captured by the following definition.

Definition 2 (Group-by) *A group-by of schema \mathcal{M} is an element $G \in L_1 \times \dots \times L_n$. A coordinate of $G = \langle l^1, \dots, l^n \rangle$ is an element $d \in \text{Dom}(l^1) \times \dots \times \text{Dom}(l^n)$ (which, with a slight abuse of formalism, we will denote with $d \in G$).*

Example 2 *Some examples of group-by on the CENSUS schema are $G_1 = \langle \text{City}, \text{Year} \rangle$ and $G_2 = \langle \text{Region}, \text{Year} \rangle$. A coordinate of G_1 is $\langle \text{Miami}, 2012 \rangle$.*

All facts in a cube are characterized by the same group-by G , that defines their aggregation level, as well as by a coordinate of G and by a numerical value v ; this can be formalized as follows:

Definition 3 (Cube) *A cube at group-by G is a function C that maps each coordinate of G either to a numerical value called measure or to NULL. Each couple $\langle d, C(d) \rangle$ with $C(d) \neq \text{NULL}$ is called a fact of C .*

The reason for using NULLs is that cubes are normally *sparse*, i.e., some facts are missing. An example of missing fact is the one for the Arlington city and year 2010 in Figure 2.4.

Example 3 *Two examples of facts of CENSUS are*

$$\langle \langle \text{Miami}, 2012 \rangle, 50 \rangle \langle \langle \text{Orlando}, 2011 \rangle, 43 \rangle$$

The measure in this case quantifies the average income of citizens. A possible cube at G_1 is depicted in Figure 2.4.

Chapter 3

Extracting Data from Service-Oriented Sources

In this chapter we describe our novel contributions related to service-oriented data sources. After a brief introduction (see Section 3.1), we proceed to present QETL (see Section 3.2), which is an on-demand ETL framework designed to cope with those scenarios where loading all available data might be either inconvenient, or even unfeasible because of their size and cost of access (i.e., data for a fee). The idea behind QETL is that of fetching facts from the source data provider, loading them into the cube only when they are needed to answer some OLAP query, and dropping them when some free space is needed to load other facts.

The other approach presented in this chapter is Tiresias (see Section 3.3), whose aim is that of automatically building cost models for web services. Delivering accurate estimates of query costs in web services is important in different context, e.g., to measure their Quality of Service. However, building a reliable cost model is difficult as (i) a web service is a black box often hiding a complex computation, (ii) a call to the same service can yield completely different costs by simply changing a parameter value, and (iii) execution costs can drift with time. Given a web service exposing an interface with a fixed number of parameters, Tiresias initializes and actively adapts a model to accurately predict query costs.

3.1 Service-Oriented Data Sources and BI

Data warehouses (DWs) have been used for almost two decades in company settings to store information useful for decision making. Most of this information is typically gathered from corporate operational databases using an ETL (Extract-Transform-Load) process that extracts relevant data, transforms them into multidimensional form, and loads them into the DW in the form of *cubes*, to be later analyzed by means of reporting and OLAP tools. Traditionally, ETL is performed on a periodic basis by fast bulk-loading techniques during a time window in which the DW is in a quiescent state, i.e., is not queried by end-users. This means that, at query time, the information available has already been loaded in its entirety into the cubes.

Over the last few years, the scope of the analyses carried out by decision makers has been increasing in size to encompass a relevant quantity of data that are not necessarily stored in corporate databases. For instance this is the case for social business intelligence, in which relevant data are fetched from the web in the form of user-generated content made available in forums, blogs, social networks, and the like; or it is the case for scientific applications where huge datasets (e.g., containing genomic data) are shared worldwide and publicly available for research purposes. These data are often accessible through web services, whose functionalities range from simply granting the access to data stored on a DBMS, to running complex analyses on raw data aimed at returning valuable information.

In this context, our twofold contribution is presented in detail in Section 3.2 and 3.3.

- In Section 3.2 we define an on-demand ETL framework that allows the user promptly extract data from web services and load them in the DW for future reuse. Indeed, in many cases loading *all* available data into the DW cubes at ETL time may be either inconvenient (because data are supplied from a provider for a fee) or unfeasible (because of their size); on the other hand, directly launching each analysis query on source data would not enable data reuse, thus leading to poor performance and high costs. The QETL approach works as follows: given a user query, QETL extracts the required data and, if needed, drops already loaded facts to make room for the new query. Furthermore, to reduce the ETL costs, QETL includes an optimization step to cheaply extract the required data based on the specific features of the data provider.
- In Section 3.3 we present instead an approach to automatically build cost models for web services. In several cases, single services must be composed so that information from different sources can be integrated and complex workflows can be obtained. When multiple service compositions are feasible, the choice is often based on the *Quality of Service* (QoS). An important indicator of QoS for web services is the cost (typically, the execution time) for answering a query. Unfortunately, obtaining good estimates is often difficult, especially in the case of services that offer complex analyt-

ics, where a call to the same interface can trigger completely different computations by simply changing a set of parameter values. To automatically build a cost model from such services, Tiresias employs regression trees trained through two interleaved querying cycles: a passive one, where the costs measured for user-generated queries are used to update the tree, and an active one, where the service is probed through system-generated queries to cope with drifts in the cost function.

Noticeably, the latter contribution can be stacked on top of the former to enhance it. The statistics obtained through the cost models of the queried services can in fact be used to optimize the extractions issued by the on-demand ETL.

3.2 On-Demand ETL from Non-Owned Data Sources

As introduced above, traditional batch ETL approaches can be inconvenient or even unfeasible in many cases. The alternative investigated here is that of incrementally fetching and storing data *on-demand*, i.e., as they are needed during the analysis process. In greater detail, this on-demand approach can be especially useful in the following main application scenarios:

- When source data are supplied for a fee by one or more data providers, on-demand ETL enables exactly extracting the data that are actually necessary and allowing reuse of that data by several users at different times, thus reducing the overall costs of analyses.
- In scientific settings, the amount of possibly useful data shared by all the specialized repositories available worldwide can be intractable [3], so on-demand ETL can effectively cut the bootstrapping time by allowing incremental extraction and reuse of data.
- In a *situational business intelligence* scenario, the decision process is empowered with open/linked data that have a narrow focus on a specific business problem and, typically, a short lifespan for a small group of users [10]; in this context, on-demand ETL is a key for fetching, at each time, the relevant data needed for each specific analysis.
- More and more companies are using so-called *data lakes* to “park” huge volumes of low-quality and heterogeneous data in their native format until they are needed; in this setting, the complexity of cleaning and transforming these data to integrate them in the decision process discourages users from adopting a traditional batch ETL approach, making an on-demand approach preferable. In the latter scenario the data lake would be treated as any other data source that feeds the DW, and ETL operations would be applied only to the data that are required by analyses.

Before presenting the outline and main contributions of this work, we provide a description of the use case (that will serve as motivating example) followed by the basics of the formal framework.

3.2.1 Motivating Example

The GenData 2020 project aims at managing genomic data through an integrated data model, expressing the various features that are embedded in the produced bio-molecular data and in their correlated phenotypic data. This goal is achieved by enabling viewing, searching, querying, and analyzing over a worldwide-available collection of shared genomic

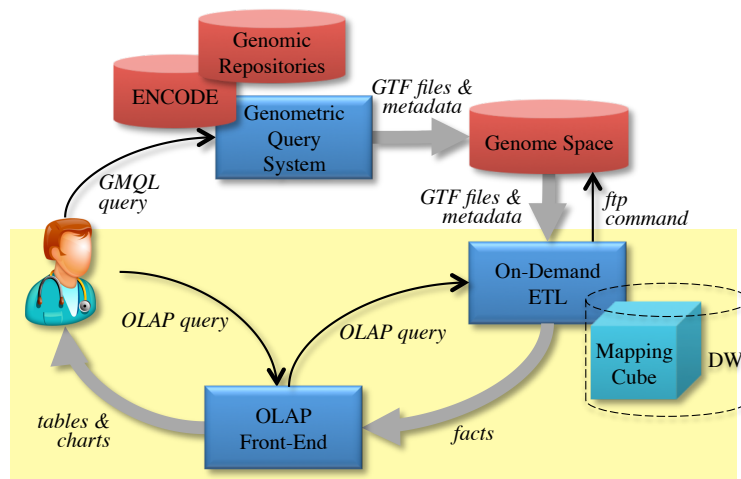


Figure 3.1: Analysis of genomic mappings in the GenData 2020 framework

data. One of the analysis services envisioned within this framework is the multi-resolution analysis of the mappings between *regions* (i.e., segments of the genome) and *samples* (i.e., sets of regions and correlated metadata resulting from an experiment) [3, 11]. As sketched in Figure 3.1, these mappings are computed by issuing a query in an ad-hoc language called GMQL (GenoMetric Query Language) against some repositories of genomic data such as ENCODE.¹ The query output, called *genome space*, comes in the form of a set of GTF (*Gene Transfer Format*, <http://genome.ucsc.edu>) files and related metadata, and due to its huge size is stored using the Hadoop platform.

OLAP-like queries are a valuable tool for biologists [3] because they enable multi-resolution analyses based on standard hierarchies of concepts; besides, they are preferred to traditional browser-based approaches because they enable a far more flexible and user-driven navigation of data. Unfortunately, the genome space generated by most biologically-relevant queries is too large to enable a traditional ETL process to load it into a multidimensional cube in a DW for OLAP analyses. This is where on-demand ETL comes into play. When a user formulates an OLAP query q , the front-end sends it to the on-demand ETL component for processing. If all the multidimensional data (called *facts* from now on, and including both the actual mappings and the correlated dimensional data about the involved regions and samples) necessary to answer q are already present in the mapping cube (i.e., they have been previously loaded), they are sent to the front-end and shown to the user. Otherwise, the genome space is accessed via FTP to fetch all the missing data, that are then transformed and loaded onto the mapping cube, so that q can be answered. Of course, from time to time, some facts used for past queries must be dropped from the cube to make room for the facts needed for new queries.

¹ENCODE, the *Encyclopedia of DNA Elements*, is a public repository (accessible via FTP) created and maintained by the US National Human Genome Research Institute to identify and describe the regions of the 3 billions base-pair human genome that are important for different kinds of functions [12].

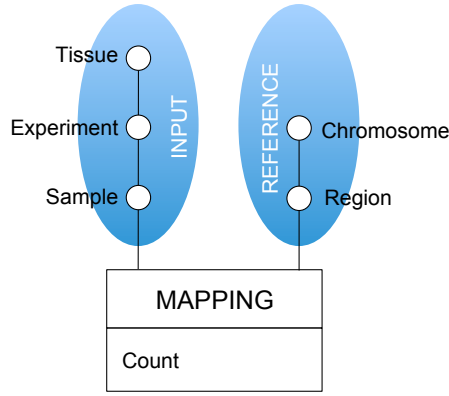


Figure 3.2: The MAPPING schema (the two All top levels are not shown)

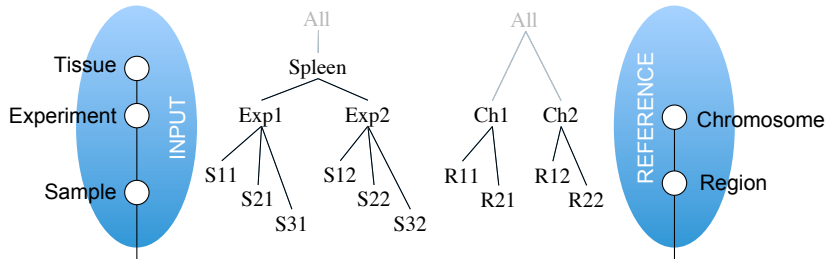


Figure 3.3: Fictitious roll-up functions for the INPUT and the REFERENCE hierarchies

3.2.2 Formal Background

In this section we recall the basic formal setting introduced in Section 2.3, adapting the presented definitions and enriching them with examples related to our case-study. For simplicity, we will consider hierarchies without branches, i.e., consisting of chains of levels, and facts with a single measure; see Section 3.2.5.7 for a discussion of how to deal with branched hierarchies and multiple measures.

We start by simplifying the notation of roll-up functions; indeed, as we only need to roll-up from members at the finest levels, i.e., $m \in l_1^i$, we will simplify the roll-up definition as $RollUp^{l_k^i} : Dom(l_1^i) \rightarrow Dom(l_k^i)$ for each level l_k^i .

Example 4 *As a working example we will use a simplified form, shown in Figure 3.2, of the MAPPING schema adopted in GenData 2020 for OLAP analysis of mappings (the complete schema is shown in [11]). The schema includes two dimensions, namely INPUT and REFERENCE. Within the INPUT hierarchy, $Sample \prec_{INPUT} Experiment \prec_{INPUT} Tissue$, $RollUp^{Experiment}(S21) = Exp1$, and $RollUp^{Tissue}(S21) = Spleen$ (see Figure 3.3).*

We also reuse Definition 2 and 3, which respectively describe the concept of group-by and cube.

Example 5 *Three examples of group-by's on the MAPPING schema are $G_{\perp} = \langle Sample, Region \rangle$,*

$G_1 = \langle \text{Sample, Chromosome} \rangle$, and $G_2 = \langle \text{Tissue, Region} \rangle$. A coordinate of G_1 is $\langle \text{S21, Ch2} \rangle$.

Example 6 Two examples of facts of MAPPING are $\langle \langle \text{S21, R22} \rangle, 2 \rangle$ and $\langle \langle \text{S21, Ch2} \rangle, 900 \rangle$. The measure in this case counts the number of regions of each input sample that overlap with each reference region.

3.2.3 Contribution and Outline

In this work we present *QETL* (Query-Extract-Transform-Load), an approach to on-demand ETL for feeding a multidimensional cube in scenarios where batch-loading the whole cube *before* query-time is either unfeasible (e.g., for space reasons) or inconvenient (e.g., for time or cost reasons). In QETL, facts are incrementally fetched from the source data provider and loaded into the cube only when they are needed to answer some OLAP query, to be possibly later dropped when they can be considered obsolete or when some free space is needed to load other facts. We remark that, in this context, with the term fact we mean not only the core multidimensional data (i.e., the measure values for a given multidimensional coordinate), but also the correlated dimensional data (i.e., the coordinate values and the corresponding hierarchy values). This means that, with reference to a classical star schema implementation, QETL works by loading/dropping tuples of fact tables and dimension tables at the same time.

The reason for storing the loaded facts into the cube (rather than simply using them to answer the OLAP query on-the-fly) is twofold. In scenarios where several users are concurrently analyzing the same cube, this caching-like mechanism encourages data reuse and cuts the cost for re-fetching the same facts twice or more. On the other hand, even when facts are mostly accessed by a single user (as in the genomic example, because each user normally builds her own mappings using custom GMQL queries), caching the facts extracted is convenient because the queries expressed during an OLAP session normally tend to be contiguous in terms of the facts they require [13].

In [3, 11] we set a case for on-demand ETL in the context of genomics and described a general framework for analyzing genome data. In this work we propose a specific solution to on-demand ETL; in particular:

- (i) We introduce an abstraction called *dice* for compactly representing the facts available in a cube at each time, and we show how dice can be used to efficiently determine the facts missing to answer an OLAP query.
- (ii) We present a heuristic algorithm that, given the missing facts and considering the features of the source data provider, finds the cheapest set of extractions that the ETL can carry out to fetch the data required.
- (iii) We discuss the result of a set of experimental tests, performed using a ROLAP

architecture, aimed at evaluating QETL from both points of view of efficiency and effectiveness and at comparing it with a previous approach in the literature.

The outline of the contribution is as follows. After surveying the related literature in Section 3.2.4, Section 3.2.5 describes QETL in detail, while Section 3.2.6 shows the results of the experimental tests we performed. Finally, in Section 3.2.7 we draw the conclusions.

3.2.4 Related Literature

Enabling OLAP-like analyses on data different from those stored in traditional corporate databases is one of the most pressing requirements from decision makers, data scientists, and researchers [14]. One of the barriers that must be broken to achieve this goal is the limitation of traditional ETL processes that are typically batch, heavy, and strongly oriented to corporate and structured data. This requirement has been discussed in several vision papers. Abelló et al. [10] envision an architecture for *situational business intelligence* and discuss the related research challenges emphasizing the need for data extractors capable of connecting to external data sources and rewriting queries on them. Morton et al. [15] push this concept further by defining a new type of analyst, the *data enthusiast*, who has very limited ICT skills but must be enabled to connect and query several different, and possibly unstructured, data sources.

The problem of loading external data into a DW is strictly related to the one of defining a multidimensional schema of these data. Several papers focus on the problem of automatically deriving a conceptual schema from non-conventional data sources such as linked data [16, 17], ontologies, and user-generated content [18]. Although some of them also propose a technique for semi-automatically deriving the ETL procedures that extract data from these sources [19] and for mapping the non-conventional sources onto a multidimensional schema [16, 17], none discusses on-demand loading and its optimization.

When users ask for strictly up-to-date data, a *right-time* DW is needed. This term is used to mean that changes in the real world are propagated to the DW in a timely fashion (not necessarily in real-time). In a right-time DW architecture [20] there are two components whose performance is crucial to assuring real-time or near-real-time processing of data: optimized ETL software and refreshing software. Logical optimization, focusing on restructuring ETL processes in order to minimize the cardinality of data flows, has been proposed by Simitsis et al. [21, 22]. In particular, [22] proposes a heuristic for searching the space of possible ETL graphs to find the most efficient execution. In [23] a new type of join, called *MeshJoin* is proposed for joining a fast update stream with a large disk-resident relation under the assumption of limited memory. Refreshing software is typically based on views and materialized views. The main issue here is to avoid inconsistencies on materialized views that are read and refreshed and, at the same time, are modified by operational transactions. There are a number of solutions that avoid this

inconsistency, based on (i) applying algorithms that compensate for out-of-date data [24], (ii) maintaining two versions of materialized views, where one version is being refreshed while the other is being read [25], or (iii) using additional data structures and transactions [26]. While the problem of delta-extraction (at each run of ETL, only the operational data updated/added since the last run are extracted) is obviously addressed in these works, no mention is made to on-demand ETL (i.e., to extracting facts as they are required by user queries).

Two approaches similar to ours are proposed by Kargin et al. [27] and Idreos et al. [14]. The first paper introduces *lazy ETL*, that delays ETL at query time. Since extraction and transformation are implemented as relational algebra operators, the applicability and expressiveness of the approach are quite limited; furthermore, cache management is left to the DBMS with no specific optimization. Similarly, Idreos et al. sketch a system capable of loading at query time portions of data involved in the current query and stored in external flat files. Like in [27], they envision a set of relational operators to be added to the current query to enable source data retrieval. The minimum portion of data to be loaded is a column value and no optimization of the data extraction process is proposed. None of the previous papers adopt a formal framework for describing the proposed solutions. Another interesting approach that delays ETL at query time is *Lenses* [28, 29], a framework for pay-as-you-go data curation (e.g., entity resolution and schema matching) based on probabilistic query processing. While that work can be placed in the context of on-demand ETL, the problem definition is quite different from ours. Indeed, in QETL the ETL effort is reduced by focusing the extractions on the facts required by user queries and by leaning on data reuse; conversely, the Lenses approach provides approximate, probabilistic answers so that not all the queried data have to be processed. While QETL provides exact answers, the basic claim in Lenses is that the user might prefer an approximate but less expensive answer; for instance, this might be the case during exploratory analysis.

A neighboring research topic is that of data caching. Indeed, the whole DW can be considered as a cache because it avoids directly accessing operational data to answer OLAP queries. The interest in caching issues in the DW literature radically changes depending on the infrastructural level considered. Caching of the query results has been pioneered by the WATCHMAN cache manager [30], which uses two algorithms for (in-memory) cache replacement and cache admission. An ad-hoc profit metric has been devised considering, for each retrieved set of data, its average rate of reference, its size, and the execution cost of the corresponding query. Noticeably, our approach bears several similarities with semantic caching [31]. The idea behind semantic caching is that of compactly representing stored queries by means of semantic descriptions, which is especially useful in client-server settings to reduce network overhead. However, approaches like semantic caching cannot be directly applied whenever data sources have limited query capabilities (e.g., they do not offer a logical negation operator). Other studies try to address these limitations by proposing systems specifically tailored for keyword-based

queries [32] or, more in general, for web databases [33]. Finally, several papers address the topic of materialized views, whose underlying idea is that of caching aggregated data within the DW to reduce the aggregation cost at query-time (see [1] for a review). Less interest has been raised in the use of caching at the ETL level, for instance, in [34] Liu proposes to use caching to optimize ETL data flows. To the best of our knowledge, no one of these papers proposes the use of the cubes themselves as a cache for data sources that cannot fit the available space —which is one of the use cases for QETL.

Our approach is also related to *chunking*, an approach for partitioning a cube into sub-cubes called *chunks* [35, 36]. Like a dice in QETL, a chunk is formally defined as a set of elements of a multidimensional array; differently from dice, chunks have fixed size and shape. While in QETL the goal of the dice construct is to enable a fast computation of the facts required by a user query and still missing from the cube, chunking is normally aimed either at minimizing the reading times from disk in MOLAP implementations of DWs, or at separating dense areas of a cube from sparse areas in HOLAP (hybrid OLAP) implementations. To the best of our knowledge, the only approach that uses chunking to reduce the complexity of caching is the one by Deshpande et al. [35]; though they need to find which chunks overlap with each multidimensional query, the computation of a “chunk difference” is rather simple: since all chunks have the same shape and size, the system simply needs to check which chunks overlap with the query and which chunks are already stored in the cache. Using chunking for our QETL approach has several drawbacks, all tied to the fact that chunking requires to specify both the number and shape of chunks beforehand: (i) a chunk envelope might not fit perfectly a given query, thus containing data that are not actually necessary; (ii) in our scenario, where some dimensional data are incrementally loaded, it might not be possible to define homogeneous chunks; and (iii) even if some algorithms to find the optimal number and shape of chunks exist [36], the solutions obtained can significantly depend on the type of workload. Despite these potential drawbacks, in Section 3.2.6.3 we experimentally compare the chunking approach with ours.

Overall, among the above-mentioned approaches, those that bear most similarity with ours are those by Kargin et al. [27] and by Ren et al. [31]. However, to address the query trimming problem (i.e., given two overlapping queries, determine the tuples contained in one but not in the other), both these approaches resort to the full expressiveness of relational algebra —which is often missing in data sources with limited query capabilities. Conversely, the querying expressiveness required by QETL is just that of selection on attributes —which is normally available in data sources. Besides, in lazy ETL [27] data extraction is not optimized when data sources with multiple interfaces are present, while with QETL we introduce a heuristics to minimize the cost of the issued queries. Finally, while the dice construct resembles chunking [35], the former overcomes many limitations of the latter by defining a more flexible and accurate envelope to represent multidimensional data.

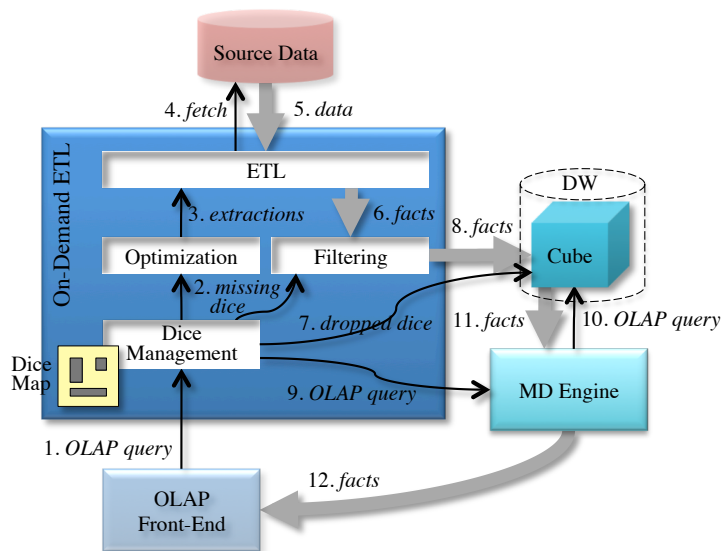


Figure 3.4: Functional architecture for on-demand ETL; black arrows represent function calls, while grey arrows indicate data flows

3.2.5 The QETL Approach

A functional view of the QETL process is shown in Figure 3.4, and its components are explained below. Though QETL can operate on multiple cubes, for simplicity we show a single cube. The abstraction we use to compactly represent the facts currently stored in the cube, those required to answer an OLAP query, those missing, and those to be requested to the source data provider through the ETL is called *dice* and is formally defined in Section 3.2.5.1; intuitively, a dice is a multidimensional interval of coordinates that determines a set of facts.

- The **dice management** process takes an OLAP query q and checks, using a map of the dice currently available in the cube (*dice map*), if q can be immediately answered or some facts are missing. In the first case, q is sent to the multidimensional engine for processing. In the second case, the difference between the dice required by q and the available dice is computed in terms of a set of missing dice and handed to the optimization process. This process is also in charge of choosing the dice to be dropped from the cube when some room is needed. Of course, when QETL operates on multiple cubes, each of them requires a separate dice map.
- **ETL**: this is a traditional ETL process that offers an interface consisting of a set of (*extraction*) services. To comply with the limitations often posed by the source data provider and by its query language, we assume that each service supports selection predicates on one or more levels (e.g., it might support selections on **Tissue** and **Chromosome**) and is capable of returning the set of facts corresponding to a single dice. When a service is called with a specific selection predicate, the ETL turns it

into a query on the source data provider, fetches the required data, and transforms them into multidimensional form. The ETL has a model for estimating the cost of each call to a service, based in general on both the cost for data fetching and those for their transformation.

- The **optimization** process knows the interface offered by the ETL and the cost for each service call as exposed by the ETL. Based on this information, it determines a set of extractions that cover all the missing dice and has total minimum cost. Each extraction entails a call to a service.
- Since the interface exposed by the source data provider does not necessarily allow full querying expressiveness, the facts fetched at each time may be a superset of those actually needed. The **filtering** process filters them before loading them into the cube; then it sends the set of loaded dice to the dice management process that updates the dice map accordingly.

With reference to the numbered arrows in Figure 3.4, the workflow of QETL can be schematically described as follows.

0. The user visually formulates an OLAP query on the OLAP front-end.
1. The OLAP front-end sends the query (e.g., in MDX format) to dice management. If some facts are missing:
 2. Dice management determines the set of missing dice and transmits them to optimization and filtering.
 3. Optimization determines a set of optimal extractions and calls the ETL service accordingly.
 4. ETL sends a fetching query to the source data provider.
 5. The source data provider returns the required data.
 6. ETL puts the data in multidimensional form and sends the resulting facts to filtering. If there is not enough room in the cube:
 7. Dice management chooses the dice to be dropped from the cube.
 8. Filtering loads the filtered facts into the cube.
9. Dice management sends the query to the MD engine.
10. The MD engine executes the query on the cube.
11. The query answer is returned to the MD engine.
12. The MD engine returns the query answer to the OLAP front-end.

In the following, we proceed to describe in greater detail the dice management, optimization, and filtering processes. As to ETL we emphasize that, in our Query-Extract-Transform-Load paradigm, each user query triggers not only an extraction and a loading, but also a transformation. However, the focus of this work is on incremental extraction and loading, so we will not specifically discuss the complexities of transformation (though in our experimental tests we will consider and single out the cost for transformation as well).

3.2.5.1 Query and Extraction Model

An OLAP query is normally defined by a group-by G and some selection predicates expressed on levels. To start simple, here we assume that facts are extracted and loaded into the cube only at their finest granularity, to be then aggregated by G by the multidimensional engine (in Section 3.2.5.7 we will discuss the implications of extracting and storing facts at different group-by's). For this reason, G is not relevant from the point of view of on-demand ETL, and we can simply represent a query q as a set of multidimensional intervals—those induced through the roll-up functions on the domains of the finest levels l_1^1, \dots, l_1^n , etc. by the selection predicates of q —that determine the coordinates of the facts to be returned to the user. The abstraction we use to this end is called *dice* and defined below.

Definition 4 (Range and Dice) *A range r^i of level l_1^i is an interval of members (m', m'') such that $m', m'' \in \text{Dom}(l_1^i)$ and $m' \leq m''$. A dice d is an n -dimensional interval of coordinates, $d = \times_{i=1}^n r^i$ where r^i is a range of l_1^i for $i = 1, \dots, n$.*

Working with ranges requires that a total order is defined on the members of each level l_1^i . To define such order we observe that, in several OLAP front-ends, the default behavior when a user clicks on a row/column of a pivot table (corresponding to a member of a level) is to disaggregate the measure values for that row/column into its components, which in OLAP terms means slicing and drilling down [13]. For instance, starting from a report showing mappings per tissue and chromosome, clicking on member Spleen would trigger a query showing mappings for experiments Exp1 and Exp2, while clicking on Ch1 would trigger a query showing mappings for regions R11 and R21. Normally, within each group, members are alphabetically sorted. For this reason, to define ranges and dice we will adopt a *hierarchy-based lexicographic order*, i.e., one in which the members that roll-up to the same member are lexicographically ordered.

From the topological point of view, two dice d and d' are either disjoint ($d \parallel d'$), overlapping ($d \sim d'$), or one of them is included in the other ($d \subseteq d'$). Of course, the exact relationship between two dice depends on whether each range in each dice is left-open/closed and right-open/closed; notice that, to avoid unnecessary notational complexity, we purposefully omitted these details from Definition 4. If you consider the example in Figure 3.5, with $d = (S11, S31) \times (R11, R14)$, $d' = (S22, S23) \times (R44, R45)$, and $d'' = (S31, S43) \times (R22, R45)$, it is always $d \parallel d'$, but the other relationships depend on the range closeness. Specifically, if d is right-closed and d'' is left-closed on the first dimension, then $d \sim d''$, otherwise $d \parallel d''$. Similarly it can be either $d' \sim d''$ (when d' is right-closed and d'' is right-open on the second dimension) or $d' \subseteq d''$ (in all other cases).

Definition 5 (OLAP Query) *An OLAP query is defined as a set Q of dice that represent the coordinates of the facts to be returned.*

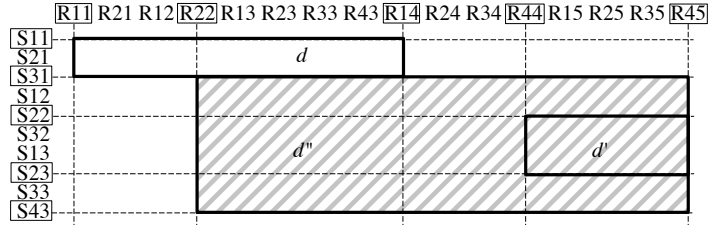


Figure 3.5: Topological relationships between three dice

Example 7 A dice of our MAPPING schema is $d = (S11, S12) \times (R22, R22)$. Adopting the hierarchy-based lexicographic order for the domains of both regions and chromosomes (like in Figure 3.3), this dice includes 4×1 coordinates. The query asking for the number of mappings between samples of tissue Spleen and regions of chromosome Ch2 is defined by $Q = \{(S11, S32) \times (R12, R22)\}$ (which includes 6×2 coordinates).

Like for OLAP queries, our model for the extractions supported by the ETL process is based on dice. However, while an OLAP query can correspond to any set of dice, a data provider normally has some limitations about the queries it can answer (for instance, selection may be possible only on a subset of levels), and these limitations restrict the set of dice that the ETL can return in practice. This is captured by the definition of service and interface. An interface is the set of services supported by ETL. A service allows the specification of selection (range) predicates on the members of one or more levels of different hierarchies, and corresponds to a sequence of queries to the source data provider to fetch the necessary data, plus some transformations to put these data in multidimensional form.

Definition 6 (Interface and Service) An interface is a set I of services. A service is defined by a group-by $S \in \times_i L_i$ that includes, for each hierarchy, the level on which it supports a selection.

An extraction is issued by calling a service with a specific selection predicate. For simplicity we will assume that each extraction returns the (non-aggregated) facts corresponding to exactly one dice.

Definition 7 (Extraction) An extraction using service $S = \langle l^1, \dots, l^n \rangle$ is any dice $e = \times_{i=1}^n r^i$ such that, for each $i = 1, \dots, n$, there exists an interval (m', m'') of $Dom(l^i)$ such that $Drill((m', m'')) \equiv r^i$, where $Drill((m', m'')) = \{m \in Dom(l^i) \mid RollUp^{l^i}(m) \in (m', m'')\}$.

Intuitively, the extractions that use service S are those whose ranges can be induced through the roll-up functions by range predicates formulated on the levels of S . Note that, as a consequence of these definitions, if $l_{all}^i \in S$ for some i , then all extractions using S are

characterized by range $(-\infty, +\infty)$ on h_i , which means that no selection on h_i is supported by S .

Example 8 A possible interface for our genomic example is $I = \{S_1, S_2\}$ where $S_1 = \langle \text{Sample, All} \rangle$ and $S_2 = \langle \text{Experiment, Chromosome} \rangle$. Examples of extractions using services S_1 and S_2 , respectively, are $e_1 = (S31, S32) \times (-\infty, +\infty)$ (which can be obtained using predicate $\text{Sample} \geq S31$) and $e_2 = (S11, S31) \times (R12, R22)$ (which can be obtained using predicate $(\text{Experiment} = \text{Exp1}) \wedge (\text{Chromosome} = \text{Ch2})$).

3.2.5.2 Dice Management

The main function of this process is that of determining the set F of missing dice to answer a given OLAP query. To this end, this process must be capable of executing dice operations; in particular, given a set Q of query dice and the set D of dice in the dice map (those currently available in the cube), it can compute their difference F using Algorithm 1. The basic idea of the dice difference operation is to split each dice in Q into fragments based on ranges “aligned” to the ranges in the dice of D , so that each resulting fragment is either included in a dice of D (in which case it needs not be loaded) or disjoint from all dice of D (in which case it is missing and must be loaded in its entirety).

Definition 8 (Range and Dice Fragmentation) Given range $r^i = (m', m'')$ of level l_1^i and an (ordered) set of members $M^i \in \text{Dom}(l_1^i)$, let $\overline{M}^i = \{m_1, \dots, m_p\}$ be the (ordered) subset of M^i included in r^i . The fragmentation of r^i according to M^i is the set of ranges $\text{Frag}_{M^i}(r^i) = \{(m', m_1), (m_1, m_2), \dots, (m_p, m'')\}$. Given dice $d = \times_i r^i$ and an n -ple of sets of members $M = \langle M^1, \dots, M^n \rangle$, where $M^i \in \text{Dom}(l_1^i)$ for $i = 1, \dots, n$, the fragmentation of d according to M is the set of dice $\text{Frag}_M(d) = \times_i \text{Frag}_{M^i}(r^i)$. The right/left openness/closure for the ranges of the dice in $\text{Frag}_M(d)$ is chosen in such a way that the fragmentation is disjoint and complete.

Remarkably, since $\text{Frag}_M(d)$ is based on the Cartesian product of ranges, it is the finest fragmentation that can be obtained starting from M . This ensures maximum flexibility in determining cheap extractions to obtain the missing dice since the optimization process (see Section 3.2.5.4) works by aggregation and does not allow further splitting of the input dice. As a further note, the reason why we must allow for open ranges in defining dice is that, since QETL is based on incremental loading, we generally do not know the complete domains of the levels in G_\perp . Indeed, if all members were known from the beginning, ranges could be easily defined with closed predicates only, thus avoiding unnecessary complexity.

Example 9 Consider again the example in Figure 3.5. Let $M = \langle \{S21, S22, S23\}, \{R14, R44\} \rangle$; the fragmentation of dice d'' according to M includes the 9 grey dice shown by dashed lines (note that member S21 is external to d'' , so it does not contribute to the fragmentation).

Algorithm 1 *DiceDifference*(Q, D)

Require: A set of query dice Q and a set D of available (disjoint) dice

Ensure: A set F of missing dice

```

1:  $F \leftarrow Q$ 
2: for all  $q \in Q$  do
3:   if  $\exists d \in D \mid q \subseteq d$  then                                ▷ Dice  $q$  is already covered by  $D$ ...
4:      $F \leftarrow F \setminus \{q\}$                                     ▷ ...so it is not missing
5:   else
6:      $O \leftarrow \{d \in D \mid d \sim q\}$                             ▷ Dice that overlap with  $q$ 
7:     if ( $O \neq \emptyset$ ) then
8:        $M \leftarrow \langle \emptyset, \dots, \emptyset \rangle$                     ▷ Members for fragmentation
9:       for all  $d \in O, i = 1, \dots, n$  do                          ▷ For each range of each overlapping dice  $d$ ...
10:         $M^i \leftarrow M^i \cup \text{Ends}^i(d)$ 
11:        ▷ ...  $\text{Ends}^i(d)$  returns both end members of the  $i$ -th range in  $d$ 
12:         $F \leftarrow F \setminus \{q\} \cup \text{Frag}_M(q)$                     ▷ Fragment  $q$  according to  $M$ 
13:  $F \leftarrow F \setminus \{f \in F \mid \exists d \in D, f \subseteq d\}$           ▷ Delete from  $F$  the dice covered by  $D$ 
14: return  $F$ 

```

Initially, Algorithm 1 considers all dice in Q to be missing (line 1). Then, for each dice q in Q it checks if there is an overlap with any dice in the dice map D (line 6). If not, q is entirely missing and stays in F . If some overlapping dice are found, the end members of their ranges are used to fragment q (line 12). Finally, we just have to delete from F the fragments of q that are already present in the dice map D (line 13). In case a dice q is completely included into a dice of D , it is simply removed from F (line 4). The details about the management of open/closed ranges are not shown in Algorithm 1 for the sake of simplicity, but they will be briefly commented in the following example.

Example 10 Consider the example in Figure 3.6, with $D = \{d, d'\}$, $Q = \{q\}$, $d = (\text{S11}, \text{S12}) \times (\text{R11}, \text{R14})$, $d' = (\text{S33}, \text{S24}) \times (\text{R25}, \text{R36})$, and $q = (\text{S31}, \text{S43}) \times (\text{R22}, \text{R45})$. Since dice d and d' are in the dice map (i.e., they have already been loaded in the cube), the missing facts that must be loaded to answer q are those in the grey-dashed area, that correspond to the following missing dice resulting from the dice difference operator (from left-top in Figure 3.6.a):

$$\begin{aligned}
d_1 &= (\text{S12}, \text{S33}) \times (\text{R22}, \text{R14}) \\
d_2 &= (\text{S33}, \text{S43}) \times (\text{R22}, \text{R14}) \\
d_3 &= (\text{S31}, \text{S12}) \times (\text{R14}, \text{R25}) \\
d_4 &= (\text{S12}, \text{S33}) \times (\text{R14}, \text{R25}) \\
d_5 &= (\text{S33}, \text{S43}) \times (\text{R14}, \text{R25}) \\
d_6 &= (\text{S31}, \text{S12}) \times (\text{R25}, \text{R45}) \\
d_7 &= (\text{S12}, \text{S33}) \times (\text{R25}, \text{R45})
\end{aligned}$$

However, the specific queries to be issued to load the missing dice depend on whether the ranges that define d , d' , and q are actually closed or open. If we assume that all ranges in d , d' , and q are closed (e.g., $\text{S31} \leq \text{Sample} \leq \text{S43}$), the actual situation is the one depicted in Figure 3.6.b. So it becomes clear that, for instance, the missing dice d_1 in this case must be left-open on S12 and left-closed on R22, while d_5 must be right-closed on S43 and

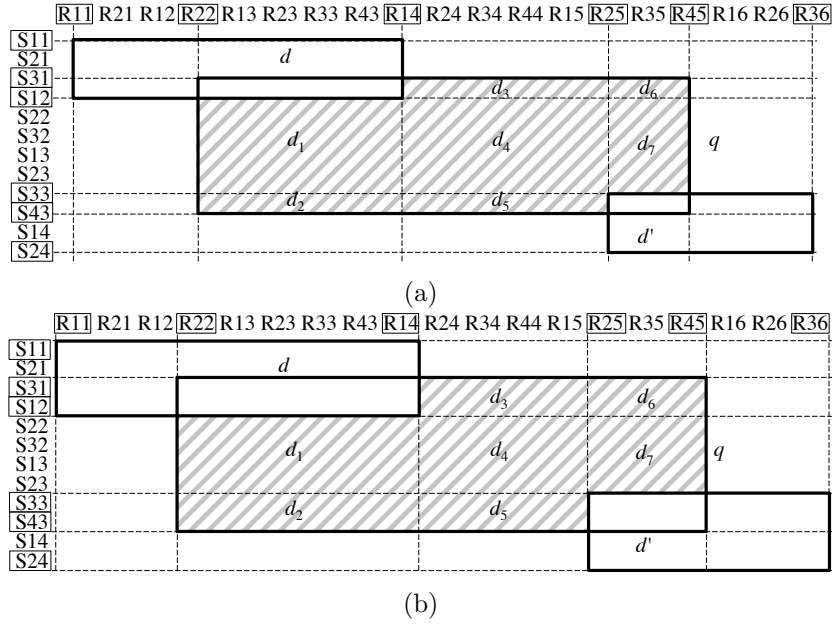


Figure 3.6: Dice difference on the dice map with generic ranges (a) and closed ranges (b)

right-open on R25.

As already stated and shown in Figure 3.4, the set of dice available at any time is stored in a dice map (D in Algorithm 1). In practice, the dice map is implemented by coupling a B⁺-tree index (to record, for each dimension, the members currently loaded in the dimension tables) and a list of dice (to keep track of the facts currently present in the fact table). All dice in the dice map are disjoint.

3.2.5.3 Dice Dropping Policies

The dice management process is also in charge of dropping some facts used for past queries from the cube to make room for the facts needed to answer new queries. To this end we implemented and compared three dropping policies. The first two, taken from the caching literature, are: LRU⁺ [32], a modification of the well-known *least recently used* policy, and *Cheapest by Size* (CS), which prioritizes the dropping of dice with lowest ratio of cost over size. When using LRU⁺, the dropping priority of a dice is updated every time a user query overlaps that dice (i.e., the query requires some facts included in the dice); specifically, the update accounts for how large the intersection between the query and the dice is. In CS, the cost of a dice is measured as the execution time of the extraction used to retrieve it, while its size is the number of facts it contains.

Both LRU⁺ and CS can yield good results with appropriate workloads but might otherwise perform poorly (see Section 3.2.6.1 for an experimental evaluation). Indeed, LRU⁺ considers the probability of a dice to be required again for a new query but ignores the

cost for reloading it in case of dropping, so it is better suited for workloads where most extractions have similar costs. Likewise, CS considers how much loading a dice costs but disregards its likelihood of being required again, hence it is better suited in situations in which the cost of extractions varies significantly. To overcome these limitations we devised a *hybrid* policy, HYB, which combines the statistics used in LRU⁺ and CS in a way that is much similar to the one used for the *size adjusted LRU* policy [32]. We extended that approach incorporating, besides dice size and age, the dice cost (as in CS) into the computation of the dropping priority to favor the persistence of more expensive dice. Formally, the dropping priority of a dice is defined as $p = \frac{\tau \cdot c}{s}$, where τ is the *aging factor* of the dice as defined in LRU⁺, while c and s are its cost and size, respectively. The lower p is, the higher the dropping priority is.

These three dropping policies are experimentally evaluated and compared in Section 3.2.6.1.

3.2.5.4 Optimization

When several missing dice must be loaded and different services are available, determining the cheapest set of extractions becomes an optimization issue related to both the specific services to be called and to the set of missing dice to be retrieved through a single extraction. Having a separate extraction for each missing dice could be quite expensive and time-consuming, for instance if most of the cost/time is paid to connect to the service rather than for data transfer and processing.

Different source data providers and different ETL processes can entail very different costs for extractions; the cost function to be used clearly depends on the features of the application domain and on how costs are measured (e.g., in terms of time, money, etc.). For the sake of flexibility we will not impose any specific constraints on the cost function, except that of being non-negative. For example, a simple family of cost functions that matches the application scenarios depicted in Section 3.2 is the one where a fixed cost for calling the service is summed to a cost proportional to the number $|e|$ of facts returned by extraction e . So in this case $cost(e) = \alpha_S + \beta_S |e|$, where $\alpha_S \geq 0$ and $\beta_S \geq 0$ depend on the service S used by e . When the application scenario is the pay-per-download one, α_S is the fee to be paid for each connection, while β_S is the cost per byte to be downloaded. In all of the other scenarios, α_S is the time needed to set up the connection while β_S is the time-per-byte needed to extract, transfer, and transform the data. To be independent of the sparsity of the cube and of the specific distribution of its facts, we will approximate the number of facts returned by extraction $e = \times_i r^i$ with the size of the corresponding dice, defined as $|e| = \prod_i |r^i|$. The problem of how to estimate the size of a dice when the level domains are not precisely known (because members are incrementally loaded together with facts) will be discussed in Section 3.2.5.6.

Given the cost function, the optimization process takes in input the set F of missing dice

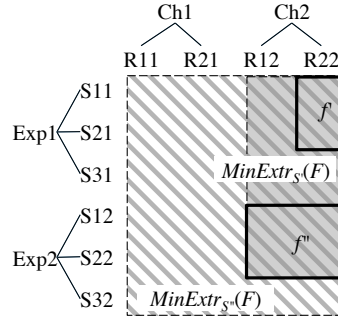


Figure 3.7: Minimal extractions with different services

produced by Algorithm 1 and produces in output a set E of extractions to be requested to the ETL. The specific problem to be solved to this end can be formulated as follows:

Problem 1 *Given a set F of (missing) dice and an interface I , find a set E of extractions such that (i) each extraction $e \in E$ uses a service in $S \in I$, (ii) $\bigcup_{f \in F} f \subseteq \bigcup_{e \in E} e$, and (iii) $\sum_{e \in E} \text{cost}(e)$ is minimal.*

Before we explain how Problem 1 can be solved, we need to show how a given set F of missing dice can be obtained by calling a service S of interface I .

Definition 9 (Minimal and Cheapest Extraction) *Let F be a set of dice, I be an interface, and S be a service. Then, the minimal extraction of F from S , denoted $\text{MinExtr}_S(F)$, is the smallest extraction e using S such that $f \subseteq e$ for each $f \in F$. The cheapest extraction of F from I , denoted $\text{CheapExtr}_I(F)$, is the extraction $\text{MinExtr}_S(F)$ such that $S \in I$ and $\text{cost}(\text{MinExtr}_S(F))$ is minimum.*

Intuitively, $\text{CheapExtr}_I(F)$ determines the cheapest way to fetch all the facts belonging to F using I .

Example 11 *With reference to Figure 3.7, let $F = \{f', f''\}$ with $f' = (S11, S21) \times (R22, R22)$ and $f'' = (S12, S22) \times (R12, R22)$. Let the interface include two services: $S' = \langle \text{Sample}, \text{Chromosome} \rangle$ and $S'' = \langle \text{Experiment}, \text{All} \rangle$. Then $\text{MinExtr}_{S'}(F) = (S11, S22) \times (R12, R22)$ (this extraction, in solid grey in the figure, is obtained calling S' with predicate $(\text{Sample} \geq S11) \wedge (\text{Sample} \leq S22) \wedge (\text{Chromosome} = \text{Ch2})$), and $\text{MinExtr}_{S''}(F) = (S11, S32) \times (R11, R22)$ (this extraction, in dashed grey in the figure, is obtained calling S'' with predicate $(\text{Experiment} \geq \text{Exp1}) \wedge (\text{Experiment} \leq \text{Exp2})$).*

Solving Problem 1 by enumeration is obviously incompatible with an interactive analysis scenario like ours. To reduce the problem complexity, we approach it as a clustering problem where each cluster corresponds to an extraction, i.e., to a set of dice whose facts are fetched by a single service call; note that, by doing so, we do not consider all solutions in which a single dice in F is further fragmented to be fetched using multiple extractions

Algorithm 2 *Optimize*(F, I)

Require: A set F of missing dice and an interface I

Ensure: A set E of extractions

```

1:  $C \leftarrow \{\{f\}, f \in F\}$  ▷ Create a clustering of singletons
2:  $E \leftarrow \{CheapExtr_I(c), c \in C\}$  ▷ Initialize the set of extractions
3: while  $|C| > 1$  do
4:   find  $c', c'' \in C$  s.t.  $c', c'' \in C \wedge c' \neq c'' \wedge \delta(c', c'')$  is minimum
5:   ▷ Find the most promising couple of clusters
6:    $C \leftarrow C \setminus \{c', c''\} \cup \{c' \cup c''\}$  ▷ Merge the two clusters
7:    $E' \leftarrow \{CheapExtr_I(c), c \in C\}$  ▷ Best extraction set of the new clustering
8:   if  $\sum_{e' \in E'} cost(e') < \sum_{e \in E} cost(e)$  then ▷ Compare the costs of the two extraction sets
9:      $E \leftarrow E'$  ▷ New extraction set
10: return  $E$ 

```

(which could allow the number of unnecessarily fetched facts to be cut down). Even with this simplifying assumption, the search space is large enough to be hardly explorable in its entirety. Indeed, given a set F of dice and an interface I , the size of the search space is $\sum_{k=1}^{|F|} \binom{|F|}{k} \cdot |I|^k$, where the first and the second terms represent, respectively, the Stirling number of second kind and the number of dispositions with repetitions.²

The greedy solution we propose to tackle Problem 1 is outlined in Algorithm 2 and is based on hierarchical clustering [37]. Starting from a clustering —i.e., a partition— of the dice in F where each cluster is a singleton, we proceed by iteratively merging the two most promising clusters, i.e., those with minimum distance. The inter-cluster distance function we use to this end is

$$\delta(c', c'') = \frac{cost(CheapExtr_I(c' \cup c''))}{|MBD(c')| + |MBD(c'')|}$$

where c' and c'' are two sets of dice and $MBD(F)$ is the minimum bounding dice of a set of dice F , i.e., the smallest dice g such that $f \subseteq g$ for each $f \in F$. To avoid favoring the merging of small clusters, the denominator of the distance function weighs the cost for fetching all the facts in the two clusters with the total size of the minimum bounding dice for the two clusters.

The merging process is iterated for $|F| - 1$ times to build a complete dendrogram. The clustering C generated at each iteration corresponds to a set of extractions defined as

$$E = \{CheapExtr_I(c), c \in C\}$$

Eventually, the clustering corresponding to an extraction set with minimum cost is chosen as the solution.

The overall computational complexity of Algorithm 2 is $O(|F|^3 \cdot |I|)$, where F is the set of missing dice and I is the interface exposed by the data provider. $O(|F|^3)$ is the total number of comparisons between clusters (at each iteration $O(|F|^2)$ comparisons are done,

²More precisely, $\binom{|F|}{k}$ counts the number of ways in which the set of dice can be partitioned into k extractions, while $|I|^k$ counts the number of ways in which such extractions can be mapped onto interface I

and the total number of iterations is $|F| - 1$). For each comparison, $|I|$ services must be evaluated to determine the cheapest extraction, from which the total complexity follows.

3.2.5.5 Filtering

The extractions determined by optimization are requested to the ETL component, which executes them by querying the provider and returns a set of facts to the filtering process. As previously mentioned, the interface exposed by the source data provider may allow for selecting the data to be extracted using predicates on some hierarchy levels only, so the facts returned may be a superset of those actually needed. The aim of this process is mainly to discard all the facts not required to answer the user’s current query, which requires the following check for each fact, whether it is included in one of the missing dice returned by Algorithm 1; since extractions could overlap, attention should be paid to the management of duplicates.

This process also determines the exact number of facts included in each extracted dice. As mentioned in Section 3.2.5.2, the exact number of facts per dice is used during the dropping process to decide how many dice must be dropped to make room for new facts (the granularity of a single drop operation is exactly one dice, i.e., a dice is either completely dropped or not dropped at all). Counting the facts per dice at this stage is necessary because the dice size used by optimization to estimate the cost of each extraction is imprecise due to the cube sparsity and to the partial knowledge of the level domains (see Section 3.2.5.6).

So far, we have assumed that, after each OLAP query, only the facts required to answer that query are loaded into the cube (*strict loading*). However, another viable approach is that of loading in the cube *all* the facts extracted (*loose loading*). The choice of the best approach depends on the particular situation at hand. For example, if the goal is just to minimize the processing time, strict loading might be more suitable because it does not overload the cube with unnecessary facts. Conversely, if data must be purchased from the provider, loose loading might be a better option. We remark that, from an implementation point of view, the only relevant difference between strict and loose loading concern the management of overlapping extractions: indeed, in case of loose loading, a dice difference operation must be performed to avoid representing overlapping dice in the dice map.

3.2.5.6 Dice Size Estimates

From a geometric point of view, the size of a dice as defined in Section 3.2.5.4 is simply the product of the cardinalities of its ranges. However we recall that, since QETL adopts an incremental loading strategy of facts, we might not know the (complete) set of members belonging to a given level—and thus to a given range. In this case we have no means to compute the exact size of a range, which must be estimated using the knowledge available.

This is the subject of this section, in which we will drop the hierarchy superscript i from all symbols to simplify the notation.

We start by observing that, inevitably, at least one level domain for each hierarchy must be completely pre-loaded into the cube to enable users to start the querying session, e.g., by selecting one member of interest. Remarkably, this is always feasible, even when $Dom(l_1)$ is too large to be managed in its entirety within a traditional cube. In fact, the roll-up functions establish a set of functional dependencies between the levels of each hierarchy, such that $l_j \rightarrow l_k$ for each $j < k$. As a consequence, the domain cardinality decreases for increasing levels along each hierarchy: $|Dom(l_j)| \geq |Dom(l_k)|$ for $j < k$. This ensures that, for each hierarchy, there is always one level l_λ , with $\lambda \geq 1$ ($l_\lambda \equiv l_{all}$ in the worst case), whose domain can be pre-loaded. We will assume that all levels l_j with $j > \lambda$ are pre-loaded too. For each pre-loaded level l_j , the size of the interval between any two members m' and m'' , $|(m', m'')|$, can be exactly determined by counting the in-between members.

For all other levels, for which the complete set of members is not known, we assume to know the domain cardinality, for example through meta-data exposed by the data source. This enables a rough estimate of the average outdegree $deg(l_j)$ of the members of each level l_j (i.e., the average number of members of l_{j-1} that roll-up to the same member of l_j ; conventionally, $deg(l_1) = 1$).

Intuitively, the size of a range is estimated through a hierarchy-based measure that uses, when possible, the pre-loaded levels (for a characterization of different hierarchy-based measures, see [38]). More precisely, the (worst-case) estimate for the size of range $r = (m', m'')$ is made as follows:

$$|r| = \begin{cases} |(RollUp^{l_\lambda}(m'), RollUp^{l_\lambda}(m''))| \cdot \prod_{j=1}^{\lambda} deg(l_j) & , \text{ if } LCA(m', m'') \in l_k, \\ & k \geq \lambda \\ \prod_{j=1}^k deg(l_j) & , \text{ otherwise} \end{cases}$$

where LCA represents the lowest common ancestor of two members. Two cases are distinguished, based on whether $LCA(m', m'')$ belongs to a pre-loaded level or not. Remarkably, in the first case it is possible to estimate the range size more precisely than in the second one, since we can exactly count the number of members between $RollUp^{l_\lambda}(m')$ and $RollUp^{l_\lambda}(m'')$. In case l_1 is pre-loaded (i.e., $\lambda = 1$), since conventionally $RollUp^{l_1}(m) = m$, we have $|r| = |(m', m'')|$ so an exact estimate is made. As to the levels that are not pre-loaded (l_j^i with $j \leq \lambda$), we observe that, as a result of the fact that all members of the same level are assumed to have the same outdegree, the more these levels are characterized by an irregular distribution of their members, the less range size estimates are precise.

