Alma Mater Studiorum - University of Bologna

PhD in Electronics, Computer Science and Telecommunication
Cycle XXV

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

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

Pervasive Business Intelligence

Elisa Turricchia

Coordinator
Prof. Alessandro Vanelli Coralli

Advisor
Prof. Matteo Golfarelli

Final Exam 2013

Contents

1 Introduction **1**
- 1.1 Business Intelligence . . . 1
- 1.2 Motivations and Contributions . . . 4
  - 1.2.1 Distributed BI . . . 5
  - 1.2.2 OLAP Personalization and Similarity . . . 6
  - 1.2.3 Agile Data Warehouse Design . . . 7

2 Background **9**
- 2.1 Basic Concepts . . . 9
  - 2.1.1 Data Warehouse . . . 9
  - 2.1.2 Architectures . . . 11
  - 2.1.3 ETL . . . 14
  - 2.1.4 OLAP Analysis . . . 15
- 2.2 Life-cycle Design . . . 16
- 2.3 Multidimensional Model . . . 17

3 Distributed BI **21**
- 3.1 Introduction . . . 21
- 3.2 Related Works . . . 24
  - 3.2.1 Warehousing Approaches . . . 26
  - 3.2.2 Federative Approaches . . . 26
  - 3.2.3 Peer-to-Peer Approaches . . . 28
- 3.3 Formal Background . . . 29
- 3.4 Mapping Language . . . 31
  - 3.4.1 Mapping Predicates . . . 32
  - 3.4.2 Mapping Accuracy . . . 34
- 3.5 A Reformulation Framework . . . 36
  - 3.5.1 Translating schemata . . . 37
  - 3.5.2 Translating queries . . . 38
  - 3.5.3 Translating mappings . . . 40
    - 3.5.3.1 Exact Mappings . . . 41
    - 3.5.3.2 Loose/Approximate Mappings . . . 43
- 3.6 Query Reformulation in a BIN . . . 44
  - 3.6.1 The Inter-Peer Reformulation Algorithm . . . 45
    - 3.6.1.1 Step 1: Mapping Selection . . . 45
    - 3.6.1.2 Step 2: Query Reformulation . . . 47
    - 3.6.1.3 Step 3: Query Expansion . . . 51

3.6.1.4 Incorporating Selection Predicates 52
3.6.2 Properties of the Inter-Peer Reformulation Algorithm 52
3.7 Implementation 54
3.8 Conclusions 56

4 OLAP Personalization 59
4.1 Introduction 59
4.2 Related Works 60
4.3 Formal Background 62
4.4 The MYMDX Preference Language 63
4.5 A Personalization Framework 64
4.5.1 Log Mining 65
4.5.2 Rule Selection 66
4.5.3 Fragment Translation 69
4.6 Implementation 69
4.7 Validation 70
4.8 Conclusions 71

5 OLAP Similarity 73
5.1 Introduction 73
5.2 Formal Background 74
5.3 Requirements for OLAP sessions similarity 75
5.4 Related Works 77
5.4.1 Sessions 77
5.4.2 Queries 79
5.5 Query Similarity 82
5.6 Session Similarity 85
5.6.1 Edit-Based Session Similarity 85
5.6.2 Subsequence-Based Session Similarity 86
5.6.3 Log-Based Session Similarity 86
5.6.4 Alignment-Based Session Similarity 89
5.7 Validation 93
5.7.1 User Tests 93
5.7.2 Objective Tests 96
5.8 Conclusions 99

6 Agile Data Warehouse Design 101
6.1 Introduction 101
6.2 Related Works 102
6.3 The Motivation for 4WD 103
6.3.1 From Problems to Goals 103
6.3.2 From Goals to Principles 105
6.3.3 From Principles to 4WD 106
6.4 The 6 Features of 4WD 108
6.4.1 Incrementality and Risk-Based Iteration 108
6.4.2 Prototyping 109
6.4.3 User Involvement 110

6.4.4 Component Reuse . . . 111
6.4.5 Formal and Light Documentation . . . 112
6.4.6 Automated Schema Transformation . . . 113
6.5 Validation . . . 114
6.6 Conclusions . . . 115

7 Project Scheduling Optimization in Agile Data Warehouse Design 117
7.1 Introduction . . . 117
7.2 Related Works . . . 119
7.3 Multi-Sprint Planning Problem . . . 121
7.3.1 Baseline Planning Optimization Model . . . 124
7.3.2 Smooth Replanning Optimization Model . . . 127
7.3.3 Implementation . . . 129
7.3.4 Validation . . . 129
7.3.4.1 Effectiveness Tests for Baseline Planning . . . 129
7.3.4.2 Efficiency Tests for Baseline Planning . . . 134
7.3.4.3 Effectiveness Tests for Smooth Replanning . . . 135
7.3.4.4 Efficiency Tests for Smooth Replanning . . . 136
7.4 Efficient Algorithms for the Multi-Sprint Planning Problem . . . 137
7.4.1 Reductions . . . 137
7.4.1.1 Modifying the Sprint Capacities . . . 137
7.4.1.2 Modifying the Weights of Stories . . . 137
7.4.2 Cover Inequalities . . . 138
7.4.3 Dominance Inequalities . . . 138
7.4.3.1 Dominance of Type 1 . . . 138
7.4.3.2 Dominance of Type 2 . . . 138
7.4.3.3 Dominance of Type 3 . . . 139
7.4.4 Greedy and Exchange Heuristics . . . 139
7.4.5 A Lagrangian Heuristic . . . 144
7.4.6 Validation . . . 146
7.5 Conclusions . . . 152

8 Conclusions and Future Works 155

A Theorem Proofs 157

Bibliography 159

Keywords

- *Business Intelligence 2.0*
- *Distributed Data Warehouses*
- *Query Reformulation*
- *Lean Development*
- *OLAP Query Personalization and Recommendation*
- *Agile Release Scheduling*
- *Similarity Measures in Multidimensional Contexts*

To my family

Acknowledgements

I would like to thank my tutor Prof. Dario Maio for the supervising activity during my PhD. I would like to express my sincere gratitude to my advisors Prof. Matteo Golfarelli and Prof. Stefano Rizzi for supporting and motivating me during these three research years. They gave me the opportunity to take part in several activities that have contributed to my professional and personal growth.

Moreover, I wish acknowledge Prof. Patrick Marcel and Julien Aligon for their collaboration and support during my research period in France at Université François-Rabelais Tours.

I would like to thank Prof. Marie-Aude Aufaure and Prof. Patrick Marcel for reviewing this thesis.

Finally, I wish to thank my family and my friends for believing in me during this journey, and in particular my friend Silvia who has encouraged me during this research activity with her motivation and enthusiasm.

Chapter 1

Introduction

1.1 Business Intelligence

Traditionally, we refer to *Business intelligence* (BI) as the process of transforming raw data into useful information to support effective and aware business strategies; capturing the business data and getting the right information to the right people, at the right time, through the right channel, is a crucial aspect of BI often referred to as *pervasiveness*.

In the last few years, a new generation of BI tools called *BI 2.0* has emerged to meet the new and ambitious requirements of business users [Nelson, 2010]. BI 2.0 not only introduces brand new topics, but in some cases it re-examines past challenges according to new perspectives depending on the market changes and needs. In this context, the term *pervasive BI* has gained increasing interest as an innovative and forward-looking perspective. Different interpretations have been proposed of this concept, mainly focused on keywords such as *Personalization*, *Timeliness*, and *Integration*:

- **Personalization (BI to ANYONE)**. In this case the term pervasive is referred to the capacity of BI tools to customize the result according to the user who takes advantage of it, facilitating the fruition of BI information by different type of users (e.g., front-line employees, suppliers, customers, or business partners). In this regard, [Markarian et al., 2007] states that "the goal of pervasive BI applications is to take the data that produced the back office *Return On Investment* (ROI) of more than 400% and deliver it to front-line employees in a form appropriate to their job functions with similar results". This work points out two critical aspects of the decision process: identifying and also presenting the most relevant information of the company's trend, depending on the specific recipients. In this direction, BI information can be exploited by a wider range of people and different perspectives of analysis can enrich the business vision as well. Typically, at the core of BI architectures, a *Data Warehouse* (DW) stores information in multidimensional form to facilitate the extraction of relevant business data. However, DW analysis is still a complex activity due to the huge quantity of data to take into account. Moreover, users might not undertake the right direction of analysis. In this scenario, user-centric BI applications could have a strategic role in driving the business analysis. For instance, the result of a user query could be personalized according to the user preferences or depending on the user context. Besides, a BI system can suggest the next query to formulate to the user, exploiting past analysis of the same user or queries of groups of analysts with similar characteristics.

- **Timeliness (BI ANYTIME)**. Here the term pervasive is related to the timely provision of business information for decision-making. Different factors may affect this capacity. First of all, as shown in [The Data Warehousing Institute, 2008], one of the major shortcoming of DW solutions is the long and complex development process, that also discourages the adoption of BI tools. Typically, DW design implies heavy *Extraction, Transformation, and Loading* (ETL) activities that delay the fruition of useful business information. Besides, the design complexity (i.e., time and cost) increases for enterprise-wide BI solutions where heterogeneity problems are more serious. The risk is to yield inadequate results with respect to needs in continuous evolution. To overcome this issue, deeper investigations on methodological aspects to make the DW development process more flexible and faster represent a promising direction. On the other hand, after the implementation of the DW, a further issue is the need to maintain fresh data to support well-informed decisions. Moreover, the integration of the DW information with structured or semi-structured data coming from additional sources (e.g., external vendors, Internet) could complete the business view (in the so-called *situational BI*). Since this type of information is constantly changing, its integration on-the-fly represents an added value. In this direction, a new generation of BI systems providing information on demand with near-0 latency (the so-called *real-time BI*) may deal with the market unpredictability and dynamism [Teradata, 2008].

- **Integration (BI ANYWHERE)**. In this case the term pervasive refers to the ability of BI tools to allow users to access information anywhere it can be found, by using the device they prefer. We can distinguish two different interpretations. In the first one, pervasiveness is related to the ability of extracting relevant information from different BI systems (i.e., BI FROM-ANYWHERE) mainly dealing with heterogeneity and security issues. This feature is particular significant in collaborative contexts (*collaborative BI*) where enterprises collaborate and share information to create new business opportunities but preserving their autonomy and independence [Lachlan, 2012]. In the second interpretation, pervasiveness is justified by the fact that the information can be analyzed with different types of devices, depending on the user and on the context of analysis (i.e., BI TO-ANYWHERE). In this direction, the fruition of BI information from mobile devices represents one of the major trend today, making the data access easier and faster. As a consequence, new issues on data visualization and transmission arise.

Each of the aforementioned features impacts on the concept of **data trust (or data quality)** which in turn strongly affects the adoption of BI tools. If the user cannot fully understand data, she cannot perceive the utility of the information provided (BI to ANYONE). At the same time, if the user cannot access data in a timely manner, she cannot have a proactive role in the market (BI ANYTIME). Finally, if the user has not the right vision of the overall business environment, described by both internal and external factors, she can hardly define effective market strategies (BI ANYWHERE).

We close this vision of BI by describing the additional BI trends for 2012 as emerging from [Lachlan, 2012] and [Chaudhuri et al., 2011]:

- **Location Intelligence (LI)**: it starts from the assumption that more than 70% of the data collected by companies have a spatial topic [Yellowfin, 2010]. Enriching traditional business data with geographical information may lead to effective geo-targeted marketing, tactical business investments and strategic customer segmentation as well. It may represents the added value to capture significant patterns from the vast amount of data gathered by a company. In this perspective, LI may yield a competitive advantage in different contexts such as healthcare, governance, communications, and banking. For instance, the work in [Weber and Chapman, 2011] describes an innovative approach to location decision-making based on a geo-business classification of the London neighbourhood, aimed at attracting foreign investments to support economy.

- **Mobile BI**: it is strictly related to the concept of BI TO-ANYWHERE, referring to the fruition of business results on mobile devices. New metaphors of data visualization and real-time transmission techniques must be investigated to make the mobile experience effective. Managers can get business information wherever they are, by using a tool they are familiar with. Easier and self-service data source access is favoured. Besides, personalization techniques can reduce query results to the most significant information, decreasing the transmission time and facilitating an effective result fruition.

- **Cloud BI**: cloud BI is tightly related to the term BI ANYWHERE, in both perspectives. It refers to the ability of using BI platforms as a service (often called *BI as a service*) where users can access information from simple web interfaces (BI TO-ANYWHERE), while data are placed on multiple remote servers

(BI FROM-ANYWHERE). The accent is on the different types of architectures characterizing the cloud and multiple types of contract for service provision. As shown in [Baars and Kemper, 2010], different cloud configurations could be applied to the BI context. A simple vision implies the inclusion of additional functional blocks to the existing traditional BI system in a grid approach. A more sophisticated model includes a complete mashup of DW components distributed on the web, making the access to business information easier and faster.

- **Big Data**: the term refers to the ability of managing increasing amount of data. As a matter of fact, the number of sources (e.g., social networks, e-mails, geo-data) storing significant business information increases year by year as the technology evolves. Besides, the quantity of data logging grows as the business activity advances. As a consequence, performance and data storage issues arise to design scalable solutions. In this regard, the MapReduce paradigm is one of the most appreciated strategies [He et al., 2011], mainly based on efficient structures for data partitioning and compressing.
- **Social BI**: it uses *Data Mining* (DM) techniques to integrate business data with social information, exploiting unstructured data retrieved from e-mails, forums, and social networks, to support marketing activities such as brand reputation, topic discovery, and sentiment analysis. The company can exploit the customer opinions to have a complete vision of its own business, and also to better understand the market position of direct competitors.

1.2 Motivations and Contributions

According to the Gartner survey [Gartner, 2012] on the major priorities of 2,335 CIOs, analytics and BI are the top-ranked technologies for 2012. This vision includes the combination of BI tools with different technologies to create new capabilities: standalone applications give way to integrated solutions supporting the whole supply chain; moreover, geo-information and social factors can be exploited to study customer behaviour and market trends.

A deeper investigation on BI topics is justified by the increasing complexity of the decision-making process. The growth of the information affecting the business process is faster than the evolution of tools and techniques to manage and analyze it effectively. The unpredictability and dynamism of the market force companies to operate under constant-pressure conditions, where a proper analysis of both their own business activity and environmental features can lead to success strategies.

Companies need significant information about the outer world, for instance about trading partners and related business areas [T.A.D. Hoang, 2009]. According to BI

ANYWHERE, cooperation is seen as a key point to improve flexibility and competitiveness. For example, in the health-care area, combining data coming from different hospitals allows to monitor global phenomena, for instance to prevent epidemics. In banking and insurance contexts, acquisitions and fusions have become more frequent. In this case, an exhaustive knowledge of the several factors affecting the group activities can enforce the decision-making process.

Moreover, the effectiveness of decision-making depends on the quality of information [Chaudhuri et al., 2011]. The fresher the data, the more relevant the business strategies. To this end, BI ANYTIME focuses on both lean DW design and real-time DW applications.

Finally, the quality of business analysis is strictly related to the capacity of users to correctly understand the data. Providing information in the right format to the right user is the goal of BI to ANYONE.

In this regard, we focus on three different aspects of pervasive BI:

- *Distributed BI*: we consider the concept of pervasiveness in terms of location of data to answer an OLAP query (i.e., analysts can retrieve relevant information from multiple and heterogeneous BI systems);
- *OLAP Personalization and Similarity*: in this case the term pervasiveness refers to the utilization of BI tools by many users characterized by different profiles (i.e., the result of an OLAP query is personalized according to the characteristics of the user who has formulated the query, leading to a better and simplified result interpretation);
- *Agile Data Warehouse Design*: in this last case, pervasiveness is used in terms of distribution of BI tools in the market (i.e., the aim is to reduce cost and duration of DW projects to favour the penetration of BI solutions even in small and medium firms);

1.2.1 Distributed BI

We envision a *Business Intelligence Network* (BIN) of heterogeneous DW systems where users can share business information, preserving their autonomy and independence. The model supports the concept of BI FROM-ANYWHERE, in the sense that we combine information coming from multiple heterogeneous sources, located in different places, offering an innovative solution to company collaboration. The model includes a peer-to-peer network of BI systems where each node contains a specific DW with a particular multidimensional schema. The user can query her own system and receive results from every node storing relevant data for her request, by exploiting semantic mappings between the different schemata. The added value of this framework is the federated structure of the net. Typically, the implementations that integrate BI systems imply the creation of a global schema for query answering, but this solution is not feasible in particular contexts where companies want to maintain their autonomy and share just limited contents. In this context, we give the following contributions:

- We design a model for the distributed DW infrastructure.
- We define a language to link heterogeneous concepts in different multidimensional schemata.
- We propose a query reformulation algorithm to propagate the user query to the multiple nodes of the network and we provide its proof of correctness.
- We discuss the main implementation issues to develop a BIN.

These issues will be discussed in Chapter 3.

1.2.2 OLAP Personalization and Similarity

The goal of personalization is to deliver information that is relevant to an individual or a group of individuals in the most appropriate format and layout. In this sense, we can use personalization to support the concept of BI to ANYONE. In the *On-Line Analytical Processing* (OLAP) area, personalization may be pursued using different approaches:

- *Result ranking*: query results are organized in a total or partial order so that the user visualizes the most relevant data first [Golfarelli et al., 2011b].
- *Query contextualization*: the query is enhanced by adding preference predicates that depend on the query context [Jerbi et al., 2008].
- *Query recommendation*: based on the current query and on the past sessions, the system suggests further queries to help users navigating the cube [Giacometti et al., 2009].
- *Personalized visualization*: users specify a set of constraints that are used to determine a preferred visualization [Bellatreche et al., 2005].

We investigate the first three points of the list. As to *result ranking* and *query contextualization*, we propose a proactive approach that couples a *Multidimensional eXpression* MDX-based language ([Microsoft, 2009]) for expressing OLAP preferences to a mining technique for automatically deriving preferences for the current query. In this regard, our main contributions are:

- We design an algorithm to mine a set of association rules that relate sets of frequent query fragments, starting from the log of past MDX queries issued by a user.
- We define a procedure that selects a subset of pertinent and effective association rules for a particular query; after that, the selected rules are translated into a preference that is used to annotate the user query.
- We discuss a set of experimental results to prove both effectiveness and efficiency of our approach.

As to *query recommendation*, we design different measures to assess the similarity between the current OLAP analysis and the past ones, to derive significant hints for the next user query. To this end, we compare OLAP concepts from different perspectives: queries and sessions. In particular, we give the following contributions:

- We carry out a case study to identify the requirements for OLAP similarity.
- We define different measures of similarity for queries and sessions in the OLAP context.
- We test the effectiveness of the aforementioned measures.

For a detailed investigation on OLAP personalization see Chapter 4, while for OLAP similarity refer to Chapter 5.

1.2.3 Agile Data Warehouse Design

This term refers to methodological investigations to make DW design faster and nimbler, so as to support the concept of BI ANYTIME. First, we analyze the potential advantages arising from the application of modern software engineering methodologies (e.g., agile approaches) to a DW project; then we define an optimization model based on agile principles to support the analyst during the planning phase. The model is flexible enough to be applied in different contexts, even in data warehousing. In the following, the main contribution of this work:

- We identify the problems arising in DW development and we investigate how they can be solved by working on four qualities of the software development process (reliability, robustness, productivity, and timeliness); we also extract the main principles for an effective DW design methodology.
- Starting from the aforementioned principles, we propose an innovative methodology, called *Four-Wheel-Drive* (4WD), for DW development.

- We formalize an optimization model for the planning problem, based on the maximization of the project utility perceived by the user and complying with different development constraints; the model manages project uncertainty allowing a smooth replanning of new or disrupted software functionalites.
- We design efficient algorithms to solve the model for complex problems.
- We evaluate the effectiveness and efficiency of the approach.

4WD is described in Chapter 6, while the optimization model is presented in Chapter 7.

Chapter 2

Background

This chapter describes the basic concepts of BI. As mentioned in chapter 1, the DW is at the core of BI technologies. It stores data in a multidimensional structure to favour the extraction of relevant business information. We explain the basic DW features and the main activities for the ETL process. We also introduce the main OLAP operators to explore a DW and the typical phases of the DW life-cycle. Finally, a formalization of the multidimensional model is provided to be used as a reference in the following chapters.

2.1 Basic Concepts

Here we focus on DW characteristics and we informally illustrate the multidimensional model. The three typical DW architectures are described, as well as the main ETL procedures and OLAP operators.

2.1.1 Data Warehouse

In today's market, the DW is the main tool to support BI in both industrial and scientific contexts. Informally, a DW is an optimized repository that stores information for the decision-making process. As a matter of fact, the increasing number of information a company has to take into account to find relevant business strategies implies more sophisticated solutions than operational databases, that store accounting data deriving from daily management activities. A typical workload on operational data involves queries asking information on a particular customer, the items included in a specif order, or the total daily revenue. We typically refer to the process of managing operational data as *Online Transactional Processing* (OLTP). On the contrary, OLAP analyses are based on historical and analytical data. Typical OLAP queries are:

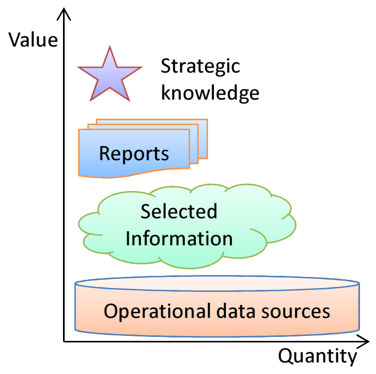


FIGURE 2.1: Information value as a function of quantity

- *Which products maximize the profit?*
- *What is the total revenue per product category and state?*
- *What is the relationship between profits gained by two different products?*
- *What is the revenue trend in the last three years?*

The previous requests can hardly be directly formulated on traditional information systems. The integration of data from different databases is needed, and historical data must be explored as well. Figure 2.1 shows how to achieve the real business knowledge through a progressive selection and aggregation process on the operational data.

To this end, [Golfarelli and Rizzi, 2009b] proposes a set of principles for the DW process, to turn operational data into information for decision-making:

- Accessibility to users not familiar with IT and data structures.
- Integration of data based on a standard enterprise model.
- Query flexibility to maximize the advantages obtained from the existing information.
- Information conciseness allowing for target-oriented and effective analyses.
- Multidimensional representation to give users an intuitive and manageable view of information.
- Correctness and completeness of integrated data.

A DW can be defined as a collection of data that supports decision-making processes. It provides the following features [Inmon, 1996]:

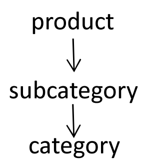


Figure 2.2: The product hierarchy

- It is subject-oriented.
- It is integrated and consistent.
- It shows its evolution over time and it is not volatile.

A DW is subject-oriented because it depends on enterprise-specific concepts, such as customers, products, sales, and orders. On the contrary, operational databases hinge on many different enterprise-specific applications. Since the DW takes advantage of multiple data sources, integration and consistency are significant properties. Moreover, it stores data covering multiple years to assess the company trend across several years and to compare data of different periods.

To facilitate OLAP analyses, the DW is typically broken up into different *data mart*, each representing a subset or an aggregation of the data stored in the primary DW. A data mart includes a set of information pieces relevant to a specific business area, corporate department, or category of users. The data mart is composed by different *facts* (e.g., orders and sales) that are the basic concepts of the multidimensional schema. Each fact is analyzed by different perspectives, called *dimensions* (e.g., products and stores). Each instance of a fact is called an *event* (e.g., a particular order or a specific sale) and it is described by the values of a set of relevant *measures* (e.g., the quantity sold) that provide a quantitative description of the event. Starting from these concepts, the multidimensional data can be represented by an n-dimensional *cube*, where n is the number of dimensions. For example, the sales in a store chain can be represented in a three-dimensional space whose dimensions are products, stores, and dates as shown in Figure 2.3. If more than three dimensions exist, the cube is called a *hypercube*. Each cell of the cube includes a value for each measure. Each dimension in the cube is associated to a *hierarchy* characterized by different levels of aggregation, called *attributes*. For instance, if we consider the dimension product (see Figure 2.2), a possible hierarchy aggregates the products (e.g., belt B) into subcategories (e.g., leather accessories) and the latter into categories (e.g., accessories).

2.1.2 Architectures

A common classification divides DW architectures in three main classes depending on the number of levels they involve.

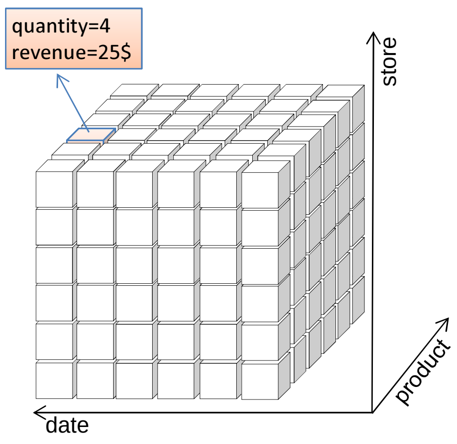


FIGURE 2.3: A three-dimensional cube modelling sales

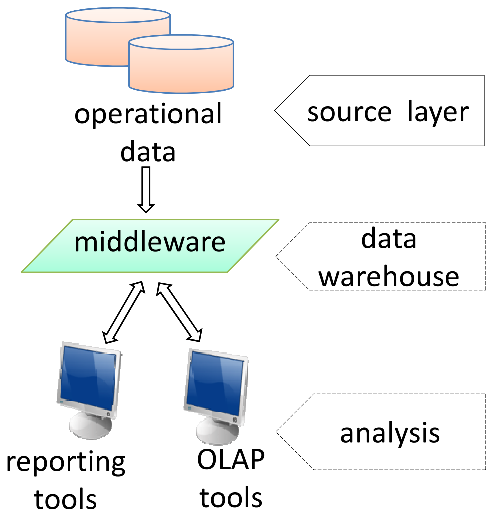


FIGURE 2.4: Single-layer architecture

- *Single-layer*: as shown in Figure 2.4 the architecture is characterized by a middleware (i.e., *data warehouse* layer) representing a virtual multidimensional view of the operational data. This intermediate level redirects the user query to the operational sources (i.e., *source layer*) and forwards the result to the OLAP layer (i.e., *analysis* layer), adapting the operational result to the multidimensional structure of the DW.

- *Two-layer*: in this architecture the DW layer is materialized. The DW can be represented by either a unique repository or different data marts. The *data staging* layer includes the ETL procedures (see Figure 2.5).

- *Three-layer*: Figure 2.6 shows that the result of ETL procedures is materialized in the *reconciled layer*. The other levels are equivalent to the ones of the two-layer architecture.

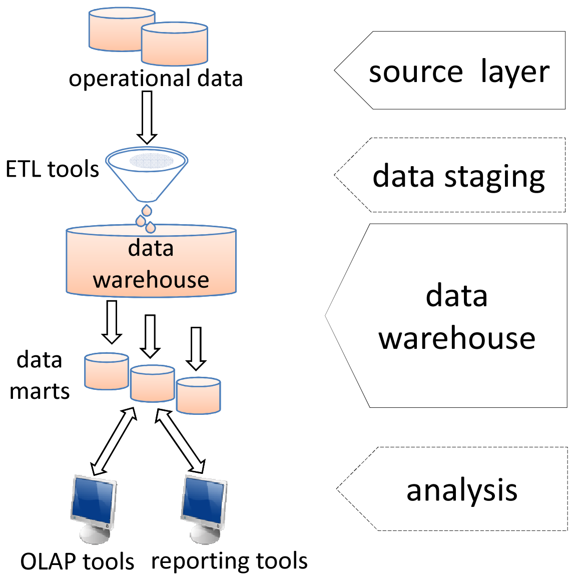


FIGURE 2.5: Two-layer architecture

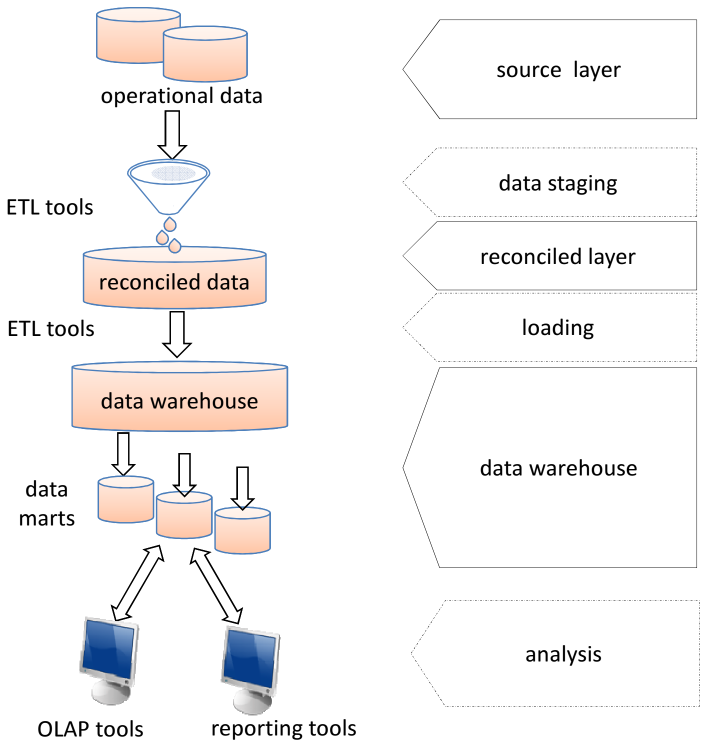


FIGURE 2.6: Three-layer architecture

2.1.3 ETL

The ETL process extracts, integrates, and cleans data from operational sources to feed the DW layer. In the following we report a brief description of the activities involved in the process, as proposed in [Golfarelli and Rizzi, 2009b]:

- *Extraction*: includes the extraction of data from the sources. We can distinguish between static and incremental extraction. We use a *static extraction* when the DW needs populating for the first time; on the contrary, an *incremental extraction* is used to update the DW regularly depending on the changes occurred in the operational data. Typically, a timestamp indicates when source data are changed or added.
- *Cleansing*: the cleansing procedures aim at improving the data quality. Typically, they are based on rectification and homogenization of the data to correct mistakes and inconsistencies:
  - **Duplicate data**: for example, a customer is recorded many times in the client database due to multiple registrations in different stores.
  - **Inconsistent values that are logically associated**: such as addresses and ZIP codes.
  - **Missing data**: such as the customer's income.
  - **Unexpected use of fields**: such as a comment field used improperly to store the fax number.
  - **Impossible or wrong values**: such as 2/30/2012.
  - **Inconsistent values for a single entity because different practises were used**: such as University of Bologna rather than Univ. of Bologna.
  - **Inconsistent values for own individual entity because of typing mistakes**: such as Oxford Steet instead of Oxford Street.
- *Transformation*: the source data are turned into the DW format. An integration procedure is required to manage data coming from different sources. To this end, a *matching* procedure is used to associates equivalent fields in different sources; a *selection* phase can reduce the number of source filed and records; finally, *conversion* and *normalization* procedures are applied to make data uniform. Typically, in the DW context, normalization is replaced by a *denormalization* phase to reduce the join operations at the query time.
- *Loading*: it is the last step of the ETL process. It can be carried out in two ways, *refresh* and *update*: in the first case, the DW is completely rewritten; in the second one, only those changes applied to source data are added to the DW.

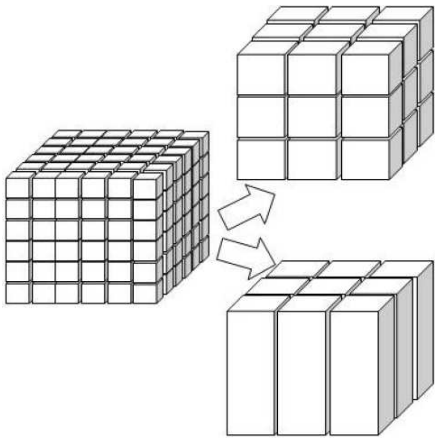


FIGURE 2.7: Roll-up operator

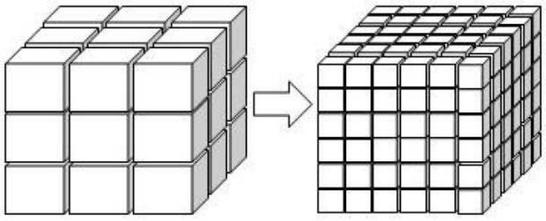


FIGURE 2.8: Drill-down operator

2.1.4 OLAP Analysis

OLAP analyses allow users to interactively navigate the DW information. Typically, the data are analyzed at different levels of aggregation, by applying subsequent OLAP operators, each yielding one or more different queries. The user can scout the multi-dimensional model choosing the next operator based on the outcome of the previous ones. In this way, the user creates a *navigation path* that corresponds to an analysis process for facts according to different points and at different detail levels. This is also informally called an *OLAP session*. In the following we describe the most common OLAP operators, referring to the cube of sales of Figure 2.3:

- **Roll-up** causes an increase in data aggregation and removes a detail level from a hierarchy (e.g., from product to subcategory), as shown in Figure 2.7.
- **Drill-down** is the complement to the roll-up operator; it reduces data aggregation and adds a new detail level to a hierarchy (e.g., from category to subcategory), as shown in Figure 2.8.
- **Slice-and-dice** reduces the number of cube dimensions after setting one of the dimensions to a specific value (e.g., product='belt B'); the dicing operation reduces the set of data being analyzed by a selection criterion (see Figure 2.9).

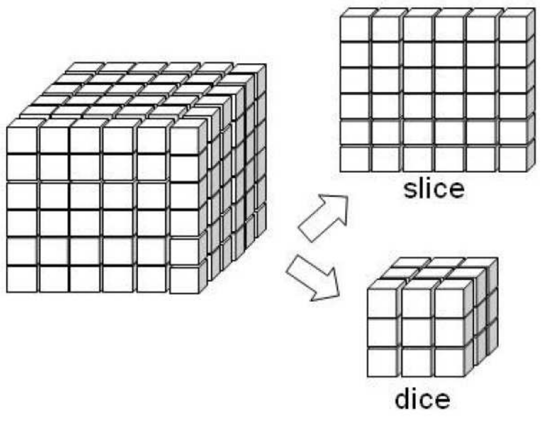


FIGURE 2.9: Slice and dice operators

- **Pivot** implies a change in layouts, aiming at analyzing a group of data from a different viewpoint.
- **Drill-across** allows to create a link between concepts in interrelated cubes, to compare them.
- **Drill-through** switches from multidimensional aggregate data to operational data in sources or in the reconciled layer.

2.2 Life-cycle Design

Typically, DW development relies on a bottom-up strategy that incrementally merges the different data marts the DW is composed of. As proposed in [Golfarelli and Rizzi, 2009b], data mart development is based on seven different phases:

- **Analysis and reconciliation of data sources**: it includes a detailed investigation of source schemata and a normalization phase to discover possible unexpressed relationships; the relevant data for the current data mart are selected, and its quality is assessed as well; if multiple sources exist, an integration process is required.
- **Requirement analysis**: the team collects the user requirements to define the main facts of the data mart and to design the preliminary workload.
- **Conceptual design**: it involves the multidimensional definition of the facts characterizing the data mart; each fact is described in terms of its measures, dimensions and hierarchies, producing a *fact schema*; a graphical specification for the fact schema is the *Dimensional Fact Model* (DFM).

- **Workload refinement, validation of conceptual schemata**: the preliminary workload is refined and the team checks that queries can be solved on the available conceptual schema, so as to validate it.
- **Logical design**: the team designs the logical implementation of the conceptual schema; the most common implementation is based on relational DBMSs and is called *Relational OLAP* (ROLAP).
- **Data staging design**: the design team and the users collaborate to define the updating process to populate both the reconciled layer and the data marts.
- **Physical design**: it involves the selection of indexes to optimize the DW performance.

These phases can accommodate both the classical approaches for data mart design: *data-driven* and *requirement-driven*. In the first case, the data mart schema is derived from the schema of source operational databases, while user requirements help designers choose facts, dimensions, and measures. In the second case, the data mart is designed starting from user requirements.

2.3 Multidimensional Model

In this section we introduce a basic formal setting to manipulate multidimensional data, and we introduce a running example based on the CENSUS [Minnesota Population Center, 2008] schema we will use in many chapters of the thesis.

Definition 2.1 (Multidimensional-Schema)**.** A multidimensional schema (or, briefly, an *md-schema*) is a triple $\mathcal{M} = \langle A, H, M \rangle$ where:

- $A = \{a_1, \dots, a_p\}$ is a finite set of *attributes*, each defined on a categorical domain $Dom(a_i)$;
- $H = \{h_1, \dots, h_n\}$ is a finite set of *hierarchies*, each characterized by (1) a subset $Attr(h_i) \subseteq A$ of attributes (such that the $Attr(h_i)$'s for $i = 1, \dots, n$ define a partition of A); (2) a *roll-up* total order $\succeq_{h_i}$ of $Attr(h_i)$ and a family of roll-up functions including a function $RollUp_{a_k}^{a_j} : Dom(a_k) \to Dom(a_j)$ for each pair of attributes a_k and a_j such that $a_k \succeq_{h_i} a_j$;
- $M = \{m_1, \dots, m_l\}$ is a finite set of *measures*, each defined on a numerical domain $Dom(m_i)$ and aggregable through a set of one or more aggregation operators, $Agg(m_i)$.

For each hierarchy h_i, the root attribute of the order is called *dimension*, denoted by DIM_i, and determines the finest aggregation level for the hierarchy. Conversely,

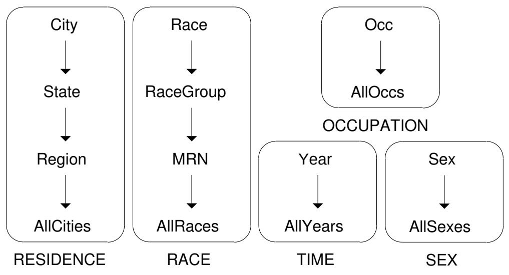


FIGURE 2.10: Roll-up orders for the five hierarchies in the CENSUS schema (MRN stands for MajorRacesNumber)

the bottom level is denoted by ALL_i, has a single possible value and determines the coarsest aggregation level. A pair $\mu = \langle m_i, \alpha_j \rangle$ such that $m_i \in M$ and $\alpha_j \in Agg(m_i)$ is called a *metric* of $\mathcal{M}$.

A group-by set includes one attribute for each hierarchy, and defines a possible way to aggregate data. A coordinate of a group-by set is a point in the n-dimensional space defined by the attributes in that group-by set.

Definition 2.2 (Group-by Set)**.** Given schema $\mathcal{M} = \langle A, H, M \rangle$, let $Dom(H) = Attr(h_1) \times \ldots \times Attr(h_n)$; each $G \in Dom(H)$ is called a *group-by set* of $\mathcal{M}$. Let $G = \langle a_{k_1}, \ldots, a_{k_n} \rangle$ and $Dom(G) = Dom(a_{k_1}) \times \ldots \times Dom(a_{k_n})$; each $g \in Dom(G)$ is called a *coordinate* of G.

Example 2.1. *The* CENSUS *schema includes five hierarchies, namely* RACE*,* TIME*,* SEX*,* OCCUPATION*, and* RESIDENCE*, and measures* AvgIncome*,* AvgCostGas*,* AvgCostWtr*, and* AvgCostElect*. It is* City $\succeq_{\text{RESIDENCE}}$ State *(the complete roll-up orders are shown in Figure 2.10); examples of group-by sets are:*

$$g_1 = \langle \text{State}, \text{Race}, \text{Year}, \text{AllSex}, \text{Occ} \rangle$$
$$g_2 = \langle \text{State}, \text{RaceGroup}, \text{Year}, \text{AllSex}, \text{Occ} \rangle$$
$$g_3 = \langle \text{Region}, \text{AllRaces}, \text{Year}, \text{Sex}, \text{Occ} \rangle$$

A schema is populated with facts, each recording a useful information for the decision-making process. A fact is characterized by a group-by set G that defines its aggregation level, by a coordinate of G, and by a value for one measure.

Definition 2.3 (Fact)**.** Given schema $\mathcal{M} = \langle A, H, M \rangle$, a group-by set $G \in Dom(H)$, and a measure $m \in M$, a fact is a couple $f_{G,m} = \langle g, v \rangle$, where $g \in Dom(G)$ and $v \in Dom(m)$. The space of all facts for $\mathcal{M}$ is

$$\mathcal{F}_{\mathcal{M}} = \bigcup_{G \in Dom(H), m \in M} (Dom(G) \times Dom(m))$$

Finally, an instance of a schema (*datacube*) is a set of facts $D \subseteq \mathcal{F}_{\mathcal{M}}$ such that no two facts characterized by the same coordinate exist in D.

Chapter 3

Distributed BI

In this chapter we describe the BIN framework to support the concept of BI FROM-ANYWHERE. The framework is aimed at manipuliting business information from different DW tools, creating complex networks of companies chasing mutual advanteges through the sharing of BI information and functionalities. A BIN is a peer-to-peer data warehousing architecture, where each peer exposes query answering functionalities. To enhance the decision making process, an OLAP query expressed on a peer needs be properly reformulated on the local multidimensional schemata of the other nodes. To this end, we present a language for the definition of mappings between the multidimensional schemata of peers and we introduce a query reformulation framework that relies on the translation of mappings, queries, and multidimensional schemata onto the relational level. Then, we formalize a query reformulation algorithm and prove two properties: correctness and closure, that are essential in a peer-to-peer setting. Finally, we describe the main implementation issues to develop a BIN.

3.1 Introduction

As mentioned in Chapter 1, one of the key features for BI in 2012 is the ability to access information anywhere it can be found, by locating it through a semantic process and performing integration on the fly. This is particularly relevant in inter-business collaborative contexts where companies organize and coordinate themselves to share opportunities, respecting their own autonomy and heterogeneity but pursuing a common goal. In such a complex and distributed business scenario, traditional BI systems —that were born to support stand-alone decision-making— are no longer sufficient to maximize the effectiveness of monitoring and decision making processes. Accessing local information is no more enough, users need to transparently and uniformly access information scattered across several heterogeneous BI platforms [Hoang and Nguyen, 2009].

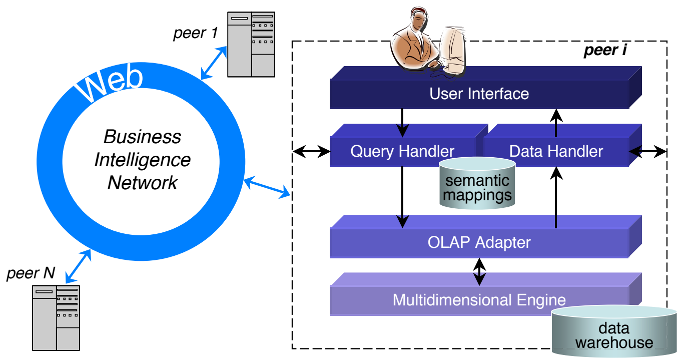


Figure 3.1: Envisioned architecture for a BIN

To fill this gap, we envision BIN ([Golfarelli et al., 2010, 2011a, 2012a,b]), a peer-to-peer data warehousing architecture sketched in Figure 3.1. A BIN is an architecture for sharing BI functionalities across a dynamic and collaborative network of heterogeneous and autonomous peers. Each peer is equipped with an independent DW system, that relies on a local multidimensional schema to represent the peer's view of the business and exposes OLAP query answering functionalities (based for instance on the MDX language, a de-facto standard for querying multidimensional databases) aimed at sharing business information, in order to enhance the decision making process and create new knowledge. The main benefits the BIN approach aims at delivering to the corporate world are the possibility of building new inter-organizational relationships and coordination approaches, and the ability to efficiently manage inter-company processes and safely sharing management information besides operational information.

The core idea of a BIN is that of enabling users to transparently access business information distributed over the network. A typical interaction sequence is the following:

1. A user formulates an OLAP query q by accessing the local multidimensional schema exposed by her peer, p.
2. Query q is processed locally on the DW of p.
3. At the same time q is forwarded to the network.
4. Each involved peer locally processes the query on its DW and returns its results to p.
5. The results are integrated and returned to the user.

The local multidimensional schemata of peers are typically heterogeneous; so, before a query issued on a peer can be forwarded to the network, it must be first *reformulated* according to the multidimensional schemata of the destination peers. In line with the approach adopted in *Peer Data Management Systems* (PDMSs) [Halevy et al., 2004], query reformulation in a BIN is based on *semantic mappings* that mediate between the different multidimensional schemata exposed by two peers, i.e., they describe how the concepts in the multidimensional schema of one peer map onto those of another peer.

Direct mappings cannot be realistically defined for all the possible couples of peers. So, to enhance information sharing, a query q issued on p is forwarded to the network by first sending it to the neighborhood of p; then, each peer in this neighborhood in turn sends q to its neighborhood, and so on. In this way, q undergoes a chain of reformulations along the peers it reaches, and results are collected from any peer that is connected to p through a path of semantic mappings.

The approach outlined above is reflected by the internal architecture of each peer, sketched in the right side of Figure 3.1, whose components are:

1. *User Interface.* A web-based component that manages bidirectional interaction with users, who use it to visually formulate OLAP queries on the local multidimensional schema and explore query results.
2. *Query Handler.* This component receives an OLAP query from either the user interface or a neighboring peer on the network, sends that query to the OLAP adapter to have it locally answered, reformulates it onto the neighborhood (using the available semantic mappings), and transmits it to the peers in that neighborhood.
3. *Data Handler.* When the peer is processing a query that was locally formulated, the data handler collects query results from the OLAP adapter and from the peers, integrates them, and returns them to the user interface. When the peer is processing a query that was formulated on some other peer p, the data handler just collects local query results from the OLAP adapter and returns them to p.
4. *OLAP Adapter.* This component adapts queries received from the query handler to the querying interface exposed by the local multidimensional engine.
5. *Multidimensional Engine.* It manages the local DW according to the multidimensional schema representing the peer's view of the business, and provides MDX-like query answering functionalities.

Interactions between peers are based on a message-passing protocol.

Query answering in a BIN architecture poses several research challenges, ranging from languages and models for semantic mediation to query reformulation issues and proper techniques and data structures for the query processing phase. Much work has been done on these issues in the context of PDMSs (e.g., [Mandreoli et al., 2006, 2007, 2009]) and relational databases [Penzo, 2005], however those results are not directly applicable in the OLAP scenario presented by the BIN. A more detailed explanation of the existing approaches is provided in Section 3.2.

The rest of the chapter is organized as follows:

- In Section 3.4, we present a language for the definition of semantic mappings between the schemata of peers, using predicates that are specifically tailored for the multidimensional model. To overcome possible differences in data formats, mappings can be associated with transcoding functions. Mappings are classified according to the accuracy they allow in query reformulation and a set of requirements for the query reformulation algorithm is introduced when dealing with mapping of different accuracies.
- In Section 3.5, we introduce a framework for OLAP query reformulation that relies on the translation of mappings and queries towards the underlying relational schemata. For simplicity, we will use standard star schemata to this end.
- In Section 3.6, we propose a query reformulation algorithm and show that it is correct for compatible reformulations, that it satisfies all the requirements, and that our language for expressing OLAP queries is closed under reformulation. Remarkably, this means that our reformulation algorithm can be safely used to implement chains of reformulations as required by the BIN setting. Besides, Appendix A gives the proofs of the theorems the algorithm is based on.
- In Section 3.7, we discuss the main implementation issues.

3.2 Related Works

In this section we discuss the related works, comparing those specifically placed in the DW context from those in the OLTP context. We focus on three different architectural approaches to share information in the two fields: warehousing, federative, and P2P approaches.

In the OLTP context, the research area sharing most similarities with warehousing approaches to the concept of BI FROM-ANYWHERE is data exchange. In data exchange, data structured under one source schema must be restructured and translated into an instance of a different target schema, that is materialized [Fagin et al., 2003]. In this scenario, the target schema is often independently created and comes with its own constraints that have to be satisfied. On the other hand, federative approaches have their OLTP counterpart in data integration systems. Data from different sources are combined to give users a unified view [Lenzerini, 2002]; in this way, users are freed from having to locate individual sources, learn their specific interaction details, and manually combine the data [Halevy, 2010]. The unified view that reconciles the sources is represented by a global schema. In this case query processing requires a reformulation step: a query over the global, target schema has to be reformulated in terms of a set of queries over the sources. Finally, P2P approaches to support BI

FROM-ANYWHERE are related to the decentralized sharing of OLTP data between autonomous sources, that has been deeply studied in the context of PDMSs. PDMSs were born as an evolution of mediator systems in the data integration field [Halevy et al., 2004] and generalize data exchange settings [Fuxman et al., 2005]. A PDMS consists of a set of peers, each with an associated schema representing its domain of interest; peer mediation is implemented by means of semantic mappings between portions of schemata that are local to a pair or a small set of peers. Every peer can act freely on its data, and also access data stored by other peers without having to learn their schema and even without a mediated schema [Tatarinov and Halevy, 2004]. In a PDMS there is no a priori distinction between source and target, since a peer may simultaneously act as a distributor of data (thus, a source peer) and a recipient of data (thus, a target peer). As in the case of data integration systems, in a PDMS data remain at the sources and queries processing entails query reformulation over the peer schemata. In all these contexts, modeling the relationships (mappings) between source and target schemata is a crucial aspect. Research in the data integration area has provided rich and well-understood schema mediation languages [Lenzerini, 2002] to this end. The two commonly used formalisms are the *global-as-view* (GAV) approach, in which the mediated (global) schema is defined as a set of views over the data sources, and the *local-as-view* (LAV) approach, in which the contents of data sources are described as views over the mediated schema. Depending on the kind of formalism adopted, GAV or LAV, queries posed to the system are answered differently, namely by means of query unfolding or query rewriting techniques [Halevy, 2001], respectively. In a data exchange setting, assertions between a source query and a target query are used to specify what source data should appear in the target and how. These assertions can be represented neither in the LAV nor in the GAV formalisms, but rather they can be thought of as *global-and-local-as-view* GLAV [Fagin et al., 2003]. A structural characterization of schema mapping languages is provided in [ten Cate and Kolaitis, 2010], together with a list of the basic tasks that all languages ought to support. In distributed OLTP environments, the schema mapping generation phase and the preceding schema matching phase pose new issues with reference to simpler centralized contexts: consistency problems are studied in [Cudré-Mauroux et al., 2006] and innovative learning techniques are presented in [Madhavan et al., 2005]. Other studies in the field have focused on integrating the computation of core solutions in the mapping generation process, aimed at determining redundancy-free mappings in data exchange settings [Fagin et al., 2005, Mecca et al., 2009]. Declaring useful mappings in the OLAP context necessarily requires also the level of instances to be taken into account. Unfortunately, in the OLTP literature the definition of mappings is typically done at the schema level, and the problem of managing differences in data formats has only marginally been considered. A seminal paper regarding this topic is [Chang and Garcia-Molina, 1999], where constraint queries are translated across heterogeneous information sources taking into account differences in operators and data formats. A related problem is that of reconciliation of results, that takes a primary role in federative and P2P approaches. In the OLTP context this issue is referred to as object fusion [Papakonstantinou et al., 1996]. This involves grouping together information (from the same or different sources) about the same real-world entity. In doing this fusion, the mediator may also refine the information by removing redundancies, resolving inconsistencies between sources in favor of the most reliable source, and so on.

3.2.1 Warehousing Approaches

As already mentioned, in this family of approaches the data that result from the process of integrating a set of component DWs according to a global schema are materialized. The main drawback of these approaches is that they can hardly support dynamic scenarios like those of mergers and acquisitions. An approach in this direction is the one proposed in [Torlone, 2008]. Given two dimensions belonging to different data marts where a set of mappings between corresponding levels has been manually declared or automatically inferred, three properties (namely coherence, soundness, and consistency) that enable a compatibility check between the two dimensions are defined. A technique that combines the contents of the dimensions to be integrated is then used to derive a materialized view that includes the component data marts. A hybrid approach between the warehouse and the federation approach is suggested in [Jiang et al., 2007] as a way to obtain a more flexible and applicable architecture. The idea is to aggregate selected data from the component DWs as materialized views and cache them at a federation server to improve query performance; a set of materialized query tables are recommended for the benefits of load distribution and easy maintenance of aggregated data. Another borderline approach is proposed in [Berger and Schrefl, 2008]: while fact data are not physically integrated, a central dimension repository is used to replicate dimensional data (according to a global schema) from the component DWs, aimed at increasing querying efficiency. To effectively cope with evolutions in the schema of the components, a fact algebra and a dimension algebra are used in this approach for declaring maintainable mappings between the component schemata.

3.2.2 Federative Approaches

A federated DW, sometimes also called distributed DW, is a logical integration of DWs that provides transparent access to the component DWs across the different functions of an organization. This is achieved through a global schema that represents the common business model of the organization [Jindal and Acharya, 2004]. Differently from warehousing approaches, the integrated data are not physically stored, so queries formulated on the global schema must be rewritten on the component schemata. This adds complexity to the query management framework, but enables more flexible architectures where new component DWs can be dynamically inserted. A distributed

DW architecture is outlined in [Albrecht and Lehner, 1998], and a prototype named CubeStar for distributed processing of OLAP queries is introduced. CubeStar includes a middleware layer in charge of making the details of data distribution transparent to the front-end layer, by generating optimized distributed execution plans for user queries. A distributed DW architecture is considered also in [Akinde et al., 2003] as a solution for contexts where the inherently distributed nature of the data collection process and the huge amount of data extracted make the adoption of a central repository impractical. The Skalla system for distributed query processing is proposed, with particular emphasis on techniques for optimizing both local processing and communication costs; however, since it is assumed that all collection points share the same schema, the approach cannot be used to cope with heterogeneous settings. In the context of a federated architecture, with specific reference to the healthcare domain, the work in [Banek et al., 2006, 2008] presents an algorithm for matching heterogeneous multidimensional structures, possibly characterized by different granularities for data. Mappings between the local schemata of the DWs to be integrated and a given global schema are discovered in a semi-automated manner, based on a measure of similarity between complex concepts. A process to build an integrated view of a set of DWs is outlined in [Schneider, 2006]. This integrated view is defined as the largest common schema to all the components, and its instances are obtained by merging the instances of the components. In [Torlone, 2008], the problem of virtual integration of heterogeneous data marts is faced in a loosely-coupled scenario where there is a need for identifying the common information (intuitively, the intersection) between the components while preserving their autonomy. A set of rules to check for dimension compatibility are declared first, then drill-across queries are used to correlate on-the-fly the component data marts. A multi DW system is introduced in [Berger and Schrefl, 2006] as one relying on a distributed architecture where users are enabled to directly access the heterogeneous schemata of the component DWs, which makes the coupling between the components looser than in federated DWs. A SQL-MDi query language is proposed to transform a cube in order to make it compatible with a global, virtual cube and ready for integration. Specific attention is devoted to solving schema and instance conflicts among the different components. An XML-based framework for supporting interoperability of heterogeneous DWs in a federation scenario is described in [Mangisengi et al., 2001]. In the proposed architecture, a federated layer allows for restructuring and merging local data and schemas to provide a global, single view of the component DWs to the end users. XML is used both to represent the local DW schemata, the global schema, and the mapping between them. Another XML-based approach is the one in [Tseng and Chen, 2005], that discusses the possible conflicts arising when heterogeneous DWs are integrated and proposes solutions to resolve the semantic discrepancies. Data cubes are transformed into XML documents and queried under a global view. XML topic maps are used in [Bruckner et al., 2001] to integrate the information stored in distributed DWs. The schema integration process is based on merging local topic maps to generate global topic maps, taking different types of semantic conflicts into account. A different approach is presented in [Zhou et al., 2000], that introduces an architecture for hierarchically distributed DWs where component DWs are organized into a tree and data are progressively summarized level over level. A local OLAP query can be posed at any node of the tree, it is rewritten on remote nodes, and the results are merged.

3.2.3 Peer-to-Peer Approaches

Though federative approaches support more flexible and dynamic architectures than warehousing ones, still they do not fully preserve the autonomy of individual actors. In complex business scenarios where no leadership can be established among a set of actors interested in cooperating, to maximize the effectiveness of monitoring and decision making processes there is a need for truly decentralized approaches. This can be achieved by relying on P2P architectures. In [Abiteboul, 2003, Abiteboul et al., 2005], the authors introduced the idea of using a P2P architecture for warehousing XML content. In their view, a P2P warehouse is not different from a centralized one from the logical point of view, while from the physical point of view information is distributed over a set of heterogeneous and autonomous peers rather than centralized. Because of this, query processing necessarily requires distributed computation. Among the advantages of this approach, we mention ownership (each peer has full control over its information) and dynamicity (peers can transparently enter and leave the system). How to map the local schema of each peer onto each other is one of the open problems. The approach proposed in [Miller et al., 2000] reformulates XML queries over a set of peers hosting XML databases with heterogeneous (and possibly conflicting) schemata, in the absence of a global schema. Reformulation is based on mapping rules inferred from informal schema correspondences. In [Espil and Vaisman, 2004, Vaisman et al., 2009] the authors present a model for multidimensional data distributed across a P2P network, together with a mapping-based technique for rewriting OLAP queries over peers. In presence of conflicting dimension members, an approach based on belief revision is proposed to revise the instance of the source peers dimension and adapt it to the instance of the target peers dimension. Another work centered on interoperability issues among heterogeneous DWs is the one by [Kehlenbeck and Breitner, 2009], that emphasizes the importance of a semantic layer to enable communication among different components. This approach supports the exchange of business calculation definitions and allows for their automatic linking to specific component DWs through semantic reasoning. Three models are suggested: a business ontology, a DW ontology, and a mapping ontology between them. As to performance aspects, in [Kalnis et al., 2002] the authors propose a P2P architecture for supporting OLAP queries focusing on the definition of a caching model to make the query rewriting process more efficient. They also define adaptive techniques that dynamically reconfigure the network structure in order to minimize the query cost. Finally, as to the data reconciliation, a typical requirement in collaborative BI is the merging of results at different levels of aggregation. In this direction, the work proposed in [Dubois and Prade, 2004] discusses a general approach on the use of aggregation operations in information fusion processes and suggests practical rules to be applied in common scenarios.

3.3 Formal Background

In this section we extend Definition 2.1 of multidimensional schema and Definition 2.2 of group-by set. We also introduce two new concepts, *transcoding* and *BIN query*, and we describe two different multidimensional schemata we adopt as reference points in this chapter.

As concerns the multidimensional schema, we consider more complex hierarchies characterized by branches. To this purpose, we relax the definition of hierarchy considering a *roll-up* **partial** order $\succeq_{h_i}$ of $Attr(h_i)$. According to the extended definition of hierarchy, we can consider a group-by set including **more** than one attribute for each hierarchy. To this end, we overwrite the definition of group-by set by using the concept of *projection*.

Definition 3.1 (Projection)**.** Given a multidimensional schema $\mathcal{M} = \langle A, H, M \rangle$, a *projection* of $\mathcal{M}$ is a subset of attributes $P \subseteq A$. The domain of P is $Dom(P) = \times_{a_i \in P} Dom(a_i)$; each value of $Dom(P)$ is called a *coordinate* of P.

We clarify the new definition of multidimensional schema and projection with a reference example:

Example 3.1. *A set of local health-care departments participate in a collaborative network to integrate their data about admissions so as to enable more effective analysis of epidemics and health-care costs by the Ministry. For simplicity we will focus on two peers: the first, located in Rome, hosting data on hospitalizations at the most detailed level; the second, located in Florence, hosting data on admissions grouped by patient gender, residence city, and birth year. The underlying md-schemata for Rome and Florence are called* HOSPITALIZATION *and* ADMISSIONS*, respectively; their roll-up orders are shown in Figure 3.2.*

Assuming that each hierarchy is named after its finest-level attribute, but capitalized, relationships $DIM_{\mathsf{Patient}} = \mathsf{patient}$ *and* $\mathsf{city} \succeq_{\mathsf{Patient}} \mathsf{region}$ *hold. The* HOSPITALIZATION *md-schema includes measures* cost *and* durationOfStay*;* ADMISSIONS *includes measures* totStayCost, totExamCost, totLength, maxLength, *and* numAdmissions*. The*

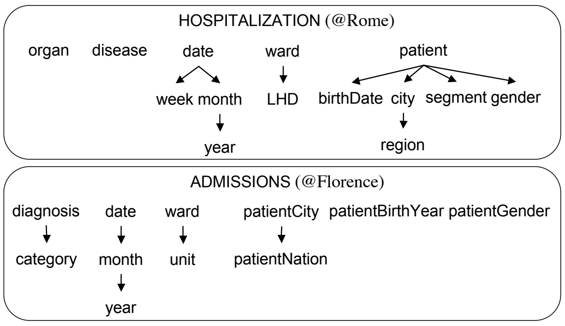


FIGURE 3.2: Roll-up orders for the hierarchies in the HOSPITALIZATION and ADMISSIONS multidimensional-schemata (LHD stands for local health department)

aggregation operators exposed by the two md-schemata are as follows:

$$Agg(\mathsf{cost}) = \{\mathtt{sum}, \mathtt{avg}\}$$
$$Agg(\mathsf{durationOfStay}) = \{\mathtt{sum}, \mathtt{avg}, \mathtt{min}, \mathtt{max}\}$$
$$Agg(\mathsf{totStayCost}) = Agg(\mathsf{totExamCost}) = Agg(\mathsf{totLength}) = \{\mathtt{sum}, \mathtt{avg}\}$$
$$Agg(\mathsf{maxLength}) = \{\mathtt{max}\}$$
$$Agg(\mathsf{numAdmissions}) = \{\mathtt{sum}\}$$

Note that the HOSPITALIZATION *md-schema stores data at the maximum detail, so all its measures can in principle be aggregated using any operator. On the other hand,* ADMISSIONS *stores pre-aggreggated data, so the (additive) measures* totStayCost*,* totExamCost*, and* totLength *can be also averaged thanks to the presence of* numAdmissions*, that acts as a support measure for the* avg *operator.*

Examples of projections of HOSPITALIZATION *and* ADMISSIONS *are* $P = \{\mathsf{week}, \mathsf{region}\}$ *and* $P' = \{\mathsf{date}, \mathsf{patientCity}\}$*, respectively.*

Definition 3.2 (Transcoding)**.** Given the multidimensional schema $\mathcal{M}$, let *Dom* be a generic domain of values. A *transcoding of* $\mathcal{M}$ is a function $f : Dom(P) \to Dom$, where P is a projection of $\mathcal{M}$, that maps each coordinate of P onto a value of *Dom*.

Note that *Dom* can be a compound domain (e.g., $Dom = Dom(\mathsf{week}) \times Dom(\mathsf{region})$); in this case, the transcoding is made of *components*, each mapping onto a simple domain.

Example 3.2. *A transcoding of* ADMISSIONS *is* $f : Dom(P') \to Dom(P)$ *whose components are* week = `weekOf`(date), region = `regionOf`(patientCity)*, where* `weekOf()` *is a common SQL function and* `regionOf()` *is a user-defined function that returns the region a city belongs to by accessing a* CITIES *relational table stored at the Florence peer.*

As concerns BIN queries, we will consider a simple form of OLAP queries characterized by an aggregation and a selection (*GPSJ - Generalized Projection / Selection /*

Join query, [Gupta et al., 1995]), where transcodings can be applied to attributes and measures can appear within expressions. To avoid getting burdened with the details of a specific multidimensional query language, we will express queries using an abstract syntax.

Definition 3.3 (BIN query)**.** A *BIN query* is a 5-tuple $q = \langle \mathcal{M}, E, p, expr, T\rangle$ where:

1. $\mathcal{M} = \langle A, H, M\rangle$ is the md-schema q is formulated on;
2. E is a generalized query group-by set, and it is a set where each element is either an attribute of $\mathcal{M}$ or a component of a transcoding of $\mathcal{M}$;
3. p is an (optional) *selection predicate*; it is a conjunction of Boolean predicates, each involving either an attribute of $\mathcal{M}$ or a component of a transcoding of $\mathcal{M}$;
4. *expr* is the expression computed by q; it is a numerical expression involving measures in M;
5. T is a list of metrics of $\mathcal{M}$, one for each measure used in *expr*, expressing the operators that will be used for aggregation.

Consistently with the behavior of the MDX language, the semantics we assume for BIN queries is that aggregation is executed first. This means that q returns an expression of aggregates (rather than an aggregation of expressions).

Example 3.3. *The query*

$$\begin{aligned} q_1 = \langle &\mathsf{HOSPITALIZATION},\\ &\{\mathsf{region}, \mathsf{year}\}, (\mathsf{gender} = \textit{'Female'}),\\ &\mathsf{cost}, \langle\langle \mathsf{cost}, \mathtt{sum}\rangle\rangle \end{aligned}$$

computes, at the Rome peer, the total hospitalization cost of female patients for each region and year. The query

$$\begin{aligned} q_2 = \langle &\mathsf{ADMISSIONS},\\ &\{\mathsf{year}, \mathtt{regionOf}(\mathsf{patientCity})\}, —,\\ &\mathsf{totLength}, \langle\langle \mathsf{totLength}, \mathtt{avg}\rangle\rangle \end{aligned}$$

computes, at the Florence peer, the yearly average admission length for each patient region.

3.4 Mapping Language

In this section we describe the language we devised for the definition of semantic mappings between the md-schemata of peers. As mentioned in Section 3.1, these mappings play a key role in a BIN because, as we will show in Section 3.5, they enable query reformulation. After introducing a set of mapping predicates in Subsection 3.4.1, in Subsection 3.4.2 we informally discuss how the mapping predicates introduced can lead to query reformulations at different levels of accurateness, and we derive a set of requirements for our reformulation algorithm accordingly.

3.4.1 Mapping Predicates

We preliminarily note that, according to the PDMS terminology, data are extracted from a source peer and are mapped onto the schema of a target peer. Accordingly, we will name the peer on whose md-schema a BIN query q is originally formulated as *target* peer, and the one on whose md-schema q has to be reformulated as *source* peer.

The language we propose to express how the md-schema $\mathcal{M}_s$ of a source peer s maps onto the md-schema $\mathcal{M}_t$ of a target peer t includes five *mapping predicates*, namely `same`, `equi-level`, `roll-up`, `drill-down`, and `related` that will be discussed in detail below. In general, a mapping establishes a semantic relationship from one or more concepts (either measures or attributes) of $\mathcal{M}_s$ to one or more concepts of $\mathcal{M}_t$, and enables a BIN query formulated on $\mathcal{M}_t$ to be reformulated on $\mathcal{M}_s$. Optionally, a mapping involving attributes can be annotated with a transcoding that specifies how values of the target concepts can be obtained from values of the source concepts. If this function is available, it is used to increase the reformulation effectiveness.

- `same` predicate: μ_t $\texttt{same}_{expr,p}$ N_s, where $\mu_t = \langle m_t, \alpha_t \rangle$ is a metric of $\mathcal{M}_t$, N_s is a subset of measures of $\mathcal{M}_s$, and *expr* is an expression involving the measures in N_s. This mapping predicate is used to state that whenever m_t is asked in a query on $\mathcal{M}_t$ using α_t, it can be rewritten as *expr* on $\mathcal{M}_s$. The `same` mapping predicate can be annotated with a conjunctive Boolean predicate p involving one or more attributes in $\mathcal{M}_t$, to restrict the validity of the rewriting for metric μ_t.

- `equi-level` predicate: P_t $\texttt{equi-level}_f$ P_s, where P_t and P_s are projections of $\mathcal{M}_t$ and $\mathcal{M}_s$, respectively. This predicate is used to state that P_t has the same semantics and granularity as P_s. Optionally, it can be annotated with an injective transcoding $f : Dom(P_s) \rightarrow Dom(P_t)$ that establishes a one-to-one relation between coordinates of P_s and P_t, and is used to integrate data returned by the source and target peers.

- `roll-up` predicate: P_t $\texttt{roll-up}_f$ P_s. This predicate states that P_t is a roll-up of (i.e., it aggregates) P_s. Optionally, it can be annotated with a non-injective transcoding $f : Dom(P_s) \rightarrow Dom(P_t)$ that establishes a many-to-one relation between coordinates of P_s and P_t, and is used to aggregate data returned by the source peer and integrate them with data returned by the target peer.

TABLE 3.1: Mappings from Florence (source peer) to Rome (target peer)

$\omega 1$	⟨ cost,sum ⟩ `same` { totStayCost, totExamCost }
$\omega 2$	⟨ cost,avg ⟩ `same` { totStayCost, totExamCost }
$\omega 3$	⟨ durationOfStay,sum ⟩ `same` { totLength }
$\omega 4$	⟨ durationOfStay,avg ⟩ `same` { totLength }
$\omega 5$	⟨ durationOfStay,max ⟩ `same` { maxLength }
$\omega 6$	{ LHD } `roll-up` { unit }
$\omega 7$	{ ward } `equi-level` { ward }
$\omega 8$	{ year } `equi-level` { year }
$\omega 9$	{ month } `equi-level` { month }
$\omega 10$	{ date } `equi-level` { date }
$\omega 11$	{ week } `roll-up` { date }
$\omega 12$	{ disease,organ } `equi-level` { diagnosis }
$\omega 13$	{ disease } `drill-down` { category }
$\omega 14$	{ patient } `drill-down` { patientGender,patientCity,patientBirthYear }
$\omega 15$	{ gender } `equi-level` { patientGender }
$\omega 16$	{ segment } `related` { patientGender,patientCity,patientBirthYear }
$\omega 17$	{ birthDate } `drill-down` { patientBirthYear }
$\omega 18$	{ city } `equi-level` { patientCity }
$\omega 19$	{ region } `roll-up` { patientCity }

- `drill-down` predicate: P_t $\texttt{drill-down}_f$ P_s. This predicate is used to state that P_t is a drill-down of (i.e., it disaggregates) P_s. Optionally, it can be annotated with a non-injective transcoding $f : Dom(P_t) \rightarrow Dom(P_s)$ that establishes a one-to-many relation between coordinates of P_s and P_t. The transcoding f cannot be used to integrate data returned by t and s because this would require disaggregating data returned by s, which obviously cannot be done univocally; however, it can be used to reformulate selection predicates expressed at t onto s.

- `related` predicate: P_t `related` P_s. This predicate is used to state that P_t coordinates have a many-to-many relationship with P_s coordinates.

Example 3.4. *The complete set of mappings and annotations for our health-care example is reported in Tables 3.1 and 3.2. Mappings $\omega 1$ and $\omega 2$ state that measure* cost *in Rome can be derived by summing measures* totStayCost *and* totExamCost *in Florence. Mappings from $\omega 1$ to $\omega 5$ are also annotated with predicate* segment `in` *{'NH','EU'}, to state that the Florence peer only stores National Health and European patients. On the other hand, as shown in Figure 3.3, mapping $\omega 12$ states that the diagnosis codes used in Florence are obtained by concatenating the fixed-length disease and organ codes used in Rome, and mapping $\omega 11$ states that weeks are an aggregation of dates. Finally, mapping $\omega 15$ uses as transcoding a* `completeGender()` *function that converts values 'M' and 'F' (Florence vocabulary) into 'Male' and 'Female' (Rome vocabulary).*

TABLE 3.2: Annotations to the mappings in Table 3.1

$\omega1$	cost = totStayCost+totExamCost, segment **in** { 'NH','EU' }
$\omega2$	cost = totStayCost+totExamCost, segment **in** { 'NH','EU' }
$\omega3$	durationOfStay = totLength, segment **in** { 'NH','EU' }
$\omega4$	durationOfStay = totLength, segment **in** { 'NH','EU' }
$\omega5$	durationOfStay = maxLength, segment **in** { 'NH','EU' }
$\omega6$	LHD = 'LHD39 - Florence'
$\omega7$	ward = ward
$\omega8$	year = year
$\omega9$	month = month
$\omega10$	date = date
$\omega11$	week = `weekOf`(date)
$\omega12$	disease = `substring`(diagnosis, 1, 40), organ = `substring`(diagnosis, 41, 80)
$\omega13$	`categoryOf`(disease) = category
$\omega14$	—
$\omega15$	gender = `completeGender`(patientGender)
$\omega16$	—
$\omega17$	`yearOf`(birthDate) = patientBirthYear
$\omega18$	city = patientCity
$\omega19$	region = `regionOf`(patientCity)

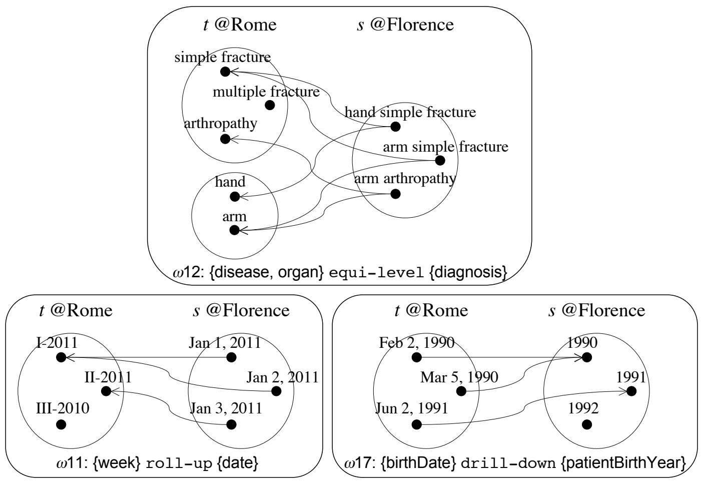


FIGURE 3.3: Transcoding examples

3.4.2 Mapping Accuracy

We start this section by classifying mappings according to their accuracy.

Definition 3.4. We say mapping ω is *exact* iff it is either an ***equi-level*** or a ***roll-up*** mapping and it has an associated transcoding, or it is a ***same*** mapping. A mapping is said to be *loose* when it is either a ***drill-down*** or a ***related*** mapping. An attribute mapping is said to be *approximate* when it has no associated transcoding.

In our example, mappings ω_{14} and ω_{16} are approximate and loose; mappings ω_{13} and ω_{17} are loose (but not approximate); all the other mappings are exact.

Let q be a BIN query formulated at peer t. The accuracy of a reformulation of q on peer s depends on the accuracy of the mappings involved. In the following, we focus on this aspect and provide a set of requirements for the query reformulation algorithm when dealing with mappings of different accuracies.

Intuitively, when (i) for each attribute mentioned in q there is an exact mapping from $\mathcal{M}_s$, and (ii) for each metric required by q there is a `same` mapping from $\mathcal{M}_s$, there exists a *compatible* reformulation of q on s, i.e., one that fully preserves the semantics of q. When a compatible reformulation is used, the results returned by s do *exactly* match with q so they can be seamlessly integrated with those returned by t. For instance, query q_1 formulated at the Rome peer in Example 3.3 has a compatible reformulation at the Florence peer: [1]

$$
\begin{aligned}
q'_1 = \langle & \mathsf{ADMISSIONS}, \\
& \{\texttt{regionOf}(\mathsf{patientCity}), \mathsf{year}\}, (\texttt{completeGender}(\mathsf{patientGender}) = \text{'Female'}), \\
& \mathsf{totStayCost} + \mathsf{totExamCost}, \langle\langle \mathsf{totStayCost}, \texttt{sum}\rangle, \langle \mathsf{totExamCost}, \texttt{sum}\rangle\rangle\rangle
\end{aligned}
$$

Of course, when a compatible reformulation exists for a query, it must be correctly generated by the reformulation algorithm.

In all the other cases, the results returned by s match with q with some approximation. Three (possibly overlapping) situations can be distinguished:

1. For at least one of the attributes mentioned in q, there is an approximate mapping from $\mathcal{M}_s$. This is a very common situation in real applications, for instance when proprietary encoding are used for attributes, because building a reliable transcoding would require a huge effort. However, the data returned by the source peer can be understood and useful, so we require that a reformulation is generated though this will lead to a *value mismatch*, meaning that the data returned by s and t will not be integrated. For instance, if mapping ω_8 were not annotated with a transcoding for year, query q_1 would still be reformulated at the Florence peer as q'_1, but there would be no guarantee that the year values returned can be integrated with those returned from the Rome peer.

2. For at least one of the attributes mentioned in q, either there is a loose mapping from $\mathcal{M}_s$ or even there is no mapping at all. Also this situation is common, because independent md-schemata often have different granularities and hierarchies have different attributes. We require that a reformulation is generated, knowing that this will lead to a *granularity mismatch*, meaning that the data returned by s and t will have different aggregation levels. Although an integration is not

[1]As a matter of fact, the reformulation generated for q_1 by the algorithm proposed in Section 3.6 includes one more predicate that constrains source data based on mapping ω_{13}. Here we do not report this predicate for simplicity.

possible, users can still exploit the results coming from s, for example by comparing them with those returned by t at the finest common aggregation level. For instance, if no mapping were defined for year, q_1 would be reformulated as

$$\begin{aligned} q_1'' = \langle &\mathsf{ADMISSIONS}, \\ &\{\texttt{regionOf}(\mathsf{patientCity})\}, (\texttt{completeGender}(\mathsf{patientGender}) = \text{'Female'}), \\ &\mathsf{totStayCost} + \mathsf{totExamCost}, \langle\langle \mathsf{totStayCost}, \texttt{sum}\rangle, \langle \mathsf{totExamCost}, \texttt{sum}\rangle\rangle\rangle \end{aligned}$$

which returns data with a different aggregation level than the one required by q_1.

3. For at least one of the metrics mentioned in q, there is no mapping from $\mathcal{M}_s$. In this case, we believe that no meaningful reformulation can be done.

3.5 A Reformulation Framework

In the BIN architecture, queries are formulated on a peer md-schema and answers can come from any other peer connected to the queried peer through a chain of semantic mappings. The key step to this end is *reformulating* a peer's query over its immediate neighbors, then over their immediate neighbors, and so on. More precisely, reformulation takes as input a BIN query on a *target* md-schema $\mathcal{M}_t$ as well as the mappings Ω between $\mathcal{M}_t$ and the schema of one of its neighbors, the *source* md-schema $\mathcal{M}_s$, to output a BIN query that refers only to $\mathcal{M}_s$.[2]

Our approach takes advantage of the well-established research results in the OLTP context, with specific reference to distributed semantic data sharing systems [Halevy et al., 2005]. The reformulation framework we propose is based on a relational setting, as depicted in Figure 3.4, where md-schemata, BIN queries, and semantic mappings at the OLAP level are translated to the relational model. As to md-schemata, without loss of generality we assume that they are stored at the relational level as star schemata. As to queries, a classic logic-based syntax is adopted to express them at the relational level. As to mappings, their representation at the relational level uses a logical formalism typically adopted for schema mapping languages, i.e., *source-to-target tuple generating dependencies* (s-t tgd's) [ten Cate and Kolaitis, 2010]. A BIN query is then reformulated starting from its relational form on a star schema, using the mappings expressed as s-t tgd's.

[2]Hereinafter, we will safely assume that Ω is consistent. Consistency checking is part of mapping management, that is out of the scope of the current work. Interested readers can refer to [Kementsietsidis et al., 2003] for an in-depth discussion on this topic.

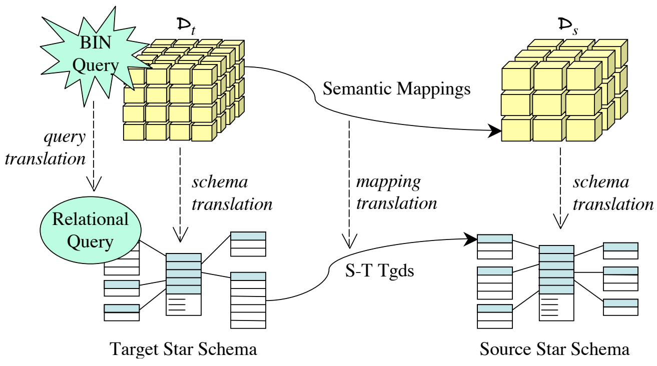


FIGURE 3.4: Reformulation framework

To simplify the query reformulation task, we translate mappings following an approach founded on the semantics of the transformations data are subjected to along the reformulation process. In this way, we are not tied to the syntax of the mapping language presented in Section 3.4, which enables users to express their specification needs through powerful predicates. In particular, the translation of a mapping depends on the mapping accuracy as follows:

- Exact mappings are translated into s-t tgd's that reconcile source data values to target data values according to either the expression or the transcoding specified.
- Loose but not approximate mappings are translated into s-t tgd's that relate source data values with target data values using a transcoding. Though this transcoding is useful for the reformulation of selection predicates expressed at the target peer, it cannot be used for data reconciliation since it is only applicable to transform target values into source values.
- Approximate mappings do not describe how source values should be transcoded to be compatible with the target domains. Thus, the corresponding s-t tgd's can only specify syntactic constraints that define how attributes in the source are related with attributes in the target.

3.5.1 Translating schemata

Let $\mathcal{M} = \langle A, H, M \rangle$ be an md-schema. We assume that $\mathcal{M}$ is stored as a standard star schema: one dimension table $dt_i(\underline{DIM_i}, a_{i_1}, \ldots, a_{i_v})$ for each hierarchy h_i, where $\{DIM_i, a_{i_1}, \ldots, a_{i_v}\} = Attr(h_i)$, and a fact table $ft(\underline{DIM_1}, \ldots, \underline{DIM_n}, m_1, \ldots, m_l)$ where each DIM_i is a foreign key, thus enforcing inclusion dependencies between the fact table and the dimension tables. For example, the star schema corresponding to the HOSPITALIZATION and ADMISSION md-schemata are shown in Figure 3.5.

It is worth noting that, while at the OLAP level both the mapping and query languages directly use the attributes and measures *names*, their counterpart borrowed from the

HospFT($\underline{\text{organ}}$,$\underline{\text{disease}}$,$\underline{\text{date}}$,$\underline{\text{ward}}$,$\underline{\text{patient}}$,cost,durationOfStay)
OrganDT($\underline{\text{organ}}$)
DiseaseDT($\underline{\text{disease}}$)
DateDT($\underline{\text{date}}$,week,month,year)
WardDT($\underline{\text{ward}}$,LHD)
PatientDT($\underline{\text{patient}}$,birthDate,city,region,segment,gender)

AdmFT($\underline{\text{diagnosis}}$,$\underline{\text{date}}$,$\underline{\text{ward}}$,$\underline{\text{patientCity}}$,$\underline{\text{patientBirthYear}}$,$\underline{\text{patientGender}}$,
totStayCost,totExamCost,totLength,maxLength,numAdmissions)
DiagnosisDT($\underline{\text{diagnosis}}$,category)
DateDT($\underline{\text{date}}$,month,year)
WardDT($\underline{\text{ward}}$,unit)
PatientCityDT($\underline{\text{patientCity}}$,patientNation)
PatientBirthYearDT($\underline{\text{patientBirthYear}}$)
PatientGenderDT($\underline{\text{patientGender}}$)

FIGURE 3.5: Star schemata for the Rome (top) and Florence (bottom) peers

relational model use star schema tables under the *unnamed* perspective, i.e., the specific attribute names are ignored, and only the number of attributes of each relation schema is available. These languages, both stemming from mathematical logic, view a database schema as a tuple $\mathbf{R} = (r_1, \ldots, r_n)$ of relation symbols, each of which has a fixed arity. Given an md-schema $\mathcal{M}$ and the corresponding star schema, we switch from the named perspective of the OLAP level to the unnamed one of the relational level through (a) n dimension table-encoding functions $\delta_i : Attr(h_i) \to \mathbb{N}$, each associating every attribute $a \in Attr(h_i)$ with the corresponding position in dt_i, and (b) a fact table-encoding function $\varphi : \{DIM_1, \ldots, DIM_n\} \cup M \to \mathbb{N}$ that associates each measure and dimension with the corresponding position in ft.

3.5.2 Translating queries

In a classical logic-based syntax, a PSJ query on a database schema $\mathbf{R}$ is expressed as a conjunction of relational atoms and Boolean predicates having the following rule-based form:

$$q(\overline{z}) \leftarrow r_1(\overline{x_1}), \ldots, r_n(\overline{x_n}), p_1(\overline{y_1}), \ldots, p_u(\overline{y_u})$$

where each r_i is a relation of $\mathbf{R}$, each $\overline{x_i}$ is a tuple of variables and/or constants of the same arity of r_i, each p_j is an atomic Boolean predicate involving variables in X (where X is the union of the variables appearing in the $\overline{x_i}$'s), and all the variables of tuple $\overline{z}$ belong to X. Given tuple $\overline{x}$, in the following we will use dot notation $\overline{x}.k$ to refer to the k-th component of $\overline{x}$.

A GPSJ query is then represented as a conjunctive query with one or more aggregate terms in its head:

$$q(\overline{z}, \alpha_1(\overline{w_1}), \ldots, \alpha_v(\overline{w_v})) \leftarrow r_1(\overline{x_1}), \ldots, r_n(\overline{x_n}), p_1(\overline{y_1}), \ldots, p_u(\overline{y_u})$$

where each α_i is an aggregation operator, no variable in $\overline{z}$ occurs in $\overline{w_1}$,..., $\overline{w_v}$, and all variables in $\overline{z}$ and $\overline{w_1}$,..., $\overline{w_v}$ belong to X. The answer to a GPSJ query q on a database instance I, $q(I)$, is a relation obtained in three steps: (1) project the tuples of I satisfying the body (i.e., the right-hand part) of q on $\overline{z}, \overline{w_1}, \ldots, \overline{w_v}$; (2) group the tuples that agree on $\overline{z}$; and (3) aggregate the values assigned to $\overline{w_1}, \ldots, \overline{w_v}$ within each group. Interested readers can refer to [Cohen et al., 2006] for a formal definition.

Now, let $q = \langle \mathcal{M}, E, p, expr, T\rangle$ be a BIN query, where $E = \{e_1, \ldots, e_g\}$, $p = p_1 \wedge \ldots \wedge p_u$, and $T = \{\langle m_1, \alpha_1\rangle, \ldots, \langle m_v, \alpha_v\rangle\}$ and let $h_1, \ldots, h_c$ be the hierarchies involved in E and p. The translation of q to the relational level relies on a variable-assignment function ν that associates each measure m in $expr$ with a free variable $\nu(m)$ as well as each attribute a in E and in p with a free variable $\nu(a)$. The join between the involved dimension tables and the fact table requires the introduction of one more free variable $\nu(DIM_i)$ for the dimension DIM_i of each involved hierarchy h_i when DIM_i is not explicitly involved in q.

The translation of q is then a GPSJ query based on the star join between the fact table ft and the dimension tables $dt_1, \ldots, dt_c$ of the hierarchies $h_1, \ldots, h_c$. The variable-assignment function ν is used in the free tuples of the atoms $ft(\overline{x})$, $dt_1(\overline{x_1}), \ldots, dt_c(\overline{x_c})$ to introduce the variables associated with the measures and the attributes involved in q. Their positions in $ft(\overline{x})$ are determined by φ, while the δ_i's are used to place the variables associated with the attributes of q in $dt_1(\overline{x_1}), \ldots, dt_c(\overline{x_c})$. Formally:

$$q(\nu(e_1), \ldots, \nu(e_g), expr(\alpha_1(\nu(m_1)), \ldots, \alpha_v(\nu(m_v)))) \\ \leftarrow ft(\overline{x}), dt_1(\overline{x_1}), \ldots, dt_c(\overline{x_c}), \nu(p_1), \ldots, \nu(p_u)$$

where

- $\nu(e_i)$ denotes the substitution of each attribute a in e_i with the corresponding variable $\nu(a)$;
- the tuple $\overline{x}$ is such that $\overline{x}.\varphi(DIM_1) = \nu(DIM_1), \ldots, \overline{x}.\varphi(DIM_c) = \nu(DIM_c)$ and $\overline{x}.\varphi(m_1) = \nu(m_1), \ldots, \overline{x}.\varphi(m_v) = \nu(m_v)$. The other variables in $\overline{x}$ are anonymous (and denoted with the symbol _);
- each tuple $\overline{x_i}$ is such that $\overline{x_i}.\delta_i(a) = \nu(a)$ for each attribute a involved from the i-th hierarchy and $\overline{x_i}.\delta_i(DIM_i) = \nu(DIM_i)$. The other variables in $\overline{x_i}$ are anonymous;

- $\nu(p_i)$ denotes the substitution of each attribute a in p_i with the corresponding variable $\nu(a)$.

Example 3.5. *The query q_1 shown in Example 3.3 translates onto the* HOSPITALIZATION *star schema to*

$$\begin{aligned} q_1(R, Y, \texttt{sum}(C)) \leftarrow & \mathsf{HospFT}(\_, \_, D, \_, P, C, \_), \\ & \mathsf{DateDT}(D, \_, \_, Y), \\ & \mathsf{PatientDT}(P, \_, \_, R, \_, G), G = \textit{'Female'} \end{aligned}$$

while q_2 translates onto the ADMISSIONS *star schema to:*

$$\begin{aligned} q_2(Y, \texttt{regionOf}(P), \texttt{avg}(T)) \leftarrow & \mathsf{AdmFT}(\_, D, \_, P, \_, \_, \_, \_, T, \_, \_), \\ & \mathsf{DateDT}(D, \_, Y), \\ & \mathsf{PatientCityDT}(P, \_) \end{aligned}$$

3.5.3 Translating mappings

We represent mappings as s-t tgd's [ten Cate and Kolaitis, 2010]. Given a source star schema $\mathbf{S}$ and a target star schema $\mathbf{T}$, an s-t tgd has the form

$$\forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y}\ \psi(\overline{x}, \overline{y}))$$

where $\phi(\overline{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ and $\psi(\overline{x}, \overline{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. Every s-t tgd expresses the containment of one conjunctive query $\phi(\overline{x})$ in another conjunctive query $\psi(\overline{x}, \overline{y})$; informally, it asserts that if a pattern of facts appears in the source, then another pattern of facts must appear in the target. In spite of their syntactic simplicity, s-t tgd's can express many data interoperability tasks arising in applications. They are also known as *global-and-local-as-view* (GLAV) dependencies, and can accommodate both GAV and LAV formalisms: in a GAV, the right-hand side of the implication consists of a single atomic formula

$$\forall \overline{x}(\phi(\overline{x}) \rightarrow U(\overline{x'}))$$

where the variables in $\overline{x'}$ are among those in $\overline{x}$, while in a LAV dependency the left-hand side of the implication consists of an atomic formula

$$\forall \overline{x}(r(\overline{x}) \rightarrow \exists \overline{y}\ \psi(\overline{x}, \overline{y}))$$

A schema mapping is then a triple $\mathcal{M}_\rceil = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ where Σ is a set of s-t tgds. The semantics of $\mathcal{M}_\rceil$ is given in terms of star schema instances: Given an $\mathbf{S}$-instance I and a $\mathbf{T}$-instance J, we say that J is *consistent* with I w.r.t $\mathcal{M}_\rceil$ if (I, J) satisfies every s-t tgd in Σ.

In the following subsections, we present the translation of the mapping predicates proposed in Section 3.4 into s-t tgd's; in case of ambiguity, we will use prefixes **S** and **T** to distinguish source tables from target ones.

3.5.3.1 Exact Mappings

We start from exact mappings, that is, those assertions that either contain expressions or transcoding functions to be used for translating source values to the target domains ($\texttt{same}_{expr}$ [3], $\texttt{equi-level}_f$, and $\texttt{roll-up}_f$). In this case, the proposed s-t tgd's express the constraints between tuples induced either by the expression *expr* that relates measure values in case of `same`, or by the transcoding f which relates attribute values in case of `equi-level` and `roll-up`.

Example 3.6. *With reference to Example 3.4, we report some examples of mapping translations:*

- *The* `same` *mapping* $\omega 1$ *translates to*

$$\forall S, E, C\ (\mathsf{AdmFT}(\_, \ldots, \_, S, E, \_, \_, \_), C = S + E \rightarrow \mathsf{HospFT}(\_, \ldots, \_, C, \_))$$

  This translation disregards the involved aggregation function, `sum`*, that will be dealt with in the query reformulation algorithm.*

- *The* `equi-level` *mapping* $\omega 8$ *translates to*

$$\begin{aligned}\forall D, Y, Y'\ (\mathbf{S}.\mathsf{DateDT}(D, \_, Y), \mathsf{AdmFT}(\_, D, \_, \ldots, \_), Y' = Y \\ \rightarrow \exists D'\ (\mathbf{T}.\mathsf{DateDT}(D', \_, \_, Y'), \mathsf{HospFT}(\_, \_, D', \_, \ldots, \_)))\end{aligned}$$

- *The* `roll-up` *mapping* $\omega 11$ *translates to*

$$\begin{aligned}\forall D, W\ (\mathbf{S}.\mathsf{DateDT}(D, \_, \_), \mathsf{AdmFT}(\_, D, \_, \ldots, \_), W = \texttt{weekOf}(D) \\ \rightarrow \exists D'\ (\mathbf{T}.\mathsf{DateDT}(D', W, \_, \_), \mathsf{HospFT}(\_, \_, D', \_, \ldots, \_)))\end{aligned}$$

Noticeably, two different predicates, `equi-level` and `roll-up`, translate to the same s-t tgd form. Indeed, an `equi-level` predicate states that two projections have the same granularity, whereas a `roll-up` is used to express a one-to-many relationship; this means that, differently from an `equi-level` predicate, a `roll-up` predicate requires to aggregate source data in order to make them compatible with the target domain. On the other hand, the proposed translation focuses on the kind of tuple dependencies each mapping defines, which is solely dependent on the existence of a transcoding that maps

[3]As to the `same` predicate, for presentation simplicity, here we assume that it is annotated with no predicates. The way predicates are dealt with, both in mappings and in queries, will be discussed in Section 3.6.1.

TABLE 3.3: Translation of the exact mapping predicates; superscripts t and s denote elements of the target and source schemata, respectively

Predicate	**Translation**
$\langle m^t, \alpha\rangle$ $\texttt{same}_{expr}$ $\{m^s_1, \dots, m^s_k\}$	$\forall \nu(m^s_1), \dots, \nu(m^s_k), \nu(m^t)$ $(\mathbf{S}.ft(\overline{y^s}), \nu(m^t) = expr(\nu(m^s_1), \dots, \nu(m^s_k)) \rightarrow \mathbf{T}.ft(\overline{y^t}))$
$\{a^t_1, \dots, a^t_j\}\texttt{equi-level}_f\{a^s_1, \dots, a^s_k\}$ $\{a^t_1, \dots, a^t_j\}\texttt{roll-up}_f\{a^s_1, \dots, a^s_k\}$	$\forall \nu(a^t_1), \dots, \nu(a^t_j), \nu(a^s_1), \dots, \nu(a^s_k), \nu(DIM^s_{l'+1}), \dots, \nu(DIM^s_k)$ $(\mathbf{S}.dt^s_1(\overline{x^s_1}), \dots, \mathbf{S}.dt^s_k(\overline{x^s_k}), \mathbf{S}.ft(\overline{x^s}),$ $\nu(a^t_1) = f(\nu(a^s_1), \dots, \nu(a^s_k)).1, \dots, \nu(a^t_j) = f(\nu(a^s_1), \dots, \nu(a^s_k)).j$ $\rightarrow \exists \nu(DIM^t_{l+1}), \dots, \nu(DIM^t_j)(\mathbf{T}.dt^t_1(\overline{x^t_1}), \dots, \mathbf{T}.dt^t_j(\overline{x^t_j}), \mathbf{T}.ft(\overline{x^t})))$

each source value into exactly one target value. The specific properties of transcodings (`equi-level` transcodings are injective, `roll-up` transcodings are not) have an impact on query reformulation but they do not affect the translation of mappings. For instance, consider mapping $\omega 12$, which translates to

$$\forall D, D', O\ (\mathsf{DiagnosisDT}(D, \_), \mathsf{AdmFT}(D, \_, \dots, \_),$$
$$D' = \texttt{substring}(D, 1, 40), O = \texttt{substring}(D, 41, 80)$$
$$\rightarrow \mathsf{DiseaseDT}(D'), \mathsf{OrganDT}(O), \mathsf{HospFT}(O, D', \_, \dots, \_))$$

When a query requires only $\mathsf{disease}$, the diagnosis values obtained from the source peer must be grouped by their disease values, because f projected on the first co-domain is no longer injective. This example also shows that, when a mapping involves no target attributes other than dimensions, the existential quantifier is not used because the join between the fact table and dimension tables is already specified by the attributes involved.