Example 12 *With reference to Figure 3.8, $LCA(S11, S29) = All$. If $|Dom(Tissue)| = 5$, $|Dom(Experiment)| = 500$, and $|Dom(Sample)| = 5000$, then $deg(Tissue) = 100$ and*

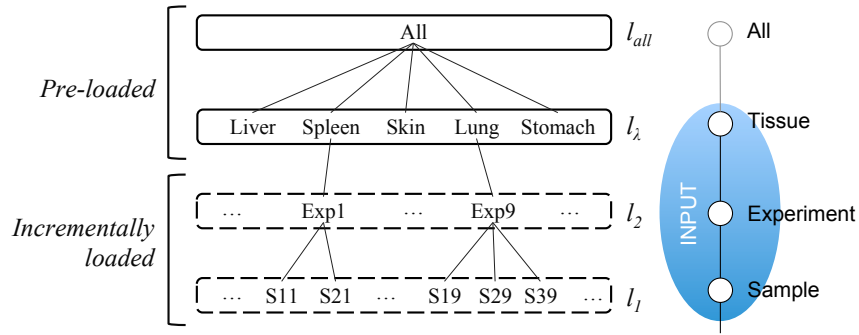


Figure 3.8: Pre-loaded vs. incrementally-loaded levels

$deg(\text{Experiment}) = 10$. So, $|(S11, S29)| = |(Spleen, Lung)| \cdot deg(\text{Sample}) \cdot deg(\text{Experiment}) \cdot deg(\text{Tissue}) = 3 \cdot 1 \cdot 10 \cdot 100 = 3000$. On the other hand, $|(S11, S21)| = deg(\text{Sample}) \cdot deg(\text{Experiment}) = 1 \cdot 10 = 10$ since $LCA(S11, S21) = \text{Exp1}$.

3.2.5.7 Additional Issues

For the sake of simplicity, in the previous sections we have shelved a couple of issues requiring a more specific discussion, namely, the implications of extracting and storing facts at different group-by's, how to deal with branched hierarchies, and how to support multiple measures.

OLAP is an exploratory process where data are analyzed at different granularities. In Section 3.2.5.1 we assumed that data are always extracted and loaded into the cube at their finest granularity, but this may determine an unnecessary computational effort because a huge amount of fine-grained facts must be extracted first and then aggregated to return a small number of coarse-grained facts. The problem of cutting aggregation costs is well-known in DW systems and is typically solved by pre-computing data at different aggregation levels, storing them in *materialized views*, and then exploiting an ad-hoc DBMS module, the *aggregate navigator*, to properly choose the smallest materialized view that can be used to answer a query. The term *data cube* is often used to denote fine-grained and coarse-grained data as a whole. Taking this into account, we claim that the QETL approach can be enhanced to operate at different aggregation levels (i) by extending the definition of a service to specify also the aggregation level at which it retrieves data; (ii) by extending the definition of a dice to specify also its aggregation level (in other terms, by building a dice map of the whole data cube); and (iii) by extending the optimization process so that it can choose extractions using services at different aggregation levels to cover the missing dice. Note that, to choose which is the cheapest service for loading a set of missing dice, the optimization process will have to trade-off between coarser services (that may not allow to express the right selection predicates) and finer ones (that may enable a precise slicing at the price of returning a large amount of fine-grained data).

As to the second issue we observe that, while the levels of a linear hierarchy are totally ordered, those of a branched hierarchy are only partially ordered. For instance, in our working example a **Sample** belongs to an **Experiment** that is related to a **Tissue**; besides, an **Experiment** is carried out in a **Laboratory**, but no roll-up relationship exists between **Laboratory** and **Tissue**. Without a total ordering between levels, no total ordering between members can be defined, which undermines the definition of dice and range. To easily cope with this as done by some multidimensional engines (e.g., Mondrian), a hierarchy with a branch can be duplicated so as to be replaced by two linear hierarchies, each based on a total order. As a side effect, the dice dimensionality will increase; however, all the proposed definitions and algorithms retain their validity.

The last issue concerns the support of multiple measures within a single cube. The simplest way to cope with this situation is to observe that, from a formal point of view, a cube schema with n dimensions and m measures is equivalent to a cube schema with $n + 1$ dimensions (where the $(n + 1)$ -th dimension is used as a sort of discriminator and takes m values, one for each measure) and one “generic” measure; this allows to transparently reuse the QETL approach.

3.2.6 Experimental Results

This section collects the main results of the tests we performed to evaluate the behavior of the QETL approach. We carried out all tests on a server machine quad core (3.6 GHz, 32 GB RAM, Windows 8-64 bit); all components were implemented in C++, and MySQL v5.6 was used as the DBMS. The average transfer speed of the connection used to extract the source data amounts at approximately 88 Mb/sec.

We based our tests on the **MAPPING** schema (whose simplified version is shown in Figure 3.2). Source data come in the form of 22 500 GTF files made available through FTP; each file contains an average of about 130 000 mappings, so the mapping cube includes about $3 \cdot 10^9$ facts overall. We set the maximum cube capacity to 1/300 of the complete cube, i.e., to 10^7 facts.

Figure 3.9 shows the star schema used in our ROLAP implementation, which extends the simplified one shown in Figure 3.2. From a design point of view, the dimension table for regions has been partially snowflaked since regions are loaded incrementally while the domains of the levels following **Region** in the **REFERENCE** hierarchy are pre-loaded. Noticeably, while in traditional ETL indexes are dropped just before loading for performance reasons, this would not be convenient in QETL because loading is incremental. Therefore, indexes are kept enabled while loading and dropping facts. Unfortunately, the time required to rebuild an index depends not only on the newly loaded data but also on the previous ones, which adds a significant overhead to the performance of QETL even for small-size queries. Another peculiarity of the presented star schema is the **DICE_ID**

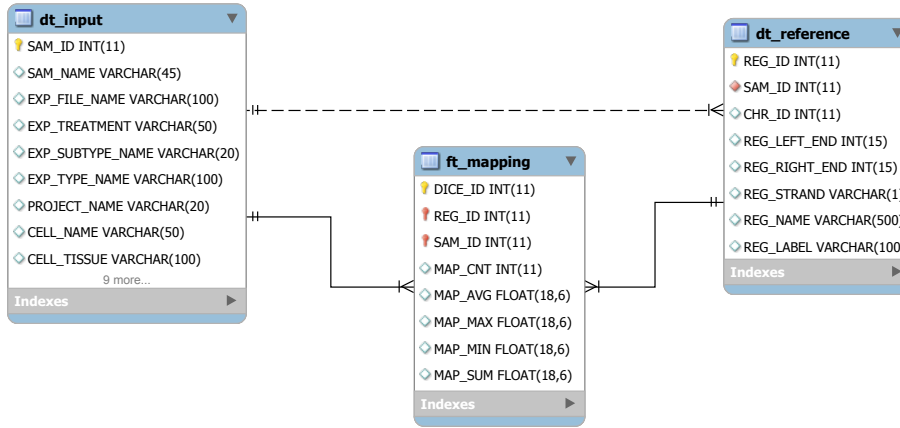


Figure 3.9: Star schema for MAPPING

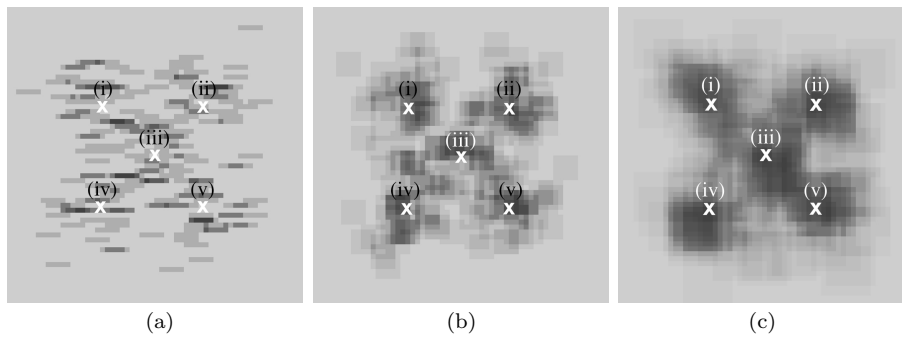


Figure 3.10: Hit distribution for small (a), medium (b), and large (c) queries (darker areas correspond to more frequently queried facts, crosses mark OLAP session centroids)

attribute in the `ft_mapping` table; this attribute is used to keep track of which facts each dice contains, thus remarkably simplifying the dropping process.

An OLAP session is a sequence of queries that tend to hit contiguous facts. To simulate this, we created five separate query sequences (numbered from (i) to (v)); the 40 queries in each sequence are normally distributed around a centroid which represents the focus of the analysis. To address a wider set of situations, we considered two different querying scenarios: *single-user*, where one user carries out five OLAP sessions one after the other, so the 5 sequences are concatenated from (i) to (v); and *multi-user*, where five different users concurrently carry out one OLAP session each, so the queries of the 5 sequences are interleaved. Finally, we considered three different query sizes, namely *small*, *medium*, and *large*, which differ for the average number of facts they require. Figure 3.10 shows the hit distribution for small, medium, and large queries (sequence centroids are marked with a cross). The three query sizes combined with the two querying scenarios result in the six workloads summarized in Table 3.1.

The workloads defined above are not very suitable for testing the three dice dropping policies described in Section 3.2.5.3 since all queries have similar costs; in absence of query

<i>Workload</i>	<i>Scenario</i>	<i># Users</i>	<i>Query size</i>	<i>Avg # facts per query</i>	<i># Files per query</i>
W_{small}^{single}	single-user	1	small	23 K	5
W_{med}^{single}	single-user	1	medium	400 K	25
W_{large}^{single}	single-user	1	large	2 M	100
W_{small}^{multi}	multi-user	5	small	23 K	5
W_{med}^{multi}	multi-user	5	medium	400 K	25
W_{large}^{multi}	multi-user	5	large	2 M	100

Table 3.1: Workload features

diversity, the CS policy cannot show its benefits and the two other policies, HYB and LRU⁺, obviously perform better. To enable a fair comparison we defined a synthetic version of workloads W_{med}^{single} and W_{med}^{multi} , which we call S_{med}^{single} and S_{med}^{multi} , respectively, where an overhead has been added to some queries to simulate a workload with high- and low-cost queries; specifically, with reference to Figure 3.10, all queries of sessions (i) and (ii), and half queries of session (iii) have a higher transformation cost than those in the other sessions. Note that these synthetic workloads are used only for testing the dropping policies in Section 3.2.6.1, while the original workloads are used for all the tests in the other subsections.

To model the cost of an extraction we used the function described in Section 3.2.5.4. The values of parameters α and β have been estimated as 2.3 seconds and $1.17 \cdot 10^{-5}$ seconds, respectively. The former is the average time required to connect to the FTP service exposed by the GenData repository (including authentication). The latter is the average time required to download and transform a fact (i.e., 85,470 tuples are extracted and transformed in 1 second). Noticeably, β does not take into account the time required to load data into the cube since, for a given set of missing dice, the number of loaded facts is the same regardless of the optimization outcome.

For a better understanding of what stress the system is subject to in these tests, in Figure 3.11 we show the number of facts currently stored in the cube and the cumulative number of facts loaded after each query in the three single-user workloads. For workload W_{small}^{single} , a total of $3.3 \cdot 10^6$ facts are loaded, which does not saturate the cube so no facts need to be dropped. When dealing with larger queries, dropping is necessary to make room for the facts required by current OLAP queries when the cube size reaches its capacity. For workload W_{medium}^{single} , a total of $2.3 \cdot 10^7$ facts are loaded, which is about twice the cube capacity; dropping is necessary starting from the 60-th query. Higher volumes of data are processed by workload W_{large}^{single} , which loads a total of $5.3 \cdot 10^7$ facts and starts dropping already from the 23-rd query.

The tests are organized as follows: in Section 3.2.6.1 we evaluate the dice dropping policies, in Section 3.2.6.2 we discuss the effectiveness of the QETL approach by focusing on how intensively it reuses data, while in Section 3.2.6.3 we compare QETL with the chunking approach. Finally, in Section 3.2.6.4 we provide a detailed analysis of the execution times

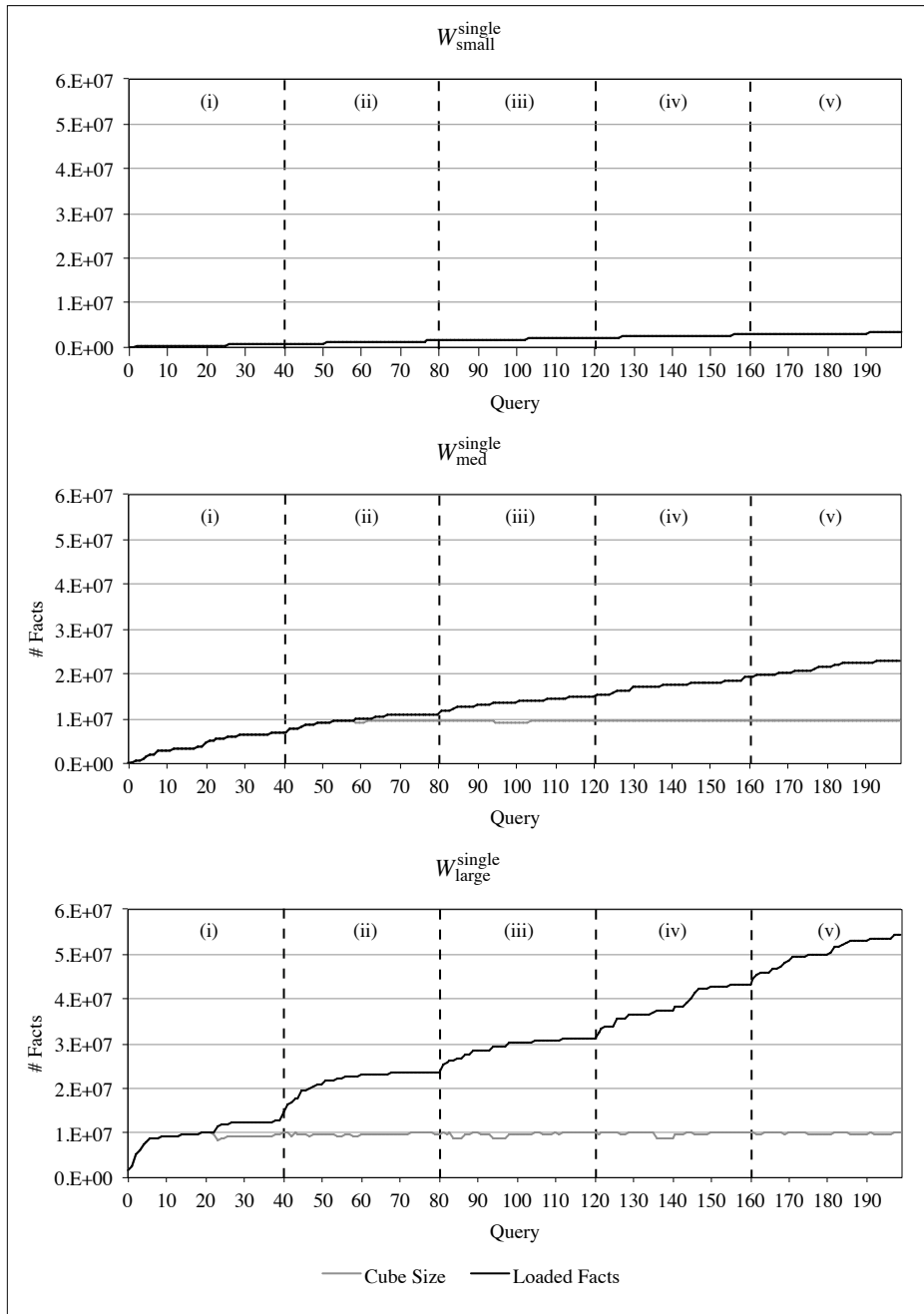


Figure 3.11: Cube size and cumulative number of loaded facts after each query for single-user workloads; dashed lines mark the transitions between separate OLAP sessions

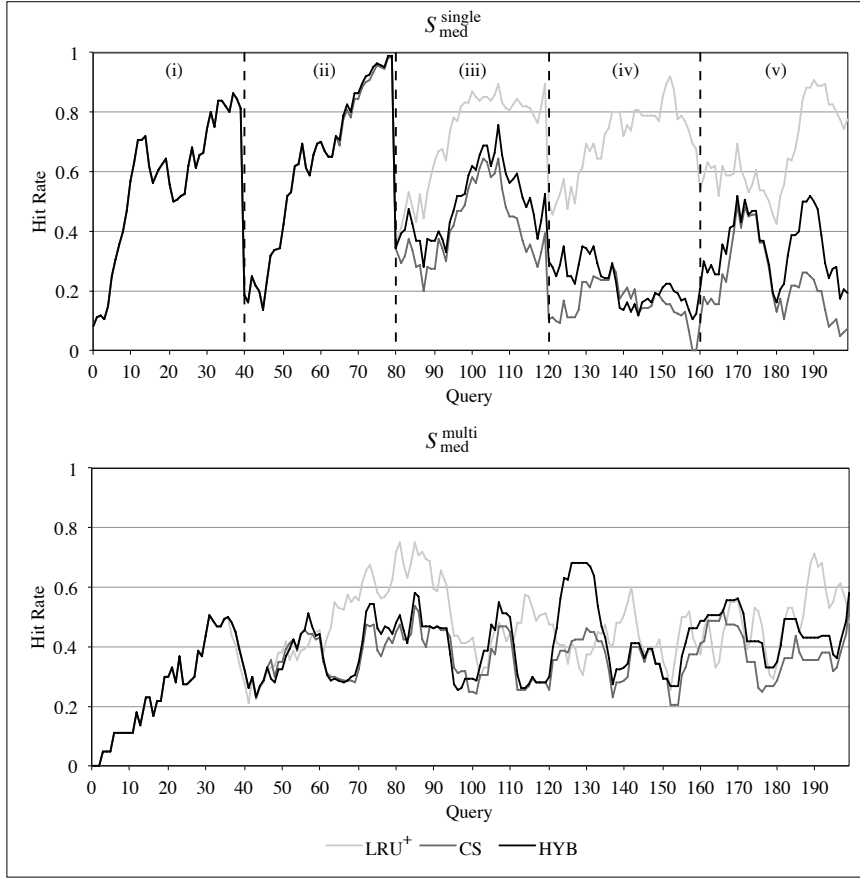


Figure 3.12: Hit rate per query for each dropping policy using synthetic workloads S_{med}^{single} and S_{med}^{multi}

and in Section 3.2.6.5 we give some insights on how sensitive the optimization process is to the cost function. Note that in all tests, except the ones in Section 3.2.6.1, the dropping policy employed is HYB due to its adaptivity to different types of scenarios.

3.2.6.1 Dropping Policy Analysis

The first test is aimed at analyzing the impact of policies on data reuse in terms of *query hit rate*, i.e., of the percentage of facts already loaded in the cube over the total number of facts required by each query. Figure 3.12 shows the trend of hit rate for each policy, both in the single- and multi-user scenario. In both scenarios the hit rate is the same for all policies until dropping starts, which happens around query 60 and query 38 for S_{med}^{single} and S_{med}^{multi} , respectively. The main difference emerging after this point is that CS and HYB behave similarly and always worse than LRU⁺ (this is less evident with S_{med}^{multi}). The reason for the wider gap between LRU⁺ and the other two policies in S_{med}^{single} is twofold: (i) the five simulated OLAP sessions are sequential rather than interleaved, thus the dice loaded farther in the past are less and less likely to be required again; (ii) the queries belonging to the first three sessions (see Figure 3.10) are more expensive, which leads CS

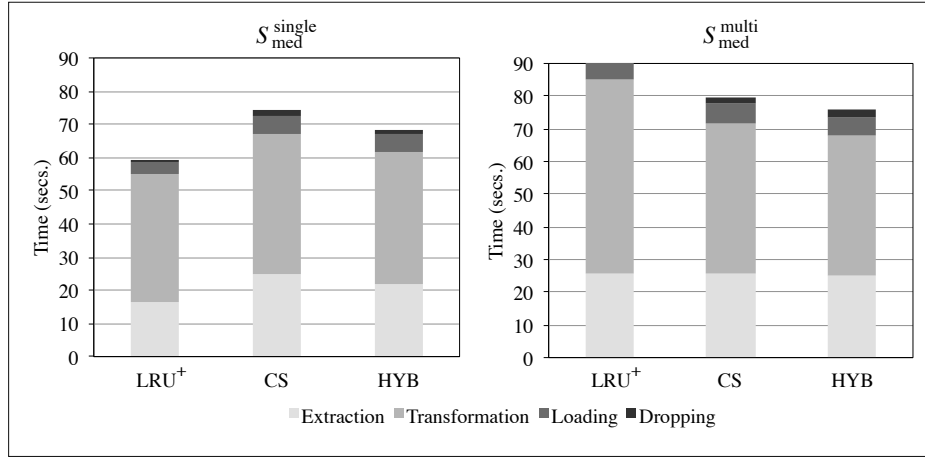


Figure 3.13: Breakdown of execution times per query for each dropping policy using synthetic workloads S_{med}^{single} and S_{med}^{multi}

to retain the dice generated by them though later on they will be required less and less. As to workload S_{med}^{multi} , since OLAP sessions are interleaved, the gap narrows considerably but still CS and HYB score lower because they still tend to favor the dropping of dice belonging to sessions (iv) and (v) even when they are less frequently used.

Considering only the hit rate as a measure of effectiveness can be misleading, because the goal of a dropping policy is that of reducing costs and not that of increasing the hit rate per se. Figure 3.13 shows a breakdown of the average execution time of queries for each dropping policy. As expected, in the single-user case LRU⁺ out-performs both CS and HYB by a noticeable margin, for the same reasons explained above. Conversely, in the multi-user scenario HYB takes the lead, closely followed by CS and finally LRU⁺. Even if the hit rate of CS is significantly lower than that of LRU⁺, the former still manages to better contain query costs. Specifically, CS is almost as efficient as HYB at reducing the transformation costs (which is where the synthetic overhead has been added); on the other hand, extraction, loading, and dropping times are higher in CS since they strictly depend on the hit rate.

In the light of these results, in all tests of the following subsections we adopt the HYB policy, which appears to be the most effective in the average case.

3.2.6.2 Reuse Analysis

Figure 3.14 shows how the hit rate changes during the sessions, focusing on the comparison between single- and multi-user workloads. For small-size queries, there is no difference between the overall average hit rates of the single- and multi-user workloads (because queries are the same and there is no dropping). This is because all the facts processed can be stored in the cube and no dropping is necessary. Besides, as shown in Figure 3.10(a),

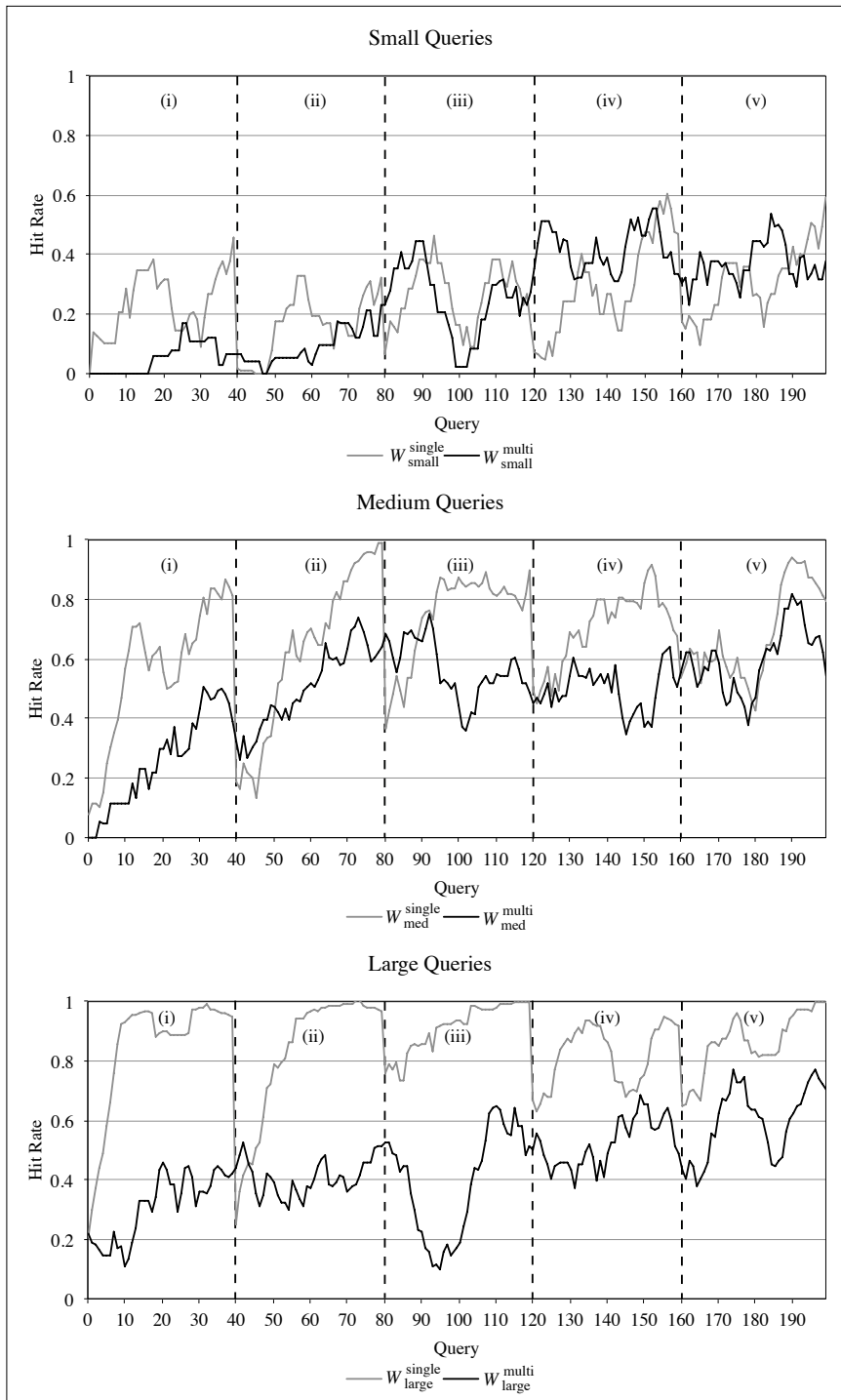


Figure 3.14: Hit rate per query for each workload

the overlap between small-size queries is much lower than that of medium- and large-size queries, which results in a lower hit rate. For medium- and large-size queries, the difference between single- and multi-user workloads becomes more significant showing that, due to the contiguity of queries within OLAP sessions, single-user workloads achieve higher reuse. In particular, when 5 users work concurrently the hit rate tends to smoothly increase; it is normally higher when a single user is working, with sharp drops when the user changes her analysis goal (immediately after each dashed line in Figure 3.14). However, even in the latter case, hit-rate drops tend to become less deep from left to right: this is mainly due to inter-session overlaps, which enable reusing facts loaded during previous sessions. This is confirmed by Figure 3.10, where there is no sharp distinction between OLAP sessions, in particular for medium- and large-size queries.

Another observation we can make by comparing the three charts in Figure 3.14 is that the single-user workloads show a constant gain in hit rate when moving from small- to large-size queries, while the same is not true for multi-user workloads. This seems counterintuitive as large-size queries overlap with each other much more than other queries; however, they require so many facts that the frequency of dropping is significantly higher.

Another angle for observing the impact of data reuse is shown in Figure 3.15, which gives a breakdown of the execution times for the workloads featuring medium-size queries (the times spent for dice difference and optimization are not singled out as they are negligible). As expected, the trend shown by Figure 3.15 is symmetrical to the one of Figure 3.14. On average, around 50% of the time is spent on extraction, around 25% on transformation, 20% on loading, and lastly, 5% on dropping. Figure 3.15 clearly shows the effectiveness of data reuse in single-user workloads; for instance, for queries from 1 to 40 in W_{med}^{single} , the total time decreases to one quarter and then it suddenly raises when the user changes her goal. Execution times for the multi-user workload initially show a decreasing trend up to a point where they become more stable due to the flattening of the hit rate.

3.2.6.3 Comparison with Chunking

In this section we compare the dice-based representation of facts described in Section 3.2.5.1 with the chunk-based one introduced in [35]. The comparison is made in terms of cost saving achieved with QETL by using either dice or chunks to represent stored facts and extractions. Before discussing the results we emphasize that, as discussed in Section 3.2.4, the original chunking approach can be applied in the QETL scenario only with some limitations:

- To define the chunk boundaries, chunking assumes that the level domains are known beforehand. Thus, we could not extend our comparison to the levels, such as **Region**, that are subject to incremental loading of the members.
- Facts must be physically organized in such a way that those belonging to the same

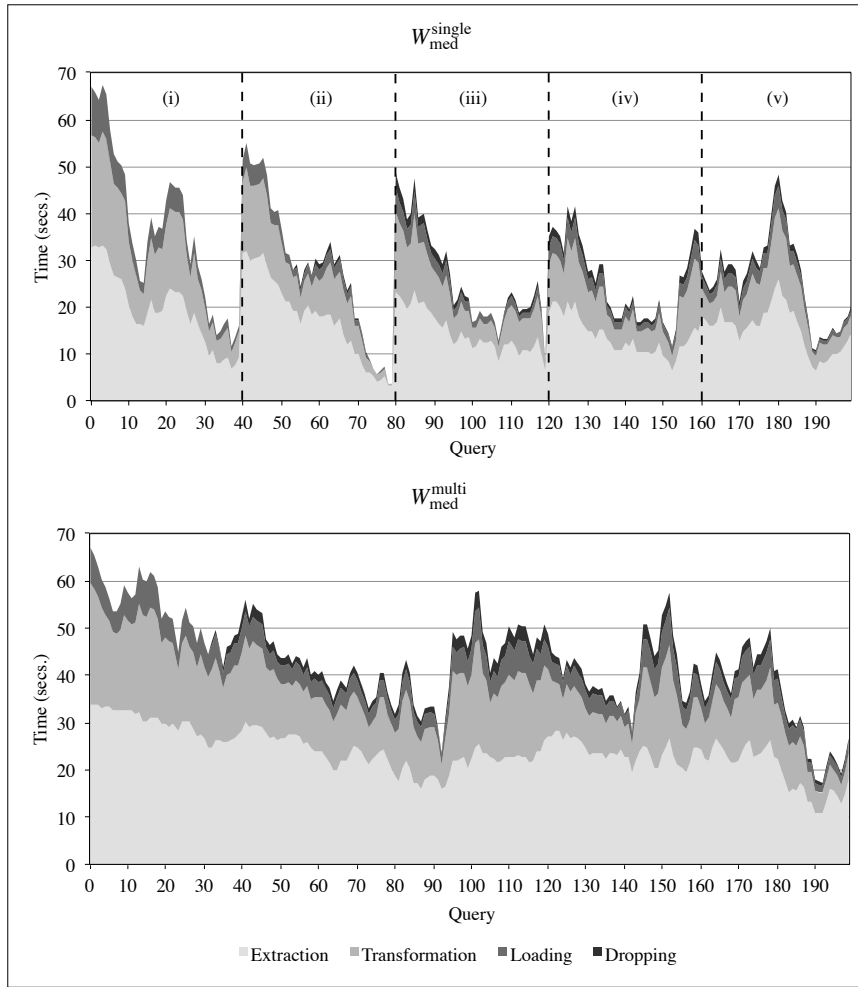


Figure 3.15: Breakdown of execution times per query for medium-size queries

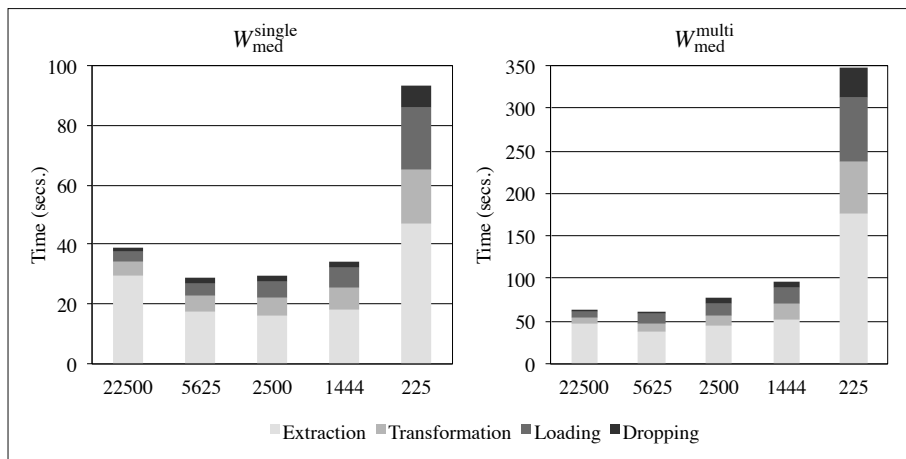


Figure 3.16: Average execution times of workloads W_{medium}^{single} and W_{medium}^{multi} with different numbers of chunks

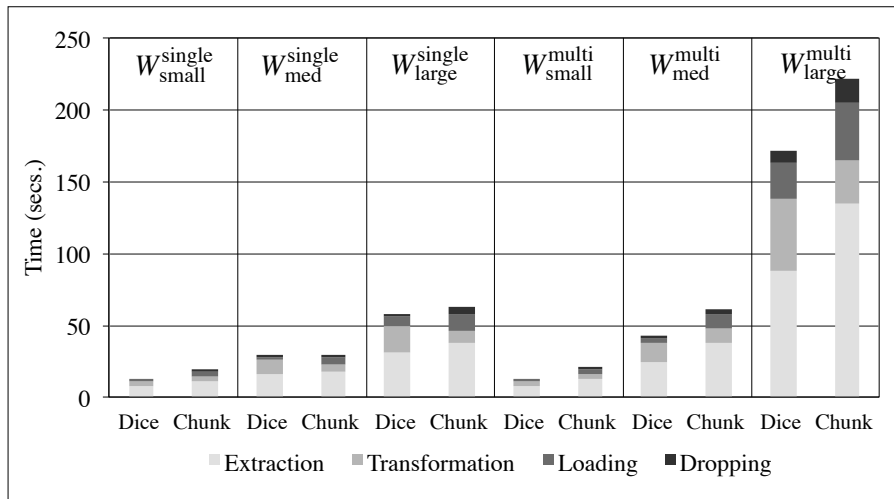


Figure 3.17: Comparison of dice- and chunk-based representations in terms of average execution times for each workload

chunk are stored contiguously [35]. In general, while it is easy to create a chunked file for the facts already loaded in the DW, this is often impossible for data stored in an external source. In our setting, we naturally have a chunk-friendly storage for source data since mappings are organized in files, each representing a specific input sample.

- Chunk shape and number must be fixed beforehand. To determine the best setting for chunking we run two tests, one for W^{single}_{med} and one for W^{multi}_{med} . The results are shown in Figure 3.16, where the average execution time of queries is shown for different numbers of chunks. It turns out that, for both workloads, the optimal number of chunks is 5625, so this is the value we used for tests.

Figure 3.17 shows the average execution times for each workload and approach. On average, the dice-based approach is 23% faster than the chunk-based one. While the gap is noticeable for multi-user workloads, the same is not true for single-user ones. This can be easily explained by recalling that one of the main differences between chunks and dice is that chunks have fixed size and shape, while dice adapt to the current query. This means that it is not always possible to perfectly represent a user query through a set of chunks; indeed, in most cases a query translates into a bigger envelope that covers more facts than those actually required. This behavior can be easily observed in W^{single}_{small} and W^{multi}_{small} , where the dice-based approach does not need to perform dropping, while the chunk-based one does. Since in single-user workloads queries are contiguous, loading more facts than (immediately) required does not result in unnecessary overhead, because those extra facts will probably be soon required anyway.

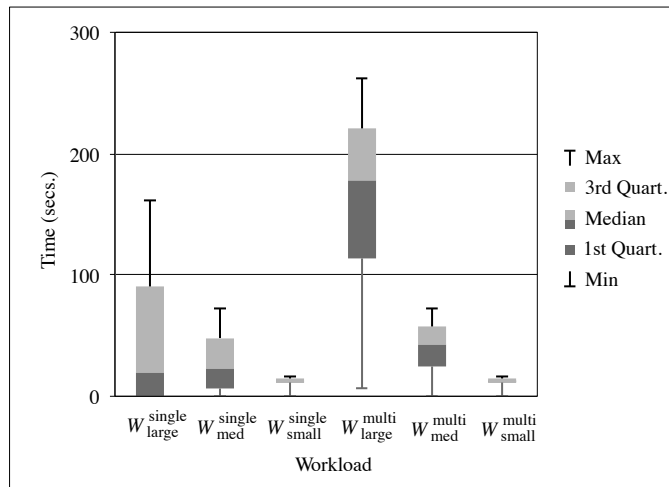


Figure 3.18: Distribution of execution times per query for each workload

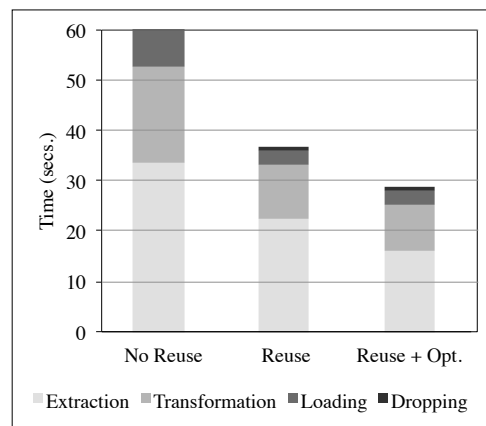


Figure 3.19: Breakdown of average execution times per query with workload $W^{\text{single}}_{\text{med}}$ for different versions of QETL

Table 3.2: Average number of extractions and fetched facts with different cost parameters for workload W_{medium}^{single}

α	β	# extractions	# fetched facts
0	$1.17 \cdot 10^{-5}$	1.48	1.42M
2.3	0	0.78	1.66M
2.3	$1.17 \cdot 10^{-5}$	1.17	1.42M

3.2.6.4 Efficiency Analysis

Figure 3.18 shows the distribution of the execution times for all six workloads. More specifically, Figure 3.19 aims at showing to what extent the different components of QETL contribute to the overall efficiency. To this end we consider three versions of QETL: in the *no reuse* version, all the facts required to answer each single query are extracted, loaded, and dropped as soon as the query is answered; the *reuse* version corresponds to the QETL approach without optimization, so each missing dice obtained by dice difference determines an extraction to be issued to the ETL; finally, *reuse + opt.* is the full QETL approach. The first observation we make is that both reuse and optimization positively impact on efficiency. Indeed, the maximum efficiency corresponds to *reuse + opt.*, followed by *reuse* and by *no reuse*; more specifically, extraction times decrease as reuse and optimization come into play. Since each extracted fact has to be transformed, transformation times decrease with extraction times. As expected, loading times are reduced by reuse but not by optimization (optimization reduces the number of extracted facts, but the loaded facts are the same). We also note that dropping times are negligible in the *no reuse* version because at each time the cube stores only the data required for the current query, and these data are dropped all together. In the other cases, dropping times follow the same trend as loading because the quantity of dropped facts is strictly correlated to the quantity of loaded facts.

We close this section by observing that, clearly, the worst-case condition for QETL is workloads where the hit rate is null or negligible and the quantity of data to be extracted and loaded is very high (either because the cube capacity is a small fraction of the complete cube, or because the queries are scattered across the cube). A null hit rate implies no reuse, so by comparing the first and last columns of Figure 3.19 we can conclude that the execution time for the worst case is about twice that of the average case.

3.2.6.5 Cost Function Analysis

Focusing again on workload W_{medium}^{single} , in this section we briefly comment on the impact on optimization of different values of α and β in the extraction cost function. To this end, in Table 3.2 we show the average, for each query, of the number of extractions issued to the ETL component and of the number of facts fetched through such extractions. As expected,

with $\alpha = 0$ the optimization algorithm seeks to minimize the amount of fetched facts, even if this requires several extractions. Conversely, when $\beta = 0$, optimization always returns one extraction (except when no missing dice are returned by dice difference). To better understand the results regarding the number of extractions, we remark that the average number of missing dice in input to optimization is 2.8. This value is an upper bound to the number of extractions since, as explained in Section 3.2.5.4, Algorithm 2 never splits the dice it gets in input. Noticeably, even with $\alpha = 0$, the number of extractions is significantly smaller than the aforementioned upper bound. This can be ascribed to the coarse extraction granularity of the service exposed by the GenData interface. Indeed, the coarser the granularity, the more likely for the cheapest extraction induced by a dice to also include other dice.

3.2.7 Wrapping up QETL

We have presented QETL, an approach to incremental on-demand ETL based on the query-extract-transform-load paradigm. Our approach is beneficial within scenarios in which traditional batch ETL is unfeasible or inconvenient for either time, space, or cost reasons. Essentially, in QETL a cube is operated as a sort of cache to enable data reuse for single and multiple users. To achieve this result we adopt a framework based on the dice construct, which allows to compactly abstract multidimensional data. On top of this building block we define the dice difference operation and a heuristic to find the cheapest extraction. More precisely, dice difference enables an easy computation of the facts required to answer a query but still missing from the cube; once the missing facts have been identified, the proposed heuristic finds the cheapest set of extraction queries to be issued to the data sources. The experimental results show that the execution times of QETL are compatible with those of OLAP querying in traditional DW settings (which normally do not exceed some minutes), and that both data reuse and extraction optimization successfully contribute to the approach efficiency.

We emphasize that QETL is meant to be applied in OLAP scenarios, i.e., where front-ends supporting multidimensional query languages such as MDX are used. Some recently-emerged tools (e.g., Tableau) can generate both multidimensional queries (on which QETL can easily operate) and more generally non-multidimensional queries. The problem with a non-multidimensional query is that it could violate our assumption on hierarchy-based lexicographic order (see Section 3.2.5.1), hence it may map to a possibly large set of dice (even one dice for each single piece of data required); in this case the QETL approach could still work in principle, but it would become impractical.

Our future work will be mainly aimed at improving the overall effectiveness of the QETL approach. One of the current limitations of QETL is the lack of a proper support and testing for the management of facts at different levels of granularity. Indeed, while the problem has been outlined in Section 3.2.5.7, a formal definition and testing of the mentioned

solution are due. Also, the optimization process could be enhanced to incrementally learn additional statistics of the data source (e.g., their data distributions) to improve the resulting extractions. Furthermore, a significant speed-up of some QETL sub-processes could be achieved using a different architecture from the one currently adopted; for instance, employing a hybrid in-memory/on-disk storage would lead to a significant reduction in loading and dropping times, though at the price of losing some data durability.

3.3 Building Adaptive Cost Models for Web Services

As already discussed, many data sources are nowadays hidden behind web services whose functionalities can range from simple access points for DBMS to complex analytics platforms. To optimize the extraction of data from these services it is important to be able to estimate the cost of each query. In several situations, especially in the database domain [39], the behavior of a system can be described through a model that estimates the query costs. The definition of cost models is often done manually by analyzing the system internal algorithms and by gathering various statistics such as data distributions. In the web service domain, due to the strong encapsulation of the applications, the only information that can be reliably obtained is the service interface definition; as a consequence, a web service must be considered as a black box for which analytically defining a cost model is unfeasible. Furthermore, since the web service behavior depends on multiple hidden factors, execution costs can drift with time; to avoid wrong estimates, the cost model must be able to reactively self-tune so as to accommodate these function drifts.

Example 13 *Grid is a Web Processing Service-compliant service of the Open Geospatial Consortium deployed at the Jet Propulsion Lab (<https://co2.jpl.nasa.gov/developer/>) to perform research on atmospheric CO₂. Grid enables the querying of data collected by several NASA Earth Science missions and can be invoked by clients through GET or POST requests. The querying interface exposed by Grid includes 27 parameters; essentially, each query returns a dataset of CO₂ measurements taken over a geographical area during a period of time. The average cost measured on a set of 30000 uniformly-distributed queries over a time span of 10 months is about 120 seconds, with a standard deviation of 82 seconds. The factors that impact the most on the query cost are the length of the time period and the extension of the geographical area; for instance, while a query returning the measurements for Europe during a time span of 60 days takes around 60 seconds to execute, while a query for the same zone over 90 days takes around 120 seconds. Function drifts periodically occur for the service; for instance, in May 2016 we experimented a reduction of the average query cost of about 30%, presumably due to an enhancement in the hardware infrastructure.*

Although some approaches to black-box cost modeling are available both in the DBMS and web-service literature, to the best of our knowledge none of them can (i) forecast the behavior of a generic web-service interface and (ii) self-tune when an unexpected drift in the cost function takes place. To address these limitations we propose an approach called *Tiresias* that, given a web service exposing an interface with a fixed number of parameters, initializes and actively adapts a model to accurately predict query costs. The cost model is represented by a regression tree trained through two interleaved querying cycles: a passive one, where the costs measured for user-generated queries are used to update the tree, and an active one, where the service is probed through system-generated queries to initialize the tree and adjust it whenever a function drift makes the estimate

accuracy unsatisfactory.

Overall, the main contributions of this work are:

- (i) An architectural framework for deriving cost models of web services using their public interfaces only (Section 3.3.2).
- (ii) An extension of the SAIRT algorithm [4] that incorporates multiple linear regression models with the result of improving the overall accuracy while keeping training costs compatible with the requirements demanded by streaming applications (Section 3.3.4).
- (iii) An active learning algorithm that initializes the cost model and dynamically adjusts it in case of function drift (Section 3.3.5).
- (iv) A set of experimental tests performed on both real and synthetic datasets to evaluate Tiresias in terms of efficiency and effectiveness (Section 3.3.6).

The outline for the contribution related to cost models for web services is completed by Section 3.3.1, which discusses the related work, Section 3.3.3, which provides the formal background, and Section 3.3.7, which draws the conclusions.

3.3.1 Related Literature

3.3.1.1 QoS Prediction

A research theme that bears many similarities with the problem tackled in this work is that of QoS prediction for web services. In contexts such as cost-based service composition [40], it is often important to be able to predict service run-time performances to optimize computations and reduce risks of *Service Level Agreement* (SLA) violations. Systems like WSPred [41] try to make time-aware and personalized QoS predictions by analyzing latent features of users, services, and time by means of tensor factorization, while [42] focuses on predicting response time by modeling service behavior using HMM. Finally, in [43] the authors propose a solution for QoS estimation specifically tailored for multimedia services (e.g., video streaming, VoIP, etc.). Several solutions exist for making predictions taking the context of the user (e.g., her geographical location) into account but, to the best of our knowledge, none of them has been devised to work with computation-intensive services exposing complex interfaces.

3.3.1.2 Cost Modeling with Machine Learning

Machine learning techniques have been previously used in the context of database optimizers to improve performances. The work in [44] explores the problem of building cost models of

local databases in a *Multidatabase System* (MDBS) using regression analysis. In the area of autonomous workload management, the solution described in [45] defines a novel tree structure called *PQR Tree* for query execution time ranges prediction; more recently, [46] defined cost models at different granularities, one at plan-level and one at operator-level. All these approaches, due to the features used, need information that common web services do not make accessible through public APIs (e.g., query execution plans and system resources measurements); furthermore, they are not self-tuning (although [46] can be trained online) and need to be rebuilt whenever the modeled system changes significantly. Several solutions have been proposed even outside the relational databases area; specifically, [47] tries to model execution costs of map-reduce jobs, while [48] and [49] are focused on XML and object-relational databases, respectively. Particularly relevant to our work is the approach described in [49], which proposes a self-tuning cost modeling solution based on quad-trees to predict *User-Defined Functions* (UDF) execution times. The main limitations of [49] are that it supports only ordinal features and that its self-tuning capabilities are limited since the model is incrementally updated with new data but obsolete observations are not discarded (except for compression purposes).

3.3.1.3 Database Histograms

In the area of database research, a lot of effort went into the development of smart techniques to gather selectivity statistics (in the form of histograms) with the aim of improving query execution plans. These approaches adopt either a pro-active or feedback-based gathering process. Pro-active solutions such as MHIST [50] employ data scans on the whole database, thus suffering from poor scalability with big tables. On the other hand, approaches like STHoles [51] and ISOMER [52] gather statistics focusing on data required by the current workload (i.e., user queries) and use this feedback as a guide to avoid scanning data of low interest. Finally, a variation of the statistics gathering problem in relational databases arises in presence of limited query expressivity, as in the context of *hidden databases* [53, 54]; in this scenario, both the query formulation and the result output could have several constraints such as the possibility to filter using only one value at a time, restricted cardinality of the results, etc. While quite mature, these techniques are not suited for predicting response times since common relational database techniques cannot be used in any context where the service has limited query expressivity, and both common relational and hidden database approaches fall short whenever a service involves non-trivial computations. Indeed, even if data statistics can bring useful insights, they cannot always be used to infer response times since they do not give information on service processes.

3.3.1.4 Active Learning

A considerable research effort is currently being put into the area of active learning [55] (also referred as *optimal experimental design*), which is a subfield of machine learning that aims at minimizing labeling costs (e.g., the time spent on manually annotating pictures with the corresponding categories) by letting the learning algorithm choose which queries should be issued. These techniques are of utter importance when gathering a large training set is difficult due to time or monetary costs, for instance in image and text classification, biological experiments, etc. One of the latest challenges in this field is the design of algorithms that can reactively adapt to function drifts [56, 57]. In [56] the authors show an evaluation of different learning approaches and their capabilities in effectively detecting and adapting to function drifts, while [57] employs active learning for sentiment analysis of tweets to infer financial predictions. The problem of learning a non-stationary function has been generally tackled in the context of streaming data, but indeed this specific topic has yet to reach maturity in an active learning context. As a final remark, none of the approaches regarding the construction of cost models that we previously introduced incorporates active learning mechanisms that enable the construction of a more dynamic and self-adaptive model.

3.3.2 Approach Overview

As sketched in Figure 3.20, Tiresias works as an intermediary between a *client* and a *service*, and uses a cost model of the server to deliver query cost estimates (yellow arrows in Figure 3.20) to the client. To train the cost model, it features two interleaved querying cycles: a passive and an active one. In the *passive querying cycle* (green arrows), each query Q sent by the client to the service for execution is intercepted by Tiresias, which waits for the response from the service to update the cost model with the measured cost y for Q (called *event* from now on). In the *active querying cycle* (red arrows), the service is probed through system-generated queries, aimed at (i) addressing the cold start problem by building a cost model from scratch with minimal user intervention; and (ii) dynamically adjusting the cost model whenever the estimate accuracy becomes unsatisfactory, for instance in case of function drifting. Before describing in greater detail the components of Tiresias, we remark that our approach is not meant to address issues related to web service availability [58]; indeed, it is focused on offering predictions on the query cost and not on the probability of queries to be answered or not. This means that all failed queries are ignored and discarded from the training process.

The components of Tiresias and their interactions are sketched in Figure 3.20 and summarized below:

- The *query manager* receives both passive and active queries and redirects them to the service. Once the service responds, the cost of the query is measured to send an

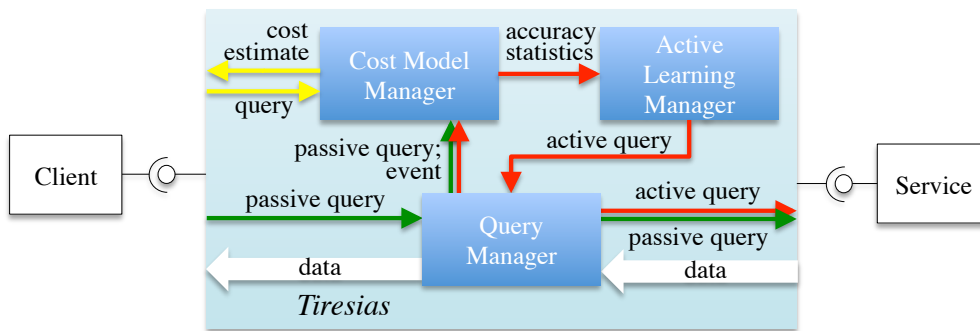


Figure 3.20: Architecture of Tiresias

event to the cost model manager.

- The *cost model manager* estimates the cost of queries, updates the cost model after receiving each event, and provides statistics on the accuracy over all the query space to the active learning manager.
- The *active learning manager* generates active queries to build and refine the cost model.

Temporally, the active querying cycle can be broken into two sequential phases:

1. **Initialization.** This start-up phase aims at gathering a sufficient number of events to obtain a first, rough approximation of the cost function. The user specifies the input parameters, namely, an *initial budget* that denotes how many active queries are to be run during initialization and a desired accuracy for the model.
2. **Refinement.** Once the initial budget has been spent, the active learning manager enters this operational phase, whose aim is to satisfy the accuracy requirement by issuing ad hoc active queries in the areas of the querying space where, according to the statistics, the cost model performs poorly. Once the accuracy requirement has been met, the active cycle pauses, to resume once again whenever the accuracy drops below a given threshold. To regulate the activity of the active learning manager it is possible to define a *refinement budget* that, differently from the initial one, has a refreshment time (e.g., it resets every day).

The passive querying cycle is interleaved with the active one, i.e., the user can issue her own queries at any time during initialization and optimization.

3.3.3 Formal Framework

In this section we define the formal framework on which Tiresias is based, starting from the concepts of service and query.

Definition 10 (Service, Query, and Query Space) A service is represented as a tuple $S = \langle s_1, \dots, s_n \rangle$ of parameters s_i , each defined on a domain $Dom(s_i)$ of values.³ The domain of a parameter can be either categorical or numerical; in the former case it is defined as a discrete and finite set of values, while in the latter it is defined as a continuous and bounded range of values. A query on S is a tuple $Q = \langle q_1, \dots, q_n \rangle$ where $q_i \in Dom(s_i)$ for $i = 1, \dots, n$. The query space of S is the set of all possible queries on S : $\mathcal{Q} = \times_{s_i \in S} Dom(s_i)$.

Notice that these definitions of service and query are more flexible than the ones given in Section 3.2.5.1 (see Definition 6 and 5). The difference lies in the dependence of the former ones on the concept of multidimensional schema. Indeed, this is a design choice to keep the applicability of the Tiresias approach as wide as possible, as general web services may or may not model their data in a multidimensional fashion.

Definition 11 (Stream and Event) A stream D over service S is a temporally ordered sequence of events, each event being defined as a tuple $d = \langle t, Q, y \rangle$ where t is a progressive integer that uniquely identifies d within D , $Q \in \mathcal{Q}$ is a query, and y is the measured cost for executing Q over S .