The formal translation of this first class of predicates is shown in Table 3.3. Without loss of generality, we assume that the first l attributes of $\{a^t_1, \dots, a^t_j\}$ and l' attributes of $\{a^s_1, \dots, a^s_k\}$ are dimensions, and that dt_i is the dimension table a_i belongs to. Tuples are then defined as follows (all the unspecified variables are anonymous, and we recall that ν is a variable assignment function that associates a free variable to each attribute and measure involved in the mapping assertions):

- the (fact table) tuple $\overline{y^s}$ is such that $\overline{y^s}.\varphi^s(m^s_i) = \nu(m^s_i)$ for $i = 1, \dots, k$, i.e., the only assigned variables are those corresponding to measures;
- the (fact table) tuple $\overline{y^t}$ is such that $\overline{y^t}.\varphi^t(m^t) = \nu(m^t)$;
- the (fact table) tuple $\overline{x^s}$ is such that $\overline{x^s}.\varphi^s(a^s_i) = \nu(a^s_i)$ for $i = 1, \dots, l'$ and $\overline{x^s}.\varphi^s(DIM^s_i) = \nu(DIM^s_i)$ for $i = l' + 1, \dots, k$, i.e., the only assigned variables are those corresponding to attributes;
- the (fact table) tuple $\overline{x^t}$ is such that $\overline{x^t}.\varphi^t(a^t_i) = \nu(a^t_i)$ for $i = 1, \dots, l$ and $\overline{x^t}.\varphi^t(DIM^t_i) = \nu(DIM^t_i)$ for $i = l + 1, \dots, j$;
- the (dimension table) tuple $\overline{x^s_i}$ is such that $\overline{x^s_i}.\delta^s_i(a^s_i) = \nu(a^s_i)$. Moreover, $\overline{x^s_i}.\delta^s_i(DIM^s_i) = \nu(DIM^s_i)$ for $i = l' + 1, \dots, k$;

TABLE 3.4: Translation of the loose/approximate mapping predicates

Predicate	**Translation**
$\{a^t_1,\ldots,a^t_j\}\texttt{drill-down}_f\{a^s_1,\ldots,a^s_k\}$	$\forall\nu(a^t_1),\ldots,\nu(a^t_j),\nu(a^s_1),\ldots,\nu(a^s_k),\nu(DIM^s_{l'+1}),\ldots,\nu(DIM^s_k)$ $(\mathbf{S}.dt^s_1(\overline{x^s_1}),\ldots,\mathbf{S}.dt^s_k(\overline{x^s_k}),\mathbf{S}.ft(\overline{x^s}),$ $\nu(a^s_1)=f(\nu(a^t_1),\ldots,\nu(a^t_j)).1,\ldots,\nu(a^s_k)=f(\nu(a^t_1),\ldots,\nu(a^t_j)).k$ $\rightarrow\exists\nu(DIM^t_{l+1}),\ldots,\nu(DIM^t_j)(\mathbf{T}.dt^t_1(\overline{x^t_1}),\ldots,\mathbf{T}.dt^t_j(\overline{x^t_j}),\mathbf{T}.ft(\overline{x^t})))$
$\{a^t_1,\ldots,a^t_j\}\texttt{equi-level}\{a^s_1,\ldots,a^s_k\}$ $\{a^t_1,\ldots,a^t_j\}\texttt{roll-up}\{a^s_1,\ldots,a^s_k\}$ $\{a^t_1,\ldots,a^t_j\}\texttt{drill-down}\{a^s_1,\ldots,a^s_k\}$ $\{a^t_1,\ldots,a^t_j\}\texttt{related}\{a^s_1,\ldots,a^s_k\}$	$\forall\nu(a^s_1),\ldots,\nu(a^s_k),\nu(DIM^s_{l'+1}),\ldots,\nu(DIM^s_k)$ $(\mathbf{S}.dt^s_1(\overline{x^s_1}),\ldots,\mathbf{S}.dt^s_k(\overline{x^s_k}),\mathbf{S}.ft(\overline{x^s})$ $\rightarrow\exists\nu(a^t_1),\ldots,\nu(a^t_j),\nu(DIM^t_{l+1}),\ldots\nu(DIM^t_j)$ $(\mathbf{T}.dt^t_1(\overline{x^t_1}),\ldots,\mathbf{T}.dt^t_j(\overline{x^t_j}),\mathbf{T}.ft(\overline{x^t})))$

- the (dimension table) tuple $\overline{x^t_i}$ is such that $\overline{x^t_i}.\delta^t_i(a^t_i)=\nu(a^t_i)$. Moreover, $\overline{x^t_i}.\delta^t_i(DIM^t_i)=\nu(DIM^t_i)$ for $i=l+1,\ldots,j$.

Note that each dimension table dt_i is related to its fact table ft through dimension DIM_i to enforce the inclusion dependency between the two tables.

3.5.3.2 Loose/Approximate Mappings

For a loose but not approximate `drill-down` mapping, a transcoding function f exists but, unlike the case of exact mappings, it cannot be used to translate source values to the target domains (but rather to translate target values to the source domains). The translation of $\texttt{drill-down}_f$ is shown in the upper part of Table 3.4.

The lower part of Table 3.4 shows the translation of approximate mappings. Importantly, in this case there is no dependency between the source and the target tuples. Since the resulting assertions do not express any relationship between the right-hand and the left-hand variables, using s-t tgd's forces a little abuse of notation.

Example 3.7. *The non-approximate* `drill-down` *mapping* $\omega 17$ *translates to*

$$\forall D,Y\ (\mathsf{PatientBirthYearDT}(Y),\mathsf{AdmFT}(\_,\_,\_,\_,Y,\_,\ldots,\_),Y=\texttt{yearOf}(D)$$
$$\rightarrow\exists P(\mathsf{PatientDT}(P,D,\_,\ldots,\_),\mathsf{HospFT}(\_,\_,\_,\_,P,\_,\ldots,\_)))$$

while the approximate `drill-down` *mapping* $\omega 14$ *translates to*

$$\forall C,G,B\ (\mathsf{PatientCityDT}(C,\_),\mathsf{PatientGenderDT}(G),$$
$$\mathsf{PatientBirthYearDT}(B),\mathsf{AdmFT}(\_,\_,\_,C,B,G,\_,\ldots,\_)$$
$$\rightarrow\exists P(\mathsf{PatientDT}(P,\_,\ldots,\_),\mathsf{HospFT}(\_,\_,\_,\_,P,\_,\ldots,\_)))$$

3.6 Query Reformulation in a BIN

Reformulation is the key step for query answering in a BIN as it actually represents a means to mediate queries among heterogeneous md-schemata. Two important issues arise:

- The inter-peer reformulation algorithm takes as input a *target* query and produces a *source* query. The fact that the BIN query language introduced in Definition 3.3 is closed under reformulation is an essential property which ensures that chains of reformulations can take place in a BIN. On the other hand, multidimensional data are accessible through multidimensional engines using query languages whose expressive power is typically different from ours (for instance, MDX does not allow a group-by set to include a transcoding). This means that, to access its local data, each queried peer may have to carry out an *intra-peer reformulation* that translates a source BIN query into a *local query.* A local query relies on a view V corresponding to a query that can be directly executed on the local engine, and contains a declarative description of how to derive the data required by the BIN query from V.

- As pointed out in Section 3.4, the results returned by a peer may match with the original query with some approximation. We recall that a non-compatible reformulation occurs every time the available mappings do not allow the values of (some of) the target attributes to be derived from the source ones. In this case, the source query does not fully preserve the semantics of the target query as it refers to attributes which are merely syntactically related with the target ones, so the returned values cannot be integrated in the target attribute columns. We believe that any information about the reasons of a non-compatible reformulation could considerably help users understand the returned results. To this end, when a query is formulated on a peer, the answers this peer receives from any other peer should at least be equipped with the knowledge about which attributes found an exact mapping and which ones did not.

We are now ready to define the query reformulation problem in a BIN.

Definition 3.5 (Query reformulation problem)**.** Given a BIN query q_t on (target) md-schema $\mathcal{M}_t$, a (source) md-schema $\mathcal{M}_s$, and a set of mappings Ω from $\mathcal{M}_s$ to $\mathcal{M}_t$:

1. An *inter-peer reformulation* of q_t on $\mathcal{M}_s$ is a BIN query q_s that refers only to $\mathcal{M}_s$, together with some information on the mapped attributes.

2. An *intra-peer reformulation* of q_t or q_s is a relational query whose body contains a view V directly computable by the local multidimensional engine.

In the remainder of this section we focus on inter-peer reformulation, while intra-peer reformulation will be discussed in Section 3.7 with specific reference to the MDX query language.

3.6.1 The Inter-Peer Reformulation Algorithm

This section presents an algorithm that, consistently with the framework proposed in Section 3.5, reformulates a target BIN query q_t onto a source peer at the relational level in three steps:

1. The mappings to be used for reformulation are selected from Ω. In presence of predicates annotating `same` mappings, q_t is expanded to include those predicates.
2. The relational translation of q_t, q, is reformulated into a relational query q' that only refers to the source star schema $\mathbf{S}$.
3. q' is expanded to include all the constraints stated by the transcodings used, then translated into a (source) BIN query q_s.

In the following we show each single step, using as an example the query asking for the average cost of female patients for each disease and segment:

$$\begin{aligned} q_t = \langle & \mathsf{HOSPITALIZATION}, \\ & \{\mathsf{disease}, \mathsf{segment}\}, (\mathsf{gender} = \text{'Female'}), \\ & \mathsf{cost}, \langle\langle \mathsf{cost}, \mathsf{avg}\rangle\rangle\rangle \end{aligned}$$

Finally, we discuss how to deal with selection predicates.

3.6.1.1 Step 1: Mapping Selection

In this step we perform a selection of the mappings in Ω that are relevant for reformulating q_t. To this end, we follow a syntactic approach that relies on the distinction between measure mappings ($\Omega^{meas} \subseteq \Omega$) and attribute mappings ($\Omega^{attr} \subseteq \Omega$). Depending on the mapping predicates involved, attribute mappings are further distinguished into three types: $Type(\omega) \in \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = \{\texttt{equi-level}_f, \texttt{roll-up}_f\}$ (exact mappings), $\tau_2 = \{\texttt{drill-down}_f\}$ (loose and non-approximate mappings), and $\tau_3 = \{\texttt{equi-level}, \texttt{roll-up}, \texttt{drill-down}, \texttt{related-to}\}$ (approximate mappings).

The selection procedure is shown in Algorithm 1. It selects and returns a set Ω_q of mappings based on two policies:

- As to metrics (Lines 2-16), we require that a source query can correctly compute the aggregate values specified in the target query with reference to its md-schema

Algorithm 1 Mapping Selection

```
Require: $\Omega = \Omega^{meas} \cup \Omega^{attr}$: mappings, $q_t = \langle \mathcal{M}_t, E, p, expr, T \rangle$: target query
 1: $\Omega_q = \emptyset$
 2: for all $\mu = \langle m, \alpha \rangle \in T$ do
 3:   for all $\omega \in \Omega^{meas}, \omega : \mu_t$ $\texttt{same}_{expr,pred} N_s$ do
 4:     if $\mu_t = \mu$ then
 5:       add $\omega$ to $\Omega_q$
 6:       $p = p \wedge pred$
 7:       if $p$ is not satisfiable then
 8:         Abort reformulation
 9:       end if
10:       Go to Line 2
11:     end if
12:   end for
13:   if $\mu$ did not find a mapping then
14:     Abort reformulation
15:   end if
16: end for
17: for all attributes $a$ appearing in $E$ do
18:   for all $\omega \in \Omega^{attr}, \omega : P_t < \texttt{mappPred} > P_s$ do
19:     if $a \in P_t$ then
20:       add $\omega$ to $\Omega_q$
21:     end if
22:   end for
23: end for
24: for all attributes $a$ appearing in $p$ but not in $E$ do
25:   for all $\omega \in \Omega^{attr}$ such that $Type(\omega) = T1$ or $Type(\omega) = T2$ do
26:     if $a \in P_t$ then
27:       add $\omega$ to $\Omega_q$
28:     end if
29:   end for
30: end for
31: return $\Omega_q$, $q_t$;
```

$\mathcal{E}_s$; therefore, the existence of a mapping for each metric $\mu \in T$ in q_t is mandatory (Line 14). The conjunctive Boolean predicate that may annotate a same mapping is added to the query selection predicate p (Line 6); this modification could make p unsatisfiable, in which case reformulation is aborted as in [Halevy et al., 2005].

- As to attributes, we distinguish those in the generalized group-by set E (Lines 17-23) from those in the selection predicate p (Lines 24-30). This distinction is necessary because in the first case we match the attributes against all the mappings in Ω whereas, in the second case, we focus our search only on those mappings that contain a transcoding (because, in the absence of a transcoding, translating a selection predicate is not possible).

Example 3.8. *The measure mapping to be used for translating q_t is ω_2, so initially $\Omega_q = \{\omega_2\}$. Since ω_2 is annotated with one predicate, at Line 6 q_t becomes:*

$$\begin{aligned} q_t = \langle & \mathsf{HOSPITALIZATION}, \\ & \{\mathsf{disease}, \mathsf{segment}\}, (\mathsf{gender} = \text{'}Female\text{'}) \wedge (\mathsf{segment}\ \mathtt{in}\ \{\text{'}NH\text{'}, \text{'}EU\text{'}\}), \\ & \mathsf{cost}, \langle\langle \mathsf{cost}, \mathsf{avg}\rangle\rangle\rangle \end{aligned}$$

Then the other two steps add mappings ω_{12}, ω_{13}, ω_{15}, and ω_{16} to Ω_q. Note that, if the predicate annotating ω_2 were gender=*'Male', reformulation would be aborted.*

3.6.1.2 Step 2: Query Reformulation

This step takes place at the relational level, and it reformulates the relational translation of q_t, q, using Σ_q, i.e., the set of s-t tgd's that translate the mappings in Ω_q onto the relational level. To this end, each s-t tgd $\sigma \in \Sigma_q$ is matched with the body of q.

As already mentioned, the s-t tgd's translating approximate mappings do not state any dependency between the left-hand variable and the right-hand one; in other words, none of the right-hand variables is used in the left-hand of such s-t tgd's. In these cases, we keep track of the syntactic relationships between the target variables and the source ones as derived from the mappings by defining a partial variable-set mapping function $Approx : 2^{Var(q)} \rightarrow 2^{Var(q')}$ that relates sets of head variables in q with sets of variables in q'. This function will be used to select the head variables of the source query q' and will annotate the results returned to the querying peer.

This step includes three phases: first we define the body of q', then we build *Approx*, and finally we define the head of q'. For simplicity, we first consider the case in which neither the same mappings nor q include Boolean predicates.

The first phase is described by Algorithm 2, where each s-t tgd having the form $\sigma : \forall\overline{x}(\phi(\overline{x}) \rightarrow \exists\overline{y}\,\psi(\overline{x},\overline{y}))$ is turned into a pair of tgd's $\sigma^{LAV} : \forall\overline{x'}(V(\overline{x'}) \rightarrow \exists\overline{y} : \psi(\overline{x'},\overline{y}))$ and $\sigma^{GAV} : \forall\overline{x}(\phi(\overline{x}) \rightarrow V(\overline{x'}))$, where the former is expressed in LAV style, the latter in GAV style, and $\overline{x'}$ is the tuple of variables shared by the two sides of σ. Essentially, the algorithm builds the body of q', *body*, by merging the left sides of the s-t tgd's in Σ_q, and returns the set of views used during this process. More specifically, for each selected s-t tgd σ, $\mathtt{unify}(\sigma^{LAV}, q)$ (Line 5) matches the atomic formulas in the body of σ^{LAV}, f_σ, to that of q, f_q, by finding a variable mapping η such that $\eta(f_\sigma) = \eta(f_q)$, if possible. The result of unifying σ^{LAV} with q, η, is then applied to the σ^{LAV} counterpart, σ^{GAV} (Line 6). Noticeably, these steps are performed only if the head of σ^{LAV} and that of σ^{GAV} share some variables, because only in this case unification affects σ^{GAV}. Moreover, some of the distinguished variables in $\overline{x'}$ may become anonymous in $\eta(\sigma^{GAV})$. Finally, the body of σ^{GAV} is merged with *body*.

Algorithm 2 *body* Setup

```
Require: $\Sigma_q$: relevant s-t tgd's
 1: $body = \epsilon$
 2: $views = \{\}$
 3: for all $\sigma \in \Sigma_q$ do
 4:   if $\overline{x'} \neq \epsilon$ then
 5:     $\eta = \texttt{unify}(\sigma^{LAV}, q)$
 6:     $\sigma_{GAV} = \eta(\sigma^{GAV})$
 7:     $views = views \cup \eta(V(\overline{x'}))$
 8:   end if
 9:   $\texttt{merge}(body, \sigma^{GAV})$
10: end for
11: return $body$, $views$
```

Example 3.9. *The query q shown in Example 3.8 translates at the relational level as follows:*

$$\begin{aligned} q(D, S, \texttt{avg}(C)) \leftarrow & \mathsf{HospFT}(\_, D, \_, \_, P, C, \_), \\ & \mathsf{DiseaseDT}(D), \\ & \mathsf{PatientDT}(P, \_, \_, \_, S, G), G = \textit{'Female'}, S \texttt{ in } \{\textit{'NH'}, \textit{'EU'}\} \end{aligned}$$

The set Σ_q corresponding to Ω_q includes the st-tgd's listed below:

$$\sigma_2 : \forall S, E, C(\mathsf{AdmFT}(\_, \ldots, \_, S, E, \_, \_, \_), C = S + E \rightarrow \mathsf{HospFT}(\_, \ldots, \_, C, \_))$$

$$\begin{aligned} \sigma_{12} : \forall D, D', O(& \mathsf{DiagnosisDT}(D, \_), \mathsf{AdmFT}(D, \_, \ldots, \_), \\ & D' = \mathsf{substring}(D, 1, 40), O = \mathsf{substring}(D, 41, 80) \\ & \rightarrow \mathsf{DiseaseDT}(D'), \mathsf{OrganDT}(O), \mathsf{HospFT}(O, D', \_, \ldots, \_)) \end{aligned}$$

$$\begin{aligned} \sigma_{13} : \forall D, D', C(& \mathsf{DiagnosisDT}(D, C), \mathsf{AdmFT}(D, \_, \ldots, \_), C = \mathsf{categoryOf}(D') \\ & \rightarrow \mathsf{DiseaseDT}(D'), \mathsf{HospFT}(\_, D', \_, \ldots, \_)) \end{aligned}$$

$$\begin{aligned} \sigma_{15} : \forall G, P(& \mathsf{PatientGenderDT}(P), \mathsf{AdmFT}(\_, \_, \_, \_, \_, P, \_, \ldots, \_), G = \mathsf{completeGender}(P) \\ & \rightarrow \exists P'(\mathsf{PatientDT}(P', \_, \ldots, \_, G), \mathsf{HospFT}(\_, \_, \_, \_, P', \_, \_))) \end{aligned}$$

$$\begin{aligned} \sigma_{16} : \forall G, C, Y(& \mathsf{PatientCityDT}(C, \_), \mathsf{PatientBirthYearDT}(Y), \\ & \mathsf{PatientGenderDT}(G), \mathsf{AdmFT}(\_, \_, \_, C, Y, G, \_, \ldots, \_) \\ & \rightarrow \exists P, S(\mathsf{PatientDT}(P, \_, \ldots, \_, S, \_), \mathsf{HospFT}(\_, \_, \_, \_, P, \_, \_))) \end{aligned}$$

The output of the body setup phase is

$$\begin{aligned}body = (&\mathsf{AdmFT}(D', \_, \_, T, Y, P, S', E, \_, \_, \_), C' = S' + E, \mathsf{DiagnosisDT}(D', C),\\ &D = \mathsf{substring}(D', 1, 40), C = \mathsf{categoryOf}(D), \mathsf{PatientGenderDT}(P),\\ &\mathsf{PatientCityDT}(T, \_), \mathsf{PatientBirthYearDT}(Y), G = \mathsf{completeGender}(P))\end{aligned}$$

Let us focus on the s-t tgd σ_{12}*, that corresponds to*

$$\sigma^{LAV} : \forall O, D'(V_{12}(O, D') \rightarrow \mathsf{DiseaseDT}(D'), \mathsf{OrganDT}(O), \mathsf{HospFT}(O, D', \_, \ldots, \_))$$

$$\begin{aligned}\sigma^{GAV} : \forall D, D', O(&\mathsf{DiagnosisDT}(D, \_), \mathsf{AdmFT}(D, \_, \ldots, \_),\\ &D' = \mathtt{substring}(D, 1, 40), O = \mathtt{substring}(D, 41, 80) \rightarrow V_{12}(O, D'))\end{aligned}$$

The result of unification, $\mathtt{unify}(\sigma^{LAV}, q)$*, is a variable mapping* η *that maps* O *into an anonymous variable and* $\eta(V_{12}(O, D')) = V_{12}(\_, D)$*. Therefore, the predicate* $\_ = \mathtt{substring}(D, 41, 80)$ *is not added to body. Moreover, note that none of the returned views,* $V_2(C)$*,* $V_{12}(\_, D)$*,* $V_{13}(D)$*, and* $V_{15}(G)$*, contains the query head variable* S *as the mapping of* segment *has type* τ_3 *and, therefore, the corresponding s-t tgd does not introduce any relationship between the left-hand variables and the right-hand ones.*

The second phase is shown in Algorithm 3, that outputs the partial variable set mapping *Approx* by leveraging on the properties of the selected mappings. If a target attribute is related to the source schema by means of a non-exact mapping, then its values cannot be derived from the values of the related source attributes. For this reason, each time we use a mapping ω of type τ_2 or τ_3, we extend *Approx* with a mapping from the set of variables corresponding to the attributes in ω that have not found an exact mapping yet, i.e., $I = (q \cap P_t) \setminus Exact$, to the set of variables corresponding to P_s (Line 6). On the contrary, if ω is exact we delete from *Approx* all the variables corresponding to the attributes in ω and add them to the attribute set *Exact*, to prevent their inclusion into *Approx* during a later step (Lines 8-10).

Example 3.10. *The partial variable set mapping Approx for the reference example consists of the pair* $\{S\} \mapsto \{T, Y, P\}$*.*

The third phase consists in deriving the head of q', $q'(\overline{z'}, expr'(\alpha'_1(w'_1), \ldots, \alpha'_{v'}(w_{v'}))$, from the head of q, $q(\overline{z}, expr(\alpha_1(w_1), \ldots, \alpha_v(w_v))$:

- As to the head variables in $\overline{z'}$, we first compare the set of variables $\overline{y}$ in *views* with the set of head variables $\overline{z}$ of q. Indeed, the only head variables of q that found a reformulation are those contained in at least the head of one view. Therefore, $\overline{z''} \subseteq \overline{z'}$ where $\overline{z''} = (\overline{y} \cap \overline{x}) \setminus Dom(Approx)$. For any head variable in $\overline{z}$ that is

Algorithm 3 *Approx* Setup

```
Require: $\Omega_q = \Omega_q^{meas} \cup \Omega_q^{attr}$: relevant mappings, $q$: target query, $\nu_q$: variable-assignment function for $q$, $body$: body of $q'$, $\nu_{body}$: variable-assignment function for $body$
1: $Approx = \{\}$
2: $Exact = \{\}$
3: for all $\omega \in \Omega_q^{attr}, \omega : P_t <$ mappPred $> P_s$ do
4:   $I = (q \cap P_t) \setminus Exact$
5:   if $Type(\omega) = \tau_2$ or $Type(\omega) = \tau_3$ then
6:     Add $\nu_q(I) \mapsto \nu_{body}(P_s)$ to $Approx$
7:   else
8:     Add $I$ to $Exact$
9:     Delete from the domain of $Approx$ the variables in $\nu_q(I)$
10:    Delete from $Approx$ each pair $V \mapsto V'$ such that $V = \{\}$
11:  end if
12: end for
13: return $Approx$
```

not contained in $\overline{z''}$, either it did not find any mapping or it found approximate or loose mappings only. For the latter case, we exploit the image of the *Approx* function since it contains the set of q' variables that are syntactically related with the head of q. Such variables are added to $\overline{z'}$ in place of the head variables in $\overline{z}$ that found approximate and loose mappings only. Therefore $\overline{z'} = \overline{z''} \cup Img(Approx)$.

- As to aggregate terms $\alpha_i(w_i)$, each w_i is necessarily contained in the *body* of q' and is involved in a comparison predicate $w_i = expr_i(v_1, \ldots, v_n)$, where $v_1, \ldots, v_n$ are variables of q', that states the relationship between one target measure and the source ones. Therefore, w_i is replaced with $expr_i(v_1, \ldots, v_n)$ in the aggregate term and the comparison predicate is deleted from *body*. Then, the aggregate function α_i is distributed inside $expr_i$, thus obtaining $expr_i(\alpha_i(v_1), \ldots, \alpha_i(v_n))$.

As to the last point note that, obviously, $expr_i(\alpha_i(v_1), \ldots, \alpha_i(v_n))$ is equal to $\alpha_i(expr_i(v_1, \ldots, v_n))$ only if α_i is distributive over $expr_i$. Ensuring that the established mappings are consistent with the semantics of measures and with the operators through which each measure can be aggregated, is the designer's responsibility.

Example 3.11. *The head of our reference query is* $q(D, S, \texttt{avg}(C'))$. *By following the steps described above, we have that* $D \in \{V_{12}, V_{13}\}$ *whereas* $S \notin views$. *Indeed,* S *found an approximate mapping only, so* $(\{S\} \mapsto \{T, Y, P\}) \in Approx$. *Therefore,* D, T, Y, P *are head variables. Moreover,* $\texttt{avg}(C')$ *translates into* $\texttt{avg}(S') + \texttt{avg}(E)$. *Summing up,* *the reformulated query q' is:*

$$
\begin{aligned}
q'(D, T, Y, P, \mathsf{avg}(S') + \mathsf{avg}(E)) \leftarrow \quad & \\
& \mathsf{AdmFT}(D', \_, \_, T, Y, P, S', E, \_, \_, \_), \\
& \mathsf{DiagnosisDT}(D', C), \\
& D = \mathtt{substring}(D', 1, 40), \\
& C = \mathtt{categoryOf}(D), \\
& \mathsf{PatientGenderDT}(P), \\
& \mathsf{PatientCityDT}(T, \_), \\
& \mathsf{PatientBirthYearDT}(Y), \\
& G = \mathtt{completeGender}(P)
\end{aligned}
$$

3.6.1.3 Step 3: Query Expansion

Once q has been reformulated into q' according to the s-t tgd's in Σ_q, it may be necessary to expand q' to include further constraints on the sources variables stated by the s-t tgd's in Σ that contain transcodings. This is done by merging the body of q' with a set Γ of atomic formulas. Since Γ is independent of q', it can be computed off-line as explained below.

Let $\Omega^c \subseteq \Omega$ be the set of mappings of type τ_1 and τ_2, i.e., that express transcoding constraints, and let Σ^c be the corresponding set of s-t tgd's. First, the s-t tgd's in Σ^c are merged to obtain a single s-t tgd, σ^c. Then, the closure of the predicates appearing on the left-hand side of σ^c is computed. Among the predicates in the closure, only those involving distinguished variables belonging only to the left-hand part of σ^c must be kept. To this end, we denote with Δ the set of these variables, and we introduce η as a variable mapping that maps each variable in Δ into itself, and makes all the remaining variables anonymous. The set Γ is finally obtained by applying η to the left-hand side of σ^c.

Example 3.12. *In our reference example, we have*

$$\Gamma = (\mathsf{AdmFT}(D', \_, \ldots, \_), \mathsf{DiagnosisDT}(D', C), C = \mathtt{categoryOf}(\mathtt{substring}(D', 1, 40)))$$

Note that several predicates translating the transcodings in Table 3.1 do not appear here; for instance, the predicate $C = P$ corresponding to ω_{18} is discarded because C and P are not both defined on the left-hand side of σ^c: the former is defined in the relational atom $\mathsf{PatientDT}(\_, \_, C, \_, \ldots)$ *of the right-hand side of σ^c, while the latter is defined in the relational atom* $\mathsf{PatientCityDT}(P, \_)$ *of the left-hand side of σ^c. Obviously, when Γ and q' are merged, predicate $C =$* $\mathtt{categoryOf}(D)$ *is discarded because, if coupled with $D =$* $\mathtt{substring}(D', 1, 40)$*, it is equivalent to $C =$* $\mathtt{categoryOf}(\mathtt{substring}(D', 1, 40))$*.*

On the other hand, $D = \texttt{substring}(D', 1, 40)$ *cannot be discarded because* D *is a head variable.*

3.6.1.4 Incorporating Selection Predicates

Selection predicates provide a very useful mechanism for specifying constraints on attribute values. The BIN framework supports the specification of selection predicates in both queries and same mappings, and Step 1 exploits them to prevent inconsistent reformulations (see Algorithm 1). On the other hand, when Step 1 succeeds, it outputs a BIN query q_t whose predicate p includes both the query predicates and the mapping ones.

At the beginning of Step 2, such predicates are encoded in the relational query q and they must reformulated on the source schema. Nevertheless, some of the atomic predicates in q may not undergo reformulation because of the head variables of q that did not find any exact mapping. More specifically, to decide which predicates of q can be incorporated in q', Step 2 must be modified as follows. At the end of Algorithm 2, for each atomic predicate c in q we check if the set of variables in c is contained in the set of variables in *views*, in which case c is merged with *body*. Generally speaking, the presence of predicates could lead to inconsistent reformulation on the source side too. To this end, each time in Step 2 we include any predicate in *body*, we also check that *body* is still satisfiable.

Example 3.13. *Given the query* q_t *shown in Example 3.8, its relational translation* q *contains two atomic predicates:* $G =$ *'Female' and* S in $\{$*'NH','EU'*$\}$. *Since* $G \in V_{15}$*, the corresponding predicate is added to the body of* q'*, whereas* S *does not belong to views and thus the corresponding predicate is discarded. The final reformulated query* q' *is:*

$$\begin{aligned} q'(D, T, Y, P, \mathsf{avg}(S') + \mathsf{avg}(E)) \leftarrow & \mathsf{AdmFT}(D', \_, \_, T, Y, P, S', E, \_, \_, \_), \\ & \mathsf{DiagnosisDT}(D', C), \\ & D = \texttt{substring}(D', 1, 40), \\ & C = \texttt{categoryOf}(\texttt{substring}(D', 1, 40)), \\ & \mathsf{PatientGenderDT}(P), \mathsf{PatientCityDT}(T, \_), \\ & \mathsf{PatientBirthYearDT}(Y), \\ & \texttt{completeGender}(P) = \textit{'Female'} \end{aligned}$$

3.6.2 Properties of the Inter-Peer Reformulation Algorithm

The query reformulation algorithm shows two properties that are essential for its working in a distributed setting (the proofs are given in A).

First, the algorithm outputs a query q' that finds a corresponding query at the BIN level. For instance, the BIN query corresponding to our reference query q' is

$$
\begin{aligned}
q_s = \langle \mathsf{ADMISSION}, \\
&\{\mathtt{substring}(\mathsf{diagnosis}, 1, 40), \\
&\mathsf{patientCity}, \mathsf{patientBirthYear}, \mathsf{patientGender}\}, \\
&(\mathtt{completeGender}(\mathsf{patientGender}) = \text{'Female'} \wedge \\
&\mathsf{category} = \mathtt{categoryOf}(\mathtt{substring}(\mathsf{diagnosis}, 1, 40))), \\
&\mathsf{totStayCost} + \mathsf{totExamCost}, \langle\langle \mathsf{totStayCost}, \mathsf{avg}\rangle, \langle \mathsf{totExamCost}, \mathsf{avg}\rangle\rangle\rangle
\end{aligned}
$$

In other words, the BIN query language is *closed under reformulation*. The practical impact of this is that our query reformulation algorithm can be used by each peer in a BIN to implement chains of reformulations. In this way, any query formulated over a peer schema can be safely distributed across the network, and answers can come from any other peer in the network which is connected to the queried peer through a chain of semantic mappings.

Theorem 3.6. *Let q_t be a (target) BIN query and q' be the output of the query reformulation algorithm when q_t is given as input. Then, there exists a (source) BIN query q_s such that q' is the relational translation of q_s.*

Secondly, the reformulation algorithm is *sound and complete* with respect to the semantics of query answering, that in data sharing settings is usually given in terms of *certain answers*.

Definition 3.7 (Certain Answers)**.** Let $\mathcal{M}_\rceil = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and q be a query over the target star schema $\mathbf{T}$. If I is a source instance, then the certain answers of q on I with respect to $\mathcal{M}_\rceil$, denoted $certain_{\mathcal{M}_\rceil}(q)(I)$, is the set [ten Cate and Kolaitis, 2010]

$$certain_{\mathcal{M}_\rceil}(q)(I) = \bigcap \{q(J) : J \text{ is a solution for } I \text{ w.r.t. } \mathcal{M}_\rceil\}$$

The computational complexity of finding all certain answers is well understood for the data integration context with a two-tiered architecture of a mediator and a set of data sources [Abiteboul and Duschka, 1998]. The same problem has been deeply investigated also in data exchange settings [Afrati and Kolaitis, 2008]. In both cases, computing the certain answers of unions of conjunctive queries has been proved to be done with polynomial time in the size of the instance I. The number of peers in the net represents just a multiplication factor of the size of the instance I. The same computational results have been proved for conjunctive queries in the context of PDMSs, under specific constraints on mappings [Halevy et al., 2005].

Given a BIN query q_t, the following theorem shows that the algorithm we have proposed for reformulating q_t into q_s is sound and complete, because evaluating q_s always produces all and only certain answers to q_t projected on the compatible part of the reformulation.

Theorem 3.8. *Let q_t be a BIN query, q_{t_1} be the output of Step 1 on q_t, $q(\overline{z}, aggrExpr)$ be the relational translation of q_{t_1}, and $\overline{z''}$ be the head variables of q that find a reformulation. The reformulation algorithm guarantees to find all certain answers of $q(\overline{z''}, aggrExpr)$.*

The proof of the Theorem above (A) states that the main difference between our algorithm and the reference algorithms for query reformulation (e.g., [Halevy et al., 2005]) lies in the selection of mappings. Indeed, differently from the logical approach usually adopted, we implement a policy for mapping selection that is *syntactically* driven (Step 1). The reason why we adopt a different approach is that the BIN framework allows for the specification of approximate mappings (type τ_3), i.e., mappings that induce no dependencies between the source and the target tuples. Such mappings are neglected by traditional algorithms because the heads of their views are empty. The syntactic approach we adopt, instead, allows not only the selection of mappings of type τ_1 and τ_2, as in traditional algorithms, but justly also those of type τ_3 as they state syntactic transformations that will be exploited in the actual reformulation phase. In this way, besides being correct, our algorithm also satisfies all the requirements listed in Section 3.4.

3.7 Implementation

To give a complete picture, here we discuss the main implementation issues raised by reformulation in a BIN, namely: how to bridge the language and expressiveness gap between the query handler and the local multidimensional engine (i.e., how to deal with intra-peer reformulation) and how to manage transcodings and share them among peers.

As to the first problem, we observe that in general a BIN query (either directly formulated by a user or reformulated across the network) cannot be directly executed on the peer local multidimensional engine. The OLAP adapter is in charge of bridging this gap by supporting intra-peer reformulation of BIN queries. Assuming that the de-facto standard MDX is the querying language of the local multidimensional engine, intra-peer reformulation must deal with the presence of transcodings in the query group-by set, and must properly manage non-distributive aggregation operators. From the reformulation point of view, this amounts to solving a problem of *query rewriting using views* [Halevy, 2001], where the set of views is made of all the possible queries that the engine supports. In particular, given the relational translation q on a BIN query, we have to find a local query q^l that refers to one view and is equivalent to q. Differently from classical approaches, in our case the required view V is constructed on-the-fly based on the body of q and on the properties of the involved aggregation operators. To this end we recall that, while for a distributive aggregation operator (like `sum`) an aggregation can be correctly computed from pre-aggregates using one aggregation operator, for an algebraic operator (like `avg`) this requires the computation of pre-aggregates using a tuple of aggregation operators and a function to combine them [Gray et al., 1997] [4]. More precisely, the group-by set of V is the projection obtained by "flattening" the group-by set of q (which means eliminating the transcodings), and the aggregations component of V computes the pre-aggregate values. Besides V, the body of q^l contains the predicates that express transcodings, while its head computes the final aggregates by properly combining the pre-aggregates.

As to the second problem, we saw that the OLAP adapter of a source peer may have to apply transcodings (i.e., functions) to the locally retrieved data, in order to express them in the target peer format and encoding. A trivial solution to make transcodings available to peers consists in assuming the existence of a shared library of transcodings. Unfortunately this solution is hardly feasible because the size of such library would be proportional to the number of mappings, thus decreasing the BIN scalability; furthermore, it does not comply with the security policies of the network since it would entail the existence of a knowledge shared by all the peers. For this reasons, we assume that transcoding functions can be classified into *public* and *protected*. Public transcodings are standard database functions (e.g., `substring` and `dateOf`) that are shared by all peers. Protected transcodings (e.g., `regionOf`) are owned by a peer, that will make them available to its neighboring peers by attaching them to query messages. If protected transcodings are expressed as procedures, a shared programming language must be available in the BIN. Otherwise, transcodings can be expressed as look-up tables to be applied by a relational engine. In this case, an obvious drawback is the quantity of information to be transmitted over the network. To reduce such overhead, look-up tables for protected transcodings of mappings from p_1 to p_2 should be stored in p_2 (i.e., the peer playing the role of the target in query reformulation) so that, when multiple transcodings are nested (e.g. `nationOf`(`regionOf`())) due to chain reformulations, the composition can be solved in p_2 and only the resulting look-up table is sent across the network.

Example 3.14. *The query q_2 shown in Example 3.3 requires the involved measure to be averaged by* year *and* `regionOf(patientCity)`. *Computing the average of pre-aggregates*

[4]Holistic operators, such as mode and median, are not considered here because there is no way to correctly compute aggregates from pre-aggregates using these operators, which makes reformulation impossible in most practical cases.

requires that both a sum and a count are calculated. Considering that measure numAdmission *actually stores a count, view V is:*

$$
\begin{aligned}
V(Y, P, \texttt{sum}(T), \texttt{sum}(A)) \leftarrow & \mathsf{AdmFT}(\_, D, \_, P, \_, \_, \_, \_, T, \_, A), \\
& \mathsf{DateDT}(D, \_, Y), \\
& \mathsf{PatientCityDT}(P, \_)
\end{aligned}
$$

where the transcoding predicate $R = \texttt{regionOf}(P)$ is not included in V because it cannot be directly expressed in MDX. The corresponding local query is:

$$
\begin{aligned}
q^l(Y, R, \texttt{sum}(sT)/\texttt{sum}(sA)) \leftarrow & V(Y, P, sT, sA), \\
& R = \texttt{regionOf}(P)
\end{aligned}
$$

To complete this example we show the MDX query corresponding to view V:

```
SELECT {[Measures].[totLength], [Measures].[numAdmissions]} ON COLUMN,
       {NonEmptyCrossJoin([Date].[year].Members,[Patient].[patientCity].Members)}
ON ROWS
FROM [Admissions]
```

and the SQL query corresponding to q^l:

```
SELECT     RS.year, LT.region, sum(RS.totLength)/sum(RS.numAdmissions) AS avgLength
FROM       ResultSet RS, LookupTable LT
WHERE      RS.patientCity=LT.city
GROUP BY   RS.year, LT.region;
```

where ResultSet *stores the result of the MDX query, whereas* LookupTable *corresponds to the* `regionOf()` *transconding.*

3.8 Conclusions

In this chapter, we introduced BIN as an architecture to support BI FROM-ANYWHERE [Golfarelli et al., 2010, 2011a, 2012a,b]. BIN is the first significant step to extend the P2P paradigm to the BI context, favouring a collaborative experience between companies in a dynamic and flexible network. We addressed the core task in a BIN framework: query reformulation. We introduced a mapping language and we demonstrated the language is closed under reformulation and the correctness of the reformulation algorithm. Nevertheless, different topics deserve further investigation to use BIN in a real context.

Managing the huge quantity of data transmitted and then reconciled is still an open issue. A first solution could be the application of data compression strategies, already investigated in the DW context to answer approximate OLAP query [Yu and Wang, 2002], or to compress text attributes in DW dimensions [Vieira et al., 2005]. An alternative approach could be the application of map-reduce solutions that have gained increasing interests in both OLTP and OLAP contexts, in recent years. To this purpose, the work by [T. et al., 2010] proposes a map-reduce data warehouse solution based on Hadoop (an open-source map-reduce implementation largely used in real contexts [Wiki, 2012]. This approach is not focused on a particular domain and can be further investigated to mine significant hints for the BIN framework.

As to the automatic definition of semantic mappings between schemata, a promising direction could be the adoption of a global ontology. On the one hand, this strategy implies the definition of an overall 'vocabulary' that integrates at a global level the multiple concepts of all peers in the net. This solution may entail a significant effort mainly due to find a common understanding on the domain concepts among all the participants. On the other end, the semantic relationships defined in the ontology could be effectively used to automatically infer links between concepts in different schemata and provide additional information for the reconciliation phase. To this end, object fusion techniques can be used to reconcile multidimensional data returned by different peers.

Finally, the BIN solution can be completed by investigating rooting strategies to select the most promising neighbouring nodes during the reformulation step and dealing with security issues depending on the degree of trust between the BIN participants.

Chapter 4

OLAP Personalization

In this chapter we describe an OLAP query personalization approach that supports BI to ANYONE by coupling an MDX-based language for expressing OLAP preferences to a mining technique for automatically deriving preferences. First, the log of past MDX queries issued by that user is mined to extract a set of association rules that relate sets of frequent query fragments; then, given a specific query, a subset of pertinent and effective rules is selected; finally, the selected rules are translated into a preference that is used to annotate the user's query. A set of experimental results proves the effectiveness and efficiency of our approach.

4.1 Introduction

As described in Section 1.2 different approaches can be pursued to deliver information to an individual or a group of individuals in the most appropriate format and layout. In this chapter, we describe a proactive approach to OLAP personalization to simplify the fruition of BI information [Aligon et al., 2011]. Indeed, in the OLAP context personalization is quite beneficial, because queries can be very complex and they may return huge amounts of data. Aimed at making the user's experience with OLAP as plain as possible, we combine MDX-based language for expressing OLAP preferences to a mining technique for automatically deriving a set of preferences for a user's query from the log of past MDX queries issued by that user. This is done in four steps:

1. The user's query log is mined off-line to extract a set of association rules that relate sets of frequent query fragments (such as group-by attributes, returned measures, selection predicates).

2. When the user formulates a query q, among the rules whose antecedent matches with q, a subset of rules is selected whose cardinality depends on a parameter set

by the user to express the desired personalization degree, i.e., the complexity of the preference that will be formulated.

3. The selected rules are translated into an OLAP preference p concerning the group-by set for aggregating data, the measures to be returned, and the values of attributes or measures.

4. Query q is annotated with p and executed. The results returned are ranked according to p, so that the user can more effectively explore them by focusing on the most relevant data first.

Remarkably, the overall set of tuples returned by q annotated with p is the same set of tuples that would be returned by q without annotation, because p expresses a soft constraint. This guarantees that the user's intentions are preserved, and makes our approach non-invasive. The chapter outline is as follows:

- In Section 4.3, we introduce a formal setting to query multidimensional data based on Definition 2.1 of multidimensional schema.
- In Section 4.4, we describe the main features of the MYMDX language we adopt to express OLAP preferences.
- In Section 4.5, we describe the approach we use to extract and apply preferences to an OLAP query.
- In Section 4.6, we show an implementation of our approach based on the MY-OLAP tool for evaluating preferences [Biondi et al., 2011].
- In Section 4.7 we report the results of a set of experimental tests to prove effectiveness and efficiency of our technique.

4.2 Related Works

Several approaches to personalization were devised in the OLAP context.

In the field of *profile-based personalization*, we mention [Bellatreche et al., 2005], that presents a framework for providing personalized visualization of OLAP results based on user profiles in form of constraints, and [Jerbi et al., 2008], that achieves OLAP personalization by dynamically enhancing queries with context-aware user preferences. Both approaches are proactive and demand low formulation effort, but in both cases the user profile is given, nothing being said on its construction. A recommendation framework for OLAP systems is presented in [Jerbi et al., 2009]; new queries are suggested to users based on the current analysis context and on the user's profile. Though the authors mention that the profile could be mined from the user's previous behavior, no specific suggestion is given to this end. A non-prescriptive approach is presented in [Biondi et al., 2011, Golfarelli et al., 2011b], where the MYOLAP algebra [Golfarelli et al., 2011b] for formulating and evaluating OLAP preferences is introduced; the proposed algebra is very expressive, but at the cost of a substantial formulation effort.

The term *history-based personalization* is borrowed from [Stefanidis et al., 2009], and refers to approaches that suggest a new database query based on the past actions recorded in a log file. The following approaches fall into this category and do not rely on a user profile; they are proactive and demand no formulation effort —like our approach—, but they are prescriptive. The approaches in [Giacometti et al., 2009, 2011] are aimed at suggesting OLAP queries based on a comparison between the current session and former sessions stored in a query log. Also [Chatzopoulou et al., 2009b] has a similar goal in the context of SPJ queries; here, recommendations are computed based on the presence of tuples in sessions. This approach is further improved in [Akbarnejad et al., 2010] by relying on query fragments instead of tuples. A query log is exploited in [Khoussainova et al., 2010a] to support users in writing new SQL queries; the log is transformed into a graph of query fragments, where edges are labelled with the conditional probability of having one fragment given another fragment. Noticeably, all these work generally assume that history is taken from a query log shared by all users.

To the best of our knowledge, our work is the first that proposes to extract preferences from database query logs. However, the same idea has been used in other contexts. In the context of information retrieval, [Veloso et al., 2008] presents algorithms to extract association rules at query time from a set of documents. These rules are used to associate the documents retrieved by a query to a relevance class and eventually to rank them. In the context of the web, [Holland et al., 2003] introduces algorithms for preference extraction from web logs, with a targeted preference language. Extraction is based on the frequency of the terms appearing in the log, and clustering is used for identifying preference constructs. A comprehensive overview of the techniques using data mining for personalization can be found in [Mobasher, 2007].

In the light of the above, we can identify three main gaps in the current literature:

- The user profile is typically given (e.g., as a set of rules) and not automatically derived, implying an initial effort to design the user characteristics and reducing the re-usability of the approaches;
- Log-based approaches (i.e., solutions that rely on the automatic derivation of the user profile) are typically devoted to OLAP recommendation and an overall solution for OLAP personalization is still missing;
- In the context of web and information retrieval, some preference-based approaches exist but they must be adapted to the OLAP context.

4.3 Formal Background

In this chapter we completely reuse the definitions of Section 2.3. In addition, we list the set of MDX statements we will use, and we introduce three new concepts to query multidimensional data, namely: *Query Fragment*, *Query*, and *Log*.

Some of distinguishing features of MDX are the possibility of returning query results that contain data with different aggregation attributes and the possibility of specifying how the results should be visually arranged into a multidimensional representation. We consider MDX queries that aggregate data at one or more group-by sets, optionally select them using a predicate in CNF, and return one or more measures. The semantics of such an MDX query is that of a union of GPSJ queries[1] whose group-by sets are the cross product of n sets of attributes, one for each hierarchy. This semantics corresponds to the following subset of MDX:

- Clauses **SELECT**, **FROM**, **WHERE** are supported.
- All functions for navigating hierarchies are supported: **AllMembers**, **Ancestor**, **Ascendants**, **Children**, etc.
- All functions for manipulating sets of members or tuples are supported (**Crossjoin**, **Except**, **Exists**, **Extract**, **Filter**, **Intersect**, etc.) except the union.
- All functions for manipulating members/tuples are supported.

To effectively use association rules for modeling frequent portions of queries, we formally split MDX queries into fragments as explained below.

Definition 4.1 (Query Fragment, Query, Log)**.** Given schema $\mathcal{M} = \langle A, H, M\rangle$ as defined in Definition 2.1, a *query fragment* is either an attribute in A, a measure in M, or a simple Boolean predicate involving an attribute and/or a measure. A *qf-set* is a set of query fragments. A *multidimensional query* (briefly, *query*) is represented by a qf-set that includes at least one attribute for each hierarchy in H and at least one measure in M. A *log* is a set of multidimensional queries.

Representing an MDX query as a qf-set q means:

1. Including a fragment m in q for each measure m returned by the MDX query.
2. Including a fragment a in q for each attribute a used in the MDX query to aggregate data.

[1]A GPSJ query takes form $\pi_{a_{k_1},\ldots,a_{k_n},T}\sigma_p(\chi)$ where, in our context: χ is the star join between the fact table and the n dimension tables; p is a selection formula in CNF; $\{a_{k_1},\ldots,a_{k_n}\}$ is a group-by set; and T is a list of aggregations of the form $Agg_j(m_j)$, where m_j is a measure and Agg_j is an aggregation operator.

3. Including a fragment $(a \in V)$ in q for each simple predicate on a attribute/measure a used in the MDX query to filter data.

Example 4.1. *The MDX query on the* CENSUS *schema presented in Example 2.1*

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.MRN),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
```

is the union of four GPSJ queries:

$$\pi_{\mathsf{AllCities,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{AllCities,MRN,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,MRN,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$

and is represented by the qf-set $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{MRN}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} \in 2009)\}$.

4.4 The myMDX Preference Language

The language we adopt to express OLAP preferences is myMDX [Biondi et al., 2011], an extension of the MDX language based on the myOLAP algebra. In this section we summarize its features of interest for our approach.

A (qualitative) preference on a datacube is a *strict partial order* (i.e., an irreflexive and transitive binary relation) on the space $\mathcal{F}_{\mathcal{M}}$ of all facts (see Definition 2.3). In the myOLAP algebra, preferences are inductively engineered by writing a *preference expression* that can be either a *base constructor* or a *composition operator* applied to two preference expressions. The constructors used are[2]:

- POS(a, V), where $V \subset Dom(a)$, that operates on attribute values; facts for which a takes a value in V are preferred to the others.
- BETWEEN(m, v_{low}, v_{high}), where m is a measure and $v_{low}, v_{high} \in Dom(m)$, that operates on measure values. Facts whose value of m is between v_{low} and v_{high} are preferred; the other facts are ranked according to their distance from the $[v_{low}, v_{high}]$ interval.

[2]The constructors we adopt are actually a generalization of those presented in [Golfarelli et al., 2011b] from two points of view. Firstly, the CONTAIN constructor is extended to work also on a fake hierarchy including all measures. Secondly, all constructors except BETWEEN are extended to operate on sets of values rather than on single values.

- CONTAIN(h, L), where h is a hierarchy and $L \subset Attr(h)$, that operates on attributes. Facts whose group-by set includes an attribute in L are preferred to the others.

- CONTAIN(measures, $Meas$), where $Meas \subset M$, that operates on measures. Facts whose measure is in $Meas$ are preferred to the others.

Preference composition relies on the Pareto operator ($\otimes$), that gives the same importance to both the composed preferences. Remarkably, the Pareto operator is closed on the set of preferences.

The MYMDX language allows an MDX query to be annotated with a preference expression through a PREFERRING clause.

Example 4.2. *The MDX query in Example 4.1 can be annotated with preference expression* BETWEEN(AvgIncome,500,1000) $\otimes$ POS(Occ,'Engineer') $\otimes$ CONTAIN (RESIDENCE, Region) *to state that facts aggregated by region and related to engineers with average income between 500 and 1000 kiloeuros are equally preferred. The corresponding* MYMDX *query is:*

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.MRN),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
PREFERRING AvgIncome BETWEEN 500 AND 1000
AND Occ POS 'Engineer' AND RESIDENCE CONTAIN Region
```

4.5 A Personalization Framework

As sketched in Section 4.1, our approach relies on four steps:

1. *Log mining.* For efficiency reasons this step is executed off-line, before the current query session starts. It consists in running a data mining algorithm on the user's query log to extract the set R of association rules whose support and confidence are above a given threshold.

2. *Rule selection.* When that user formulates an MDX query q, a subset $R_q \subseteq R$ of rules is selected. Each rule in R_q is *pertinent*, meaning that its antecedent matches with q, and *effective*, meaning that the preference it would be translated into can actually induce an ordering on the facts returned by q. Then, let a positive integer *personalization degree* α be chosen by the user to express the desired

Algorithm 4 Extract rules with support and confidence adjustment

Input: Log: A set of queries; $minSup, minConf$: Floats
Output: R: A set of association rules
Uses: $mine(set, float, float)$: An association rule extractor
Variables: $stop$: A Boolean; $confidence, support$: Floats; $Covered$: A set of qf-sets

```
1:  $stop$ =false
2:  $confidence = 1$
3:  $support = 1$
4:  while $!stop$ do
5:    $R = mine(Log, support, confidence)$ {Mine rules above $support$ and $confidence$}
6:    $R = R \setminus \{r \in R$ s.t. $|r.cons| > 1\}$ {Only keep rules with singleton consequent}
7:    $Covered = \emptyset$
8:    for each rule $r \in R$ do
9:      $Covered = Covered \cup \{q \in Log | r.ant \cup r.cons \subseteq q\}$
10:   end for
11:   if $Covered = Log$ then
12:     $stop = true$
13:   else
14:     $confidence = confidence - 0.1$
15:     if $confidence < minConf$ then
16:       $support = support - 0.1$
17:       $confidence = 1$
18:       if $support < minSupp$ then
19:         $stop = true$
20:       end if
21:     end if
22:   end if
23: end while
24: return $R$
```

preference complexity. A qf-set F_α is generated from R_q in such a way that α base constructors are included in the overall preference expression the fragments of F_α will be translated into.

3. *Fragment translation.* Each fragment in F_α is translated into a base constructor; the resulting base constructors are then coalesced and composed using the Pareto operator into a preference expression p.

4. *Querying.* Query q is annotated with p, translated into MYMDX, and executed. As shown in [Biondi et al., 2011], the user can effectively explore query results by visually interacting with a graph-like structure that emphasizes the better-than relationships induced by p between different sets of facts. Preferred facts are then displayed in a multidimensional table.

The following subsections explain in detail how steps 1, 2, and 3 are carried out. For details about step 4, see [Biondi et al., 2011, Golfarelli et al., 2011b].

4.5.1 Log Mining

We now briefly describe the mining step. The input of this step is a set of qf-sets that represents the user's query log, while the output is a set R of association rules.

Interestingly, the problem of associating a query with a set of fragments representing user preferences bears resemblance to the problem of associating objects with a set of most relevant labels. This problem, named *label ranking*, is a form of classification. Both label ranking and classification have been proved to be effectively handled by association rules (see for instance [Li et al., 2001, Sá et al., 2011]). In this context, rules have a set of features that should match the object to be classified as antecedent, and one label as consequent. We adopt a similar approach here, and we search for rules having exactly one item as consequent, so each rule $r \in R$ takes the form $ant \rightarrow cons$, where ant is a qf-set and $cons$ is a single query fragment. In the following, $r.cons$ (resp., $r.ant$) denotes the consequent (resp., antecedent) of rule r, and $conf(r)$ its confidence.

The mining step is done off-line, and uses any classical association rule extractor that is parametrized by support and confidence thresholds (e.g., Apriori [Agrawal and Srikant, 1994]). The only issue in this step is to extract rules that faithfully represent the user's query log. Since the user is not involved at this step, support and confidence have to be adjusted automatically [Sá et al., 2011]. Algorithm 4 is used for this purpose, and it extracts rules until the whole log is covered by the set of rules extracted. More precisely, the algorithm starts extracting rules with confidence and support equal to 1 (lines 2,3). If the set of rules covers the entire log, then the algorithm stops (line 11,12). Otherwise, extraction starts again with a lower confidence (line 13), and confidence is decreased until the log is entirely covered or the confidence is considered too low (line 14). In this case, confidence goes back to 1 and support is decreased (line 16,17), and extraction is launched again. If both support and confidence are considered too low, then the algorithm stops.

Algorithm 4 needs two thresholds, $minConf$ and $minSupp$. Realistic values for these thresholds can be learned by training the algorithm on query logs, or be derived from log properties like size and sparseness.

4.5.2 Rule Selection

The output of the mining step, R can be a large set. In this section we present the algorithm that first selects, among the rules in R, the subset R_q of pertinent and effective rules for query q, and then returns a qf-set F_α including a subset of the query fragments that appear as consequents of the rules in R_q. These fragments will be used for annotating q with a preference.

Following the approach presented in [Veloso et al., 2008], the selection of query fragments is made by associating a score to each group of rules in R_q having the same fragment φ as consequent. This score is the average confidence of the rules in the group, i.e., $score(\varphi) = avg_{r \in R_\varphi} conf(r)$ where $R_\varphi \subseteq R_q$ is the subset of rules having φ as a consequent. The selected query fragments are those with highest scores, and are limited by the number α of base preference constructors that the user wants to annotate her queries with.

Algorithm 5 Select Consequents

```
Input: $R$: A set of rules; $q$: A query represented as a qf-set; $\alpha$: A user-defined personalization degree
Output: $F_\alpha$: A qf-set that will be used to annotate $q$ with a preference
Variables: $numBC$: The current number of base constructs; $R_q$: The set of pertinent and effective rules; $F$, $F_{sim}$: Two qf-sets
1: $R = R \setminus \{r \in R | r.ant \not\subseteq q\}$ {Drop non-pertinent rules}
2: $R_q = R \setminus \{r \in R | r.cons \in A \cup M, r.cons \notin q\}$ {Drop non-effective rules}
3: $F = \{r.cons | r \in R_q\}$ {Consequents of the rules in $R_q$}
4: $F_\alpha = \emptyset$
5: $numBC = 0$
6: while $numBC \leq \alpha$ and $F \neq \emptyset$ do
7:   let $\varphi = ArgMax_F\ score(\varphi)$ {...starting with the fragment having highest score}
8:   $F = F \setminus \{\varphi\}$
9:   if $makesIneffective(\varphi, F_\alpha, q)$ then
10:    $F_{sim} = \{\varphi' \in F_\alpha | similar(\varphi, \varphi')\}$ {...find the similar fragments, if any...}
11:    $F_\alpha = F_\alpha \setminus F_{sim}$ {...and drop them}
12:    if $F_{sim} \neq \emptyset$ then
13:      $numBC - -$
14:    end if
15:  else
16:    if $\exists \varphi' \in F_\alpha | similar(\varphi, \varphi')$ then
17:      $F_\alpha = F_\alpha \cup \{\varphi\}$
18:    else
19:      if $numBC < \alpha$ then
20:        $F_\alpha = F_\alpha \cup \{\varphi\}$
21:      end if
22:      $numBC + +$
23:    end if
24:  end if
25: end while
26: return $F_\alpha$
```

Given schema $\mathcal{M} = \langle A, H, M \rangle$ and a qf-set F, we adopt the following notation:

- $F.hier(h) = F \cap Attr(h)$ is the set of attributes of hierarchy $h \in H$ in F;
- $F.meas = F \cap M$ is the set of measures in F;
- $F.val(a) = \bigcup_{(a \in V_k) \in F} V_k$ denotes the set of selected values for attribute/measure $a \in A \cup M$ in F.

Function 6 makesIneffective

```
Input: $\varphi$: A fragment; $F_\alpha$: A qf-set; $q$: a query represented as a qf-set
Output: A Boolean
1: if $\exists h \in H | \varphi \in Attr(h)$ then
2:   if $(F_\alpha.hier(h) \cup \{\varphi\}) = q.hier(h)$ then
3:     return true
4:   end if
5: end if
6: if $\varphi \in M$ then
7:   if $(F_\alpha.meas \cup \{\varphi\}) = q.meas$ then
8:     return true
9:   end if
10: end if
11: if $\varphi = (a \in V)$ then
12:   if $q.val(a) \neq \emptyset$ and $!((F_\alpha.val(a) \cup V) \subset q.val(a))$ then
13:     return true
14:   end if
15: end if
16: return false
```

Algorithm 5 selects, among the set R of association rules mined from the log, the consequents of rules that will be used to annotate the current query with preferences.

Function 7 similar

```
Input: $\varphi_1$: A fragment; $\varphi_2$: A fragment
Output: A Boolean
 1: if $\exists h \in H | \varphi_1 \in Attr(h)$ and $\varphi_2 \in Attr(h)$ then
 2:     return true
 3: end if
 4: if $\varphi_1 \in M$ and $\varphi_2 \in M$ then
 5:     return true
 6: end if
 7: if $\varphi_1 = (a \in V_1)$ and $\varphi_2 = (a \in V_2)$ then
 8:     return true
 9: end if
10: return false
```

It starts by removing from R all non-pertinent rules (i.e., those whose antecedent does not match q — line 1), and some non-effective rules (those whose consequent, if it is an attribute or a measure, does not appear in the list of group-by attributes or returned measures of q — line 2). The remaining rules are grouped by their consequent and the score of each group is computed (line 3). Then the top consequents corresponding to α base constructors are returned (lines 4-21). If a fragment φ that is about to be selected drives the preferences ineffective because it states that *all* the query results are preferred (Function 6), it is removed together with the other similar fragments (lines 10-13).

Example 4.3. *Consider the qf-set of Example 4.1,* $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{MRN}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} \in 2009)\}$*. Let the set* R *of rules extracted from the log be as follows:*

r_1:	$(\mathsf{Region} \in \{\textit{'Pacific'}, \textit{'Atlantic'}\}) \rightarrow \mathsf{Year}$	*(0.8)*
r_2:	$\mathsf{Year} \rightarrow \mathsf{Region}$	*(0.80)*
r_3:	$\mathsf{Year} \rightarrow \mathsf{AllCities}$	*(0.60)*
r_4:	$\mathsf{AvgIncome} \rightarrow \mathsf{Region}$	*(0.60)*
r_5:	$\mathsf{Year} \rightarrow \mathsf{Sex}$	*(0.90)*
r_6:	$(\mathsf{Year} \in 2009) \rightarrow \mathsf{Region}$	*(0.70)*
r_7:	$\mathsf{Year} \rightarrow (\mathsf{Year} \in 2009)$	*(0.50)*
r_8:	$\mathsf{Year} \rightarrow (\mathsf{AvgIncome} \in [500, 1000])$	*(0.55)*
r_9:	$\mathsf{AvgIncome} \rightarrow \mathsf{MRN}$	*(0.45)*
r_{10}:	$\mathsf{Occ} \rightarrow \mathsf{Region}$	*(0.70)*
r_{11}:	$\mathsf{Occ} \rightarrow \mathsf{Year}$	*(0.10)*
r_{12}:	$\mathsf{AvgIncome} \rightarrow \mathsf{Year}$	*(0.70)*

and let Algorithm 5 be called with $\alpha = 2$*. First, the algorithm removes* r_1 *(non pertinent) and* r_5 *(non effective). Then the remaining rules are grouped by their consequents, resulting in the set of fragments* $F = \{\mathsf{Region}, \mathsf{AllCities}, (\mathsf{AvgIncome} \in [500, 1000]), (\mathsf{Year} \in 2009), \mathsf{MRN}, \mathsf{Year}\}$ *(listed by decreasing order of score). The fragments in* F *are now orderly explored. The first two fragments are not selected since, together, they drive the preference ineffective (they are exactly the fragments of hierarchy* RESIDENCE *included in* q*). Fragment* $(\mathsf{AvgIncome} \in [500, 1000])$ *is selected.*

Fragment (Year $\in$ 2009) *is not selected since it corresponds precisely to the selection on* Year *of* q. *Then fragment* MRN *is selected and, finally, Algorithm 5 outputs* $F_\alpha = \{(\mathsf{AvgIncome} \in [500, 1000]), \mathsf{MRN}\}$.

4.5.3 Fragment Translation

The output F_α of Algorithm 5 is a qf-set used to annotate the current query q with a preference. To this end, each query fragment $\varphi \in F_\alpha$ is translated into a base constructor (see Section 4.4); the resulting base constructors are then coalesced and composed using the Pareto operator.

The rules for translating fragment φ are explained below:

- if φ is an attribute $a \in A$, it is translated into a constructor CONTAIN(h, a), where h is the hierarchy a belongs to.
- If φ is a measure $m \in M$, it is translated into a constructor CONTAIN(measures,m).
- If φ is a Boolean predicate on a attribute, $(a \in V)$, it is translated into a constructor POS(a, V).
- If φ is a Boolean predicate on a measure, $(m \in [v_{low}, v_{high}])$, it is translated into a constructor BETWEEN(m, v_{low}, v_{high}).

The resulting base constructors are coalesced by merging all CONTAIN's on the same hierarchy, all POS's on the same attribute, and all BETWEEN's on the same measure.

Example 4.4. *The preference expression that translates the qf-set* F_α *in Example 4.3 is* $p =$ BETWEEN(AvgIncome,500,1000) $\otimes$ CONTAIN(RACE, MRN). *The* MYMDX *formulation for* q *annotated with* p *is:*

```
SELECT AvgIncome ON COLUMNS,
          Crossjoin(OCCUPATION.members,
                    Crossjoin(Descendants(RACE.AllRaces,RACE.MRN),
                    Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
PREFERRING AvgIncome BETWEEN 500 AND 1000 AND RACE CONTAIN MRN
```

4.6 Implementation

The approach was implemented in Java, using the Mondrian API for handling MDX queries, the Weka implementation of Apriori for rule extraction, and the MYOLAP tool for evaluating preferences. The tests were conducted starting from synthetic MDX logs generated through Algorithm 8, that uses the *Diff* operator proposed in [Sarawagi, 1999]. This operator explores the reasons why an aggregate is significantly lower in one fact compared to another. It takes as parameters two facts f and f' and an integer N, and looks into the two isomorphic sub-cubes C and C' that detail the two facts (i.e., that are aggregated to form f and f'). As a result, it summarizes the differences in these two sub-cubes by providing the top-N informative pairs of cells. Our generator simulates OLAP sessions on a datacube by starting from a random query q and then deriving the subsequent queries in the session using the result of the *Diff* operator applied to q. The Java implementation of *Diff* was obtained from [Sarawagi, 2009]; N is set to 20 to simulate OLAP sessions including no more than 20 queries.

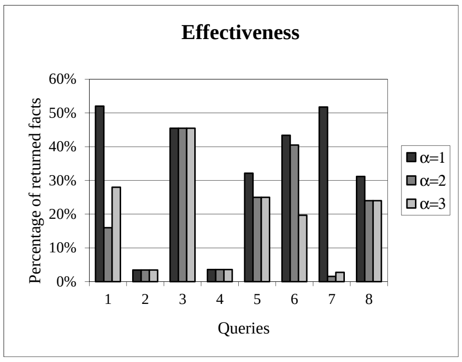
Algorithm 8 Generate a log

```
Input: $minSize$: Minimum log size
Output: $Log$: A set of queries
Uses: $Diff(cell, cell)$: The Diff operator defined in [Sarawagi, 1999]
Variables: $q$: A query ; $nbGenerated$: Integer
 1: $nbGenerated = 0$
 2: while $nbGenerated < minSize$ do
 3:     randomly generate a query $q$ on a sub-cube
 4:     $Log = Log \cup \{q\}$
 5:     $nbGenerated{+}{+}$
 6:     let $f_1, f_2$ be facts that show the maximum difference in the result of $q$
 7:     for each pair $\langle f'_1, f'_2 \rangle \in Diff(f_1, f_2)$ do
 8:         let $q'$ be the drill-down of $q$ to the group-by set of $f'_1$ and $f'_2$
 9:         $Log = Log \cup \{q'\}$
10:         $nbGenerated{+}{+}$
11:     end for
12: end while
13: return $Log$
```

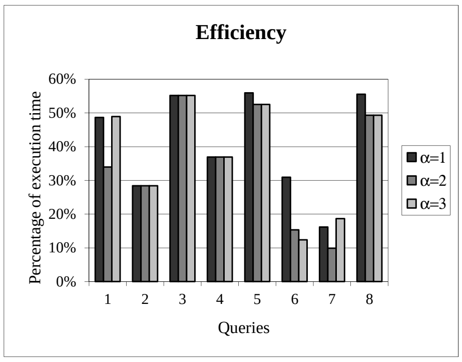
4.7 Validation

We validated our approach to proof both effectiveness and efficiency.

The architecture used for testing is an Intel Core 2 Duo 3 GHz, with 4GB RAM. All tests were made on the CENSUS schema presented in Example 2.1, corresponding to about 10^7 facts stored on Oracle 11g. For our tests, we generated a log of about 1000 queries; the initial query of each session was generated randomly by selecting group-by sets, measures and selections from a small pool. A small selection pool (3 selections on different dimensions) is used to simulate the log of a single user querying a sub-cube. Then, 8 queries to be personalized were extracted randomly from the log and removed from it. Minimum support and confidence were adjusted with Algorithm 4 to 0.6 and 0.7, respectively, resulting in 20 rules that cover the log and have an average support and confidence of 0.63 and 0.85, respectively. The confidence ranges from 0.76 to 1, with a standard deviation of 0.063.



(a)



(b)

FIGURE 4.1: Effectiveness and efficiency of our approach

As to effectiveness, Figure 4.1.a reports, for each query in the benchmark, the ratio between the number of preferred facts returned by the annotated query (i.e., those included in the *best-match only* result of the query [Golfarelli et al., 2011b]) and the one returned by the original query, when the personalization degree ranges between 1 and 3. Our approach is always effective in reducing the number of facts returned to the user. Though in general the reduction gets stronger as the personalization degree is increased, two different trends are apparent. In some cases (queries 2, 3, and 4) the reduction is independent on the personalization degree since only one pertinent and effective fragment was found. In other cases (queries 1 and 7), as the complexity of the preference increases, there are no facts that fully satisfy it so a larger set of facts that partially satisfy the preference are returned.

As to efficiency, we point out that the log mining step was executed in less than 4 secs, while the time for rule selection and fragment translation never exceeded 5 msecs. Figure 4.1.b reports the ratio between the time taken to execute each annotated query and the time to execute the original query. The reduction is always above 40%, and it is not relevantly affected by the personalization degree. Overall, we can conclude that our approach to personalization not only puts no overhead on the querying process, but it significantly reduces query response times.

4.8 Conclusions

In this chapter we described a personalization framework to annotate OLAP queries with preferences, so as to support the concept of BI to ANYONE [Aligon et al., 2011]. In particular, we improve the user experience with OLAP relying on three different aspects:

- Formulation effort: typically, personalization criteria for queries may be either manually specified by users, or transparently inferred from the context and from the user profile. Our approach is based on log mining and rule selection techniques allowing an automatically extraction of the user preferences.

- Prescriptiveness: personalization criteria may either be used as “hard” constraints that are added to queries, or be meant as “soft” constraints. We annotate query with preferences preserving the initial user’s intentions (i.e., low prescriptiveness).
- Proactiveness: in the literature, some approaches propose new queries to the user based on the query log and on the context, while others change the current query or post-process its results before returning them to the user. We enhance proactiveness by transparently changing the current query.

While in this chapter we used preference mining for result ranking, in the next chapter we will attempt to generalize it to address query recommendation as well. Besides, we will investigate the feasibility of extending our approach to incrementally manage OLAP sessions, i.e., to take delta queries into account at runtime without having to mine the log from scratch.

Chapter 5

OLAP Similarity

In this chapter, we support BI to ANYONE by proposing different similarity measures oriented to the OLAP recommendation. A recommendation system suggests the most promising direction of analysis to extract relevant BI information. In this context, exploiting the similarity between the current OLAP session and those issued in the past by the same user or by a group of similar users, represents an added value. In this direction, we devised several similarity measures to compare OLAP sessions from two perspectives: queries and sessions. We prove the effectiveness of our measures with both synthetic and real data.

5.1 Introduction

The OLAP paradigm has revolutionized the way users access information in multidimensional databases. This paradigm achieves the ambitious goal of coupling a large querying expressiveness with a small query formulation effort, by providing a set of operators (such as drill-down and slice-and-dice) to transform one multidimensional query into another. As a consequence, OLAP queries are not normally formulated in isolation, but in the form of sequences (*OLAP sessions*). During an OLAP session focused on a phenomenon –such as sales– the user analyzes the results of a query and, depending on the specific data she sees, interactively chooses to apply one operator to determine a new query that will give her a better view of that phenomenon. The extemporary sequences of queries that are created this way are strongly related to the issuing user, to the analyzed phenomenon, and to the current data. Though some works are focused on assessing the similarity between OLAP queries [Aouiche et al., 2006, Golfarelli, 2003, Sapia, 2000], similarity of OLAP sessions has been only marginally taken into account. The similarity of sessions of SQL queries, disregarding order, is assessed by [Agrawal et al., 2006]. [Aouiche et al., 2006] proposes a basic measure for similarity between sets of OLAP queries (again disregarding query order) aimed at clustering a workload. [Giacometti et al., 2009] compares OLAP sessions based on the order of queries, using edit distance, but at the extensional attribute —which may create efficiency problems. However, no systematic study exist to compare different similarity measures for OLAP sessions; in particular, though both [Giacometti et al., 2009] and [Agrawal et al., 2006] aim at assisting the user, no users were apparently involved in the design of the similarity measures proposed. To fill these gaps, we devised a two-attribute approach to compare OLAP sessions based on the similarity of their queries [Aligon et al., 2013]. In particular, we gave the following contributions:

- In Section 5.3, we propose a set of criteria for OLAP sessions similarity derived from the results of a user study conducted with a set of practitioners and researchers in the OLAP field.
- In Section 5.5, we propose a function for estimating the similarity between OLAP queries based on three components: the query group-by set, its selection predicate, and the measures required in output.
- In Section 5.6, we study session similarity investigating the feasibility of two-attribute extensions (i.e., that compare query sequences based on the similarity between their elements) of four popular methods for measuring similarity, namely the Levenshtein distance, the Dice coefficient, the tf-idf weight, and the Smith-Waterman algorithm.
- In Section 5.7, we experimentally compare these four extensions from both points of view of efficiency and effectiveness. The results clearly show that the Smith-Waterman extension is the one that best captures the users' criteria for session similarity.

5.2 Formal Background

Starting from Definition 2.1 of multdimensional schema and Definition 2.2 of group-by set, we introduce the concepts of OLAP query and OLAP session.

Definition 5.1 (OLAP Query)**.** A *query* on schema $\mathcal{M} = \langle A, H, M \rangle$ is a triple $q = \langle g, Pred, Meas \rangle$ where:

1. $g \in Dom(H)$ is the query group-by set;
2. $Pred = \{p_1, \ldots, p_n\}$ is a set of Boolean predicates, one for each hierarchy, whose conjunction defines the *selection predicate* for q; conventionally, $p_i = TRUE_i$ if no selection on h_i is made in q;
3. $Meas \subseteq M$ is the measure set whose values are returned by q.

TABLE 5.1: Queries for Example 5.1

		Queries									
		q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
Group-by set		g_1	g_2	g_2	g_2	g_2	g_3	g_3	g_2	g_1	g_1
Measures	AvgCostWatr	✓	✓	✓	✓	✓	✓	✓	✓		
	AvgCostElect	✓	✓	✓		✓	✓		✓		
	AvgCostGas			✓						✓	✓
	AvgIncome							✓	✓		✓
Selection predicates		c_1	c_1	c_1	c_2	c_3	c_1	c_1	c_1	c_1	c_1

Definition 5.2 (OLAP Session). An *OLAP session* of length v is a sequence $s = \langle q_1, \ldots, q_v \rangle$ of v queries on schema $\mathcal{M}$.

Example 5.1. *In the following an example of simple log, based on CENSUS schema:*

$$\begin{aligned} s &= \langle q_1, q_2, q_3 \rangle \\ s' &= \langle q_4, q_5, q_6, q_7, q_8 \rangle \\ s'' &= \langle q_9, q_{10} \rangle \end{aligned}$$

Table 5.1 represents each query in terms of our query model; the involved group-by sets are those used in Example 2.1 of Chapter 2, while the selection predicates are:

$$\begin{aligned} c_1 &= \{TRUE_{\mathsf{RESIDENCE}}, \ldots, (\mathsf{Year} = 2005), \ldots, TRUE_{\mathsf{SEX}}\} \\ c_2 &= \{TRUE_{\mathsf{RESIDENCE}}, (\mathsf{RaceGroup} = Chinese), \ldots, \ldots, TRUE_{\mathsf{SEX}}\} \\ c_3 &= \{TRUE_{\mathsf{RESIDENCE}}, (\mathsf{RaceGroup} = Chinese), (\mathsf{Year} = 2005), \ldots, TRUE_{\mathsf{SEX}}\} \end{aligned}$$

□

5.3 Requirements for OLAP sessions similarity

The goal of this section is to list a number of requirements to be used for (i) understanding which approaches, among all those proposed in the literature for query and sequence comparison, are eligible for the OLAP context; and (ii) driving the adaptation and extension of the eligible approaches towards the development of an original approach to OLAP session comparison.

We start by proposing a first set of requirements, suggested by the specific features of the OLAP context and by our experience in the field:

♯1 Multidimensional databases store huge amounts of data, and OLAP queries may easily return large volumes of results. Computing similarity at the extensional attribute, i.e., by comparing the data resulting from queries, would pose serious efficiency problems in this context, and would discourage the use of the approach for recommendation and personalization —that require a fast interaction with users. Indeed, as noted by [Chatzopoulou et al., 2011] in the case of recommendation of SQL queries, there is a clear trade-off between efficiency and quality, when a fragment based model or a tuple based model is used. For this reason we compute similarity at the intensional attribute, i.e., considering only query expressions.

♯2 It is unlikely that two OLAP sessions share identical queries; this feature is better managed by having comparisons of single queries result in a score rather than in a Boolean.

♯3 A typical OLAP query is defined by the fact to be analyzed, one or more measures to be computed, a set of hierarchy attributes for aggregating measure values, a predicate for filtering a subset of events, and a presentation. Though the presentation chosen for displaying the results of an OLAP query (e.g., a cross-tab or a pie-chart) certainly has an influence on how easily users can interpret these results, it does not affect the actual informative content, so it should not be considered when comparing queries.

To discover additional requirements for OLAP sessions similarity, we conducted a user study. We prepared a questionnaire asking to give a qualitative evaluation of the similarity between couples of OLAP queries and couples of OLAP sessions over a simple multidimensional schema (more details will be given in Subsection 5.7.1). The questionnaire[1] was submitted to all the teachers and PhD students of the First European Business Intelligence Summer School (eBISS 2011)[2], as well as to the master students of two specialistic courses on DW design at the Universities of Bologna (Italy) and Tours (France). All people involved had some experience as OLAP users, most of them had some practice of multidimensional design too. Overall, 41 answers were collected. The additional requirements emerging from an analysis of the questionnaire results can be summarized as follows:

♯4 The selection predicate is the most relevant component in determining the similarity between two OLAP queries, followed by the group-by set. The less important component is the set of measures to be returned.

♯5 The order of queries is relevant in determining the similarity between two sessions, i.e., two sessions sharing the same queries but in different orders have low similarity.

♯6 Recent queries are more relevant than old queries in determining the similarity between two OLAP sessions. Since the time actually elapsed between two consequent

[1] Available at `http://www.julien.aligon.fr/recherche/similarityform.aspx`.

[2] `http://cs.ulb.ac.be/conferences/ebiss2011/`

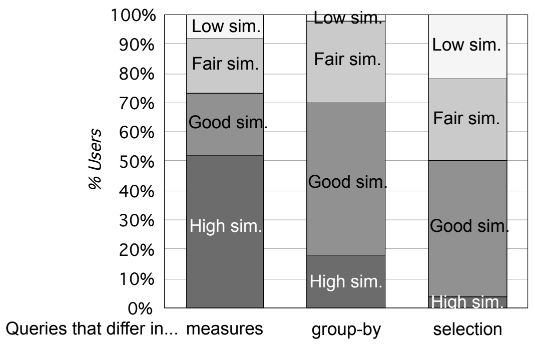


FIGURE 5.1: Perceived similarities for OLAP queries only differing in one of their three main components

queries in a session depends on several unpredictable factors (e.g., the query execution time, the size and complexity of the data returned, the user's query formulation skills), only the *order* of queries will be considered.

♯7 The longest the matching fraction of two sessions, the highest their similarity.

♯8 Two sessions that match with one or more gaps (i.e., one or more non-matching queries are present) are similar, but their similarity is lower than the one of two sessions that match with no gaps.

In particular, as to point ♯4, in Figure 5.1 we show the percentages of users that perceive a given attribute of similarity for couples of queries that only differ in either their measure sets, or their selection predicates, or their group-bys. Apparently, measures are the less important component in determining similarity since most users perceive as highly similar two queries that only differ in their measures. The opposite holds for the selection predicate component.

5.4 Related Works

This section reviews the literature for similarity functions that could possibly be used to compare OLAP sessions. Since OLAP sessions are sequences of queries, we first review the approaches for comparing sequences and then those for comparing database queries. The requirements expressed in Section 5.3 are used to restrict the set of approaches that are candidate to be adopted in the OLAP context.

5.4.1 Sessions

Comparing sequences has attracted a lot of attention especially in the context of string processing, with applications like information retrieval, spell-checkers, bioinformatics, and record linkage [Moreau et al., 2008]. The existing approaches are inspired by different principles.

In *token-based approaches* sequences are treated as bags of elements, and classical set similarity functions like Jaccard and Hausdorff, and all their variants, can be used or adapted. Of course, these approaches are not sensible to the order of sequence elements. When the sequences to be compared are taken from a *corpus*, the popular *term frequency-inverse document frequency* (tf-idf) weight can be adopted, which weights each element of a sequence using (positively) their frequency in the sequence and (negatively) their frequency in the corpus. A cosine is then used to measure the similarity between two vectors of weights.

Some approaches compare two sequences by comparing their subsequences. A basic approach here is to use the size of the longest common subsequence (LCS).[3] An approach often used in statistical natural language processing relies on n-grams, i.e., substrings of size n of a given sequence [Brown et al., 1992]. A popular similarity function using n-grams is the *Dice coefficient*, an extension of the Jaccard index defined as twice the number of shared n-grams over the total number of n-grams:

$$Sim_{Dice}(s, s') = \frac{2|ngrams(s) \cap ngrams(s')|}{|ngrams(s)| + |ngrams(s')|}$$

Other approaches compare sequences based on their *edit distance*, i.e., in terms of the cost of the atomic operations necessary to transform one sequence into another. Many edit distances have been proposed that differ on the number, type, and cost of the edit operations. The most popular are the *Levenshtein distance*, that allows insert, delete, and substitute, and the *sequence alignment distance*, that allows match, replace, delete, and insert [Navarro, 2001].

Finally, in *two-attribute* approaches sequences are compared based on the similarity between their elements. A simple example is the Hausdorff distance between sets, that relies on the distance between elements of the set. In [Monge and Elkan, 1997] the similarity between sequences s and s' is the average of the highest similarities between pairs of elements of s and s':

$$Sim_{M\&E}(s, s') = \frac{1}{|s|} \sum_{s_i \in s} max_{s'_j \in s'} \{Sim_{elem}(s_i, s'_j)\}$$

where Sim_{elem} measures the similarity between single elements. In *soft tf-idf* [Cohen et al., 2003], the tf-idf weight is extended using the similarity of sequence elements; more precisely,

$$Sim_{soft}(s, s') = \sum_{s_i \in Close_\theta(s,s')} T(s_i, s) \cdot T(s_i, s') \cdot max_{s'_j \in s'} \{Sim_{elem}(s_i, s'_j)\}$$

[3]Note that, while substrings are consecutive parts of a string, subsequences need not be.

where $T(s_i, s)$ is a normalized form of the tf-idf of element s_i within sequence s, θ is a threshold, and $Close_\theta(s, s')$ is the set of elements $s_i \in s$ such that there is at least an element $s'_j \in s'$ with $Sim_{elem}(s_i, s'_j) > \theta$. While the two previous two-attribute approaches do not consider the ordering of elements within sequences, the *Smith-Waterman algorithm* relies on element ordering; it can be used to efficiently find the best alignment between subsequences of two given sequences by ignoring the non-matching parts of the sequences [Smith and Waterman, 1981]. It is a dynamic programming algorithm based on a matrix H whose value in position (i, j) expresses the score for aligning subsequences of s and s' that end in elements s_i and s'_j, respectively. This matrix is recursively defined based on the following formula:

$$H(i,j) = max \left\{ \begin{array}{c} 0; \\ H(i-1, j-1) + Sim_{elem}(s_i, s'_j); \\ max_{k \geq 1}\{H(i-k, j) - cost_k\}; \\ max_{k \geq 1}\{H(i, j-k) - cost_k\} \end{array} \right\}$$

where $cost_k$ is the cost of introducing a gap of length k in the matching between s and s'. Note that, here, the similarity between two elements can be negative, to express that there is a mismatch between them; intuitively, the algorithm seeks an optimal trade-off between the cost for introducing a gap in the matching subsequences and the cost for including a poorly matching pair of elements.

We conclude this overview with a couple of brief observations about the features a sequence comparison approach should have to be used for OLAP sessions:

- In OLAP sessions, the order of queries is relevant (requirement ♯5), which discourages from taking token-based approaches.
- Mostly, OLAP sessions do not share the very same queries (requirement ♯2). This makes two-attribute approaches, that take advantage of a similarity function for OLAP queries, more suitable for our purposes.
- Following requirement ♯8, it is important to be able to determine similar regions in two globally different sessions, which favors a sequence alignment approach.

5.4.2 Queries

As to query similarity, we can distinguish two main motivations for comparing database queries. The first one is query optimization, where a query q to be evaluated is compared to another query q', with the goal of finding a better way of evaluating q. This motivation attracted a lot of attention, and covers classical problems like view usability [Garcia-Molina et al., 2008, Gupta and Mumick, 1999], query containment [Abiteboul et al., 1995], plan selection [Ghosh et al., 2002], view selection [Golfarelli, 2003], and data prefetching [Sapia, 2000]. The second, more recent, motivation is to suggest a query to the user without focusing on its evaluation. In this context, a query is compared to another one with the goal of helping the user exploring or analyzing a database. This includes query completion [Yang et al., 2009] and query recommendation [Akbarnejad et al., 2010, Chatzopoulou et al., 2009a, 2011, Drosou and Pitoura, 2011, Giacometti et al., 2009, Stefanidis et al., 2009].

From a technical point of view, the approaches found in the literature can be classified according to (i) the *query model* they adopt, i.e., the structure used to compactly represent queries; (ii) the *information source* from which the representation of each query is derived; and (iii) the *function* used to compute similarity.

Query models range from a string corresponding to the uninterpreted SQL sentence [Yao et al., 2005] to the set of tuples resulting from the query evaluation [Drosou and Pitoura, 2011, Stefanidis et al., 2009]. Queries can also be modeled as vectors of features with either a score or a Boolean for each feature [Agrawal et al., 2006, Akbarnejad et al., 2010, Aouiche et al., 2006, Ghosh et al., 2002], or as sets of *fragments*, each representing a particular part of the query, such as the attributes required in output (SELECT clause) or the table names in the cross product (FROM clause) [Sapia, 2000]. Finally, queries are sometimes modeled as graphs, following the database schema like in [Yang et al., 2009].

As to the information source, it can be the query expression, e.g., the uninterpreted query text [Yao et al., 2005] or the list of query fragments (selection predicates, projection, etc.) [Garcia-Molina et al., 2008, Yang et al., 2009]. When fragments are used, only some of them may be taken into account; for instance, only the selection attributes are used by [Agrawal et al., 2006] and [Yang et al., 2009] whereas all fragments are used by [Garcia-Molina et al., 2008] and [Gupta and Mumick, 1999]. The information source can also be related to the database queried; more precisely, it can be:

- The database instance, e.g., the query result or the active domain of the database attributes [Agrawal et al., 2006, Chatzopoulou et al., 2009a, 2011, Drosou and Pitoura, 2011, Giacometti et al., 2009, Stefanidis et al., 2009]. In the former case, the query can be evaluated either fully [Drosou and Pitoura, 2011, Stefanidis et al., 2009] or partially [Giacometti et al., 2009]. In this category we also include an approach for measuring similarity between multidimensional cubes [Baikousi et al., 2011], because obviously an OLAP query returns a multidimensional cube.

- The statistics used by the query optimizer, like table sizes and attribute cardinalities [Ghosh et al., 2002].

- The database schema, e.g., the keys defined or the index used to process a selection [Ghosh et al., 2002, Golfarelli, 2003].

Table 5.2: Query comparison approaches at a glance

Ref.	*Motivation*	*Model*	*Source*	*Similarity Function*
[Gupta and Mumick, 1999]	optimization	sets	S, P, C	fragment tests
[Chatzopoulou et al., 2011]	recommend.	vector	db instance, log	cosine
[Akbarnejad et al., 2010]	recommend.	vector	S, P, log	cosine
[Agrawal et al., 2006]	optimization	vector	S, db instance	cosine
[Aouiche et al., 2006]	optimization	vector	S, P, log	Hamming distance
[Ghosh et al., 2002]	optimization	vector	S, C, db statistics	Hamming distance
[Stefanidis et al., 2009] (1)	recommend.	vector	log	inner product
[Stefanidis et al., 2009] (2)	recommend.	set	db instance	Jaccard index
[Giacometti et al., 2009]	recommend.	set	db instance	Hausdorff distance
[Sapia, 2000]	optimization	sets	S, P	query repres. equality
[Golfarelli, 2003]	optimization	set	P, db schema & statistics	group-by lattice
[Yao et al., 2005]	recommend.	string	SQL sentence	entropy
[Yang et al., 2009]	recommend.	graph	S, P, C	query repres. equality

- The query log, if the query model relies on other queries that have previously been launched on the same database. For instance, [Chatzopoulou et al., 2009a], [Chatzopoulou et al., 2011], [Akbarnejad et al., 2010], [Aouiche et al., 2006], and [Stefanidis et al., 2009] model a query in terms of its links with other queries or how many times it appears in the log.

Finally, the result of query comparison can be a Boolean or a score, usually normalized in the [0..1] interval. The first case applies when queries are tested for equivalence [Abiteboul et al., 1995] or view adaptation [Gupta and Mumick, 1999], or when the goal is to group queries based on some criteria [Sapia, 2000, Yang et al., 2009]. In this case, the comparison can be a simple equality test of the query representations [Sapia, 2000, Yang et al., 2009] or it can be based on separate tests of query fragments [Gupta and Mumick, 1999]. In the second case, the comparison is normally based on classical functions applied to the query representations. For instance, if the query is modeled as a vector, cosine [Agrawal et al., 2006, Akbarnejad et al., 2010, Chatzopoulou et al., 2009a, 2011], inner product [Stefanidis et al., 2009], or Hamming distance [Aouiche et al., 2006] can be used; if the query is modeled as a set, the Jaccard index [Stefanidis et al., 2009] or the Hausdorff distance [Giacometti et al., 2009] can be used. Sometimes, more sophisticated similarity functions are used. For instance, [Yao et al., 2005] uses a measure based on entropy to cluster queries modelled as strings. In [Golfarelli, 2003], similarity between OLAP queries is computed based on the relative position of the query group-by sets within the group-by lattice.

Table 5.2 summarizes the approaches reviewed in this section. Note that [Stefanidis et al., 2009] proposes two ways of comparing queries: (1) based on the frequency of the query in the log, and (2) based on the query result. Letters S, P, and C indicate the fragments used by the approach (S for selection, P for generalized projection — including the group-by set and the aggregation operator—, and C for cross-product).

We conclude this overview with some brief observations about the features a query comparison approach should have to be used for OLAP queries:

- Following requirement ♯1, we solely rely on query expressions to derive query representations. Then we exclude the approaches based on query evaluation [Drosou and Pitoura, 2011, Giacometti et al., 2009, Stefanidis et al., 2009], those depending on database instances [Agrawal et al., 2006, Baikousi et al., 2011, Chatzopoulou et al., 2009a, 2011], and those using query logs [Akbarnejad et al., 2010, Aouiche et al., 2006, Stefanidis et al., 2009].

- Our goal is not query optimization, so we drop the approaches aimed at optimization like [Ghosh et al., 2002]. In that particular work, the idea is to reuse execution plans, that heavily rely on "physical" properties (like statistics and presence of indexes); thus, query similarity is more related to how queries are evaluated than to what they mean to users. This means that two queries that should be very similar for our purposes could be found to be very dissimilar using that approach if their execution plans are different (for instance, if one has a WHERE clause and the other does not).

- According to requirement ♯2, query comparison should result in a score. So, Boolean approaches like [Gupta and Mumick, 1999] and [Yang et al., 2009] are less relevant in our context.

- OLAP queries are expressed using a friendly visual interface, and the syntax of the underlying query language (e.g., MDX) is typically transparent to users. This discourages the adoption of uninterpreted approaches like [Yao et al., 2005].

- According to requirement ♯3, the OLAP semantics is carried by a number of different components (e.g., the aggregation attribute), which encourages the adoption of a fragment-based query model like in [Sapia, 2000], also taking into account the peculiarities of the multidimensional model like in [Golfarelli, 2003].

Among the query similarity functions proposed in the OLAP area, the one that captures the above requirements at best is [Aouiche et al., 2006]. In that approach, similarity between queries q and q' is based on the number of attributes they share within their SELECT, WHERE, and GROUP-BY clauses; the normalized form we adopt here for comparison purposes (Section 5.7.1) is

$$\sigma_{AJD}(q, q') = \frac{|L \cap L'|}{|L \cup L'|}$$

where L and L' are the attributes appearing in q and q', respectively.

5.5 Query Similarity

In this section we define the similarity function used in our two-attribute approach to compare OLAP queries. As remarked in the Section 5.3, this function must consider the peculiarities of the multidimensional model, be computable based on query expressions only, and result in a score. Consistently with Definition 5.1, the function we propose is a combination of three components: one related to group-by sets, one to selection predicates, and one to measure sets.

To define group-by set similarity, we first introduce the notion of distance between attributes in a hierarchy.

Definition 5.3 (Distance between hierarchy attributes)**.** Let $\mathcal{M} = \langle A, H, M \rangle$ be a schema, $h_i \in H$ be a hierarchy, and $l, l' \in Attr(h_i)$ be two attributes. The *distance* between l and l', $Dist_{lev}(l, l')$, is the difference between the positions of l and l' within the roll-up order $\succeq_{h_i}$.

Definition 5.4 (Group-by set similarity)**.** Let q and q' be two queries, both on schema $\mathcal{M}$, with group-by sets g and g', respectively, and let $g.h_i$ ($g'.h_i$) denote the attribute of h_i included in g (g'). The *group-by set similarity* between q and q' is

$$\sigma_{gbs}(q, q') = 1 - \frac{\sum_{i=1}^{n} \frac{Dist_{lev}(g.h_i, g'.h_i)}{|Lev(h_i)|-1}}{n}$$

where n is the number of hierarchies in $\mathcal{M}$.

Our definition of selection similarity takes into account both the attributes and the constants that form the selection predicates. In particular, for each hierarchy, two identical clauses are given maximum similarity, and non-identical clauses are given decreasing similarities according to the distance between the hierarchy attributes they are expressed on.

Definition 5.5 (Distance between selection clauses)**.** Let $\mathcal{M} = \langle A, H, M \rangle$ be a schema, and c_i and c'_i be two selection clauses over hierarchy $h_i \in H$. Let $c_i.h_i \in Attr(h_i)$ denote the attribute of h_i involved in c_i (conventionally, $TRUE_i.h_i = ALL_i$). The *distance* between c_i and c'_i is

$$Dist_{clau}(c_i, c'_i) = \begin{cases} 0, \text{ if } c_i = c'_i; \\ Dist_{lev}(c_i.h_i, c'_i.h_i) + 1, \text{ otherwise} \end{cases}$$

According to this definition, the distance between two selection clauses on h_i is 0 if they are expressed on the same attribute and the same constant, 1 if they are defined on the same attribute but not on the same constant, greater than 1 if they are defined on different attributes.

Definition 5.6 (Selection similarity)**.** Let q and q' be two queries, both on schema $\mathcal{M}$, with selection predicates P and P', respectively, with $P = \{c_1, \ldots, c_n\}$ and $P' =$

TABLE 5.3: Query similarities for Example 5.2

	q_4	q_5	q_6	q_7	q_8
q_1	0.694	0.927	0.844	0.622	0.866
q_2	0.716	0.950	0.866	0.644	0.888
q_3	0.661	0.838	0.755	0.616	0.833

$\{c'_1, \dots, c'_n\}$. The *selection similarity* between q and q' is

$$\sigma_{sel}(q, q') = 1 - \frac{\sum_{i=1}^{n} \frac{Dist_{clau}(c_i, c'_i)}{|Lev(h_i)|}}{n}$$

Finally, to define the measure similarity, we use the Jaccard index.

Definition 5.7 (Measure similarity)**.** Let q and q' be two queries, both on schema $\mathcal{M}$, with measure sets $Meas$ and $Meas'$, respectively. The *measure similarity* between q and q' is

$$\sigma_{meas}(q, q') = \frac{|Meas \cap Meas'|}{|Meas \cup Meas'|}$$

We can now define the similarity between two OLAP queries as the weighted average of the three similarity components defined above.

Definition 5.8 (Similarity of OLAP queries)**.** Let q and q' be two queries, both on schema $\mathcal{M}$. The *similarity* between q and q' is

$$\sigma_{que}(q, q') = \alpha \cdot \sigma_{gbs}(q, q') + \beta \cdot \sigma_{sel}(q, q') + \gamma \cdot \sigma_{meas}(q, q')$$

where α, β, and γ are normalized to 1.

Example 5.2. *The similarity between queries q_1 and q_4 of Example 5.1 is computed as follows:*

$$\begin{aligned}
\sigma_{gbs}(q_1, q_4) =& 1 - \frac{(0/3 + 1/3 + 0/1 + 0/1 + 0/1)}{5} = 0.933 \\
\sigma_{sel}(q_1, q_4) =& 1 - \frac{(0/4 + 3/4 + 2/2 + 0/2 + 0/2)}{5} = 0.650 \\
\sigma_{meas}(q_1, q_4) =& \frac{1}{2} = 0.500 \\
\sigma_{que}(q_1, q_4) =& 0.694
\end{aligned}$$

(assuming for simplicity $\alpha = \beta = \gamma = 0.333$). The overall query similarities for sessions s and s' are summarized in Table 5.3. □

5.6 Session Similarity

5.6.1 Edit-Based Session Similarity

The Levenshtein distance compares two strings in terms of the cost of the atomic operations (typically insertion, deletion, and substitution of a character) necessary to transform one string into another [Ristad and Yianilos, 1998]. Given two strings s and s' of v and v' characters, respectively, a $(v+1) \times (v'+1)$ distance matrix D of reals is recursively defined in terms of the deletion, insertion, and substitution costs; the Levenshtein distance between s and s' is found in the bottom-right cell of D, that represents the minimum sum of the operation costs to transform s in s'.