Each event d corresponds to either a passive (i.e., user-formulated) or an active (i.e., system-generated) query formulated on S . The components of d will be singled out using the dot notation (e.g., $d.Q$).

Example 14 Among the 27 parameters made available for querying in Grid, we mention `latMin`, `latMax` (lower and upper latitude bound in degrees, with domain $[-90, 90]$), and `processingLevel` (which controls whether the requested data should be gridded to the specified space and time resolution, domain $\{L2, L3\}$):

$$S = \langle \text{dataset}, \text{latMin}, \text{latMax}, \text{processingLevel}, \dots \rangle$$

A query on S is

$$Q = \langle \text{ACOSv3.4r01}, 25, 80, L2, \dots \rangle$$

An event for this query is $\langle 1, Q, 27 \rangle$, meaning that Q took 27 seconds to execute.

The cost model for S is a function $\hat{f} : \mathcal{Q} \rightarrow \mathbb{R}$ that associates each query Q with its estimated cost $\hat{f}(Q)$ based on the events in D . In our approach the cost model is expressed by a regression tree that partitions the query space and stores a multiple linear regression (MLR) model in each leaf. We recall that an MLR model is formally defined as

$$\hat{y} = \mathbf{x}\boldsymbol{\lambda} \tag{3.1}$$

³Our approach relies on the assumption that the domain of each service parameter is known; in practice, this kind of information can be usually obtained by either querying the service itself or examining the related documentation.

where \mathbf{x} is a row m -vector representing the independent values, \hat{y} is the corresponding dependent value, and $\boldsymbol{\lambda}$ is a column m -vector representing the regression coefficients⁴. In our context, \mathbf{x} stores the m ($m \leq n$) numerical values of query Q , corresponding to the numerical parameters⁵ of service S ; \hat{y} corresponds to the estimated cost for Q ; and stream $D = (d_1, \dots, d_p)$, with $d_i = \langle t_i, Q_i, y_i \rangle$, is represented by a $(p \times m)$ matrix

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \dots \\ \mathbf{x}_p \end{pmatrix}$$

(where each \mathbf{x}_i is the row m -vector of the numerical values of Q_i) and by the column p -vector \mathbf{y} of the measured costs for the queries in D .

When training the cost model, to estimate its accuracy in a particular region of the query space defined by a leaf l we use the RMSE measure defined as:

$$RMSE(l) = \sqrt{\frac{\sum_{d \in M_l} (d.y - \hat{f}(d.Q))^2}{|M_l|}} \quad (3.2)$$

Note that, while in principle we could give an error estimation of *any* region, even one that does not correspond to a leaf of the tree, in practice this would require a finer partition of the query space, thus increasing the computational effort for maintaining the cost model.

Definition 12 (Cost Model) *A cost model for S , \hat{f} , is expressed by a binary tree $T = (V, E)$ where (i) each internal node $v \in V$ is associated with a parameter s_i of S and a value $q_i \in \text{Dom}(s_i)$, (ii) each edge $e \in E$ is labeled with a Boolean predicate on s_i involving q_i , and (iii) each leaf $l \in V$ is associated with an (ordered) set of events $M_l \subseteq D$ and with a regression model $\boldsymbol{\lambda}_l$. Given internal node v associated to s_i and q_i , if s_i is categorical the two edges exiting v are labeled with predicates $s_i = q_i$ and $s_i \neq q_i$, while if s_i is numerical they are labeled with $s_i \leq q_i$ and $s_i > q_i$.*

For each leaf l of T , we denote with \mathcal{Q}_l the subset of queries in the query space \mathcal{Q} that satisfy all the predicates associated with the internal nodes in the path of T from the root to l . As a consequence of the disjointness and completeness of the couples of predicates corresponding to each internal node, the leaves of T induce a disjoint and complete partition of \mathcal{Q} . So, given query Q , its estimated cost $\hat{f}(Q)$ is computed by finding the (unique) leaf l such that $Q \in \mathcal{Q}_l$ and by applying the regression model $\boldsymbol{\lambda}_l$ to Q .

Example 15 *An example of regression tree expressing a cost model for Grid is shown in Figure 3.21. The tree has four splits, two on categorical parameters and two on numerical*

⁴Equation 3.1 does not explicitly model the intercept value, which in practice can be obtained by adding a dummy parameter with domain $[1, 1]$

⁵To avoid introducing arbitrary mappings from categorical to numerical domains, the regression model does not involve categorical parameters, which are used to partition the query space. Thus, all the parameters are exploited while keeping the regression tree easily interpretable.

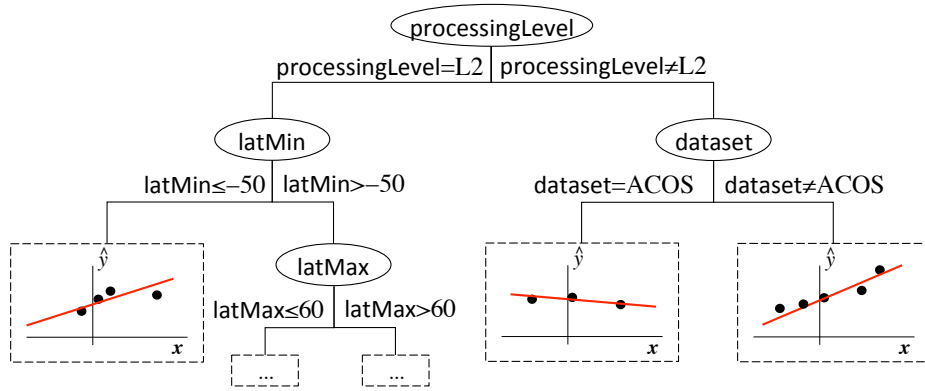


Figure 3.21: A regression tree for *Grid*

parameters. Each leaf is associated with a set of events (black dots) that relate points in the query space with their measured cost and with a local linear regression model (represented as a red line). The cost for query Q in Example 14 is estimated by applying the regression model in the third leaf from the left.

3.3.4 Cost Model Management

This section explains how a cost model is trained in our approach. First, in Section 3.3.4.1 we introduce the SAIRT algorithm [4] for building regression trees, then in Section 3.3.4.2 we describe in detail how SAIRT can be extended by incorporating MLR models to improve accuracy while still retaining stream processing capabilities.

3.3.4.1 The SAIRT Algorithm

SAIRT [4] is an algorithm for the incremental building of binary regression trees tailored for data streams and capable of quickly adapting whenever function drifts are detected. A regression tree T is built starting with a single node that covers the whole space of events available; then this space is partitioned into smaller regions aimed at minimizing the RMSE of the events falling in each region. Each region corresponds to a leaf in T ; the regression model associated with each leaf is the median of its events. Partitioning the space of events is done iteratively by splitting the leaves of T . A leaf l is split into two new leaves when the *instantaneous error rate* (IER) of l , $ier(l)$, is higher than a threshold. The split is done by selecting the parameter-value pair that maximizes the reduction of the RMSE of the events between the original set and the resulting subsets; for numerical parameters, monodimensional k-means clustering is adopted to select the best split value, while each distinct value of the categorical parameters is used to create a binary condition (e.g., `processingLevel = L2`). To prevent a leaf from being split when it contains few events, so as to avoid overfitting, a heuristic is adopted [4].

Now let \hat{f} be a cost model for service S expressed by regression tree T , and v be a node of T . If v is an internal node, let $Children(v)$ denote the set of its direct children and $M_v = \bigcup_{v' \in Children(v)} M_{v'}$ denote the union of the sets of events associated to all the leaves descending from v . The IER of v depends on the number of events in M_v and, in SAIRT, is recursively defined as follows:

$$ier(v) = \begin{cases} \frac{\sum_{d \in M_v} (d.y - \hat{f}(d.Q))^2}{|M_v| \cdot (y_{max} - y_{min})^2} & \text{if } v \text{ is a leaf} \\ \sum_{v' \in Children(v)} \frac{|M_{v'}|}{|M_v|} \cdot ier(v') & \text{otherwise} \end{cases} \quad (3.3)$$

where y_{max} and y_{min} are the maximum and minimum costs, respectively, of all the events $d \in M_v$.

The IER is used when initializing the cost model starting from a first given set of events to iteratively build T . Then of course, as new events are made available, T must be adapted. This is done again by considering if splitting the leaf l where each new event belongs based on its new ier . Additionally, due to the new event, some internal nodes on the path from the root to l may become no more significant and must be eliminated; specifically, the children v' and v'' of a node v are pruned if v is not *coherent*, i.e., if the distribution of the costs in M_v is very similar to those of $M_{v'}$ and $M_{v''}$.

As already mentioned, a relevant feature of SAIRT is its capability to detect function drifts and adapting to them. Unfortunately, the high variability of ier prevents from using it directly to guide the adaptation strategy in case of function drifting. Thus, a smoothing formula is applied to derive the *performance* of node v at time t :

$$per_t(v) = \begin{cases} \frac{7}{8} \cdot per_{t-1}(v) + \frac{1}{8}(1 - ier(v)) & \text{if } t > 0 \\ 1 - ier(v) & \text{if } t = 0 \end{cases} \quad (3.4)$$

Each time t the performance of a node decreases, SAIRT marks t as an *anomaly time*. After an anomaly has been detected, SAIRT starts to forget the oldest events to adapt the model; the number of forgotten events is dictated by how much the performance has decreased and by how long the anomaly persists.

3.3.4.2 Extending SAIRT with Multiple Linear Regression

In this section we describe how we extend the SAIRT algorithm by using MLRs instead of medians as regression models in the leaves of the regression tree. As we will show in Section 3.3.6, this improves the accuracy of the estimations while preserving good execution times. Furthermore, a more powerful leaf model produces a more compact representation of the regression tree which can help the user better understand her data.

Substituting a median-based model with a MLR-based one is not trivial; on the one hand, it invalids some of the (often implicit) assumptions made by the original SAIRT algorithm,

on the other it paves the way for other possible improvements. The specific extensions we made to SAIRT are listed below:

- use of incremental MLR models instead of median;
- a more MLR-friendly split criterion based on residual analysis for numerical parameters;
- a new coherence test based on the RMSE rather than on a distribution distance;
- an adapted definition of IER that (differently from the original one) can be used with MLR models.

Incremental MLR Model. The regression model $\boldsymbol{\lambda}$ for stream D represented as matrix \mathbf{X} can be obtained by applying an ordinary least squares optimization:

$$\boldsymbol{\lambda} = \mathbf{Z}^{-1} \mathbf{X}^T \mathbf{y} \quad (3.5)$$

where $\mathbf{Z} = \mathbf{X}^T \mathbf{X}$. The computation of the optimal coefficients $\boldsymbol{\lambda}$ as in Equation 3.5 is quite expensive since it involves a matrix inversion operation, thus repeating the whole process for each new event would lead to execution times that are not compatible with our stream processing requirements. To address this issue we adopt an incremental update technique that works by initializing the regression model with a small amount of events and then updating it with every new event. Specifically, given stream D we can use its first events \mathbf{X} to compute \mathbf{Z}^{-1} and initialize a regression model $\boldsymbol{\lambda}$ using Equation 3.5. Now, given a new event d with numerical values \mathbf{x} and measured cost y , for the new matrix $\bar{\mathbf{Z}}$ the following identity holds:

$$\bar{\mathbf{Z}} = \bar{\mathbf{X}}^T \bar{\mathbf{X}} = \begin{pmatrix} \mathbf{X} \\ \mathbf{x} \end{pmatrix}^T \begin{pmatrix} \mathbf{X} \\ \mathbf{x} \end{pmatrix} = \mathbf{X}^T \mathbf{X} + \mathbf{x}^T \mathbf{x}$$

so that we can apply the *Sherman-Morrison formula* [59] to update the regression model:

$$\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda} + \frac{\mathbf{Z}^{-1} \mathbf{x}^T (y - \mathbf{x} \boldsymbol{\lambda})}{1 + \mathbf{x} \mathbf{Z}^{-1} \mathbf{x}^T} \quad (3.6)$$

$$\bar{\mathbf{Z}}^{-1} = \mathbf{Z}^{-1} - \frac{\mathbf{Z}^{-1} \mathbf{x}^T \mathbf{x} \mathbf{Z}^{-1}}{1 + \mathbf{x} \mathbf{Z}^{-1} \mathbf{x}^T} \quad (3.7)$$

This incremental update process can be repeated each time a new event arrives and only requires the additional cost of storing the $(m \times m)$ matrix \mathbf{Z}^{-1} . Note that the regression model does not only need to be updated when a new event arrives, but also when an old event is discarded (e.g., during a function drift). Specifically, if an event d is discarded,

the update can be computed as

$$\bar{\lambda} = \lambda - \frac{\mathbf{Z}^{-1}\mathbf{x}^T(y - \mathbf{x}\lambda)}{1 - \mathbf{x}\mathbf{Z}^{-1}\mathbf{x}^T} \quad (3.8)$$

$$\bar{\mathbf{Z}}^{-1} = \mathbf{Z}^{-1} + \frac{\mathbf{Z}^{-1}\mathbf{x}^T\mathbf{x}\mathbf{Z}^{-1}}{1 - \mathbf{x}\mathbf{Z}^{-1}\mathbf{x}^T} \quad (3.9)$$

Split Criterion. The original SAIRT algorithm uses monodimensional k-means clustering to select the split value for numerical parameters as follows. Given leaf l and numerical parameter s_i for which a split value must be computed, the k-means algorithm is applied to all values $\{d.x[i] : d \in M_l\}$ with $k = 2$. The result of this procedure is composed by two different clusters whose centroids are averaged to obtain the final split value.

In Tiresias we opt for a more sophisticated technique specifically tailored for MLR models; this technique has already been used and tested in the SUPPORT algorithm [60] and works as follows. Given cost model \hat{f} and leaf l of the corresponding regression tree, the candidate split value c_i for numerical parameter s_i is chosen as

$$c_i = \frac{r_i^+ + r_i^-}{2} \quad (3.10)$$

where

$$\begin{aligned} r_i^+ &= \text{mean} \{d.x[i] : d \in M_l \wedge \hat{f}(d.Q) < d.y\} \\ r_i^- &= \text{mean} \{d.x[i] : d \in M_l \wedge \hat{f}(d.Q) > d.y\} \end{aligned}$$

Intuitively, this approach exploits the positive and negative residuals of the regression model to identify potential non-linear areas in the underlying function. Consider for instance Figure 3.22, which shows a set of events each characterized by a value of a numerical parameter (on the x-axis) and by the measured cost of the corresponding query (on the y-axis). In Figure 3.22.a all these events belong to a single leaf l of the regression tree, so they are (badly) fitted with a single linear regression model. If k-means clustering were used, the split value for l would be fixed at 50, producing a fairly accurate fit as shown in Figure 3.22.c. Conversely, by adopting our approach, positive and negative residuals are considered. In Figure 3.22 the events are represented with a plus mark when $\hat{f}(d.Q) < d.y$ and with a minus mark when $\hat{f}(d.Q) > d.y$; the former are used to compute r_i^+ , while the latter are used to compute r_i^- . The split value is fixed at about 65, producing the most accurate fit as in Figure 3.22.b.

Coherence Test. The coherence test made in SAIRT after the insertion of each new event consists in checking whether or not the distributions of internal node v and its children v' and v'' are different enough; precisely, if the *Kolmogorov-Smirnov distance* between the distributions is higher than a given threshold the split of v is coherent and is preserved, otherwise it is eliminated. This approach is sound when the median is used as

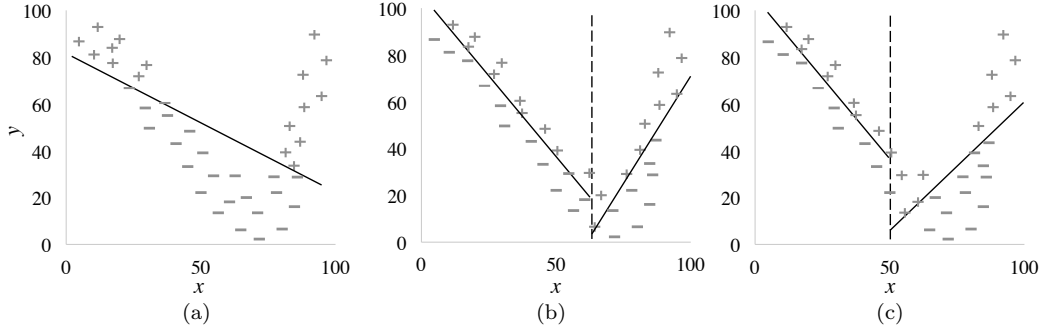


Figure 3.22: The same set of events is approximated using one (a) or two (b) linear regression models (represented as continuous black lines)

a regression model since, given two sets of values, if their medians are different then their distributions are necessarily different; then, if the coherence test is not passed, the two medians must be similar.

The same is not true when using MLR models. Referring again to Figure 3.22, the data points should clearly be fitted with two different linear regression models as in Figure 3.22.b. However, the resulting set of events would have the same cost distributions, thus the split would be considered not coherent. To address this issue we consider the RMSE of the regression models instead of the distributions of the dependent values. Specifically, given a split of v into two children v' and v'' , we check the difference between the RMSE induced by the regression model built using the events in M_v and the RMSE induced by the models built with $M_{v'}$ and $M_{v''}$. If the difference between the RMSEs is below a given tolerance, then the split is not coherent. Of course this requires to keep track not only of the regression models corresponding to the leaves, but also of those corresponding to the internal nodes.

Instantaneous Error Rate. Before explaining why Equation 3.3 cannot be directly applied with MLR, we first briefly analyze how it works. In the case of a leaf node we have at the numerator the *residual sum of squares* (RSS), while at the denominator we have the number of events and the squared difference between the maximum and minimum dependent values; all these terms are computed using the events currently stored in the leaf. The two factors at the denominator act as a weighting mechanism to keep the RSS comparable among different leaves.

The problem in applying Equation 3.3 with a linear model is that it favors hyperplanes with extreme slopes; indeed, the more the slope coefficients are far from 0, the higher difference between the maximum and minimum values tends to be. To cope with this issue, we still apply Equation 3.3 but with

$$y_{max} = \max\{d.y - \hat{f}(d.Q) : d \in M_v\}$$

$$y_{min} = \min\{d.y - \hat{f}(d.Q) : d \in M_v\}$$

Remarkably, this formulation can be considered a generalization of the previous one, since the two equations are the same when considering \hat{f} as a simple constant model (e.g., median).

Complexity. To assess the impact of the extensions that we made to SAIRT, we close this section with a few remarks on computational complexity. Whenever a regression tree is trained with a new incoming event we can distinguish two main scenarios: either (i) all the nodes affected by the new event are coherent and the IER is such that a new split is not required, or (ii) one or more nodes are not coherent or the IER is too high. The key difference between the two scenarios is that the first one does not bring any structural change to the tree, while the second might. Indeed, a pruning operation and a new split are required if, respectively, at least one node is not coherent and the IER is higher than the chosen threshold. With scenario (i) we need to account only for the cost of updating the MLR models and the statistics of the nodes affected by the new event. A single incremental update of an MLR model with m numerical parameters has complexity $\Theta(m^2)$, while the cost for updating the statistics of a node v is $\Theta(|M_v|m)$ or $\Theta(1)$ depending on whether v is a leaf or an inner node. The complexity $\Theta(|M_v|m)$ is due to the computation of the residuals necessary to update the IER. With scenario (ii) the algorithm must find a new best split, which means computing two new regression models for each numerical parameter and for each distinct value of each categorical one. With reference to Equation 3.5, the cost of building an MLR model from scratch is dominated by the computation of matrix Z^{-1} which, using the Gauss-Jordan elimination algorithm, amounts at $\mathcal{O}(m^3)$.

3.3.5 Active Learning

As mentioned in Section 3.3.2, the active querying cycle includes an initialization and a refinement phase. The goal of initialization is to get a general approximation of the costs over all the query space, so the active queries generated are uniformly distributed in the query space. Initialization ends after the initial budget of queries has been spent; of course, the higher the initial budget value, the higher the accuracy of the cost model obtained at this stage.

After the initialization phase, the active cycle kicks in to refine the cost estimates and preserve their accuracy. Specifically, active queries are issued when the RMSE of at least one leaf goes over a user-defined threshold γ . We assume that the noise is uniform over all the query space and does not change with time, so that γ can be considered to be global and constant. Obviously, the intrinsic noise in the events sets a lower bound to the accuracy achievable, thus γ must be higher than such lower bound.

The RMSE can be over the threshold for two different reasons. The first one is that the initial budget was too small to enable a sufficient reduction of the RMSE, so some more events are needed to take the RMSE below γ . The second reason is that there has been a

Algorithm 3 Initialization

Require: A service S , an initialization budget β'

Ensure: A cost model \hat{f} for S

```

1:  $\mathcal{Q} = \times_{s_i \in S} Dom(s_i)$ 
2: Initialize cost model  $\hat{f}$  ▷ An empty regression tree is created
3:  $t \leftarrow 1$ 
4: while  $\beta' > 0$  do
5:   Generate random query  $Q \in \mathcal{Q}$  ▷ A uniform distribution over  $\mathcal{Q}$  is followed
6:    $y \leftarrow Cost(Q)$  ▷ Query  $Q$  is executed
7:    $d \leftarrow \langle t, Q, y \rangle$  ▷ A new event  $d...$ 
8:    $\hat{f} \leftarrow Update(\hat{f}, d)$  ▷ ...is added to the regression tree
9:    $\beta' \leftarrow \beta' - 1$ 
10:   $t \leftarrow t + 1$ 

```

function drift in the service costs and some more events are necessary to adapt the cost model accordingly.

To cope with scenarios in which a few regions of the query space with very accurate estimates mask other regions with poor accuracy, our active learning strategy operates locally, i.e., at the leaf level. First of all, only the leaves whose RMSE is above γ are selected for active query generation. Then, following an *uncertainty sampling* strategy [55], the probability of having one of these leaves queried depends on its RMSE. Specifically, the probability of leaf l to be selected for active querying is proportional to its *informativeness*, defined as

$$\phi(l) = RMSE(l) \cdot coverage(l) \quad (3.11)$$

where

$$coverage(l) = \frac{area(l)}{|M(l)|}$$

and $area(l)$ measures the extension of the query space covered by \mathcal{Q}_l . The extension of leaf l is defined as $area(l) = \prod_{i=1}^n length_l(s_i)$, where

$$Length_l(s_i) = \begin{cases} \frac{|Dom_l(s_i)|}{|Dom(s_i)|} & \text{if } s_i \text{ is categorical} \\ \frac{max(Dom_l(s_i)) - min(Dom_l(s_i))}{max(Dom(s_i)) - min(Dom(s_i))} & \text{if } s_i \text{ is numerical} \end{cases}$$

and $Dom_l(s_i)$ is the projection of \mathcal{Q}_l on the i -th parameter s_i . The coverage factor aims at improving the robustness of the optimization strategy when the noise is not uniform across the query space; intuitively, it favors regions in the query space for which few events are currently stored.

The pseudocode of Algorithms 3 and 4 summarizes the complete active cycle, distinguishing its two phases. During initialization (Algorithm 3) the query space of S is defined and active queries are uniformly generated on it as long as the initialization budget β' allows. Algorithm 4 is invoked immediately after initialization, and then after each new passive query has been executed and the cost model has been updated accordingly. Here, as already stated, querying generation is focused on the regions of the query space where the RMSE is above γ ; at each iteration, while the target leaf \bar{l} is selected based on its

Algorithm 4 Refinement

Require: A service S , a time t , a cost model \hat{f} for S , a refinement budget β'' , an RMSE threshold γ
Ensure: A (refined) cost model \hat{f} for S

```

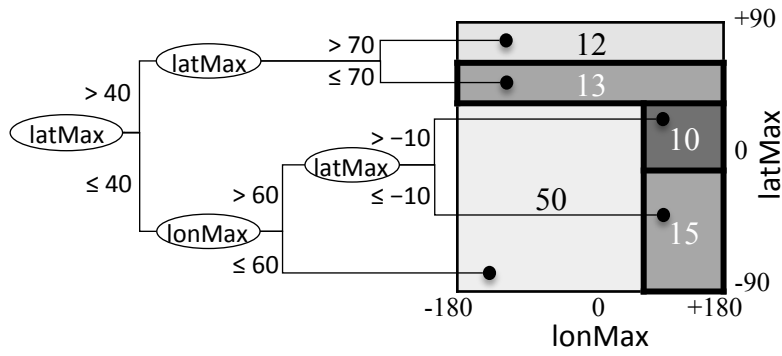
1:  $L \leftarrow \{l : l \text{ is a leaf of } \hat{f} \wedge \text{RMSE}(l) > \gamma\}$ 
2: while  $L \neq \emptyset \wedge \beta'' > 0$  do
3:   Randomly choose  $\bar{l} \in L$ 
4:   Generate random query  $Q \in \mathcal{Q}_{\bar{l}}$ 
5:    $y \leftarrow \text{Cost}(Q)$ 
6:    $d \leftarrow \langle t, Q, y \rangle$ 
7:    $\hat{f} \leftarrow \text{Update}(\hat{f}, d)$ 
8:    $\beta'' \leftarrow \beta'' - 1$ 
9:    $t \leftarrow t + 1$ 
10:   $L \leftarrow \{l : l \text{ is a leaf of } \hat{f} \wedge \text{RMSE}(l) > \gamma\}$ 

```

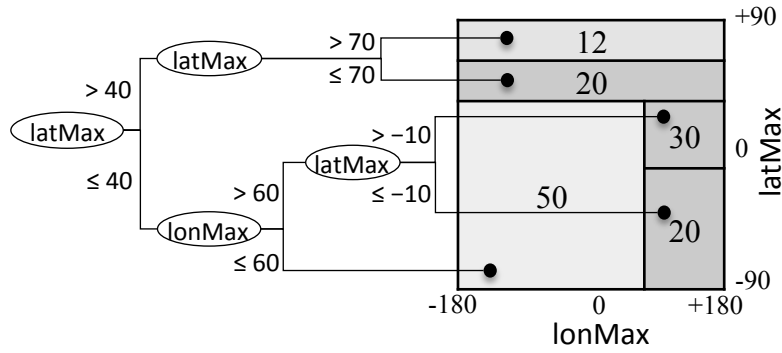
informativeness $\phi(\bar{l})$, the specific query generated for \bar{l} is randomly chosen following a uniform distribution over $\mathcal{Q}_{\bar{l}}$. Like for initialization, there is a specific budget β'' that limits the amount of queries issued.

Example 16 *To show a comprehensive example of the role that the active learning strategy plays in Tiresias, we focus on a simplified version of the Grid service, $S = \langle \text{latMax}, \text{lonMax} \rangle$, where both parameters `latMax` and `lonMax` are numerical. Figure 3.3.5 shows, at four consecutive times, the current regression tree (on the left) and the partition it induces on the query space (on the right). Each leaf corresponds to a region of the query space and is described by its RMSE (the darker the shade of gray, the higher the RMSE) and by the number of the events currently stored. The initialization and refinement budget are set to $\beta' = 90$ and $\beta'' = 70$, respectively (note that the refinement budget is assumed to be periodically replenished).*

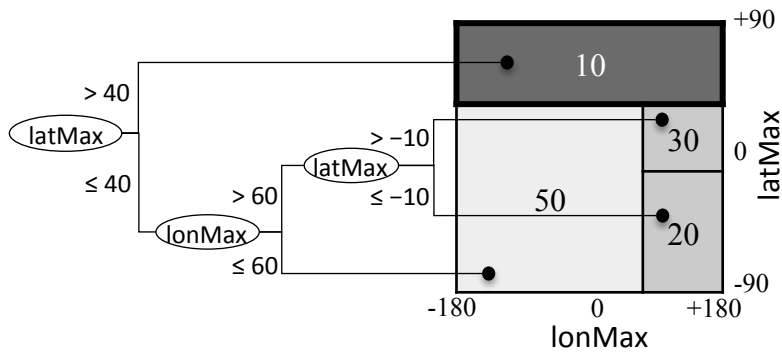
- (a) *In the initialization phase (see Algorithm 3) the active learning manager generates random queries uniformly over all the query space, to obtain $\beta' = 90$ events that are used to build the cost model. Eventually, let the resulting tree be the one shown in Figure 3.23a, where the RMSE is above threshold for three leaves (corresponding to the three tick-bordered regions in Figure 3.23a). Note that, since the query distribution is uniform at this stage, the leaves that cover larger regions have a higher number of events.*
- (b) *Since $L \neq \emptyset$ and $\beta'' > 0$ (line 2 of Algorithm 4), the refinement phase is entered and some active queries targeting the leaves in L are generated to improve the cost model. We assume that 42 such queries are needed to take the RMSE below the threshold for all leaves. Let the tree at the end of the refinement phase be the one in Figure 3.23b; the generation of queries has followed a ϕ -based distribution (see Equation 3.11), thus leaves with higher values of ϕ have been prioritized.*
- (c) *A stream of passive queries is then expressed by the user. Let a function drift occur at this time: the performance of one or more leaves decrease, so an anomaly is detected and the oldest events in the affected leaves are forgotten to try to adapt the cost model. We assume that, due to the adaptation mechanism of the regression tree, the two top*



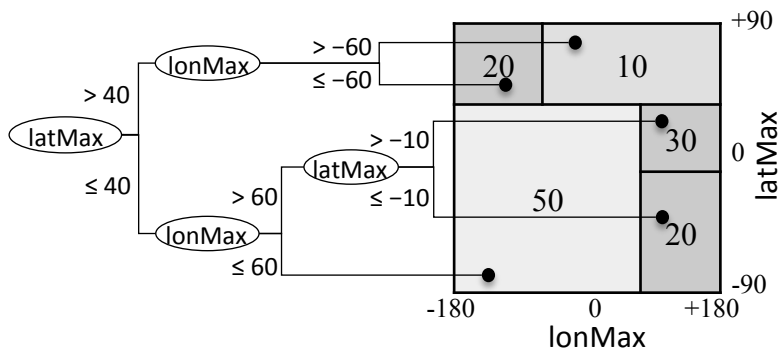
(a) Cost model after the initialization phase



(b) Cost model after the refinement phase



(c) Cost model after a function drift



(d) Cost model after adapting to the drift

Figure 3.23: An example of cost model evolution in Tiresias

Table 3.3: Services used in the experiments

Service	Drift	Parameters		Events		Mean Cost
		Categorical	Numerical	Training	Test	
<i>Synth</i>	No	1	1	25k	5k	1.4 secs
<i>DB</i>	No	5	2	10k	5k	12.3 secs
<i>Synth+</i>	Yes	1	1	25k	10k	1.4 / 1.3 secs
<i>Grid</i>	Yes	2	8	30k	*	118.9 secs

leaves of Figure 3.23b are merged since the split is no more coherent. The resulting tree is shown in Figure 3.23c: it has four leaves only, and for one of them (the dark one resulting from the merge) the RMSE is above the threshold.

- (d) Since again $L \neq \emptyset$ and $\beta'' > 0$, the active learning manager starts generating additional active queries as described in Algorithm 4 to refine the cost model after the drift. All new events fall into the top leaf of Figure 3.23c and may induce a new split on parameter `lonMax`, to produce the result in Figure 3.23d where the RMSE of all leaves is below threshold.

3.3.6 Experimental Results

This section collects the main results of the experiments we carried out to evaluate our approach. All the algorithms are implemented in Scala (version 2.11.7) and run on a desktop PC quad-core (3.6GHz, 32 GB RAM, Windows 8-64 bit). Table 3.3 shows a summary of the main features of the four services used for the experiments, namely, *Synth*, *DB*, *Synth+*, and *Grid*. The first three services run in controlled environments, so that we know exactly if (and when) a drift happens. *Synth* and *DB* do not suffer from drift, i.e., the underlying cost function is constant, while in *Synth+* the cost function changes in time. Finally, *Grid* is a real-life web service running in an environment of which we have very limited information, that is, we only know the definition of its public interface. In the following we describe in greater detail each of these services and the events gathered by querying them.

- The events gathered from the *Synth* service were procedurally generated according to a cost function with a perturbation given by Gaussian noise. The service contains two parameters: `cat`, with a binary categorical domain, and `num`, with a numerical domain. For `cat = Simple` the cost function is linear (see light gray points in Figure 3.24); for `cat = Complex` we adopted a piece-wise function made of two linear and one exponential parts (see dark gray points in Figure 3.24). Splitting the query space into two regions with different degrees of complexity enables a better assessment of the behavior of the active learning strategy.
- The *DB* service operates on top of a DBMS and each event was generated by running

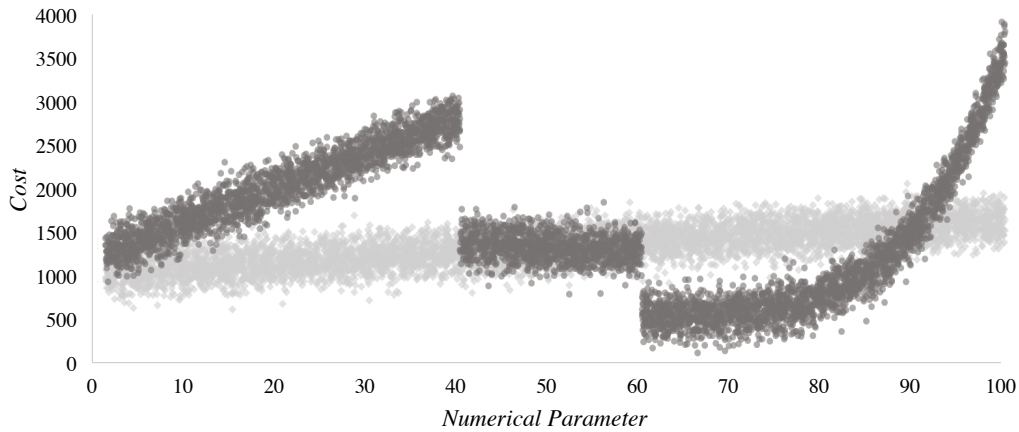


Figure 3.24: The events of the *Synth* service; light and dark gray points represent the events for the two values of the categorical parameter

a SQL query and using its execution time as the cost value. To diversify the query space we populated two distinct databases; both contain the same data from the TPC-D benchmark [61], but only one of them was indexed. The SQL query that we wrote to simulate the service makes use of two numerical and two categorical parameters. All four parameters support filtering (i.e., conditions in the WHERE clause), while only categorical parameters support aggregation (i.e., they can appear in the GROUP BY clause). In practice, since in our approach services are supposed to expose an interface with a fixed number of parameters, a call to *DB* is made by mapping each condition in the SQL query to a parameter in the service query; two additional parameters are used to enable aggregation for the two categorical parameters, and one more parameter determines if the SQL query will be launched on the database with or without indexes.

- The *Synth+* service shares the same interface as *Synth* and was designed to simulate a drifting environment by changing its cost function after a certain amount of generated events. Specifically, every five thousand generated events the service switches between the cost function used in *Synth* and a simple linear function holding for both values of the parameter *cat*. This type of drift is meant to be as disruptive as possible, since we observed that the the first (root) split of the regression trees built for the *Synth* service always involve the categorical parameter.
- As mentioned in Example 13, the *Grid* web service gives access to several CO₂-related measurements gathered from satellites and ground stations. In addition to filtering, *Grid* can aggregate the available measurements along the time and spatial dimensions. A call to the service consists in an HTTP request where several parameters must be specified; the result is a textual or graphical representation of the requested measurements. For our experiments we considered a reduced query space including a subset of ten parameters. We remark that, differently from the other services, we

Table 3.4: Comparison between SAIRT and SAIRT-MLR

Service	SAIRT			SAIRT-MLR				
	RMSE	Rel. Error	# Leaves	# Events	RMSE	Rel. Error	# Leaves	# Events
<i>Synth</i>	166	0.11	102	13	152	0.11	23	55
<i>DB</i>	7116	0.72	89	16	5530	0.36	87	19
<i>Synth+</i>	242	0.13	16	658	196	0.11	16	595
<i>Grid</i>	37870	0.49	79	74	23585	0.29	45	78

have no knowledge of the internals of *Grid*.

To evaluate the efficiency of Tiresias, we measure the average number of events per second that it can ingest for training purposes. To evaluate its effectiveness, we measure the accuracy of the cost models it builds using again the RMSE:

$$RMSE(\hat{f}) = \sqrt{\frac{\sum_{d \in D} (d.y - \hat{f}(d.Q))^2}{|D|}}$$

where D is a test set of events gathered by uniformly querying the services.⁶ The number of test events for each service is shown in Table 3.3. Differently from *Synth* and *DB*, for *Synth+* we generated two different test sets of 5000 events each, one for each underlying function; at each time instant we compute the RMSE using only the appropriate test set, i.e., the one belonging to the function currently used by the service. For *Grid* it is not possible to gather a reliable test set since its underlying dynamics are unknown and out of our control, so we had to use an alternative approach to measure the cost model accuracy. Specifically, at each new event d corresponding to a passive query we measure the RMSE (which is a simple deviation in this case) between the actual cost and the one predicted by the model *before* d is used to update it.

The remainder of this section is organized into two parts: in Section 3.3.6.1, the results of our algorithm for building regression trees (called SAIRT-MLR from here onward) are evaluated using the original SAIRT as a baseline; in Section 3.3.6.2, the focus is on assessing the behavior of the active learning strategy.

3.3.6.1 Comparison with SAIRT

With reference to the framework presented in Section 3.3.2, in the first part of the evaluation the active querying cycle is switched off. The passive cycle is simulated through queries uniformly generated over all the query space; each passive query corresponds to an event used to train the cost model.

SAIRT and SAIRT-MLR are compared on the querying timeline in terms of their accuracy, the size of the trees that they produce, and their efficiency. In particular, Table 3.4 shows

⁶The test sets are used only to compute the RMSE values and not to train the cost model.

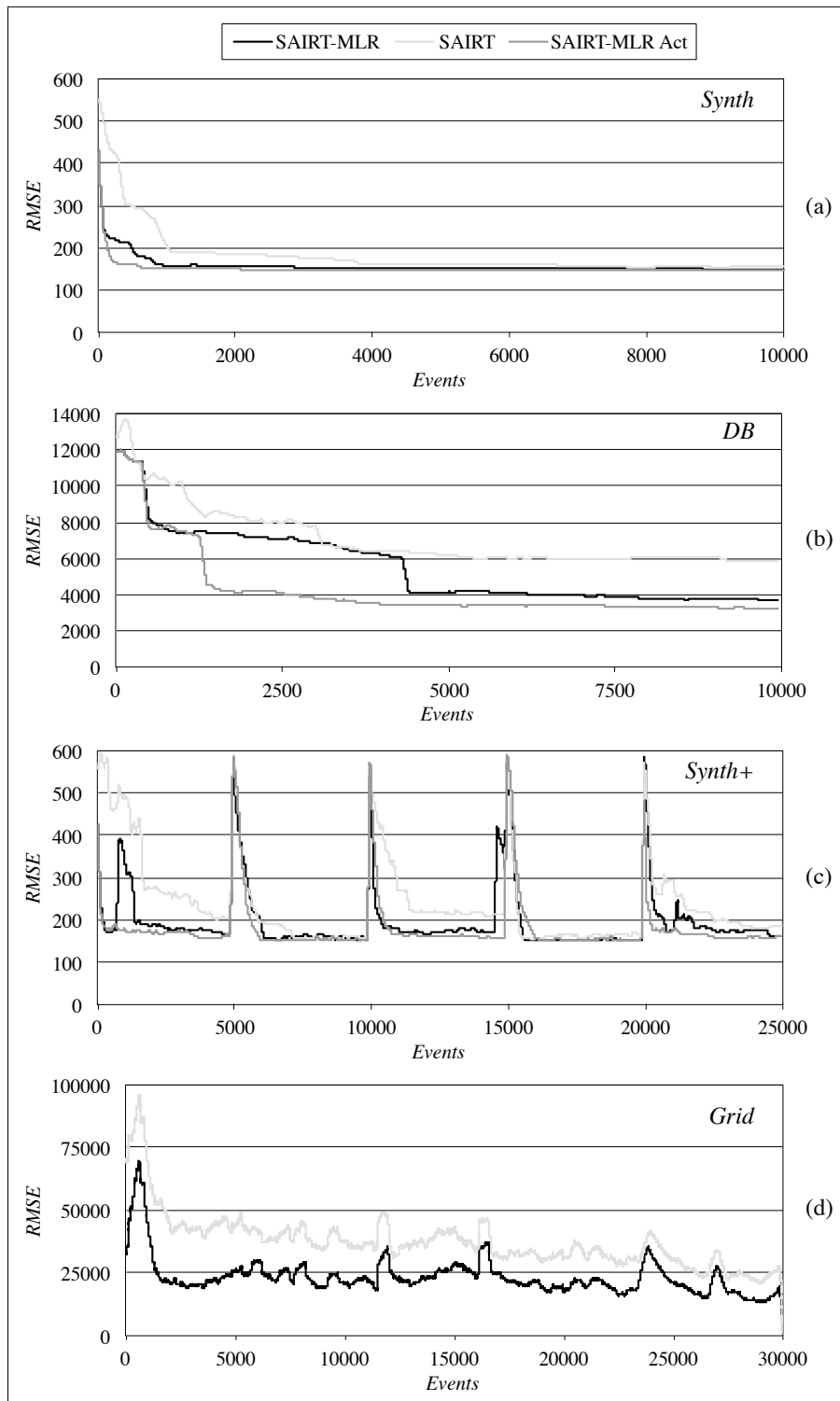


Figure 3.25: RMSE (the lower the RMSE, the higher the accuracy) for each algorithm and service

for each service and algorithm the average prediction error (column **RMSE**), average size of the generated regression trees (**# Leaves**), and average number of processed events per second (**# Events**) during training. Our evaluations of effectiveness are mainly based on

the RMSE since our approach aims to minimize it; however, to present a more intuitive measure of accuracy, in Table 3.4 we also include the average relative prediction error of the cost models (column **Rel. Error**).

More specifically, Figure 3.25 shows how the RMSE changes during training. Clearly, the SAIRT-MLR algorithm achieves more accurate predictions than SAIRT, both in stable (*Synth* and *DB*) and dynamic (*Synth+* and *Grid*) contexts. For *Synth* (Figure 3.25.a), the error rates of SAIRT-MLR are significantly smaller than those of SAIRT during the first 5000 events, while for later events the error rates are comparable: at the 10000-th event there is only a 4% difference between them (the remaining 15000 events are omitted since there are no significant changes). The *DB* service (Figure 3.25.b) proved to be more difficult to model. Here the differences in prediction errors are significant over all the 10000 events; at the 10000-th event the RMSEs of SAIRT-MLR and SAIRT have a 57.3% difference. The *Synth+* service (Figure 3.25.c) simulates a variable cost function by changing its behavior every 5000 events; for both algorithms there are error spikes whenever a function drift occurs, then the models quickly adapt to the new behavior leading first to a decrease and then to the stabilization of the prediction errors. Finally, the results for *Grid* (Figure 3.25.d) confirm that, though both algorithms react similarly to the dynamics of the service, SAIRT-MLR consistently obtains more accurate predictions than SAIRT; overall, *Grid* presents a much more unstable behavior than the other services, as reflected by the noticeable fluctuations of the RMSE.

Going back to Table 3.4, it is worth discussing the size of the regression trees generated to model the different services. For *Synth*, the trees obtained after 25000 events by SAIRT-MLR and SAIRT include 26 and 152 leaves, respectively. This sharp difference is not surprising as the former algorithm uses MLRs, while the latter uses simple median values, which require a finer partitioning of the query space to compensate a lower expressiveness. For instance, with reference to Figure 3.24, the events in light gray are modeled with a single leaf by SAIRT-MLR, while SAIRT fragments that region of the query space into 24 leaves. Conversely, for *DB*, the trees generated by SAIRT-MLR and SAIRT after 10000 events include roughly the same number of leaves. The reason for this different behavior lies in the different complexities of the cost functions for the two services: while for *Synth* the cost function is linear in several regions of the query space, this is not the case with *DB*. *Synth+* and *Grid* have variable cost functions, so we measured the average number of leaves of the trees generated after each event. For *Synth+*, the average tree size is the same for SAIRT-MLR and SAIRT. Finally, for *Grid*, the trees generated by SAIRT-MLR and SAIRT are composed by an average of 45 and 79 leaves, respectively.

We close this section with an efficiency evaluation. Table 3.4 shows the average number of events processed (i.e., used to update the model) per second by each algorithm on each service. Surprisingly, SAIRT-MLR is more efficient on all the services but *Synth+*. While it is counterintuitive that a more complex model achieves not only better accuracy but also better performances, this can be explained as follows. As discussed in Section 3.3.4, the

MLR models in SAIRT-MLR are incrementally updated, while the median values in SAIRT must be computed from scratch after each new event; thus, updating an MLR model is actually cheaper than updating a median value. However, whenever a structural change is performed (i.e., a split or a pruning operation), both MLR models and median values must be completely recomputed, thus removing the advantage of incremental processing. Indeed, the service in which SAIRT-MLR most clearly takes the lead is *Synth*, where a stable (and accurate) model that needs few structural changes is produced quite soon. On the other hand, in more complex and variable services, the gap between the two algorithms becomes very small. Interestingly, the performances of both algorithms on the variable services (*Synth+* and *Grid*) are better than those on *Synth* and *DB*: drifting induces more structural changes but the number of events stored by the models is smaller, thus yielding better performances. This is particularly true for SAIRT which, as already mentioned, cannot incrementally update the median values stored in the leaves.

3.3.6.2 Active Learning Analysis

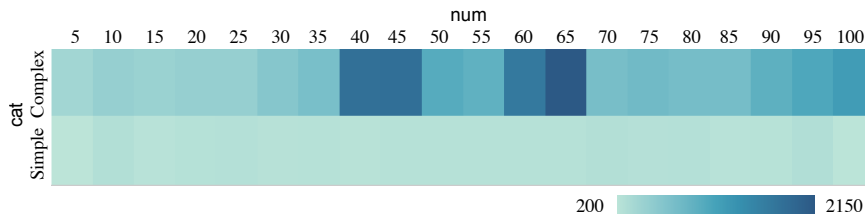


Figure 3.26: Distribution of active queries over the query space of *Synth* (the darker the color, the higher the number of queries)

In this section we assess the effectiveness of our active learning strategy by comparing the accuracy of Tiresias with and without the active cycle. This cannot be done for *Grid* since, as already mentioned, we are unable to gather a test set that can be used on different test runs; even comparing the prediction values with the real costs for each event (as done in Section 3.3.6.1) would be misleading, due to the different distributions of the queries issued. For each remaining service we used an initialization budget of 200 queries, while the RMSE threshold γ and the refinement budget have been set to 0 and ∞ , respectively, so that the active learning manager continuously generates queries. To emphasize the contribution of the active cycle we limit as much as possible the number of passive queries; more precisely, with *Synth* and *DB* no passive queries are issued, while with *Synth+* 200 passive queries are interleaved with the active ones each time there is a function drift.

Referring again to Figure 3.25, switching the active learning strategy on (SAIRT-MLR Act) clearly improves the accuracy over SAIRT-MLR for all three services. For *Synth*, the RMSE decreases faster in SAIRT-MLR Act than in SAIRT-MLR during the first 1000 events, then both algorithms reach stability. The higher complexity of *DB* makes the differences more apparent: at the 10000-th event there is still a 13.3% difference in RMSE.

The results for *Synth+* confirm that the active strategy is more effective than a simple uniform querying strategy; however, as in the case of *Synth*, the low complexity of the cost function leads to a small difference in RMSE (4.8%).

As to efficiency, we remark that the cost for generating active queries is negligible, so the same considerations made in Section 3.3.6.1 hold.

We close this section by analyzing the distribution of the active queries. To this end we focus on service *Synth*, whose query space is simple enough to be visually represented but at the same time features regions with different degrees of complexity. The heat map in Figure 3.26 reproduces the two-dimensional query space of *Synth* as a matrix of 40 cells (the numerical parameter `num` has been discretized). Region `cat = Simple` is covered lightly and uniformly. The active learning strategy mostly focuses on the `cat = Complex` region, issuing here 80% of all the queries; a few “hot” areas are clearly visible where the cost function abruptly changes (`num` \approx 40 and `num` \approx 65) and where it assumes a pronounced exponential shape (`num` \gtrsim 90). These results are quite intuitive and confirm that the informativeness ϕ defined in Equation 3.11 is indeed useful to pinpoint the regions of the query space whose accurate modeling is most difficult.

3.3.7 Wrapping up Tiresias

We have proposed Tiresias, an approach to build a cost model that accurately predicts query costs in web services by building and actively adapting a regression tree. Tiresias extends the SAIRT algorithm for building regression trees by incorporating MLR models; besides, it triggers an active learning approach whenever the prediction accuracy decreases below a threshold. The tests we made show that our extension to SAIRT outperforms the original algorithm in terms of prediction accuracy and efficiency, and also builds more compact regression trees; besides, the introduction of the active querying cycle further improves the prediction accuracy in case of function drifts, with a negligible impact on the overall efficiency of the approach.

Three interesting issues need further investigation and are left as part of our future work. The first one is related to considering additional features (e.g., the extension of the geographical area queried in *Grid*) when training the regression tree to improve the prediction accuracy, which however would significantly impact on our active learning strategy. The second one is to better deal with services where the values of a set of parameters determines the applicability of other parameters (e.g., in *Grid*, four parameters are valid only when `processingLevel=L3`). Finally, *transfer learning* strategies could be adopted to reuse the cost model developed for a service for another service with similar features [62].

Chapter 4

Compact Visualization of Multidimensional Data

In this chapter we focus on the contributions related to the shrink operator, which has been previously proposed by Golfarelli et al. [5].

The only implementation of the shrink operator proposed in literature is based on a greedy heuristic that, in many cases, is far from reaching a desired level of effectiveness. In Section 4.4 we propose a model for optimizing the implementation of the shrink operation which considers two possible problem types. The first type minimizes the loss of precision ensuring that the resulting data do not exceed the maximum size allowed. The second one minimizes the size of the resulting data ensuring that the loss of precision does not exceed a given maximum value. We model both problems as a set partitioning problem with a side constraint. To solve the model we propose a dual ascent procedure based on a Lagrangian pricing approach, a Lagrangian heuristic, and an exact method. All the novel methods are experimentally evaluated and compared with the greedy implementation.

Furthermore, the previous implementation of shrink requires the user to choose the dimension that will be reduced through hierarchical clustering. To improve the efficacy of the operator, in Section 4.5 we propose a multidimensional generalization that can work on all dimensions simultaneously. Multidimensional shrink comes in two flavors: lazy and eager, where the bounds posed by hierarchies are respectively weaker and stricter. Greedy algorithms based on agglomerative clustering are presented for both lazy and eager shrink, and experimentally evaluated in terms of efficiency and effectiveness.

4.1 Motivation and Outline

One of the key factors that rule the effectiveness of analyses is the achievement of a satisfactory (from the users' viewpoint) compromise between the precision and the size of the information being displayed while analyzing multidimensional cubes. The OLAP paradigm gives a significant support in this direction by enabling users to interactively slice, dice, and aggregate cube facts, but this is not always sufficient: more detail gives more information, but at the risk of missing the overall picture, while focusing on general trends may prevent users from observing specific small-scale phenomena [63]. This is also strictly related to the “information flooding” problem, which may happen because the user drilled down a cube up to a very detailed level, where a huge number of facts are to be returned. In this case, it may be very hard for the user to browse and analyze the results, especially if the device used has limited visualization and data-transmission capabilities.

Different approaches can be taken to cope with this issue. For instance, in *query personalization* there is an attempt to tune the size and pertinence of facts returned by considering the users' preferred aggregation levels, measures, and slices [64]. In *approximate query answering*, the focus is on quickly returning an answer at the price of some imprecision in the returned values [65]. In *intensional query answering*, the set of facts returned by a query is summarized with a concise description of the properties shared by those facts [63]. Other papers couple the OLAP paradigm with data mining techniques to create an *OLAM approach* where cubes can be mined “on-the-fly” to extract concise patterns for user's evaluation [66].

		Year		
		2010	2011	2012
City	<i>Miami</i>	47	45	50
	<i>Orlando</i>	44	43	52
	<i>Tampa</i>	39	50	41
	<i>Washington</i>	47	45	51
	<i>Richmond</i>	43	46	49
	<i>Arlington</i>	—	47	52

Figure 4.1: A simple pivot table showing data by City and Year (this picture is a duplicate of Figure 2.4 repeated here to ease the reading)

The *shrink* approach [5] is a form of OLAM based on hierarchical clustering, specifically aimed at balancing precision with size in visualization of multidimensional cubes via pivot tables like the one shown in Figure 4.1. The shrink operator can be applied during an

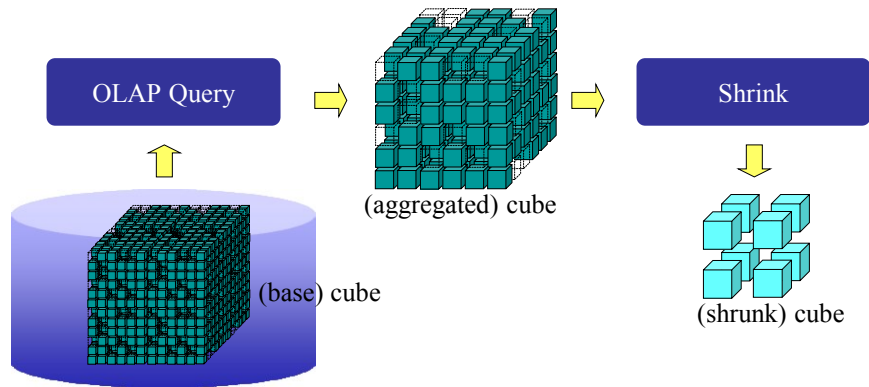


Figure 4.2: Functional overview of the shrink approach

OLAP session to the cube resulting from a query to decrease its size while controlling the approximation introduced, like sketched in Figure 4.2. The idea is to *fuse* similar facts together and replace them with a single representative fact (computed as their average), respecting the bounds posed by dimension hierarchies.

The rest of the chapter is organized as follows. In Section 4.2 we present the related work, while Section 4.3 briefly introduces the main concepts behind the shrink operator. In Section 4.4 and 4.5 we respectively describe our improved algorithms (both heuristic and exact) and a new multidimensional generalization for shrink. Notice that, while sharing many basic concepts, Section 4.4 and 4.5 approach and model the shrink problem from two very different angles. Indeed, Section 4.4 focuses on improving the computational aspects of the problem by employing techniques borrowed from the operational research area, while Section 4.5 takes a more BI centered approach to expand the capabilities of the operator.

4.2 Related Work

In the following we survey the related research work, mostly in the query personalization area. Overall, it appears that *shrink* differs from the previous approaches in mainly three ways: (i) because it requires no manual specification of preference criteria for driving visualization; (ii) because the obtained visualization does not depend on the user history; and (iii) because visualizations are built without violating the constraints posed by hierarchies.

Our work can be situated in the area of pervasive BI, and more specifically in that of OLAP personalization, that enables users to fine-tune query results in terms of relevance, size, and presentation. The authors in [67] give a survey of the main research direction in OLAP personalization, namely OLAP preferences [64], OLAP visualization [68], OLAP recommendation [69], and schema personalization [70]. The first two types of approach are most closely related to *shrink*.