In the traditional formulation, an operation is applied in absence of a perfect match (i.e., of an identity) between the compared characters. In our case this is too restrictive, because OLAP queries are complex objects whose match is not effectively captured by identity (see requirement ♯2). So we consider two queries as matching when their similarity is above a given threshold θ, and we apply a transformation operation when the similarity is under θ. Besides, we normalize distances using the length of the longest of the two sessions involved, so that the cost of a single mismatch is lower for longer sessions.

Definition 5.9 (Edit-Based Similarity of OLAP Sessions)**.** Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively. Given a matching threshold θ, the distance matrix for s and s' is a $(v+1) \times (v'+1)$ matrix D_θ of reals recursively defined as follows:

$$D_\theta(i,j) = \begin{cases} 0,\ when\ i=0\ or\ j=0 \\ D_\theta(i-1,j-1),\ when\ i,j>0\ and\ \sigma_{que}(s_i,s'_j) \geq \theta \\ min \left\{ \begin{array}{c} D_\theta(i-1,j)+1; \\ D_\theta(i,j-1)+1; \\ D_\theta(i-1,j-1)+1 \end{array} \right\},\ when\ i,j>0\ and\ \sigma_{que}(s_i,s'_j) < \theta \end{cases}$$

where s_i is the *i-th* query of session s. The *edit-based similarity* between s and s' is:

$$\sigma_{edit}(s,s') = 1 - \frac{D_\theta(v,v')}{max\{v,v'\}}$$

Note that, like in most applications of the Levenshtein distance, all transformation costs are set to 1.[4] As to complexity of this function, in the general case it is $O(v \cdot v')$ where v and v' are the lengths of the two sessions [Wagner and Fischer, 1974].

[4]In the formula, the three rows of the *min* argument deal with deletions, insertions, and substitutions, respectively.

Example 5.3. *With reference to Example 5.1 and using $\theta = 0.7$, the minimum cost to transform s' to s is obtained by matching queries as follows: $\langle q_1, q_5\rangle$, $\langle q_2, q_6\rangle$, $\langle q_3, q_8\rangle$ and deleting q_4 and q_7. Thus, it is $\sigma_{edit}(s, s') = 1 - \frac{2}{5} = 0.60$.* □

5.6.2 Subsequence-Based Session Similarity

An n-gram is a substring of size n of a given string [Brown et al., 1992]. A popular string similarity function based on n-grams is the *Dice coefficient*, an extension of the Jaccard index defined as twice the number of shared n-grams over the total number of n-grams in the two strings.

In the OLAP context, the concept of "shared" n-grams becomes that of "similar" n-grams. Two n-grams r and r' are similar if their queries are pairwise similar, i.e., if their similarity is above threshold θ. To ensure symmetry while being consistent with the original definition, in our two-attribute extension similarity is defined as follows.

Definition 5.10 (Subsequence-Based Similarity of OLAP Sessions)**.** Let s and s' be two OLAP sessions on schema $\mathcal{M}$, and $n \geq 1$. Given a matching threshold θ, the *subsequence-based similarity* between s and s' is

$$\sigma_{sub}(s, s') = \frac{2 \times min\{|SNgram_\theta(s, s')|, |SNgram_\theta(s', s)|\}}{|Ngram(s)| + |Ngram(s')|}$$

where $Ngram(s)$ is the set of n-grams of s and $SNgram_\theta(s, s') \subseteq Ngram(s)$ is the set of n-grams of s that have a similar n-gram in s':

$$SNgram_\theta(s, s') = \{r \in Ngram(s) | \exists r' \in Ngram(s'), \sigma_{que}(r_i, r'_i) \geq \theta \ \forall i = 1, \ldots, n\}$$

The complexity of this function is that of finding the n-grams of the two sessions, which is $O(v)$ (where v is the length of the longest one), plus that of computing the sets $SNgram_\theta(s, s')$, which is $O((v - n)^2)$.

Example 5.4. *Applying the above definition to Example 5.1, with n=1, we obtain $\sigma_{sub}(s, s') = \frac{2 \times min\{1,2\}}{1+2} = 0.67$.* □

5.6.3 Log-Based Session Similarity

In the tf-idf approach, the similarity between two sets of tokens (in information retrieval applications, tokens are lemmas and sets of tokens are documents) depends on both the frequency of each token in the sets and its frequency in a corpus. In our context, this approach can be adopted if the OLAP sessions to be compared are taken from a log, to penalize the non-distinctive queries (i.e., those that are more frequent in the log) when assessing similarity.

To propose an extension of the tf-idf method we start by applying the definition of *soft tf-idf* given by [Moreau et al., 2008]:

$$Sim_{soft}(s,s') = \sum_{s_i \in Close_\theta(s,s')} T(s_i,s) \cdot T(s'_{j_i},s') \cdot \sigma_{que}(s_i,s'_{j_i})$$

where θ is a threshold,

$$\begin{aligned} Close_\theta(s,s') &= \{s_i \in s | \exists s'_j \in s', \sigma_{que}(s_i,s'_j) > \theta\}, \\ T(s_i,s) &= \frac{tfidf(s_i,s)}{\sqrt{\sum_{s_k} tfidf(s_k,s)^2}}, \\ tfidf(s_i,s) &= tf(s_i,s) \cdot idf(s_i,s) = \frac{n_{s_i,s}}{|s|} \cdot log\frac{|L|}{|\{s \in L | s_i \in s\}|}, \\ s'_{j_i} &= argmax_{s'_j \in s'}\{\sigma_{que}(s_i,s'_j)\}, \end{aligned}$$

$n_{s_i,s}$ is the number of times s_i appears in s, and L is the set of OLAP sessions in the log. Intuitively, $Close_\theta(s,s')$ is the set of queries in sessions s that have some similarity to a query in session s'; $tfidf(s_i,s)$ is directly proportional to the frequency of query s_i in session s and inversely proportional to the frequency of s_i in the log L ($tfidf(s_i,s) = 0$ when all session in L include s_i); $T(s_i,s)$ is a normalized form of $tfidf(s_i,s)$; s'_{j_i} is the query in s' that is most similar to s_i.

This definition cannot be immediately used in our case for the following reasons:

1. It uses the "crisp" definition of tf-idf in the definition of T whereas in our case, given that it is unlikely to find the same query twice in an OLAP log, a "soft" version (i.e., one based on query similarity) should be used instead.
2. The soft tf-idf is not symmetric, which is not desirable for a similarity function.
3. There may be more than one query s'_{j_i} in s' that maximizes σ_{que} with s_i, which may not be relevant in the context of named entity matching [Moreau et al., 2008], but is definitely relevant in the OLAP context.
4. As pointed out by [Moreau et al., 2008], there is a problem with counting that makes the similarity not normalized.

To cope with the first issue, we inject the similarity σ_{que} in the definition of tf-idf. By replacing equality with similarity, a two-attribute tf-idf can be computed as:

$$tfidf_2(s_i,s) = \frac{|Close_\theta(s_i,s)|}{\sum_{s_k \in Q} |Close_\theta(s_k,s)|} \cdot log\frac{|L|}{|\{s \in L | Close_\theta(s_i,s) \neq \emptyset\}|}$$

where Q is the set of all queries in L and $Close_\theta(s_i,s)$ is the set of queries of s that are similar to s_i.

Symmetry can be achieved by modifying the definition of similarity to work on pairs of queries, each relating a query in one session with one of its closest queries in the other session. This set of pairs is defined by:

$$R_\theta(s,s') = \{\langle s_i, s'_k\rangle | s_i \in s, s'_k \in Closest_\theta(s_i, s')\} \cup \{\langle s_l, s'_j\rangle | s'_j \in s', s_l \in Closest_\theta(s'_j, s)\}$$

where $Closest_\theta(s_i, s)$ is the set of queries of s that have maximum similarity with s_i. Note that a query in a session appears more than once in $R_\theta(s,s')$ if there is more than one query in the other session with maximum similarity. This solves the third issue.

Finally, to cope with the fourth issue, the similarity is computed as the cosine of the two vectors obtained by taking the $tfidf_2$ of all the first (respectively, second) queries of the pairs.

Definition 5.11 (Log-Based Similarity of OLAP Sessions)**.** Let s and s' be two OLAP sessions on schema $\mathcal{M}$. The *log-based similarity* between s and s' is

$$\sigma_{log}(s,s') = \sum_{\langle s_i, s'_j\rangle \in R_\theta(s,s')} T_2(s_i, s, s') \times T_2(s'_j, s', s) \times \sigma_{que}(s_i, s'_j)$$

where

$$T_2(s_i, s, s') = \frac{tfidf_2(s_i, s)}{\sqrt{\sum_{\langle s_i, s'_j\rangle \in R_\theta(s,s')} tfidf_2(s_i,s)^2 + \sum_{Closest_\theta(s_i,s')=\emptyset} tfidf_2(s_i,s)^2}}$$

$$T_2(s'_j, s', s) = \frac{tfidf_2(s'_j, s')}{\sqrt{\sum_{\langle s_i, s'_j\rangle \in R_\theta(s,s')} tfidf_2(s'_j,s')^2 + \sum_{Closest_\theta(s'_j,s)=\emptyset} tfidf_2(s'_j,s')^2}}$$

The complexity of this function should obviously be expressed not only in terms of the sessions to be compared but also in terms of the size of the log; it turns out that the complexity of computing $R_\theta(s,s')$ is $O(v^2)$, while that for computing all the $tfidf_2$ terms it is $O(v \times |Q|)$ where v the length of the longest session in the log.

Note that, as any cosine similarity, σ_{log} can be easily turned into the angle distance $arcos(\sigma_{log})$, which is a metric [Bustos and Skopal, 2011].

Example 5.5. *With reference to Example 5.1, we focus on computing the log-based similarity between s and s'. The set of query pairs used in the computation of $\sigma_{log}(s,s')$ is $R_{0.7}(s,s') = \{\langle q_1,q_5\rangle, \langle q_2,q_5\rangle, \langle q_3,q_5\rangle, \langle q_2,q_4\rangle, \langle q_2,q_6\rangle, \langle q_2,q_8\rangle\}$; the two components* *of the $tfidf_2$ weights for each of these queries are as follows:*

$$
\begin{aligned}
tf_2(q_1,s) =&0.333\ , \quad idf_2(q_1,s) =0.176\\
tf_2(q_2,s) =&0.333\ , \quad idf_2(q_2,s) =0.176\\
tf_2(q_3,s) =&0.333\ , \quad idf_2(q_3,s) =0.000\\
tf_2(q_4,s') =&0.117, \quad idf_2(q_4,s') =0.176\\
tf_2(q_5,s') =&0.235, \quad idf_2(q_5,s') =0.176\\
tf_2(q_6,s') =&0.235, \quad idf_2(q_6,s') =0.176\\
tf_2(q_8,s') =&0.235, \quad idf_2(q_8,s') =0.000
\end{aligned}
$$

Note that, though q_3 and q_8 are similar (same group-by set, same selection predicate, and nearly the same set of measures) and should positively contribute to the similarity of s and s', they do not actually enter in the computation of $\sigma_{log}(s,s')$. Indeed, queries similar to q_3 and q_8 can be found in each session of the log, making their idf weight 0. By applying Definition 5.11 we get $\sigma_{log}(s,s') = 0.479$, while $\sigma_{log}(s,s'') = \sigma_{log}(s',s'') =$ 0. □

5.6.4 Alignment-Based Session Similarity

As emerged in Section 5.4, a comparison of OLAP sessions should support subsequence alignment, keep query ordering into account, and allow gaps in the matching subsequences. The Smith-Waterman algorithm mentioned in Section 5.4 has all these features. It relies on a distinction between *matching* elements (whose similarity is positive) and *mismatching* elements (whose similarity is negative), and is based on a matrix whose cells show the score for aligning two sequences starting from a specific couple of elements. Each score is the result of a trade-off between the cost for introducing a gap in the matching subsequences and the cost for including a mismatching pair of elements.

Unfortunately, none of the implementations available in the literature can be directly applied here for different reasons:

- The algorithm was originally aimed at molecular comparison, so sequence elements were taken from a set that is known a priori (the set of all amino acids). This allows matching and mismatching pairs to be enumerated and a similarity score to be assigned in advance to each possible couple of elements. In the OLAP context matching elements are queries, and the domain of the possible OLAP queries is huge (requirement ♯2); besides, the similarity between two queries is always positive, so separating matching and mismatching queries requires the adoption of a threshold.

- For the same reason mentioned above, in all previous implementations the cost for introducing a gap could be assigned in advance to each possible couple of elements. Conversely, in our case it must be determined at runtime based on the two specific sessions being compared (requirement ♯8).
- In all previous implementations all matchings were considered to be equally important, while in OLAP sessions a matching between recent queries should be given more relevance (requirement ♯6).

To address all these issues, we propose an extension of the Smith-Waterman algorithm that relies on the matrix defined below. The value in position (i, j) of this matrix is a score that expresses how "well" two sessions s and s' match when they are aligned ending in queries s_i and s'_j. Intuitively, each score is recursively calculated by progressively adding the similarities between all pairs of matching queries in the two sessions. Threshold θ is used to distinguish matches from mismatches; a *time-discounting* function $\rho(i, j)$ is used to promote alignments based on recent queries; finally, a *gap penalty* δ is used to discourage discontinuous alignments.

Definition 5.12 (OLAP Session Alignment Matrix)**.** Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively. Given a matching threshold θ, the *(OLAP session) alignment matrix* for s and s' is a $(v + 1) \times (v' + 1)$ matrix A of reals recursively defined as follows:

$$A(i,j) = \begin{cases} 0, \text{ when } i = 0 \text{ or } j = 0 \\ max \left\{ \begin{array}{c} 0; \\ A(i-1, j-1) + (\sigma_{que}(s_i, s'_j) - \theta) \cdot \rho(v-i, v'-j); \\ max_{1 \leq k < i}\{A(k,j) - \delta \cdot (i-k)\}; \\ max_{1 \leq k < j}\{A(i,k) - \delta \cdot (j-k)\} \end{array} \right\}, \text{ else} \end{cases}$$

where δ is the average similarity between all couples of queries in s and s' whose similarity is above θ:

$$\delta = avg_{(i,j):\sigma_{que}(s_i, s'_j) \geq \theta}\{\sigma_{que}(s_i, s'_j)\} ,$$

ρ is a two-dimensional logistic sigmoid function:

$$\rho(i,j) = 1 - \frac{1 - \rho_{min}}{1 + e^{slope-i-j}} ,$$

ρ_{min} is the minimal value assumed by ρ (i.e., the maximum time discount), and *slope* rules the position where the slope is steepest (Figure 5.2).

Some observations on the above definition:

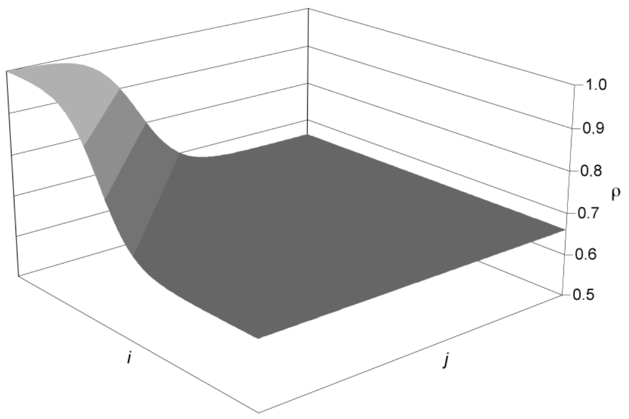


FIGURE 5.2: The time-discounting function $\rho(i, j)$ with $\rho_{min} = 0.66$ and $slope = 4$

- The use of the term $\sigma_{que}(s_i, s'_j) - \theta$ implies that query pairs whose similarity is above (below) θ are considered as matches (mismatches). Although a "sharp" threshold is used, the score of a matching pair and the cost of a mismatching pair turn out to be proportional to the distance of that pair similarity from θ.
- The definition given of the gap penalty δ is such that it guarantees a gap penalty to be payed if it enables a good match (i.e. a match higher than the average). Note that a penalty only related to the threshold could lead to underestimating or overestimating the impact of a gap on the overall similarity.
- The time-discounting function ρ leads match and mismatch scores to decay when moving backwards along the two sessions; it is maximum and equal to 1 for the ending queries of the two sessions.

The optimal alignment between s and s' is determined by the highest value in A, $\overline{A}$, that we call *alignment score*. The positions $\overline{i}$ and $\overline{j}$ such that $A(\overline{i}, \overline{j}) = \overline{A}$ mark the end of the matching subsequences of s and s'.

The alignment score is not really a similarity value, since it is not limited in the interval [0..1]. This creates problems when comparing sessions with difference length. Then we define OLAP session similarity by normalizing the alignment score:

Definition 5.13 (Alignment-Based Similarity of OLAP Sessions)**.** Let s and s' be two OLAP sessions on schema $\mathcal{M}$, of lengths v and v' respectively (with $v \leq v'$), and let $\overline{A}$ be the alignment score for s and s'. The *alignment-based similarity* between s and s' is

$$\sigma_{ali}(s, s') = \frac{\overline{A}}{(1-\theta)\sum_{k=1}^{v} \rho(k, k)}$$

where the normalizing factor is the alignment score for two identical sessions of length v.

TABLE 5.4: Threshold-filtered and discounted query similarities, $(\sigma_{que}(s_i, s'_j) - \theta) \cdot \rho(v - i, v' - j)$, for Example 5.6

	q_4	q_5	q_6	q_7	q_8
q_1	-0.004	0.171	0.120	-0.071	0.160
q_2	0.013	0.208	0.151	-0.053	0.186
q_3	-0.032	0.126	0.053	-0.082	0.132

TABLE 5.5: OLAP session alignment matrix for Example 5.6

	q_4	q_5	q_6	q_7	q_8
q_1	0.000	**0.171**	0.120	0.000	0.160
q_2	0.013	0.208	**0.322**	*0.191*	0.186
q_3	0.000	0.139	0.261	0.241	**0.323**

Like for edit-based similarity, the complexity of this function is known to be $O(v \cdot v')$ where v and v' are the lengths of the two sessions [Li and Durbin, 2010].

Example 5.6. *Again we focus on comparing s and s' of Example 5.1. Table 5.4 reports the results obtained by filtering query similarities with $\theta = 0.7$ and applying the time-discounting function ρ as shown in Definition 5.12. Note that a negative value represents a mismatch, and a positive one a match. Table 5.5 shows the OLAP session alignment matrix for s and s'; the cells in bold denote alignments between two queries (e.g., q_1 is aligned with q_5), those in italics refer to gaps. Alignments on recent queries are favored, so q_3 is aligned with q_8. Query q_4 is not involved in the alignment due to the low similarity it has with the other queries in s. In q_7, a gap penalty is paid to gain the good match between q_3 and q_8. The overall similarity between s and s' is 0.323 (the highest value in the matrix). After normalization, we obtain $\sigma_{ali}(s, s')$=0.387.* □

The properties of the proposed similarity function can be evaluated in terms of the distance function it induces using the standard transformation $\sigma_{ali} = 1/(1 + Dist_{ali})$. As stated by [Bustos and Skopal, 2011] for the original Smith-Waterman approach, $Dist_{ali}$ is not a metric because, while it is non-negative and symmetrical, it is not reflexive and it does not satisfy the triangular inequality as shown in Example 5.7. In particular, the triangular inequality cannot be satisfied because this approach is based on a local alignment.

Example 5.7. *Let $s = \langle q_1, q_2 \rangle$, $s' = \langle q_1, q_2, q_3, q_4 \rangle$, and $s'' = \langle q_3, q_4 \rangle$ be three sequences, where $\sigma_{que}(q_i, q_j) = 0$ if $i \neq j$. It is*

$$Dist_{ali}(s, s'') = \infty \,,\ Dist_{ali}(s, s') = Dist_{ali}(s', s'') = 0$$

which obviously contradicts the triangle inequality axiom. Besides, s' has zero distance from both s and s'' though $s \neq s' \neq s''$. □

5.7 Validation

This section discusses the outcomes of the tests we run to answer three main questions: *Do the proposed solutions properly capture the idea of similarity as perceived by the users? Do they adequately express the similarity criteria proposed in Section 5.3? What are their discriminant capabilities?* While the first question will be answered in Subsection 5.7.1, the remaining two questions will be discussed in Subsection 5.7.2.

5.7.1 User Tests

As stated in Section 5.3, we submitted a questionnaire to 41 persons with different OLAP skills. The results have been used in the first stages of this work to understand how OLAP session similarity is perceived by users, and they will be used here to verify if the proposed methods capture the users' perception of similarity. To enable a better interpretation of the results, for each questionnaire test we show the *consensus* ϕ, i.e., the degree of agreement among raters, defined as the percentage of users who gave the majority judgement.

The first four tests of the questionnaire were focused on OLAP query comparison. In each test the users were asked to rate the similarity between a given query q_c and three other queries $\{q_1, q_2, q_3\}$ in both absolute (using four scores: *low*, *fair*, *good*, and *high*) and relative terms (i.e., by ranking queries in order of similarity). All queries were focused on the complete CENSUS schema (including 5 hierarchies and 6 measures); they were basic OLAP queries as of Definition 5.1 and were presented in a graphical way. We used the results obtained in two ways: (i) to compare σ_{que} with function σ_{AJD} mentioned in Section 5.5 in terms of compliance with the users' judgments; and (ii) to set the weights of the three components of our query similarity function σ_{que}.

As to (i), we defined two matching factors as follows:

- The *score matching factor SM* for σ is the percentage of times the score given by a user is the same returned by σ. To compute it, we first discretized the values returned by σ in ranges corresponding to *low*, *fair*, *good*, and *high*.
- The *rank matching factor RM* for σ is the percentage of cases in which the rankings σ provides match with those given by users (e.g., q_c was judged to be more similar to q_i than to q_j, and $\sigma(q_c, q_i) > \sigma(q_c, q_j)$).

As to (ii), we tuned the weights through an optimization process whose goal function was the maximization of the correspondence with the questionnaire results. To avoid overfitting we used a ten folds cross-validation approach. The ranges for the weights were chosen consistently with requirement ♯4 in Section 5.3: $\alpha \in [0.2, 0.5]$,

Table 5.6: Consensus and matching factors for OLAP query comparison user tests

	Consensus		σ_{AJD}		σ_{que}	
	ϕ_{score}	ϕ_{rank}	SM	RM	SM	RM
Test 1	70%	94%	70%	94%	70%	94%
Test 2	56%	70%	56%	56%	56%	70%
Test 3	41%	64%	34%	57%	41%	64%
Test 4	73%	93%	49%	93%	59%	93%

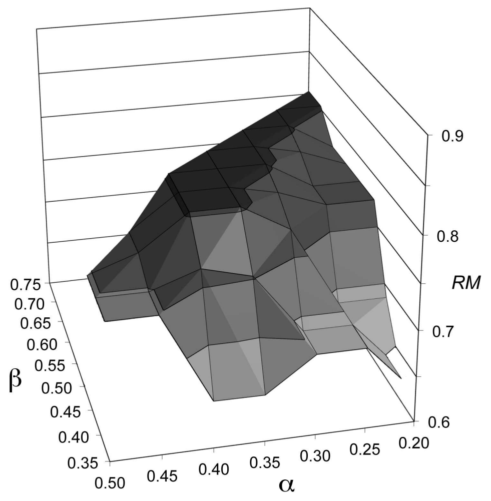


Figure 5.3: Questionnaire matching for σ_{que} as a function of weights α and β

$\beta \in [0.35, 0.75]$, $\gamma \in [0.05, 0.45]$. The function to be optimized was the average value of RM for σ_{que} in Tests 1 to 4, that measures the percentage of cases in which the rankings provided by σ_{que} match with those given by users. Figure 5.3 shows the average RM as a function of α and β (γ is set so that they sum up to 1). The optimal weights turned out to be $\alpha = 0.35$, $\beta = 0.5$, and $\gamma = 0.15$ ($\beta > \alpha$, consistently with requirement $\sharp 4$); noticeably, RM smoothly decreases for increasing distances from these optimal values, which proves that the setting is robust.

The comparison results are reported in Table 5.6. For all the tests, σ_{que} matches the users' judgement at least like σ_{AJD} thanks to its fine-grained definition. In particular, σ_{que} returns the same answers given by the majority of the users (i.e. the highest possible values for SM and RM) in Tests 1, 2, and 3, while σ_{AJD} returns the same answers only in Test 1. Note that σ_{AJD} falls short both when there is high user consensus (Test 4) and when user consensus is low because queries are very similar to each other (Tests 2 and 3). Overall, these results confirm a strong correlation between the query similarity computed through σ_{que} and the one perceived by users. Since σ_{que} is more sensitive than σ_{AJD} and it shows better results, in the remaining tests we will focus on the former.

The second part of the questionnaire included five more tests focused on OLAP session comparison. In each test, the users were asked to evaluate the similarity of a given session s_c against three candidate sessions $\{s_1, s_2, s_3\}$ in absolute and relative terms.

TABLE 5.7: Consensus and matching factors for OLAP session comparison user tests

	Consensus		σ_{edit}		σ_{sub}		σ_{log}		σ_{ali}	
	ϕ_{score}	ϕ_{rank}	SM	RM	SM	RM	SM	RM	SM	RM
Test 1	51%	75%	51%	-	29%	-	51%	75%	51%	71%
Test 2	43%	70%	33%	-	9%	-	39%	70%	43%	70%
Test 3	51%	64%	41%	-	4%	-	51%	46%	51%	46%
Test 4	36%	80%	19%	-	26%	-	35%	65%	35%	65%
Test 5	38%	78%	33%	-	13%	-	33%	70%	33%	70%

Sessions were graphically presented to users as sequences of queries, emphasizing the OLAP operator used to move from one query to the next one. The results are summarized in Table 5.7 for the four functions described in Section 5.6, by applying SM and RM to sequences rather than to single queries. Note that the edit-based and the subsequence-based approaches, that do not directly incorporate the σ_{que} score in their definitions, are not sensitive enough to rank the sessions proposed in our tests. In fact, they return the same similarity for most sessions involved in each test, so their RM cannot be determined. This also penalizes SM, that is significantly low.

Conversely, both the log-based and the alignment-based approaches perform very well and the scores returned are, in most cases, those of the majority of users (i.e., $SM = \phi_{score}$ and/or $RM = \phi_{rank}$, that is the maximum attainable). The errors always involve sequences that are quite similar, making the comparison more subjective. Note that the absolute consensus is always much lower than the relative one; this can be explained considering that scoring entails a 4-valued choice, while ranking only requires choosing between two alternatives (s_c is either more similar to s_i than s_j or not), thus making inter-user agreement more likely. Some more detailed comments for single tests of log-based and alignment-based approaches follow:

- In test 1, candidate sessions differ in the length of the match. s_1 and s_2 are very similar to each other and determine a long match with s_c, while s_3 is quite different from the others. While the log-based approach returns the same results as the majority of users, the alignment-based approach returns an inverted ranking between s_1 and s_2, which is a minor issue due to their strong similarity.
- In test 2, candidate sessions differ in the position of the match. The log-based approach returns a score that is slightly different from the one of the majority group since it does not give different relevance to matches of recent and old queries.
- In test 3, all three candidate sessions are quite similar to each other and to s_c, leading to a difficult ranking operation for both functions.
- In test 4, each candidate session differs from the reference only for one of the components of its queries (group-by set, predicates, and measures). Both approaches agree with the users majority in indicating the session that differs in

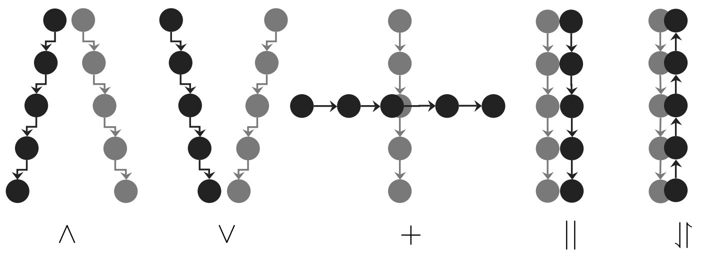


FIGURE 5.4: The templates used to generate sessions. Overlapping circles represent identical queries, near circles represent similar queries. For template ||, the queries are pairwise separated by one atomic OLAP operation

their selection predicates as the less similar to the reference session. However, both approaches return an inverted ranking between the sessions that differ in their group-by sets and in their predicates, respectively. This is probably due to the weight we use for measure similarity, $\gamma = 0.15$, that in this particular case is not low enough to counterbalance the relevant difference on measure sets.

- In test 5, session s_1 is very similar to s_c; s_2 and s_3 are similar to each other and quite different from s_c. Both approaches agree with the users majority in indicating s_1 as the most similar to s_c, but they disagree in ranking the other two sessions. This is actually not surprising in light of the low relative consensus ($\phi_{rank}(s_2, s_3) = 61\%$).

5.7.2 Objective Tests

In this subsection we compare the four functions described in Section 5.6; for subsequence-based similarity we use 3-grams (empirically tested for best results). All tests were conducted on a 64-bits Intel Xeon quad-core 3GHz, with 8GB RAM, running Windows 7 pro SP1; the similarity threshold was tuned to $\theta = 0.8$ to achieve the best results.

Our benchmark includes a set of synthetic sessions over the CENSUS schema, generated based on Definition 5.2 with our own log generator developed in Java. A session is generated starting from an initial query and a final query, both obtained by randomly choosing a group-by set, a selection predicate, and a subset of measures. Intermediate queries are then generated by applying, one at a time in a random order, the minimal atomic OLAP operations that transform the initial query into the final one. The atomic OLAP operations considered are: change attribute along one hierarchy in the group-by set, add or remove a clause from the selection predicate, change the constant appearing in a selection clause, and add or remove a measure.

To generate logs we considered the five templates depicted in Figure 5.4, that model intuitive notions of what similar sessions might look like:

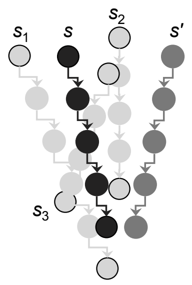


FIGURE 5.5: The seed session s (in black), its mate s' according to template $\wedge$ (in dark gray), and three random sessions (in light gray). The first and last queries of sessions are circled.

- In template $\wedge$, the two sessions have similar starting queries then they diverge to radically different queries.
- In template $\vee$, the two sessions have radically different starting queries then they converge to similar ending queries.
- In template $+$, the two sessions converge to the same query then they diverge.
- In template $||$, the second session is constructed by "shifting" all queries in the first session by one OLAP operation.
- In template $\downharpoonleft\upharpoonleft$, the two sessions have the same queries in reverse order.

In light of the requirements expressed in Section 5.3, some of these templates should yield higher similarities. In particular, we want template $\vee$ to yield higher similarities than $\wedge$ due to requirement $\sharp$6. For requirement $\sharp$7, we also expect $||$ to yield higher similarities than $\vee$, $\wedge$, and $+$. As to $\downharpoonleft\upharpoonleft$, requirement $\sharp$5 imposes that it yields low similarities.

The first test assesses the capabilities of the similarity functions. In this test, for each template we generated a log as follows (see also Figure 5.5 for an example):

1. Generate a pair of sessions, s and s', that respect the template.
2. Generate 5 more sessions $s_1, \ldots s_5$ using s as a seed. The first and the last query of s_i are obtained by applying three random atomic OLAP operations to the first and the last query of s, respectively; then, the intermediate queries of s_i are generated as described above.
3. Repeat the two previous steps 5 times.

This means generating overall 5 logs, each including 35 sessions. Then, for each log and each similarity function, we computed the ratio τ of the average similarity σ_t between the two sessions respecting the template and the average similarity σ_r between each

Table 5.8: Ratio τ for template-based OLAP session comparison objective tests

Log	σ_{edit}	σ_{sub}	σ_{log}	σ_{ali}
$\wedge$	1.39	1.16	1.39	2.32
$\vee$	1.46	1.52	1.31	3.21
$+$	1.44	1.23	1.32	2.15
$\|\|$	1.79	1.57	1.51	5.23
$\downharpoonleft\upharpoonright$	1.08	1.57	1.42	0.78
average	1.40	1.35	1.35	2.51

Table 5.9: Ratio τ for increasing distances in the $||$ template

$\|\|$ dist	σ_{edit}	σ_{sub}	σ_{log}	σ_{ali}
1	1.79	1.57	1.51	5.23
2	1.91	1.55	1.51	3.78
3	1.86	1.56	1.45	3.48
4	1.81	1.52	1.42	2.80
5	1.81	1.52	1.55	2.68

seed and the 5 sessions generated from it; the higher τ, the better the function can distinguish a template from the background. Table 5.8 reports the results. Noticeably, the alignment-based approach largely outperforms the others; besides yielding an average τ that is almost twice that of the other approaches, it meets the expectations as to template similarities. Template $||$ is correctly recognized as the one with highest similarity; $\vee$ clearly yields higher similarities than $\wedge$, while $\downharpoonleft\upharpoonright$ yields low similarities since it does not fulfill requirement $\sharp$5 about query ordering. The only other function that captures requirement $\sharp$5 is σ_{edit}. Noticeably, though all the other functions return an average ratio τ higher than 1, they are not sensitive enough to distinguish and rank the different templates.

The purpose of the second objective test is to discover how sensitive each function is to the distance between the two sessions that form template $||$; to this end, the number of atomic OLAP operations that separate these two sessions is varied from 1 to 5 (using the same log-generation algorithm explained for the first test). Even in this test σ_{ali} turns out to be more effective than the other functions. Indeed, as shown in Table 5.9, the ratio τ for σ_{ali} progressively decreases for increasing distances, while for the other functions it is almost constant. This is because σ_{ali} is sensitive to the specific values of similarity between each couple of queries, while for the other functions each couple of queries either match or do not match.

The next test measures the time for computing each similarity function. For this test we generated a log, randomly chose one session s, and compared all prefixes of s with 10 other sessions randomly chosen from the log. Note that, for log-based similarity, we disregard the time for building the frequency matrix used in the computation of all the idf's. We report the results for a minimum prefix of 1 query and a maximum prefix of

13 queries. As expected, the subsequence-based approach is the most efficient (from 0.4 ms to 3.6 ms for a single comparison), followed by the alignment-based approach (from 1.1 ms to 7.1 ms) and by the edit-based approach (from 1.3 ms to 8.3 ms). Log-based similarity is the less efficient (from 30.4 to 75.1 ms).

We close this subsection with a final remark related to efficiency. OLAP sessions are inherently interactive; to understand to what extent our approach can realistically be adopted to compare sessions at user-time, we made two tests using the same protocol adopted for the test above:

- We measured how many comparisons can be made for each similarity function during 100 ms, which is usually considered to be the maximum interactive response time [Khoussainova et al., 2010b]. The number of comparisons ranges from 109 for subsequence-based similarity to 3 for log-based similarity, with alignment-based and edit-based similarity scoring 32 and 31 comparisons, respectively.
- We measured how many comparisons can be made during the average time it takes to evaluate a query. To this end we randomly chose a session in the log and computed the average execution time for its queries, expressed in MDX; we used real data extracted from the IPUMS database [Minnesota Population Center, 2008], corresponding to about 500,000 facts stored on Oracle 11g. The average query execution time turned out to be 553.46 ms, which corresponds to 607 comparisons for subsequence-based similarity, 177 and 175 comparisons for alignment-based and edit-based similarity respectively, and 18 comparisons for log-based similarity.

5.8 Conclusions

In this chapter we investigated different approaches for defining a similarity function to compare OLAP sessions, based on the requirements deduced from a user study conducted with practitioners and researchers [Aligon et al., 2013]. We considered and compared two functions for OLAP query similarity and four functions for OLAP session similarity; in particular, the latter were obtained by extending popular approaches for string comparison. Overall, the experimental results we obtained show that the alignment-based approach (an extension of the Smith-Waterman algorithm, coupled with a three-component query similarity function) is the one that best matches the users' judgements. It is also the one that clearly gives best results on a synthetic benchmark in terms of sensitivity and capability of correctly ranking different templates of session similarity. Finally, from the point of view of efficiency, the time required for comparing two sessions is perfectly compatible with complex applications. As to future works, we propose to exploit the result of the similarity comparison between the current user session and the past ones to recommend the next query to formulate.

Chapter 6

Agile Data Warehouse Design

In this chapter, we support BI ANYTIME by proposing a new methodology, 4WD, that combines agile principles with DW peculiarities to accelerate the DW development. We prove the effectiveness of our methodology with a case study on a pay-tvs project.

6.1 Introduction

DW systems are characterized by a long and expensive development process that hardly meets the ambitious requirements of today's market. This is one of the main causes behind the low penetration of DW systems in small-medium firms, and even behind the failure of whole projects [Ramamurthy et al., 2008].

As a matter of fact, DW projects often leave both customers and developers dissatisfied. The main reasons for low customers' satisfaction are the long delay in delivering a working system and the large number of missing or inadequate (functional and non-functional) requirements. As to developers, they complain that —mainly due to uncertain requirements— it is overly difficult to accurately predict the resources to be allocated to DW projects, which leads to gross errors in estimating design times and costs.

In the light of the above, we believe that the methodological issues related to DW design deserve some further investigation aimed at improving the development process from different points of view, such as efficiency and predictability.

The available literature on DW design mainly focuses on traditional, linear approaches such as the *waterfall approach*, and it appears to be only loosely related to the sophisticated design methodologies that have been emerging in the software engineering community. Though some works about agile data warehousing have appeared [Hughes, 2008], there are also evidences that applying an agile approach *tout court* to DW design has several risks, such as that of inappropriately narrowing the DW scope [Beyer and Richardson, 2010].

In this chapter, we analyze the potential advantages arising from the application of modern software engineering methodologies to a DW project and we propose 4WD, a design methodology that aims at coupling the main principles emerging from these methodologies to the peculiarities of DW projects [Golfarelli et al., 2011c]. The chapter outline is as follows:

- In Section 6.3, we better explain the motivation of 4WD, starting from the problems of the existing methodologies to reach the goals of a better and innovative DW development approach.
- In Section 6.4, we list the main features of 4WD, explaining how these characteristics may address the aforementioned goals.
- In Section 6.5, we propose a case study on a pay-tvs project to validate our methodology.

6.2 Related Works

DW design has been investigated by the research community since the late nineties. A classic waterfall approach was first proposed in [Golfarelli and Rizzi, 1998]; a distinguishing feature was the inclusion of a conceptual design phase aimed at better formalizing the data schema. A sequential approach to design is also followed in [Luján-Mora and Trujillo, 2003], where an object-oriented method based on UML is proposed to cover analysis, design, implementation, and testing. Another UML-based method is presented in [Prat et al., 2006]; here, the use of the *Common Warehouse Metamodel* (CWM) is suggested to promote a more standard approach to conceptual design. All these methodologies follow a linear approach that hardly adapts to changes and is unsuitable when requirements are uncertain.

To overcome these issues, iterative solutions have been proposed in the literature. Iterative approaches are typically adopted by methodologies like RAD and Agile. The work in [Hughes, 2008] breaks with strictly sequential approaches by applying two Agile development techniques, namely *scrum* and *eXtreme Programming*, to the specific challenges of DW projects. To better meet user needs, the work suggests to adopt a *user stories decomposition* step based on a set of architectural categories for the back-end and front-end portions of a DW. However, it does not deeply discuss how this decomposition impacts on modeling and design.

A different approach to tackle the DW design complexity is the MDA methodology proposed in [Mazón and Trujillo, 2009] to better separate the system functionality

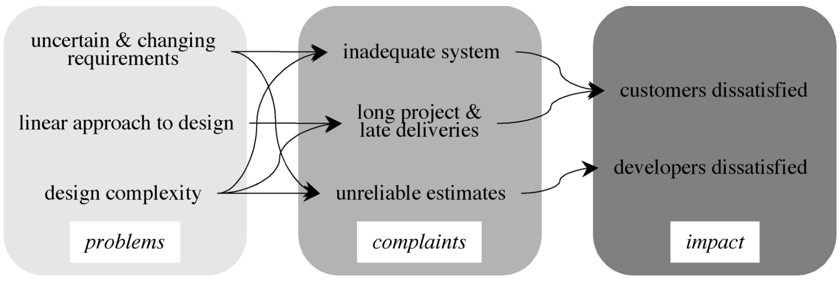


Figure 6.1: Cause-effect relationships in customer and developer dissatisfaction

from its implementation. Strong relevance is given to the development of the DW repository; the three main perspectives of MDA (CIM, PIM, and PSM) are defined using extensions of UML and CWM, and the inter-model transformations are described using the *Query/View/Transformation* (QVT) language. In practice, strictly applying this methodology may be hard due to the poor aptitude of users for reading formal models and investing resources in low-values activities.

A pragmatic comparison between DW design methodologies is offered in [Sen and Sinha, 2005], where 15 different solutions proposed by BI software vendors are examined. The authors emphasize the lack of software-independent approaches, and point out that all the proposed solutions hardly can deal with changes and market evolution, which creates a robustness problem.

6.3 The Motivation for 4WD

The tern *Problems-Goals-Principles* represents the 'fil rouge' of our research method. First, we carried out a deep investigation on the main reasons of failure of the current DW methodologies (i.e., problems). Second, we discuss the goals to improve the DW development process (i.e., goals). Third, we find the principles to pursue the goals (i.e., principles). Finally, we describe how 4WD encompasses the principles.

6.3.1 From Problems to Goals

Our experience with real projects led us to attempt a classification of the main reasons why customers (meant as both sponsors and users) and developers often end up with being dissatisfied. Figure 6.1 summarizes the results of this investigation, distinguishing between problems, complaints, and their human impact, and emphasizing the existing cause-effect relationships between them. A closer glance at the *problems* column reveals that:

- Requirements for data analyses are often unclear and uncertain, mainly because decision processes are flexibly structured and poorly shared across large organizations, but also because of a difficult communication between users and analysts. Besides, the fast evolution of the business conditions may cause requirements to drastically change even in the short-term [Giorgini et al., 2008]. Failing to address these problems dramatically contributes to making users perceive the system as inadequate from the functional point of view and leads to inflating the overall project duration and cost by introducing unexpected delays in the development process.

- DWs are normally built one data mart at a time; each data mart is developed following a linear approach, which means that the different phases are organized into a rigid sequence. Releasing a data mart requires 4-6 months, and it is very difficult to provide intermediate deliveries to be discussed and validated with users, who may easily feel not sufficiently involved and understood, and loose interest in the project.

- The intrinsic complexity of DW design depends on several issues. Among the most influential ones, we mention a couple: DW design leans on data integration, that in most cases is a hard problem; the huge data volume and the workload unpredictability make performance optimization hard. Problems related to data quality and performances have a particularly negative impact on the perceived system inadequacy.

We argue that these problems can be solved by working on four qualities of the software development process [Ghezzi et al., 2002], as explained below.

1. The *reliability* of a development process is the probability that the delivered system completely and accurately meets user requirements. In our context, increasing the reliability of the design process can contribute to addressing the "inadequate system" complaint, i.e., to ensuring a high-quality and satisfactory final system.

2. By *robustness* we mean the process flexibility, i.e., its capability of quickly and smoothly reacting to unanticipated changes in the environment. A robust process can more effectively accommodate both uncertain and changing requirements.

3. The process *productivity* measures how efficiently it uses the resources assigned to the project to speed up system delivery. Increasing productivity leads to shorter and cheaper projects.

4. The *timeliness* of a process is related to how accurately the times and costs for development can be predicted and respected. A timely process makes resource estimates more reliable.

6.3.2 From Goals to Principles

To understand how the main software engineering methodologies devised in the last thirty years can help designers achieve our four quality goals, we analyzed the objectives and underlying principles of seven methodologies, namely *Waterfall* [Royce, 1987], *Rapid Application Development* [Martin, 1991], *Prototyping-Oriented Software Development* [Pomberger et al., 1991], *Spiral Software Development* [Boehm, 1988], *Model-Driven Architecture* [Kruchten, 1995], *Component-Based Software Engineering* [Heineman and Councill, 2001], and *Agile Software Development* [Agile Manifesto, 2010]. Overall, the emerging methodological principles can be condensed as follows:

- *Incrementality and risk-based iteration.* Developing and releasing the system in increments leads to a better management of the project risks, thanks to a proper prioritization of activities aimed at letting the most critical requirement features drive the design of the skeleton architecture. A stepwise refinement based on short iterations increases the quality of projects by supporting rapid feedback and quick deliveries [Boehm, 1988, Martin, 1991].

- *Prototyping.* Complex projects are conveniently split into smaller units or increments corresponding to sub-problems that can be more easily solved and released to users. To facilitate requirement validation and obtain better results, system development is achieved by refining and expanding an evolutionary prototype that progressively integrates the implementation of each increment [Pomberger et al., 1991].

- *User involvement.* Project specifications are difficult to be understood during the preliminary life-cycle phases. A user-centered design increases customer satisfaction and promotes a high level of trust between the parties. Indeed, this feature focuses on constant communication and user participation at every stage of software development.

- *Component reuse.* The reuse of predefined and tested components speeds up product releases and promotes cost reduction as well as software reliability [Heineman and Councill, 2001].

- *Formal and light documentation.* A well-defined documentation is a key feature to comply with user requirements. Moreover, formal analysis leads to clear and non-ambiguous specifications, and user involvement enables light and up-to-date documentation [Agile Manifesto, 2010, Kruchten, 1995, Royce, 1987].

- *Automated schema transformation.* This feature involves the use of formal and automated transformations between schemata representing different software perspectives (e.g., between conceptual and logical schemata). This accelerates software development and promotes standard processes [Kruchten, 1995].

Table 6.1: Expected impact of methodological principles on process quality goals

	Reliability	*Robustness*	*Productivity*	*Timeliness*
Incrementality and risk-based iteration	continuous feedback, clearer requirements	better management of change	better management of project resources, rapid feedback	early detection of errors
Prototyping	frequent tests, easier error detection		early deliveries	
User involvement	better requir. validation, better data quality			early error detection
Component reuse	error-free components		faster design	predictable development
Formal & light documentation	clearer requirements	easier evolution	faster design	
Autom. schema transformation	optimized performances	easier evolution	faster design	predictable design

Table 6.1 summarizes the relationship between these methodological principles and the four quality goals introduced in Subsection 6.3.1, i.e., it gives an idea of how each principle can help increase each quality factor with specific reference to a DW project. More details are given in the following section.

6.3.3 From Principles to 4WD

In this section we propose an innovative design methodology, called 4WD, leaning on the principles discussed in the previous section. These principles are applied in such a way as to effectively balance their pros and their cons, as resulting from practical evidences emerged during the real DW projects 4WD was applied to. Besides the projects we were directly involved in, our findings are based on an elaboration of the experiences collected during the last five years by some practitioners we collaborate with.

As sketched in Figure 6.2, 4WD is based on nested iteration cycles. The external one is called *data mart cycle*; it defines and maintains the global plan for the development of the whole DW and, at each iteration, it incrementally designs and releases one data mart. After completing the activities related to the data mart planning, the team proceeds with the data mart design. It is achieved by the *fact cycle*, that refines the data mart plan and incrementally designs and releases its facts. Finally, fact design is based on two cycles (*modeling* and *implementation* cycles, respectively), that include the core of analysis, design, and implementation activities for delivering reports and

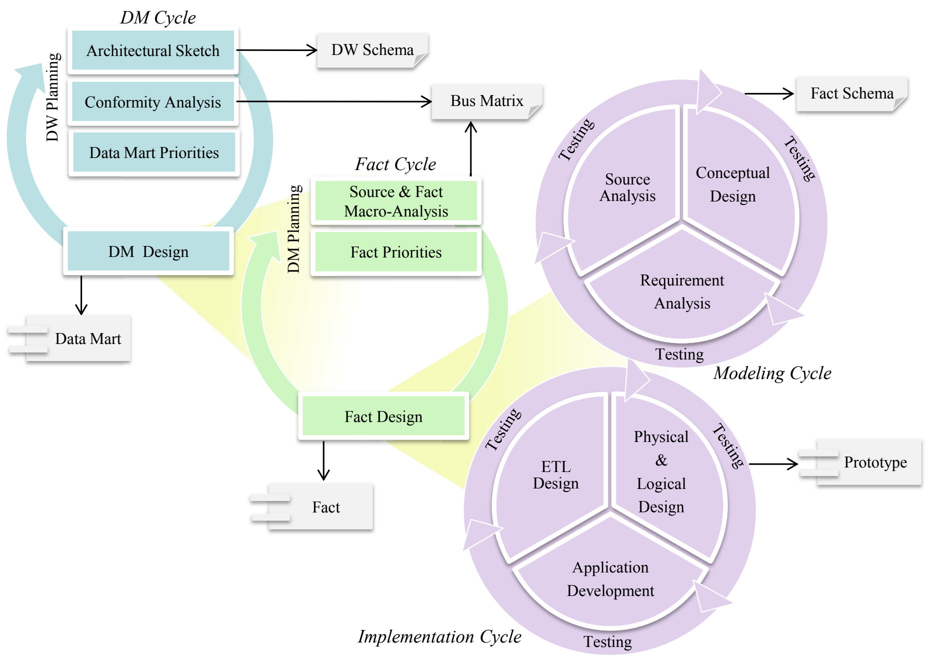


FIGURE 6.2: A sketch of the 4WD methodology

applications concerning a single fact. The activities in a cycle can be carried out in parallel. The documents produced can be distinguished into releases (that correspond to project milestones) and deliveries (used for testing and validation). Remarkably, cycles are nested in a way that enables a reassessment of the decisions made during an outer iteration based on the evidences emerging from an inner iteration.