An algebra for defining *OLAP preferences* has been defined by Golfarelli et al. in [64]. Their approach is powerful from an expressiveness point of view, but differently from the *shrink* operator, it requires users to manually specify their preference criteria. Also most other work on OLAP content personalization, such as the one by [71], shares with the one mentioned above the need for some manual intervention to define preferences. In a few cases, such as [72], preferences are automatically derived by analyzing the log of the user queries, which makes the results of current queries dependent on the past queries.

OLAP visualization techniques are another way to deliver personalized results to users. The authors in [73] propose a new presentation model to separate the logical data retrieval part from the presentation part and show how the *table lens* technique can be easily mapped on their model. Based on the intuition that compressing data can efficiently aid data presentation, [74] devise a new OLAP visualization technique where a cube is compressed through a quad-tree based approach, but without actually reducing the size of the resulting table. In [68] a new visual hierarchical structure is introduced whose nodes contain different non-hierarchical presentations, and that can be used to explore data.

At the borderline between OLAP visualization and preferences, [75] propose a technique for obtaining a personalized visualization of query results in presence of constraints on the maximum number of returned cells for each dimension of analysis; user preferences are manually defined in terms of a total order of dimension members.

Overall, the work in OLAP personalization that shares most similarities with *shrink* is the one by [76], where K-means clustering is used to dynamically create semantically-rich aggregates of facts. This work, like *shrink*, can be classified as OLAM because it couples mining and OLAP techniques; however, while with *shrink* the resulting clustering follows the constraints imposed by the hierarchy, here this is not the case.

Outside of the query personalization area, another way to concisely answer a query is that of giving it an *intensional answer*, i.e., one that summarizes the set of tuples to be returned with a description of the properties these tuples share [77]. Intensional query answering has been applied in several areas but, to the best of our knowledge, the only work related to OLAP is [63], that proposes a framework for computing an intensional answer to an OLAP query by leveraging the previous queries in the current session. Like for [72], even here query results strongly depend on the user history. Another approach that aids the user during OLAP analyses by enriching and summarizing query results is CineCubes [78]. This system enriches the results of the query issued by the user with other complementing data, which are then analyzed to identify and extract the most interesting patterns. The resulting highlights are finally visualized and described through text generation.

A related research topic is *approximate query answering*, whose main goal is to increase query efficiency by returning a reduced result while minimizing the approximation introduced. Some approximate query answering approaches were specifically devised for OLAP. For instance, [79] propose a technique that, given a fixed amount of space, returns the cells that maximize the accuracy for all possible group-by queries on a set of columns. While the shrink operator uses approximation to reduce the size of query results, in their work sampling is used to quickly compute measure values but does not change the size of results. [80] use quad-trees to hierarchically partition data to obtain a concise representation of the original data. In [81] cubes are divided in chunks; chunks are then clustered and each cluster is represented by its centroid. Both works do not consider hierarchy constraints while shrink does. The work in [82] has a different focus: tuples are sent to a data warehouse as separate streams that must be integrated, and the approximation of queries when a small quantity of memory is available is studied.

Finally, an approach similar to shrink in the area of temporal databases is *parsimonious temporal aggregation* (PTA), a novel temporal aggregation operator that merges tuples related to the same subject and with consecutive time intervals to compute aggregation summaries that reflect the most significant changes in the data over time [83]. Though shrink shares the same basic principle, it bears more complexity because (a) the constraints deriving from temporal consecutiveness strongly reduce the PTA search space; (b) preserving hierarchy semantics introduces additional complexity.

4.3 Background on the Shrink Operator

Given a multidimensional cube and chosen one of its dimensions, the shrink operator works by merging the dimensional values, together with the corresponding slices of cells. The aim is to obtain a compact representation of the input data that either minimizes the approximation error and satisfies a given size constraint (*size-bound shrink*), or minimizes the size of the result while satisfying a given error constraint (*loss-bound shrink*). The resulting representation must also be compliant with respect to the constraints imposed by the structure of the involved hierarchy. Before describing the greedy implementation of the shrink operator proposed in [5], we need to briefly introduce the concept of *hierarchy compliance* and how we compute the approximation error (for an exhaustive and formal definition, see [5]).

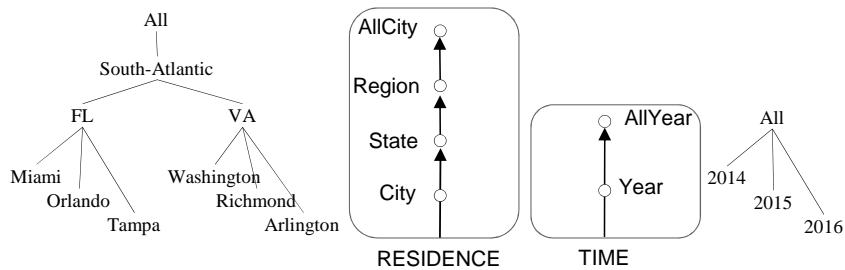


Figure 4.3: Two examples of hierarchies showing both their values and their aggregation structures (this picture is a duplicate of Figure 2.2 repeated here to ease the reading)

Intuitively, given a cluster composed by the values of a dimension on top of which is built hierarchy h , we say that such cluster is *hierarchy compliant* (or *h-compliant*) if and only if all its elements are values of h belonging to a same level and with the same parent. The complete enumeration of the h-compliant clusters associated to the RESIDENCE hierarchy of Figure 4.3 is listed in Table 4.1. An example of a non h-compliant cluster is instead $\{Miami, Washington\}$, because to be able to have *Miami* and *Washington* in the same cluster it would be necessary to also merge together *Orlando*, *Tampa*, *Richmond*, and *Arlington*. When a cluster includes all and only children of one or more elements of the parent level, it can be represented as the set of the corresponding parent values (i.e., $\{Miami, Washington, Orlando, Tampa, Richmond, Arlington\} \simeq \{FL, VA\}$).

		Year		
		2014	2015	2016
City	Miami, Orlando	45.5	44	51
	Tampa	39	50	41
	VA	45	46	50.6

(a)

		Year		
		2014	2015	2016
South-Atlantic		44	46	49.2

(b)

Figure 4.4: Two reductions of the same cube

Table 4.1: Clusters for the example reported in Figures 4.1 and 4.3

Level 0
$\{\textit{South-Atlantic}\} \simeq \{FL, VA\}$
Level 1
$\{\textit{Miami, Orlando, Tampa}\} \simeq \{FL\}$
$\{\textit{Washington, Richmond, Arlington}\} \simeq \{VA\}$
Level 2
$\{\textit{Miami}\}$
$\{\textit{Orlando}\}$
$\{\textit{Tampa}\}$
$\{\textit{Miami, Orlando}\}$
$\{\textit{Orlando, Tampa}\}$
$\{\textit{Miami, Tampa}\}$
$\{\textit{Washington}\}$
$\{\textit{Richmond}\}$
$\{\textit{Arlington}\}$
$\{\textit{Washington, Richmond}\}$
$\{\textit{Richmond, Arlington}\}$
$\{\textit{Washington, Arlington}\}$

Each member at the finest level of detail (i.e., a dimensional value) is associated to a slice of cells, e.g., with reference to Figure 4.1, the slice associated with the value *Miami* of the City level is composed by values 47, 45, and 50. To compactly represent cells of several members that have been merged together, the shrink operator uses their average. The approximation error introduced by representing a set of slices with an average slice is computed as the *Sum Squared Error* (SSE) between the average and the original values. Two different examples of reductions induced through the shrink operator are shown in Figure 4.4. Specifically, the SSE associated to the average slice $\{\textit{Miami, Orlando}\}$ in Figure 4.4.a is $(1.5^2 + 1.5^2) + (1^2 + 1^2) + (1^2 + 1^2) = 8.5$. Remarkably, the error introduced by merging two or more values is never negative.

The greedy implementation of the shrink operator for both size- and error-constrained problems is based on agglomerative hierarchical clustering. Specifically, the algorithm works bottom-up by merging at each iteration the two clusters of members (and their slices) that lead to the minimum increase in SSE. Of course, the two clusters can be merged only if the result is still h-compliant. This iterative process ends when the size constraint is satisfied or, conversely, when the result is such that no more values can be merged without violating the error threshold.

Consider again the cube in Figure 4.1. In the following we show in detail how the greedy shrink algorithm computes a reduction that solves the error-constrained problem with a maximum total SSE of 20 (Figure 4.5).

1. First, six singleton clusters are created, one for each member.
2. The most promising merge is the one between the *Arlington* and the *Washington*

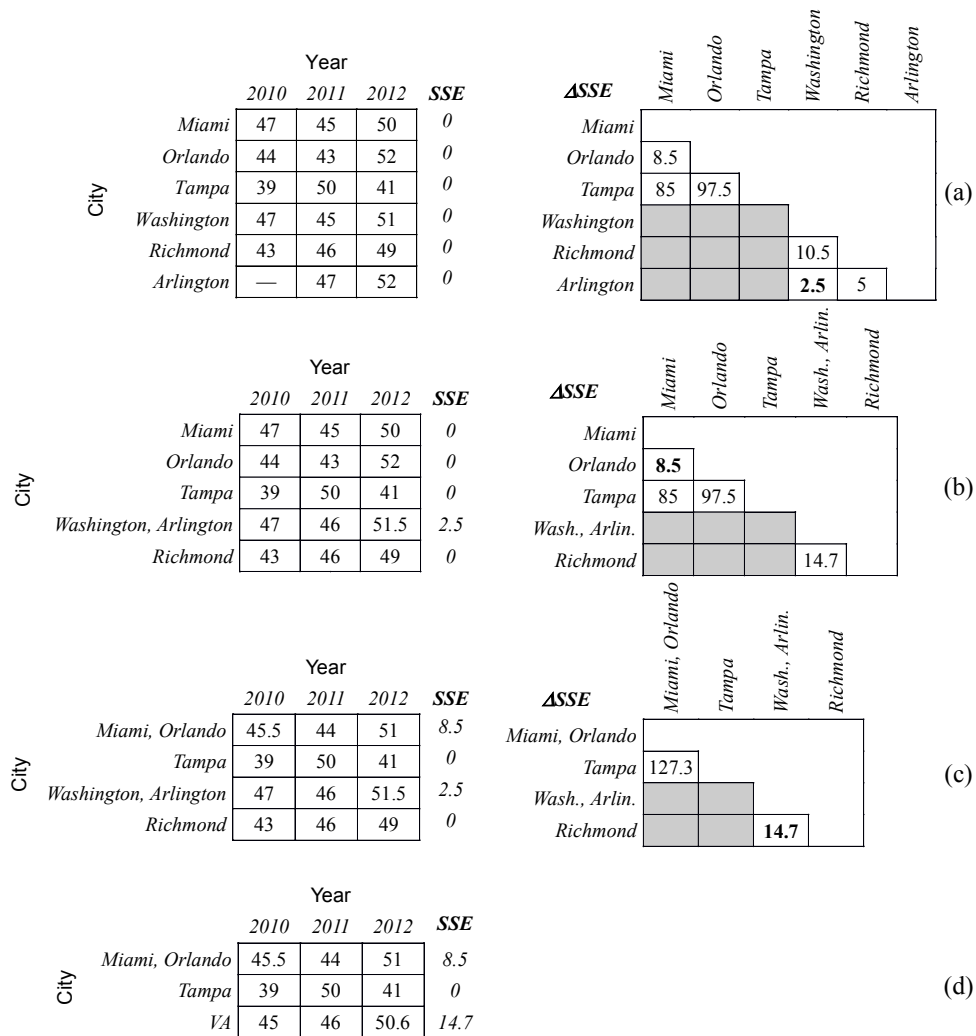


Figure 4.5: Applying the greedy algorithm for shrinking. The left column shows the pivot tables, the right column reports the SSE increase for each feasible merge. Grey cells correspond to non h-compliant merges.

clusters, that yields SSE equal to 2.5 (Figure 4.5.a, right). The SSE of the resulting reduction (Figure 4.5.b, left) is 2.5, which meets the SSE constraint, so there is still room for shrinking.

3. The most promising merge is now the one between the *Miami* and the *Orlando* clusters (Figure 4.5.b, right). The total SSE is 11, so the iterative approach can be repeated.
4. At the next iteration, the algorithm merges *Richmond* cluster with the *Washington—Arlington* cluster (Figure 4.5.c, right). Since the resulting reduction has SSE higher than 20 (Figure 4.5.d), the algorithm stops. The reduction returned is the one shown in Figure 4.5.c, left.

4.4 Optimization Techniques for the Shrink Operator

The shrink implementation proposed in [5] (introduced in Section 4.3) is based on a simple greedy algorithm which is able to find a solution in a small amount of computing time. Unfortunately, as shown in Section 4.4.5, the greedy heuristic may generate solutions too weak for some target applications as the percentage gap from the optimal solution could be of some units. In this work we propose:

- (i) A original formulation of the problem as a set partitioning problem with side constraints.
- (ii) A heuristic method based on dual ascent procedure that exploit pricing and Lagrangian relaxation. The dual ascent procedure provides a near optimal solution for the dual problem and a Lagrangian heuristic generates feasible solutions. The feasible solution generated by the Lagrangian heuristic is usually of good quality and the percentage gap from the optimal solution value is much better than that of the greedy heuristic.
- (iii) An exact method which solves the problem starting from the dual solution found by the dual ascent procedure and using a limited set of variables.

Our contributions create a bridge between BI and optimization techniques. These solutions have become very common in recent years since BI approaches have become more sophisticated and often they require the support of optimization techniques for an effective implementation. One of the classical application of optimization in BI is the development of learning algorithms, where classification, clustering, and regression problems must be solved (e.g., [84], [85]). An interesting introduction to operations research and data mining can be found in the special issue [86] and in the survey [87]. Some mathematical formulations and challenges are also discussed in [88] and [89]. Operational research inspired techniques have been also adopted during the design of BI solutions, for example the problem of selecting the most effective subset of materialized views in Data Warehouse is discussed in [90] and [91]. Operations research is also very useful for optimizing the query execution (e.g., [92], [93]) or the data visualization and discretization (e.g., [94], [95], [96]). The latter is the topic on which this work focuses.

The outline for the contributions introduced above is as follows. In Section 4.4.1 we define the set partitioning formulation of the problem, whereas the dual ascent procedure and the Lagrangian heuristic are described in Sections 4.4.2 and 4.4.3, respectively. The exact method is presented in Section 4.4.4. In Section 4.4.5 we discuss the computational results and in Section 4.4.6 we draw the conclusions.

4.4.1 Mathematical Formulation

In order to achieve a better understanding of the model for optimizing the shrink operator in Figure 4.6 we provide a graphical representation of the optimization process. The algorithms proposed in the next sections are implemented by the main computational module denoted with *Shrink*. The input for such module are:

- The index set $V = \{1, \dots, n\}$ of the n dimensional values of the hierarchy involved in the shrink operation.
- The index set \mathbb{C} of all the feasible (i.e., h-compliant) clusters together with the associated loss of precision, which are computed as described in Section 4.3. For each cluster $j \in \mathbb{C}$ the loss of precision is denoted by e_j .
- The parameter α denoting the maximum size or the maximum loss allowed depending on whether you are solving the size-bound ($goal = S$) or loss-bound ($goal = L$) version of the problem, respectively.

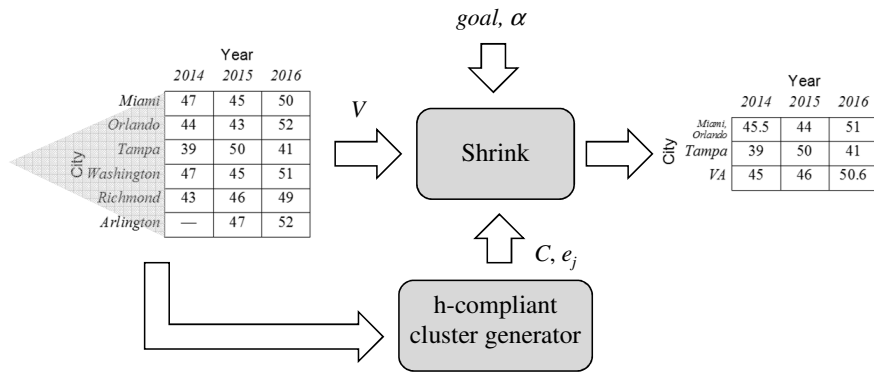


Figure 4.6: The shrink optimization process.

The *h-compliant cluster generator* module is in charge of generating in advance the whole set of h-compliant clusters induced by the involved hierarchy. As we will show in Section 4.4.5, this task can be accomplished in a negligible time when compared with the one required by the *Shrink* module.

We denote with $\mathbb{C}_i \subseteq \mathbb{C}$ the subset of clusters involving the values $i \in V$. C_j represents the index set of the values contained in the cluster $j \in \mathbb{C}$. Let x_j be a 0 – 1 binary variable equal to one if and only if the cluster $j \in \mathbb{C}$ is in the optimal solution. The problem can

be formulated as a set partitioning problem with a side constraint as follows:

$$(P) \quad z_P = \min \sum_{j \in \mathbb{C}} c_j x_j \quad (4.1)$$

$$s.t. \quad \sum_{j \in \mathbb{C}_i} x_j = 1, \quad i \in V \quad (4.2)$$

$$\sum_{j \in \mathbb{C}} a_j x_j \leq \alpha \quad (4.3)$$

$$x_j \in \{0, 1\}, \quad j \in \mathbb{C} \quad (4.4)$$

If $goal = S$, setting $c_j = e_j$ the objective function 4.1 minimizes the loss of precision; conversely, if $goal = L$, setting $c_j = 1$ the objective function 4.1 minimizes the size of the resulting data. Constraints 4.2 ensure that each original dimensional value is included in a cluster. Constraint 4.3 guarantees that the resulting data do not exceed the maximum size allowed by setting $a_j = 1$ and $\alpha = MaxSize$, if $goal = S$, or the maximum loss of precision by setting $a_j = e_j$ and $\alpha = MaxLoss$, if $goal = L$.

Let u_i and v be the dual variables associated to constraints 4.2 and 4.3, respectively. The dual of the LP-relaxation of problem P is the following:

$$(D) \quad z_D = \min \sum_{i \in V} u_i + \alpha v \quad (4.5)$$

$$s.t. \quad \sum_{i \in \mathbb{C}_j} u_i + a_j v \leq c_j, \quad j \in \mathbb{C} \quad (4.6)$$

$$u_i \text{ unconstrained}, \quad i \in V \quad (4.7)$$

$$v \leq 0 \quad (4.8)$$

The dual D is used for defining the dual ascent procedure, described in Section 4.4.2, which is based on a Lagrangian relaxation of the problem P . The dual ascent procedure iteratively improves the dual solution which is used for defining a *core* subset of clusters by means of a pricing procedure. The dual ascent ends providing a near optimal dual solution for the problem D .

The dual solution is also used to define a core subproblem for the exact method proposed in Section 4.4.4. The exact method solves the problem P starting from the dual solution found by the dual ascent and using a limited set of variables.

4.4.2 A Dual Ascent

The dual ascent is based on a *parametric relaxation* of problem P and its Lagrangian relaxation. The resulting problem is solved by a subgradient algorithm which makes use of column generation and embeds a Lagrangian heuristic.

4.4.2.1 Parametric Relaxation

Parametric relaxation is a well-known approach in the literature. Some interesting applications are described by [97] for vehicle routing and by [98] and [99] for crew scheduling. Recently, dual ascent procedures based on a parametric relaxation are proposed by [100] for the set partitioning problem and by [101] for the set covering problem with side constraints. In this section we describe a parametric relaxation of problem P .

We associate with each dimensional values $i \in V$ a positive real weight q_i . Let $q(C_j) = \sum_{i \in C_j} q_i$ be the total weight of column (cluster) $j \in \mathbb{C}$. Since weights $\{q_i\}$ are positive, $q(C_j) > 0$ for every column $j \in \mathbb{C}$. We replace each variable x_j by a new set of $|C_j|$ variables y_j^i , $i \in C_j$, as follows:

$$x_j = \sum_{i \in C_j} \frac{q_i}{q(C_j)} y_j^i, \quad j \in \mathbb{C} \quad (4.9)$$

and the resulting mathematical formulation of the parametric relaxation of problem P is the following:

$$(PR(\mathbf{q})) \quad z_{PR}(\mathbf{q}) = \min \sum_{j \in \mathbb{C}} \sum_{i \in C_j} c_j \frac{q_i}{q(C_j)} y_j^i \quad (4.10)$$

$$s.t. \quad \sum_{j \in \mathbb{C}} \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h = 1, \quad i \in V \quad (4.11)$$

$$\sum_{j \in \mathbb{C}} a_j \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h \leq \alpha, \quad (4.12)$$

$$y_j^i \in \{0, 1\}, \quad j \in \mathbb{C}, i \in V \quad (4.13)$$

Constraints 4.11 and 4.12 correspond to constraints 4.2 and 4.3 of problem P , respectively. Notice that if $y_j^i = 1$ no constraint imposes that $y_j^h = 1$ for every cluster $h \in C_j$ covered by column j , therefore $PR(\mathbf{q})$ is a relaxation of problem P .

4.4.2.2 Lagrangian Relaxation

Problem $PR(\mathbf{q})$ can be relaxed by dualizing constraints 4.11 and 4.12 in a Lagrangian fashion, by means of the penalty vector $\boldsymbol{\lambda} \in \mathbb{R}^{n+1}$ having the first n components λ_i , $i \in V$, unconstrained and $\lambda_{n+1} \leq 0$.

The resulting Lagrangian problem is:

$$(LR(\boldsymbol{\lambda}, \mathbf{q})) \quad z_{LR}(\boldsymbol{\lambda}, \mathbf{q}) = \min \sum_{j \in \mathbb{C}} \sum_{i \in C_j} (c_j - \lambda'(C_j)) \frac{q_i}{q(C_j)} y_j^i + \sum_{i \in V} \lambda_i + \alpha \lambda_{n+1} \quad (4.14)$$

$$s.t. \quad y_j^i \geq 0, \quad i \in V, j \in \mathbb{C} \quad (4.15)$$

where $\lambda'(C_j) = \lambda(C_j) + a_j \lambda_{n+1}$ and $\lambda(C_j) = \sum_{h \in C_j} \lambda_h$. The optimal value of problem $LR(\boldsymbol{\lambda}, \mathbf{q})$ is a valid lower bound for the original problem P and it can be strengthened adding the constraint $\sum_{j \in \mathbb{C}_i} y_j^i = 1$ for every $i \in V$.

Problem $LR(\boldsymbol{\lambda}, \mathbf{q})$ is decomposable into $|V|$ subproblems, one for each row $i \in V$:

$$(LR^i(\boldsymbol{\lambda}, \mathbf{q})) \quad z_{LR}^i(\boldsymbol{\lambda}, \mathbf{q}) = \min \sum_{j \in \mathbb{C}_i} c_j^i(\boldsymbol{\lambda}, \mathbf{q}) y_j^i + \lambda_i \quad (4.16)$$

$$s.t. \quad \sum_{j \in \mathbb{C}_i} y_j^i = 1 \quad (4.17)$$

$$y_j^i \in \{0, 1\}, \quad j \in \mathbb{C}_i \quad (4.18)$$

where the cost of each variable y_j^i is $c_j^i(\boldsymbol{\lambda}, \mathbf{q}) = c_j' \frac{q_i}{Q(C_j)}$ and $c_j' = c_j - \lambda(C_j) - a_j \lambda_{n+1}$. Hence, the overall value of the Lagrangian problem is $z_{LR}(\boldsymbol{\lambda}, \mathbf{q}) = \sum_{i \in V} z_{LR}^i(\boldsymbol{\lambda}, \mathbf{q}) + \alpha \lambda_{n+1}$.

Theorem 1 shows that any optimal solution of problem $LR(\boldsymbol{\lambda}, \mathbf{q})$ provides a feasible solution (\mathbf{u}, v) of cost $z_{LR}(\boldsymbol{\lambda}, \mathbf{q})$ for the dual problem D .

Theorem 1 *Let $\boldsymbol{\lambda}$ be a vector of $n + 1$ real numbers, where $\lambda_i, i \in V$, are unconstrained and $\lambda_{n+1} \leq 0$. Let \mathbf{q} be a vector of n positive real numbers, i.e., $q_i > 0$, for every $i \in V$. A feasible dual solution (\mathbf{u}, v) of cost $z_{LR}(\boldsymbol{\lambda}, \mathbf{q})$ for dual problem D can be obtained by means of the following expressions:*

$$\begin{aligned} u_i &= q_i \min_{j \in \mathbb{C}_i} \left\{ \frac{c_j'}{Q(C_j)} \right\} + \lambda_i, \quad i \in V \\ v &= \lambda_{n+1}, \end{aligned} \quad (4.19)$$

where $c_j' = c_j - \lambda(C_j) - a_j \lambda_{n+1}$, $\lambda(C_j) = \sum_{i \in C_j} \lambda_i$, and $Q(C_j) = \sum_{i \in C_j} q_i$.

Proof: Let us consider the dual constraint 4.6 corresponding to column $j \in \mathbb{C}$ of the LP-relaxation of P . For every column j , the following inequalities hold:

$$\min_{h \in \mathbb{C}_i} \left\{ \frac{c_h'}{Q(C_h)} \right\} \leq \frac{c_j'}{Q(C_j)}, \quad \text{for every } i \in C_j \quad (4.20)$$

From expression 4.19 we obtain

$$u_i \leq q_i \frac{c_j'}{Q(C_j)} + \lambda_i, \quad i \in C_j, j \in \mathbb{C} \quad (4.21)$$

and by adding inequalities 4.21 we derive

$$\sum_{i \in C_j} u_i \leq \sum_{i \in C_j} \left(q_i \frac{c_j'}{Q(C_j)} + \lambda_i \right), \quad j \in \mathbb{C} \quad (4.22)$$

Therefore, considering the dual constrain 4.6 for every $j \in \mathbb{C}$, we have

$$\begin{aligned}
\sum_{i \in \mathbb{C}_j} u_i + a_j v &\leq \frac{c'_j}{Q(C_j)} \sum_{i \in \mathbb{C}_j} q_i + \sum_{i \in \mathbb{C}_j} \lambda_i + a_j v \\
&\leq \frac{c'_j}{Q(C_j)} Q(C_j) + \lambda(C_j) + a_j v \\
&\leq c'_j + \lambda(C_j) + a_j v \\
&\leq c_j - \lambda(C_j) - a_j v + \lambda(C_j) + a_j v \\
&\leq c_j
\end{aligned} \tag{4.23}$$

It is straightforward to show that the dual solution (\mathbf{u}, v) is of cost $z_D(\mathbf{u}, v) = \sum_{i \in V} u_i + \alpha v = z_{LR}(\boldsymbol{\lambda}, \mathbf{q})$.

The dual solution obtained according to Theorem 1 can be further improved by applying the greedy procedure described in [102] or [103].

Corollary 1 shows that the best lower bound that can be achieved using expression 4.19 is equal to the optimal solution cost z_D of the dual problem D and that this value can be obtained searching the maximum of the function $z_{LR}(\boldsymbol{\lambda}, \mathbf{q})$ with respect to $\boldsymbol{\lambda}$.

Corollary 1 *For every $\mathbf{q} > \mathbf{0}$, $\mathbf{q} \in \mathbb{R}^n$, the following equality holds:*

$$\max\{z_{LR}(\boldsymbol{\lambda}, \mathbf{q}) : \boldsymbol{\lambda} \in \mathbb{R}^{n+1}, \lambda_{n+1} \leq 0\} = z_D \tag{4.24}$$

Proof: Let (\mathbf{u}^*, v^*) be an optimal solution of problem D of cost z_D . For every $j \in \mathbb{C}$, we have

$$c_j - \sum_{h \in \mathbb{C}_j} u_h^* - a_j v^* \geq 0 \tag{4.25}$$

and for every $i \in V$, there exists at least a column $j' \in \mathbb{C}_i$ such that

$$c_{j'} - \sum_{h \in \mathbb{C}_{j'}} u_h^* - a_{j'} v^* = 0. \tag{4.26}$$

If for a given $i \in V$ a column j' satisfying equality 4.26 does not exist, we can improve the “optimal dual solution” increasing the corresponding dual variable u_i , in contradiction with the hypothesis.

By setting $\boldsymbol{\lambda} = (\mathbf{u}^*, v^*)$, when we evaluate the dual solution by expression 4.19 we have $u_i = q_i \min_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\} + u_i = 0 + u_i$, for every $i \in V$, and $v = v^*$. Therefore, $z_{LR}(\boldsymbol{\lambda}, \mathbf{q}) = \sum_{i \in V} z_{LR}^i(\boldsymbol{\lambda}, \mathbf{q}) + \alpha \lambda_{n+1} = \sum_{i \in V} u_i + \alpha v = z_D$.

In order to find the optimal (or near optimal) dual solution of cost z_D we need to solve the Lagrangian Dual $\max\{z_{LR}(\boldsymbol{\lambda}, \mathbf{q}) : \boldsymbol{\lambda} \in \mathbb{R}^{n+1}, \lambda_{n+1} \leq 0\}$.

We propose a dual ascent procedure based on a subgradient algorithm and on column generation following the approach proposed by [100] for the set partitioning problem.

DUAL ASCENT PROCEDURE

Step 1. **Initial setup**

Set $z_{LB} = -\infty$, $\beta = \beta_0$, and the initial penalty vector $\boldsymbol{\lambda} = \mathbf{0}$.
Generate an initial *core* subset of columns $\mathbb{C}' \subseteq \mathbb{C}$.

Step 2. **Solve Lagrangian Problem**

Solve $LR(\boldsymbol{\lambda}, \mathbf{q})$ using only the columns in the core \mathbb{C}' .
Compute (\mathbf{u}, v) according to Theorem 1 and improve it using the greedy algorithm described in [103].

Step 3. **Pricing**

Generate a subset $Q \subseteq \mathbb{C}$ of columns having negative reduced costs with respect to (\mathbf{u}, v) , i.e., $Q = \{j \in \mathbb{C} : c_j - \sum_{i \in C_j} u_i - a_j v < 0\}$.
Add subset Q to the core \mathbb{C}' , i.e., $\mathbb{C}' = \mathbb{C}' \cup Q$.
If $Q = \emptyset$, then (\mathbf{u}, v) is a feasible dual solution for problem P , therefore $z_{LB} = \max\{z_{LB}, LR(\boldsymbol{\lambda}, \mathbf{q})\}$ and all columns of reduced cost larger than $\varepsilon_0 z_{LB}$ are removed from \mathbb{C}' .

Step 4. **Update Lagrangian penalties**

Compute subgradient components:

- $\theta_i = 1 - \sum_{j \in C_i} \sum_{h \in C_j} \frac{q_h}{q(C_j)} y_j^h$, for every $i \in V$
- $\theta_{n+1} = \alpha - \sum_{j \in \mathbb{C}} \sum_{h \in C_j} a_j \frac{q_h}{q(C_j)} y_j^h$

Compute the step size $\sigma = \beta \frac{0.01 \times z_{LR}(\boldsymbol{\lambda}, \mathbf{q})}{\sum_{i=1}^{n+1} \theta_i^2}$ and update the Lagrangian vector $\boldsymbol{\lambda}$:

- $\lambda_i = \lambda_i + \alpha \theta_i$, for every $i \in V$
- $\lambda_{n+1} = \min\{0, \lambda_{n+1} - \sigma \theta_{n+1}\}$

Step 5. **Stop Conditions**

If the maximum number of iterations $MaxIter$ is not reached and the lower bound has improved enough (i.e., the improvement is larger than $\varepsilon_1 z_{LB}$) during last $MaxIter_0$ iterations go to Step 2.

In this work we generate the full set \mathbb{C} in advance, before starting the DUAL ASCENT PROCEDURE, because it is not time consuming. But the dual ascent procedure works with a small subset of columns, called *core*, adding new columns only when required. Working with a core allows a large computing time saving. The initial core is generated by considering in turn the columns in \mathbb{C} sorted for non-decreasing order of the value $c_j/|C_j|$. If the column covers a row already covered by another column in the core or violates the side constraints is ignored, otherwise it is added to the core.

Notice that $z_{LR}(\boldsymbol{\lambda}, \mathbf{q})$ is a valid lower bound for problem P if and only if no columns of negative reduced costs exist (i.e., $Q = \emptyset$), with respect to the corresponding dual solution (\mathbf{u}, v) , which is feasible in this case. When the dual solution (\mathbf{u}, v) is feasible, we remove from \mathbb{C}' all columns of reduced cost larger than $\varepsilon_0 z_{LB}$ to maintain the core as small as possible. Instead, the parameter ε_1 is used in the *stop conditions* to check if the lower bound has been improved enough during the last $MaxIter_0$ iterations.

In order to improve the convergence to a near optimal dual solution, we update the step-size parameter β during the execution. If after a given number of iterations $MaxIter_1$, the lower bound is not improved, we decrease β , i.e., $\beta = \gamma_1 \beta$, where $\gamma_1 < 1$. As soon as the lower bound is improved we increase β , i.e., $\beta = \gamma_2 \beta$, where $\gamma_2 > 1$.

The complete definition of the parameter values can be found at Section 4.4.5, where the computational results are described.

4.4.3 A Lagrangian Heuristic

The dual ascent procedure can provide an effective lower bound for problem P . However, following a “*matheuristic*” approach (see [104, 105]), a further possible by-product could be a Lagrangian heuristic algorithm.

The proposed Lagrangian heuristic is applied at each iteration of the dual ascent procedure, where the lower bound is improved, a simple greedy approach based on the solution of the Lagrangian problem $LR(\boldsymbol{\lambda}, \mathbf{q})$ and on the corresponding *penalized cost*.

At the beginning, the procedure builds an initial partial solution using the columns (i.e., configurations) $\mathbb{C}'' = \left\{ j' = \operatorname{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\} : i \in V \right\}$. Then, the procedure tries to complete the emerging solution considering the remaining columns of the core \mathbb{C}' sorted in non decreasing order of their penalized cost $c'_j = c_j - \sum_{i \in \mathbb{C}_j} u_i - a_j v$.

LAGRANGIAN HEURISTIC

Step 1. Initial setup

Let z_{UB}^{best} be the best upper bound found so far.
Set $z_{UB} = 0$, $x'_j = 0$, for every $j \in \mathbb{C}$, and $iter = 1$.

Step 2. Phase 1: Build a partial solution from the LR solution

For each $i \in V$ try to add to the emerging solution column $j' = \operatorname{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(C_j)} \right\}$. If $\sum_{i' \in \mathbb{C}_{j'}} \sum_{j \in \mathbb{C}_{i'}} x'_j = 0$, $\sum_{j \in \mathbb{C}} a_j x'_j \leq \alpha - a_{j'}$, and $z_{UB} + c_{j'} < z_{UB}^{best}$, column j' is added to the emerging solution, i.e., $x'_{j'} = 1$ and $z_{UB} = z_{UB} + c_{j'}$.

Step 3. Check if the emerging solution is complete

If $\sum_{j \in \mathbb{C}_i} x'_j = 1$ for every $i \in V$, the solution is feasible, therefore update the current best solution $z_{UB}^{best} = z_{UB}$, $\mathbf{x}^{best} = \mathbf{x}'$, and STOP.

Step 4. Phase 2: Complete the emerging solution

If there exists at least a row $i \in V$ such that $\sum_{j \in \mathbb{C}_i} x'_j = 0$, we try to *complete* the emerging solution by considering the remaining columns in the core \mathbb{C}' in non-decreasing order of their penalized cost $c'_j = c_j - \sum_{h \in \mathbb{C}_j} u_h - a_j v$. If $\sum_{i' \in \mathbb{C}_{j'}} \sum_{j \in \mathbb{C}_{i'}} x'_j = 0$, $\sum_{j \in \mathbb{C}} a_j x'_j \leq \alpha - a_{j'}$, and $z_{UB} + c_{j'} < z_{UB}^{best}$, column j' is added to the emerging solution, i.e., $x'_{j'} = 1$ and $z_{UB} = z_{UB} + c_{j'}$.

Step 5. Check if the emerging solution is complete

If $\sum_{j \in \mathbb{C}_i} x'_j = 1$ for every $i \in V$, the solution is feasible, therefore update the current best solution $z_{UB}^{best} = z_{UB}$, $\mathbf{x}^{best} = \mathbf{x}'$; otherwise the Lagrangian heuristic was not able to find a feasible solution of cost smaller than z_{UB}^{best} .

Notice that when the LAGRANGIAN HEURISTIC adds a column j' to the emerging solution all the rows are covered by at most one column, the side constraint is satisfied, and its cost z_{UB} is less than z_{UB}^{best} . Therefore, as soon as the emerging solution covers all rows, it is certainly feasible and better than the current best solution of cost z_{UB}^{best} .

In the computational results, the LAGRANGIAN HEURISTIC is performed twice, changing the order of the dimensional values $i \in V$ in phase 1. At the first iteration we consider the data in the same order given in input, whereas in the second iteration the dimensional values $i \in V$ are considered in non-increasing order of value $y_{j'}^i$, where $j' = \operatorname{argmin}_{j \in \mathbb{C}_i} \left\{ \frac{c'_j}{Q(\mathbb{C}_j)} \right\}$. The Lagrangian heuristic is performed only after the percentage gap between the lower and the upper bounds is under the 50%.

4.4.4 An Exact Method

Using heuristic algorithms we can obtain effective feasible solutions in a small computing time, and by the dual ascent procedure we can evaluate the maximum distance from the optimal solution value. But when we need to evaluate the optimal value, the only possibility is the use of an *exact* method.

In this work we propose an exact method based on an approach similar to the ones described in [99], and [100].

The proposed approach computes a near optimal dual solution by the DUAL ASCENT PROCEDURE and using the corresponding reduced cost $c'_j = c_j - \sum_{i \in \mathbb{C}_j} u_i - a_j v$ it generates a reduced problem P' replacing in P the set \mathbb{C} with the subset \mathbb{C}' and replacing the original cost c_j with the reduced cost c'_j . The subset \mathbb{C}' is the largest subset of the lowest reduced cost variables such that $c'_j < \min\{g^{max}, z_{UB} - z_{LB}\}$ and $|\mathbb{C}'| < \Delta^{max}$. The resulting reduced problem P' is solved by a MIP solver. Given the solution of P' , we are able to check if it is optimal for the original problem P . If it is not optimal the subset \mathbb{C}' is enlarged and we solve again the new reduced problem.

The resulting exact method could be summarized as follows.

EXACT ALGORITHM

Step 1. **Initial setup**

Set $z_{LB} = -\infty$, $z_{UB} = \infty$, $iter = 1$, and $\Delta^{max} = \Delta_0$.

Step 2. **Computing a lower bound z'_D**

Compute a solution (\mathbf{u}', v') of the dual problem D of cost $z_{LB} = z'_D$ using the DUAL ASCENT embedding the LAGRANGIAN HEURISTIC which provides an upper bound z_{UB} .
Set $g^{max} = \mu_1 z_{LB}$.

If $z_{LB} = z_{UB}$, then STOP.

Step 3. **Define a reduced problem P'**

Let $c'_j = c_j - \sum_{i \in C_j} u_i - a_j v$ be the reduced cost of cluster $j \in \mathbb{C}$ with respect to the dual solution (\mathbf{u}', v') .

Let \mathbb{C}' be the largest subset of the lowest reduced cost variables such that $c'_j < \min\{g^{max}, z_{UB} - z_{LB}\}$ and $|\mathbb{C}'| < \Delta^{max}$.

Define the reduced problem P' replacing in P the set \mathbb{C} with \mathbb{C}' and replacing the original cost c_j with the reduced cost c'_j .

Step 4. **Solve problem P'**

Solve problem P' using a general purpose MIP solver (e.g., IBM Ilog Cplex).

Let $z_{P'}^*$ be the cost of the optimal solution \mathbf{x}^* obtained (we assume $z_{P'}^* = \infty$ if the set \mathbb{C}' does not contain any feasible solution).

Update $z_{UB} = \min\{z_{UB}, z_{P'}^* + z'_D\}$.

Step 5. **Test if \mathbf{x}^* is optimal for the original problem P**

Let $c_{max} = \max\{c'_j : j \in \mathbb{C}'\}$, if $\mathbb{C}' \subset \mathbb{C}$, otherwise $c_{max} = \infty$, if $\mathbb{C}' = \mathbb{C}$. We have two cases:

- (a) $z_{P'}^* \leq c_{max}$, then Stop because \mathbf{x}^* is guaranteed to be an optimal solution for the original problem P .
- (b) $z_{P'}^* > c_{max}$, then \mathbf{x}^* is not guaranteed to be an optimal solution for the original problem P , however $z'_D + c_{max}$ is a valid lower bound on the optimal solution value of problem P .

Step 6. **Update the parameters** If $iter < MaxIter$, then increase $\Delta^{max} = \mu_2 \Delta^{max}$ and $g^{max} = \mu_2 g^{max}$, $\mu_2 > 1$, set $iter = iter + 1$ and go to Step 3.

The procedure terminates when the optimal solution of P is obtained or the maximum number of iterations is reached. Notice that if we set $MaxIter = \infty$, the procedure converges to the optimal solution because in the worst case at a given iteration $\mathbb{C}' = \mathbb{C}$.

4.4.5 Computational Results

The algorithms presented in this work have been coded in C++ using Microsoft Visual Studio 2010, and run on a workstation equipped with an Intel Core i7-3770, 3.40 GHz,

32Gb of RAM, and operating system Windows 10 64bit. IBM Ilog CPLEX 12.5 is used as LP and MIP solver.

For our experiments we considered four different hierarchies: RESIDENCE, OCCUPATION, PROD_DEPARTMENT, and PROD_BRAND. The former two come from the IPUMS database [9], while the others two are extracted from the Foodmart database that can be found with the Pentaho suite [106]. After aggregating the data along these hierarchies (e.g., by state or by city), we generated by means of sampling several test instances with varying characteristics, such as size and average fan-out (i.e., children per parent ratio). During the rest of this work we refer to these instances using the name of the hierarchy level used to aggregate the data followed by a progressive number; for example, CITY-1 means that the data has been first aggregated by city, and then a sampling process has been performed to create the dataset. To observe how the algorithms behave not only with different problem sizes (i.e., number of clusters) but also with different data distributions, we generated instances CITY_UNI and OCCUPATION_UNI by reusing the hierarchical structure of RESIDENCE and OCCUPATION, but with uniformly random data slices. Finally, we generated some hard instances to show that the new proposed algorithm solves instances where a general purpose solver fails. These instances are the ones reported in Tables 4.4, 4.5, 4.8, and 4.9.

For each instance, the number of values in the hierarchy and the number of generated clusters are shown alongside the results of the experiments in Tables 4.2–4.5. We also remark that a thorough description of the dataset can be found in [5].

For every test instance we solve both versions of the problem. In Tables 4.2, 4.4, 4.6, and 4.8, we solve the problem of *Type A*, where the objective function minimizes the size of the resulting data and the side constraint guarantees that the loss of precision does not exceed a given maximum value. Whereas, in Tables 4.3, 4.5, 4.7, and 4.9 we solve the problem of *Type B*, where the objective function minimizes the loss of precision and the side constraint guarantees that the size of the resulting data does not exceed a given maximum value.

For every problem type we solve the problem for different values of the maximum loss of precision or of the maximum size of the data.

When we report in Tables 4.2–4.9 the value of the the maximum loss of precision, we use the notation $1.00M$ and $1.00G$ for representing the values 1.00×10^6 and 1.00×10^9 , respectively. When a computing time or a percentage gap is equal to 0.00, it means that its real value is smaller than 0.01.

4.4.5.1 Dual Ascent procedure

In Tables 4.2, 4.3, 4.4, and 4.5 we compare the results obtained by the CPLEX LP solvers and by the dual ascent procedure, described in Section 4.4.2.

In our computational experiments the parameters of the dual ascent are set as follows. The parameters for defining the step size are $\beta_0 = 20$, $\gamma_1 = 0.90$, $\gamma_2 = 1.02$, for problems of Type A, and $\beta_0 = 1$, $\gamma_1 = 0.90$, $\gamma_2 = 1.005$, for problems of Type B. The parameter ε_0 used for reducing the size of the subset \mathbb{C}' is 0.05, i.e., 5% of the value of the current lower bound. Instead, the parameter ε_1 used to check if the lower bound has improved enough during the last $MaxIter_0$ iterations is $\varepsilon_1 = 0.001$. The maximum number of iterations are $MaxIter = 100000$ (i.e., virtually infinite), $MaxIter_0 = 200$, and $MaxIter_1 = 5$.

The choice of the parameter values is done empirically because the purpose of the computational tests is to show that just with a *good* choice we can achieve effective results. Therefore, a better analysis on the choice of the parameter values is out of scope for this work, but it will be an interesting research direction for the future.

For each test instance we report the number of clusters m , the number of dimensional values n , the computing time for generating the clusters T_{Gen} , and the right-hand side of the side constraint α , which is the maximum loss of precision for problem of Type A and the maximum size of the data for problem of Type B.

For the CPLEX solvers we report the optimal value z_{LP} of the LP-relaxation of the problem P and the computing times $Time_x$ for each solver available: primal (P), dual (D), network (N), barrier (B), and sifting (S).

For the dual ascent we report the best lower bound z_{LB} corresponding to the best feasible dual solution generated, its percentage gap from z_{LP} $Gap_{LP} = 100 \times \frac{z_{LP} - z_{LB}}{z_{LP}}$, the number of iterations $Iter$, the computing time $Time$, and the size of subset \mathbb{C}' at the end of the execution.

The more effective CPLEX LP solver is the network simplex, in particular for problems of Type B, at the contrary the barrier is very time consuming for both problem types.

The dual ascent provides near optimal solution in a smaller computing time with respect to CPLEX LP solvers for problem of type A. It generates lower bounds having an average percentage gap Gap_{LP} from the optimal solution value z_{LP} equal to 0.02% and it is on average about 2.4 time faster than the better CPLEX LP solver. For problems of Type B, the dual ascent is a little worse only with respect the network simplex and the average percentage gap Gap_{LP} is under 0.01%. Even the results on the hard instances, reported in Tables 4.4 and 4.5, confirm these figures.

Table 4.2: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type A.

Instances	Size and Generation			Cplex				Dual Ascent								
	m	n	T_{Gen}	Side Cons.	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Time_S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$ C $
state	629	46	0.00	10.00G	10.00G	29.60	0.00	0.00	0.00	0.00	0.00	29.60	0.00	303	0.00	35
state	629	46	0.00	1000.00M	1000.00M	32.00	0.00	0.00	0.00	0.00	0.00	32.00	0.00	202	0.01	32
prod.department	32809	28	0.04	1000.00M	1000.00M	3.51	0.05	0.07	0.19	0.10	0.10	3.51	0.00	783	0.03	19
prod.department	32809	28	0.04	100.00M	100.00M	9.31	0.04	0.04	0.04	0.03	0.05	9.31	0.00	454	0.01	17
prod.brand	37233	697	0.06	10.00M	10.00M	33.01	0.67	0.66	1.16	1.86	1.86	33.00	0.00	774	0.23	175
prod.brand	37233	697	0.06	1000000.00	1000000.00	167.70	0.63	0.64	0.62	1.00	1.00	167.68	0.01	1025	0.09	1180
city-1	64069	495	0.09	100.00G	100.00G	72.04	0.26	0.32	0.29	1.34	1.34	72.01	0.05	688	0.14	1013
city-1	64069	495	0.09	10.00G	10.00G	140.34	0.17	0.20	0.25	1.53	1.53	140.34	0.00	603	0.11	2547
city-1	64069	495	0.09	100.00G	100.00G	306.22	0.09	0.08	0.09	0.12	0.14	306.21	0.00	708	0.06	807
city-2	121413	523	0.07	100.00G	100.00G	75.55	0.67	0.41	0.38	5.60	2.41	75.49	0.08	746	0.33	1789
city-2	121413	523	0.07	10.00G	10.00G	148.69	0.42	0.35	0.39	3.96	0.55	148.69	0.00	759	0.27	3699
city-2	121413	523	0.07	100.00M	100.00M	324.56	0.15	0.15	0.16	0.23	0.24	324.52	0.01	660	0.09	920
city-3	227909	549	0.22	100.00G	100.00G	77.91	1.46	1.06	0.96	12.66	4.43	77.90	0.02	767	0.67	7820
city-3	227909	549	0.22	100.00M	100.00M	153.97	0.87	0.66	0.80	153.97	0.00	153.97	0.00	746	0.49	5392
city-3	227909	549	0.22	100.00G	100.00G	341.60	0.28	0.28	0.30	0.40	0.39	341.54	0.02	615	0.16	996
city-4	432709	574	0.24	100.00G	100.00G	82.46	3.23	2.09	2.06	34.18	7.06	82.43	0.04	670	1.21	12144
city-4	432709	574	0.24	10.00G	10.00G	158.09	1.70	1.35	1.76	18.59	2.21	158.09	0.00	746	0.78	5337
city-4	432709	574	0.24	100.00M	100.00M	353.65	0.56	0.55	0.58	0.80	0.71	353.61	0.01	615	0.28	1226
city-5	647749	584	0.35	100.00G	100.00G	81.23	4.54	4.19	4.45	61.05	8.93	81.22	0.02	669	1.88	15801
city-5	647749	584	0.35	10.00G	10.00G	158.45	2.74	2.37	2.77	29.88	2.73	158.43	0.01	645	1.25	7534
city-5	647749	584	0.35	100.00M	100.00M	355.73	0.84	0.84	0.88	1.22	1.13	355.62	0.03	638	0.47	1281
city-6	793157	596	0.56	100.00G	100.00G	5.64	5.43	3.48	3.48	73.03	9.94	84.17	0.12	567	2.34	28823
city-6	793157	596	0.56	10.00G	10.00G	161.79	3.39	2.55	3.31	48.16	3.53	161.78	0.01	652	1.60	8636
city-6	793157	596	0.56	100.00M	100.00M	364.93	1.03	1.02	1.06	1.51	1.33	364.93	0.00	620	0.54	1316
city-7	1184157	516	0.69	100.00G	100.00G	71.71	8.54	7.40	6.94	101.42	12.07	71.70	0.02	574	3.16	40178
city-7	1184157	516	0.69	10.00G	10.00G	132.34	4.84	3.92	4.43	64.41	6.38	132.33	0.01	745	2.53	11153
city-7	1184157	516	0.69	100.00M	100.00M	313.63	1.58	1.59	1.62	1.99	1.98	313.59	0.01	588	0.71	1072
city-8	2167197	531	1.29	100.00G	100.00G	72.30	15.99	20.25	14.16	225.72	23.79	72.29	0.01	670	5.67	28582
city-8	2167197	531	1.29	10.00G	10.00G	138.62	9.16	8.00	10.41	145.72	10.87	138.61	0.01	784	4.56	15619
city-8	2167197	531	1.29	100.00M	100.00M	318.95	3.11	3.09	3.13	3.80	3.67	318.93	0.00	592	1.44	1401
city-9	4133801	573	2.12	100.00G	100.00G	78.83	29.22	33.53	29.25	589.45	34.50	78.81	0.02	688	11.00	29060
city-9	4133801	573	2.12	10.00G	10.00G	146.90	21.82	17.51	22.02	375.21	19.58	146.88	0.02	749	8.89	22182
city-9	4133801	573	2.12	100.00M	100.00M	347.41	6.40	6.35	6.37	7.52	7.46	347.40	0.00	601	2.75	1422
city-10	2693701	635	1.57	100.00G	100.00G	93.37	19.77	15.85	15.85	385.89	30.56	93.37	0.00	722	8.61	58052
city-10	2693701	635	1.57	10.00G	10.00G	172.91	11.70	8.54	12.05	179.82	16.12	172.89	0.01	764	5.45	13882
city-10	2693701	635	1.57	100.00M	100.00M	399.50	3.86	3.85	3.89	4.80	4.70	399.49	0.00	604	1.81	1520
city-11	4921925	652	2.93	100.00G	100.00G	93.89	41.65	45.34	44.68	799.01	49.17	93.88	0.00	694	13.83	39078
city-11	4921925	652	2.93	10.00G	10.00G	174.12	21.81	19.17	24.76	434.77	19.06	174.11	0.01	824	10.15	18963
city-11	4921925	652	2.93	100.00M	100.00M	405.85	7.53	7.49	7.52	8.77	8.93	405.83	0.01	574	3.18	1567
occupation-1	875995	357	0.53	100.00G	100.00G	66.82	5.79	4.38	3.43	84.24	6.58	66.80	0.03	601	1.77	3043
occupation-1	875995	357	0.53	10.00G	10.00G	129.32	2.32	1.66	1.95	19.48	1.93	129.29	0.03	669	0.97	1120
occupation-1	875995	357	0.53	100.00M	100.00M	259.20	1.14	1.14	1.14	1.35	1.35	259.06	0.05	495	0.39	469
occupation-2	3300827	382	2.01	100.00G	100.00G	69.54	23.83	33.56	23.21	527.44	22.95	69.52	0.03	753	8.50	17176
occupation-2	3300827	382	2.01	10.00G	10.00G	135.67	9.43	7.68	8.43	104.43	7.45	135.62	0.03	612	3.49	1818
occupation-2	3300827	382	2.01	100.00M	100.00M	279.65	5.00	4.96	4.96	4.47	5.50	279.60	0.02	593	1.89	523
city.uni-1	1184157	516	0.67	1000.00G	1000.00G	67.72	7.21	9.56	6.66	89.46	14.74	67.71	0.02	672	3.12	12499
city.uni-1	1184157	516	0.67	100.00G	100.00G	126.71	5.61	4.37	6.29	73.31	6.30	126.70	0.01	593	1.96	10400
city.uni-1	1184157	516	0.67	1000.00M	1000.00M	303.57	1.60	1.58	1.63	2.34	2.03	303.56	0.00	644	0.82	1071
city.uni-2	227909	549	0.13	1000.00G	1000.00G	72.07	1.43	0.97	0.80	14.09	3.21	72.05	0.03	715	0.69	7947
city.uni-2	227909	549	0.13	100.00G	100.00G	147.19	0.98	0.73	0.90	8.40	1.43	147.18	0.01	729	0.51	5019
city.uni-2	227909	549	0.13	1000.00M	1000.00M	337.34	0.30	0.30	0.32	0.49	0.46	337.34	0.00	723	0.18	1131
city.uni-3	647749	584	0.37	1000.00G	1000.00G	74.99	4.28	3.92	3.50	57.19	8.01	74.98	0.01	665	2.28	31434
city.uni-3	647749	584	0.37	100.00M	100.00M	151.69	3.10	2.42	2.99	32.62	4.48	151.69	0.00	723	1.38	9696
city.uni-3	647749	584	0.37	1000.00M	1000.00M	352.75	0.85	0.85	0.91	1.41	1.14	352.75	0.00	667	0.47	1663
occupation.uni	875995	357	0.52	1000.00G	1000.00G	64.24	5.63	4.92	3.75	77.25	7.75	64.23	0.02	666	2.48	24496
occupation.uni	875995	357	0.52	100.00G	100.00G	124.11	2.68	1.80	2.11	25.12	2.05	124.10	0.01	738	1.21	1462
occupation.uni	875995	357	0.52	1000.00M	1000.00M	253.41	1.13	1.13	1.12	1.00	1.26	253.41	0.00	524	0.40	452
Avq			0.76			5.59	5.49	5.55	83.47	7.01	660	660	0.02	660	2.27	9188

Table 4.3: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type B.

Instances	Size and Generation				Side Cons.		Cplex					Dual Ascent				
	m	n	T_{Gen}	α	z_{LP}	$Time_P$	$Time_D$	$Time_N$	$Time_B$	$Times$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$ C $	
state	629	46	0.00	10	8295.08G	0.00	0.00	0.00	0.01	0.01	8295.08G	0.00	650	0.01	12	
prod_department	32809	28	0.04	10	3274.11G	0.00	0.00	0.00	0.01	0.02	3273.71G	0.00	1152	0.01	28	
prod_department	32809	28	0.04	10	65.37M	0.07	0.07	0.06	1.12	0.07	65.37M	0.00	559	0.02	14	
prod_brand	37233	697	0.06	50	4.70M	0.07	0.06	0.06	1.12	0.07	4.70M	0.00	512	0.01	15	
prod_brand	37233	697	0.06	100	4.63M	1.41	1.23	1.09	88.72	0.99	4.63M	0.00	692	0.09	1219	
city-1	64069	495	0.09	50	1.81M	1.80	1.22	1.17	15.69	0.96	1.81M	0.01	806	0.07	1191	
city-1	64069	495	0.09	100	198.40G	0.37	0.50	0.21	4.86	1.28	198.34G	0.03	998	0.48	1201	
city-1	64069	495	0.09	100	39.57G	0.37	0.45	0.25	1.41	0.82	39.56G	0.03	1014	0.42	1070	
city-1	64069	495	0.09	150	7.50G	0.32	0.38	0.25	1.21	0.30	7.50G	0.00	954	0.30	970	
city-2	121413	523	0.07	50	214.22G	0.96	1.09	0.40	8.62	12.11	214.21G	0.01	1108	0.96	1302	
city-2	121413	523	0.07	100	45.78G	0.74	0.95	0.46	2.50	1.33	45.78G	0.00	936	0.85	1155	
city-2	121413	523	0.07	150	9.65G	0.80	0.78	0.49	2.29	0.59	9.65G	0.00	993	0.67	1076	
city-3	227909	549	0.22	50	230.64G	1.81	2.36	0.80	16.74	17.09	230.64G	0.00	1061	1.83	1406	
city-3	227909	549	0.22	100	50.62G	1.78	1.90	0.93	3.90	2.16	50.62G	0.01	1176	1.76	1230	
city-3	227909	549	0.22	150	11.15G	1.34	1.57	0.99	4.23	1.04	11.15G	0.00	1062	1.39	1178	
city-4	432709	574	0.24	50	259.88G	3.34	4.58	2.04	33.39	27.56	259.87G	0.01	1147	3.59	1639	
city-4	432709	574	0.24	100	57.25G	3.42	4.25	2.14	6.65	3.18	57.25G	0.00	1167	3.47	1403	
city-4	432709	574	0.24	150	12.40G	3.46	3.53	2.07	6.39	1.83	12.40G	0.00	1034	2.71	1274	
city-5	647749	584	0.35	50	256.84G	4.16	8.03	2.81	59.48	27.70	256.81G	0.01	1160	5.84	1511	
city-5	647749	584	0.35	100	56.70G	4.79	6.79	3.34	8.54	11.67	56.70G	0.00	1086	5.29	1458	
city-5	647749	584	0.35	150	12.56G	4.65	5.45	3.31	11.50	2.45	12.56G	0.00	990	3.97	1327	
city-6	793157	596	0.56	50	275.43G	6.00	10.17	3.07	68.69	37.11	275.43G	0.00	1340	7.51	1579	
city-6	793157	596	0.56	100	61.62G	5.72	9.56	4.20	9.06	6.28	61.61G	0.02	1034	6.46	1536	
city-6	793157	596	0.56	150	13.68G	5.45	7.54	4.08	11.19	3.54	13.68G	0.00	1139	5.28	1351	
city-7	1184157	516	0.69	50	222.25G	8.49	16.63	5.11	141.02	37.85	222.24G	0.00	1188	10.62	1795	
city-7	1184157	516	0.69	100	33.08G	8.26	12.47	6.67	19.43	8.46	33.08G	0.00	974	8.71	1289	
city-7	1184157	516	0.69	150	5.83G	9.11	9.37	7.52	17.35	5.16	5.83G	0.00	1009	5.38	1144	
city-8	2167197	531	1.29	50	224.64G	17.45	39.32	10.67	324.29	96.00	224.63G	0.01	1139	19.92	1444	
city-8	2167197	531	1.29	100	34.16G	15.90	30.90	14.11	36.53	19.14	34.16G	0.00	877	16.09	1338	
city-8	2167197	531	1.29	150	6.07G	16.77	19.64	14.26	38.83	6.90	6.07G	0.01	1089	10.31	1153	
city-9	4133801	573	2.12	50	266.62G	36.82	79.63	21.65	769.48	179.98	266.62G	0.00	946	37.54	1856	
city-9	4133801	573	2.12	100	47.22G	30.99	70.09	40.43	77.40	34.10	47.22G	0.00	1146	37.60	1522	
city-9	4133801	573	2.12	150	9.17G	35.67	51.02	26.61	86.40	16.36	9.17G	0.00	944	22.33	1415	
city-10	2693701	635	1.57	50	357.98G	23.72	52.68	15.36	397.38	92.24	357.98G	0.00	1055	25.06	2254	
city-10	2693701	635	1.57	100	82.41G	21.47	44.01	20.10	41.58	32.61	82.41G	0.00	999	23.64	1689	
city-10	2693701	635	1.57	150	17.91G	23.16	35.63	17.79	58.50	12.29	17.91G	0.00	1079	18.45	1507	
city-11	4921925	652	2.93	50	361.60G	49.56	107.11	41.68	890.04	248.35	361.59G	0.00	1213	49.05	1828	
city-11	4921925	652	2.93	100	84.01G	43.82	104.51	38.57	91.72	95.68	84.00G	0.00	1081	45.01	1835	
city-11	4921925	652	2.93	150	18.17G	49.73	69.82	43.10	107.06	21.37	18.17G	0.01	1024	34.26	1627	
occupation-1	875995	357	0.53	50	214.30G	5.89	7.97	3.89	87.73	6.65	214.26G	0.02	1074	7.42	943	
occupation-1	875995	357	0.53	100	26.12G	5.58	4.90	3.44	10.17	2.36	26.12G	0.00	1088	5.20	719	
occupation-1	875995	357	0.53	150	5.15G	4.80	3.47	2.63	11.29	2.15	5.15G	0.00	890	2.51	566	
occupation-2	3300827	382	2.01	50	231.22G	28.21	45.42	16.72	600.76	25.45	231.15G	0.03	1032	30.70	2064	
occupation-2	3300827	382	2.01	100	30.49G	23.21	23.73	15.40	63.59	10.57	30.49G	0.01	1089	22.35	835	
occupation-2	3300827	382	2.01	150	6.66G	22.75	16.46	13.05	72.36	9.74	6.66G	0.00	930	9.98	646	
city-uni-1	1184157	516	0.67	50	1828.53G	8.17	16.56	5.44	154.12	32.18	1828.53G	0.00	1009	10.40	2838	
city-uni-1	1184157	516	0.67	100	269.73G	9.73	14.57	7.30	17.40	12.56	269.73G	0.00	897	8.66	1424	
city-uni-1	1184157	516	0.67	150	47.47G	9.04	9.01	6.12	20.75	3.67	47.46G	0.00	1015	5.48	1165	
city-uni-2	227909	549	0.13	50	1920.07G	1.74	2.41	0.97	16.95	23.20	1919.84G	0.01	1111	1.88	1530	
city-uni-2	227909	549	0.13	100	416.04G	1.48	2.04	0.75	3.22	2.74	416.04G	0.00	989	1.71	1310	
city-uni-2	227909	549	0.13	150	92.18G	1.37	1.79	1.00	3.31	1.03	92.17G	0.01	1020	1.38	1215	
city-uni-3	647749	584	0.37	50	2109.61G	5.51	8.06	2.49	62.21	20.99	2109.61G	0.00	1176	5.53	1594	
city-uni-3	647749	584	0.37	100	469.84G	5.69	7.13	3.23	9.65	4.79	469.84G	0.00	980	5.25	1550	
city-uni-3	647749	584	0.37	150	104.77G	3.95	5.92	3.16	11.44	2.65	104.76G	0.01	1022	4.11	1393	
occupation_uni	875995	357	0.52	50	1831.35G	7.06	7.66	3.88	87.83	5.97	1831.35G	0.00	1049	7.48	983	
occupation_uni	875995	357	0.52	100	229.59G	5.49	4.75	3.32	11.95	2.44	229.59G	0.00	1035	5.26	721	
occupation_uni	875995	357	0.52	150	42.41G	4.70	3.60	2.76	14.05	2.05	42.41G	0.00	1095	2.61	604	
Avg			0.76			10.49	17.56	7.95	82.85	21.72		0.00	1018	9.73	1262	

Table 4.4: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type A.

Instances	Size and Generation			Side Cons.			Cplex						Dual Ascent		
	m	n	T_{Gen}	α	z_{LP}	$Time P$	$Time D$	$Time N$	$Time B$	$Time S$	z_{LB}	Gap_{LP}	$Iter$	$Time$	$ C $
city-12	7804077	598	1.24	100.00G	120.87	56.24	67.74	69.66	1364.26	51.89	120.85	0.01	686	18.76	123863
city-12	7804077	598	1.24	10.00G	210.70	27.75	19.16	24.98	431.13	23.04	210.70	0.00	755	10.99	10352
city-12	7804077	598	1.24	100.00M	418.14	12.91	12.87	12.85	111.87	15.15	418.11	0.01	815	6.30	938
city-13	9116229	668	1.58	100.00G	138.34	71.42	78.55	83.32	1480.30	64.10	138.33	0.00	880	30.78	202568
city-13	9116229	668	1.58	10.00G	242.09	30.20	21.97	27.56	522.72	27.14	242.09	0.00	804	13.20	11164
city-13	9116229	668	1.58	100.00M	474.01	14.97	14.91	14.78	13.55	17.08	473.99	0.00	785	7.16	1115
city-14	12883061	536	2.01	100.00G	103.76	88.01	128.21	110.85	2805.24	110.45	103.75	0.00	765	37.77	250015
city-14	12883061	536	2.01	10.00G	180.12	43.72	36.61	53.17	842.61	35.83	180.12	0.00	726	16.92	20298
city-14	12883061	536	2.01	100.00M	364.51	22.61	22.44	22.34	20.55	25.91	364.40	0.03	683	8.98	880
city-15	15144109	612	2.46	100.00G	123.54	126.37	164.65	179.05	3121.75	114.32	123.53	0.00	629	33.77	222015
city-15	15144109	612	2.46	10.00G	213.99	51.56	38.09	63.67	1025.41	43.53	213.98	0.01	845	23.50	14821
city-15	15144109	612	2.46	100.00M	426.80	25.94	26.03	25.69	23.52	29.60	426.69	0.03	591	9.44	994
Avg			1.82		52.60	47.64	52.60	57.33	971.91	46.50		0.01	747	18.13	71585

Table 4.5: Dual Ascent procedure is compared with CPLEX LP solvers: Problem Type B.

Instances	Size and Generation			Side Cons.	Cplex								Dual Ascent			
	m	n	T_{Gen}		z_{LP}	$Time P$	$Time D$	$Time N$	$Time B$	$Time S$	z_{LB}	Gap LP	Iter	Time	$ C $	
city-12	7804077	598	1.24	150	44.78G	75.05	110.80	61.91	205.57	31.77	44.75G	0.05	998	50.28	1430	
city-12	7804077	598	1.24	100	205.76G	77.50	171.48	101.74	199.84	53.76	205.75G	0.00	1024	66.46	3886	
city-12	7804077	598	1.24	50	1097.79G	66.40	204.13	78.17	2171.16	238.95	1097.75G	0.00	1222	76.08	2114	
city-13	9116229	668	1.58	150	74.43G	109.05	185.79	74.80	270.50	40.02	74.42G	0.00	1184	69.24	1748	
city-13	9116229	668	1.58	100	323.05G	112.40	219.34	130.65	186.98	76.74	323.05G	0.00	1134	84.35	1918	
city-13	9116229	668	1.58	50	1437.34G	83.19	243.98	55.89	2433.99	334.06	1437.20G	0.01	1132	92.02	1911	
city-14	12883061	536	2.01	150	22.87G	129.44	158.41	95.21	346.52	50.65	22.87G	0.00	1071	62.24	1239	
city-14	12883061	536	2.01	100	115.08G	137.20	207.50	139.87	312.45	89.97	115.07G	0.01	1020	103.44	2219	
city-14	12883061	536	2.01	50	774.04G	119.98	406.66	163.27	1174.22	362.86	773.96G	0.01	1208	134.37	1814	
city-15	15144109	612	2.46	150	46.98G	154.49	236.06	154.76	440.68	62.71	46.98G	0.01	1102	103.02	1535	
city-15	15144109	612	2.46	100	217.84G	153.01	430.12	207.46	39.41	93.24	217.83G	0.01	1108	137.93	1699	
city-15	15144109	612	2.46	50	1112.72G	147.34	515.81	242.37	39.58	597.88	1112.68G	0.00	1236	153.85	2586	
Avg			1.82			113.75	257.51	125.51	651.74	169.38		0.01	1120	94.44	2008	

Setting a smaller $MaxIter_0$ and a most aggressive value for β_0 we can obtain a faster convergence of the dual ascent with a smaller worsening of the lower bound provided. However, the convergence is more erratic and the quality of the solutions generated by the embedded Lagrangian heuristic is a little worse. Since we are mainly interested in the heuristic or optimal solution of the problem, we prefer a slower dual ascent which enhances the quality of the solutions generated by the Lagrangian heuristic.

In Tables 4.2, 4.3, 4.4, and 4.5 column $|\mathbb{C}'|$ shows that the size of the subset of columns \mathbb{C}' evaluated in the dual ascent procedure is always very small with respect to the total number of columns n of the original instances.

Notice that the computing time required for generating the full set of column \mathbb{C} is usually very small and it never dominates the time for solving the instances.

4.4.5.2 Greedy and Lagrangian Heuristics

In Tables 4.6, 4.7, 4.8, and 4.9 we compare the greedy and the Lagrangian heuristics, described in Sections 4.3 and 4.4.3. The results of the Lagrangian heuristic include the dual ascent procedure which embeds it.

For each test instance we report the right-hand side of the side constraint α , which is the maximum loss of precision for problem of Type A and the maximum size of the data for problem of Type B, and for both heuristics we report the best upper bound provided z_{UB} , the percentage gap from the value of the optimal integer solution $Gap = 100 \times \frac{z_{UB} - z_{Opt}}{z_{Opt}}$, and the computing time *Time*.

For the dual ascent with the Lagrangian heuristic we also report the best lower bound z_{LB} corresponding to the best feasible dual solution generated, the number of iterations *Iter*, and the size of \mathbb{C}' .

The computing time of the greedy heuristic is negligible but the percentage gap from the value of the optimal solution is on average 1.28% and 3.74% for the problems of type A and B reported in Tables 4.6 and 4.7, respectively, and is on average 0.57% and 5.09% for the hard instances of type A and B reported in Tables 4.8 and 4.9. The maximum gap is the 25% and often is greater than the 5%.

The Lagrangian heuristic is more time consuming but the percentage gap from the value of the optimal solution is much smaller. For problems of type A in Tables 4.6 and 4.8, only for three instances the optimal solution value is not found, the average gap is about 0.04% and the maximum gap is 1.47%. The quality of the upper bound is worse for problem of type B, where the average gap is 1.05% and 2.82% for instances in Tables 4.7 and 4.9, respectively, and the maximum gap is 8.66%, but it is still much better than the greedy heuristic and several instances are solved to optimality.

4.4.5.3 Exact Method

In order to evaluate the effectiveness of the Lagrangian heuristic, described in Section 4.4.3, and of the exact method, described in Section 4.4.4, in Tables 4.6, 4.7, 4.8, and 4.9 we compare them with the CPLEX MIP solver.

For the proposed exact method in our computational results we set $MaxIter = 10$, $\Delta_0 = 1000$, $\mu_1 = 0.001$, and $\mu_2 = 10$.

For CPLEX MIP solver we report the integer optimal solution value z_{Opt} and the computing time $Time$. We report the symbol “–” in column z_{Opt} when CPLEX MIP solver fails because of an “*out of memory*”. For these instances column $Time$ reports the computing time spent to generate the error, but they are not considered when we evaluate the average computing time.

For the dual ascent embedding the Lagrangian heuristic we report the best lower and upper bounds found z_{LB} and z_{UB} , the computing time $Time$, and the size of the subset of columns \mathbb{C}' at the end of the execution.

For the exact method we report the integer optimal solution value z_{Opt} , the size of the subset of columns \mathbb{C}' considered in the integer reduced problem P' , the number of iterations $Iter$, the computing time $Time$ for solving the reduced problem P' (even more than one time if $Iter > 1$), and the overall computing time T_{Tot} which also includes the computing time of the dual ascent procedure and of the embedded Lagrangian heuristic. When the problem is solved by the dual ascent we report $|\mathbb{C}'| = 0$.

The exact method performs very well for problem of Type A, where it needs to solve the integer reduced problem P' only for four instances using a very small subset of columns \mathbb{C}' . Only for one instance the exact method requires two iterations. The proposed exact method is on average about seven times faster than the CPLEX MIP solver and for two hard instances CPLEX MIP solver runs out of memory.

Table 4.6: Comparison among CPLEX MIP solver, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type A.

Instances	Side Con.			Greedy			Cplex			Dual Ascent + Lagr. Heu.						Exact Method		
	α	z_{UB}	Gap	Time	z_{opt}	Time	z_{LB}	z_{UB}	Gap	Iter	Time	$ C^* $	z_{opt}	$ C^* $	Iter	Time	T_{Tot}	
state	10.00G	30	0.00	0.00	30	0.00	29.60	30	0.00	303	0.00	35	30	0	1	0.00	0.00	
state	1000.00M	32	0.00	0.00	32	0.01	32.00	32	0.00	202	0.01	32	32	0	1	0.00	0.01	
prod.department	1000.00M	5	25.00	0.00	4	0.33	3.51	4	0.00	783	0.06	19	4	0	1	0.00	0.06	
prod.department	100.00M	10	0.00	0.00	10	0.10	9.31	10	0.00	454	0.01	17	10	0	1	0.00	0.01	
prod.brand	10.00M	37	8.82	0.01	34	1.63	33.00	34	0.00	774	0.34	175	34	245	2	0.00	0.51	
prod.brand	1000000.00	169	0.60	0.01	168	0.64	167.68	169	0.60	1025	0.24	1180	168	533	1	0.08	0.52	
city-1	100.00G	75	2.74	0.01	73	1.00	72.01	73	0.00	688	0.21	1013	73	0	1	0.00	0.21	
city-1	10.00G	141	0.00	0.01	141	0.00	140.34	141	0.00	603	0.21	2547	141	0	1	0.00	0.21	
city-1	100.00M	307	0.00	0.00	307	0.15	306.21	307	0.00	708	0.11	807	307	0	1	0.00	0.11	
city-2	100.00G	78	2.63	0.01	76	1.94	76.49	76	0.00	746	0.71	1789	76	0	1	0.00	0.71	
city-2	10.00G	150	0.67	0.01	149	1.41	148.69	149	0.00	759	0.55	3689	149	0	1	0.00	0.55	
city-2	100.00M	325	0.00	0.01	325	0.24	324.52	325	0.00	660	0.16	920	325	0	1	0.00	0.16	
city-3	100.00G	80	2.56	0.01	78	4.25	77.90	78	0.00	767	1.47	7820	78	0	1	0.00	1.47	
city-3	10.00G	155	0.65	0.01	154	3.04	153.97	154	0.00	746	1.16	5392	154	0	1	0.00	1.16	
city-3	100.00M	343	0.29	0.00	342	0.40	341.54	342	0.00	615	0.25	996	342	0	1	0.00	0.25	
city-4	100.00G	84	1.20	0.01	83	121.04	82.43	83	0.00	670	3.18	12144	83	0	1	0.00	3.18	
city-4	10.00G	159	0.00	0.01	159	5.84	158.09	159	0.00	746	2.14	5337	159	0	1	0.00	2.14	
city-4	100.00M	354	0.00	0.01	354	0.73	353.61	354	0.00	615	0.51	1226	354	0	1	0.00	0.51	
city-5	100.00G	83	1.22	0.01	82	16.60	81.22	82	0.00	669	5.13	15801	82	0	1	0.00	5.13	
city-5	10.00G	160	0.63	0.01	159	10.75	158.43	159	0.00	645	3.48	7534	159	0	1	0.00	3.48	
city-5	100.00M	356	0.00	0.01	356	1.14	355.62	356	0.00	638	1.02	1281	356	0	1	0.00	1.02	
city-6	100.00G	86	1.18	0.01	85	16.98	84.17	85	0.00	567	6.29	28823	85	0	1	0.00	6.29	
city-6	10.00G	163	0.62	0.01	162	12.25	161.78	162	0.00	652	4.36	8636	162	0	1	0.00	4.36	
city-6	100.00M	366	0.27	0.01	365	1.36	364.93	365	0.00	620	1.21	1316	365	0	1	0.00	1.21	
city-7	100.00G	73	1.39	0.01	72	29.49	71.70	72	0.00	574	7.80	40178	72	0	1	0.00	7.80	
city-7	10.00G	133	0.00	0.01	133	15.24	132.33	133	0.00	745	7.00	11153	133	0	1	0.00	7.00	
city-7	100.00M	315	0.32	0.01	314	2.00	313.59	314	0.00	588	1.54	1072	314	0	1	0.00	1.54	
city-8	100.00G	73	0.00	0.01	73	110.95	72.29	73	0.00	670	15.20	28582	73	0	1	0.00	15.20	
city-8	10.00G	135	0.75	0.01	134	56.08	133.61	134	0.00	784	12.21	15619	134	0	1	0.00	12.21	
city-8	100.00M	320	0.31	0.01	319	3.73	318.93	319	0.00	592	3.21	1401	319	0	1	0.00	3.21	
city-9	100.00G	80	1.27	0.01	79	175.55	78.81	79	0.00	688	33.46	29060	79	0	1	0.00	33.46	
city-9	10.00G	148	0.68	0.01	147	145.11	146.88	147	0.00	749	28.42	22182	147	0	1	0.00	28.42	
city-9	100.00M	349	0.29	0.01	348	7.40	347.40	348	0.00	601	5.98	1422	348	0	1	0.00	5.98	
city-10	100.00G	96	2.13	0.01	94	132.42	93.37	94	0.00	722	24.11	58052	94	0	1	0.00	24.11	
city-10	10.00G	174	0.58	0.01	173	67.16	172.89	173	0.00	764	16.25	13882	173	0	1	0.00	16.25	
city-10	100.00M	401	0.25	0.01	400	4.77	399.49	400	0.00	604	3.77	1520	400	0	1	0.00	3.77	
city-11	100.00G	96	2.13	0.02	94	275.86	93.88	94	0.00	694	40.69	39078	94	0	1	0.00	40.69	
city-11	10.00G	175	0.00	0.02	175	226.66	174.11	175	0.00	824	31.45	18963	175	0	1	0.00	31.45	
city-11	100.00M	407	0.25	0.01	406	9.18	405.83	406	0.00	574	6.88	1567	406	0	1	0.00	6.88	
occupation-1	100.00G	68	1.49	0.00	67	47.40	66.80	67	0.00	601	5.81	3043	67	0	1	0.00	5.81	
occupation-1	10.00G	132	1.54	0.00	130	10.56	129.29	130	0.00	669	2.88	1120	130	0	1	0.00	2.88	
occupation-1	100.00M	260	0.00	0.00	260	1.27	259.06	260	0.00	495	0.83	469	260	0	1	0.00	0.83	
occupation-2	100.00G	71	1.43	0.01	70	345.62	69.52	70	0.00	753	27.18	17176	70	0	1	0.00	27.18	
occupation-2	10.00G	138	1.47	0.01	136	44.30	135.62	136	0.00	612	10.67	1818	136	0	1	0.00	10.67	
occupation-2	100.00M	280	0.00	0.01	280	5.28	279.60	280	0.00	593	3.74	523	280	0	1	0.00	3.74	
city.uni-1	1000.00G	69	1.47	0.01	68	143.43	67.71	69	1.47	672	8.34	12499	68	310	1	0.22	8.56	
city.uni-1	100.00G	127	0.00	0.01	127	24.93	126.70	127	0.00	593	5.46	10400	127	0	1	0.00	5.46	
city.uni-1	1000.00M	304	0.00	0.01	304	2.59	303.56	304	0.00	644	1.93	1071	304	0	1	0.00	1.93	
city.uni-2	1000.00G	75	2.74	0.01	73	6.58	72.05	73	0.00	715	1.36	7947	73	0	1	0.00	1.36	
city.uni-2	10.00G	149	0.68	0.01	148	5.94	147.18	148	0.00	729	1.35	5019	148	0	1	0.00	1.35	
city.uni-2	1000.00M	338	0.00	0.00	338	0.68	337.34	338	0.00	723	0.32	1131	338	0	1	0.00	0.32	
city.uni-3	1000.00G	77	1.32	0.01	76	300.23	74.98	76	0.00	665	4.64	31434	76	250	1	0.10	4.74	
city.uni-3	100.00G	154	1.32	0.01	152	16.54	151.69	152	0.00	723	3.64	9696	152	0	1	0.00	3.64	
city.uni-3	1000.00M	353	0.00	0.01	353	1.68	352.75	353	0.00	667	0.95	1663	353	0	1	0.00	0.95	
occupation.uni	1000.00G	65	0.00	0.00	65	55.71	64.23	65	0.00	666	6.79	24496	65	0	1	0.00	6.79	
occupation.uni	100.00G	125	0.00	0.01	125	11.20	124.10	125	0.00	738	3.87	1462	125	0	1	0.00	3.87	
occupation.uni	1000.00M	254	0.00	0.00	254	1.28	253.41	254	0.00	524	0.79	452	254	0	1	0.00	0.79	
Avg			1.28	0.01		43.67			0.04	660	6.34	9188			0.01	6.35		

Table 4.8: Comparison among CPLEX MIP solver, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type A.

Instances	Side Con.			Greedy			Cplex			Dual Ascent + Lagr. Heu.			Exact Method				
	α	z_{UB}	Gap	T_{ime}	z_{opt}	T_{ime}	z_{LB}	z_{UB}	Gap	Iter	T_{ime}	$ \mathcal{C} $	z_{opt}	$ \mathcal{C} $	Iter	T_{ime}	T_{Tot}
city-12	100.00G	123	1.65	0.01	121	596.90	120.85	122	0.83	686	51.19	1238663	121	1000	1	1.47	52.66
city-12	10.00G	213	0.95	0.01	211	345.48	210.70	211	0.00	755	33.70	10352	211	0	1	0.00	33.70
city-12	100.00M	419	0.00	0.01	419	13.88	418.11	419	0.00	815	13.22	938	419	0	1	0.00	13.22
city-13	100.00G	141	1.44	0.01	139	704.35	138.33	139	0.00	880	81.77	202568	139	0	1	0.00	81.77
city-13	10.00G	244	0.41	0.01	243	483.84	242.09	243	0.00	804	42.66	11164	243	0	1	0.00	42.66
city-13	100.00M	475	0.00	0.01	475	15.89	473.99	475	0.00	785	15.97	1115	475	805	1	1.81	17.78
city-14	100.00G	105	0.96	0.01	-	774.02	103.75	104	0.00	765	95.54	250015	104	0	1	0.00	95.54
city-14	10.00G	182	0.55	0.01	181	283.83	180.12	181	0.00	726	53.91	20298	181	0	1	0.00	53.91
city-14	100.00M	365	0.00	0.00	365	23.66	364.40	365	0.00	683	20.52	880	365	0	1	0.00	20.52
city-15	100.00G	124	0.00	0.01	-	155.23	123.53	124	0.00	629	100.11	222015	124	0	1	0.00	100.11
city-15	10.00G	216	0.93	0.01	214	328.44	213.98	214	0.00	845	80.11	14821	214	0	1	0.00	80.11
city-15	100.00M	427	0.00	0.01	427	27.78	426.69	427	0.00	591	21.63	994	427	0	1	0.00	21.63
Avg			0.57	0.01		282.41			0.07	747	50.86	71585				0.27	51.13

Table 4.9: Comparison among CPLEX, Dual Ascent with Lagrangian Heuristic, and Exact method: Problem Type B.

Instances	Side Con.	Greedy			Cplex			Dual Ascent + Lagr. Heu.				Exact Method					
		α	z_{UB}	Gap	Time	z_{opt}	Time	z_{LB}	z_{UB}	Gap	Iter	Time	$ C $	z_{opt}	$ C $	Iter	Time
city-12	150	46.36G	3.54	0.01	327.70	44.78G	44.75G	46.36G	3.54	998	69.94	1430	44.78G	233	1	1.39	71.33
city-12	100	217.43G	5.67	0.01	385.38	205.76G	205.75G	205.76G	0.00	1024	84.06	3886	205.76G	157	1	1.47	85.53
city-12	50	1194.95G	8.61	0.01	454.15	—	1097.75G	1194.95G	8.61	1222	106.83	2114	1100.19G	10000	2	4.80	111.63
city-13	150	77.20G	3.73	0.01	774.15	74.43G	74.42G	75.85G	1.92	1184	105.08	1748	74.43G	271	1	1.73	106.82
city-13	100	344.85G	6.75	0.01	864.26	323.05G	323.05G	323.05G	0.00	1134	113.88	1918	323.05G	193	1	1.75	115.63
city-13	50	1498.44G	4.25	0.01	868.12	1437.34G	1437.20G	1437.34G	0.00	1132	131.51	1911	1437.34G	116	1	1.86	133.37
city-14	150	23.17G	1.34	0.01	384.99	22.87G	22.87G	22.87G	0.00	1071	85.92	1239	22.87G	195	1	2.48	88.40
city-14	100	117.34G	1.96	0.01	552.27	115.07G	115.07G	115.10G	0.01	1020	135.70	2219	115.08G	125	1	2.60	138.29
city-14	50	836.71G	7.01	0.01	395.82	773.96G	773.96G	836.71G	7.01	1208	192.68	1814	781.88G	1000000	4	775.99	968.67
city-15	150	48.87G	4.03	0.01	160.35	46.98G	46.98G	48.87G	4.03	1102	165.77	1535	46.98G	236	1	3.06	168.84
city-15	100	229.83G	5.50	0.01	160.10	217.83G	217.91G	217.91G	0.03	1108	210.67	1699	217.84G	174	1	3.09	213.76
city-15	50	1211.59G	8.66	0.01	162.32	1112.68G	1112.68G	1211.59G	8.66	1236	231.87	2586	1115.07G	10000	2	9.10	240.97
Avg			5.09	0.01	593.84				2.82	1120	136.16	2008				67.44	203.60

Problem of Type B is more difficult to solve, but the exact method is on average about six times faster than the CPLEX MIP solver. As shown by Table 4.7, for many instances the exact method needs to solve the integer reduced problem P' . However, the size of the subset \mathbb{C}' is still small and the computing time for solving the reduced problem P' is very small. We need two iterations only for seven instances, three iterations for one instance, and four iterations for one hard instance. All the remaining instances are solved in one iterations. For only one hard instance of type B, the exact method requires about 16 minutes, but for the same instance CPLEX MIP solver fails. Overall, CPLEX MIP solver fails for five hard instances of type B.

4.4.6 Wrapping up the new Shrink Implementations

In this work we have proposed an integer linear programming model for solving the problem of implementing in the *best way* the OLAP shrink operator.

We have modelled the problem as a set partitioning problem with side constraint and we have considered two different approaches for finding the best implementation. In the first one (problem of Type A) we have supposed to minimize the size of the resulting data and the side constraint guarantees that the loss of precision does not exceed a given maximum value. Whereas, the second approach (problem of Type B) minimizes the loss of precision and the side constraint guarantees that the size of the resulting data do not exceed a given maximum value.

The proposed mathematical formulation is able to model both problem types. For switching from one type to the other it is sufficient to modify the coefficients of the objective function and of the side constraint, along with its right-hand side.

The first solution method considered is a greedy heuristic, which is very fast but often generates solution of unsatisfactory quality. Therefore, we have proposed a dual ascent which embeds a Lagrangian heuristic.

The dual ascent generates at each iteration a dual solution of the LP-relaxation of the problem, hopefully feasible. The dual ascent only considers a reduced subset of columns to solve the problem and uses the generated feasible dual solutions for adding columns to the *reduced problem*, using the pricing. The computing time allows an operational use of the procedure and the quality of the solution generated is of very good quality. For problem of Type A the dual ascent significantly outperforms general purpose LP solvers as CPLEX. It is able to generate a near optimal dual solution in a short computing time.

We have also proposed an exact method to use when the optimal solution is required. Adding a very small computing time to the dual ascent, the proposed exact procedure generates an optimal solution using a very small subset of the original columns. Therefore, the exact procedure has the potential for an operational use, while a general purpose solver, like CPLEX, is time consuming and, above all, if fails for some instances.

For the instances used in this work, the computing time for generating the clusters is almost negligible with respect to the time for solving the problem. Whereas, the use of a limited size subset of columns, in a column generation fashion, allows a huge reduction of the overall computing time without compromising the optimal solution of the problem. However, future developments can investigate an actual column generation process to embed in the proposed algorithms in order to solve much larger instances, where the complete generation of clusters may take time.

4.5 Multidimensional Shrink

A mono-dimensional version of the shrink operator has been proposed by [5]. In that work one shrink dimension is explicitly chosen by the user, and cube slices are fused together along that dimension until a user-specified precision/size trade-off is achieved. Though the mono-dimensional version has been shown to be quite effective and efficient in delivering compact visualizations of cubes, it suffers from two main drawbacks: (i) since the shrink dimension is fixed a priori, some possibly more effective directions for shrinking may be lost; and (ii) the approach is subject to the user’s discretion in choosing the shrink dimension.

		Year		
		2010	2011	2012
City	Miami	47	45	46
	Orlando	44	43	40
	Tampa	55	52	53
	Washington	47	45	51
	Richmond	43	46	49
	Arlington	—	47	52

Figure 4.7: A simple pivot table showing data by City and Year

In this work we propose a multidimensional generalization of the shrink operator, where facts are fused along multiple dimensions. Multidimensional shrink comes in two flavors: lazy and eager.

- In *lazy shrink*, facts can or cannot be fused together depending on dimension hierarchies. For instance, with reference to Figure 4.7, the facts for Miami and Washington cannot be fused together because the two cities belong to different states, while Miami can be fused with Orlando because they both are in Florida. Miami, Orlando, and Tampa can be all fused together, to obtain a single fact representing Florida. Finally, all six cities can be fused together to represent the whole South-Atlantic region (see also Figure 4.3).
- In *eager shrink* the constraint is stricter, and either all or none of the facts belonging to the same hierarchy group can be fused together. In the above example, if the facts for all the cities in a state are similar enough they are all fused into a single fact representing that state, otherwise they all must be left separate.

The two approaches proposed achieve effectiveness in different ways. Lazy shrink suffers from less constraints than eager shrink in creating groups, so it tends to deliver solutions with better approximation. On the other hand, the visualization of the results obtained by eager shrink is more compact, because each row of the pivot table is labeled with a single

member (e.g., FL) while with lazy shrink each row can be labeled with several members, unless some concise form of labeling is adopted. Since computing an optimal result for the shrink operator is unfeasible due to the exponential nature of the problem, for both lazy and eager shrink we propose a greedy algorithm that computes a sub-optimal solution.

The original contributions we propose in this thesis as compared to our previous work [5] are as follows:

- (i) The formalization of the shrink framework, including the computation of the shrink approximation, is generalized from the mono- to the multidimensional case.
- (ii) Two different forms of the h-compliance constraints are defined (lazy and eager).
- (iii) The size of the search space for computing both lazy and eager shrink is characterized, and greedy algorithms are proposed.
- (iv) The approach is evaluated in terms of efficiency and effectiveness, also in comparison to those achieved by traditional roll-up and mono-dimensional shrink.

Remarkably, though our approach has been devised to enable compact visualization of pivot tables resulting from users' queries, it can also be used to preprocess a cube for making its size compatible with computation-intensive applications such as data mining.

The outline for the contributions related to multidimensional shrink is as follows. In Section 4.5.1 we describe the shrink intuition and provide a formal framework to support it. The greedy algorithms for lazy and eager shrink are described in Section 4.5.2 and 4.5.3, respectively, and experimentally evaluated in Section 4.5.4. Finally, in Section 4.5.5 we draw the conclusions.

4.5.1 The Multidimensional Shrink Framework

Before moving to the formal framework for multidimensional shrink, note that we will use again the definitions presented in Section 2.3, which are the definitions of multidimensional schema (Definition 1), group-by (Definition 2), and cube (Definition 3).

As stated in Section 4.5 and shown in Figure 4.2, our approach can be applied to any cube at any group-by resulting from an OLAP query. However, to avoid useless formalization complexity and without loss of generality, we will assume that the shrink operator is always applied to a cube C at its finest group-by, $G = \langle l_1^1, \dots, l_1^n \rangle$.

First of all we recall that cube C can be represented as a set of facts:

$$C = \{ \langle d, C(d) \rangle, d \in G \wedge C(d) \neq NULL \}$$

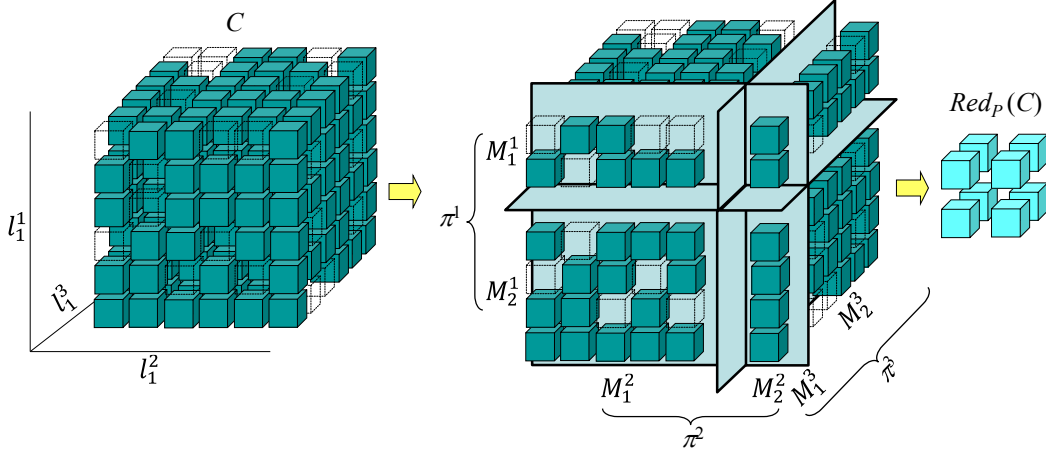


Figure 4.8: The intuition of multidimensional shrink: cube (left), dices (middle), and f-dices (right)

When the shrink operator is applied to C , the domain of each group-by level is (completely and disjointly) partitioned into a number of subsets; consistently with the classical OLAP terminology, each n -ple D of subsets taken from the n partitions is called a *dice* to emphasize that it determines a subset of facts of C . All the facts in each dice are then fused together into a single, approximate fact (which we call *f-dice*) by averaging their measure values. This means that an f-dice in the shrunk cube (which, from now on, we will call *reduction*) may refer not to a single coordinate, but rather to subsets of coordinates of the group-by. This process is exemplified in Figure 4.8 using a 3-dimensional cube C : the domains of the three dimensions are partitioned into two subsets each, determining eight dices overall; then, the facts in each dice are fused into the eight f-dices that constitute the reduction.

Definition 13 (Dice and F-Dice) Given cube C at group-by $G = \langle l_1^1, \dots, l_1^n \rangle$, a dice is an n -ple of sets of members of the group-by attributes: $D = \langle M^1, \dots, M^n \rangle$ (where $M^i \subseteq \text{Dom}(l_1^i)$ for $i = 1, \dots, n$). A dice determines a set of coordinates $M^1 \times \dots \times M^n$; so, with a slight abuse of formalism, given coordinate $d \in G$ we will write $d \in D$ to denote that $d \in M^1 \times \dots \times M^n$. The f-dice of C corresponding to D is the couple $\langle D, C^D \rangle$ where

$$C^D = \text{avg}\{C(d)\}, d \in D$$

(note that, in case $C(d) = \text{NULL}$ for all $d \in D$, it is also $C^D = \text{NULL}$)

Notice that this definition of dice is more flexible than the one previously given in Section 3.2.5.1. The difference is that this definition requires an n -ple of *sets* of members, while the other one requires an n -ple of *ranges*, thus a total order of the members is not required anymore.

Definition 13 shows that a dice has a two-fold nature: that of n sets of members, one for

each dimension, and that of a set of coordinates; the correspondence between these two natures is established through the Cartesian product. To ensure that this correspondence is preserved by the union operation (on which our agglomerative approach for computing shrink is based) we define union-compatibility as follows.

Definition 14 (Dice Union) *Dices D_1, \dots, D_k are union-compatible iff*

$$M^1 \times \dots \times M^n = \bigcup_{j=1}^k D_j$$

where $M^i = \bigcup_j M_j^i$ for $i = 1, \dots, n$. The union of k union-compatible dices D_1, \dots, D_k is defined by the n -ple $\langle M^1, \dots, M^n \rangle$.

Intuitively, two dices are union-compatible if their union made along each dimension corresponds to the union of their sets of coordinates, which means that the union operation is closed on dices. This will be clarified by the following example.

Example 17 *With reference to Figure 4.7, let*

$$\begin{aligned} D_1 &= \langle \{\text{Miami, Orlando}\}, \{2010, 2011\} \rangle \\ D_2 &= \langle \{\text{Tampa}\}, \{2012\} \rangle \end{aligned}$$

These dices are not union-compatible since the Cartesian product of $\bar{M}^1 = \{\text{Miami, Orlando, Tampa}\}$ and $\bar{M}^2 = \{2010, 2011, 2012\}$ includes coordinates, such as $\langle \text{Miami}, 2012 \rangle$ and $\langle \text{Tampa}, 2010 \rangle$, that are not in the union of the sets of coordinates of D_1 and D_2 . On the other hand, dices

$$\begin{aligned} D_1 &= \langle \{\text{Miami}\}, \{2010\} \rangle \\ D_2 &= \langle \{\text{Miami}\}, \{2011\} \rangle \\ D_3 &= \langle \{\text{Orlando}\}, \{2010\} \rangle \\ D_4 &= \langle \{\text{Orlando}\}, \{2011\} \rangle \end{aligned}$$

are union-compatible because the Cartesian product of $\bar{M}^1 = \{\text{Miami, Orlando}\}$ and $\bar{M}^2 = \{2010, 2011\}$ includes exactly the 4 coordinates in the union of the sets of coordinates of D_1, \dots, D_4 .

Theorem 2 (Sufficient Condition for Union-Compatibility) *Dices D_1, \dots, D_k are union-compatible if they are defined by identical sets of members along at least $n - 1$ dimensions.*

Proof: If all dices are identical the result is obvious. Otherwise, if the i -th is the one along which the dices differ, it is $D_j = \langle M^1, \dots, M_j^i, \dots, M^n \rangle$ for all j . But then it is $\bar{M}^1 \times \dots \times \bar{M}^n = M^1 \times \dots \times \bigcup_j M_j^i \times \dots \times M^n = \bigcup_j (M^1 \times \dots \times M_j^i \times \dots \times M^n) = \bigcup_j D_j$ because the Cartesian product is distributive with union. \square

A counter-example of dices that do not meet this sufficient condition yet are union-compatible is the one already given in Example 4.

The following definition will be used to concisely characterize sets of members based on the structure of the member tree:

Definition 15 (Representative Ancestor) *Given level l_j^i and a set of members $M \subseteq \text{Dom}(l_j^i)$, we denote with $LCA(M)$ the lowest common ancestor of M , i.e., the lowest single member in the member tree to which all members in M roll-up (conventionally, $LCA(\{m\}) = m$). Let l_k^i ($k \geq j$) be the level to which $LCA(M)$ belongs. The representative ancestor of M is a set defined as follows:*

$$RA(M) = \begin{cases} \{LCA(M)\} & , \text{ if } Drill_j(LCA(M)) = M \\ \{Roll^{k-1}(m), m \in M\} & , \text{ otherwise} \end{cases}$$

Intuitively, the representative ancestor of M is the set of children of $LCA(M)$ to which the members in M roll-up; in case M includes all the descendants of $LCA(M)$, the representative ancestor is $LCA(M)$ instead. It can be easily verified that $|RA(M)| \leq |M|$, and that for singletons it is $RA(\{m\}) = \{m\}$.

Example 18 *Consider again Figure 4.3, if $M = \{\text{Miami, Orlando}\}$ it is $LCA(M) = \text{FL}$ and $RA(M) = \{\text{Miami, Orlando}\}$, while if $M = \{\text{Miami, Washington}\}$ it is $LCA(M) = \text{South - Atlantic}$ and $RA(M) = \{\text{FL, VA}\}$. Finally, if $M = \{\text{Miami, Orlando, Tampa}\}$ it is $LCA(M) = RA(M) = \text{FL}$.*

To preserve the semantics of hierarchies in a reduction, dices must meet some further constraint called *hierarchy-compliance* (briefly, *h-compliance*). H-compliance comes in two flavors, to be used for lazy and eager shrink respectively. The idea is that, though formally speaking each dice refers to a set of coordinates of G , in case of lazy h-compliance it logically refers, for each hierarchy i , to the members of $RA(M^i)$, all of which belong to some level in L_i . The eager form of h-compliance is stricter, and it states that each dice must logically refer, for each hierarchy i , to exactly one member of some level in L_i .

Definition 16 (H-Compliance – Lazy Form) *Dice $D = \langle M^1, \dots, M^n \rangle$ is said to be h-compliant if, for each $i = 1, \dots, n$, it is*

$$Drill_1(RA(M^i)) = M^i \tag{4.27}$$

The group-by of D is the group-by $G' = \langle l_{k_1}^1, \dots, l_{k_n}^n \rangle$ such that $RA(M^i) \subseteq \text{Dom}(l_{k_i}^i)$ for each i .

Definition 17 (H-Compliance – Eager Form) *Dice $D = \langle M^1, \dots, M^n \rangle$ is said to be*

h-compliant if, for each $i = 1, \dots, n$, it is

$$Drill_1(LCA(M^i)) = M^i \quad (4.28)$$

The *group-by* of D is the *group-by* $G' = \langle l_{k_1}^1, \dots, l_{k_n}^n \rangle$ such that $LCA(M^i) \in Dom(l_{k_i}^i)$ for each i .

Essentially, Equation 4.27 states that, in a dice D with *group-by* G' , all members of M^i roll-up to the same member of $l_{k_{i+1}}^i$ (and therefore to the same member of all the subsequent levels in h_i) for all i 's. Besides, for each member m of $l_{k_i}^i$, either all or none of the members of $Drill_1(m)$ are included in M^i . On the other hand, Equation 4.28 states that there exists one member m in $l_{k_i}^i$ such that M^i includes all and only the members of $Drill_1(m)$. It is easy to check that eager h-compliance is a particular case of lazy h-compliance.

Example 19 Consider the following examples of M^i for the RESIDENCE hierarchy:

$$\begin{aligned} &\{\text{Miami, Orlando, Washington}\} \\ &\{\text{Miami, Orlando, Tampa, Washington}\} \{\text{Miami, Orlando}\} \\ &\quad \{\text{Miami}\} \\ &\quad \{\text{Miami, Orlando, Tampa}\} \end{aligned}$$

The first two sets are not h-compliant because some —but not all— cities of different states are put together. The third set is h-compliant in the lazy form (with *group-by* City) but not in the eager one (because one city in Florida is missing). The fourth and the fifth sets are h-compliant in both forms (with *group-by*'s City and State, respectively).

A reduction is now defined as the set of f-dices obtained from a partition of the *group-by* coordinates.

Definition 18 (MD-Partition) Given *group-by* $G = \langle l_1^1, \dots, l_1^n \rangle$, a multidimensional partition (or, briefly, md-partition) of G is a set of dices $P = \pi^1 \times \dots \times \pi^n$, where each π^i ($i = 1, \dots, n$) is a (total and disjoint) partition of $Dom(l_1^i)$.

It is easy to verify that an md-partition defined in this way (i.e., starting from monodimensional partitions of the single domains) induces a (total and disjoint) partition of the coordinates of G . Besides, it preserves the possibility of viewing the result of the shrink operator in the form of a “normal” pivot table. Indeed, if an md-partition were defined as *any* partition of the coordinates of G instead, the rows/columns of the resulting table could have different granularities, which would make the tabular representation hardly readable by a business user (for instance, this would be the case if, in Figure 4.7, the facts for the cities in Florida were fused together only for 2011 but not for 2010 and 2012).

Definition 19 (Reduction) Let C be a cube and let P be an md-partition of G . The

reduction of C induced by P , $Red_P(C)$, is the set of f -dices corresponding to the dices of P :

$$Red_P(C) = \{\langle D, C^D \rangle, D \in P \wedge C^D \neq NULL\}$$

The size of a reduction is the number of f -dices it includes. An md -partition and the reduction it induces are h -compliant if all their dices are h -compliant.

In the remainder of this work we will always deal with h -compliant reductions, that have two advantages over the others: (i) they fuse facts into f -dices in a way that does not violate the member groups implicitly defined by hierarchies, thus following the same behavior of the other OLAP operators, in particular of roll-up; and (ii) they enable a concise tabular representation of the reduction in terms of row and column labels. As concerns (ii), in particular, we must distinguish between the two flavors of h -compliance:

- In a lazy reduction, each f -dice $D = \langle M^1, \dots, M^n \rangle$ can be labeled, for each hierarchy h_i , not with all the elements in M^i , but with those in $RA(M^i)$. The size of $RA(M^i)$ depends on the hierarchy level to which its members belong, which in turn depends on whether *all* the children of a member are fused together or not (e.g., an f -dice can be concisely labeled with FL only if all cities in Florida are fused together). Should even $RA(M^i)$ be too large to be exhaustively displayed on the screen, we envision an approximate form of labeling based on a representative member of M^i chosen, for instance, as the centroid of M^i . The resulting label could be for instance something like “Miami and other 15 cities in FL” [5].
- In an eager reduction, the tighter constraint placed on partitions enables an even more concise representation, because it is always $RA(M^i) \equiv \{LCA(M^i)\}$.

Example 20 Figure 4.9 shows two possible h -compliant reductions, with size 6 and 4 respectively, of the cube shown in Figure 4.7. In (a) the md -partition is defined by $\pi_{RESIDENCE} = \{\{Miami, Orlando\}, \{Tampa\}, \{Washington, Richmond, Arlington\}\}$ and $\pi_{TIME} = \{\{2010, 2011\}, \{2012\}\}$. The reduction obtained is h -compliant in lazy but not in eager form, because only some cities in Florida and some years are fused together. The two f -dices in the bottom row have group-by $\langle State, Year \rangle$, so they can be labeled with the name of a state, while the other four f -dices have group-by $\langle City, Year \rangle$. In (b) the md -partition is defined by $\pi_{RESIDENCE} = \{\{Miami\}, \{Orlando\}, \{Tampa\}, \{Washington, Richmond, Arlington\}\}$ and $\pi_{TIME} = \{\{2012, 2011, 2012\}\}$ so the resulting reduction is h -compliant also in eager form.

4.5.1.1 The Shrink Approximation

Fusing the facts in a dice into a single f -dice of the reduction gives raise to an approximation. Like done by [83], to measure this approximation we use the *sum squared error*.

	2010, 2011	2012
Miami, Orlando	44.8 (8.8)	43 (18.0)
Tampa	53.5 (4.5)	53 (0.0)
VA	45.6 (11.2)	50.7 (4.7)

	AllTime
Miami	46 (2.0)
Orlando	42.3 (8.7)
Tampa	53.3 (4.7)
VA	47.5 (64.0)

(a)
(b)

Figure 4.9: Two h-compliant reductions of the same cube, one lazy (a) and one eager (b). SSE's of f-dices are in gray.

Definition 20 (Sum Squared Error) Given cube C at group-by $G = \langle l_1^1, \dots, l_1^n \rangle$, let D be a dice. The sum squared error (SSE) associated to f-dice $\langle D, C^D \rangle$ is

$$SSE(\langle D, C^D \rangle) = \sum_{d \in D} (C(d) - C^D)^2$$

(conventionally, $C(d) - C^D = 0$ if $C(d)$ is NULL). Let P be an (h-compliant) md-partition of G ; the SSE associated to reduction $Red_P(C)$ is

$$SSE(Red_P(C)) = \sum_{D \in P} SSE(\langle D, C^D \rangle)$$

Example 21 The SSE's associated to the each f-dice in the reduction of Figure 4.9 are shown in the same figure in gray. The overall SSE for Figure 4.9.a is 47.1, the one for Figure 4.9.b is 79.3.

The results in [5] showed that, in the mono-dimensional case, the SSE of a reduction built using an agglomerative approach can be incrementally computed. To generalize this result to the multidimensional case, we must verify that the SSE of the f-dice arising from the union of k dices can be computed starting from the SSE's of the single f-dices. Theorem 2 shows that, if the dices are union-compatible, this can be done.

Theorem 3 Let $Red_P(C)$ be a reduction of C , D_1, \dots, D_k be a set of disjoint union-compatible dices, and $D = \bigcup_{j=1}^k D_j$. Then it is

$$\begin{aligned} \Delta SSE(\{D_1, \dots, D_k\}) &= SSE(\langle D, C^D \rangle) - \sum_{j=1}^k SSE(\langle D_j, C^{D_j} \rangle) = \\ &= \sum_{j=1}^k \eta_j \cdot (C^{D_j})^2 - \eta \cdot (C^D)^2 \end{aligned} \quad (4.29)$$

where η and η_j are the number of coordinates of dices D and D_j respectively corresponding to non-null facts.