The main activities carried out in the data mart cycle are:

- *Architectural sketch*, during which the overall functional and physical architecture of the DW is progressively drawn based on a macro-analysis of user requirements and an exploration of data sources as well as on budget, technological, and organizational constraints.
- *Conformity analysis*, aimed at determining which dimension of analysis will be conformed across different facts and data marts. Conforming hierarchies in terms of schema and data is a key element to allow cross-fact analysis and obtain consistent results.
- *Data mart prioritization*, based on a trade-off between user priorities and technical constraints.
- *Data mart design*, which builds and releases the top-priority data mart. After each data mart has been built, the three phases above are iterated to allow the DW plan to be refined and updated.

The activities carried out within a fact cycle are:

- *Source and fact macro-analysis*, aimed at checking the availability, quality, and completeness of the data sources and determining the main business facts to be analyzed by users.
- *Fact prioritization* that, like for data marts, is the result of a trade-off between user requirements and technical priorities.
- *Fact design*, which develops and releases the top-priority fact. After that, the two phases above are iterated to allow the data mart plan to be refined and updated.

Finally, the activities necessary to release a single fact (or even a small set of strictly related facts) are grouped into two separate sub-cycles to emphasize that releasing a conceptual schema of a fact marks a clear separation between a modeling and an implementation phase for the fact itself. Validating the conceptual schema of a fact before implementation leads to reducing the number of implementation cycles, i.e., to faster fact cycles. While modeling should come before implementation, the activities included in each sub-cycle are not strictly sequential and can be differently prioritized by each project team. Each sub-cycle can be iterated a number of times before its results (the conceptual schema in the first case, the analysis applications in the second) are validated and released.

6.4 The 6 Features of 4WD

4WD is characterized by 6 features complying with the principles described in Subsection 6.3.2.

6.4.1 Incrementality and Risk-Based Iteration

As suggested by the RAD approach, iteration is at the core of 4WD and is coupled with incremental development, that aims at slicing the system functionality into increments; in each increment, a portion of the system is designed, built, and released. Developing a system through repeated cycles leads to lower risk of misunderstood requirements (higher reliability and timeliness), to faster software deliveries (higher productivity), and to more flexible management of evolving requirements and emerging critical issues (higher robustness) [Martin, 1991].

Though these advantages are largely acknowledged in all modern methodologies, the type of iterations and their frequencies vary from one another depending on the type of software to be developed. For example, agile methodologies pushes segmentation to the limit by centering iteration on the so-called *user stories*, meant as high-level functional requirements —concisely expressed by users in their business language— that can be released in a few days. Since functional requirements in DW projects are mainly expressed in terms of analysis capabilities, agile DW design often focuses each iteration on a small set of reporting or OLAP functionalities. While this may sound natural to business users, it can lead to dramatically increasing the overall design effort, because it gives little or no relevance to the multidimensional schemata adopted to store information. Indeed, as reported by designers who adopt functionality-centered iterations in DW projects, a common problem is that they fail in recognizing that apparently different analyses, designed during separate iterations, are actually supported by the very same multidimensional schema.

In 4WD, the shortest iterations that release a tangible result to users are those for modeling and implementing a single fact, that are normally completed in 2-4 weeks overall. This release rate could seem to be not very high, but it is backed by quite more frequent deliveries. Indeed, the modeling and implementation cycles have a daily to weekly frequency; the deliveries they produce enable a progressive refinement of the fact conceptual schema and implementation through a massive test based on active involvement of users.

Incremental techniques require a driver to define an order for developing increments. In 4WD this is done when deciding data mart and fact priorities, and in both cases risk is the driver —as suggested by the Spiral Software Development approach [Boehm, 1988]. The project team should balance the risk of early releasing data marts/facts that are not highly valuable to users —which would lead users to lose interest in the project— against the risk of ordering design activities in a non-optimal way —which would determine higher costs and a longer overall project duration. Some guidelines for reducing the risk in data mart prioritization are: (a) Give priority to data marts that include widely shared hierarchies, which makes the overall schema more robust and ensures that dimensions are fully conformed; (b) Give priority to data marts that are fed from stable and well-understood data sources; and (c) Postpone data marts based on unclear requirements, assuming that these requirement will be better understood as the user's involvement in the project increases. As to facts: (a) Give priority to facts that include the main business hierarchies and require the most complex ETL procedures; (b) Adopt a data-driven approach to design rather than a requirement-driven one whenever users do not appear to have a deep knowledge of the business domain; and (c) Plan the length of an iteration in proportion to the complexity of the fact, since failing a release in the early stage of a project will undermine the team credibility.

6.4.2 Prototyping

Prototyping has a crucial role in most modern software projects. In a DW project, an *evolutionary* (where a robust prototype is continuously refined) and *incremental* (where the prototype is gradually enlarged by adding new sub-systems) approach to prototyping is generally preferable to a *throw-away* approach (where the prototype is used to demonstrate a small set of functions and then is abandoned). In fact, the effectiveness of prototyping is maximized when the prototype is tested together with users, and in a DW project this requires the whole data flow —from operational sources to the front-end through ETL— to be prototyped: a large effort, that should not be wasted. The main advantages of prototyping, with particular reference to a DW project, can be summarized as follows:

- Prototypes help designers to validate requirements, because they allow users to evaluate designers' proposals by trying them out, rather than interpreting design documents. This is particularly crucial to enable a better understanding of hierarchies by users [Sommerville, 2004].
- Prototypes are especially valuable to improve the design of reports and analysis applications, due to their interactive nature. In general, prototype-based user-interfaces have higher usability [Gordon and Bieman, 1995].
- Prototypes can be used to advance testing to the early phases of design, thus reducing the impact of error corrections. For instance, an early loading test can be effectively coupled with a preliminary functional test of front-end applications to check for correct data balancing [Golfarelli and Rizzi, 2009a].
- Prototypes can be used to evaluate the feasibility of alternative solutions during logical design of multidimensional schemata and during ETL design. This typically leads to improved performance and maintainability, and to reduced development costs [Sommerville, 2004].

The above points are basically associated with an increase in reliability and productivity. More specifically, the impact on reliability is related to both data schemata, data quality, and performances. First of all, having a working prototype available during the early project phases enables the designer to keep a strict and constant control over the data schema to ensure that it fully supports user requirements. Then, data quality can be improved by closely involving users in testing the prototype using both real and ad-hoc generated data. Finally, an incremental approach can also be used to take better care of performance issues by following the modularity principle to separate correctness from efficiency. This means that a working prototype can be delivered first; then, performances can be improved during the following iteration to deliver an increment in the form of a working *and efficient* prototype.

6.4.3 User Involvement

Recent years have been characterized by a growing awareness that human resources are one of the keys to a project success. In this direction, some modern software design methodologies tend to emphasize organizational factors rather than technical aspects. For instance, agile approaches pursue the idea of creating responsible and self-organizing teams to maximize participation of developers and their productivity. They also focus on user involvement as a means to reduce the risk of expressing ambiguous requirements and make software validation easier and more effective [Agile Manifesto, 2010].

4WD pays a large attention to user involvement because it has a substantial influence on process reliability and timeliness. User involvement can be promoted in different ways:

- All users should preliminary receive a comprehensive training to clarify the project goals, explain the multidimensional model, and introduce a shared language for conceptual design.
- Prototyping is the most effective way to have users participate in the design process and keep them aware of the project status.
- Due to the complex data transformation that is inherent to DW systems, only users —who have insight of business data— can easily detect problems and errors. So, most testing activities should be based on user feedback. User involvement is specifically crucial for usability tests of reporting and OLAP front-ends, and for functional tests of ETL procedures.

6.4.4 Component Reuse

Applying a component-based methodology means using predefined elements to support the software development process [Heineman and Councill, 2001]. This is often done by DW designers, though mostly in an unstructured way. The components that can most effectively be reused in a DW project are:

- Conformed hierarchies, that are reused in different facts and data marts. Using conformed hierarchies not only accelerates conceptual design, but is also the key for achieving an enterprise view of business in a DW.
- Library hierarchies, that model common hierarchy structures for a given business domain. For instance, a customer hierarchy in a sales analysis has some basic features that can be easily reused in different DW projects to reduce the effort in designing facts.
- Library facts, that define common measure and dimension structures as emerging from design best practices for a given business domain. Of course, library facts must be tailored to specific user needs; nevertheless, they may be very useful in

requirement-driven approaches to give designers and users a starting point for conceptual design.

- ETL building blocks, meant as predefined extraction, transformation, cleaning, and loading routines (e.g., a routine for cleaning a geographical attribute against the list of ISO 3166-2 codes for administrative divisions, or one for loading a type-3 slowly-changing dimension from an operational data store). Reusing such routines reduces the ETL design effort and makes ETL more reliable due to the use of largely-tested algorithms.
- Analysis templates, that define a reference structure for reports and applications. In particular, sharing an analysis template across a DW project is warmly suggested to standardize the interface presented to users.

4WD takes advantage of component reuse to accelerate development and increase robustness. While ETL tools already include some building blocks that can be easily reused through parameterization, identifying hierarchies and facts to be reused deserves more attention. 4WD devotes an ad-hoc phase (*conformity analysis*) to identifying hierarchies to be conformed using a bus matrix. Besides, conceptual schemata are a very effective tool to formalize the structure of facts and hierarchies and support their matching against the available libraries.

6.4.5 Formal and Light Documentation

In waterfall approaches, documentation is extensively used during the whole life-cycle to support the design process and represent and validate requirements. Other approaches, like RAD and agile methodologies, tend to discourage the use of documentation (other than the one automatically produced by tools) because it may lead to prematurely freezing requirements and slowing down iterations, and suggest to replace it with continuous communication with users [Agile Manifesto, 2010, Martin, 1991].

While we agree that textual documentation should be reduced to the minimum, we firmly believe that formal documentation is a key factor to promote precise formalization of requirements, clear communication between designers and users, accurate design, and maintainability. In 4WD, the main role to this end is played by conceptual schemata. In particular:

- At the DW level, we mostly use a simple but effective schema that summarizes the data marts, their data sources, and the profiles of the users who access them [Golfarelli and Rizzi, 2009b]. This high-level schema is first drawn during the architectural sketch phase, and refined after each data mart cycle. It is essentially used to share the basic functional architecture with users and to support the discussion of data mart priorities.

- At the data mart level, an important role is played by a *bus matrix* that associates each fact with its dimensions, thus pointing out the existence of conformed hierarchies. This schema is built and progressively refined during the *conformity analysis* and *fact macro-analysys* phases, and is used to test that the designers has properly captured the existing similarities between different facts and different data marts, thus ensuring their integrability [Golfarelli and Rizzi, 2009b].
- At the fact level, we force designers to complete and release the conceptual schema of a fact *before* proceeding with implementation. Indeed, having users and designers clearly agree on the fact granularity and measures, as well as on the hierarchy structures and semantics, is the most effective way to avoid misunderstandings and omissions. Finding this agreement informally, or leaning on the logical/physical schema of the fact, is obviously hard and error-prone, while a (graphical) conceptual schema is clearly understood even by non-technical users. In particular, we adopted the DFM in a number of projects for public administrations (such as local health authorities, the Ministry of Justice, the State Accounting Department) and we verified that fact schemata are also understood by non-IT people such as physicians and jurists.

A major role in this context is also played by metadata, that multidimensional engines store to describe the structure of a data mart. Metadata can typically be exported to generate a documentation based on standard languages (such as XML) and models (such as the CWM); this also encourages interoperability, that is normally seen as a crucial issue in DW projects.

6.4.6 Automated Schema Transformation

To reduce design complexity, the MDA approach proposes to use formal models for separately specifying a *Platform Independent Model* (PIM, it represents system functionalities at a conceptual level) and a *Platform Specific Model* (PSM, it gives a logical and platform-dependent representation of system functionalities), and to use automated transformations to derive a PSM from a PIM. In a DW project, this can be applied to design both ETL procedures and multidimensional schemata, as shown in [Mazón and Trujillo, 2009, Simitsis and Vassiliadis, 2008].

In 4WD, automated schema transformations are encouraged, mainly to speed up design and simplify evolution, as long as they need a reasonable effort from users to understand formal models and they do not require to invest too many resources in activities that are not directly valuable to users. We propose two metadata-based activities for automation, possibly supported by CASE tools:

- Supply-driven conceptual design. In supply-driven approaches, a basic conceptual schema for a fact can be automatically derived starting from the logical schema of operational data sources [Moody and Kortink, 2000]. When applicable, this is a very effective way to cut design costs.

- Logical design. A logical schema can be automatically obtained from a conceptual schema by applying a set of transformations that express common design rules and best practices, possibly based on the expected workload [Golfarelli and Rizzi, 2009b].

6.5 Validation

4WD was applied to a project in the area of pay-tvs (PayTV project, in the following). The project had an overall duration of 8 months and was carried out by an Italian system integrator specialized in BI applications.

During DW planning two data marts were identified, namely administration and management control, that were prioritized according to their importance for users: the administration data mart was given higher priority because its size is definitely larger (10 vs. 4 facts). During data mart planning we organized the overall project in 10 releases (7 for the first data mart, 3 for the second one), each centered on at most 3 facts and taking from 10 to 26 days. Facts were grouped into a single release when they either shared several dimensions or had similar ETL processes (e.g., because measures were extracted from the same data sources and tables), as emerging from conformity analysis and source and facts macro-analysis. Each release was then assigned a value from the users point of view, an estimated nominal complexity, and a risk expressed as a percentage complexity overhead (ranging from 19 to 35%) to determine a worst-case complexity. The criteria used for establishing release priorities were: (1) advance the most valuable facts to early releases; (2) uniformly distribute the worst-case complexity; and (3) respect the dependencies in fact implementation. Besides, some fact were delayed because the development of specific extraction interfaces by external consultants was required for some of their source data; other facts were postponed due to some uncertainty on the requirements. After each release, its actual duration was compared to the estimated complexity. In 2 cases it turned out that the estimation was inaccurate; this was fixed right away by revising the remaining estimates and by changing the team composition.

One of the benefits of adopting 4WD in this project was the speed-up due to large user involvement and extensive prototyping. Users were enabled to access a web portal to signal the errors, and monitor the team's answers and the project state. This was particularly effective for improving the structure of reports and the business rules for detecting source data errors. Noticeably, all errors signalled by users were related to wrong data: user mainly own empirical knowledge, so it may be hard for them to reason from an abstract point of view (e.g., to evaluate an ETL flow or a report structure with no data loaded). The implementation effort was reduced by partially reusing existing reports and dimension tables, because those required by administration and management control users are quite standard. This was not the case for ETL, that required a strong personalization, so reuse was limited to some basic routines made available by the adopted ETL suite. Finally, adopting the DFM as a conceptual model enabled designers to produce a concise but exhaustive documentation, and to use a CASE tool to automate logical design [Golfarelli and Rizzi, 2001].

6.6 Conclusions

In this chapter, we proposed a new methodology to make DW development nimbler and faster, so as to support BI ANYTIME [Golfarelli et al., 2011c]. 4WD relies on three key factors: (a) iteration breaks the linear development process by offering frequent deliveries and reviewing points; (b) a formal and light documentation provides a clear picture of the current specifications, facilitating the identification of the units to be evolved; (c) automating schema transformations reduces the time needed to propagate changes to the different levels. Our methodology has been successfully applied to a case study on a real project in the pay-tv area, leading to the following advantages: (1) reduction of the implementation effort by re-using existing reports and dimension tables; (2) project speed-up thanks to large user involvement and exhaustive prototyping; (3) concise and clear documentation by adopting the DFM specification. As to future works, additional real case studies may help to better refine and improve 4WD.

Chapter 7

Project Scheduling Optimization in Agile Data Warehouse Design

This chapter closes the two works on BI ANYTIME. In Chapter 6 we described a new methodology (4WD) to facilitate the DW development; here, we formalize an optimization model for the project scheduling that is compliant with the 4WD principles and improves the effectiveness of resource allocation. This chapter includes two main sections: the first is devoted to the model formalization and its extension to deal with project uncertainty, the second proposes different optimization algorithms to efficiently solve the planning problem.

7.1 Introduction

In iterative and incremental approaches, such as 4WD, the planning problem has a key role to ensure the project success [Svahnberg et al., 2010].

We better describe the problem adopting the terminology of *Scrum* and *eXtreme Programming* (XP), that are the two most common methodologies in agile (thus, iterative) contexts nowadays [Dybå and Dingsøyr, 2008]. The software is described in terms of detailed user functionalities (*user stories* or *stories* for short) and at each iteration (*sprint* in the Scrum terminology), the set of user stories that maximizes the utility for the users and fulfills a set of development constraints is delivered [Schwaber, 1995]. Typical constraints include limiting the duration of an iteration, respecting correlations among user stories, and containing the non-delivery risk. We can now refer to the planning problem as the *multi-sprint planning problem*, emphasizing that we define a plan spanning multiple sprints.

In this context, user story prioritization and definition of sprint boundaries are obtained by sharing and averaging the estimates given by the different team members about story complexity, utility, and precedences. For example, advancing high-valued stories could lead to an early significant result for users and encourage the team awareness; similarly, developing affine user stories within the same sprint can increase their perceived value. Moreover, new requirements may arise during the project, and the plan should be flexible enough to accommodate them; otherwise, it may be impossible for the project team to perfectly stick to the baseline plan for various reasons, such as underestimation of story complexity, unavailability of team members, or changing requirements, which may lead some sprints to fail, meaning that their results cannot be delivered as expected [Beck, 1999].

In this direction, a number of approaches have been devised in the literature, but none of them provides comprehensive coverage of all the features involved in iterative approaches (see Section 7.2 for a detailed explanation). To fill this gap we analyze the multi-sprint planning problem from two perspectives:

- *Mathematical formulation*: in Section 7.3, we propose a mathematical formulation of the multi-sprint planning problem that, given the team estimates and a set of development constraints, produces a plan that maximizes the business value perceived by users, thus relieving the team from the difficult task of quickly producing an optimal plan (see Golfarelli et al. [2012c]). The optimal plan must be seen as an initial recommendation for the team, and it can be manually adjusted. The "best" plan may be one that also considers the personal experiences of the team members and some additional constraints that could not be formally modeled. For this reason, our model allows user stories to be explicitly *forced* into sprints. Moreover, to cope with the possible failure of a sprint (one or more user stories could not be delivered as expected), with the emergence of new requirements (one or more user stories are added), and with intrinsic changes in the development process (the development speed estimated must be adjusted), our model provides capabilities of *smooth replanning*, meant as revising and re-optimizing a baseline plan during project execution without disrupting it. Finally, the section includes both effectiveness and efficiency tests. As to efficiency, we just test performance by using a general-purpose *Mixed Integer Programming* (MIP) solver, such as IBM Ilog Cplex [IBM, 2011]. More sophisticated solutions to improve performance will be presented in the next section.

- *Performance*: in Section 7.4, we propose different strategies to efficient solve the multi-sprint planning problem. As a matter of fact, our mathematical formulation is a generalized assignment problem [Martello and Toth, 1990] with side constraints, where the knapsacks are the sprints and the items are the user stories. The generalized assignment problem is NP-hard [Martello and Toth, 1990], thus, for difficult instances, the model cannot be solved to optimality by a MIP solver. For this reason we propose an effective Lagrangian heuristic based on a relaxation of the proposed model and some greedy and exchange algorithms.

Computational results on both real and synthetic projects show the effectiveness of the proposed approach.

7.2 Related Works

In recent years, different models for the planning problem have been devised in iterative and incremental contexts. [Denne and Cleland-Huang, 2004] proposes a software development strategy based on financial factors. An optimal sequence of requirements to deliver is generated by maximizing along time the *net present value*, i.e., a combination of revenues, costs, and risks of each requirement. Two solution strategies are proposed: a greedy algorithm and a look-ahead approach. The first one selects the next requirement to deliver by considering the requirements with no unfulfilled precursors and maximum net present value; the second one extends the greedy approach by analyzing subsets of profitable precedence sequences.

[Szoke, 2011] describes a conceptual model for release scheduling and provides an optimization model aimed at assigning requirements to the different iterations of a release by maximizing the overall value delivered and considering precedences and coupling conditions. Then, it describes a branch-and-bound algorithm to solve the model incorporating risk management. Another work situated in the agile context is the one by [van Valkenhoef et al., 2011], that is mainly focused on managing risk and uncertainty in XP projects. To this end, the authors estimate the team development speed and consider multiple sets of user stories with decreasing relevance ("must have", "should have", "could have" sets); the goal is to assign each user story to the most proper set by maximizing the overall value of the sets and respecting precedences and correlations between user stories. A branch-and-bound algorithm is used to find the best solution. The limited number of sets they consider leads to a coarse-grained plan that must be refined to obtain an operative schedule (e.g., by breaking sets according to budget bounds and splitting user stories into smaller tasks).

None of the above-mentioned works specifically deals with change management, an approach in this direction is *Evolve* [Greer and Ruhe, 2004], that is aimed at iterative and incremental contexts. A release plan includes different increments; at each stage, a set of requirements is allocated to the current and the future increments in such a way as to return the best trade-off between stakeholder priorities and development constraints (such as increment capacity, precedences, and coupling conditions). The model is formalized as a multiple knapsack problem and a genetic algorithm is used to solve it. To deal with change, Evolve includes a partial strategy for replanning: at each increment, new requirements and changes in priorities and/or constraints are allowed, and a new solution is generated from scratch. In the context of scrum planning, [Li et al., 2010] gives a knapsack formulation of an optimization model for single-iteration planning that selects the requirements maximizing the profit of the next iteration, coping with

TABLE 7.1: Features of planning approaches for iterative life-cycles

Approach	Scope	Hard Constr.	Soft Constr.	Risk	Change Mgmt.	Planning
Greer, 2004	multi-iter.	preced., coupling	no	partial	partial	heuristic
Denne, 2004	multi-iter.	preced.	no	yes	no	greedy
Saliu, 2007	single-iter.	preced., coupling	no	no	yes	exact
Li, 2010	single-iter.	preced.	no	no	partial	exact
Szoke, 2011	multi-iter.	preced., coupling	no	yes	no	exact
Valkenhoef, 2011	multi-iter.	preced.	coupling	yes	no	exact
Our approach	*multi-iter.*	*preced., forced*	*coupling*	*yes*	*yes*	*exact*

development requirements. Evolution is managed by allowing changes in parameters after each iteration and coping with their impact on the model. A new solution for next iteration is produced from scratch, by incorporating changes and additional stories. A more sophisticated approach is *bi-objective planning* [Saliu and Ruhe, 2007], in which the next iteration is planned considering the impact of new requirements or changes on the existing system from either the business or the development perspective. A set of plans is generated, each reflecting a different importance of business and implementation aspects, then the optimal plan is chosen as the one that best satisfies a group of interdependencies (called *SD-couplings*) between requirements identified through impact analysis.

We close this section classifying the aforementioned approaches according to the selection of relevant features of planning for iterative life-cycles, proposed by [Saliu and Ruhe, 2005]; the slightly different set of features we adopt here is aimed at providing better insight into the planning model. Table 7.1 shows that none of the models provides comprehensive coverage of the features. Most noticeably, there is partial support to change management, that has such a crucial role in iterative projects. Managing change becomes critical in approaches that produce a look-ahead plan covering multiple iterations, because a significant alteration of future iterations may create problems with resource allocation and frustrate the users' expectations. The only multi-iteration approach that gives some support to change is the one by [Greer and Ruhe, 2004]; however, a new plan is produced from scratch after each iteration without any correlation with the previously produced plan. To fill this gap, we formalize a *multi-sprint* planning problem by taking into account all the features of Table 7.1.

With reference to Table 7.1, the key features of our approach can be summarized as follows:

Scope Ours is a multi-iteration approach that supports a single co-located and cross-functional team during a medium- to long-term planning, in a look-ahead perspective.

Hard Constraints Precedences, that typically characterize the development process, are modeled as hard constraints. Besides, a story can be forced to be included into a given sprint.

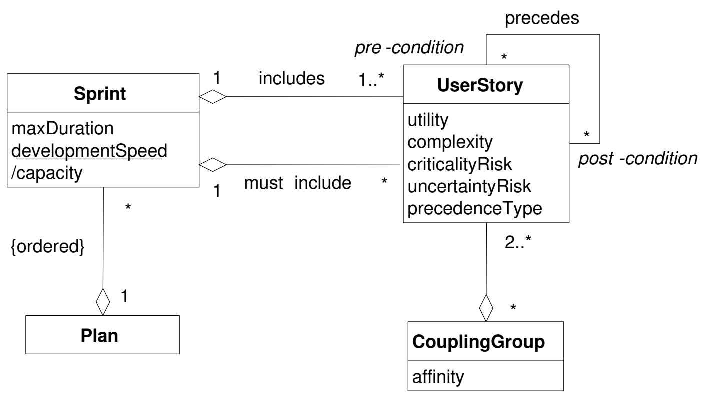


FIGURE 7.1: The user story model as a UML class diagram (static attributes are underlined, roles are in italics)

Soft Constraints Consistently with the agile philosophy, couplings are modeled as soft constraints by increasing the business value perceived by users if two or more affine stories are developed in the same sprint.

Risk We deal with the risk related to both uncertain and critical stories: an uncertain story is one whose complexity can hardly be estimated, a critical story is one that has a strong impact on the quality of the system being developed.

Change Management We called our way of managing plan evolution in multi-iteration scenarios *smooth replanning*. The idea is to allow a baseline plan to be revised and re-optimized, if necessary, during project execution without disrupting it so as to protect the allocation of resources and preserve the milestones agreed with users.

Planning We provide exact solutions to small and medium problems and sub-optimal solutions (less than 1% worse than the optimal one) for more complex problems (e.g., problems with 100 user stories) in a few seconds.

7.3 Multi-Sprint Planning Problem

Our formulation of the multi-sprint planning problem is based on the static model shown in Figure 7.1, that takes into account the main variables that affects user stories prioritization and sprint composition. The concepts represented are:

- *Plan*: a sequence of sprints.
- *Sprint*: the time-bound unit of iteration, typically a one- to four-week period, depending on the project complexity and risk assessment. A sprint includes a set of user stories. A maximum duration is fixed for each sprint.

- *User story*: a relatively small piece of functionality valuable for users [Cohn, 2004]. It represents a light specification that can be later detailed thanks to a continuous communication with the user; at the same time, it must be sufficiently described to estimate its development complexity. It represents a means to communicate between users and developers. In some situations the project team may want, for various reasons, to constrain some user stories (which we will call *forced*) to be included in a specific sprint.

- *Utility*: the business value of a user story as perceived by the user that defines it. As stressed by [Racheva et al., 2009], a detailed definition of *business value* is still missing in agile methodologies; a general and self-evident interpretation is normally assumed, related to the *earned value* defined in economics and transformable into dollar value. In practice, users normally express the value of each story using a single number, though they may implicitly take into account and combine different utility criteria.[1] In some approaches it is only required to define an ordering for user stories (i.e., user story 1 is more useful than user story 2), but in general it can be quantified through a positive numerical score typically ranging between 10 and 100 [Nichols, 2009]. For instance, a story for having a site map effectively indexed by a research engine could have utility 80, because it relevantly impacts on the site visibility on the web, while a story for showing photographs of the staff members on the site could have utility 10 because it adds small value to the site content.

- *Complexity*: the development effort for a user story measured in *story points*. Team members assign story points to each user story based on their experience and knowledge of the domain and project specificities. Story points are non-dimensional and are preferred to time/space measures to avoid subjective and incomparable estimates. Typical complexities of user stories range between 1 and 10 story points [Nichols, 2009]. For instance, the indexing story mentioned above could have complexity 7, while the photograph story could have complexity 1.

- *Risk*: we consider risks related to two different characteristics of user stories. A *critical* story is one that may have a strong impact on the quality of the system delivered, so that taking a wrong solution for it dramatically affects the success of the project (e.g., a story for defining the deployment architecture that heavily impacts on performances and security). An *uncertain* story is one for which it is somehow hard to estimate the complexity due to unexpected problems that could arise (e.g., a story for feeding a database from data flows produced by a third-party company).

[1] Our model can seamlessly accommodate different types of utility, meant both from the users point of view (e.g., positive impact of a story on sales and revenues, or effects on customer fidelity) or from other points of view (e.g., not degrading the overall software architecture) as long as these can be combined into a formula.

- *Coupling*: a correlation between two or more affine user stories. Affine stories have higher utility if they are included in the same sprint, because users better perceive the overall business value of the functionality delivered. For instance, a "zoom-out" story may have low utility on its own, but its utility may increase if delivered together with the complemental "zoom in" story. A user story can be included in several coupling groups, each characterized by an affinity: the higher the affinity, the higher the utility in jointly delivering the functionalities.

- *Precedence*: a hard constraint stating that a user story can be developed only after one or more other user stories (called *pre-conditions*) have been completed. For instance, a database can be created and populated only after its conceptual schema has been designed and documented. A conjunctive precedence (AND-type precedence) implies all pre-conditions must be completed, while a disjunctive precedence (OR-type precedence) implies at least one of the pre-conditions must be completed.[2]

- *Development speed*: the number of story points the team can deliver per day. It is used to convert the sprint duration into the *sprint capacity* (i.e., the maximum number of story points the team can deliver in a sprint).

To make this model applicable, reference values and ranges must be chosen for its concepts. We estimate both types of risk by associating values in the range [1..2] to four classes of risk: 1 (no risk), 1.3 (low risk), 1.7 (medium risk), and 2 (high risk). Besides, the affinity range we adopt is [0, 0.5], meaning that the utility of a story can be increased at most by 50%. We remark that the validity of our approach does not depend on the reference values and ranges proposed, that were chosen to fit the specific features and needs of the teams we worked with. Different teams may take advantage from using finer or coarser classes and different ranges, depending on the typical precision of their estimates.

We can now list the goals an optimal baseline plan should pursue:

♯1 *Customer satisfaction.* It can be obtained by early delivering high-valued sprints. In the agile philosophy, this also increases the user awareness and trust.

♯2 *Coupling management.* Affine stories should be carried out in the same sprint to increase their utility for users. We argue that the increase in utility comes from the presence of any affine stories in the same sprint, i.e., users perceive higher utility even if only *some* of the stories in a coupling group are delivered together. In light of this, couplings can be managed by increasing sprint utility proportionally to the number of affine stories jointly delivered.

[2]More complex expressions, such an OR of AND's and the like, could easily be used to model precedences, with small effects on the overall complexity of the optimization model. However, here we prefer to adopt a simpler form for precedences because, in our experience, it is largely adequate to accommodate the expressiveness required in practice by project teams.

♯3 *Risk management.* It can be achieved by (i) advancing critical user stories to avoid late side-effects, on the one hand; (ii) distributing uncertain stories in different sprints to reduce the risk that a sprint delivery is delayed, on the other hand.

Besides, all constraints related to the sprint capacity, forced user stories, and inter-story precedences must obviously be met.

As anticipated in the previous section, the problem of determining an optimal baseline plan, i.e., one that achieves these goals, can be converted into a generalized assignment problem with side constraints, where the knapsacks are the sprints and the items are the user stories. Story points measure the weight of an item, while utility represents its value. Knapsack capacity (i.e., sprint capacity) is measured as the story points that the team can deliver given the sprint duration and the development velocity. The objective function to be maximized is the cumulative utility of the project (goal 1), where the utility of each story is increased if some affine stories are included in the same sprint (goal 2) and/or if that story is critical (goal 3-i). Finally, in the formulation of the capacity constraint, the story points of user stories are increased by their uncertainty, which discourages the inclusion of two or more uncertain stories in the same sprint (goal 3-ii).

7.3.1 Baseline Planning Optimization Model

Let $U = \{1, \ldots, n\}$ be the index set of the n user stories to be assigned to sprints. Each story $j \in U$ is associated with its utility u_j, its criticality risk r_j^{cr}, its uncertainty risk r_j^{un}, and its complexity p_j in story points. Let Y_j be the set of stories affine to story j and a_j be the increment in utility for each affine story assigned to the same sprint (if $Y_j = \emptyset$, we set $a_j = 0$).

Let U^{OR} and U^{AND} be the subsets of stories having precedence type OR and AND, respectively. For each story $j \in U^{OR}$, let D_j^{OR} be the sets of stories such that at least one of them must be assigned the same sprint of story j or to a previous one. Similarly, for each story $j \in U^{AND}$, let D_j^{AND} be the set of stories that must be assigned to the same sprint of story j or to a previous one. Note that each story can be involved in both OR and AND precedences.

Let $S = \{1, \ldots, m\}$ be the index set of the m sprints. Each sprint $i \in S$ has a capacity of p_i^{max} story points.

Let x_{ij} be a binary variable equal to one if the story $j \in U$ is included in sprint $i \in S$, zero otherwise. Let y_{ij} be a non-negative variable equal to the number of stories of Y_j, $j \in U$, included in sprint $i \in S$. Let $B \subseteq U$ be the set of forced stories, and b_j for $j \in B$ be the sprint forced to include story j; The mixed integer linear programming model is the following:

$$(P) \qquad z_P = \max \sum_{k=1}^{m} \sum_{i=1}^{k} \sum_{j=1}^{n} u_j \left(r_j^{cr} x_{ij} + a_j y_{ij} \right) \tag{7.1}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{max}, \qquad i \in S \tag{7.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j \in U \tag{7.3}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} \geq x_{ij}, \qquad i \in S, j \in U^{OR} \tag{7.4}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} \geq x_{ij} |D_j^{AND}|, \quad i \in S, j \in U^{AND} \tag{7.5}$$

$$y_{ij} \leq \sum_{k \in Y_j} x_{ik}, \qquad i \in S, j \in U \tag{7.6}$$

$$y_{ij} \leq |Y_j| x_{ij}, \qquad i \in S, j \in U \tag{7.7}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in S, j \in U \tag{7.8}$$

$$x_{b_j j} = 1, \qquad j \in B \tag{7.9}$$

$$y_{ij} \geq 0, \qquad i \in S, j \in U \tag{7.10}$$

The objective function (7.1) maximizes the cumulative utility function. The utility u_j of story j is increased by its criticality risk r_j^{cr}, thus encouraging an early placement of critical stories, and by the affinity a_j for each affine story included in the same sprint. Given story j, the number of affine stories included in sprint i is $y_{ij} = \sum_{k \in Y_j} x_{ik}$, if $x_{ij} = 1$, and $y_{ij} = 0$ otherwise. Since we deal with a maximization problem, constraints (7.6) and (7.7) guarantee the correct evaluation of each variable y_{ij}, that does not require an explicit integrality constraint.

Constraints (7.2) ensure that the overall complexity of all the stories assigned to each sprint does not exceed the sprint capacity, while constraints (7.3) guarantee that each story is assigned to a sprint.

Precedences are imposed by constraints (7.4) and (7.5). If a story j has an OR precedence, constraints (7.4) enable j to be assigned to sprint i only if at least one story in set D_j^{OR} is assigned to a sprint $i' \leq i$. Similarly, if j has an AND precedence, constraints (7.5) enable j to be assigned to sprint i only if all stories in set D_j^{AND} are assigned to sprints less than or equal to i. Finally (7.9) correctly places forced stories in their sprints.

IBM Ilog Cplex solves this optimization problem using a branch-and-cut approach [Caprara and Fischetti, 1997], that is, a method of combinatorial optimization for

Table 7.2: A sample of user stories from the case study

Story Id	*Story Name*	*Utility*	*St. Points*	*Crit. Risk*	*Uncert. Risk*
$s1$	fee configuration	80	5	low	low
$s2$	cash cost computation	85	2	medium	medium
$s3$	import from IBMS	75	2	medium	medium
$s4$	parameterization logic	30	1	medium	medium
$s5$	amortization mask	60	2	no	no
$s6$	exchange computation	60	2	low	low
$s7$	exchange import from SAP	60	7	low	low
$s8$	management control reporting	85	4	medium	medium
$s9$	operational reporting	100	10	low	low
$s10$	scenario management mask	65	3	low	low

solving integer linear programming problems (i.e., linear programming problems where some or all the unknowns are restricted to integer values —the x_{ij}'s and y_{ij}'s in our case). The method is an hybrid of branch-and-bound and cutting plane methods that dramatically improves the performance of classic branch-and-bound methods by incorporating *cutting planes*, that is, inequalities that improve the linear programming relaxation of integer linear programming problems.

Example 7.1. *The example we report here is a simplified excerpt from the case study on the* P*ayTv project presented in the previous Chapter (Section 6.5). The user stories considered are listed in Table 7.2 together with their estimations, and are allocated into 4 sprints with capacity equal to 20 story points each —except the third sprint that was given capacity 14 to model the fact that one team member is temporarily unavailable. Two precedence (from* $s7$ *to* $s6$*, from* $s1$ *to* $s2$*) and one coupling constraint (0.3 between* $s2$ *and* $s10$*) were introduced. The optimal baseline planning for this example is shown in Table 7.3; for each sprint we report its complexity (i.e., the total number of story points for the stories it includes), its uncertainty risk (i.e., the overall additional story points arising from uncertain stories), and its cumulative utility. The integral* z_P *of the cumulative utility turns out to be 3474.5. A few remarks:*

- *The capacity constraint is always respected (for instance, for the first sprint,* $14 + 5.2 < 20$*).*
- *The uncertainty risk is well distributed over the first three sprints, but advanced to the first two sprints.*
- *The stories with higher utilities are advanced to the first sprint, also taking into account the coupling constraint and respecting the precedence from* $s1$ *to* $s2$*.*
- *The precedence from* $s7$ *to* $s6$ *is solved within the second sprint; these two stories have low utility and risk so they can be postponed.*

TABLE 7.3: Optimal baseline planning for the stories in Table 7.2

Sprint	*Stories*	*Complexity*	*Uncertainty Risk*	*Cumulative Utility*
♯1	$s1$, $s2$, $s3$, $s5$, $s10$	14	5.2	565.5
♯2	$s6$, $s7$, $s8$	13	5.5	866.0
♯3	$s9$	10	3.0	996.0
♯4	$s4$	1	0.7	1047.0
				$z_P = 3474.5$

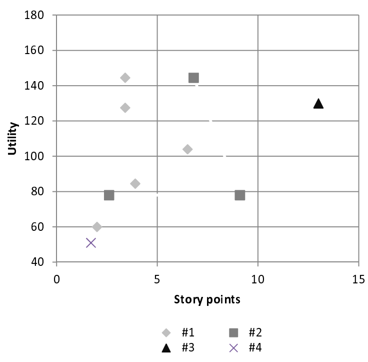


FIGURE 7.2: Sprint composition in function of the utility and complexity of user stories for the plan in Table 7.3

- *Story s9 is placed in the third sprint in spite of its high utility. In fact, if it were advanced to the first sprint it would take most of it, so it would become impossible to advance other stories with higher risk and still respect the precedences.*
- *The fourth sprint is not completely full; leaving some space in the last sprint is common in real projects because it allows for better managing unexpected events.*

The way stories are distributed in sprints according to their utilities and complexities is illustrated in Figure 7.2.

7.3.2 Smooth Replanning Optimization Model

As mentioned in Section 7.1, the project uncertainties and the inherent flexibility of iterative approaches often lead to some disruptions from the original baseline plan. We use the term smooth replanning to emphasize that the new plan delivered should limit as much as possible the changes made to the baseline plan; smoothness is important to protect the allocation of resources made to the projects and to preserve the milestones agreed with users.

Given the current optimal plan R (either the baseline plan or the result of a previous replanning), let U^{done} be the subset of the stories that were actually carried out at the end of sprint i, and U^{new} be the set of new stories that arose due to additional requirements. A new plan R' can be easily obtained by running again the optimization model for baseline planning on a new set of stories $U' = U - U^{done} \cup U^{new}$, and by adjusting the other variables and constraints accordingly; however, most probably, R and R' would be substantially different in the assignment of stories to sprints.

To add some smoothness to the replanning process, a proper *minimum perturbation strategy* must be adopted. Like done by [Alagoz and Azizoglu, 2003] we pursue a trade-off between effectiveness and stability, that are respectively measured by the objective function z_P and by the percentage α of stories that were scheduled in corresponding sprints in R and R'. In particular, we say a new plan R' is *dominant* when for each other possible plan R'' it is either $z_{P_{R''}} < z_{P_{R'}}$ or $\alpha_{R''} < \alpha_{R'}$. Picking one dominant plan means solving a bicriteria optimization problem, which can be done in two ways. The *hierarchical approach* minimizes the secondary (i.e., less important) criterion subject to the constraint that the value of the primary (more important) criterion is kept at its optimum. The *simultaneous approach* optimizes a weighted combination of the two criteria. We adopt a hierarchical approach since we argue that maximizing utility is definitely more important in the agile context. Furthermore, the use of a complex objective function would require a parameter-tuning step to achieve the desired trade-off.

More precisely, we extend the optimization model proposed in the previous subsection by adding a new constraint on *suggested stories*, that is, stories whose allocation into certain sprints is desirable but not mandatory:

$$\sum_{j \in T} x_{t_j j} \geq \alpha |T| \tag{7.11}$$

where $T \subseteq U$ is the subset of suggested stories, t_j for $j \in T$ is the sprint that should include story j, and α (*stability*) is the percentage of stories in T whose suggested allocation is to be respected.

This extended formulation can be used for smooth replanning by setting T to the set of stories that during the previous planning were scheduled to belong to sprints other than the current one, i.e., $T = U - U^{done}$. Noticeably, constraint (7.11) can also be used to deal with forced stories in a less prescriptive way; in fact, it can be seen as a relaxation of constraint (7.9).

Example 7.2. *Going on with Example 7.1, we suppose that, at the end of sprint ♯1, stories $s2$ and $s10$ were not completed and must be rescheduled. Smooth replanning is carried out with $T = \{s4, s6, s7, s8, s9\}$, which means that all the stories that were previously planned for sprints from ♯2 to ♯4 are suggested, while $s2$ and $s10$ can be freely allocated. By setting $\alpha = 0.8$, the team decides that at most one story in T can be disrupted ($|T| \times \alpha = 4$ stories out of 5 must be preserved). The new plan is shown in Table 7.4; sprint ♯1 is in gray since it is not actually part of the current plan and it has been reported for clarity. A few remarks:*

TABLE 7.4: New plan after smooth replanning with $\alpha = 0.8$

Sprint	*Stories*	*Complexity*	*Uncertainty Risk*	*Cumulative Utility*
♯1	$s1$, $s3$, $s5$	9	2.9	291.5
♯2	$s2$, $s6$, $s7$, $s10$	14	5.0	721.5
♯3	$s9$	10	3.0	851.5
♯4	$s4$, $s8$	5	3.5	1047.0
				$z_P = 2911.5$

- *No precedence constraint is posed on* $s2$ *since* $s1$ *has been carried out in sprint ♯1.*
- $s8$ *has been postponed since* $s2$ *and* $s10$ *bring a higher utility and they are affine.*
- *The reason why* $s8$ *has been postponed instead of* $s6$ *(that has lower utility) is to leave enough space (i.e., story points) in sprint ♯2 to contain both* $s2$ *and* $s10$*.*

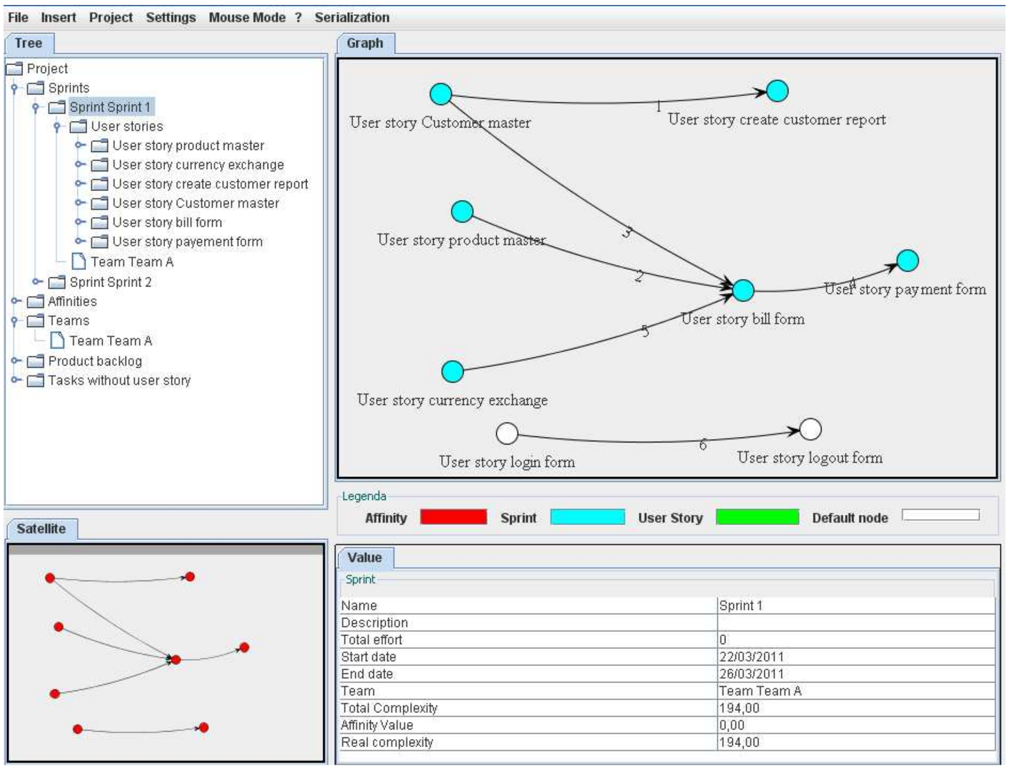
7.3.3 Implementation

In the market, different solutions for the agile project management are available. For example, *AgileFant* [Aalto University, SoberIT, 2011] offers a set of basic functionalities to monitor the progress of project iterations; *Mingle* [ThoughtWorks Studios, 2011] and *ScrumWorks* [Collabnet, 2011] provide a more complete set of agile parameters to deal with user story risk, complexity, and business value. However, all these tools lack in providing an automated solution to the multi-sprint optimization problem. We developed a stand-alone Java application that provides a graphical interface to collect the project specifications and automatically defines the optimization model that can be solved by IBM Ilog Cplex. Figure 7.3 shows the main interface of the software that allows users to set the different model parameters and manage user story precedence by using a graph representation.

7.3.4 Validation

7.3.4.1 Effectiveness Tests for Baseline Planning

To verify the effectiveness of our model we carried out a case study. According to the classification proposed by Runeson and Höst [2009], our case study can be described as *explanatory* (it aims at confirming the effectiveness of our optimization model in real contexts), *positivist* (it tests the quality of the optimal plan produced by our model), *quantitative* and *qualitative* (it quantitatively measures the quality of the optimal plan by computing the user story gap, but it also collects a qualitative judgment by the team manager), and *flexible* (the model parameters can change during the case study).



Project
- Sprints
  - Sprint Sprint 1
    - User stories
      - User story product master
      - User story currency exchange
      - User story create customer report
      - User story Customer master
      - User story bill form
      - User story payement form
    - Team Team A
  - Sprint Sprint 2
- Affinities
- Teams
  - Team Team A
- Product backlog
- Tasks without user story

Sprint

Name	Sprint 1
Description	
Total effort	0
Start date	22/03/2011
End date	26/03/2011
Team	Team Team A
Total Complexity	194,00
Affinity Value	0,00
Real complexity	194,00

FIGURE 7.3: The graphical interface for planning

A more complete description can be given by answering the basic questions proposed by Robson [2002]:

- *Objective—What to achieve?*: the case study aimed at proving the effectiveness of our approach to multi-sprint planning in the context of agile methods.
- *The case—What is studied?*: we studied two real projects with different characteristics and in different areas, namely, Web and PayTV; both projects were carried out by Italian companies that have been successfully adopting agile methods for several years.
- *Theory—Frame of reference*: the theoretical framework we adopted is the one defined by our model of planning and the related linear programming formulation.
- *Research questions—What to know?*: we studied how the optimal plan differs from the one manually produced by the project team in terms of sprint composition, risk distribution, and delivered utility.
- *Methods—How to collect data?*: for each project we collected data based on the static model of Figure 7.1 during a couple of meetings (with an overall duration of three hours) made *a posteriori* with the team; the estimates and constraints were collected via the user interface shown in Figure 7.3. There were no interactions with the team during the projects.