Proof: Let $\eta = \sum_j \eta_j$. By Definitions 11 and 5 we have that

$$\begin{aligned} SSE(\langle D, C^D \rangle) &= \sum_{d \in D} (C(d) - C^D)^2 = \\ &= \sum_{d \in D} C(d)^2 + \eta \cdot (C^D)^2 - 2C^D \sum_{d \in D} C(d) = \\ &= \sum_{d \in D} C(d)^2 - \eta \cdot (C^D)^2 \end{aligned}$$

Similarly, for each dice D_j :

$$SSE(\langle D_j, C^{D_j} \rangle) = \sum_{d \in D_j} C(d)^2 - \eta_j \cdot (C^{D_j})^2$$

Based on the two formulas above, we can rewrite the SSE as

$$\begin{aligned} SSE(\langle D, C^D \rangle) &= \sum_j \sum_{d \in D_j} C(d)^2 - \eta \cdot (C^D)^2 = \\ &= \sum_j \left(\sum_{d \in D_j} C(d)^2 - \eta_j \cdot (C^{D_j})^2 \right) + \sum_j \eta_j \cdot (C^{D_j})^2 - \eta \cdot (C^D)^2 = \\ &= \sum_j SSE(\langle D_j, C^{D_j} \rangle) + \sum_j \eta_j \cdot (C^{D_j})^2 - \eta \cdot (C^D)^2 \end{aligned}$$

which proves the assumption. □

Through some algebraic manipulations it is possible to rewrite ΔSSE as follows:

$$\Delta SSE(\{D_1, \dots, D_k\}) = \frac{1}{\eta} \sum_{i < j} \eta_i \eta_j (C^{D_i} - C^{D_j})^2$$

Though this equation is not the most efficient one to incrementally compute the SSE because it is quadratic in k , it clearly shows that ΔSSE is never negative. So, Theorem 2 confirms the intuition according to which all unions of dices lead to a non-decrease in the total SSE of a reduction.

4.5.1.2 The Reduction Problems

Given a cube C at group-by G , a combinatorial number of possible reductions can be operated on it, one for each way of partitioning the domains of levels in G while preserving h-compliance. Of course, the more the reduction process is pushed further, the lower the number of resulting f-dices; hence, the lower the size of the data returned to the user but the more the approximation introduced. So, the reduction process can be driven by a parameter α expressing the trade-off between size and precision. We envision two possible roles for this parameter: (i) the maximum tolerable number of f-dices in the reduction (determined for instance by the size of the display and/or by the network bandwidth of the device), and (ii) the maximum tolerable SSE of the reduction. Depending on the role of α , two alternative formulations can be given to the problem of finding a reduction of C :

Reduction Problem 1 Find a reduction that has the minimum size among those whose SSE is not larger than α_{SSE} .

Reduction Problem 2 Find a reduction that has the minimum SSE among those whose size is not larger than α_{size} .

4.5.1.3 Problem Search Space

In this section we define the size of the search space for both reduction problems. For each form of h-compliance —lazy and eager— we consider first its worst-case, and then a more general scenario.

Let C be an n -dimensional cube at a group-by $G = \langle l_1^1, \dots, l_1^n \rangle$, with $|Dom(l)| = \delta \forall l \in G$. In the lazy form of h-compliance, the worst-case takes place when each hierarchy has exactly two levels, so that each member $m \in l_1^i$ rolls-up to the same member $All \in l_2^i$ for each $i = 1, \dots, n$, meaning that no hierarchy constraints are actually posed. In this case, the total number of md-partitions is $(B_\delta)^n$, where B_δ is the δ -th Bell number and it counts the number of mono-dimensional partitions for each dimension. In a more general scenario, we consider the case where the members in each hierarchy are structured into a balanced tree with fixed fan-out $f > 0$ and height $z > 0$ (a tree with height $z = 0$ has only one node). In this case all hierarchies have exactly $z + 1$ levels, with $|Dom(l_j^i)| = f^{z-j+1} \forall j = 1, \dots, z + 1$, and the total number of md-partitions is given by

$$S_{lazy}(n, z, f) = Q_{lazy}(z, f)^n$$

where $Q_{lazy}(z, f)$ is the number of mono-dimensional partitions for each dimension, and it is recursively computed as

$$Q_{lazy}(z, f) = \begin{cases} Q_{lazy}(z-1, f)^f + \sum_{i=0}^{f-2} \binom{f}{i} Q_{lazy}(z-1, f)^i \cdot \hat{P}(f-i) & , \text{ if } z > 1 \\ B_f & , \text{ if } z = 1 \end{cases} \quad (4.30)$$

$\hat{P}(k)$ is the number of partitions of k elements without singletons and is defined as

$$\hat{P}(k+1) = \sum_{i=0}^{k-1} (-1)^i B_{k-i}$$

The recursive part of Equation 4.30 can be conceptually split into two terms: $Q_{lazy}(z-1, f)^f$ and the summation from $i = 0$ to $f - 2$. The former counts the number of combinations of the partitions that can be made on the children of each single member of $Dom(l_z^i)$, while the latter counts the number of partitions where at least two of such members are fused together. Once the cardinality of the mono-dimensional partitions for each dimension is known, the number of md-partitions is easily calculated as the number of choices of n

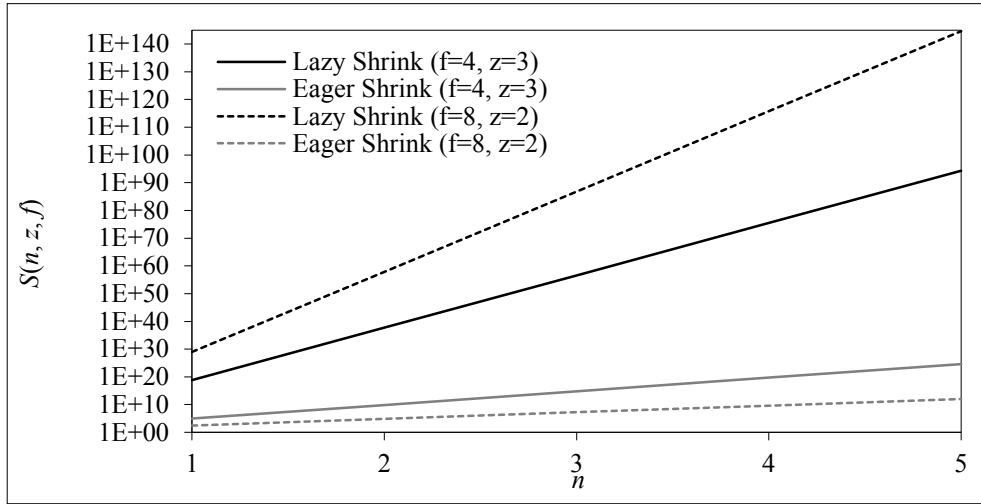


Figure 4.10: Search Space Size for lazy and eager shrink for different cube dimensionalities and hierarchy fan-outs with constant domain cardinality $|Dom(l_1^i)| = 64$ for all i 's

partitions, each belonging to a different dimension.

Considering again hierarchies structured as described above, in the eager form of h-compliance the worst-case takes place when each hierarchy has a fixed fan-out $f = 2$. Interestingly, this is quite different from the case of lazy h-compliance, where the smaller the fan-out, the fewer the partitions. Here, the function to count the exact number of md-partitions is

$$S_{eager}(n, z, f) = Q_{eager}(z, f)^n$$

$$Q_{eager}(z, f) = \begin{cases} Q_{eager}(z-1, f)^f + 1 & , \text{ if } z > 1 \\ 2 & , \text{ if } z = 1 \end{cases} \quad (4.31)$$

The reasoning is similar to the previous one, though Equation 4.31 is much simpler than Equation 4.30; indeed, the second term in the recursive part is 1 since we can either aggregate all the members, or none of them.

Figure 4.10 shows how $S_{lazy/eager}(n, z, f)$ changes with n , z , and f when the domain cardinality of each group-by level is fixed to 64. Note that, consistently with Equations (4) and (5), when f increases the search space grows larger for lazy shrink, while it gets smaller for eager shrink.

4.5.1.4 A Heuristic Approach

Since a reduction is determined starting from an md-partition of G , it appears that both reduction problems can be modeled as clustering problems with constraints, where the objects to be clustered are coordinates and each cluster corresponds to a dice. More

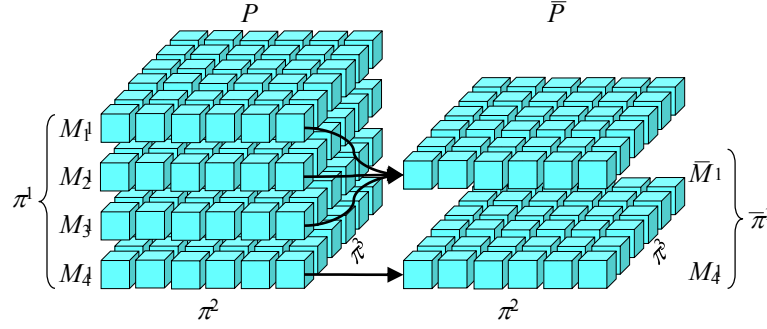


Figure 4.11: Agglomeration

precisely, since an md-partition is induced by mono-dimensional partitions (Definition 9), finding a reduction means finding n clusterings of members, one for each group-by level. Unfortunately, to the best of our knowledge, no clustering algorithm in the literature deals with constraints like those posed by h-compliance —except *cannot-link* [107], that cannot be used here since the h-compliance constraint depends on the composition of clusters. The approaches we will propose in the following sections are sub-optimal and belong to the class of *agglomerative algorithms for hierarchical clustering* [108], where each element initially stands in its own cluster, then pairs of clusters are iteratively merged; at each step, the decision of which pair of clusters will be merged is usually taken in a greedy (i.e., locally optimal) manner.

In our context, merging two clusters means unifying two or more dices in the current md-partition. This requires that (i) the dices to be unified are union-compatible, and (ii) that the result is still an md-partition. To meet both requirements we unify dices in slice-shaped batches, as formalized by the following definition.

Definition 21 (Agglomeration) Let $P = \pi^1 \times \dots \times \pi^n$ be an md-partition of group-by G , $N \subseteq \pi^i$ be a set of sets of members of $\text{Dom}(l_1^i)$, and $\bar{M}^i = \bigcup_{M_j^i \in N} M_j^i$. The agglomeration of P on \bar{M}^i , denoted $\text{Aggl}_{\bar{M}^i}^i(P)$, is the md-partition $\bar{P} = \pi^1 \times \dots \times \bar{\pi}^i \times \dots \times \pi^n$, where $\bar{\pi}^i = (\pi^i \setminus N) \cup \bar{M}^i$.

To clarify agglomeration, consider the md-partition $P = \pi^1 \times \pi^2 \times \pi^3$ in Figure 4.11, where $\pi^1 = \{M_1^1, \dots, M_4^1\}$, and let $N = \{M_1^1, M_2^1, M_3^1\}$. Agglomerating P on N means unifying $|\pi_2| \times |\pi_3|$ triplets of dices to obtain $|\pi_2| \times |\pi_3|$ new dices, all corresponding to the set of members \bar{M}^1 . The result of agglomeration is still an md-partition (Definition 9), and only entails unions between union-compatible dices that differ along one dimension (Theorem 1). The SSE variation of an agglomeration, denoted $\Delta SSE_{\bar{M}^i}(P)$, is the sum of the ΔSSE 's for all the dice unions entailed.

Using agglomeration defined as above in our greedy algorithms means unifying, at each iteration, member sets of a single dimension. In principle, we could provide a more general definition of agglomeration where member sets are unified along multiple dimensions at the

Algorithm 5 Heuristic Lazy MD-Shrink

Require: A cube C at a group-by $G = \langle l_1^1, \dots, l_1^n \rangle, \alpha_{SSE}$
Ensure: An h-compliant md-partition P of G

```

1: for all  $j = 1, \dots, n$  do ▷ For each dimension build a partition of singletons
2:    $\pi^j \leftarrow \{ \{m\}, m \in \text{Dom}(l_1^j) \}$ 
3:  $P \leftarrow \pi^1 \times \dots \times \pi^n$  ▷ Create the initial md-partition of singletons
4: while  $|P| > 1$  do ▷ Agglomerating is still possible
5:   Find  $i, 1 \leq i \leq n$  and  $M', M'' \in \pi^i$  s.t. ▷ Search, on all dimensions, the pair of member sets...
6:    $\text{Drill}_1 \left( \text{RA} \left( M' \cup M'' \right) \right) = M' \cup M''$  ▷ ... whose union satisfies the lazy h-compliance constraint...
7:   and  $\frac{\Delta SSE_{M' \cup M''}(P)}{|\pi_j \neq i| |\pi^j|}$  is minimal ▷ ... and that has minimal weighted  $\Delta SSE$ 
8:    $\bar{P} \leftarrow \text{Aggl}_{M' \cup M''}^i(P)$  ▷ Agglomerate  $P$ 
9:   if  $SSE(\text{Red}_{\bar{P}}(C)) \leq \alpha_{SSE}$  then ▷ If the new md-partition does not violate the SSE constraint...
10:     $P \leftarrow \bar{P}$  ▷ ... update the current solution
11:   else
12:     break
13: return  $P$ 

```

same time, while still meeting requirements (i) and (ii). However, this would dramatically increase the computational complexity of our algorithms since the number of possible n -dimensional unions of members grows exponentially with n .

4.5.2 Lazy Shrink Computation

In the light of the above, in this section we present an agglomerative greedy algorithm for lazy shrink. As a merging criterion we adopt *Ward's minimum variance method* [108], i.e., at each agglomeration step we select the pair of member sets yielding minimal ΔSSE weighted on the number of dices that will be unified. The reason for this weighting is that not all agglomerations result in the same decrease in size (i.e., number of f-dices). Without considering this weighting factor, agglomerations involving few dices would generally be favored, leading to less optimal choices. Of course, two member sets can be selected for agglomeration only if the resulting md-partition is h-compliant.

The pseudo-code for Reduction Problem 1 is given in Algorithm 5. After initializing P as a md-partition of singletons (lines 1-3), the process iterates until the SSE constraint is violated (line 10) or there are no more possible agglomerations (line 4). At each iteration, the pair of member sets which has minimal weighted ΔSSE and whose union fulfills the lazy h-compliance constraint is searched (line 5). If the resulting agglomeration still satisfies the SSE constraint (line 7), then the current md-partition is updated to the new one (line 8). The algorithm for Problem 2 can be obtained by simply replacing the if condition at line 7 with $|P| > \alpha_{\text{size}}$, in which case the process is stopped as soon as the size constraint is fulfilled.

Example 22 We now show a walkthrough of Algorithm 5 using the cube in Figure 4.7 and $\alpha_{SSE} = 80$ as input (Figure 4.12):

1. The initial md-partition (Figure 4.12.a) is the Cartesian product of the singleton

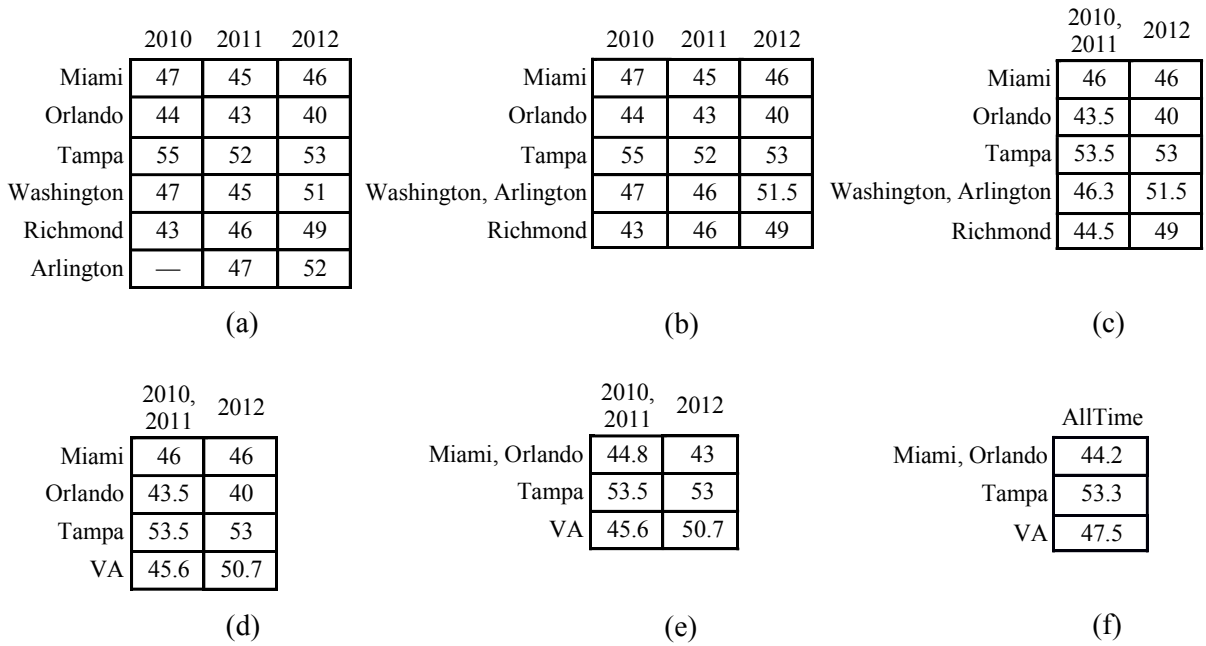


Figure 4.12: Applying the greedy algorithm for lazy shrink

partitions π_1 and π_2 of hierarchies RESIDENCE and TIME, respectively, defined as

$$\pi_1 = \{\{\text{Miami}\}\{\text{Orlando}\}\{\text{Tampa}\}, \{\text{Washington}\}, \{\text{Richmond}\}, \{\text{Arlington}\}\}$$

$$\pi_2 = \{\{2010\}, \{2011\}, \{2012\}\}$$

Since this md-partition has size $6 \times 3 = 18$, the while cycle is entered.

2. The first best candidate agglomeration involves Arlington and Washington, and has a weighted ΔSSE of 0.8. The resulting reduction (Figure 4.12.b) has a total SSE of 2.5 and size $5 \times 3 = 15$, so the while cycle is entered again.
3. At each following iteration, the best h-compliant agglomeration is searched, yielding the reductions shown in Figure 4.12.c, 8.d, and 8.e. The total SSE now amounts to 47.1, the size to $3 \times 2 = 6$.
4. At the last iteration, the best agglomeration involves $\{2010, 2011\}$ and $\{2012\}$ and has a weighted ΔSSE of 17.5 (Figure 4.12.f). However, the total SSE would be 99.5, which exceeds α_{SSE} . So, the algorithm stops and returns the reduction shown in Figure 4.12.e.

For the sake of simplicity, the process of computing the SSE variations of agglomerations has been hidden in Algorithm 5. However, since most of the computational complexity of the algorithm lies there, we will describe it in more detail. The **Find** statement at line 5 requires the ΔSSE 's of all feasible agglomerations. To avoid computing these ΔSSE 's at each iteration, besides the current md-partition P and its corresponding reduction

Figure 4.13: Updating the two distance matrices for ΔSSE computation in the 2-dimensional case (the bottom-right matrix is the current reduction $Red_P(C)$, the parts in gray are those affected by the update)

$Red_P(C)$ we store, for each dimension j , a *distance matrix*¹ containing the ΔSSE 's induced by agglomerating any two member sets in π^j . Of course at each iteration, when an agglomeration is made on a pair of sets $M', M'' \in \pi^i$, all the ΔSSE 's stored for these two sets must be updated since M' and M'' are replaced in π^i with their union (see Definition 12). In particular, updating the n distance matrices requires:

- Updating the matrix for dimension i , which only involves the two rows and columns corresponding to M' and M'' .
- Updating the matrices of the remaining dimensions, which involves all their cells.² However, the new values need not be computed from scratch because, consistently with the definition of SSE variation of an agglomeration, they can be obtained from the previous ones by adding the ΔSSE 's of the newly created dices and subtracting the ΔSSE 's of the original ones.

Figure 4.13 exemplifies the distance matrix update process for the 2-dimensional case.

We close this section by estimating the asymptotic computational cost of Algorithm 5 considering its worst-case, which takes place when the following two conditions are met: (i) no h-compliance constraints are present, and (ii) the α parameter is such that the resulting multidimensional partition is composed by exactly one dice. In this case, given a cube C at group by $G = \langle l_1^1, \dots, l_1^n \rangle$ with $|Dom(l)| = \delta \forall l \in G$, the total complexity is $(n^2\delta^3)$ in terms of comparisons and $(n^2\delta^{n+1})$ in terms of ΔSSE computations. More in detail, to search for the minimum weighted ΔSSE we scan n distance matrices —each requiring (δ^2) accesses; this is repeated at each of the $n(\delta - 1)$ iterations necessary to reduce the cube to one dice. As to ΔSSE computation, the operation that determines the highest cost is the update of the distance matrices for the $n - 1$ dimensions that have not been agglomerated at the current iteration. This requires updating all the $O(\delta^2)$ cells of each distance matrix by reading $O(\delta^{n-2})$ f-dices for each cell, and must be repeated for each of the $n(\delta - 1)$ iterations.

¹We use the term “distance matrices” because, in clustering approaches, matrices like these ones are normally used to store inter-cluster distances.

²More precisely, only the cells corresponding to agglomerations that meet h-compliance constraints are updated.

Algorithm 6 Heuristic Eager MD-Shrink

Require: A cube C at a group-by $G = \langle l_1^1, \dots, l_1^n \rangle, \alpha_{SSE}$ **Ensure:** An h-compliant md-partition P of G

```
1: for all  $j = 1, \dots, n$  do ▷ For each dimension build a partition of singletons
2:    $\pi^j \leftarrow \{ \{m\}, m \in \text{Dom}(l_1^j) \}$ 
3:  $P \leftarrow \pi^1 \times \dots \times \pi^n$  ▷ Create the initial md-partition of singletons
4:  $Cand \leftarrow \bigcup_j \text{Dom}(l_1^j)$  ▷ Initialize the set of candidate members for fusion
5: while  $|P| > 1$  do ▷ Agglomerating is still possible
6:   Find  $m_k^i \in Cand$  s.t.  $\frac{\Delta SSE(\text{Drill}_1(m_k^i))}{|\text{Drill}_{k-1}(m_k^i)| \cdot \prod_{j \neq i} |\pi^j|}$  is minimal ▷ Find the best candidate...
7:    $Cand \leftarrow Cand \setminus \{m_k^i\}$  ▷ ... remove it from the set of candidates...
8:   if  $Cand \cap \text{Drill}_k(\text{Roll}^{k+1}(m_k^i)) = \emptyset$  then ▷ If the new md-partition does not violate the SSE constraint...
9:      $P \leftarrow \bar{P}$  ▷ ... update the current solution
10:  else
11:    break
12: return  $P$ 
```

4.5.3 Eager Shrink Computation

Like lazy shrink, eager shrink is based on agglomeration as defined in Definition 12. We recall that the eager form of the h-compliance constraint is stricter than the lazy form. For instance, with reference to our working example, while lazy shrink allows reductions where only a subset of cities in the same state are fused together, in eager shrink either all or none cities in a state are fused together. For this reason, in a greedy algorithm for eager shrink, the generation of candidate unions can more efficiently be driven “from above”, i.e., from LCA’s, like shown in Algorithm 6.

Algorithm 6 solves Problem 1 by hinging on variable $Cand$, that stores the LCA’s of candidate members for fusion. At each iteration (line 5), the most promising candidate m_k^i is picked from $Cand$ by considering its weighted ΔSSE (line 6); in this case, weighing depends on the number of children members of m_k^i being unified as well as on the size of the current partitions along the hierarchies different from i . If at some point all the siblings of a given member m_k^i have been agglomerated, then the father of m_k^i must be added to $Cand$ (lines 8-9). For instance, when the cities of the last state of the South-Atlantic region are fused into an f-dice (which means that no state of that region is in $Cand$), member South-Atlantic is added to $Cand$. The algorithm for Problem 2 can be obtained by simply replacing the if condition at line 11 with $|P| > \alpha_{size}$, in which case the process is stopped as soon as the size constraint is fulfilled. Note that, differently from Algorithm 5, the **Find** statement here does not computes all the possible combinations between couples of member sets. Thus, instead of storing n distance matrices, it is sufficient to store a ΔSSE for each element in $Cand$. Actually, for optimization purposes we implemented n separate candidate lists, one for each dimension.

Example 23 Consider again the cube in Figure 4.7. In the following we show in detail how Algorithm 6 computes a reduction that solves Problem 1 with $\alpha_{SSE} = 80$ (Figure 4.14).

1. The initial partition is the same seen in Example 8. Since its size is 18 the while

	2010	2011	2012
Miami	47	45	46
Orlando	44	43	40
Tampa	55	52	53
Washington	47	45	51
Richmond	43	46	49
Arlington	—	47	52

(a)

	2010	2011	2012
Miami	47	45	46
Orlando	44	43	40
Tampa	55	52	53
VA	45	46	50.7

(b)

	AllTime
Miami	46
Orlando	42.3
Tampa	53.3
VA	47.5

(c)

	AllTime
FL	47.2
VA	47.5

(d)

Figure 4.14: Applying the greedy algorithm for eager shrink

cycle is entered, with

$$Cand = \{FL, VA, AllTime\}$$

2. *The best candidate is VA, whose weighted ΔSSE is 2.4. The SSE of the resulting reduction (Figure 4.14.b) is 14.7, which meets the α_{SSE} constraint. Now it is*

$$\pi_1 = \{\{Miami\}, \{Orlando\}, \{Tampa\}, \{Washington, Richmond, Arlington\}\}$$

$$\pi_2 = \{\{2010\}, \{2011\}, \{2012\}\}$$

3. *Since this md-partition has size $4 \times 3 = 12$ the while cycle is entered, with*

$$Cand = \{FL, AllTime\}$$

4. *The best candidate is now AllTime, whose weighted SSE is 8.1. The total SSE is 79.3 (Figure 4.14.c), so the while cycle is entered again, with*

$$\pi_1 = \{\{Miami\}, \{Orlando\}, \{Tampa\}, \{Washington, Richmond, Arlington\}\}$$

$$\pi_2 = \{2010, 2011, 2012\}$$

$$Cand = \{FL\}$$

5. *At the next iteration, the only possibility would be to fuse together the cities of Florida. However, since the resulting reduction has SSE higher than α_{SSE} (Figure 4.14.d), the algorithm stops. The reduction finally returned is the one shown in Figure 4.14.c.*

Like for Algorithm 5, to estimate the computational cost of Algorithm 6 we characterize its worst-case, whose conditions are: (i) hierarchies are structured as binary trees, i.e., each member drills-down to exactly two child members; and (ii) α is such that the resulting md-partition is composed of exactly one f-dice. While condition (ii) is the same as the one of lazy shrink, here condition (i) is the opposite (as already mentioned in the section

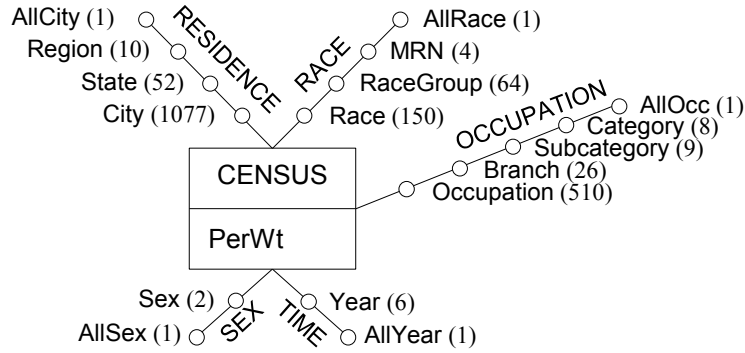


Figure 4.15: The complete CENSUS schema

about the search space size). Again, we consider a cube C at group-by $G = \langle l_1^1, \dots, l_1^n \rangle$ with $|Dom(l)| = \delta \forall l \in G$. The total complexity is $(n^2\delta^2)$ in terms of comparisons and $(n^2\delta^n)$ in terms of SSE computations. To search for the minimum weighted ΔSSE we scan the $Cand$ set, whose size is (n) ; this is repeated for each of the $n(\delta - 1)$ iterations needed to reduce the cube at exactly one f-dice. As to ΔSSE computation, similarly to Algorithm 5, the operation that determines the highest cost is the update of the candidate lists for the $n - 1$ dimensions that have not been agglomerated at the current iteration. Updating each of the δ elements in a candidate list requires reading $O(\delta^{n-2})$ f-dices, and this must be repeated for each of the $n(\delta - 1)$ iterations.

4.5.4 Experimental Results

This section collects the results of the tests that evaluate the shrink operator in its two forms. The algorithms are implemented in C++ and ran on a desktop PC dual-core (3GHz, 4 GB RAM, Windows 7-64 bit). The tests used the complete 5-dimensional CENSUS schema, sketched in Figure 4.15 using the Dimensional Fact Model notation (numbers next to levels show their domain cardinality).

We recall that the shrink operator is meant to be applied to any cube at any group-by resulting from an OLAP query. So, to enable a thorough discussion of the behavior of shrink under different conditions, we extracted from the IPUMS database [9] seven different cubes differing in their size —number of facts— and dimensionality as shown in Table 4.10. Noticeably our benchmark also includes cubes, such as *Census7*, that are quite large in size to be the results of a typical OLAP query, and whose dimensionalities are too high for a profitable visualization with a pivot table. The reason for this is that, as mentioned in Section 4.5, though the ultimate goal of shrink is to provide compact approximations of pivot tables for users' direct inspection, it can also be applied to generate accurate and slightly-condensed representations of huge datasets to be transferred to other analysis applications.

Table 4.10: The cubes used for tests

Name	Group-by	# Facts	Dimensionality
<i>Census1</i>	⟨State, RaceGroup⟩	3328	2
<i>Census2</i>	⟨City, RaceGroup⟩	68928	2
<i>Census3</i>	⟨City, Race⟩	161550	2
<i>Census4</i>	⟨State, Branch, Year⟩	8112	3
<i>Census5</i>	⟨State, RaceGroup, Branch⟩	86528	3
<i>Census6</i>	⟨City, Race, Branch⟩	4200300	3
<i>Census7</i>	⟨City, Race, Branch, Year⟩	25201800	4

Before delving into the evaluation of the obtained results, we briefly formalize the notations we will use in our charts to enable a fair comparison between cubes with very different features. The approximation introduced by a reduction $Red_P(C)$ is measured through its *relative SSE*

$$SSE \% = \frac{SSE(Red_P(C))}{SSE(Red_{P_{\min}}(C))}$$

where $Red_{P_{\min}}(C)$, with $|P_{\min}| = 1$, is the smallest reduction obtainable from C . Similarly, the *relative size* of $Red_P(C)$ is expressed as

$$Size \% = \frac{Size(Red_P(C))}{Size(C)}$$

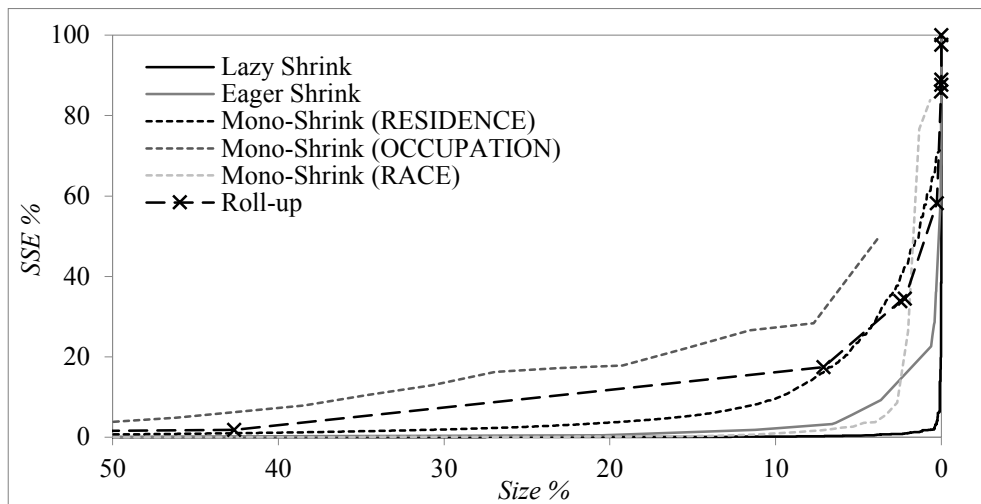
Finally, given $Red_P(C)$ with $P = \pi^1 \times \dots \times \pi^n$, for each hierarchy h_i we define

$$Dimension\ Size \% = \frac{|\pi^i|}{|Dom(l^i)|}$$

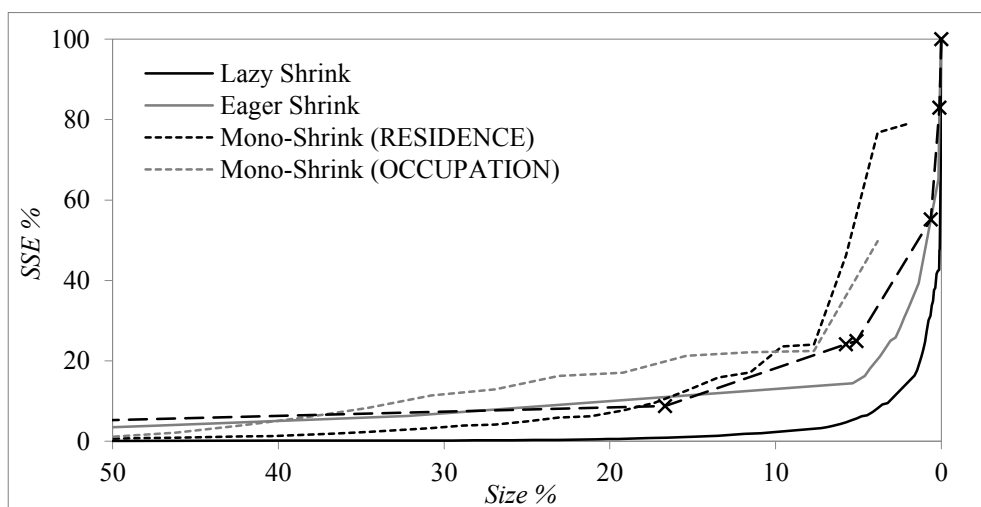
where $G = \langle l^1, \dots, l^i, \dots, l^n \rangle$ is the group-by of C . Intuitively, this indicator measures to what extent a reduction impacts on the number of rows/columns of the pivot table that shows data.

4.5.4.1 Effectiveness Analysis

In this section we evaluate shrink in terms of effectiveness. We start with a comparison between lazy/eager shrink, mono-dimensional shrink [5], and classical OLAP roll-up, which could be seen as a simpler alternative to reduce the size of a cube. Since the number of possible roll-up's grows exponentially with the cube dimensionality, for a fair comparison we implemented a greedy multidimensional roll-up operator which basically works as a variant of shrink where the smallest reduction step is that of a roll-up. As done in Algorithm 5 and 6, at each iteration we roll-up to the group-by with minimal ΔSSE weighted on the number of dices that will be unified. Figure 4.16 shows the results of the comparison tests performed on cubes *Census7* and *Census4*. The mono-dimensional version of shrink (called Mono-Shrink in the diagrams) has multiple data series —one for



(a)

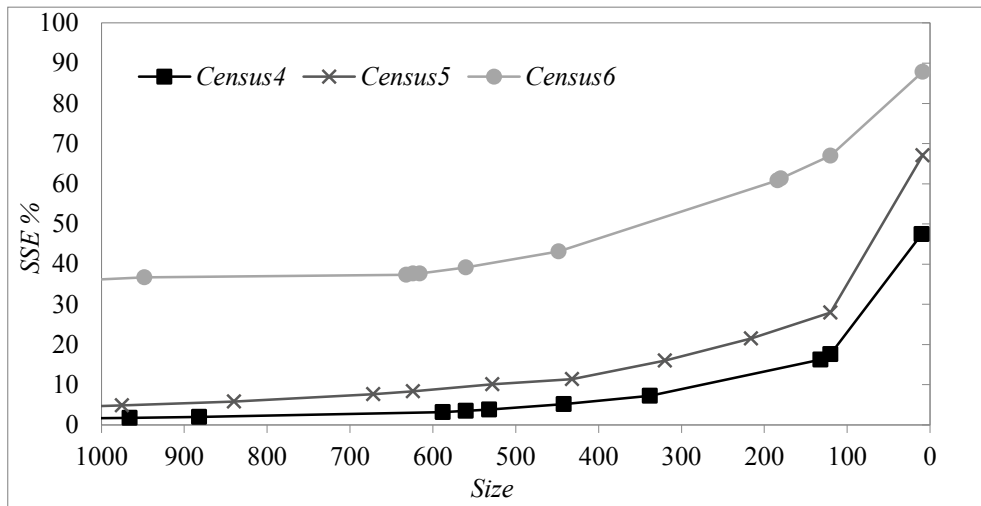


(b)

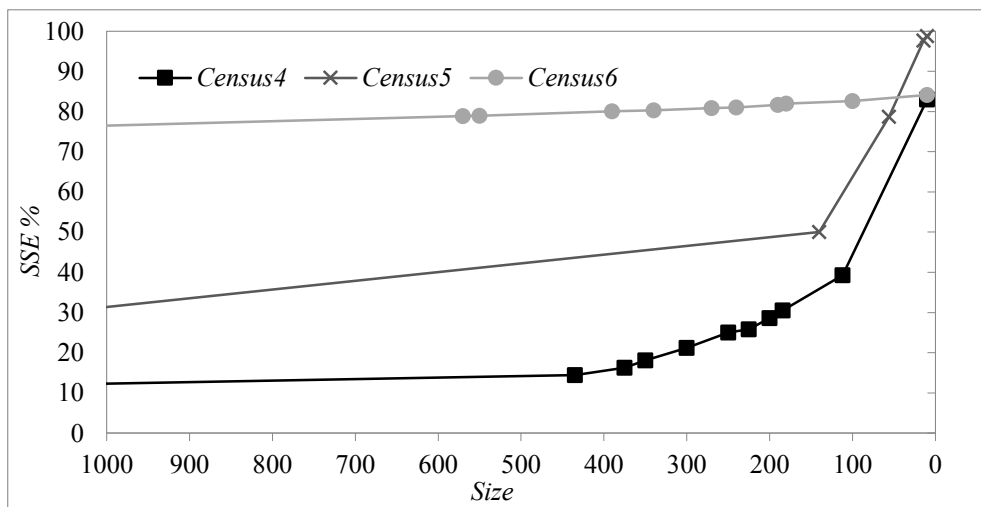
Figure 4.16: Reduction size vs. approximation for Census7 (a) and Census4 (b)

each hierarchy— since shrinking along different hierarchies can yield very different results. Due to the low cardinality of the **Year** level, the mono-dimensional series for the **TIME** hierarchy has been omitted to improve readability.

The first look at Figure 4.16 reveals that, for all the approaches, the SSE grows exponentially as the reduction process is pushed further. This is because (i) null facts give no contribution to the SSE and sparseness is higher for reductions with larger size; (ii) agglomerations leading to low SSE’s are chosen first by our algorithms. Besides, all the algorithms yield negligible approximations at large sizes —this is why we restrict the chart to sizes lower than 50%. A closer comparison of the different curves shows that lazy shrink performs remarkably well, with low SSE’s even at 1% size. In some cases, eager shrink is less effective than mono-dimensional shrink; however, eager shrink offers a simpler presentation of data due to its strict h-compliance constraints, so depending on the user’s goals, the



(a)

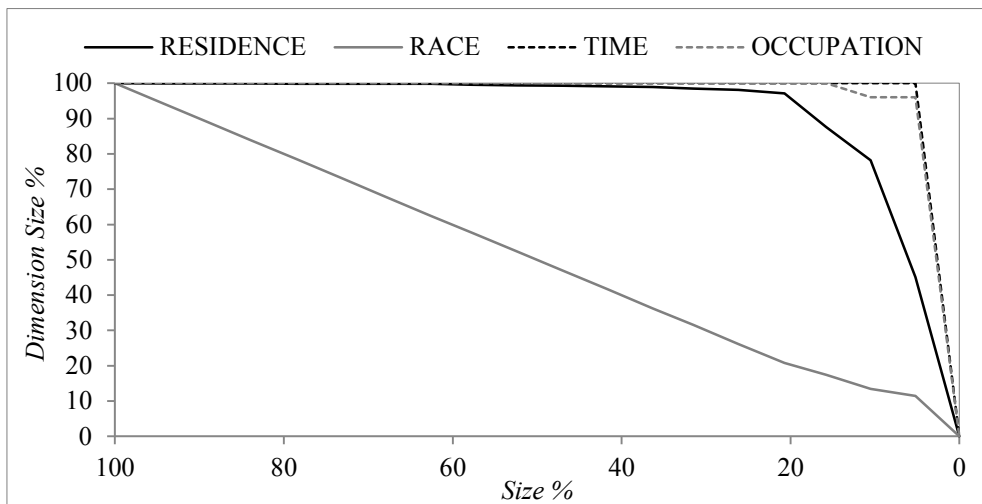


(b)

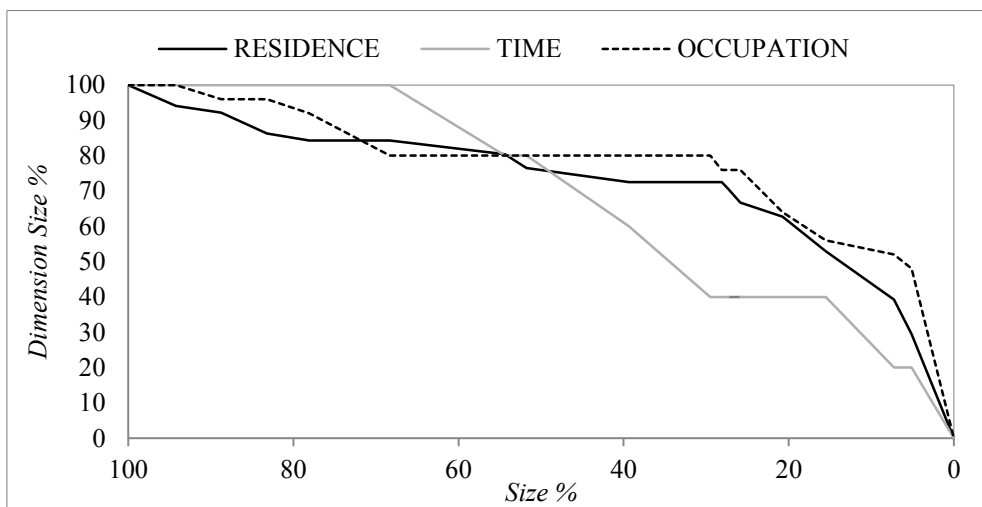
Figure 4.17: Reduction size vs. dimension size for Census7 (a) and Census4 (b)

trade-off with accuracy might be justified. As expected, both versions of shrink perform better than roll-up, which has the additional shortcoming of allowing only a few possible reductions (shown by cross-shaped markers in the chart). Interestingly, the effectiveness gap between shrink and mono-dimensional shrink tends to be greater for *Census4* than for *Census7*, even in the case of eager shrink; an explanation for this will be given below.

Figure 4.17 shows how the dimension size scales, for each hierarchy, with the reduction size. For *Census7* (Figure 4.17.a) the reductions are mostly driven by RACE until only a few possible agglomerations (all with high SSE's) remain on that dimension. To understand how shrink behaves in the absence of a dominant dimension like RACE, we need to consider *Census4* (Figure 4.17.b), which has the same dimensions of *Census7* except RACE. In this case the different dimensions are used in a more balanced way for shrinking from the very beginning of the reduction process which leads, as mentioned above, to the greater



(a)



(b)

Figure 4.18: Absolute reduction size vs. approximation of three cubes for lazy (a) and eager (b) shrink

effectiveness gap between the multi- and mono-dimensional versions of shrink.

Another factor that we deemed interesting to analyze in the context of a comparison between the different approaches, is the dimensionality of the input cube. So we repeated the size vs. approximation test with cubes *Census3* and *Census6*, that have two and three dimensions respectively (while *Census7* has four). It turned out that a higher dimensionality favors shrink against its mono-dimensional version but, unexpectedly, to a small extent.

To introduce the last group of tests we recall that the main scenario for shrink is the one where the size of the resulting reduction is small enough to enable the user to fruitfully analyze its contents. For this reason in the following tests we will focus on a smaller —yet

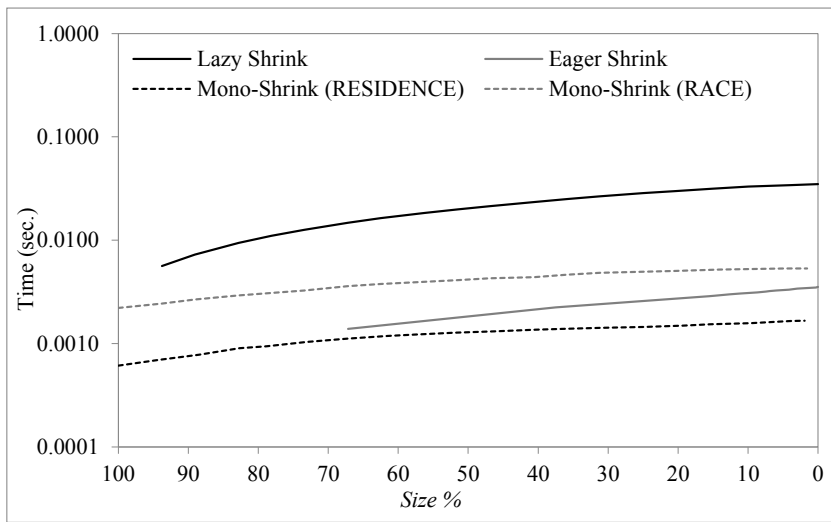
more significant— *absolute* size range that can be considered the normal operation range of shrink, namely, from 1000 to 10 f-dices. Figure 4.18 shows how the approximation changes for cubes of different sizes within this operation range. The results clearly show that larger cube sizes lead to worse approximations for the same absolute reduction size. This is coherent with what we observed in the case of mono-dimensional shrink and can be ascribed to a dimensionality problem. Indeed, as depicted in Figure 4.11, an agglomeration involves two or more sets of members (M_1^1 , M_2^1 , and M_3^1 in the figure) of one level; each member set determines a set of f-dices that can be seen as a multidimensional object (with dimensionality 36 in the figure). When dimensionality increases, so does the difficulty in finding agglomerations that determine a small SSE increase [109]. Notably, while for the cubes in Figure 4.18 this trend neatly emerges thanks to a relevant difference in size, in other situations things may be blurrier due to other factors such as measure values—for example, cubes *Census2* and *Census3* yield mostly overlapping curves. As a final remark, a comparison between Figures 14.a and 14.b reveals that—especially for *Census5*—the curves of eager shrink include a lower number of points than those for lazy shrink because the granularity of agglomerations is coarser when using eager h-compliance.

4.5.4.2 Efficiency Analysis

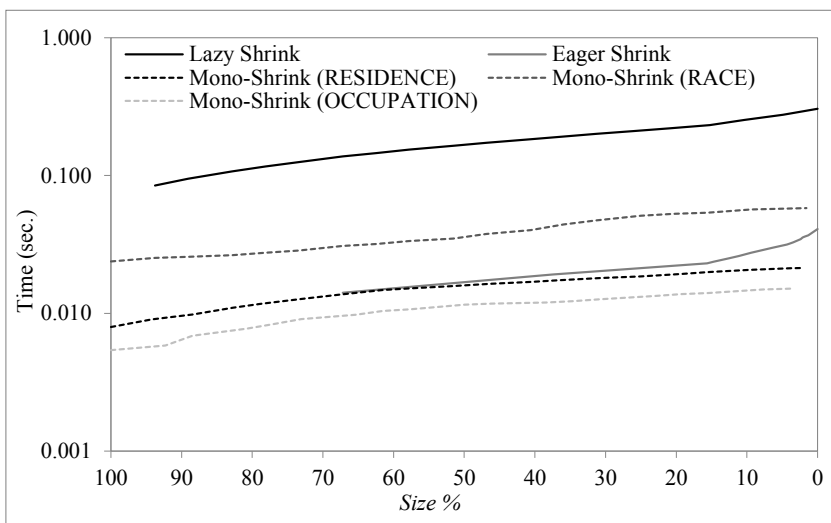
In this section we discuss the execution time of the shrink operator.

Figure 4.19 compares the multi- and mono-dimensional versions for three cubes using a logarithmic scale. Eager and mono-dimensional shrink yield similar results, while lazy shrink takes significantly longer to be computed. The marked difference between lazy and eager shrink can obviously be ascribed to the different strength of the constraints used: in lazy h-compliance, many more feasible md-partitions have to be computed than in eager h-compliance. Except for *Census7*, all the approaches appear to be compatible with the near-real-time requirements of OLAP analyses. In the case of *Census7*, lazy shrink takes a long computation time—about 330 sec. to reduce the cube to one f-dice—which is not surprising considering the large size and dimensionality of the input cube.

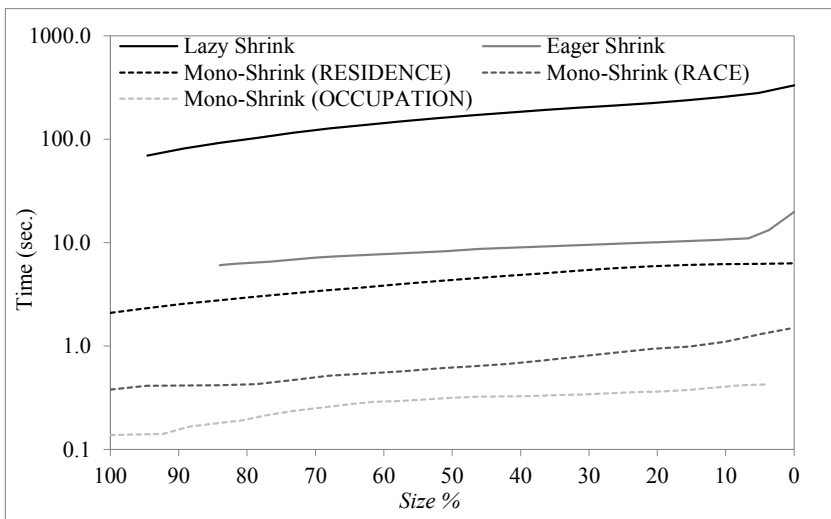
Finally, Figure 4.20 shows a comparison of lazy and eager shrink with cubes of different size and dimensionality. As expected, in both cases there is a steep increment in execution times from *Census5* to *Census7*. The main factors that affect the performance of shrink are the size and dimensionality of the input cube, as well as the hierarchy structure. The first factor is easily explained, as more facts require more Δ SSE's to be computed and, on the other hand, a higher dimensionality requires more md-partitions to be generated. As to the second factor, the hierarchy structure determines the tightness of h-compliance constraints, which in turn affects the number of h-compliant md-partitions (see Section 4.5.1.3).



(a)

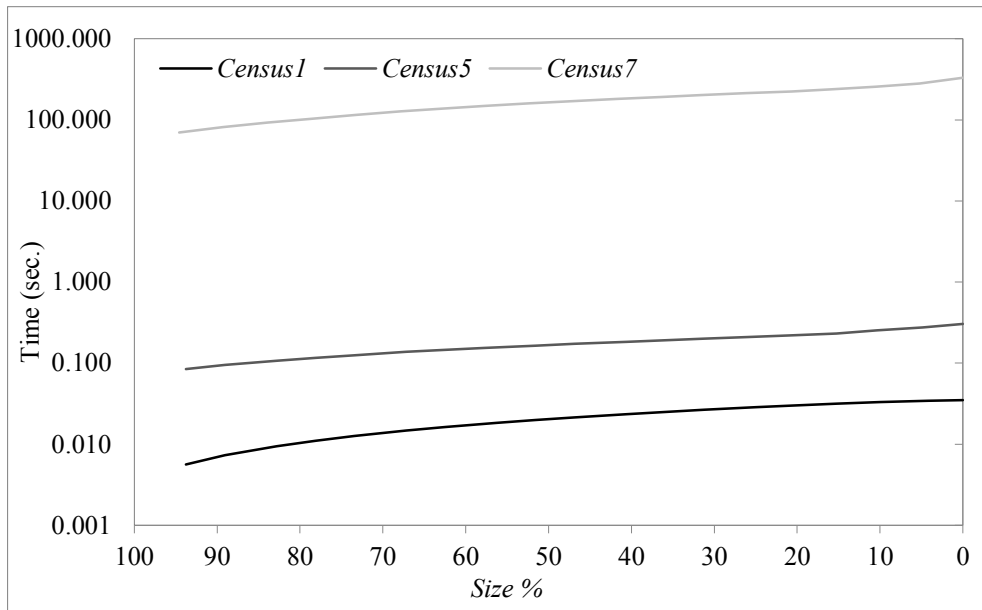


(b)

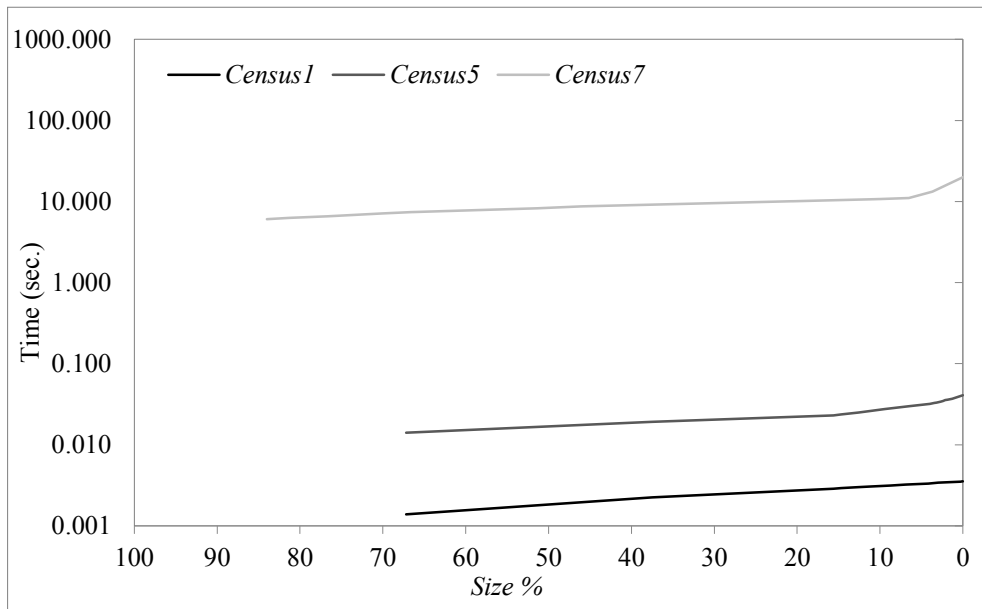


(c)

Figure 4.19: Execution time vs. reduction size for Census1 (a), Census5 (b) and Census7 (c)



(a)



(b)

Figure 4.20: Execution time vs. reduction size for lazy (a) and eager (b) md-shrink

4.5.5 Wrapping up Multidimensional Shrink

In this work we have presented a multidimensional generalization of the shrink operator to balance the size of a pivot table and its approximation. Two variants of the operator have been devised, based on hierarchy constraints with different strictness, and for each variant a greedy algorithm has been proposed. The experimental results show that multidimensional shrink overcomes both the previous mono-dimensional version and the

classical roll-up OLAP operator in terms of effectiveness, i.e., it tends to achieve a better conciseness/approximation trade-off. Besides, for cubes whose size is not prohibitive from the point of view of a tabular visualization, the performance of shrink is perfectly compatible with the real-time requirement posed by an interactive use during OLAP sessions. Finally, we saw that the eager variant is less effective than the lazy one, but it enables more compact visualizations (in terms of row/column labeling) with smaller execution times.