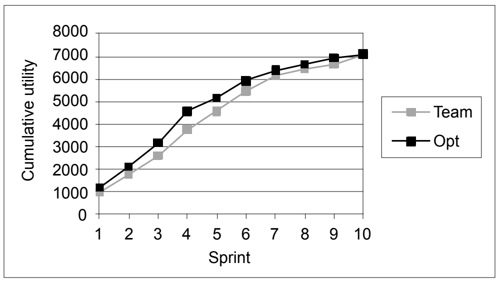


FIGURE 7.4: Comparison of cumulative utilities for the PayTV case study

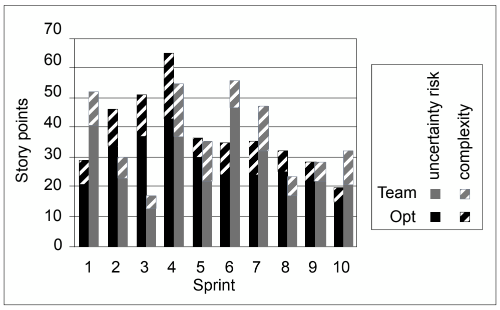


FIGURE 7.5: Comparison of risk distributions for the PayTV case study

- *Selection strategy—Where to seek data?*: we selected two different projects to cover all the aspects involved in multi-sprint planning. Web is a typical agile project on web applications, with a large set of user stories and a small number of precedences; PayTV has a smaller number of user stories but it includes a larger set of complex precedences and couplings. PayTV is the one we used for the 4WD validation.

PayTV includes 44 user stories and 52 precedences (mainly of AND type) and just one coupling constraint is involved. The development speed we used to run the optimization model is 2.43 story points per day and is empirically determined relying on historical data.

Figure 7.4 compares the cumulative utilities of the optimal plan (Opt) and of the plan defined by the team (Team). The curve of the optimal plan is always higher mainly due to a better optimization of sprint composition, but also to a better handling of risk. Indeed, in the teams plan some critical stories with low utility (essentially related to infrastructural needs) were advanced too much.

Figure 7.5 shows the distribution of story points among the different sprints for the two plans. Remarkably, the optimal plan achieves a uniform distribution, with a light advancing of risk to the first sprints.

The third comparison aims at measuring how the two plans differ in terms of sprint composition. The index we define to measure the difference between the two plans is the average of the gaps of all user stories, where the gap of a user story expresses the normalized lag of an optimally scheduled story relative to the team plan:

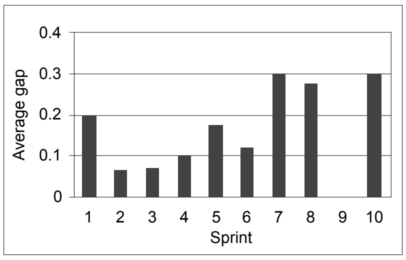


FIGURE 7.6: Difference in sprint composition between the optimal and the team plans for the PayTV case study

Definition 7.1 (User Story Gap)**.** Let j be a story. Let i^{team} and i^{opt} be the sprints j belongs to in the team plan and in the optimal plan, respectively. The *gap* of story j is

$$gap(j) = \frac{1}{N-1}|i^{team} - i^{opt}|$$

where N is the maximum number of sprints in the two plans.

The user story gap ranges from 0 to 1, where 0 means that the story belongs to the same sprint in both plans. As shown in Figure 7.6, the average gap is always lower then 0.3, denoting a good correspondence between the two plans. The main difference arises in sprints 1, 7, 8, and 10. In particular, in sprint 1, the team plan aimed at anticipating critical stories, thus exceeding the sprint capacity. The strong difference in the composition of the first sprint necessarily affected the subsequent sprints. Noticeably, both plans made a good use of couplings.

In order to have a further evaluation of the optimal plan, we discussed it with the team manager after the project end. Here are the main outcomes:

- The team spent a couple of days in defining their plan, while the optimal plan was generated in a few seconds.
- The team used to collect user story estimates using standard forms, but the level of detail required by our framework is slightly higher. This was perceived as a positive aspect since it leads to more refined estimates, thus producing a better plan. The graphical interface we provided was considered a valuable tool to support a deeper project understanding.
- The team manager recognized that his plan failed in properly distributing risks, which led to some delay in the first sprint.
- The optimal plan was judged to be feasible and realistic, showing that the elements considered in our model are sufficient to provide a good distribution of user stories.
- Most of the differences in sprint compositions were evaluated as improvements over the team plan. In particular, the team plan did not take into account the

side effects of postponing some stories, thus causing the stories depending on them to be delayed too much.

Web was aimed at developing a complex web site based on a Content Management System. It is larger than PayTV in terms of number of user stories (105 user stories); it was organized in 4 sprints of 10 days each, so it had a shorter overall duration (40 days). This difference is due to the lower complexity of the single user stories and to a higher development speed (6 story points per day). Compared to PayTV, Web includes a small number of chain precedences (6 overall) and no couplings. The input data were collected in 4 hours through an assessment with the whole project team, plus an extra session with the team manager who expressed some extra desiderata that had not emerged before:

- Web was the first project with a new customer; gaining its loyalty by delivering all the functionalities on time was a crucial goal of the project. Besides assigning each critical story an appropriate risk level, the team decided to anticipate some of them to the first sprint. This strategic decision goes beyond the typical development constraints; rather than modeling it by changing the risk parameters (i.e., the maximum values for r_j^{cr}), which could have undesired impacts on overall risk management, we explicitly forced the most complex user stories to the first sprint.
- Some of the requested functionalities come for free in the Content Management Systems, so they have no development complexity. Though they could be delivered in the first sprints from a technical point of view, they had better be postponed since the user cannot perceive their utility until correlated stories are completed. We modeled these specific constraints using chain precedences.

After running our optimization model we compared our solution with the baseline plan devised by the project team:

- The cumulative utility of the optimal plan is higher than the one obtained by the team (see Figure 7.7) and the team manager recognized that our solution is feasible and it has a better trade-off between utility and complexity.
- The user story gap (see Figure 7.8) is very low (less than 0.22 for each sprint) and is higher in the first sprint. As discussed with the team manager, two are the main motivations: (1) due to the lack of constraints and to the similar values for the utilities and complexity of user stories it was quite hard to manually define an optimal schedule; (2) the team was biased in its choices by the urge to completely deliver the first sprint, so it adopted an over-conservative solution.

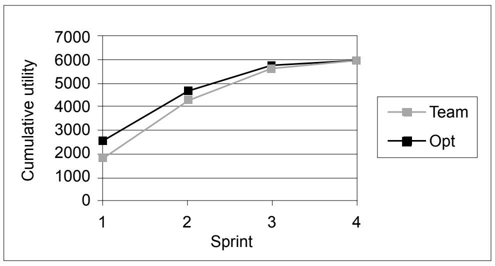


FIGURE 7.7: Comparison of cumulative utilities for the Web project.

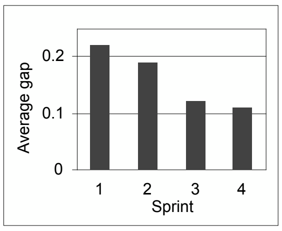


FIGURE 7.8: Difference in sprint composition between the optimal and the team plans for the Web project.

Overall, from an analysis of the two case studies it is apparent that not only our model returns an optimal schedule, but it is also flexible and expressive enough to handle projects with different characteristics (in terms of sprint features and constraints) and it can support team-specific desiderata.

7.3.4.2 Efficiency Tests for Baseline Planning

These tests were carried out on an Intel Core 2 Duo platform with 3 Gb of RAM, running at 3 GHz under Windows XP professional. To test the model behavior on a broad benchmark we generated a set of 58 synthetic projects; utility and story points of the user stories were randomized in the intervals [10,100] and [1,10], respectively. The maximum sprint duration was set to 15 days, while the development speed was set to 3 story points per day (i.e., sprint capacity was 45 story points). All problems were solved using IBM Ilog Cplex; performances were measured in seconds.

First of all we evaluate performances in function of the total number of user stories on projects that do not include precedences. Figure 7.9 reports the average time needed to compute the exact solution. As expected for a generalized assignment problem, the computation time grows non-linearly, reflecting an exponential increase in the search space. In Section 7.4, we will present sophisticated strategies to decrease the computational time for complex problems.

The presence of precedences makes planning harder for the project team. To study their impact on our model, two types of precedences were added to our benchmark projects: (1) *chain* precedences, where each story depends on at most another story; and (2) *graph* precedences, where a story can depend on several stories. In both cases

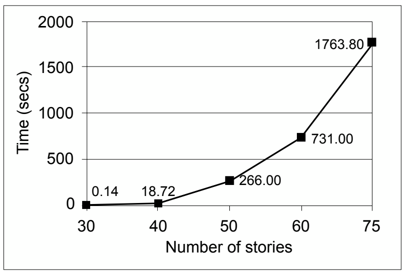


FIGURE 7.9: Time for computing the optimal plan for projects with an increasing number of stories and no precedences

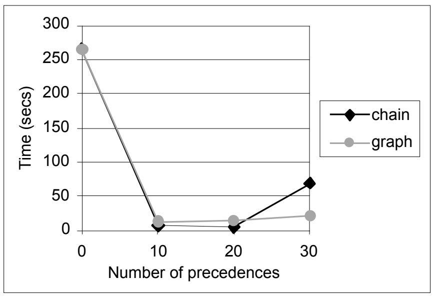
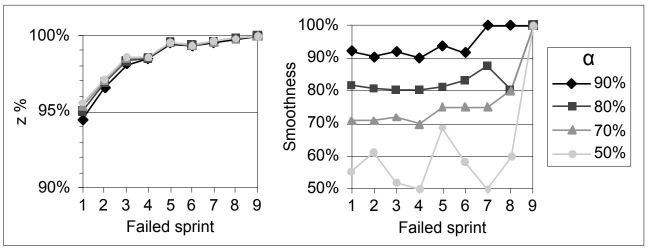


FIGURE 7.10: Time for computing the optimal plan for projects with an increasing number of precedences and 50 stories

precedences were obviously acyclic. Figure 7.10 shows how the computation time changes in function of the number of precedences. This figure suggests that a small number of precedences tends to reduce the computation time because precedences allow a set of unfeasible plans to be pruned, thus reducing the search space. However, when the number of precedences is high, the computation time increases again because finding a feasible plan becomes harder for the solver. Noticeably, both chain and graph precedences show similar trends.

7.3.4.3 Effectiveness Tests for Smooth Replanning

The effectiveness of smooth replanning can be evaluated by analyzing to what extent the previous plan is disrupted when a sprint partially fails, i.e., when it cannot deliver its expected results. To this end we considered a 50-story synthetic project and we measured the model performance when 33% of the user stories where not completed in one of its sprints. Figure 7.11.a shows how the value obtained for the objective function z of the new plan varies (as a percentage of the objective function value for the previous plan) in function of the sprint where the failure took place and of the stability α. As expected, due to the adoption of a cumulative objective function, the earlier the failure takes place, the worse its effects on z_P. Remarkably, if the failure takes place after the first sprint, the reduction in effectiveness is always less than 4% independently of the stability constraint.



(a) (b)

FIGURE 7.11: Percentage objective function (a) and smoothness (b) in function of the sprint where failure took place

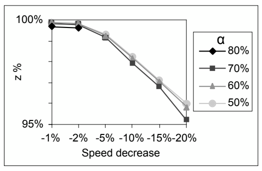


FIGURE 7.12: Percentage objective function in function of speed decrease

Figure 7.11.b illustrates how the actual smoothness (meant as the percentage of stories that are not disrupted after replanning) changes with α. Noticeably, only when $\alpha =$ 50% there are cases when less stories than the maximum allowed are disrupted; in all the other cases, the smoothness fluctuations are actually due to the rounding of the number of suggested stories (e.g., given 29 suggested stories, if $\alpha = 90\%$ then 2.9 stories can be disrupted; since user stories are atomic, only 2 of them can be actually moved to different sprints).

The effectiveness of smooth replanning can be also evaluated when intrinsic changes in the development process arise. In agile projects, during the review phase at the end of each sprint the development speed is estimated again, and it may be adjusted considering the feedback of past sprints and possible changes in the team composition. Then replanning is necessary to smoothly adapt the old plan to the new project parameters. An increase in speed implies an increase in the sprint capacities, that may lead to an earlier placement of useful stories. Conversely, a significant speed reduction could dramatically reduce sprint capacities, forcing a late delivery of high-valued user stories. In this case, the lower the stability α, the higher the probability that a good cumulative utility is preserved at the expense of smoothness. The trade-off between quality and stability is well illustrated by Figure 7.12, that shows how the objective function z of the new plan decreases with the development speed for different values of α (on the same 50-story project used in Figure 7.11).

7.3.4.4 Efficiency Tests for Smooth Replanning

Figure 7.13 shows the average execution time of the smooth replanning model on our 58-project benchmark. The computation time is always much lower than that of

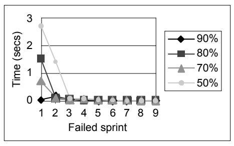


FIGURE 7.13: Time for replanning in function of the sprint where failure took place

baseline plans, because most of the user stories have already been assigned to sprints so that the search space is narrower.

7.4 Efficient Algorithms for the Multi-Sprint Planning Problem

As shown in Subsection 7.3.4, the computing time to solve to optimality medium-size instances can be very large. Here, we propose different strategies to improve the performance of our approach, namely *reductions*, *cover inequalities*, *dominance inequalities*, *greedy and exchange heuristics* and a *Lagrangian heuristic.*

7.4.1 Reductions

The reduction procedures try to strengthen the capacity constraints (7.2) of the baseline problem formulation by modifying either the sprint capacities or the weights $pr_j = p_j r_j^{un}$ of the user stories. Similar reductions are used for packing problems in [Boschetti and Montaletti, 2010, Boschetti and Mingozzi, 2003, Boschetti et al., 2002].

7.4.1.1 Modifying the Sprint Capacities

If no combination of user stories exactly filling the capacity p_i^{max} of sprint $i \in S$ exists, then there are useless story points that can be removed from the sprint capacity without modifying the optimal solution value. The capacity of a sprint i can be updated by solving the following subset sum problem:

$$p_i^{max} = \max\left\{\textstyle\sum_{k \in U} pr_k \xi_k : \sum_{k \in U} pr_k \xi_k \leq p_i^{max}, \xi_j \in \{0,1\}, j \in U\right\} \tag{7.12}$$

The subset sum problem can be solved using a simple dynamic programming procedure.

7.4.1.2 Modifying the Weights of Stories

It is straightforward to observe that a sprint containing the user story $j \in U$ remains feasible if the weight of j is increased to $pr_j = pr_j + (p_i^{max} - p_{ij}'')$, where p_{ij}'' is the optimal solution cost of the following subset sum problem:

$$p''_{ij} = \max\left\{p = \sum_{h\in U} pr_h \xi_h : p \leq p_i^{max}, \xi_j = 1, \xi_k \in \{0,1\}, k \in U \setminus \{j\}\right\} \tag{7.13}$$

Since we would like to maximize the number of updated weights, we heuristically consider the user stories ordered by non increasing weights, i.e., $pr_1 \geq pr_2 \geq \ldots \geq pr_n$.

7.4.2 Cover Inequalities

We have also investigated the classic *Lifted Cover Inequalities* (LCIs) corresponding to the capacity constraints (7.2), as done in the literature for the generalized assignment problem (see [Avella et al., 2010]).

We have tried to separate LCIs at each node of the tree search solving the required knapsack problems by a simple dynamic programming procedure. LCIs are usually able to reduce the number of tree search nodes, but, unfortunately, the average computing time to solve each tree node increases too much (see section 7.4.6).

7.4.3 Dominance Inequalities

Dominance inequalities can be applied to stories without couplings (i.e., for which $Y_j = \emptyset$) and only with some combinations of precedences.

Let $U^D = U^{OR} \cup U^{AND}$ be the set of stories having precedences and let $D = \bigcup_{j\in U}(D_j^{OR} \cup D_j^{AND})$ be the set of stories on which other stories depend. We define $U' = \{j \in U : Y_j = \emptyset \text{ and } j \notin U^D\}$ and $U'' = \{j \in U : Y_j = \emptyset \text{ and } j \notin D\}$. Furthermore, we define $ur_j = u_j r_j^{cr}$.

7.4.3.1 Dominance of Type 1

If there exists a pair of user stories $j \in U'$ and $j_1 \in U''$ such that $ur_j > ur_{j_1}$ and $pr_j = pr_{j_1}$, then the following inequalities hold:

$$\sum_{k=1}^{i-1} x_{kj_1} \leq 1 - x_{ij}, \text{ for every } i \in S \tag{7.14}$$

7.4.3.2 Dominance of Type 2

If there exists a triplet of user stories $j \in U'$ and $j_1, j_2 \in U''$ such that $ur_j > ur_{j_1} + ur_{j_2}$ and $pr_j = pr_{j_1} + pr_{j_2}$, then the following inequalities hold:

$$x_{i'j_1} + x_{i'j_2} + x_{ij} \leq 2, \text{ for every } i, i' \in S \text{ such that } i' < i \tag{7.15}$$

Similarly, if there exists a triplet of user stories $j \in U''$ and $j_1, j_2 \in U'$ such that $ur_j < ur_{j_1} + ur_{j_2}$ and $pr_j = pr_{j_1} + pr_{j_2}$, then the following inequalities hold:

$$x_{i'j_1} + x_{i'j_2} + x_{ij} \leq 2, \text{ for every } i, i' \in S \text{ such that } i < i' \tag{7.16}$$

7.4.3.3 Dominance of Type 3

If there exists a quadruplet of user stories $j \in U'$ and $j_1, j_2, j_3 \in U''$ such that $ur_j > ur_{j_1} + ur_{j_2} + ur_{j_3}$ and $pr_j = pr_{j_1} + pr_{j_2} + pr_{j_3}$, then the following inequalities hold:

$$x_{i'j_1} + x_{i'j_2} + x_{i'j_3} + x_{ij} \leq 3, \text{ for every } i, i' \in S \text{ such that } i' < i \tag{7.17}$$

Similarly, if there exists a quadruplet of user stories $j \in U''$ and $j_1, j_2, j_3 \in U'$ such that $ur_j < ur_{j_1} + ur_{j_2} + ur_{j_3}$ and $pr_j = pr_{j_1} + pr_{j_2} + pr_{j_3}$, then the following inequalities hold:

$$x_{i'j_1} + x_{i'j_2} + x_{i'j_3} + x_{ij} \leq 3, \text{ for every } i, i' \in S \text{ such that } i < i' \tag{7.18}$$

It is quite obvious that dominance inequalities can be easily generalized with respect a parameter $k \leq n$, if there exists at least a story j such that $pr_j = \sum_{i=1}^{k} pr_{j_i}$. However, computational results show that there are no benefits in spite of an increasing computational complexity.

7.4.4 Greedy and Exchange Heuristics

In this subsection we propose two greedy heuristics and a post-optimization procedure based on exchanges.

The first heuristc is based on a simple idea. Following a greedy approach, the procedure starts by optimizing sprint $i = 1$ and, then, optimizes the remaining sprints in turn, one at a time, following chronological order. Therefore, at each iteration, the greedy procedure considers a sprint $i \in S$ and assigns to it the stories that maximize the

Algorithm 9 Algorithm *GreedyHeuristic*

```
Input: Set $F_1 = \emptyset, i = 1$.
1: while $F_i \neq U$ and $i \leq m$ do
2:     Compute the optimal solution $\mathbf{x}^*$ of subproblem $SP_i$
3:     Set $F_{i+1} = F_i \cup \{j \in U : x^*_{ij} = 1\}$
4:     Set $i = i + 1$
5: end while
```

utility by solving the following subproblem:

$$(SP_i) \qquad z_{SP_i} = \max \sum_{j=1}^{n} (m-i+1) u_j \left(r_j^{cr} x_{ij} + a_j y_{ij} \right) \tag{7.19}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{max} \tag{7.20}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} \geq x_{ij} - |D_j \cap F_i|, \qquad j \in U^{OR} \tag{7.21}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} \geq x_{ij}|D_j| - |D_j \cap F_i|, \quad j \in U^{AND} \tag{7.22}$$

$$y_{ij} \leq \sum_{k \in Y_j} x_{ik}, \qquad j \in U \tag{7.23}$$

$$y_{ij} \leq |Y_j| x_{ij}, \qquad j \in U \tag{7.24}$$

$$x_{ij} \in \{0, 1\}, \qquad j \in U \setminus F_i \tag{7.25}$$

$$x_{ij} = 0, \qquad j \in F_i \tag{7.26}$$

$$y_{ij} \geq 0, \qquad i \in S, j \in U \tag{7.27}$$

where F_i represents the stories already assigned to sprints considered in the previous iterations (at the beginning, $F_1 = \emptyset$). Note that the stories F_i cannot be allocated to sprint i (see constraints (7.26)) and must be considered in precedence constraints (7.21) and (7.22).

The greedy heuristic, summarized in Algorithm 9, is very fast, as shown in section 7.4.6, but if it must be repeated many times, as in the Lagrangian heuristic described in section 7.4.5, the overall computing time can become too large. For example, in Section 7.4.6 we show that for some large instances *GreedyHeuristic* requires more than one second, thus if it is repeated for thousands of iterations, thousands of seconds are wasted just for the greedy. Therefore, we propose a modified greedy heuristic, where the subproblem SP_i is relaxed removing constraints (7.21)-(7.24) and the resulting subproblem, called SP'_i, is a knapsack problem which can be efficiently solved by dynamic programming. Unfortunately, the relaxed subproblem has two weaknesses. First, without the *linking* constraints (7.23) and (7.24), variables $\{y_{ij}\}$ are independent.

We solve this issue ignoring variables $\{y_{ij}\}$ when the knapsack problem SP'_i is solved and we *post*-evaluate them using the variables $\{x_{ij}\}$ corresponding to the solution of SP'_i (i.e., we set $y_{ij} = \sum_{k \in Y_j} x_{ik}$, if $x_{ij} = 1$, or $y_{ij} = 0$ otherwise). Second, without constraints (7.21) and (7.22), some precedences can be violated. Given the current sprint i and a user story j having a precedence constraint violated, to recover feasibility we propose four different strategies:

(i) Forbid the use of user story j in sprint i (i.e., fix $x_{ij} = 0$) and reoptimize the knapsack problem SP'_i.

(ii) Fix in the solution the user story $j' \in D^{OR}_j$ (or $j' \in D^{AND}_j$, depending on the violated constraint), not included in the current SP'_i solution, that maximizes the ratio $\frac{u_j r^{cr}_j}{p_j r^{un}_j}$. That is, fix $x_{ij'} = 1$ and reoptimize the knapsack problem SP'_i.

(iii) Only for the AND precedence constraints, fix in the solution all the user stories $j' \in D^{AND}_j$ not included in the current SP'_i solution (i.e., fix $x_{ij'} = 1$) and reoptimize the knapsack problem SP'_i. For the OR precedence constraints apply strategy (ii).

(iv) For every user story j define a coefficient κ_j to increase its profit in every knapsack problems SP'_i (i.e., the profit is multiplied by κ_j). For every user story $j' \in D^{OR}_j \setminus F_i$ (or $j' \in D^{AND}_j \setminus F_i$, depending on the violated constraint) having $x_{ij} = 0$ in the current solution, increase the coefficient $\kappa_{j'}$ using one of the following rules: (a) $\kappa_{j'} = \rho \times \kappa_{j'}$ or (b) $\kappa_{j'} = \kappa_{j'} \times \kappa_{j'}$. Restart the greedy heuristic from the first sprint, i.e., set $i = 1$ and $F_1 = \emptyset$.

Remarkably, the first strategy guarantees the convergence to a feasible solution, whereas the second and the third strategies may give rise to unfeasible knapsack instances, in particular the third one. When a knapsack problem SP'_i has not a feasible solution we skip to the next greedy heuristic iteration. The fourth strategy could require too many iterations to reach a feasible solution, therefore a maximum number of iterations *MaxIter* must be set.

The modified greedy procedure, called *QuickGreedyHeuristic*, is summarized in Algorithm 10. In our computational results we set *MaxIter* $= 1000$ and the fourth strategy is applied one time using rule (b) setting $\kappa_j = 1.025$, for every $j \in U$, and two times using rule (a) setting $\kappa_j = 1$, for every $j \in U$, and $\rho = 2$ or $\rho = 5$. The choice of parameters κ_j and ρ takes into account the trade-off between the time for obtaining a feasible solution and its quality. If some profits increase too quickly we obtain a feasible solution in a few iterations but probably its quality is poor (because we quickly move the corresponding user stories to the first sprints without taking enough care of their real utilities), whereas if profits increase too slowly we need too many iterations for obtaining a feasible solution.

Algorithm 10 Algorithm *QuickGreedyHeuristic*

```
Input: Set $z^* = -\infty$.
1: for all Strategy s=1,2,3,4 do
2:   Set $z' = 0$, $F_1 = \emptyset$, $i = 1$, $Iter = 0$.
3:   while $F_i \neq U$ and $i \leq m$ do
4:     while $\mathbf{x}'$ is not feasible and $SP_i'$ has a feasible solution and $Iter \leq MaxIter$ do
5:       Compute the optimal solution $\mathbf{x}'$ of knapsack problem $SP_i'$.
6:       if solution $\mathbf{x}'$ violates some precedence then
7:         Apply strategy $s$.
8:         if $s = 4$ then
9:           Set $z' = 0$, $F_1 = \emptyset$, $i = 1$, $Iter = Iter + 1$.
10:        end if
11:      end if
12:    end while
13:    if $SP_i'$ has not a feasible solution or $Iter > MaxIter$ then
14:      Set $z' = -\infty$ and $i = m + 1$.
15:    else
16:      Set $z' = z' + z_{SP_i}$.
17:      Set $F_{i+1} = F_i \cup \{j \in U : x_{ij}' = 1\}$.
18:      Set $i = i + 1$.
19:    end if
20:  end while
21:  if $z^* < z'$ then
22:    Set $z^* = z'$ and $\mathbf{x}^* = \mathbf{x}'$.
23:  end if
24: end for
```

Both algorithms *GreedyHeuristic* and *QuickGreedyHeuristic* may terminate without finding a feasible solution, because all the sprints are considered (i.e., $i > m$) but not all the user stories have been assigned to the available sprints (i.e., $F_i \neq U$). Moreover, a feasible solutions provided by *QuickGreedyHeuristic* can be usually further improved, because applying the precedence feasibility recovering strategies can generate non locally-optimal solution.

To improve a feasible solution, a local search based on exchanges can be applied. We propose an exchange heuristic, called *ExchangeHeuristic*, described in Algorithm 11. Procedure *ExchangeHeuristic* tries to exchange user stories between two sprints. Namely, the procedure tries the following exchanges:

1-0: move a story $j \in U$ from sprint i to sprint i';

1-1: exchange a story $j \in U$ executed in sprint i with a story $j' \in U$ executed in sprint i';

2-1: exchange two stories $j, j' \in U$ executed in sprint i with a story $j'' \in U$ executed in sprint i'.

Algorithm 11 Algorithm *ExchangeHeuristic*

```
1: Let $\mathbf{x}'$ be the solution to improve.
2: while no exchange occurs do
3:   Apply 1-0 exchanges
4:   for all sprint i=1,...,m-1 do
5:     for all sprint i'=i+1,...,m do
6:       for all $j \in U$ such that $x'_{i'j} = 1$ do
7:         if moving $j$ from $i'$ to $i$ is feasible and profitable then
8:           $x'_{i'j} = 0$ and $x'_{ij} = 1$
9:         end if
10:      end for
11:    end for
12:  end for
13:  Apply 1-1 exchanges
14:  for all sprint i=1,...,m-1 do
15:    for all sprint i'=i+1,...,m do
16:      for all $j, j' \in U$ such that $x'_{ij} = x'_{i'j'} = 1$ do
17:        if exchanging $j$ and $j'$ is feasible and profitable then
18:          $x'_{ij} = x'_{i'j'} = 0$ and $x'_{ij'} = x'_{i'j} = 1$
19:        end if
20:      end for
21:    end for
22:  end for
23:  // Apply 2-1 exchanges
24:  for all sprint i=1,...,m do
25:    for all sprint i'=1,...,m do
26:      for all $j, j', j'' \in U$ such that $x'_{ij} = x'_{ij'} = x'_{i'j''} = 1$ do
27:        if exchanging $j$ with $j'$ and $j''$ is feasible and profitable then
28:          $x'_{ij} = x'_{ij'} = x'_{i'j''} = 0$ and $x'_{i'j} = x'_{i'j'} = x'_{ij''} = 1$
29:        end if
30:      end for
31:    end for
32:  end for
33: end while
```

An exchange is performed only if it is *feasible* and *profitable.* It is feasible if, after the exchange, the capacity constraints of the corresponding sprints i and i' are still satisfied and precedence constraints are not violated. It is profitable if the overall objective function is increased.

Procedure *ExchangeHeuristic* could be extended adding other more complex exchanges, but in spite of the increasing computational complexity the improvements are usually negligible.

7.4.5 A Lagrangian Heuristic

The literature is rich with heuristics based on decomposition methods. An excellent introduction to the whole topic of Lagrangean relaxation, and of related heuristics, can be found in [Beasley, 1993, Boschetti and Maniezzo, 2009, Boschetti et al., 2009].

The Lagrangian relaxation is obtained from model P, described in section 7.3.1, by dualizing constraints (7.3), (7.4), (7.5), (7.6), and (7.7) by means of penalties $\{\lambda_j\}$, $\{\lambda_{ij}^{OR}\}$, $\{\lambda_{ij}^{AND}\}$, $\{\lambda_{ij}^{Y1}\}$, and $\{\lambda_{ij}^{Y2}\}$, respectively. Lagrangian penalties λ_j, $j \in U$, are unconstrained, whereas the remaining penalties are non-positive. The corresponding Lagrangian problem is the following:

$$(LR) \qquad z_{LR}(\boldsymbol{\lambda}) = \max \sum_{k=1}^{m} \sum_{i=1}^{k} \sum_{j=1}^{n} (u'_{ij}(\boldsymbol{\lambda}) x_{ij} + u''_{ij}(\boldsymbol{\lambda}) y_{ij}) + \sum_{j=1}^{n} \lambda_j \tag{7.28}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \le p_i^{max}, \qquad i \in S \tag{7.29}$$

$$x_{ij} \in \{0,1\}, \qquad i \in S, j \in U \tag{7.30}$$

$$0 \le y_{ij} \le |Y_j|, \qquad i \in S, j \in U \tag{7.31}$$

where the *penalized utilities* $u'_{ij}(\boldsymbol{\lambda})$ and $u''_{ij}(\boldsymbol{\lambda})$ are given by:

$$\begin{aligned} u'_{ij}(\boldsymbol{\lambda}) &= u_j r_j^{cr} - \lambda_j + \left(\lambda_{ij}^{OR} - \sum_{k=i}^{m} \sum_{j' \in \bar{D}_j^{OR}} \lambda_{kj'}^{OR} \right) + \\ &+ \left(|D_j^{AND}| \lambda_{ij}^{AND} - \sum_{k=i}^{m} \sum_{j' \in \bar{D}_j^{AND}} \lambda_{kj'}^{AND} \right) - \lambda_{ij}^{Y1} - |Y_j| \lambda_{ij}^{Y2} \\ u''_{ij}(\boldsymbol{\lambda}) &= u_j a_j + \lambda_{ij}^{Y1} + \lambda_{ij}^{Y2} \end{aligned} \tag{7.32}$$

where $\bar{D}_j^{OR} = \{j' \in U : j \in D_{j'}^{OR}\}$ and $\bar{D}_j^{AND} = \{j' \in U : j \in D_{j'}^{AND}\}$.

The Lagrangian problem LR can be decomposed into $2m$ independent subproblems, two for each sprint $i \in S$, as shown in the following:

$$(LR_i^1) \qquad z_{LR_i^1}(\boldsymbol{\lambda}) = \max \sum_{j=1}^{n} u'_{ij}(\boldsymbol{\lambda}) x_{ij} \tag{7.33}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \le p_i^{max} \tag{7.34}$$

$$x_{ij} \in \{0,1\}, \qquad j \in U \tag{7.35}$$

and

$$(LR_i^2) \qquad z_{LR_i^2}(\boldsymbol{\lambda}) = \max \sum_{j=1}^{n} u''_{ij}(\boldsymbol{\lambda}) y_{ij} \tag{7.36}$$

$$s.t.\ 0 \leq y_{ij} \leq |Y_j|,\ j \in U \tag{7.37}$$

Subproblem LR_i^1 is a knapsack problem whereas subproblem LR_i^2 can be easily solved by inspection (i.e., if $u''_{ij} > 0$, $y_{ij} = |Y_j|$, otherwise $y_{ij} = 0$). The overall optimal solution value of the Lagrangian problem LR is given by:

$$z_{LR}(\boldsymbol{\lambda}) = \sum_{i=1}^{m}(m-i+1)(z_{LR_i^1}(\boldsymbol{\lambda}) + z_{LR_i^2}(\boldsymbol{\lambda})) + \sum_{j=1}^{n} \lambda_j \tag{7.38}$$

that is a valid upper bound for the original problem P. In order to find the penalty vector $\boldsymbol{\lambda}^*$ that minimizes the upper bound $z_{LR}(\boldsymbol{\lambda})$ we must solve the *Lagrangian Dual* $z_{LR}(\boldsymbol{\lambda}^*) = \min_{\boldsymbol{\lambda}} \{z_{LR}(\boldsymbol{\lambda})\}$. This can be done heuristically by a subgradient algorithm [Shor, 1985], i.e., an iterative procedure that, at each iteration k, computes a new approximation $\boldsymbol{\lambda}^{k+1}$ of the Lagrangian multipliers in such a way that, for $k \to +\infty$, $\boldsymbol{\lambda}^k$ is an optimal or a near-optimal solution to the corresponding Lagrangian Dual.

Let $(\boldsymbol{x}, \boldsymbol{y})$ be the solution of cost $z_{LR}(\boldsymbol{\lambda})$ obtained at a given iteration by solving the Lagrangian problem LR. The Lagrangian multipliers can be updated as follows:

$$\begin{aligned}
\lambda_j &= \lambda_j + \alpha g_j, & j &\in U \\
\lambda_{ij}^{OR} &= \max\{0, \lambda_{ij}^{OR} + \alpha g_{ij}^{OR}\}, & i &\in S, j \in U^{OR} \\
\lambda_{ij}^{AND} &= \max\{0, \lambda_{ij}^{AND} + \alpha g_{ij}^{AND}\}, & i &\in S, j \in U^{AND} \\
\lambda_{ij}^{Y1} &= \max\{0, \lambda_{ij}^{Y1} + \alpha g_{ij}^{Y1}\}, & i &\in S, j \in U \\
\lambda_{ij}^{Y2} &= \max\{0, \lambda_{ij}^{Y2} + \alpha g_{ij}^{Y2}\}, & i &\in S, j \in U
\end{aligned} \tag{7.39}$$

where α is the length of the step along the search direction given by the subgradient $\boldsymbol{g}$ whose components are:

$$\begin{aligned}
g_j &= \sum_{i=1}^{m} x_{ij} - 1, & j &\in U \\
g_{ij}^{OR} &= \sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} - x_{ij}, & i &\in S, j \in U^{OR} \\
g_{ij}^{AND} &= \sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} - x_{ij}|D_j^{AND}|, & i &\in S, j \in U^{AND} \\
g_{ij}^{Y1} &= \sum_{k \in Y_j} x_{ik} - y_{ij}, & i &\in S, j \in U \\
g_{ij}^{Y2} &= |Y_j| x_{ij} - y_{ij}, & i &\in S, j \in U
\end{aligned} \tag{7.40}$$

Algorithm 12 Algorithm *ExchangeHeuristic*

```
Input: Set $\boldsymbol{\lambda} = \mathbf{0}$, $z^* = -\infty$
 1: while the subgradient end conditions are NOT satisfied do
 2:   Compute $z_{LR}(\boldsymbol{\lambda})$ solving the Lagrangian problem LR
 3:   Compute a heuristic solution $\mathbf{x}'$ with QuickGreedyHeuristic using
          the penalized utilities and improve $\mathbf{x}'$ with ExchangeHeuristic
 4:   Let $z'$ be the value of the improved solution $\mathbf{x}'$
 5:   if $\gamma z^* \leq z'$ then
 6:     Compute a heuristic solution $\mathbf{x}''$ with GreedyHeuristic
            using the penalized utilities
 7:     Let $z''$ be the value of solution $\mathbf{x}''$
 8:     if $z^* < z''$ then
 9:       $z^* = z''$ and $\mathbf{x}^* = \mathbf{x}''$
10:     end if
11:   end if
12:   if $z^* < z'$ then
13:     $z^* = z'$ and $\mathbf{x}^* = \mathbf{x}'$
14:   end if
15:   Update penalties $\boldsymbol{\lambda}$
16: end while
```

In our computational experiment $\alpha = \beta \frac{0.1 z_{LR}(\boldsymbol{\lambda})}{||\boldsymbol{g}||_2^2}$, where β is initialized with a value that is problem-dependent (in our case, $\beta = 3$) and, if after a given number of steps (in our case, 10) the solution value $z_{LR}(\boldsymbol{\lambda})$ is not improved, then β is reduced (in our case, $\beta = 0.85\beta$). The maximum number of iterations is 5000, but if within 50 iterations $z_{LR}(\boldsymbol{\lambda})$ is not improved by at least 0.01%, the subgradient algorithm is stopped in advance.

The heuristic procedure based on the proposed Lagrangian relaxation is summarized in Algorithm 12. At each iteration of the sugradient algorithm a heuristic solution $\mathbf{x}'$ is computed with procedure *QuickGreedyHeuristic* using the penalized utilities computed according to expression (7.32). The heuristic solution $\mathbf{x}'$ is further improved by procedure *ExchangeHeuristic*. The solution $\mathbf{x}'$ of value z' replaces the best solution found so far $\mathbf{x}^*$ if z' improves z^* (i.e., $z^* < z'$). Moreover, if $\gamma z^* \leq z'$ a new heuristic solution $\mathbf{x}''$ is also computed with the more expensive procedure *GreedyHeuristic*. If we choose $\gamma = 1$, we execute *GreedyHeuristic* only if $\mathbf{x}'$ is the best solution found so far, while if $\gamma < 1$, we execute *GreedyHeuristic* if z' has a percentage distance from the best value z^* within $100 \times (1 - \gamma)$. The solution $\mathbf{x}''$ of value z'' replaces the best solution found so far $\mathbf{x}^*$ if z'' improves z^* (i.e., $z^* < z''$).

7.4.6 Validation

The algorithms presented in this section have been executed on a workstation equipped with an Intel Xeon X7350 2.94 GHz, 16Gb of RAM and operating system Windows Server 2003 64bit. IBM Ilog Cplex 12.4 was used as the MIP solver.

We have used datasets coming from both real and synthetic projects. As to real projects, we used PayTv and Web; as to synthetic project, we implemented a generator that initially creates user stories by randomly assigning their utility, risk, and complexity. Then, it randomly adds groups of precedences organized either in chains or in graphs. Finally, sets of coupling stories are defined. We set the sprint capacity and the development velocity to 45 story points and 3 story points per day, respectively (i.e., each sprint takes 15 days).

Table 7.5 summarizes the key features of each project: the number n of stories; the maximum number m of sprints; the number n_{aff} of stories involved in at least one coupling; the cardinality of U^{OR} and U^{AND}; the maximum length l_{max} of groups of precedences; and the maximum number d_{max} of precedences involving a single user story. Projects are clustered into five groups: group A contains the real projects, while the projects in groups B and C show a mix of the previous parameters and vary in size, types of precedences, and presence of couplings. In Group D, the utility of stories is strongly correlated to their complexity (the complexity is always twice the utility); finally, the projects in group E are characterized by stories with high complexity so that each sprint can include at most 5 stories.

The computational experiments are reported in Tables 7.6, 7.7, 7.8, and 7.9, that include the following columns:

z : the value of the best feasible solution found by each algorithm;

Gap : the percentage gap between the best feasible solution and the upper bound associated to the best node remaining provided by IBM Ilog Cplex;

Nodes : the number of tree nodes generated by IBM Ilog Cplex;

Cuts : the number of valid inequalities added, using the IBM ILog Cplex *callbacks* (also constraints (7.4) and (7.5) are added in a cutting plane fashion and are included in this sum);

LGap : the percentage gap between the value z_{Heu} of the best feasible solution found and the upper bound z_{LR} provided by *LagrangianHeuristic*, i.e., **LGap** $= 100 \times \frac{z_{LR}-z_{Heu}}{z_{Heu}}$;

RGap : the percentage gap between the value z_{MIP} of the best feasible solution found by IBM Ilog Cplex and the value z_{Heu} found by *GreedyHeuristic* or *Lagrangian-Heuristic*, i.e., **RGap** $= 100 \times \frac{z_{MIP}-z_{Heu}}{z_{MIP}}$;

Time : the overall computing time in seconds.

In our computational tests we set a time limit of 600 seconds for the results reported in Tables 7.6 and 7.7, of 60 seconds for Table 7.8, and of 10 seconds for Table 7.9. When IBM Ilog Cplex does not find a feasible solution for an instance within the given time limit, we report the character "–" in columns z, *Gap*, and *RGap*.