Chapter 5

Modeling of Unconventional Data Sources

In this chapter we present two different approaches to obtain multidimensional schemata; specifically, we focus on two different types of data sources, namely data vaults and sensor networks.

The data vault model natively supports data and schema evolution, so it is often adopted to create operational data stores. However, it can hardly be directly used for OLAP querying. In Section 5.1 we propose an approach called *Starry Vault* for finding a multidimensional structure in data vaults. Starry Vault builds on the specific features of the data vault model to automate multidimensional modeling, and uses approximate functional dependencies to discover out of data the information necessary to infer the structure of multidimensional hierarchies. The manual intervention by the user is limited to some editing of the resulting multidimensional schemata, which makes the overall process simple and quick enough to be compatible with the situational analysis needs of a data scientist.

The approach that we employ to obtain multidimensional schemata for sensor data is quite different from Starry Vault. Indeed, in this case we propose manually designed multidimensional schemata that cover the main requirements usually related to the domain of sensor networks. Furthermore, we describe in detail how these schemata can be used in practice by showing their application in two different use-cases: the first one is related to the monitoring of air quality in a chemical facility, while the second one focuses on risk management for landslides.

5.1 Automating Multidimensional Modeling from Data Vaults

Since their adoption as an enabling technology for information systems, one of the goals of databases has been to provide a unified, integrated, and consistent repository for *all* enterprise data; this repository should act as a hub for different activities such as process coordination, auditing, historical data storage, etc. Among the solutions devised in this direction we mention Master Data Management and ERPs in the area of operational systems; in the area of business intelligence, Operational Data Stores and, more recently, data lakes. Another solution that has been progressively gaining attention and diffusion since its official release in 2000 is the *data vault*, a practitioner-driven proposal for designing a database that provides long-term historical storage of data coming in from multiple sources. The main goals of the data vault can be summarized as (i) maximize resilience to change in the business environment when storing historical data; (ii) accommodate data regardless of their quality and of their conformity to standard and business rules; and (iii) enable parallel loading so that very large implementations can scale out without the need of major redesign. While the 1.0 version of the data vault was strictly relational, version 2.0 (released in 2015) relies on Hadoop-Hive for delivering scalability and performance at a big data level. However, in spite of its undeniable informative value, a data vault is not suitable for direct multidimensional querying both for performance reasons (it is not optimized for OLAP workloads) and because it is hardly supported by OLAP front-ends.

In this work we propose an approach called *Starry Vault* aimed at finding a multidimensional structure in data vaults so that their data can be fed into a data warehouse (DW) for OLAP querying. On the one hand, our approach builds on the specific features of the data vault model to automate multidimensional modeling, on the other it uses approximate functional dependencies [110] to discover out of data the information necessary to infer the structure of multidimensional hierarchies. The Starry Vault approach is mainly aimed at being used at design time, to support a supply-driven design of a DW from a source data vault [111]. However, the manual intervention by the user is limited to some editing of the resulting multidimensional schemata, which makes the overall process simple and quick enough to be also compatible with the situational analysis needs typical of a data scientist.

5.1.1 Related Work

The data vault model has hardly been explored in the academic literature. Besides the official model specification [6], to the best of our knowledge only a couple of works were made: [112], which provides a conceptualization of the data vault physical model, and [113], which describes an approach for designing DWs where the data vault model is used instead of the standard star/snowflake schemata to physically implement the multidimensional

model. On the other hand, there are evidences that the data vault can be used in agile design contexts [114], and some CASE tools generate DW schemata based on the data vault model (e.g., Quipu [115]).

The problem of how to support or even automate the design of DWs has been widely explored. In particular, in *supply-driven* approaches multidimensional modeling starts from an analysis of data sources—which is in line with the goal of this work. The first approaches to supply-driven design date back to the late 90’s [116, 117, 118, 119] and propose algorithms that create multidimensional schemata starting from Entity-Relationship diagrams or relational schemata. The basic idea is that of following the functional dependencies (FDs) expressed in the source schema to build the multidimensional hierarchies. In the following years, there have been some attempts to obtain multidimensional schemata out of XML source data (e.g., [120]). In this case, the main problem is that some FDs are not intensionally expressed, so they must be checked extensionally, i.e., by properly querying the XML database at design time.

More recently, a few works appeared focused on *RDF analytics* [121, 122], i.e., on how to query RDF data in an OLAP-like fashion. Though still at an early stage of development, these works may pave the way towards the design of multidimensional schemata starting from ontologies. In the same direction, in [123] the AMDO approach is proposed to derive a multidimensional schema from a conceptual formalization of the domain; the data sources are analyzed to look for multidimensional patterns, then the results are combined with the knowledge expressed by a domain ontology to support the elicitation of requirements. Along these lines, semantic web technologies have also been applied to enable exploratory OLAP on external data [124]. To achieve this, source data are modeled as an ontology, which then is used to derive a multidimensional schema.

The main inspiration for our current work comes from the supply-driven approaches that use relational schemata as a source. However, these approaches cannot be smoothly reused in our case because (i) while in traditional (normalized) relational databases all FDs are made explicit, several FDs are normally hidden in data vaults; (ii) the peculiar structure of data vaults, lets us make some specific assumptions which are not possible with traditional relational databases; (iii) while relational-based approaches do not use many-to-many relationships for design, these must always be considered when designing from data vaults. On the other hand, the idea of querying data vaults to establish the missing FDs is borrowed from the approaches using XML sources.

Among the works on supply-driven design of DWs, some also consider the problem of supporting the designer in detecting potential facts. For instance, in [117] all the entities with numeric fields are selected as candidate facts. Not only the presence of measures, but also table cardinality is considered to identify facts in [118], while in [119] all entities with a high number of many-to-one relationships are candidates to become facts. A model-driven approach to detect fact is proposed in [125], based on a heuristics that considers the cardinality and in-degree of each table, together with its ratio of numerical fields. Finally,

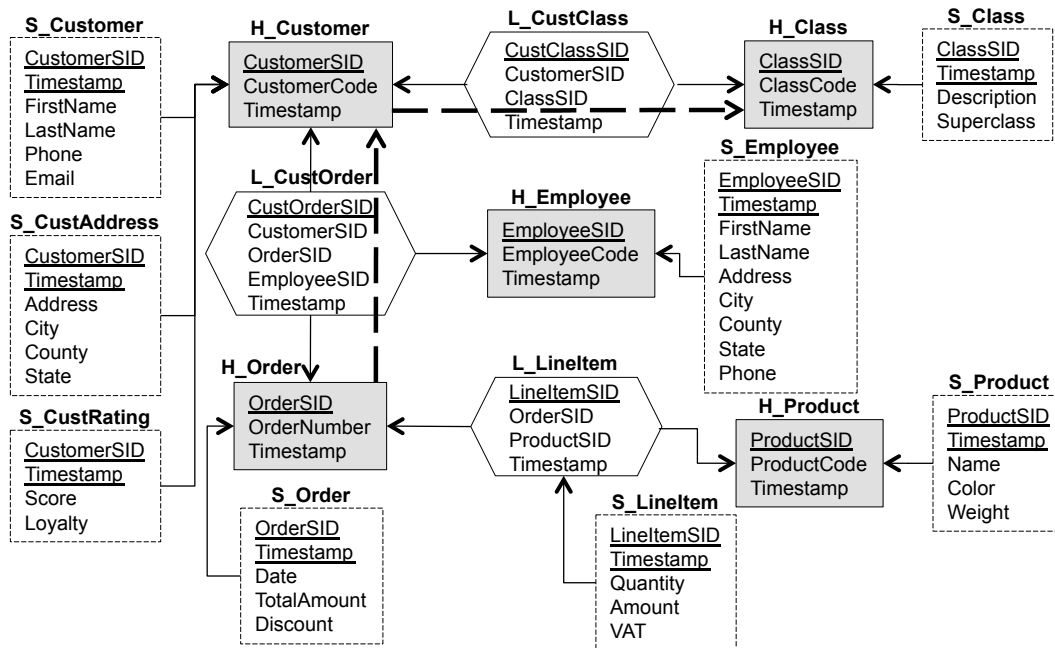


Figure 5.1: A sale data vault. Grey boxes, hexagons, and dashed boxes represent hubs, links, and satellites, respectively; additional FDs are shown with thick dashed arrows

in [123] potential facts are selected by searching specific topological patterns in source data. The criteria we use in this work for ranking candidate md-schemata are partially inspired and adapted from the ones mentioned above.

5.1.2 Data Vault Basics

The data vault model was conceived by Dan Linstedt in 1990 and then released in 2000 as a public domain modeling method [6]. Its basic goal is that of dealing with data and schema changes by separating the business keys (that are basically stable, because they uniquely identify a business entity) and the associations between them, from their descriptive attributes (that may change frequently). The data vault is based on three components [126]:

- *Hubs.* A hub is a table that models a core concept of business; each of its tuples corresponds to a single business object with a unique enterprise-wide key, and is timestamped with the moment that object was first loaded into the database. The primary key of a hub is always a surrogate key.
- *Links.* A link is a table that models a business relationship between hubs. To establish this relationship, a link includes foreign keys referencing the hubs/links involved. Like a hub, it has a surrogate as the primary key and it includes a load timestamp. To ensure that the schema can be easily evolved, all relationships are modeled as potentially many-to-many regardless of their actual multiplicity.

- *Satellites.* A satellite is a table that includes a set of attributes describing one hub or one link. Its primary key combines a foreign key that references the corresponding hub/link with a timestamp, so that multiple temporal version of attribute values can be stored.

Example 24 *The simple data vault we will use as a working example models sale orders and is shown in Figure 5.1 (adapted from [126]).*

5.1.3 Formal Background

In this section we give a graph-based formalization of data vaults and multidimensional schemata, which will be respectively the input and output of our design algorithm.

Definition 22 (Data Vault Schema) *A data vault schema (briefly, dv-schema) is a directed graph $\mathcal{V} = (T, F)$ where $T = T_H \cup T_L \cup T_S$ and:*

1. $T_H, T_L,$ and T_S are, respectively, sets of hub, link, and satellite tables;
2. each arc $\langle t, t' \rangle$ in F represents an FD from a foreign key of table t to the primary key of table t' , which we will denote with $t \rightarrow t'$ to emphasize that one tuple of t determines one tuple of t' ;
3. $F \subseteq (T_S \times (T_H \cup T_L)) \cup (T_L \times T_H)$;
4. exactly one arc exits from each satellite $s \in T_S$ (entering a hub or a link);
5. at least two arcs exit from each link.

Given point (3) of Definition 22, all FDs explicitly modeled in a dv-schema take either form $s \rightarrow h$, $s \rightarrow l$, or $l \rightarrow h$. Each hub in $h \in T_H$ has one business key, denoted $BusKey(h)$. Each satellite s has a set of business attributes, $BusAttr(s)$; for each hub or link t , we denote with $BusAttr(t)$ the union of the sets of business attributes included in all satellites s such that $s \rightarrow t$.

Example 25 *With reference to the sale data vault in Figure 5.1, it is*

$$\begin{aligned} T_H &= \{H_Customer, H_Order, H_Employee, H_Class, H_Product\} \\ T_L &= \{L_CustClass, L_CustOrder, L_LineItem\} \\ T_S &= \{S_Customer, S_CustAddress, S_CustRating, \dots\} \end{aligned}$$

An example of arc is $\langle L_CustClass, H_Class \rangle$, which corresponds to the inter-table FD $L_CustClass \rightarrow H_Class$. Finally, it is

$$\begin{aligned} BusKey(H_Customer) &= CustomerCode \\ BusAttr(H_Customer) &= \{FirstName, LastName, Phone, Email, Address, \\ &\quad City, County, State, Score, Loyalty\} \end{aligned}$$

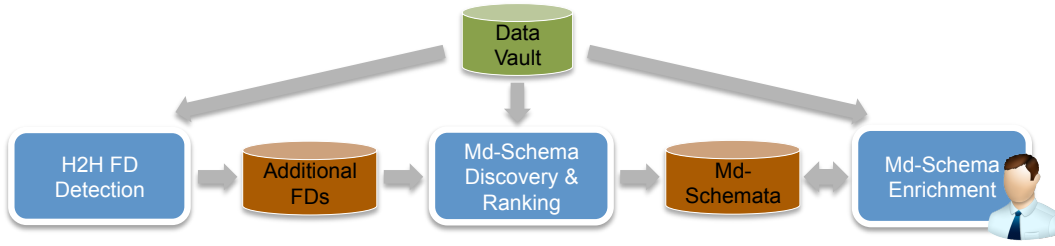


Figure 5.2: Process architecture of the Starry Vault approach

Definition 23 (Multidimensional Schema (Graph Form)) *In the context of this work, we define a multidimensional schema (or md-schema) as a directed acyclic graph $\mathcal{M} = (A, E)$ where each node in A is an attribute, each arc in E is an FD involving two attributes, and there exists one node $f \in A$, called fact, such that each other node in A can be reached from f through a directed path (which implies that f has no entering arcs). The set of direct children of f is partitioned into a set of dimensions, D , and a set of measures, M . All measures in M are leaves of \mathcal{M} . For each dimension $d \in D$, the subgraph of \mathcal{M} that can be reached from d is called a hierarchy.*

5.1.4 The Starry Vault Approach

A functional overview of the approach we use to obtain an md-schema out of a source dv-schema is sketched in Figure 5.2; three processes are included:

1. *Hub-To-Hub FD Detection.* This process aims at detecting additional FDs not explicitly modeled in the dv-schema, in particular those between two or more hubs connected by a link, by querying the source data vault.
2. *Md-Schema Discovery and Ranking.* A set of candidate facts is heuristically determined; for each of them, a draft md-schema is built based on both the FDs explicitly modeled in the dv-schema and those detected by process (1). The md-schemata obtained are then heuristically ranked based on how comprehensive they are from the intensional and extensional points of view.
3. *Md-Schema Enrichment.* The user selects one or more draft md-schemata, then edits and enriches them based on her knowledge of the application domain. To further improve the quality of the md-schemata, additional FDs hidden in satellites can be discovered by querying the source data vault.

5.1.4.1 Hub-To-Hub FD Detection

In a dv-schema each relationship between two or more hubs is modeled through a link that contains the foreign keys referencing the connected hubs. As already mentioned,

this implies that all relationships are modeled as if they were many-to-many, so it is not possible to determine if there are any FDs between two hubs (i.e., if a relationship is really many-to-many or is actually many-to-one) based on the dv-schema alone. For instance, looking at Figure 5.1 it is impossible to say if the binary relationship between customer and classes is many-to-many or, more realistically in this case, many-to-one.

Things get even more complex with n-ary relationships, like the one expressed by `L_CustOrder` that features three branches. Indeed, in this case there are different possibilities:

1. The relationship between the hubs involved really has many-to-many multiplicity in all directions. In particular, in case of the `L_CustOrder` link, this would mean that one order can be made by several customers with the support of several employees.
2. The relationship has many-to-one multiplicity from one branch towards the others. In our example, this happens if one order is always made by one customer with the support of one employee.
3. There are mixed multiplicities from the same branch. For instance, this is the case if one order is always made by one customer with the support of several employees.

Note that, while in a standard relational schema only case (1) corresponds to a good design practice for normalization reasons (in the other cases the n -ary relationship should be substituted by $n - 1$ binary relationships, each with its multiplicity), within a dv-schema all three cases are considered equally good for the sake of maintainability.

To disambiguate relationship multiplicities in all cases above and detect FDs with reasonable confidence, we must resort to the data stored in the source data vault. Clearly, there is a chance that an FD holds for the specific data stored at design time but does not hold in general in the application domain, which means that it will probably be contradicted in the future when new data will be added. Fortunately, since data vaults usually host great amounts of data, these can realistically be considered to be representative of the application domain. More probably, the data will be affected by noise in the form of errors (e.g., spelling errors) that “hide” an existing FD. The tool we use to cope with this issue are *approximate functional dependencies* (AFDs) [110], i.e., FDs that “almost hold”, which normally arise when there is a natural FD between attributes but data are dirty or present exceptions. Given AFD $a \rightsquigarrow b$, where a and b are attributes, one way to define its approximation $e(a \rightsquigarrow b)$ is to count the minimum number of distinct values of ab that must be removed to enforce $a \rightarrow b$. We will then consider $a \rightsquigarrow b$ to hold if $e(a \rightsquigarrow b) < \epsilon$, where ϵ is a threshold.

The approach we adopt to detect AFDs is an adaptation of the well-known TANE algorithm [110]. Given a table r with schema R , TANE computes all the valid AFDs $X \rightsquigarrow a$ with $X \subseteq R$ and $a \in R \setminus X$ by relying on a level-wise (small-to-big) enumeration strategy to

navigate the search space of all possible subsets of R (i.e., the containment lattice). Though TANE applies a set of pruning rules to avoid computing/returning trivial and non-minimal dependencies, its complexity remains exponential due to the number of candidate attribute sets X . Specifically, the worst-case complexity of TANE is $O(|r| + |R|^{2.5})2^{|R|}$, where $|r|$ is the cardinality of table r and $|R|$ is its number of attributes. Noticeably, since our goal here is to build hierarchies, we can restrict our search to simple AFDs ($|X| = 1$). In the remainder of this section we describe an original enumeration strategy that works for simple AFDs and cuts the complexity of TANE down to $O(|r| \cdot |R|^2)$ in the worst case and to $O(|r| \cdot |R|)$ in the best one.

Let us start by considering “traditional” FDs. Given schema R , the set of candidate FDs $a \rightarrow b$, with $a, b \in R$, can be represented using an $|R| \times |R|$ matrix Z whose rows and columns represent left- and right-hand sides of FDs, respectively, so that $Z[a, b]$ corresponds to $a \rightarrow b$. If FD $a \rightarrow b$ is found to hold on the stored data, cell $Z[a, b]$ is set to true, otherwise it is set to false. A naive approach to fill Z would check each single cell, i.e., each possible simple FD by accessing data; actually, most checks can be avoided by orderly exploring the cells of Z . Our exploration strategy requires the rows and columns of Z to be ordered by descending cardinality of the corresponding attribute domain. Given the ordered matrix, we initially note that only the cells over the diagonal must be checked since (i) the cells on the diagonal correspond to trivial FDs like $a \rightarrow a$, and (ii) the cells below the diagonal correspond to unfeasible FDs like $b \rightarrow a$ with $|b| < |a|$. Among the cells above the diagonal of Z , we can avoid checking those corresponding to transitive FDs by applying the following exploration strategy:

- *Rule 1:* First check the (unchecked) cells $Z[a, b]$ such that $|b|$ is maximum and, among them, give priority to the one with minimum $|a|$.
- *Rule 2:* If the FD corresponding to $Z[b, c]$ is found to be true, set to true all the FDs corresponding to cells $Z[*, c]$ such that $Z[*, b]$ holds.

To understand why Rules 1 and 2 avoid checking transitive FDs, consider FDs $a \rightarrow b$ and $b \rightarrow c$, which transitively imply $a \rightarrow c$. Then it must be $|c| \leq |b| \leq |a|$, so due to Rule 1 the check of $a \rightarrow c$ is scheduled after those of $a \rightarrow b$ and $b \rightarrow c$. But since $b \rightarrow c$ holds, Rule 2 sets $a \rightarrow c$ to true before it is checked.

According to the previous enumeration rule, the number of candidate FDs that must be verified depends, given the number of attributes, on the number of transitive FDs in R . The worst case arises when no transitive FDs hold between the attributes in R , because all the cells in the upper-right half of Z (i.e., $|R| \times (|R| - 1)/2$ cells) must be checked. The best case takes place when the attributes of R are involved into a linear hierarchy, because the number of checks drops to $|R| - 1$. Considering that the complexity of TANE is determined by its enumeration strategy and that TANE checks the FDs in linear time, the complexity of our approach turns out to be $O(|r| \cdot |R|^2)$ and $O(|r| \cdot |R|)$ in the worst and best cases respectively.

The enumeration strategy described above for traditional FDs relies on the ordering of attributes. Unfortunately, when working with AFDs, we must allow some tolerance on attribute cardinalities (hence, on the ordering of attributes) to accommodate possible errors in data. Consider two attributes a and b such that $|a| \gtrsim |b|$. If we were searching for FDs, we would check for $a \rightarrow b$ and not for $b \rightarrow a$ ($Z[b, a]$ lies in the lower-left part of Z and would be skipped). Conversely, when looking for AFDs, we must also consider the possibility that the higher cardinality of a is due to some errors in data; in other words, we must also check for $b \rightsquigarrow a$. In practice, this situation may occur if $|a| - \epsilon < |b| < |a|$. So, to preserve the correctness of our enumeration strategy when dealing with AFDs, we must check both cells $Z[a, b]$ and $Z[b, a]$ whenever $abs(|a| - |b|) < \epsilon$. Obviously, as a side effect, our pruning capability will be slightly reduced since some more cells need to be checked; however, the best and worst complexity remain unchanged.

As mentioned at the beginning of this section, in this phase our goal is to detect the FDs holding between hubs related by a link l , which we actually achieve by detecting the AFDs involving the foreign keys in l . Specifically, given dv-schema $\mathcal{V} = (T, F)$, let $l \in T_L$ be a link that connects hubs $h_1, \dots, h_n \in T_H$, which means that l includes n foreign keys, k_1, \dots, k_n , where k_i references hub h_i . Considering Definition 22, this already implies $l \rightarrow h_i$ for $i = 1, \dots, n$. Additionally, we will say that $h_i \rightarrow h_j$ ($1 \leq i, j \leq n, i \neq j$) if $k_i \rightsquigarrow k_j$. All the FDs determined are stored into a metadata repository, to be used at the next step for md-schema discovery and ranking. Note that, with reference to the complexity of detecting these AFDs, it is $|R| \equiv n$ and $|r| \equiv |l|$.

Example 26 *In our sale example, we can realistically assume that an order is made by one customer and that a customer belongs to one class. A customer normally issues several orders, each normally including several lines. Finally, the company will reasonably have more customers than employees. So, for instance, within link L_CustOrder it must be $|OrderSID| > |CustomerSID| > |EmployeeSID|$. The first AFD checked is $OrderSID \rightsquigarrow CustomerSID$, which is found to be true. Then $CustomerSID \rightsquigarrow EmployeeSID$ is checked, and we assume it does not hold. Finally, $OrderSID \rightsquigarrow EmployeeSID$ is checked, and again we assume that this does not hold in our application domain (i.e., several employees may be involved in the same order). We assume that overall, based on the data stored, two additional FDs are discovered for the sale dv-schema, namely $H_Order \rightarrow H_Customer$ and $H_Customer \rightarrow H_Class$ (a customer belongs to one class). These two FDs are shown in thick dotted lines in Figure 5.1.*

5.1.4.2 Md-Schema Discovery and Ranking

This process determines which elements of the source dv-schema are candidate to play the role of fact and, for each of them, creates an md-schema. Since the number of candidate facts may be large, the corresponding md-schemata are heuristically ranked before they are presented to the user.

Algorithm 7 *MDSConstruction*(\mathcal{V})

Require: A dv-schema $\mathcal{V} = (T, F)$ **Ensure:** A set of md-schemata $\{\mathcal{M}_l\}$

```
1: for all  $l \in T_L$  do ▷ For each potential fact  $l$ ...
2:    $A \leftarrow \{l\} \cup BusAttr(l)$ 
3:    $E \leftarrow \{\langle l, a \rangle \mid a \in BusAttr(l)\}$ 
4:    $\mathcal{M}_l \leftarrow (A, E)$  ▷ ...initialize the md-schema with fact  $l$ ...
5:   for all  $h \in T_H \mid \langle l, h \rangle \in F$  do
6:      $\mathcal{M}_l \leftarrow Explore(\mathcal{V}, \mathcal{M}_l, l, h)$  ▷ ...and build a DAG
7: return  $\{\mathcal{M}_l\}$ 
```

5.1.4.3 Candidate Selection

The selection of candidates is based on two specific features of the data vault model:

- A satellite s contains a foreign key referencing the associated hub or link t , which means that each tuple of s is related to exactly one tuple of t ($s \rightarrow t$) but several tuples of s are associated to the same tuple of t . However, since satellite are normally used to historicize attribute values, we can safely assume that, at each point in time, at most one tuple of s is valid for each tuple of t , i.e., that $t \rightarrow s$.
- A hub h is connected to at least one link l (unless it is disconnected from all other business concepts, in which case it is most probably not a fact candidate), and $l \rightarrow h$.

It follows that, for each satellite and hub in a dv-schema, there exists a link from which that satellite or hub can be reached through at most two FDs (in case of a satellite s of a hub h , it is $l \rightarrow h \rightarrow s$). So, since the algorithm we will use to build an md-schema for each fact navigates FDs, we can restrict the set of fact candidates to the set T_L of links without loss of generality.

5.1.4.4 Md-Schema Construction

The goal of this step is to automatically build, for each candidate fact (i.e., for each link) a draft md-schema starting from the dv-schema and from the additional FDs previously discovered. To this end, all the FDs (both those explicitly modeled by the dv-schema and the additional ones discovered by accessing data) must be “navigated” starting from the candidate fact, to build a DAG of attributes that will then be ranked and enriched in the next phase to become an md-schema.

The pseudo-code for building draft md-schemata is sketched in Algorithms 7 and 8. Algorithm 7 iterates on all links in the source dv-schema. For each link l , it initializes a draft md-schema \mathcal{M}_l with fact l , adds the attributes of the satellites of l (if any), and triggers procedure *Explore* to recursively build a hierarchy for each hub connected to l .

The goal of Algorithm 8 is to extend \mathcal{M}_l by “exploring” hub h . First it creates a node labelled with the business key of h , k , and attaches it to the previous node g (lines 1-3). All

Algorithm 8 *Explore*($\mathcal{V}, \mathcal{M}_l, g, h$)

Require: A dv-schema \mathcal{V} , an md-schema \mathcal{M}_l , a node $g \in \mathcal{M}_l$, and a hub $h \in T_H$
Ensure: An (extended) md-schema \mathcal{M}_l

```

1:  $k \leftarrow \text{BusKey}(h)$ 
2:  $A \leftarrow A \cup \{k\}$ 
3:  $E \leftarrow E \cup \{\langle g, k \rangle\}$ 
4: if  $h$  not explored yet then
5:   Mark  $h$  as explored
6:    $A \leftarrow A \cup \text{BusAttr}(h)$ 
7:    $E \leftarrow E \cup \{\langle k, a \rangle \mid a \in \text{BusAttr}(h)\}$ 
8:   for all  $l \in T_L \mid \langle l, h \rangle \in F$  do
9:      $Z \leftarrow \{z \in T_H \mid z \neq h \wedge \langle l, z \rangle \in F\}$ 
10:    if  $\exists z \in Z \mid h \rightarrow z$  then
11:       $A \leftarrow A \cup \text{BusAttr}(l)$ 
12:       $E \leftarrow E \cup \{\langle k, a \rangle \mid a \in \text{BusAttr}(l)\}$ 
13:      for all  $z \in Z \mid h \rightarrow z$  do
14:         $\mathcal{M}_l \leftarrow \text{Explore}(\mathcal{V}, \mathcal{M}_l, k, z)$ 
15: return  $\mathcal{M}_l$ 
  
```

▷ Add business key k ...
 ▷ ...and its incoming arc to \mathcal{M}_l
 ▷ Add satellite attributes...
 ▷ ...and their arcs to \mathcal{M}_l
 ▷ For each link l connected to h ...
 ▷ ...find other hubs connected to l
 ▷ Add satellite attributes of l to \mathcal{M}_l
 ▷ Use additional FDs to trigger recursion

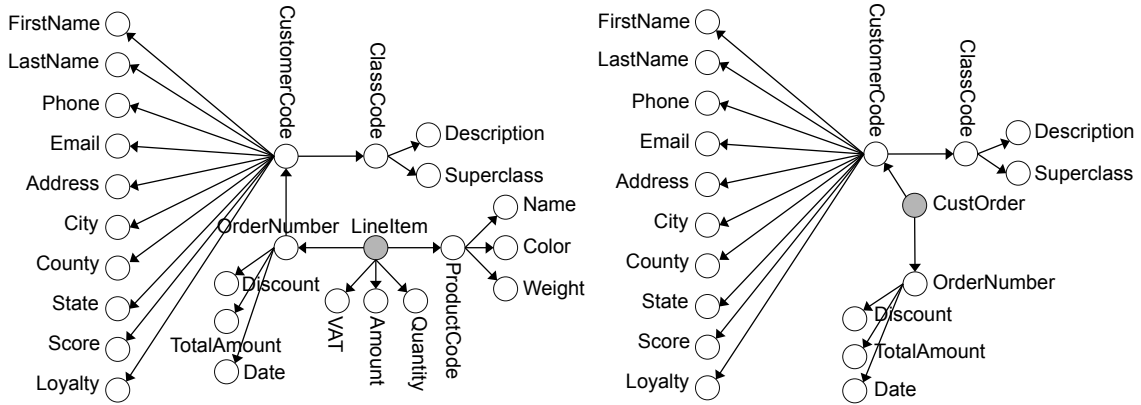


Figure 5.3: Draft md-schemata of facts L_LinItem and L_CustOrder

the attributes of its satellites are then attached to k (lines 6-7). To continue exploration, the algorithm now checks if there are additional FDs from h to some other hub (lines 8-18). In particular, if there is an FD to at least one hub z through link l , before triggering recursion on z (line 18) all the satellite attributes of l must be added as children of k (lines 12-15). Repeated explorations of parts of the source dv-schema when the same hub is reached twice from different directions are avoided by marking a hub as explored when it is reached for the first time (lines 4-5).

Example 27 *In our sale example, three draft md-schemata are built for facts L_LinItem, L_CustOrder, and L_CustClass (two of them are shown in Figure 5.3). To better describe the construction algorithms, we follow them step by step with reference to the first md-schema (the one of fact L_LinItem). Firstly, procedure MDSConstruction creates the fact node (in grey) and its satellite children VAT, Amount, and Quantity. Then, procedure Explore is called twice for hubs H_Order and H_Product. In the first case, Explore starts by creating node OrderNumber (line 2), connecting it to node LinItem (line 3), and adding the two satellite children (lines 6-7). Then, since link L_CustOrder is connected to H_Order and FD $H_Order \rightarrow L_CustOrder$ holds (lines 8-12), Explore is called for hub H_Customer (L_CustOrder has no satellites, so lines 13-15 have no effect). When Explore is called*

for `H_Customer`, 10 satellite children are added, then the procedure is called again for hub `H_Class`. Similarly for hub `H_Product`.

5.1.4.5 Ranking

At the previous step, for each candidate fact l a draft md-schema $\mathcal{M}_l = (A_l, E_l)$ has been constructed. Now, the md-schemata obtained are ranked to support the user in choosing the most comprehensive ones.

The ranking of md-schemata is based on a linear combination of three heuristics that consider, for each candidate fact, (i) its cardinality, (ii) the number of potential measures, and (iii) the number of potential attributes. While heuristic (i) is extensional in nature because it is data-based, the remaining two (which are partially inspired by [119]) are intensional because they consider the dv-schema.

- (i) Business events are dynamic in nature and generated with high frequency, so the tables that store them have a large number of instances. A link $l \in T_L$ is more likely to be a fact if it has high cardinality [125].
- (ii) Business events are quantitatively described by several measures, i.e., numerical attributes. We quantify the probability that a link l is a fact as the number of numerical attributes that are functionally determined from l , i.e., as the number of numerical attributes in $A_l \setminus l$.
- (iii) At query time, business events are selected and aggregated by users using the dimensions and their levels. We quantify the probability that a link l is a fact as the number of non-numerical attributes that are functionally determined from l , i.e., as the number of non-numerical attributes in $A_l \setminus l$.

Note that the last heuristic closely recalls the *connection topology value*, defined in [119] as the number of entities that can be (either directly or indirectly) reached within an Entity-Relationship diagram by starting from the fact and recursively navigating many-to-one relationships.

Example 28 *Heuristics (ii) and (iii) for the three sales draft md-schemata return the following values for the number of numerical and non-numerical attributes: 7, 17 (`L_LineItem`); 1, 13 (`L_CustClass`); and 3, 14 (`L_CustOrder`). Considering that the cardinality of link `L_LineItem` will surely be quite higher than the one of the other two links (the cardinality of `L_CustClass` is at most the same of `H_Customer` and a customer normally issues several orders; the cardinality of `L_CustOrder` is at most the same of `H_Order`, and an order normally has several lines), we can conclude that the top ranked md-schema is the one of fact `L_LineItem` whatever the weights of the linear combination of the three heuristics.*

5.1.4.6 Md-Schema Enrichment

The last phase starts with the user selecting one or more draft md-schemata of interest, supported by the ranking previously obtained. Some editing is normally necessary at this stage, typically to remove uninteresting attributes from the md-schema. Specific situations such as one-to-one relationships between hubs and multiple arcs entering the same node in the md-schema must be also dealt with, as discussed in [1]. Then, measures are chosen among the numerical attributes in the md-schema. Finally, all the direct children of the fact that have not been chosen as measures are labelled as dimensions, which completely defines the output md-schema.

One further way to enrich the md-schema by making its hierarchies more faithful to the application domain is to search for FDs hidden in satellites. In a data vault, the grouping of attributes in satellites is generally oriented more to cheap maintainability and querying than to normalization. For instance, in our sale example, satellites `S_CustAddress` and `S_Employee` contain attributes `City`, `County`, and `State` that are obviously related to one another, so the following FDs hold: `City` \rightarrow `County` and `County` \rightarrow `State`. While in this simple case it will probably be easy for the user to detect these FDs and manually add them to the md-schema as a part of editing, in other cases the user may be unsure of whether an FD holds or not, so automating FD detection is highly desirable. How to cope with this issue is the subject of the remainder of this section.

When dealing with satellites, we must keep in mind that data vaults are natively oriented to storing time-variant data, so we can expect that a single tuple of a hub (or link) is related to several tuples in a connected satellite, one for each version of data. As a consequence, if we used traditional FD (or even AFD) discovery techniques on the `S_CustAddress` satellite for instance, we might not find the FD `City` \rightarrow `County` in case a city has been moved to a different county at some time. The most natural way to formalize this problem is by using *temporal FDs* [127]. Intuitively, in its simplest form, a temporal FD $a \xrightarrow{T} b$ is an FD that is valid within a time-variant relation at any time slice. In our example, though `City` \rightarrow `County` may be not true overall, it must be true at any time slice, so `City` \xrightarrow{T} `County`. If we also consider the possibility that a temporal FD holds on *most* tuples of a satellite, we have *approximate temporal FDs* (ATFDs) [128], i.e., FDs that are valid for specific time periods and possibly subject to errors.

In [128], the detection of ATFDs is achieved through some preprocessing that turns them into AFDs, that can then be discovered using TANE [110]; this preprocessing is made by temporally grouping either on sliding windows or on temporal granules. The type of temporal evolution that is relevant to the Starry Vault approach is captured by grouping on temporal granules, i.e., by partitioning the values in the domain of the time attribute into indivisible groups called *granules*. Examples of possible granularities are hours, days, months, etc. To understand how this preprocessing works, consider a table r with schema $R = v \cup W$, where v and W are respectively a time attribute and a set of other attributes.

Table 5.1: Sample data for the S_CustAddress satellite

CustomerSID	Timestamp	Address	City	County	State	Granule
1	1-1-2015	Gandalf Street	Minas Tirith	Gondor	Middle-Earth	January 2015
1	1-6-2015	Gandalf Street	Minas Tirith	Rohan	Middle-Earth	June 2015
2	1-3-2015	Frodo Road	Minas Tirith	Gondor	Middle-Earth	March 2015
2	1-6-2015	Frodo Road	Minas Tirith	Rohan	Middle-Earth	June 2015

A new relation is created from r by adding a granule attribute g whose domain is the set of granules included in the time-span described by the instances of r . Intuitively, for each tuple in r , the value of v is converted into its corresponding granule identifier. The new relation obtained is then processed with TANE to discover AFDs of type $g \cup X \rightsquigarrow Y$, with $X, Y \subseteq W$.

To apply this technique to a satellite s , we consider its timestamp and its business attributes $BusAttr(s)$, thus neglecting its foreign key. After the the granule attribute g has been added, the ATFDs can be computed using the following variation of the enumeration strategy proposed in Section 5.1.4.1:

- Instead of searching for AFDs of the form $a \rightsquigarrow b$, we consider all AFDs of the form $ga \rightsquigarrow gb$ (i.e., due to the decomposition rule, $ga \rightsquigarrow b$), where $a, b \in BusAttr(s)$. This means that the ordering for rows and columns in matrix Z will be defined by the cardinality of ga rather than by that of a .
- The pruning rule seen in Section 5.1.4.1 would avoid checking all AFDs $b \rightsquigarrow a$ with $|a| > |b| + \epsilon$. Conversely, in this case a check can be avoided if $|ga| > |gb| + \epsilon$.

It is easy to see that the size of matrix Z is still $|R|^2 \equiv |BusAttr(s)|^2$ since we are just adding the granule attribute g to both the left- and right-hand sides of the AFDs. As to the correctness of the pruning rule, we remark that the error $e(ga \rightsquigarrow b)$ is defined as the minimum number of distinct values of gab that must be removed to enforce $ga \rightarrow b$; therefore, an error ϵ can at most impact on the cardinality of b for an amount equal to ϵ itself.

Example 29 Consider the sample data for the S_CustAddress satellite in Table 5.1, showing that on June 1 the city of Minas Tirith has moved from the Gondor county to that of Rohan. If we considered traditional FDs or even AFDs, we would probably conclude that one city can belong to different counties (i.e., that $City \rightarrow County$). Let us consider ATFDs instead, choosing for instance a month granularity. The table created after preprocessing has the new column Granule, and it is easy to verify that $Granule \text{ City} \rightarrow County$, so $City \xrightarrow{T} County$. The final md-schema obtained from the draft md-schema of fact L_LineItem (Figure 5.3, top) is depicted in Figure 5.4 using the DFM notation [1]. Attribute OrderNumber has been deleted and all numerical attributes have been chosen as measures; besides, the missing FDs between City, County, and State have been added.

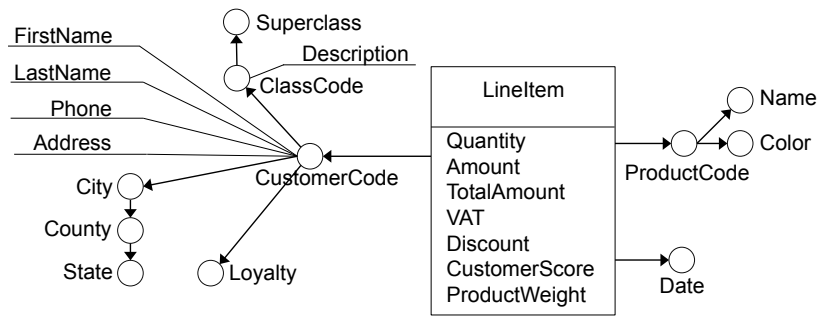


Figure 5.4: The enriched md-schemata of fact $S_LinItem$ (descriptive attributes, non usable for aggregation, are underlined)

5.1.5 Wrapping up Starry Vault

In this work we have described the Starry Vault approach for detecting a multidimensional schema out of a source data vault. Both schema-based and data-based FDs are used to this end, with a small intervention by the user. In particular we have shown how to use extensional techniques for discovering hidden FDs, with some tolerance to errors in data and taking into account the temporal aspects related to historicization, to automatically deliver the md-schemata that better fit the business domain. To this end we have proposed an original exploration strategy that allows to significantly reduce the complexity of the TANE algorithm when applied to simple ATFDs. To the best of our knowledge, ours is the first approach that adopts advanced types of FDs to infer md-schemata.

Automatic derivation of md-schemata is a widely explored topic in the DW literature; nonetheless we believe that it is worth reconsidering it in the era of big data and data science, in which the need for on-the-fly analyses creates a strong requirement for a smarter design process. Based on these considerations, our future work on this topic will be mainly focused on investigating ad hoc techniques to support the data scientist in discovering a multidimensional structure even in situations in which the source data are poorly-structured or schemaless, as is the case for document databases.

5.2 Multidimensional Modeling Over Sensor Data

The availability of sensor data is projected to mushroom in the coming years, fueled by growth in both “explicit” sensor systems and the increased capabilities of Internet of Things (IoT) devices. Sensing provides stakeholders with data upon which to base decisions that have increasing importance in managing an enormous range of both user-centred and societal problems.

Much of the value of sensor data comes from examining the trends and variations that occur over long timescales – where “long” ranges from hours to years depending on the context of data collection. This is the point at which sensor data becomes “big data” that can be analysed using machine learning and other tools from data analytics. The goal of such approaches is to extract further global information from the time series, over and above the local information (in space and time) that may be used operationally. That is to say, the value of a sensor dataset may come both from short-term, tactical use of the information it contains, but also from long-term, strategic uses that permit identification of trends and features that are relevant to future contingency planning and *post facto* analysis of operations. In many contexts these multiple uses massively increase the importance and value of any data collected, making the financial case for instrumenting environments far stronger.

There is, however, a challenge implicit in this scenario. Making use of data a long time after it has been collected implies a number of things about the data handling system used to manage it. The data must be *searchable* and *findable* in order to be retrieved when required; it must be *parseable* in terms of extracting the data types and ranges used for representation; and – most importantly — it must be *contextualised* into the environment in which it was collected. By “contextualised” we mean that it must be possible to retrieve the details of the sensors, their installation and operation, as well as simply their data.

What makes contextualisation important is the exposed lifecycle of a typical sensor. By way of example, consider an air-quality sensor deployed to measure pollutant gases in an industrial setting. The data acquired from this sensor will typically consist of a stream of numbers, and clearly we need to retain the units and range of these data points for later analysis. But the sensor’s readings also have metadata such as precision, accuracy, sampling frequency and the like, which must also be recorded if we are to have confidence in analysis. Furthermore the sensor will deteriorate the longer it is in the field, both through mechanical ageing and (in many cases) chemical and physical changes in the sensor itself, both of which lead to decalibration. It may become occluded by plant life, biofilmed by bacterial action, and dirtied by the actions of the very pollutants it is deployed to measure – all of which will affect its behaviour, and so the values in the time series dataset. Cleaning the sensor, whether *ad hoc* or according to a schedule, may re-set some of these influences and return the sensor to (some part) its pristine state. Unless we record *all* (or at least a large part) of this context, the dataset will become increasingly misleading when

analysed.

Similar challenges are well-known in database technology, where the techniques of *data warehousing* have been developed to ensure that data held for the long term remain robustly analysable. These techniques have not, to our knowledge, been applied in the context of sensing, and in this paper we aim to address this omission by developing data warehousing schemata for sensor data. We develop an extensible model that allows for the capture of contextual metadata alongside sensor datasets, and show how this can be used to support the long-term analysis of historical datasets. Remarkably, while our proposal is aimed at long-term data management with storage and explorative analyses in mind, it is fully compatible with the adoption of other problem-specific analytical solutions. Actually, the proposed architecture and multidimensional schemas are meant to provide a solid foundation on which other analytical modules can rely to easily access integrated and consistent data. We argue that such an approach is essential if we are to increase the value we can extract from collected data, as well as maximise our confidence in the strategies we use such datasets to support.

Section 5.2.1 reviews some recent related work in sensor data representation. Section 5.2.2 presents our baseline reference framework for sensor context, from which Section 5.2.3 then derives multidimensional schemata representing different modeling and analysis scenarios. We exercise the model in Section 5.2.4 with a case study, which we then use in Section 5.2.5 to draw some conclusions and future directions.

5.2.1 Related Literature

The topic of sensor networks modeling has been tackled from different perspectives, the result is a plethora of approaches whose main goals are either enabling interoperability between processes, or easing the analysis of data and the design of applications. However, to the best of our knowledge, there has been very limited research on multidimensional modeling of data produced by sensor networks. Indeed, while in many scenarios data warehouse technologies are part of the proposed architecture, the details on how the different data cubes have been modeled is omitted. An example of this can be found in [129], where the authors present a framework for fault-diagnosis that employs an enterprise data warehouse as an integrated repository for monitoring data. Scriney et al. [130] propose instead an approach whose goal is much closer to ours. The authors present a methodology to obtain multidimensional data cubes starting from XML and JSON data sources. The methodology is composed by two main steps: the first one consists in the conversion of a XML or JSON schema definition into a novel graph structure called StarGraph; while the second step aims at populating the data cubes with the events coming from the network. The main limitation of this approach is that it works only with XML and JSON data sources that provide a schema definition. The idea of supporting, or even automating, the design of data warehouses is not new, as a matter of fact, early

approaches date to the '90s [131], and even recently [132] the research effort is still ongoing. However, the issue with these techniques is that they are often tied with specific types of data sources (e.g., XML/JSON for [130] and data vaults for [132]). Furthermore, we argue that, while automatic approaches are more flexible, the produced results are still of lower quality with respect to schemata that have been manually designed by domain experts.

Among the approaches that focus on the interoperability aspect, the SensorML [133] framework provides a set of XML schemata to describe sensor networks, from the physical systems to the measurements and processes involved. Another work aiming at improving the interoperability aspect of IoT is presented in [134]; the result is a semantic model approach where the resources of the network are exposed through web services, thus enabling semantic search and reasoning over both devices and the data they provide. BOnSAI [135] takes instead a more focused approach by specializing other ontologies (e.g., OWL-S [136]) for the ambient intelligence scenario; once again, the goal is that of creating a machine interpretable representation of sensor networks to improve interoperability.

An interesting tool to support the design of applications over sensor networks is SEM [137], which proposes a framework that represents the sensor network through two different metamodels, namely the functional metamodel and the data metamodel. The former describes the network as a set of exposed services, while the latter describes the available data sources. Through these two metamodels the application designer can obtain a high-level view of the network to better manage its complexity.

5.2.2 Reference Architecture and Domain Model

In this section we lay down the reference framework, which is composed by the functional architecture and the domain model, respectively presented in Section 5.2.2.1 and Section 5.2.2.2. This framework will be later used as a reference for all the proposed multidimensional schemata and use cases.

5.2.2.1 Functional Architecture of the Analytical System

Figure 5.5 shows the proposed functional architecture that we will use to contextualize the multidimensional schemata presented in Section 5.2.3 and the use cases presented in Section 5.2.4. We start by describing the processes and how they interact, then we proceed to explain how we envision its implementation from a technological point of view.

The architecture in Figure 5.5 is separated into two different macro areas by a vertical dashed line; on the left there is the sensor network (SN), while on the right there are all those processes that elaborate and store the events coming from the network. In the following we describe the main processes of the architecture.

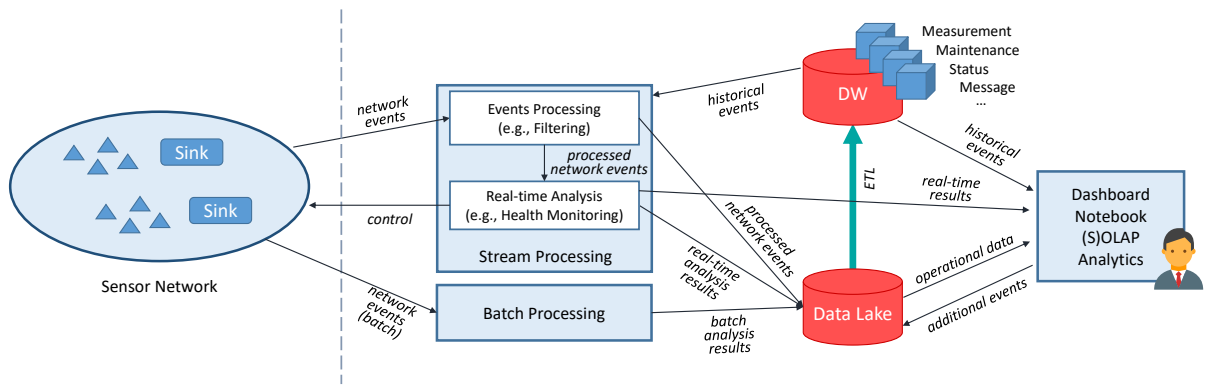


Figure 5.5: Functional architecture of the analytical system for sensor networks

- *Sensor Network* is a collection of (typically computationally-constrained) nodes collecting data from the physical environment. The nodes often make use of mesh or other network protocols and use spanning trees for collecting and aggregating data to a small set of “sink” nodes – none of the details of which affect the current work. We simply assume that the data collected is somehow returned to the cloud for processing. (It is worth noting in passing that many *sensor* networks are actually *sensor/actuator* networks that need to make local control decisions. Our architecture supports these too, and would probably want to record *both* the sensed data *and* the decisions made as a result, for later analysis.)
- *Stream Processing* is one of the two intermediaries between the SN on one end, and users and storages on the other. The typical output of the SN is considered as a stream of events (e.g., temperature measurements) that must be processed in a timely fashion. In the presented architecture this process is decomposed into two different subprocesses, namely, *Events Processing* and *Real-time Analysis*. The former includes simple operations such as filtering and smoothing of the input events, while the latter uses the output of the former and the historical data in the DW to perform more complex (but still in a streaming fashion) analyses, such as forecasting, network monitoring, etc. As represented through black arrows, both the processed events and the results of the real-time analysis are stored in the *Data Lake*; furthermore, the *Real-time Analysis* process can directly interact with the SN, for instance for activity regulation purposes.
- In certain scenarios it might be either necessary (e.g., due to the lack of connection) or sufficient (e.g., because the timeliness constraint is relaxed) to treat the events coming from the SN as a sequence of batches. In these cases the events produced by the SN are handled by *Batch Processing* rather than by *Stream Processing*. By relaxing the timeliness constraint it is possible to perform more complex computations that would not be possible in a real-time scenario. Of course, these two styles of processing are not mutually exclusive and, indeed, a mix of the two can be employed to have both real-time results through *Stream Processing* and higher quality results

from *Batch Processing*.

- All the results of *Stream Processing* and *Batch Processing* are loaded into a *Data Lake*, i.e., a repository capable of storing as many (typically unstructured) data as necessary. Data lakes ensure a high-speed ingestion rate and can store data in different formats ranging from raw data to transformed data which is used for various tasks including reporting, visualization, analytics, and machine learning.
- Although a data lake is suitable to support analysis by skilled users (i.e. data scientists), it falls shortly in a classical business analytics scenario, where users with limited technical capabilities (i.e. business users) require a more structured solution. This is where the *Data Warehouse (DW)* comes into play. A Data Warehouse is a multidimensional collection of data that supports decision-making process. Data warehouses store historical, consistent and integrated information that ensures high quality analyses [138].
- The users of the proposed architecture can be grouped into three main figures, namely the *data scientist*, the *BI user*, and the *technician*. With data scientist we refer to a user that has both high IT and analytical skills. Due to their skills, this kind of users can exploit both the well-structured and organized data stored in the DW and the less consistent and structured contents of the data lake to carry out their analyses. A step below in terms of IT skills there is the BI user, who has broad knowledge about the domain of the data but needs accessible analytical tools (e.g., Tableau ¹, Micro Strategy ², etc.) to accomplish her goals. Due to her limitations, the BI user only has access to structured and well-organized data, which are stored in the DW. Finally, the technician is a user who has very specific domain knowledge but lacks IT skills. This figure comprises for instance any maintenance worker who might be interested in monitoring in real-time the status of the network. The capabilities of the user interface will change depending on the involved type of user and the type of final application. Typically, data scientists and skilled domain users prefer having more control on data and more powerful tools (e.g., Notebook tools, OLAP & SOLAP interfaces, and analytics), while domain users prefer simple and focused information such as KPIs that can be obtained through dashboarding tools. We will not discuss in details the wide range of possible applications the proposed architecture can be used for, instead, in the following we will discuss in depth the domain model that is needed to support them.

The typical data flow in the architecture sketched in Figure 5.5 can be described as follows. Every event sent by a device in the SN to the analytical system is processed either in a streaming or in a batch fashion. In the former case the results of the computation can be both directly presented to the end user and also stored in the data lake for future use;

¹<https://www.tableau.com>

²<https://www.microstrategy.com>

while in the latter case the results are simply stored in the data lake. When processing events in real-time the system might also react and respond to control the network, thus creating a feedback loop. Once the results of the processing have been stored in the data lake, they can be directly accessed by users (specifically the data scientist) and applications for further analyses and elaborations. Furthermore, periodically (e.g., weekly) the data in the data lake is used to feed the DW through the ETL process. Once in the DW, the events become also available to BI users by means of OLAP analysis tools.

From a technological point of view the presented architecture can be implemented as follows. As to the implementation of the SN, we remark that the focus of this work is the analytical system operating with the data coming from the SN, thus, the only requirement that we place on the SN is that it must be able to send data regarding the sensed events. The most promising technology for the analytic system is the popular distributed platform Hadoop³, which provides distributed storage and processing capabilities. The choice of a distributed system is quite crucial as a traditional centralized approach cannot generally accommodate the high amount of data that a SN might produce. Furthermore, the Hadoop platform is very flexible and can support many other frameworks that are vital for some of the above described tasks. Specifically, a couple of examples of frameworks for streaming processing duties are Apache Storm⁴, Apache Spark[139], and Apache Flink⁵. Spark and Flink, can also take on batch processing tasks. As for storage, the system must be able to aptly handle both unstructured and structured data. The simplest and most general solution is accommodating all kinds of unstructured data as simple distributed files in the HDFS filesystem[140] and structured data through Apache Hive[141]. An alternative to using HDFS for all unstructured data is that of adopting more specialized storage types, which generally fall into the NoSQL[142] category; however, we avoid discussing these solutions as they are out of the scope of this work.

5.2.2.2 Domain Model

In the following we briefly describe the main concepts represented in Figure 5.6, which shows the domain model of an analytical system for sensor networks using the notation of a UML class diagram. When feasible, we also use the SensorML framework [133] as a reference and link the described concepts to the corresponding ones in SensorML.

- *Agent*: in this context, with agent [143] we refer to anything that can be viewed as perceiving its environment and that can act upon it. Specifically, we distinguish the following types of agent.
 - *Physical Device, Logical Device, and Assigned Device*: a physical device is any device inherently associated to a physical object. This concept covers

³<http://hadoop.apache.org>

⁴<http://storm.apache.org>

⁵<https://flink.apache.org>

- *Measurement*: it refers to an observation of a property made at a specific time. Usually, but not necessarily, a measurement refers to a specific location. Furthermore, a measurement is usually the result of some kind of transformation (e.g., smoothing); this aspect is modeled by the *Transformation* class. A typical example of measurement is the temperature sensed by a thermometer. Additionally, a measurement is characterized by an estimated accuracy, which defines how reliable that particular measurement is. Indeed, adverse environmental conditions, wear, and other aspects might adversely influence the accuracy of sensors. For these reasons, when available, an estimate of the accuracy of the measurement can greatly improve the quality of the analyses. In SensorML, the definition of a measurement is quite loose and is not explicitly modelled, however, the underlying meaning remains the same.
- *Maintenance Operation*: it refers to any operation (e.g., calibration, component substitution, etc.) performed either on a physical device or on a process by another agent. In SensorML, *Maintenance Operation* would be included in the general definition of *Event*, which is used to track the history of a device.
- *Status Check*: it represents a check performed by an agent over another agent, specifically over either a process or a physical device. A check can refer to many different types of assessments, for instance, the assessment of the battery level, the confirmation of a malfunctioning device, or even a notification of a device that goes into battery saving mode.
- *Message*: it refers to any message sent by an agent to one (i.e., unicast) or many (i.e., multicast) other agents.

5.2.3 Multidimensional Schemata

Figure 5.7 shows the four multidimensional schemata derived from the domain model presented in Section 5.2.2.2 using classical techniques for multidimensional design [138]. The notation we adopt is that of the DFM (Dimensional Fact Model), where the box represents a fact to be analysed, circles represent its dimensions, and measures are included in the lower part of the box; arcs represent many-to-one relationships to be used for aggregation (see [138] for further details). Among the concepts represented in the class diagram in Figure 5.6, those whose dynamic nature makes them suitable to be modeled as facts are *Measurement*, *Maintenance Operation*, *Status Check*, and *Message*. Before describing these facts one by one, we make a few general observations related to all the schemata:

- The concept of agent is represented as a dimension, and its specialization into persons, processes, and devices (as of Figure 5.6) is translated in the DFM as an additional level **Agent Type**, whose domain has values 'Person', 'Process', and 'Device'. The

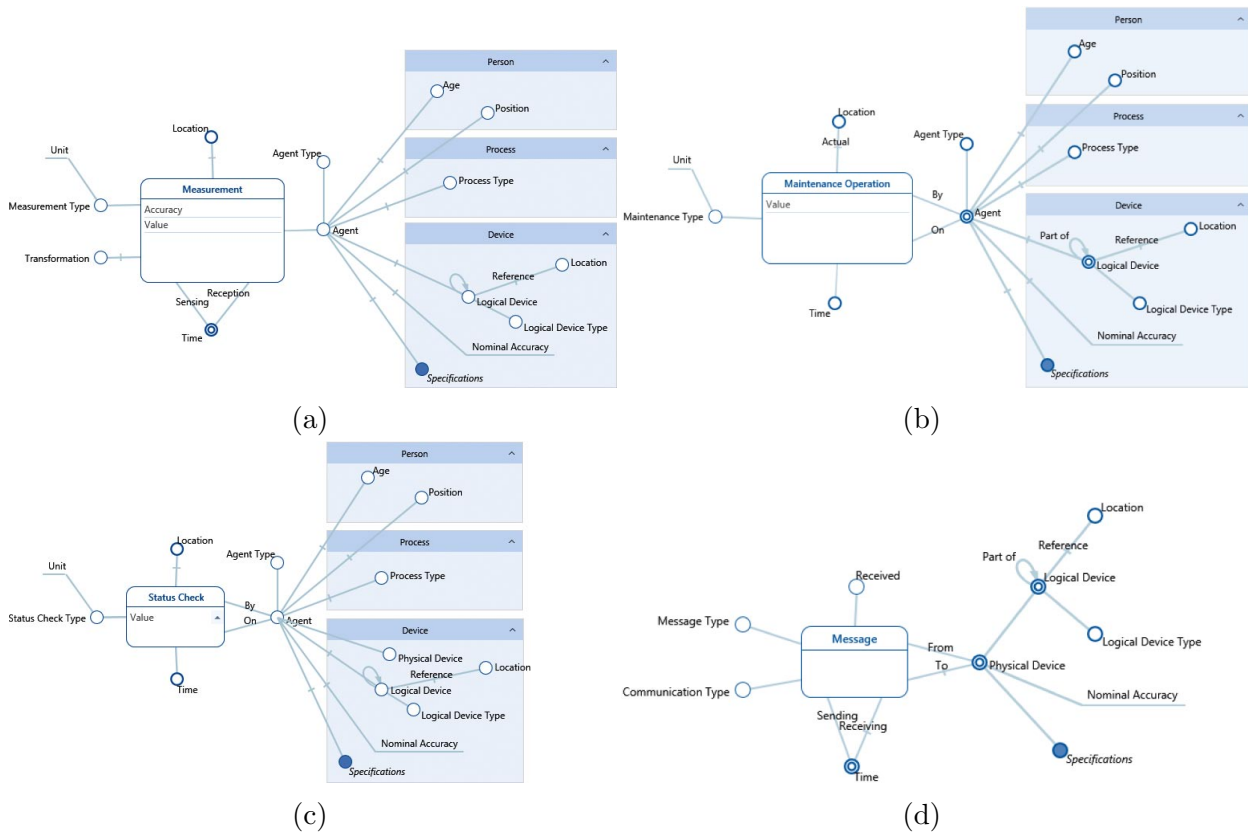


Figure 5.7: Multidimensional schemata for the measurement (a), maintenance (b), status (c), and message (d) events

levels that branch from **Agent** are optional and grouped into three different sets, one for each type of agent. Based on the value of **Agent Type**, only one group of levels will be meaningful, e.g., if **Agent Type** = 'Person', then only **Age** and **Position** will take a value.

- As to devices, the domain model shows that one assigned device is related to one physical and one logical device. So, the agents corresponding to assigned devices can be grouped by either **Logical Device** or **Physical Devices**. The recursive aggregation of logical devices is represented in the DFM using a recursive hierarchy (the looping arrow on **Logical Device**).
- As done for the specialization of **Agent**, the two different types of logical devices (i.e., stationary and mobile) are differentiated through an additional level called **Logical Device Type**, while the **Location** level is optional as it is relevant only for stationary devices.
- The **Location** level also deserves a clarification, as it can be used in two different ways. The easiest way is that of giving it a simple categorical domain (e.g., {Rooftop, Control Room, ...}) to approximate the geographical positioning of the devices and events. The more complex (but also more expressive) way is that of employing

geographical coordinates and geometries that can be fully exploited through *Spatial OLAP* (SOLAP) technologies [144], which allow the user to perform specific OLAP operations tailored for spatio-temporal data and enable so-called *location intelligence*.

The first schema we discuss is **Measurement**, shown in Figure 5.7a. Each measurement event is described by two measures and is defined by six dimensions. The two measures are **Value** and **Accuracy**, which respectively describe the result of the measurement and its accuracy. Dimension **Measure Type**, together with the **Unit** level, defines the context of the measurement. For instance, if **Measure Type** = 'Temperature' and **Unit** = 'Celsius Degrees', then the meaning of **Value** is precisely defined as the temperature in Celsius degrees. The **Sensing Time** and **Reception Time** dimensions define the temporal aspects of the measurement. The remaining dimensions define where the measurement has been taken (**Location**), who took it (**Agent**), and what kind of processing has been applied to it (**Transformation**).

The **Maintenance** and **Status Check** schemata, shown in Figures 5.7b and 5.7c respectively, are quite similar to **Measurement**. Here the **Maintenance Type** and **Status Check Type** levels play the same role that **Measurement Type** plays in **Measurement**.

The last schema we discuss is **Message**, shown in Figure 5.7d. Differently from the other facts, messages regard only a specific type of agent, i.e., assigned devices, thus the **Agent** dimension used in all the other schemata has been replaced by **Assigned Device**. Furthermore, the **Message** fact has no measures, which means that a message event either exists or not. Other modeling choices worth discussing stem from the fact the not all sent messages are actually received and, additionally, a communication can be a broadcast, which means that we know who sent the message but not who is supposed to receive it. These two issues are addressed through the addition of the **Communication Type** and **Received** levels. A few examples that summarize the supported scenarios are described in the following.

- (i) A successfully delivered message sent from a device *A* to a device *B* is represented as an event with **Communication Type** = 'Unicast', **Received** = 'True', **From Physical Device** = 'A', **To Physical Device** = 'B', with the corresponding sending and receiving times.
- (ii) A failed attempt to deliver a message sent from *A* to *B* is represented as an event with **Communication Type** = 'Unicast', **Received** = 'False', **From Physical Device** = 'A', **To Physical Device** = 'B', with only the corresponding sending time.
- (iii) A broadcast message sent from *A* and received from *B* is represented as in the first example but with **Communication Type** = 'Broadcast'. Each message received is represented as a separate event.
- (iv) Finally, a broadcast message sent from *A* but not received by any other device is represented as a single event with **Communication Type** = 'Broadcast', **From Physical**

Table 5.2: Requirements for the air quality scenario

	Name	Description	Stakeholders	Main Data Sources
(1)	<i>Air Quality Measurements</i>	Quantitatively measure the presence of pollutants in the air in proximity of the facility	Facility, Civilians, Agencies	DW (Measurement)*
(2)	<i>Alerts for High Pollution</i>	Timely detect the presence of pollutants in quantities exceeding safe thresholds	Facility, Civilians, Agencies	Data Lake
(3)	<i>Environmental Conditions</i>	Monitor natural phenomena and properties such as wind, humidity, and temperature in proximity of the facility	Facility, Agencies	DW (Measurement)*
(4)	<i>Network Health Status</i>	Detect potentially malfunctioning parts of the network	Facility	Data Lake, DW (Status Check)
(5)	<i>Maintenance Operations</i>	Keep track of the maintenance operations executed on the network	Facility, Agencies	DW (Maintenance Op.)

Device = 'A', and the corresponding sending time. In this case the receiving device and time are not specified.

In the following section we present some examples of how these models can be used in a practical context to carry out several interesting analyses.

5.2.4 Case Studies

As examples of this framework in action, we consider the case of air quality monitoring at an industrial facility (see Section 5.2.4.1) and the case of landslides risk management in the Italian territory [145] (see Section 5.2.4.5). These two examples have been chosen due to their requirements, which are rich and varied. In both cases we start by discussing the operational requirements for a sensing system in terms of scientific and business challenges, then we describe a possible suite of sensors to address the problem, and finally we show how the data from this sensor network can be modelled in a DW.

5.2.4.1 Air Quality Monitoring

Several large industrial installations operate under tight safeguard and permission regimes designed to protect civilians near the plants as well as the broader environment. Typically a plant is licensed to emit particular maximum quantities of specific pollutants, and will incur fines or other sanctions for exceeding these limits. Plants will often be required to install systems to monitor their emissions, either directly as point of generation, or indirectly through wider sensing—or both. There may also be further monitoring performed by third-party or regulatory agencies to ensure compliance.

5.2.4.2 Requirements for Air Quality Monitoring

The main requirements identified for this scenario are listed in Table 5.2. For each requirement we give a brief description and list the related stakeholders and data sources. There are three main types of stakeholders, namely, the facility, the civilians who are not directly involved with the facility but who can be affected by its emissions, and finally the regulatory agencies that have the duty of checking whether or not the facility operates respecting environmental norms. Each requirement can be satisfied by accessing the data stored in the DW and in the data lake. For those requirements that need to access the DW we also specify which cubes are needed. Notice that, for some requirements (denoted with a star in Table 5.2), we list only the main required cube, however, other cubes might be useful to get contextual data; for instance, for Requirement (1) it could also be useful to know the status of the network to better assess the accuracy of the measurements.

As well as the obvious detection of leaks and excess emissions, plant managers, regulators, and other agencies have an interest in the long-term relationship of a plant to its environment. One example would be to detect the build-up of pollutants in the local environment even if the plant were operating as licensed. Another would be to implement a market for pollution with a view to encouraging plants to invest to reduce emissions. These scenarios are covered by Requirement (1) and (2) in Table 5.2. Specifically, the first requirement covers the monitoring of air quality in the long term, while the second covers real-time needs where timely alerts are mandatory (e.g., in case of leaks). Requirement (1) does not need up-to-date data, so it draws from the DW where data are of higher quality. On the other hand, Requirement (2) cannot afford using stale data, thus it draws directly from the data lake, where measurements are stored in real-time and at the finest level of detail.

One challenge often faced in wider sensing is the problem of attribution: given that a particular situation is observed, who was responsible for it? Suppose residents near a plant wake up one morning to discover a fine white powder coating their cars: what is the powder, where did it come from, and who is to blame? Answering these questions requires a combination of direct analysis (what is the powder?) and potentially the fusion of several data streams to determine possible causes: wind direction may exonerate some plants from consideration, for example, but this requires that such data is available, accessible, and reliable. All these challenges are covered by Requirement (3).

Beside the challenges strictly related to the management of measurements, the facility also needs to monitor the sensor network itself. Indeed, failures of the network must be detected and addressed as quickly as possible for safety reasons. Moreover, failures imply maintenance costs, which should be kept to a minimum. Requirements (4) and (5) respectively cover the issues related to the (both real-time and off-line) monitoring of the health status of the network and the issues related to the maintenance operations executed to keep the network running.

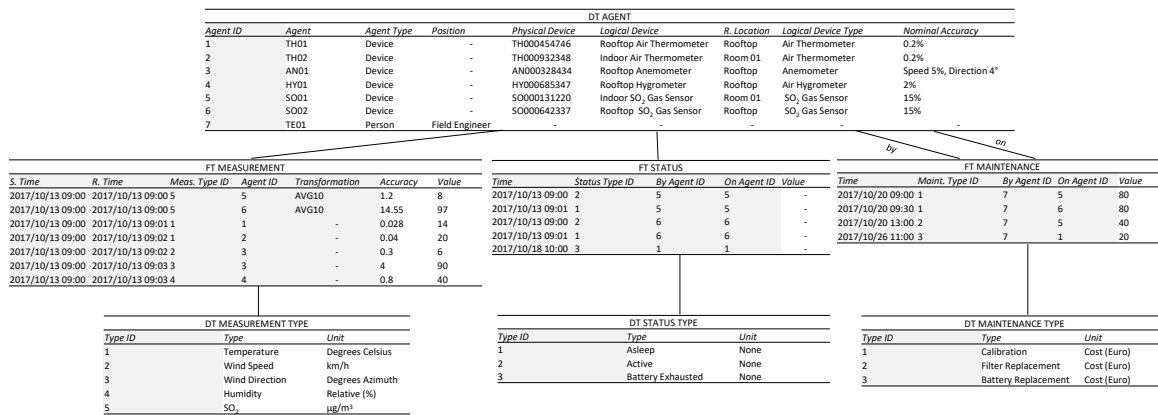


Figure 5.8: Star schema and sample data for the air quality scenario

5.2.4.3 Sensing Air Quality

From the above scientific and business case we can synthesise a set of sensors that we need to deploy in order to address a set of scenarios:

- operational sensing, to ensure that the plant is behaving correctly;
- acute event sensing, to detect unexpected emissions as quickly as possible; and
- contextual sensing, to contextualise the other datasets.


With this in mind, we might decide to install a network of sensors aimed at detecting the possible pollutant chemicals. Such sensors typically require chemical reactions in their operation, meaning that their reagents need to be replaced on a regular basis to avoid decalibration through changing chemistry. This addresses the operation and acute scenarios.

To provide context we might also decide to record environmental (or “met”) data such as air temperature and pressure, local wind direction, local humidity, and precipitation. These time series are not needed operationally, but might be crucial in *post facto* analysis after an incident, where (for example) the chemistry at play might be changed by an excess of water. They also suggest that we should position sensors differently than we might otherwise, both close to and further from the emission sites (to measure dispersion), and in areas of particular stakeholder interest (wetlands, residential areas, other industrial facilities) to detect potentially dangerous interactions.

5.2.4.4 Warehousing Air Quality Data

As noted in Table 5.2, the data needed to satisfy the requirements for this scenario can be drawn from the data lake (for real-time tasks) and from the DW, specifically from cubes

Weekly SO ₂ Levels				
Week	Location	SO ₂	Wind (km/h)	Temperature
2017/10/02 – 2017/10/08	Rooftop	98	7	13
	Room 01	11	-	20
2017/10/09 – 2017/10/15	Rooftop	105	9	15
	Room 01	13	-	20
2017/10/16 – 2017/10/22	Rooftop	144	4	15
	Room 01	16	-	20

 *drill-down*

Daily SO ₂ Levels for Week 2017/10/16 and Rooftop SO ₂ Gas Sensor		
Date	SO ₂	Days Since Last Calibration
2017/10/16	135	15
2017/10/17	170	16
2017/10/18	192	17
2017/10/19	205	18
2017/10/20	106	0
2017/10/21	102	1
2017/10/22	98	2

Figure 5.9: A simple analysis performed on air quality data

Measurement, Status Check, and Maintenance Operation. We focus on the DW and show a practical example of how the conceptual models of Figure 5.7 can be implemented. Figure 5.8 shows a basic star schema implementation of the multidimensional schemata proposed in Section 5.2.3 accompanied by a sample of (fictitious) data. Fact tables and dimension tables are denoted by prefixes FT and DT, respectively.

We start by commenting the implementation of the **Agent** dimension, which is shared by all cubes. Each level (except those not used in this scenario) becomes a column in table DT AGENT. The recursive arc on **Logical Device** has been ignored here as the topology of the sensor network is quite simple and comprises only one layer of sensors (in case of more complicated topologies, a parent-child table would be required [138]). Level R. **Location** is used to identify a generic location but, in more complex networks, it could also be used to refer to specific geographical coordinates. Levels **Process Type** and **Specifications** have been omitted as not necessary here. The sample data for DT AGENT represent how both devices and persons can be stored together; of course, for each type of agent, only a subset of levels are relevant and thus filled with meaningful values. The remaining dimensions are quite straightforward, especially the time-related ones and **Transformation**, which do not need a separate table and can thus be directly included in the fact tables.

For each multidimensional schema we have a fact table, composed by one column for each dimension and one column for each measure. As noted in Table 5.2 and shown in the sample data in Figure 5.8, Requirements (1) and (2) are both supported by the **Measurement** schema. For instance, the first two events in table FT MEASUREMENT represent pollution-related events (**Meas. Type ID** = 5) computed as the mean of 10 measurements (**Transformation** = AVG10) made by (logical) devices **Indoor SO₂ Gas Sensor** and **Rooftop SO₂ Gas Sensor** (**Agent ID** = {5, 6}). The other events are all contextual measurements, such as temperature, wind speed, etc. The remaining fact tables are used to store events respectively related to the status of the network and to the maintenance operations.

To close this section, in Figure 5.9 we show a simple example that combines data from

Table 5.3: Requirements for the landslides scenario

	Name	Description	Stakeholders	Main Data Sources
(1)	<i>Ground Movements</i>	Monitor the behavior of the landslide	Geologists	DW(Measurement)*
(2)	<i>Environmental Conditions</i>	Measure natural phenomena and properties such as wind, temperature, and humidity	Geologists	DW(Measurement)*
(3)	<i>Network Synchronization</i>	Gather statistics on the synchronization among the nodes of the network	Engineers	DW(Status Check)
(4)	<i>Communications</i>	Monitor communications between the nodes of the network	Engineers	DW(Message)
(5)	<i>Maintenance Operations</i>	Keep track of the maintenance operations executed on the network	Engineers, Geologists	DW(Maintenance Op.)

different cubes to perform an analysis on pollution levels. The pivot table on top shows the weekly average levels of pollution alongside some weather measurements (average wind speed and temperature). Since during the week from 2017/10/16 to 2017/10/22 the sensors outside the facility have measured particularly high levels of pollution, the user might decide to *drill-down* (i.e., zoom in) on this particular week and on a particular sensor placed outside. The bottom pivot table shows the result of the drill down that disaggregates the measurements of the chosen sensor on a daily basis; furthermore, to check for the possibility of calibration problems, the user also visualizes the number of days since the last calibration has been made.

5.2.4.5 Landslides Risk Management

Another interesting scenario to show the application of the proposed framework is the one presented by Giorgetti et al. [145], which describes a wireless sensor network for the monitoring of landslides. The network, deployed on a rockslide in central Italy, gathered both operational data (e.g., communication statistics) and measurements of natural phenomena (e.g., ground movement) in the time period from February to October 2013.

5.2.4.6 Requirements for Landslides Risk Management

Table 5.3 describes the requirements for this case study. The involved stakeholders are geologists and network engineers. The geologists are mainly interested in the collected measurements and, to better assess their validity, to the history of maintenance operations. The engineers are mainly interested in the operational data instead, to check if the network is working as intended.

The main goal of this sensor network is to collect data to observe and analyze landslides; to this end, both ground movement data (Requirement (1)) and other environmental measurements (Requirement (2)) must be taken into account. These data can be used to identify relevant factors causing landslides, thus aiding in preventing and reducing their

negative impact.

Similarly to the air quality scenario presented in Section 5.2.4.1, during the monitoring campaign the network itself was observed to collect operational data. Specifically, the network periodically collected and sent data regarding its synchronization mechanism (Requirement (3)) and the messages exchanged between the nodes (Requirement (4)). These data are crucial to determine if the network is behaving as intended; indeed, the network employs adapting routing strategies designed to cope with harsh environmental conditions (e.g., foliage, debris, ground movements, etc.). Finally, maintenance operations (Requirement (5)) must also be recorded to enable efficient management of the network and reduce costs.

5.2.4.7 Sensing Landslide Risk Data

The deployed sensor network for the scenario described above is composed by 15 wireless nodes. The nodes are equipped with 3 clinometers, 4 wire extensometers, 2 bar extensometers, and 4 soil hygrometers. One of these nodes acts as a coordinator that sends data to a remote unit for further elaboration. The coordinator node is also equipped with a weather station with several sensors such as an air thermometer, air hygrometer, etc. Sensor nodes are equipped with lead batteries and a solar cell.

The logical topology of the network is a tree where the root is the coordinator node. Each node can send data only to its father, thus the sensed data of a node can reach the coordinator only through child-parent communications. Notice that the network is self-organizing and has a dynamically defined topology to ensure fault tolerance and adaptability with minimal manual intervention.

To communicate, the network adopts a synchronization mechanism where a given node can be in one of four different phases, namely: association phase, receive phase, transmit phase, and sleep phase. A node goes in association phase when it requests to become part of the network. The receive and transmit phases are respectively related to the reception and transmission of data. Finally, to save energy, a node might begin a sleep phase where neither reception nor transmission of data are possible.

5.2.4.8 Warehousing Landslides Risk Data

As for the air quality case, Figure 5.10 shows a possible relational implementation with some representative data. We omitted the implementations of **Agent** and **Measurement** since they do not significantly differ from the ones presented in the previous scenario. In this case we focus on facts **Message** and **Status Check**, which are used to support Requirement (3) and (4).

The synchronization mechanism described in the previous section is at the core of Require-

FT STATUS				
Time	Status Type	By Agent ID	On Agent ID	Value
2013/04/08 07:00	Sleep	2	2	-
2013/04/08 08:58	Association	2	2	-
2013/04/08 09:00	Receive	2	2	-
2013/04/08 09:01	Transmit	2	2	-
2013/04/08 09:02	Sleep	2	2	-

FT MESSAGE					
S. Time	R. Time	Communication	Message Type	Received	From P. Device ID To P. Device ID
2013/04/08 09:00	2013/04/08 09:00	Unicast	Tilt Meas.	Yes	4 2
2013/04/08 09:01	2013/04/08 09:01	Unicast	Tilt Meas.	Yes	2 1
2013/04/16 16:00	2013/04/16 16:00	Unicast	Ext. Meas.	Yes	5 3
2013/04/16 16:01	-	Unicast	Ext. Meas.	No	3 1
2013/04/16 16:15	2013/04/16 16:15	Unicast	Ext. Meas.	Yes	3 1

Figure 5.10: Star schema and sample data for the landslide risk management scenario

ment (3). As shown in Figure 5.10, table FT STATUS, each phase change is recorded as a status check event performed by an agent on itself. The type of phase (sleep, association, receive, and transmit) is represented through the Status Check Type level, while the Time level marks the beginning of the current phase and the end of the previous one. Optionally, the Value measure could be used to explicitly denote the duration of each phase.

As for Requirement (4), each packet exchanged between the nodes of the network can be represented by a message event as shown in Figure 5.10, table FT MESSAGE. Specifically, the analyses performed in the study presented by Giorgetti et al. [145] involved the computation of two scores, namely the *fraction of packets sent* (FPS) and the *packet retransmission rate* (PRR). The former is computed as the number of packets sent from node T to node R , divided by the total number of packets sent by R ; FPS is thus used to identify which network paths are most (or least) used. The latter score quantifies the quality of a given link between two nodes and is computed as the percentage of retransmitted packets. Both scores can be easily computed based on the data contained in FT MESSAGE shown in Figure 5.10. Indeed, each event represents either an attempt of transmitting a packet (i.e., Received = 'No') or a successful transmission (i.e., Received = 'Yes'). To compute FPS is sufficient to count the number of successful transmission between all pairs of nodes, while for PRR the relevant events that have to be counted are the failed transmissions.

5.2.5 Wrapping up Sensor Data Modeling

This work aims to cross fertilize the Sensor Network field with contributions coming from the Data Analysis and Big Data ones. Although the three fields are perceived as quite closed, the requests from both academics and practitioners for a well-defined, robust, and comprehensive architecture and model is apparent and frequent. Our experiences in IoT projects show that the adoption of naive or off-the-shelf solutions reduce the analysis capabilities and, once adopted, they can hardly be recovered. Technology is evolving fast in the Data Analysis and Big Data fields, nonetheless the functional architecture proposed as well as the usage of multidimensional cubes are well-established in the field. For this reason we believe that our proposal can be a solid starting point for a data analysis solution.

Chapter 6

Conclusions

In this thesis we described our novel contributions that aim at enabling ubiquitous OLAP analyses. To achieve this goal, we touched several important topics that range over data extraction from service-oriented sources, representation of multidimensional data and, finally, multidimensional modeling. Each of these topics is crucial to enabling ubiquitous OLAP analyses. Greater extraction capabilities means being able to perform analyses on richer data. Compact multidimensional representations enable the fruition of OLAP on devices that usually do not support it. Finally, clever multidimensional modeling approaches make it easier for the user to obtain multidimensional structures, which are essential to carry out OLAP analyses. In the following, we briefly wrap up the different contributions presented in the previous chapters, while trying to lay down the direction of future work.

The first topic that we treated is that of data extraction from service-oriented sources, which are nowadays employed in many scenarios related to the fruition of both analytic functions and simple access of raw dataset. In this context, we presented two different frameworks, namely QETL and Tiresias. The former represents an approach to incremental on-demand ETL based on the query-extract-transform-load paradigm suited for those situations in which traditional batch ETL is either unfeasible or inconvenient. In QETL, a data cube is operated as a cache to enable data reuse and thus reduce as much as possible costs related to querying data sources, while still managing as efficiently as possible a limited storage space. The key elements of this framework are: the dice construct, which allows to compactly abstract multidimensional data; the dice-difference operation, which enables an easy computation of the facts required to answer a query but still missing from the cube; a novel optimization technique to minimize the extraction costs of the missing facts. The latter framework, Tiresias, focuses instead on building a cost model that accurately predicts query costs in web services, and can thus be used to obtain useful statistics for interacting with them. In Tiresias, a cost model is implemented as a regression tree that can be initialized and actively adapted with minimal user intervention. The key for automating and optimizing the training of the cost model is a novel active

learning technique, which is composed by two different cycles: a passive one, to initialize the model; and an active one, which is used when the prediction accuracy goes below a given threshold (e.g., due to a function drift). To close this brief recap of the contributions related to service-oriented data sources, we outline some possible directions for future work. Overall, both frameworks presented promising results, however there is still room for improvement. The most straightforward developments are related to performances, for instance, the support of extractions at different levels of aggregation in the case of QETL. Other less straightforward, but more interesting, advances regard the extension of the applicability of the frameworks. Indeed, both QETL and Tiresias have some important limitations that either reduce their effectiveness, or even prevent them to be used in certain situations. For QETL, the greatest limitation is perhaps the fact that it is strictly tied to the OLAP paradigm, and can sometimes be impractical with modern tools that mesh this type of analyses with more advanced ones (e.g., Tableau). While this issue goes beyond the topic of this thesis, it is nonetheless an interesting starting point for future research. As for Tiresias, its major limitation is the fact that each service interface is modeled as a separate regression tree. In practice, different services might share many features and behaviors, thus it would make sense to reuse the work done for a service to obtain a cost model for another one (e.g., through *transfer learning* [146]).

The second topic tackled during this thesis is that of compact visualization of multidimensional data. The novel work that we presented is tied to the shrink operator, which has been previously introduced by Golfarelli et al. [5]. The first main contribution is an integer linear programming model for solving the shrink problem as efficiently as possible, i.e., both in terms of performances and accuracy. In this contribution, shrink is modeled as a partitioning problem with side constraints. To improve on the original greedy implementation of the operator, we have proposed a dual ascent which embeds a Lagrangian heuristic, and an exact method to be used when an optimal solution is needed. As the results have shown, the new heuristic implementation is able to generate a near optimal dual solutions in a short computing time, while the exact procedure outperforms general purpose solvers like CPLEX, which in some cases failed to produce a solution. The second contribution related to visualization that we have presented is a multidimensional generalization of shrink. This new generalization enables the reduction of a pivot table along multiple dimensions, which improves the quality of the final approximation and simplifies the usage of the operator from the user's viewpoint. The experiments shown that multidimensional shrink outperforms the original operator in terms of accuracy, while still keeping short computational times. Both contributions presented quite solid results and, even if they still have room for improvement, we believe that future developments should be focused on topics beyond the shrink operator. Indeed, while pivot tables are quite powerful tools, there are many other ways to effectively visualize data. In this regard, a relevant issue still to be addressed in a satisfying fashion is that of actively guiding the user in the choice of how to best visualize data during an analysis session to better identify interesting patterns. This kind of feature coupled with personalization approaches such as

shrink could greatly improve the results achieved through explorative analyses.

The last research theme we addressed is multidimensional modeling. The novel contributions related to this topic are Starry Vault, an approach to automatically derive multidimensional schemata from data vaults, and a set of multidimensional blueprints tailored for sensor data. The Starry Vault approach allows to detect a multidimensional schema out of a source data vault by exploiting both schema-based and data-based functional dependencies. In particular, this approach uses extensional techniques for discovering hidden dependencies, with some tolerance to errors in data and taking into account the temporal aspects related to historicization. Our second contribution follows a quite different direction; indeed, focusing on a specific domain (i.e., sensor networks) allowed us to manually tailor multidimensional blueprints that, with little editing, are able to cover a broad variety of requirements. To better contextualize these blueprints, we have also presented two scenarios to show how they can be used in practice. Regarding interesting future work related to multidimensional modeling, we remark that automatic derivation of schemata is a widely explored topic in the DW literature. However, in the era of big data and data science the need for on-the-fly analyses creates a strong requirement for a smarter design process. An example of such requirements is supporting the data scientist in discovering a multidimensional structure in situations in which the source data are poorly-structured or schemaless. Finally, regarding BI in the area of sensor networks, while there are research issues that are still challenging (e.g., dealing with probabilistic data), one of the biggest problems is the gap between the sensor network community and the BI one. Indeed, with our multidimensional blueprints, the goal is precisely that of bridging this gap, which sometimes prevents the application of even well tested and consolidated techniques.

Bibliography

- [1] M. Golfarelli and S. Rizzi. *Data Warehouse design: Modern principles and methodologies*. McGraw-Hill, 2009.
- [2] Matteo Francia, Enrico Gallinucci, Matteo Golfarelli, and Stefano Rizzi. Social business intelligence in action. In *International Conference on Advanced Information Systems Engineering*, pages 33–48. Springer, 2016.
- [3] Lorenzo Baldacci, Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. GOLAM: A framework for analyzing genomic data. In *Proc. DOLAP*, pages 3–12, Shanghai, China, 2014.
- [4] Raúl Fidalgo-Merino and Marcos Nunez. Self-adaptive induction of regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1659–1672, 2011.
- [5] Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. Shrink: An olap operation for balancing precision and size of pivot tables. *Data & Knowledge Engineering*, 93: 19–41, 2014.
- [6] Dan Linstedt. DV modeling specification v1.09. danlinstedt.com, 2013.
- [7] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.
- [8] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data integration flows for business intelligence. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 1–11. Acm, 2009.
- [9] Minnesota Population Center. Integrated public use microdata series. <http://www.ipums.org>, 2008.
- [10] Alberto Abelló, Jérôme Darmont, Lorena Etcheverry, Matteo Golfarelli, Jose-Norberto Mazón, Felix Naumann, Torben Bach Pedersen, Stefano Rizzi, Juan Trujillo, Panos Vassiliadis, and Gottfried Vossen. Fusion cubes: Towards self-service business intelligence. *IJDWM*, 9(2):66–88, 2013.

- [11] Lorenzo Baldacci, Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. Analyzing genomic mappings with the GOLAM framework. In *Proc. SEBD*, pages 80–87, Gaeta, Italy, 2015.
- [12] Brian Raney, Melissa Cline, Kate Rosenbloom, Timothy Dreszer, Katrina Learned, Galt Barber, Laurence Meyer, Cricket Sloan, Venkat Malladi, Krishna Roskin, Bernard Suh, Angie Hinrichs, Hiram Clawson, Ann Zweig, Vanessa Kirkup, Pauline Fujita, Brooke Rhead, Kayla Smith, Andy Pohl, Robert Kuhn, Donna Karolchik, David Haussler, and James Kent. ENCODE whole-genome data in the UCSC genome browser. *Nucleic Acids Res.*, (39):D871–D875, 2011.
- [13] Stefano Rizzi and Enrico Gallinucci. CubeLoad: A parametric generator of realistic OLAP workloads. In *Proc. CAiSE*, pages 610–624, Thessaloniki, Greece, 2014.
- [14] Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. Here are my data files. here are my queries. where are my results? In *Proc. Conf. on Innovative Data Systems Research*, pages 57–68, 2011.
- [15] Kristi Morton, Magdalena Balazinska, Dan Grossman, and Jock Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *Proc. VLDB Endowment*, 7(6), 2014.
- [16] Benedikt Kämpgen and Andreas Harth. Transforming statistical linked data for use in OLAP systems. In *Proc. Int. Conf. on Semantic systems*, pages 33–40, 2011.
- [17] Victoria Nebot, Rafael Berlanga, Juan Manuel Pérez, María José Aramburu, and Torben Bach Pedersen. Multidimensional integrated ontologies: a framework for designing semantic data warehouses. *Journal on Data Semantics*, XIII:1–36, 2009.
- [18] Svetlana Mansmann, Nafees Ur Rehman, Andreas Weiler, and Marc H Scholl. Discovering OLAP dimensions in semi-structured data. *Information Systems*, 44: 120–133, 2014.
- [19] Oscar Romero, Alkis Simitsis, and Alberto Abelló. GEM: Requirement-driven generation of ETL and multidimensional conceptual designs. In *Proc. DaWaK*, pages 80–95, Toulouse, France, 2011.
- [20] Florian Waas, Robert Wrembel, Tobias Freudenreich, Maik Thiele, Christian Koncilia, and Pedro Furtado. On-demand ELT architecture for right-time BI: Extending the vision. *IJDWM*, 9(2):21–38, 2013.
- [21] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. State-space optimization of ETL workflows. *TKDE*, 17(10):1404–1419, 2005.
- [22] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. Optimizing ETL processes in data warehouses. In *Proc. ICDE*, pages 564–575, 2005.

- [23] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and Nils Frantzell. Meshing streaming updates with persistent data in an active data warehouse. *TKDE*, 20(7):976–991, 2008.
- [24] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View maintenance in a warehousing environment. *ACM SIGMOD Record*, 24(2):316–327, 1995.
- [25] Dallan Quass and Jennifer Widom. On-line warehouse view maintenance. *ACM SIGMOD Record*, 26(2):393–404, 1997.
- [26] Inderpal Singh Mumick, Dallan Quass, and Barinderpal Singh Mumick. Maintenance of data cubes and summary tables in a warehouse. *ACM Sigmod Record*, 26(2):100–111, 1997.
- [27] Yagiz Kargin, Holger Pirk, Milena Ivanova, Stefan Manegold, and Martin L. Kersten. Instant-on scientific data warehouses: Lazy ETL for data-intensive research. In *Proc. BIRTE*, pages 60–75, Istanbul, Turkey, 2012.
- [28] Ying Yang, Niccolo Meneghetti, Ronny Fehling, Zhen Hua Liu, and Oliver Kennedy. Lenses: An on-demand approach to etl. *Proc. VLDB Endowment*, 8(12):1578–1589, 2015.
- [29] Arindam Nandi, Ying Yang, Oliver Kennedy, Boris Glavic, Ronny Fehling, Zhen Hua Liu, and Dieter Gawlick. Mimir: Bringing CTables into practice. *CoRR*, abs/1601.00073, 2016.
- [30] Peter Scheuermann, Junho Shim, and Radek Vingralek. WATCHMAN : A data warehouse intelligent cache manager. In *Proc. VLDB*, pages 51–62, Mumbai, India, 1996.
- [31] Qun Ren, Margaret Dunham, and Vijay Kumar. Semantic caching and query processing. *TKDE*, 15(1):192–210, 2003.
- [32] Boris Chidlovskii and Uwe Borghoff. Semantic caching of web queries. *VLDB Journal*, 9(1):2–17, 2000.
- [33] Dongwon Lee and Wesley Chu. Towards intelligent semantic caching for web sources. *Journal of Intelligent Information Systems*, 17(1):23–45, 2001.
- [34] Xiufeng Liu. Optimizing ETL dataflow using shared caching and parallelization methods. *CoRR*, abs/1409.1639, 2014.
- [35] Prasad M. Deshpande, Karthikeyan Ramasamy, Amit Shukla, and Jeffrey F. Naughton. Caching multidimensional queries using chunks. *SIGMOD Rec.*, 27(2):259–270, 1998.

- [36] E. J. Otoo, Doron Rotem, and Sridhar Seshadri. Optimal chunking of large multi-dimensional arrays for data warehousing. In *Proc. DOLAP*, pages 25–32, Lisbon, Portugal, 2007.
- [37] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. Wiley, 2009.
- [38] Matteo Golfarelli and Elisa Turricchia. A characterization of hierarchical computable distance functions for data warehouse systems. *Decision Support Systems*, 62:144–157, 2014.
- [39] Stefan Manegold, Peter Boncz, and Martin L Kersten. Generic database cost models for hierarchical memory systems. In *Proc. VLDB*, pages 191–202, 2002.
- [40] Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. Cost-based optimization of service compositions. *IEEE Transactions on Services Computing*, 6(2): 239–251, 2013.
- [41] Yilei Zhang, Zibin Zheng, and Michael R Lyu. WSPred: A time-aware personalized QoS prediction framework for web services. In *Proc. ISSRE*, pages 210–219, 2011.
- [42] Waseem Ahmed, Yongwei Wu, and Weimin Zheng. Response time based optimal web service selection. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):551–561, 2015.
- [43] Li Kuang, Zhifang Liao, Wentao Feng, Haoneng He, and Bei Zhang. Multimedia services quality prediction based on the association mining between context and QoS properties. *Signal Processing*, 120:767–776, 2016.
- [44] Qiang Zhu and Per-Åke Larson. Building regression cost models for multidatabase systems. In *Proc. PDIS*, pages 220–231, 1996.
- [45] Chetan Gupta, Abhay Mehta, and Umeshwar Dayal. PQR: Predicting query execution times for autonomous workload management. In *Proc. ICAC*, pages 13–22, 2008.
- [46] Mert Akdere, Ugur Cetintemel, Matteo Riondato, Eli Upfal, and Stanley B Zdonik. Learning-based query performance modeling and prediction. In *Proc. ICDE*, pages 390–401, 2012.
- [47] Adrian Daniel Popescu, Vuk Ercegovic, Andrey Balmin, Miguel Branco, and Anastasia Ailamaki. Same queries, different data: Can we predict runtime performance? In *Proc. ICDEW*, pages 275–280, 2012.
- [48] Ning Zhang, Peter J Haas, Vanja Josifovski, Guy M Lohman, and Chun Zhang. Statistical learning techniques for costing XML queries. In *Proc. VLDB*, pages 289–300, 2005.

- [49] Zhen He, Byung Suk Lee, and Robert Snapp. Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *ACM TODS*, 30(3):812–853, 2005.
- [50] Viswanath Poosala and Yannis E Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. VLDB*, volume 97, pages 486–495, 1997.
- [51] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. STHoles: a multidimensional workload-aware histogram. *ACM SIGMOD Record*, 30(2):211–222, 2001.
- [52] Utkarsh Srivastava, Peter J Haas, Volker Markl, and TM Tran. ISOMER: Consistent histogram construction using query feedback. In *Proc. ICDE*, pages 39–39, 2006.
- [53] Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. Optimal algorithms for crawling a hidden database in the web. *Proc. VLDB*, 5(11):1112–1123, 2012.
- [54] Arjun Dasgupta, Gautam Das, and Heikki Mannila. A random walk approach to sampling hidden databases. In *Proc. SIGMOD*, pages 629–640, 2007.
- [55] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.
- [56] Indre Zliobaite, Albert Bifet, Bernhard Pfahringer, and Graham Holmes. Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):27–39, 2014.
- [57] Jasmina Smailović, Miha Grčar, Nada Lavrač, and Martin Žnidaršič. Stream-based active learning for sentiment analysis in the financial domain. *Information Sciences*, 285:181–203, 2014.
- [58] Marin Silic, Goran Delac, Ivo Krka, and Sinisa Sribljic. Scalable and accurate prediction of availability of atomic web services. *IEEE Transactions on Services Computing*, 7(2):252–264, 2014.
- [59] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.
- [60] Probal Chaudhuri, Min-Ching Huang, Wei-Yin Loh, and Ruji Yao. Piecewise-polynomial regression trees. *Statistica Sinica*, pages 143–167, 1994.
- [61] Meikel Poess and Chris Floyd. New tpc benchmarks for decision support and web commerce. *ACM Sigmod Record*, 29(4):64–71, 2000.
- [62] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions of Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

- [63] Patrick Marcel, Rokia Missaoui, and Stefano Rizzi. Towards intensional answers to OLAP queries for analytical sessions. In *Proc. DOLAP*, pages 49–56, Maui, USA, 2012.
- [64] Matteo Golfarelli, Stefano Rizzi, and Paolo Biondi. myOLAP: An approach to express and evaluate OLAP preferences. *IEEE Trans. Knowl. Data Eng.*, 23(7): 1050–1064, 2011.
- [65] Jeffrey Scott Vitter and Min Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. SIGMOD*, pages 193–204, Philadelphia, USA, 1999.
- [66] Jiawei Han. Olap mining: An integration of olap with data mining. In *Proceedings of the 7th IFIP*, volume 2, pages 1–9, 1997.
- [67] Natalija Kozmina and Laila Niedrite. Research directions of olap personalization. In *Information Systems Development*, pages 345–356. Springer, 2011.
- [68] Svetlana Mansmann and Marc Scholl. Exploring OLAP aggregates with hierarchical visualization techniques. In *Proc. ACM SAC*, pages 1067–1073, 2007.
- [69] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for olap discovery driven analysis. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, pages 81–88. ACM, 2009.
- [70] Irene Garrigós, Jesús Pardillo, Jose-Norberto Mazón, and Juan Trujillo. A conceptual modeling approach for olap personalization. *ER*, 9:401–414, 2009.
- [71] Houssein Jerbi, Franck Ravat, Olivier Teste, and Gilles Zurfluh. A framework for OLAP content personalization. In *Proc. ADBIS*, pages 262–277, Novi Sad, Serbia, 2010.
- [72] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Mining preferences from OLAP query logs for proactive personalization. In *Proc. ADBIS 2011*, pages 84–97, Vienna, Austria, 2011.
- [73] Andreas Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, Yannis Vassiliou, George Mavrogonatos, and Ilias Michalarias. A presentation model & non-traditional visualization for OLAP. *IJDWM*, 1(1):1–36, 2005.
- [74] Alfredo Cuzzocrea, Domenico Sacca, and Paolo Serafino. Semantics-aware advanced OLAP visualization of multidimensional data cubes. *IJDWM*, 3(4):1–30, 2007.
- [75] Ladjel Bellatreche, Arnaud Giacometti, Patrick Marcel, Hassina Mouloudi, and Dominique Laurent. A personalization framework for OLAP queries. In *Proc. DOLAP*, pages 9–18, Bremen, Germany, 2005.

- [76] Fadila Bentayeb and Cécile Favre. RoK: Roll-up with the k-means clustering method for recommending OLAP queries. In *Proc. DEXA*, pages 501–515, Linz, Austria, 2009.
- [77] Amihai Motro. Intensional answers to database queries. *IEEE Trans. Knowl. Data Eng.*, 6(3):444–454, 1994.
- [78] Dimitrios Gkesoulis, Panos Vassiliadis, and Petros Manousis. Cinecubes: aiding data workers gain insights from olap queries. *Information Systems*, 53:60–86, 2015.
- [79] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *Proc. SIGMOD*, pages 487–498, Dallas, 2000.
- [80] Francesco Buccafurri, Filippo Furfaro, Giuseppe M. Mazzeo, and Domenico Saccà. A quad-tree based multiresolution approach for two-dimensional summary data. *Inf. Syst.*, 36(7):1082–1103, 2011.
- [81] Yu Feng and Shan Wang. Compressed data cube for approximate OLAP query processing. *Journal of Computer Science and Technology*, 17(5):625–635, 2002.
- [82] Michel de Rougemont and Phuong Thao Cao. Approximate answers to OLAP queries on streaming data warehouses. In *Proc. DOLAP*, pages 121–128, Maui, USA, 2012.
- [83] Juozas Gordevicius, Johann Gamper, and Michael H. Böhlen. Parsimonious temporal aggregation. *VLDB J.*, 21(3):309–332, 2012.
- [84] John F. Kros, Marvin L. Brown, and Mike Lin. Effects of the neural network s-Sigmoid function on KDD in the presence of imprecise data. *Computer & Operations Research*, 33(11):3136–3149, 2006.
- [85] Parag C. Pendharkar. A data mining constraint satisfaction optimization problem for cost effective classification. *Computer & Operations Research*, 33(11):3124–3135, 2006.
- [86] Sigurdur Ólafsson. Introduction to operations research and data mining. *Computer & Operations Research*, 33(11):3067–3069, 2006.
- [87] Sigurdur Ólafsson, Xiaonan Li, and Shuning Wu. Operations research and data mining. *European Journal of Operational Research*, 187(3):1429–1448, 2008.
- [88] P. S. Bradley, Usama M. Fayyad, and O. L. Mangasarian. Mathematical programming for data mining: Formulations and challenges. *INFORMS Journal on Computing*, 11(3):217–238, 1999.
- [89] Balaji Padmanabhan and Alexander Tuzhilin. On the use of optimization for data mining: Theoretical interactions and eCRM opportunities. *Management Science*, 49(10):1327–1343, 2003.

- [90] Imene Mami and Zohra Bellahsene. A survey of view selection methods. *SIGMOD Rec.*, 41(1):20–29, April 2012. ISSN 0163-5808.
- [91] Jeffrey Xu Yu, Xin Yao, Chi-Hon Choi, and Gang Gou. Materialized view selection as constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 33(4):458–467, 2003.
- [92] Marek Łatuszko and Radosław Pytlak. Methods for solving the mean query execution time minimization problem. *European Journal of Operational Research*, 246:582–596, 2015.
- [93] Xinjian Lu and Franklin Lowenthal. Arranging fact table records in a data warehouse to improve query performance. *Computer & Operations Research*, 31:2165–2182, 2004.
- [94] Roselyn Abbiw-Jackson, Bruce Golden, S. Raghavan, and Edward Wasil. A divide-and-conquer local search heuristic for data visualization. *Computers & Operations Research*, 33(11):3070–3087, 2006.
- [95] Chun-Che Huang, Tzu-Liang (Bill) Tseng, Ming-Zhong Li, and Roger R. Gung. Models of multi-dimensional analysis for qualitative data and its application. *European Journal of Operational Research*, 174(2):983–1008, 2006.
- [96] Davy Janssens, Tom Brijs, Koen Vanhoof, and Geert Wets. Evaluating the performance of cost-based discretization versus entropy- and error-based discretization. *Computers & Operations Research*, 33(11):3107 – 3123, 2006.
- [97] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981. ISSN 0025-5610.
- [98] A. Mingozzi, M.A. Boschetti, L. Bianco, and S. Ricciardelli. A set partitioning approach to the crew scheduling problem. *Operations Research*, 47:883–888, 1999.
- [99] M.A. Boschetti, A. Mingozzi, and S. Ricciardelli. An exact algorithm for the simplified multiple depot crew scheduling problem. *Annals of Operations Research*, 127:177–201, 2004.
- [100] M.A. Boschetti, A. Mingozzi, and S. Ricciardelli. A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5(4):735–747, 2008. ISSN 1572-5286.
- [101] M.A. Boschetti and V. Maniezzo. A set covering based matheuristic for a real world city logistics problem. *International Transactions in Operational Research*, 22(1): 169–195, 2015.
- [102] E. Balas and M.C. Carrera. A dynamic subgradient-based branch-and-bound procedure for the set covering. *Operations Research*, 44:875–890, 1996.

- [103] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000.
- [104] M.A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufe Rohler. Matheuristics: Optimization, simulation and control. In M.J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Samples, and Schaerf A., editors, *Hybrid Metaheuristics*, volume 5818 of *LNCS*, pages 171–177. Springer, 2009.
- [105] Marco Boschetti and Vittorio Maniezzo. Benders decomposition, Lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15:283–312, 2009. ISSN 1381-1231.
- [106] Pentaho Corporation. Pentaho Web Site. www.pentaho.com, 2016.
- [107] Ian Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Proc. PKDD*, pages 59–70, Porto, Portugal, 2005.
- [108] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson International, 2006.
- [109] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proc. ICDT*, pages 217–235, 1999.
- [110] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.
- [111] Robert Winter and Bernhard Strauch. A method for demand-driven information requirements analysis in data warehousing projects. In *Proc. HICSS*, page 231, Big Island, 2003.
- [112] Vladan Jovanovic and Ivan Bojicic. Conceptual data vault model. In *Proc. SAIS*, volume 23, pages 1–6, Atlanta, Georgia, 2012.
- [113] Dragoljub Krneta, Vladan Jovanovic, and Zoran Marjanovic. A direct approach to physical data vault design. *Comput. Sci. Inf. Syst.*, 11(2):569–599, 2014.
- [114] Ralph Hughes. *Agile Data Warehousing for the Enterprise*. Elsevier Science, 2015.
- [115] QOSQO. QUIPU 1.1 Whitepaper. www.datawarehousemanagement.org, 2016.
- [116] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. Conceptual design of data warehouses from E/R schemes. In *Proc. HICSS*, pages 334–343, Kohala Coast, HI, 1998.

- [117] Cassandra Phipps and Karen C. Davis. Automating data warehouse conceptual schema design and evaluation. In *Proc. DMDW*, pages 23–32, Toronto, Canada, 2002.
- [118] Mikael R. Jensen, Thomas Holmgren, and Torben Bach Pedersen. Discovering multidimensional structure in relational data. In *Proc. DaWaK*, pages 138–148, Zaragoza, Spain, 2004.
- [119] Jinho Kim et al. SAMSTARplus: An automatic tool for generating multi-dimensional schemas from an entity-relationship diagram. *Revista de Informática Teórica e Aplicada*, 16(2):79–82, 2009.
- [120] Matteo Golfarelli, Stefano Rizzi, and Boris Vrdoljak. Data warehouse design from XML sources. In *Proc. DOLAP*, pages 40–47, Atlanta, Georgia, 2001.
- [121] Stefan Anderlik, Bernd Neumayr, and Michael Schrefl. Using domain ontologies as semantic dimensions in data warehouses. In *Proc. ER*, pages 88–101, Florence, Italy, 2012.
- [122] Dario Colazzo, François Goasdoué, Ioana Manolescu, and Alexandra Roatis. RDF analytics: lenses over semantic graphs. In *Proc. WWW*, pages 467–478, Seoul, Republic of Korea, 2014.
- [123] Oscar Romero and Alberto Abelló. A framework for multidimensional design of data warehouses from ontologies. *Data Knowl. Eng.*, 69(11):1138–1157, 2010.
- [124] Alberto Abelló, Oscar Romero, Torben Bach Pedersen, Rafael Berlanga, Victoria Nebot, Maria Jose Aramburu, and Alkis Simitsis. Using semantic web technologies for exploratory olap: a survey. *IEEE transactions on knowledge and data engineering*, 27(2):571–588, 2015.
- [125] Andrea Carmè, Jose-Norberto Mazón, and Stefano Rizzi. A model-driven heuristic approach for detecting multidimensional facts in relational data sources. In *Proc. DAWAK*, pages 13–24, Bilbao, Spain, 2010.
- [126] Hans Hultgren. Data vault modeling guide. hanshultgren.files.wordpress.com, 2012.
- [127] Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng.*, 8(4): 563–582, 1996.
- [128] Carlo Combi, Paolo Parise, Pietro Sala, and Giuseppe Pozzi. Mining approximate temporal functional dependencies based on pure temporal grouping. In *Proc. ICDM Workshops*, pages 258–265, Dallas, USA, 2013.

- [129] Zhiqiang Huo, Mithun Mukherjee, Lei Shu, Yuanfang Chen, and Zhangbing Zhou. Cloud-based data-intensive framework towards fault diagnosis in large-scale petrochemical plants. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International*, pages 1080–1085. IEEE, 2016.
- [130] Michael Scriney, Martin F O’Connor, and Mark Roantree. Generating cubes from smart city web data. 2017.
- [131] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. Conceptual design of data warehouses from e/r schemes. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, volume 7, pages 334–343. IEEE, 1998.
- [132] Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. Starry vault: Automating multidimensional modeling from data vaults. In *East European Conference on Advances in Databases and Information Systems*, pages 137–151. Springer, 2016.
- [133] Open Geospatial Consortium et al. Ogc® sensorml: Model and xml encoding standard. *online article*], Retrieved from: <http://www.opengeospatial.org/standards/sensorml> (accessed on 16 Dec 2015), 2014.
- [134] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 949–955. IEEE, 2011.
- [135] Thanos G Stavropoulos, Dimitris Vrakas, Danai Vlachava, and Nick Bassiliades. Bonsai: a smart building ontology for ambient intelligence. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, page 30. ACM, 2012.
- [136] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srinu Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.
- [137] Franco Cicirelli, Giancarlo Fortino, Antonio Guerrieri, Giandomenico Spezzano, and Andrea Vinci. Metamodeling of smart environments: from design to implementation. *Advanced Engineering Informatics*, 2016.
- [138] Matteo Golfarelli and Stefano Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc., 2009.
- [139] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

- [140] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.
- [141] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 996–1005. IEEE, 2010.
- [142] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.
- [143] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.
- [144] Sonia Rivest, Yvan Bédard, Marie-Josée Proulx, Martin Nadeau, Frederic Hubert, and Julien Pastor. Solap technology: Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data. *ISPRS journal of photogrammetry and remote sensing*, 60(1):17–33, 2005.
- [145] Andrea Giorgetti, Matteo Lucchi, Emanuele Tavelli, Marco Barla, Giovanni Gigli, Nicola Casagli, Marco Chiani, and Davide Dardari. A robust wireless sensor network for landslide risk analysis: System design, deployment, and field testing. *IEEE Sensors Journal*, 16(16):6374–6386, 2016.
- [146] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.