Table 7.5: Problem instances

Group	**Proj. Name**	n	m	n_{aff}	$\lvert U^{OR} \rvert$	$\lvert U^{AND} \rvert$	l_{max}	d_{max}
A - Real	PayTV	44	12	2	8	27	6	5
	Web	104	6	5	0	4	4	1
B - Basic	25Chain-1	25	9	0	0	12	4	1
	25Graph-1	25	8	0	10	1	2	2
	25Affinity-1	25	9	6	5	5	2	2
	50Chain-1	50	12	0	0	20	4	1
	50Graph-1	50	11	0	8	10	2	2
	50Affinity-1	50	12	6	8	9	2	2
	75Chain-1	75	17	0	0	35	5	1
	75Graph-1	75	19	0	13	20	2	2
	75Affinity-1	75	17	6	17	13	2	3
	100Chain-1	100	20	0	0	40	5	1
	100Graph-1	100	23	0	20	14	2	3
	100Affinity-1	100	22	6	16	14	3	4
C - Basic	25Chain-2	25	8	0	0	12	2	1
	25Graph-2	25	8	0	3	8	3	2
	25Affinity-2	25	9	6	7	4	3	2
	50Chain-2	50	13	0	0	10	5	1
	50Graph-2	50	13	0	13	8	4	5
	50Affinity-2	50	13	6	13	9	3	3
	75Chain-2	75	17	0	0	36	3	1
	75Graph-2	75	18	0	14	16	2	2
	75Affinity-2	75	18	6	17	13	2	2
	100Chain-2	100	22	0	0	30	5	1
	100Graph-2	100	22	0	12	15	5	7
	100Affinity-2	100	22	6	11	14	3	2
D - Correlated	25Chain-3	25	15	0	0	12	4	1
	25Graph-3	25	15	0	5	7	5	4
	25Affinity-3	25	13	6	5	4	2	2
	50Chain-3	50	22	0	0	20	4	1
	50Graph-3	50	22	0	7	9	2	2
	50Affinity-3	50	21	6	9	6	2	2
	75Chain-3	75	31	0	0	36	6	1
	75Graph-3	75	29	0	12	17	2	2
	75Affinity-3	75	33	6	20	7	2	2
	100Chain-3	100	38	0	0	40	8	1
	100Graph-3	100	40	0	16	14	3	3
	100Affinity-3	100	43	6	16	13	2	2
B - Few	25Chain-4	25	12	0	5	7	4	1
	25Graph-4	25	13	0	5	6	5	4
	25Affinity-4	25	14	6	3	7	2	2
	50Chain-4	50	21	0	9	11	4	1
	50Graph-4	50	21	0	2	13	2	2
	50Affinity-4	50	21	6	6	8	2	2
	75Chain-4	75	27	0	15	21	6	1
	75Graph-4	75	29	0	12	12	2	4
	75Affinity-4	75	29	6	10	12	2	2
	100Chain-4	100	40	0	24	21	9	1
	100Graph-4	100	40	0	8	17	2	2
	100Affinity-4	100	39	6	18	10	2	2

Table 7.6: Results obtained solving the basic model with IBM Ilog Cplex and adding valid inequalities

	Basic Model					**Basic Model + DIs**					**Basic Model + DIs + LCIs**				
Name	z	**Gap**	**Nodes**	**Cuts**	**Time**	z	**Gap**	**Nodes**	**Cuts**	**Time**	z	**Gap**	**Nodes**	**Cuts**	**Time**
PayTV	93330.0	0.00	130908	238	50.21	93330.0	0.00	130908	238	50.71	93330.0	0.00	41573	3384	96.03
Web	32683.6	0.00	0	2	0.30	32683.6	0.00	0	2	0.30	32683.6	0.00	0	3	0.30
25Chain-1	16627.2	0.00	3747	52	0.80	16627.2	0.00	2989	56	0.72	16627.2	0.00	2790	450	1.15
25Graph-1	13515.1	0.00	601	9	0.19	13515.1	0.00	601	9	0.19	13515.1	0.00	528	93	0.20
25Affinity-1	18523.6	0.00	1223	21	0.87	18523.6	0.00	1223	21	0.89	18523.6	0.00	875	136	1.01
50Chain-1	41244.8	0.05	2284200	150	600.26	41206.6	0.22	2180919	291	600.94	41244.8	0.13	129462	7224	600.06
50Graph-1	34686.0	0.00	55654	43	16.41	34686.0	0.00	208673	103	57.63	34686.0	0.00	176231	3920	300.57
50Affinity-1	40545.1	0.00	3486	46	2.50	40545.1	0.00	2993	68	2.33	40545.1	0.00	699	244	1.11
75Chain-1	88713.3	0.91	874300	369	600.22	87879.6	1.96	616962	1202	600.47	88754.5	1.24	43100	11288	600.35
75Graph-1	92959.2	0.23	873145	188	600.27	92911.4	0.32	548418	1517	600.75	92952.8	0.29	42113	13028	600.39
75Affinity-1	76076.2	0.00	165046	80	193.37	76076.2	0.00	162370	359	191.02	76076.2	0.00	88979	8904	508.81
100Chain-1	136240.7	0.96	608866	529	600.42	135965.9	1.40	306300	2758	600.68	136159.3	1.65	34401	13408	600.29
100Graph-1	149517.0	0.23	647382	173	600.40	149453.7	0.28	277600	2600	600.99	149462.2	0.28	50300	12315	600.63
100Affinity-1	136008.3	0.25	281057	138	600.26	135975.9	0.29	307700	2222	600.25	135965.8	0.31	54300	11950	600.20
25Chain-2	13214.8	0.00	212	30	0.17	13214.8	0.00	230	29	0.19	13214.8	0.00	128	68	0.16
25Graph-2	17202.6	0.00	199	19	0.17	17202.6	0.00	223	23	0.17	17202.6	0.00	355	83	0.20
25Affinity-2	13007.4	0.00	199	26	0.22	13007.4	0.00	199	30	0.20	13007.4	0.00	128	94	0.20
50Chain-2	46629.1	0.00	29211	71	10.84	46629.1	0.00	109674	170	33.10	46629.1	0.00	31345	2818	82.32
50Graph-2	37699.9	0.00	375337	103	116.81	37699.9	0.00	373877	238	126.02	37699.9	0.00	14907	2868	36.78
50Affinity-2	46153.1	0.00	34993	112	35.09	46153.1	0.00	78374	230	73.60	46153.1	0.00	25176	2509	46.66
75Chain-2	78754.0	1.62	829001	476	600.42	79260.6	0.97	593600	831	600.38	79175.2	1.48	40277	12123	600.35
75Graph-2	72519.7	0.24	1026580	116	600.34	72569.1	0.23	814000	1069	600.85	72525.0	0.29	50721	13383	600.59
75Affinity-2	84372.4	0.06	314700	133	600.15	84284.1	0.17	342500	461	600.22	84274.4	0.28	51029	13626	600.11
100Chain-2	134485.9	0.26	654377	419	600.40	–	–	272000	3873	601.25	133814.5	0.97	34311	13574	600.50
100Graph-2	134975.2	0.31	644156	170	600.46	134866.8	0.41	296500	3038	601.15	135023.3	0.28	49300	12415	600.71
100Affinity-2	136435.5	0.25	301800	144	600.29	136447.8	0.25	329765	4076	600.26	136471.8	0.23	75570	11976	600.21
25Chain-3	2855.2	0.00	26378	122	12.79	2855.2	0.00	48121	128	21.59	2855.2	0.00	3940	467	5.21
25Graph-3	1935.6	0.00	59936	89	13.96	1935.7	0.00	49015	144	11.89	1935.7	0.00	4243	526	3.87
25Affinity-3	2002.3	0.00	4538	45	5.41	2002.3	0.00	3217	49	4.01	2002.3	0.00	2132	275	3.93
50Chain-3	6199.7	2.21	374400	318	600.20	6156.4	3.13	228157	1525	600.48	6192.1	2.51	42696	5807	600.31
50Graph-3	5944.0	1.06	936113	254	600.29	5933.4	1.39	333380	3733	601.12	5944.3	1.13	60300	6905	600.52
50Affinity-3	5224.9	0.81	169000	150	600.28	5229.6	0.73	276128	1440	600.18	5220.4	1.05	64117	6808	600.13
75Chain-3	12206.2	2.71	274600	901	600.35	12056.2	4.08	175200	2308	600.51	12156.2	3.64	21386	10584	600.29
75Graph-3	11376.9	1.05	744845	566	600.60	11365.3	1.18	198676	5382	600.98	11355.6	1.28	31400	9151	600.43
75Affinity-3	14415.6	1.92	91417	280	600.19	14366.6	2.32	88000	3926	600.12	–	–	28150	9459	600.09
100Chain-3	–	–	188586	1247	600.65	–	–	90828	5960	600.70	–	–	9591	8884	600.28
100Graph-3	19952.5	0.80	431293	329	600.51	–	–	96716	6466	601.03	–	–	20170	13436	600.46
100Affinity-3	25412.8	2.96	41960	712	600.31	–	–	39039	1620	600.57	–	–	16349	6570	600.53
25Chain-4	18240.4	0.00	3230	94	1.37	18240.4	0.00	4984	98	1.79	18240.4	0.00	1381	554	1.69
25Graph-4	20561.7	0.00	21922	79	6.49	20561.7	0.00	17496	78	5.41	20561.7	0.00	4500	556	4.35
25Affinity-4	20308.8	0.00	708	57	1.81	20308.8	0.00	699	57	1.81	20308.8	0.00	1031	171	2.39
50Chain-4	60331.1	1.59	868901	295	600.26	60451.9	1.55	625123	936	600.62	60485.2	1.69	52008	5794	600.40
50Graph-4	66335.3	0.75	936300	179	600.32	66394.0	0.83	781206	594	600.60	66394.0	0.61	97300	4458	600.51
50Affinity-4	60073.0	0.04	218987	59	600.09	60073.0	0.09	200625	236	600.23	60006.5	0.31	94500	4147	600.16
75Chain-4	110688.1	5.86	487300	774	600.67	113571.4	3.35	328700	1247	600.56	111994.3	5.25	26277	11272	600.35
75Graph-4	130342.6	1.26	354591	229	600.32	130590.7	1.12	326200	1501	600.79	130230.8	1.42	41529	8534	600.50
75Affinity-4	124913.5	1.22	91800	230	600.14	124825.3	1.15	112200	918	600.18	124852.5	1.30	40300	9515	600.10
100Chain-4	–	–	145225	1346	600.46	–	–	133782	1952	600.63	–	–	13034	11503	600.28
100Graph-4	246939.9	0.86	221700	278	600.47	246754.3	0.98	158700	2655	600.90	–	–	26140	11014	600.45
100Affinity-4	237587.9	0.92	102582	282	600.40	237691.5	0.91	91800	2236	600.25	237406.5	1.08	28851	10859	600.15

Table 7.7: Comparison among IBM Ilog Cplex, *GreedyHeuristic*, and *LagrangianHeuristic*

	IBM Ilog Cplex			**GreedyHeuristic**			**LagrangianHeuristic** ($\gamma = 1$)				**LagrangianHeuristic** ($\gamma = 0.995$)			
Name	z	**Gap**	**Time**	z	**RGap**	**Time**	z	**LGap**	**RGap**	**Time**	z	**LGap**	**RGap**	**Time**
PayTV	93330.0	0.00	50.21	77994.0	16.43	0.12	92646.0	6.01	0.73	19.85	92646.0	6.01	0.73	25.42
Web	32683.6	0.00	0.30	32572.9	0.34	0.03	32683.6	0.41	0.00	99.57	32683.6	0.41	0.00	99.66
25Chain-1	16627.2	0.00	0.80	16454.7	1.04	0.28	16613.6	1.41	0.08	0.53	16617.1	1.39	0.06	2.79
25Graph-1	13515.1	0.00	0.19	13514.7	0.00	0.05	13515.1	0.95	0.00	0.17	13515.1	0.95	0.00	1.34
25Affinity-1	18523.6	0.00	0.87	18171.0	1.90	0.06	18505.1	1.98	0.10	0.83	18505.1	1.98	0.10	3.90
50Chain-1	41244.8	0.05	600.26	40992.8	0.61	0.14	41103.9	1.79	0.34	4.52	41103.9	1.79	0.34	12.92
50Graph-1	34686.0	0.00	16.41	34686.0	0.00	0.08	34686.0	2.20	0.00	1.70	34686.0	2.20	0.00	3.98
50Affinity-1	40545.1	0.00	2.50	40373.3	0.42	0.08	40545.1	0.19	0.00	6.38	40545.1	0.19	0.00	14.63
75Chain-1	88713.3	0.91	600.22	88470.2	0.28	0.27	88470.2	3.20	0.28	46.97	88522.0	3.14	0.22	49.39
75Graph-1	92959.2	0.23	600.27	92595.2	0.39	0.16	92844.1	1.59	0.12	63.99	92892.8	1.53	0.07	99.26
75Affinity-1	76076.2	0.00	193.37	75757.2	0.42	0.25	75876.8	0.55	0.26	35.63	75897.9	0.52	0.24	62.14
100Chain-1	136240.7	0.96	600.42	135991.8	0.18	0.69	135991.8	3.09	0.18	73.67	135991.8	3.09	0.18	73.82
100Graph-1	149517.0	0.23	600.40	148993.6	0.35	0.28	149452.0	1.46	0.04	123.58	149452.0	1.46	0.04	162.57
100Affinity-1	136008.3	0.25	600.26	135470.9	0.40	0.38	135910.4	1.31	0.07	158.93	135910.4	1.31	0.07	221.44
25Chain-2	13214.8	0.00	0.17	13214.8	0.00	0.05	13214.8	0.43	0.00	0.58	13214.8	0.43	0.00	2.51
25Graph-2	17202.6	0.00	0.17	17202.6	0.00	0.03	17202.6	1.09	0.00	0.34	17202.6	1.09	0.00	2.12
25Affinity-2	13007.4	0.00	0.22	12700.3	2.36	0.05	13007.4	1.86	0.00	0.75	13007.4	1.86	0.00	4.62
50Chain-2	46629.1	0.00	10.84	46500.7	0.28	0.09	46545.8	0.85	0.18	3.00	46545.8	0.85	0.18	7.47
50Graph-2	37699.9	0.00	116.81	37693.0	0.02	0.11	37693.0	3.15	0.02	7.74	37693.0	3.15	0.02	8.11
50Affinity-2	46153.1	0.00	35.09	45766.6	0.84	0.19	45995.6	2.54	0.34	7.54	45995.6	2.54	0.34	10.87
75Chain-2	78754.0	1.62	600.42	78816.4	-0.08	0.62	78816.4	4.50	-0.08	29.38	78816.4	4.50	-0.08	30.55
75Graph-2	72519.7	0.24	600.34	72408.5	0.15	0.16	72408.5	5.54	0.15	39.84	72408.5	5.54	0.15	44.27
75Affinity-2	84372.4	0.06	600.15	83417.6	1.13	0.23	83758.7	4.90	0.73	68.36	83758.7	4.90	0.73	80.67
100Chain-2	134485.9	0.26	600.40	134217.7	0.20	0.86	134217.7	1.25	0.20	49.30	134217.7	1.25	0.20	56.18
100Graph-2	134975.2	0.31	600.46	134871.6	0.08	0.31	134871.6	1.03	0.08	124.69	134871.6	1.03	0.08	181.13
100Affinity-2	136435.5	0.25	600.29	136171.7	0.19	0.47	136329.0	0.76	0.08	106.38	136329.0	0.76	0.08	171.79
25Chain-3	2855.2	0.00	12.79	2821.9	1.17	0.13	2855.2	0.67	0.00	0.72	2855.2	0.67	0.00	4.59
25Graph-3	1935.6	0.00	13.96	1862.2	3.79	0.11	1919.4	5.03	0.84	0.50	1919.4	5.03	0.84	3.09
25Affinity-3	2002.3	0.00	5.41	1976.3	1.30	0.13	1994.9	2.86	0.37	0.89	1994.9	2.86	0.37	1.45
50Chain-3	6199.7	2.21	600.20	6117.0	1.33	0.62	6167.1	4.07	0.53	7.05	6167.1	4.07	0.53	10.36
50Graph-3	5944.0	1.06	600.29	5918.4	0.43	0.75	5918.4	2.53	0.43	4.71	5918.4	2.53	0.43	8.05
50Affinity-3	5224.9	0.81	600.28	5149.2	1.45	0.44	5210.2	3.66	0.28	8.14	5210.2	3.66	0.28	23.76
75Chain-3	12206.2	2.71	600.35	11830.6	3.08	1.39	12125.9	6.32	0.66	47.24	12125.9	6.32	0.66	75.10
75Graph-3	11376.9	1.05	600.60	11305.3	0.63	1.83	11375.4	8.18	0.01	7.30	11375.4	8.18	0.01	46.04
75Affinity-3	14415.6	1.92	600.19	14226.8	1.31	1.17	14436.8	4.00	-0.15	13.00	14436.8	4.00	-0.15	72.61
100Chain-3	–	–	600.65	19336.0	–	2.40	19337.9	5.15	–	66.10	19337.9	5.15	–	73.01
100Graph-3	19952.5	0.80	600.51	19859.0	0.47	2.40	19925.1	3.30	0.14	85.24	19925.1	3.30	0.14	280.38
100Affinity-3	25412.8	2.96	600.31	24967.7	1.75	2.61	25371.2	5.60	0.16	53.35	25371.2	5.60	0.16	74.46
25Chain-4	18240.4	0.00	1.37	17500.5	4.06	0.16	18240.4	3.43	0.00	1.97	18240.4	3.43	0.00	2.43
25Graph-4	20561.7	0.00	6.49	20237.9	1.58	0.16	20551.4	5.37	0.05	0.89	20551.4	5.37	0.05	3.28
25Affinity-4	20308.8	0.00	1.81	18820.9	7.33	0.19	20286.8	6.57	0.11	0.52	20286.8	6.57	0.11	3.25
50Chain-4	60331.1	1.59	600.26	59741.3	0.98	0.47	60033.9	4.67	0.49	9.77	60033.9	4.67	0.49	18.91
50Graph-4	66335.3	0.75	600.32	66166.2	0.26	0.83	66166.2	4.11	0.26	7.97	66166.2	4.11	0.26	10.78
50Affinity-4	60073.0	0.04	600.09	59856.4	0.36	0.52	59912.3	5.12	0.27	13.66	59912.3	5.12	0.27	65.02
75Chain-4	110688.1	5.86	600.67	111538.6	-0.77	1.34	112185.6	8.52	-1.35	67.13	112188.4	8.51	-1.35	91.09
75Graph-4	130342.6	1.26	600.32	128400.2	1.49	1.39	129963.0	5.11	0.29	23.82	129963.0	5.11	0.29	57.67
75Affinity-4	124913.5	1.22	600.14	123198.2	1.37	1.29	124196.3	5.54	0.58	31.48	124196.3	5.54	0.58	62.93
100Chain-4	–	–	600.46	212681.5	–	2.96	216381.8	9.06	–	202.88	216381.8	9.06	–	231.12
100Graph-4	246939.9	0.86	600.47	246505.4	0.18	1.97	246511.5	4.37	0.17	53.60	246511.5	4.37	0.17	198.41
100Affinity-4	237587.9	0.92	600.40	234254.3	1.40	2.29	236779.7	5.84	0.34	53.02	236779.7	5.84	0.34	131.12

Table 7.6 shows the computational results obtained solving with IBM Ilog Cplex the basic model (7.1)–(7.10) proposed in section 7.3.1, and adding to this model first the Dominance Inequalities (DIs) described in subsection 7.4.3, and then also the Lifted Cover Inequalities (LCIs) described in subsection 7.4.2. Many instances are not solved to optimality by IBM Ilog Cplex within the given time limit of 600 seconds (30 out of 50 instances). In particular, for instances "*100Chain-3*" and "*100Chain-4*" IBM Ilog Cplex cannot find a feasible solution. When we add the DIs and LCIs to the basic model, the results do not improve on average; on the contrary, often IBM Ilog Cplex generates worse solutions and only for some instances the results are improved. The basic model with DIs performs better for 15 out of 50 instances (e.g., for instances "*75Chain-2*", "*25Affinity-3*", "*75Chain-4*", etc.). The basic model with DIs and LCIs performs better only for 13 out of 50 instances (e.g., for instances "*50Graph-2*", "*25Chain-3*", etc.).

These results show that DIs and LCIs are usually able to reduce the number of tree nodes, but the cost for separating the inequalities and solving the increased model are not repayed. However, sometimes the added inequalities increase the number of tree nodes, in particular for DIs, probably because they induce an increasing number of fractional variables. Not reported in our computational results, the contribution of the reduction procedures is negligible. Probably, as in cutting problems, they are effective only for those instances where only a few user stories can be executed at each sprint.

Table 7.7 provides a comparison between IBM Ilog Cplex applied to the basic model, procedure *GreedyHeuristic* presented in section 7.4.4, and *LagrangianHeuristic* presented in section 7.4.5. For *LagrangianHeuristic* we perform two computational tests with two different settings of parameter γ. In the first setting it is $\gamma = 1$, therefore at each subgradient iteration the more expensive procedure *GreedyHeuristic* is performed only if *QuickGreedyHeuristic* and *ExchangeHeuristic* provide the best feasible solution computed so far. In the second setting it is $\gamma = 0.995$, therefore *GreedyHeuristic* is performed if *QuickGreedyHeuristic* and *ExchangeHeuristic* provide a feasible solution whose value is at least 99.5% the current best solution value. Procedure *GreedyHeuristic* can solve each instance very quickly, but sometimes it yields unsatisfactory solutions whose value is even 16.43% or 7.33% worse than the best solution value found by IBM Ilog Cplex for instances PayTV and "25Affinity-4", respectively. *LagrangianHeuristic* computes solutions of better quality instead; with $\gamma = 0.995$ it always generates solutions whose maximum gap from the best feasible solution is under 1%, and for 5 instances it outperforms IBM Ilog Cplex. *LagrangianHeuristic* requires a larger computing time when $\gamma = 0.995$ with respect to $\gamma = 1$, because it executes procedure *GreedyHeuristic* a larger number of times.

Tables 7.8 and 7.9 compare *LagrangianHeuristic* with IBM ILog Cplex setting the time limit to 60 and 10 seconds, respectively. In these computational tests we set $\gamma = 1$. The results show that *LagrangianHeuristic* finds very good-quality solutions in a very short time with respect to IBM ILog Cplex. In fact, Table 7.8 shows that, setting a

TABLE 7.8: Comparison between IBM ILog Cplex and *LagrangianHeuristic* setting a time limit of 60 secs

	IBM Ilog Cplex			**LagrangianHeuristic**		
Name	z	**Gap**	**Time**	z	**RGap**	**Time**
PayTV	93330.0	0.00	51.01	92646.0	0.73	20.34
Web	32683.6	0.00	0.30	32683.6	0.00	60.31
25Chain-1	16627.2	0.00	0.80	16613.6	0.08	0.52
25Graph-1	13515.1	0.00	0.19	13515.1	0.00	0.17
25Affinity-1	18523.6	0.00	0.87	18505.1	0.10	0.86
50Chain-1	41182.3	0.46	60.14	41103.9	0.19	4.49
50Graph-1	34686.0	0.00	16.60	34686.0	0.00	1.72
50Affinity-1	40545.1	0.00	2.48	40545.1	0.00	6.58
75Chain-1	84711.8	6.34	60.08	88470.2	-4.44	47.49
75Graph-1	92844.7	0.39	60.11	92844.1	0.00	60.06
75Affinity-1	76076.2	0.03	60.06	75876.8	0.26	35.85
100Chain-1	–	–	60.22	135991.8	–	60.11
100Graph-1	149350.9	0.35	60.20	149452.0	-0.07	60.45
100Affinity-1	135934.8	0.33	60.15	135910.4	0.02	60.40
25Chain-2	13214.8	0.00	0.16	13214.8	0.00	0.59
25Graph-2	17202.6	0.00	0.16	17202.6	0.00	0.34
25Affinity-2	13007.4	0.00	0.20	13007.4	0.00	0.75
50Chain-2	46629.1	0.00	10.70	46545.8	0.18	2.96
50Graph-2	37699.9	0.06	60.15	37693.0	0.02	7.84
50Affinity-2	46153.1	0.00	34.67	45995.6	0.34	7.65
75Chain-2	72821.8	10.75	60.10	78816.4	-8.23	29.25
75Graph-2	72515.2	0.28	60.11	72408.5	0.15	39.86
75Affinity-2	84198.7	0.37	60.08	83758.7	0.52	60.09
100Chain-2	133767.7	0.94	60.17	134217.7	-0.34	49.66
100Graph-2	134900.3	0.37	60.22	134871.6	0.02	60.67
100Affinity-2	136355.8	0.32	60.19	136329.0	0.02	60.45
25Chain-3	2855.2	0.00	12.40	2855.2	0.00	0.73
25Graph-3	1935.6	0.00	13.65	1919.4	0.84	0.48
25Affinity-3	2002.3	0.00	5.20	1994.9	0.37	0.87
50Chain-3	6221.7	2.07	60.06	6167.1	0.88	6.88
50Graph-3	–	–	60.08	5918.4	–	4.55
50Affinity-3	5194.6	1.78	60.08	5210.2	-0.30	8.10
75Chain-3	–	–	60.06	12125.9	–	46.75
75Graph-3	11316.4	1.62	60.12	11375.4	-0.52	6.46
75Affinity-3	14290.4	2.98	60.09	14436.8	-1.02	12.17
100Chain-3	–	–	60.23	19337.9	–	60.23
100Graph-3	19897.1	1.10	60.22	19925.1	-0.14	60.08
100Affinity-3	–	–	60.22	25371.2	–	53.01
25Chain-4	18240.4	0.00	1.25	18240.4	0.00	1.89
25Graph-4	20561.7	0.00	6.33	20551.4	0.05	0.83
25Affinity-4	20308.8	0.00	1.73	20286.8	0.11	0.44
50Chain-4	60490.4	1.74	60.08	60033.9	0.76	9.20
50Graph-4	66262.9	1.00	60.09	66166.2	0.15	7.47
50Affinity-4	59890.1	0.63	60.06	59912.3	-0.04	13.01
75Chain-4	111280.0	5.85	60.08	112185.6	-0.81	60.09
75Graph-4	129958.9	1.74	60.09	129963.0	-0.00	23.01
75Affinity-4	124616.6	1.64	60.06	124196.3	0.34	31.17
100Chain-4	–	–	60.19	216381.8	–	60.22
100Graph-4	244866.7	1.85	60.15	246511.5	-0.67	52.04
100Affinity-4	236642.0	1.45	60.14	236779.7	-0.06	51.85

time limit of 60 seconds, IBM ILog Cplex generates a worse solution for 19 out of 50 instances and, in particular, it cannot find a feasible solution for 6 out of 50 instances. Setting a time limit of 10 seconds, the situation worsens further for IBM Ilog Cplex. Table 7.9 shows that IBM ILog Cplex generates a worse solution for 24 out of 50 instances and it cannot find a feasible solution for 10 out of 50 instances.

7.5 Conclusions

In this chapter, we formalized the multi-sprint planning problem and proposed a generalized assignment model to solve it (see [Golfarelli et al., 2012c]). Our model was conceived for an interactive and flexible use by a design team that progressively defines the best plan or revises it during its execution.

Table 7.9: Comparison between IBM ILog Cplex and *LagrangianHeuristic* setting a time limit of 10 secs

	IBM Ilog Cplex			**LagrangianHeuristic**		
Name	z	**Gap**	**Time**	z	**RGap**	**Time**
PayTV	92781.0	1.40	10.02	92066.0	0.77	10.08
Web	32683.6	0.00	0.30	32683.6	0.00	10.75
25Chain-1	16627.2	0.00	0.81	16613.6	0.08	0.52
25Graph-1	13515.1	0.00	0.19	13515.1	0.00	0.17
25Affinity-1	18523.6	0.00	0.87	18505.1	0.10	0.83
50Chain-1	41164.9	0.63	10.03	41103.9	0.15	4.48
50Graph-1	34686.0	0.05	10.03	34686.0	0.00	1.73
50Affinity-1	40545.1	0.00	2.45	40545.1	0.00	6.40
75Chain-1	84711.8	6.74	10.06	88470.2	-4.44	10.08
75Graph-1	92753.6	0.58	10.06	92679.3	0.08	10.28
75Affinity-1	75958.6	0.25	10.03	75841.3	0.16	10.30
100Chain-1	–	–	10.19	135991.8	–	10.22
100Graph-1	149312.2	0.48	10.17	148993.6	0.21	10.23
100Affinity-1	135727.6	0.49	10.14	135574.6	0.11	10.56
25Chain-2	13214.8	0.00	0.16	13214.8	0.00	0.58
25Graph-2	17202.6	0.00	0.16	17202.6	0.00	0.33
25Affinity-2	13007.4	0.00	0.19	13007.4	0.00	0.78
50Chain-2	46629.1	0.02	10.02	46545.8	0.18	2.96
50Graph-2	37669.9	0.26	10.05	37693.0	-0.06	7.72
50Affinity-2	46143.6	0.28	10.02	45995.6	0.32	7.52
75Chain-2	–	–	10.05	78816.4	–	10.08
75Graph-2	72478.0	0.37	10.06	72408.5	0.10	10.20
75Affinity-2	84047.4	0.61	10.06	83741.6	0.36	10.06
100Chain-2	132748.5	1.85	10.12	134217.7	-1.11	10.05
100Graph-2	134632.3	0.58	10.15	134871.6	-0.18	10.02
100Affinity-2	135996.2	0.60	10.14	136171.7	-0.13	10.02
25Chain-3	2855.2	0.44	10.02	2855.2	0.00	0.70
25Graph-3	1934.7	0.26	10.02	1919.4	0.79	0.62
25Affinity-3	2002.3	0.00	5.26	1994.9	0.37	0.84
50Chain-3	6094.3	4.36	10.02	6167.1	-1.19	6.85
50Graph-3	–	–	10.06	5918.4	–	4.52
50Affinity-3	5189.1	2.15	10.03	5210.2	-0.41	8.00
75Chain-3	–	–	10.11	12048.9	–	10.23
75Graph-3	–	–	10.08	11375.4	–	6.47
75Affinity-3	–	–	10.16	14436.8	–	10.09
100Chain-3	–	–	10.09	19336.0	–	10.39
100Graph-3	19850.9	1.36	10.19	19859.0	-0.04	10.08
100Affinity-3	–	–	10.27	25371.2	–	10.39
25Chain-4	18240.4	0.00	1.25	18240.4	0.00	1.84
25Graph-4	20561.7	0.00	6.32	20551.4	0.05	0.84
25Affinity-4	20308.8	0.00	1.81	20286.8	0.11	0.45
50Chain-4	60153.6	2.85	10.03	60033.9	0.20	9.39
50Graph-4	66099.6	1.36	10.05	66166.2	-0.10	7.46
50Affinity-4	59604.8	1.23	10.05	59912.3	-0.51	10.02
75Chain-4	–	–	10.05	111538.6	–	10.03
75Graph-4	128261.4	3.26	10.08	129963.0	-1.33	10.14
75Affinity-4	123522.5	2.68	10.05	123763.8	-0.19	10.02
100Chain-4	–	–	10.06	215393.1	–	10.26
100Graph-4	241622.6	3.42	10.16	246505.4	-2.02	10.28
100Affinity-4	232747.4	3.43	10.12	236441.2	-1.59	10.16

Our model can be applied whenever the basic assumptions of agile methods hold, namely, definition of requirements in form of micro-functionalities (i.e., stories), capability of producing estimates of story utility, complexity, and correlation, and frequent iterations based on user feedback (which implies allocation of stories to sprints). Noticeably, our model is not geared towards a specific type of project: though in bespoke projects it can benefit from user involvement during story definition and estimation (mainly for utility assessment), it can also be effectively applied in market-driven projects where the team experience and the feedbacks received during the beta-test phase can cope with the absence of key users.

The tests we carried out show that, for medium-sized problems, an exact solution is found in a time that is fully compatible with the development process (i.e., from some seconds to a few minutes), while for large problems a heuristic solution that is less than 1% far from the exact one can be returned in a few seconds. Moreover, we proposed different algorithms to further improve the model performance for complex problems. As to effectiveness, the team managers judged the optimal plans to be feasible and realistic, and most of the differences in sprint composition were evaluated as improvements over the team plan. In smooth replanning, the trade-off between the quality and the stability of the new plan is always very good. For these reasons, we believe that our optimization module could be a very convenient and powerful add-on to the existing softwares for agile project management.

Finally, we planned to extend our work from different perspectives: (1) allowing different development speeds for different sprints due to a variable team composition; (2) modeling different team capabilities (e.g., design, implement, test) so that, in each sprint, the team will be able to deliver a different number of story points for each capability; (3) extending the model to support multiple teams working on the same project, which requires to introduce a concept of chunks of stories like done by Szoke [2011]; (4) implementing a structured approach to utility definition and measure its impact on the accuracy of the estimates and consequently on the effectiveness of plans.

Chapter 8

Conclusions and Future Works

In this thesis we described the main contributions we gave on three different aspects of pervasive business intelligence: *Distributed BI*, *OLAP Personalization and Similarity*, and *Agile Data Warehouse Design*.

In the context of *Distributed BI*, the BIN framework represents an innovative approach to support company collaboration, thus favouring BI ANYWHERE. The distributed solution we envisioned supports high scalability, dynamism, and peer autonomy as well. On the other hand, it poses many issues related to routing strategies, especially when the number of network nodes grows. Our main contribution here was to devise a query reformulation approach, made necessary by the heterogeneity of peers. Besides, we proved the correctness of the reformulation algorithm and gave an estimation of the reformulation quality. We provided a basic implementation solution for the peer infrastructure based on the MDX language and the Mondrian suite. Nevertheless, several aspects deserve further investigations, namely: (1) how to efficiently process queries across the network by applying routing strategies that select a subset of neighboring peers for reformulation; (2) how to automatically detect semantic mappings between concepts in different schemata; (3) how to efficiently transmit huge quantity of data in the net; (4) how to reconcile multidimensional data returned by different peers through object fusion techniques; (5) how to rank peer results depending on how compliant they are with the original local query; (6) how to deal with security depending on the degree of trust between the BIN participants.

As concerns *OLAP Personalization and Similarity*, we investigated different perspectives to enhance the OLAP navigation, supporting the concept of BI to ANYONE. We designed a soft approach that personalizes the current user queries starting from the log of past queries. This technique allows to automatically mine relevant preferences to annotate each query in order to refine its result. On the other hand, the user experience can be further improved by directly suggesting the user the next query to formulate. In this direction, we studied different measures to compare the OLAP session a user is currently involved in, with the sessions that were issued in the past by the same or other users, namely: edit-, subsequence-, log-, and alignment-based similarity approaches. We extended each measure based on the result of a real case study we carried out to extract the requirements to compare OLAP sessions. It turned out that the alignment-based measure is the one that best fit the user requirements. Starting with this result, our future works will use this measure to design a method for recommending the next OLAP query to formulate. We will pay particular attention in mixing intensional and extensional information, as suggested in [Chatzopoulou et al., 2011], in order to support OLAP exploratory analysis. This will have a major impact on improving OLAP-based interactions from both points of view of efficiency (by reducing the query formulation effort) and effectiveness (by suggesting popular/successful trends of analysis).

Finally, in the field of *Agile Data Warehouse Design* we contributed to BI ANYTIME proposing two different solutions. First, we designed 4WD that represents a new methodology to combine agile principles with traditional DW development approaches. 4WD has been successfully applied to the PayTV case study, allowing a reduction of the implementation effort and favouring the early detection of errors. Second, we proposed a multi-sprint planning model, based on 4WD principles, to support the analyst during the project scheduling. This model produces an automatically optimized plan that can be used as an initial suggestion for the analyst. Moreover, the model includes a smooth replanning solution to allow the management of disrupted or new stories, and to accommodate changes during the project life-cycle. The case study on PayTV and Web proved the effectiveness of our model in terms of sprint composition, risk distribution, and delivered utility. As to efficiency, the model performs well on small-medium projects. For large projects (e.g., more than 100 stories) we proposed different algorithms to improve the performance, namely *reductions*, *cover inequalities*, *dominance inequalities*, *greedy and exchange heuristics*, and a *Lagrangian heuristic*. The *Lagragian heuristic* produces the best result with respect to the one returned by a general MIP solver such as IBM ILog Cplex. We stressed the algorithms with both synthetic and real problems, varying the number of stories and correlations, and the type of precedences. Though we obtained positive feedback from both efficiency and effectiveness tests, we plan to extend our model to better address real project issues: allowing different development speeds for different sprints due to a variable team composition, modeling different team capabilities (e.g., design, implement, test) so that, in each sprint, the team will be able to deliver a different number of story points for each capability, and providing a detailed definition of the different components affecting the utility concept, to have more precise estimations.

Appendix A

Theorem Proofs

Theorem A.1. *Let q_t be a (target) BIN query and q' be the output of the query reformulation algorithm when q_t is given as input. Then, there exists a (source) BIN query q_s such that q' is the relational translation of q_s.*

Proof. Let us consider $q'(\overline{z'}, expr'(\alpha'_1(w'_1), \ldots, \alpha'_{v'}(w'_{v'})) \leftarrow body$. Note that the way attribute mappings have been defined ensures that *body* follows a star form, i.e., it contains the star join that is necessary to relate the source fact table with the involved dimension tables. This implies that a BIN query must exist whose encoding is q'. Then, without loss of generality, we assume that q' is equipped with a variable assignment function ν that can be easily derived from the variable assignment functions of q and of the involved mappings. With a little abuse of notation we will apply the inverse of ν, ν^{-1}, also to atomic formulas, meaning that ν^{-1} is applied to its arguments. We define $q_s = \langle \mathcal{M}_s, E_s, p_s, expr_s, T_s \rangle$ on the source schema $\mathcal{M}_s$ as follows:

G_s : for each head variable $v \in \overline{z'}$, check whether there is a comparison predicate $v = f(\overline{v'})$ in the *body*. If it exists, then add $f(\nu^{-1}(\overline{v'})$ to E_s and delete the predicate from *body*, otherwise add $\nu^{-1}(v)$ to E_s.

p_s : for each left predicate p in *body*, add $\nu^{-1}(p)$ to p_s.

$expr_s$: is the expression $expr'(\nu^{-1}(w'_1), \ldots, \nu^{-1}(w'_{v'}))$.

T_s : is the list $\langle\langle \nu^{-1}(w'_1), \alpha'_1\rangle, \ldots, \langle \nu^{-1}(w'_{v'}), \alpha'_{v'}\rangle\rangle$

Then, it can be easily shown that the relational translation of q_s is q' modulo variable mappings. □

Theorem A.2. *Let q_t be a BIN query, q_{t_1} be the output of Step 1 on q_t, $q(\overline{z}, aggrExpr)$ be the relational translation of q_{t_1}, and $\overline{z''}$ be the head variables of q that find a reformulation. The reformulation algorithm guarantees to find all certain answers of $q(\overline{z''}, aggrExpr)$.*

Proof. In the following we show that our algorithm reduces to the algorithm for query reformulation shown in [Halevy et al., 2005], which is proved to find all and only certain answers. Our s-t tgd's are in GLAV-style. As in Halevy et al. [2005], we transform them in pairs of LAV and GAV s-t tgd's; Step 2 deals with the set of GAV and LAV s-t tgd's by first matching the LAV s-t tgd's and then the GAV ones. The main difference lies in the selection of mappings. Indeed, our algorithm performs a syntactic selection whereas the algorithm in [Halevy et al., 2005] refers to the logical form of the mappings. Nevertheless, for mappings of types τ_1 and τ_2 (i.e., those that define $\overline{z''}$) it can be easily shown that Step 1 follows the same selection principles adopted by usual approaches for answering queries using views [Halevy, 2001], thus making the mapping selection phase equivalent. Finally, the computation of Γ at Step 3 has the objective of determining additional constraints between source variables induced by the mappings of types τ_1 and τ_2. Without the merging of Γ with the body of the reformulated query q', q' would return a superset of the answers, also including results that do not satisfy all the implicit constraints derived in Γ. Thus Step 3 is fundamental for determining the certain answers. $\square$

Bibliography

Aalto University, SoberIT. Agilefant. http://www.agilefant.org/, 2011.

S. Abiteboul. Managing an XML warehouse in a P2P context. In *Proc. CAiSE*, pages 4–13, Klagenfurt, Austria, 2003.

S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. PODS*, pages 254–263, Seattle, Washington, USA, 1998.

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

S. Abiteboul, I. Manolescu, and N. Preda. Constructing and querying peer-to-peer warehouses of xml resources. In *ICDE*, pages 1122–1123, 2005.

F. N. Afrati and P. G. Kolaitis. Answering aggregate queries in data exchange. In *Proc. PODS*, pages 129–138, Vancouver, BC, Canada, 2008.

Agile Manifesto. Manifesto for agile software development. http://agilemanifesto.org/, 2010.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. VLDB*, pages 487–499, Santiago de Chile, Chile, 1994.

R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 383–394, Chicago, IL, 2006.

J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. Swarubini Vindhiya Varman. SQL QueRIE recommendations. *PVLDB*, 3(2): 1597–1600, 2010.

M. O. Akinde, M. H. Böhlen, T. Johnson, L. V. S. Lakshmanan, and D. Srivastava. Efficient OLAP query processing in distributed data warehouses. *Inf. Syst.*, 28(1-2): 111–135, 2003.

O. Alagoz and M. Azizoglu. Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149(3):523–532, 2003.

J. Albrecht and W. Lehner. On-line analytical processing in distributed data warehouses. In *Proc. IDEAS*, pages 78–85, Cardiff, Wales, U.K., 1998.

J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Mining preferences from olap query logs for proactive personalization. In *ADBIS*, pages 84–97, 2011.

J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measure for olap sessions. *To appear on KAIS*, 2013.

K. Aouiche, P.-E. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In *Proceedings East European Conference on Advances in Databases and Information Systems*, pages 81–95, Thessaloniki, Greece, 2006.

P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact kanpsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45:543–555, 2010.

H. Baars and H.-G. Kemper. Business intelligence in the cloud? In *PACIS*, page 145, 2010.

E. Baikousi, G. Rogkakos, and P. Vassiliadis. Similarity measures for multidimensional data. In *Proc. ICDE*, pages 171–182, Hannover, Germany, 2011.

M. Banek, A. Min Tjoa, and N. Stolba. Integrating different grain levels in a medical data warehouse federation. In *DaWaK*, pages 185–194, 2006.

M. Banek, B. Vrdoljak, A. Min Tjoa, and Z. Skocir. Automated integration of heterogeneous data warehouse schemas. *IJDWM*, 4(4):1–21, 2008.

J.E. Beasley. Lagrangean relaxation. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. Blackwell Scientific Publications, 1993.

K. Beck. Embracing change with extreme programming. *IEEE Computer*, 32(10): 70–77, 1999.

L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent. A personalization framework for OLAP queries. In *Proc. DOLAP*, pages 9–18, Bremen, Germany, 2005.

S. Berger and M. Schrefl. Analysing multi-dimensional data across autonomous data warehouses. In *DaWaK*, pages 120–133, 2006.

S. Berger and M. Schrefl. From federated databases to a federated data warehouse system. In *HICSS*, page 394, 2008.

M. Beyer and J. Richardson. Agile techniques augment but do not replace business intelligence and data warehouse best practice. Technical Report G00201031, Gartner Research, 2010.

P. Biondi, M. Golfarelli, and S. Rizzi. Preference-based datacube analysis with myOLAP. In *Proc. ICDE*, 2011.

B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988.

M. A. Boschetti and L. Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58(6):1774–1791, November 2010. ISSN 0030-364X.

M.A. Boschetti and V. Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15:283–312, 2009. ISSN 1381-1231.

M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR*, 1:27–42, 2003.

M.A. Boschetti, E. Hadjinconstantinou, and A. Mingozzi. New upper bounds for the finite two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

M.A. Boschetti, V. Maniezzo, and M. Roffilli. Decomposition techniques as metaheuristic frameworks. In *Matheuristics, Hybridizing Metaheuristics and Mathematical Programming*, pages 135 – 158. Springer, NEW YORK, 2009.

P. F. Brown, V. J. Della Pietra, P. V. de Souza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

R. M. Bruckner, T. Wang Ling, O. Mangisengi, and A. Min Tjoa. A framework for a multidimensional olap model using topic maps. In *WISE (2)*, pages 109–118, 2001.

B. Bustos and T. Skopal. Non-metric similarity search problems in very large collections. In *ICDE*, pages 1362–1365, Hannover, Germany, 2011.

A. Caprara and M. Fischetti. Branch-and-cut algorithms. In M. Dell'Amico and F. Maffioli, editors, *Annotated Bibliographies in Combinatorial Optimization.* Wiley Interscience Series in Discrete Mathematics, 1997.

K. C. Chang and H. Garcia-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources. In *Proc. SIGMOD*, pages 335–346, Philadelphia, Pennsylvania, USA, 1999.

G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Proceedings International Conference on Scientific and Statistical Database Management*, pages 3–18, New Orleans, LA, 2009a.

G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Proc. SSDBM*, pages 3–18, New Orleans, USA, 2009b.

G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. Swarubini Vindhiya Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

S. Chaudhuri, U. Dayal, and V.R. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, 2011.

S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Transactions on Database Systems*, 31(2):672–715, 2006.

W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings IJCAI-03 Workshop on Information Integration on the Web*, pages 73–78, Acapulco, Mexico, 2003.

M. Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.

Collabnet. ScrumWorks. http://www.danube.com/, 2011.

P. Cudré-Mauroux, K. Aberer, and A. Feher. Probabilistic message passing in peer data management systems. In *ICDE*, page 41, 2006.

M. Denne and J. Cleland-Huang. *Software by Numbers.* Prentice Hall, 2004.

M. Drosou and E. Pitoura. ReDRIVE: result-driven database exploration through recommendations. In *Proceedings CIKM*, pages 1547–1552, Glasgow, United Kingdom, 2011.

D. Dubois and H. Prade. On the use of aggregation operations in information fusion processes. *Fuzzy Sets and Systems*, 142(1):143–161, 2004.

T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information & Software Technology*, 50(9-10):833–859, 2008.

M. M. Espil and A. A. Vaisman. Aggregate queries in peer-to-peer OLAP. In *DOLAP*, pages 102–111, Washington, DC, USA, 2004.

R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. ICDT*, pages 207–224, Siena, Italy, 2003.

R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

A. Fuxman, P. G. Kolaitis, R. J. Miller, and W.-C. Tan. Peer data exchange. In *Proc. PODS*, pages 160–171, Baltimore, Maryland, USA, 2005.

H. Garcia-Molina, J. D. Ullman, and J. D. Widom. *Database Systems: The Complete Book, Second edition.* Prentice Hall, 2008.

Gartner. Amplifying the enterprise, 2012.

C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of software engineering*. Prentice Hall, 2002.

A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa. Plan selection based on query clustering. In *Proceedings International Conference on Very Large Data Bases*, pages 179–190, Hong Kong, China, 2002.

A. Giacometti, P. Marcel, and E. Negre. Recommending multidimensional queries. In *Proc. DaWaK*, pages 453–466, Linz, Austria, 2009.

A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for OLAP discovery driven analysis. *IJDWM*, 2011.

P. Giorgini, S. Rizzi, and M. Garzetti. GRAnD: A goal-oriented approach to requirement analysis in data warehouses. *Decision Support Systems*, 45(1):4–21, 2008.

M. Golfarelli. Handling large workloads by profiling and clustering. In *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, pages 212–223, Prague, Czech Republic, 2003.

M. Golfarelli and S. Rizzi. A methodological framework for data warehouse design. In *Proc. DOLAP*, pages 3–9, 1998.

M. Golfarelli and S. Rizzi. WAND: A CASE tool for data warehouse design. In *Proc. ICDE*, pages 7–9, 2001.

M. Golfarelli and S. Rizzi. A comprehensive approach to data warehouse testing. In *Proc. DOLAP*, pages 17–24, 2009a.

M. Golfarelli and S. Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, 2009b.

M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, and E. Turricchia. Towards olap query reformulation in peer-to-peer data warehousing. In *DOLAP*, pages 37–44, 2010.

M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, and E. Turricchia. BIN: Business intelligence networks. In *Business Intelligence Applications and the Web: Models, Systems and Technologies*. IGI Global, 2011a.

M. Golfarelli, S. Rizzi, and P. Biondi. myOLAP: An approach to express and evaluate OLAP preferences. *IEEE TKDE*, 2011b.

M. Golfarelli, S. Rizzi, and E. Turricchia. Modern software engineering methodologies meet data warehouse design: 4wd. In *DaWaK*, pages 66–79, 2011c.

M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, and E. Turricchia. A query reformulation framework for p2p olap. In *SEBD*, pages 147–154, 2012a.

M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, and E. Turricchia. Olap query reformulation in peer-to-peer data warehousing. *Inf. Syst.*, 37(5):393–411, 2012b.

M. Golfarelli, S. Rizzi, and E. Turricchia. Sprint planning optimization in agile data warehouse design. In *Proc. Int. Conf. on Data Warehousing and Knowledge Discovery*, Vienna, Austria, 2012c.

V. S. Gordon and J. M. Bieman. Rapid prototyping: Lessons learned. *IEEE Software*, 12(1):85–95, 1995.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information & Software Technology*, 46(4):243–253, 2004.

A. Gupta and I. Mumick. *Materialized views: techniques, implementations, and applications.* MIT Press, 1999.

A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proceedings International Conference on Very Large Data Bases*, pages 358–369, Zurich, Switzerland, 1995.

A. Y. Halevy. Technical perspective - schema mappings: rules for mixing data. *Commun. ACM*, 53(1), 2010.

A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza peer data management system. *IEEE TKDE*, 16(7):787–798, 2004.

A.Y. Halevy. Answering queries using views: A survey. *VLDBJ*, 10(4):270–294, 2001.

A.Y. Halevy, Z.G. Ives, D. Suciu, and I. Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDBJ*, 14(1):68–83, 2005.

Y.g He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *ICDE*, pages 1199–1208, 2011.

G. T. Heineman and W. T. Councill. *Component-based software engineering: Putting the pieces together.* Addison-Wesley, 2001.

T. A. D. Hoang and T. Binh Nguyen. State of the art and emerging rule-driven perspectives towards service-based business process interoperability. In *Proc. Int. Conf. on Comp. and Comm. Tech.*, pages 1–4, Danang City, Vietnam, 2009.

S. Holland, M. Ester, and W. Kiessling. Preference mining: A novel approach on mining user preferences for personalized applications. In *Proc. PKDD*, pages 204–216, Cavtat-Dubrovnik, Croatia, 2003.

R. Hughes. *Agile Data Warehousing: Delivering world-class business intelligence systems using Scrum and XP*. IUniverse, 2008.

IBM. IBM ILOG CPLEX optimizer. http://www-01.ibm.com/, 2011.

W.H. Inmon. *Building the data warehouse*. John Wiley & Sons, 1996.

H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh. Management of context-aware preferences in multidimensional databases. In *Proc. ICDIM*, pages 669–675, London, UK, 2008.

H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh. Applying recommendation technology in OLAP systems. In *Proc. ICEIS*, pages 220–233, Milan, Italy, 2009.

H. Jiang, D. Gao, and W.-S. Li. Exploiting correlation and parallelism of materialized-view recommendation for distributed data warehouses. In *Proc. ICDE*, pages 276–285, Istanbul, Turkey, 2007.

R. Jindal and A. Acharya. Federated data warehouse architecture. http://www.wipro.com/, 2004.

P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. In *Proc. SIGMOD*, pages 25–36, Madison, Wisconsin, USA, 2002.

M. Kehlenbeck and M. H. Breitner. Ontology-based exchange and immediate application of business calculation definitions for online analytical processing. In *Proc. DaWaK*, pages 298–311, Linz, Austria, 2009.

A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proc. SIGMOD*, pages 325–336, San Diego, California, USA, 2003.

N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010a.

N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010b.

P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

J. Lachlan. Top business intelligence predictions for 2012. http://www.yellowfinbi.com/YFWebsite-Yellowfin-Business-Intelligence-77988, 2012.

M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS*, pages 233–246, Madison, Wisconsin, USA, 2002.

C. Li, M. van den Akker, S. Brinkkemper, and G. Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requir. Eng.*, 15: 375–396, 2010.

H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proc. ICDM*, pages 369–376, 2001.

S. Luján-Mora and J. Trujillo. A comprehensive method for data warehouse design. In *Proc. DMDW*, 2003.

J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.

F. Mandreoli, R. Martoglia, W. Penzo, and S. Sassatelli. SRI: Exploiting semantic information for effective query routing in a PDMS. In *Proc. WIDM*, pages 19–26, Arlington, Virginia, USA, 2006.

F. Mandreoli, R. Martoglia, W. Penzo, S. Sassatelli, and G. Villani. SRI@work: Efficient and effective routing strategies in a PDMS. In *Proc. WISE*, pages 285–297, Nancy, France, 2007.

F. Mandreoli, R. Martoglia, W. Penzo, and S. Sassatelli. Data-sharing P2P networks with semantic approximation capabilities. *IEEE Internet Computing*, 13(5):60–70, 2009.

O. Mangisengi, J. Huber, C. Hawel, and W. Essmayr. A framework for supporting interoperability of data warehouse islands using xml. In *DaWaK*, pages 328–338, 2001.

J. Markarian, S. Brobst, and J. Bedell. Critical success factor deploying pervasive bi, 2007.

S. Martello and P. Toth. *Knapsack Problems: Algorithm and Computer Implementation.* John Wiley and Sons Ltd, 1990.

J. Martin. *Rapid application development.* MacMillan, 1991.

J.-N. Mazón and J. Trujillo. An MDA approach for the development of data warehouses. In *Proc. JISBD*, pages 208–208, 2009.

G. Mecca, P. Papotti, and S. Raunich. Core schema mappings. In *Proc. SIGMOD*, pages 655–668, Providence, Rhode Island, USA, 2009.

Microsoft. MDX reference. http://msdn.microsoft.com/en-us/library/ms145506.aspx, 2009.

R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *Proc. VLDB*, pages 77–88, Cairo, Egypt, 2000.

Minnesota Population Center. Integrated public use microdata series. http://www.ipums.org, 2008.

B. Mobasher. Data mining for web personalization. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 90–135. Springer, 2007.

A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

D. Moody and M. Kortink. From enterprise models to dimensional models: A methodology for data warehouse and data mart design. In *Proc. DMDW*, 2000.

E. Moreau, F. Yvon, and O. Cappé. Robust similarity measures for named entities matching. In *Proceedings International Conference on Computational Linguistics*, pages 593–600, Manchester, UK, 2008.

G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surveys*, 33(1):31–88, 2001.

G. Nelson. Business intelligence 2.0: Are we there yet? In *SAS Global Forum 2010*, Seattle, USA, 2010.

A. Nichols. Agile planning, estimation and tracking. http://www.slideshare.net/andrewnichols/agile-planning-estimation-and-tracking, 2009.

Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB*, pages 413–424, 1996.

W. Penzo. Rewriting rules to permeate complex similarity and fuzzy queries within a relational database system. *IEEE Trans. Knowl. Data Eng.*, 17(2):255–270, 2005.

G. Pomberger, W. R. Bischofberger, D. Kolb, W. Pree, and H. Schlemm. Prototyping-oriented software development — concepts and tools. *Structured Programming*, 12 (1):43–60, 1991.

N. Prat, J. Akoka, and I. Comyn-Wattiau. A UML-based data warehouse design method. *Decision Support Systems*, 42(3):1449–1473, 2006.

Z. Racheva, M. Daneva, and K. Sikkel. Value creation by agile projects: Methodology or mystery? In *Proc. PROFES*, pages 141–155, Oulu, Finland, 2009.

K. Ramamurthy, A. Sen, and A. P. Sinha. An empirical investigation of the key determinants of data warehouse adoption. *Decision Support Systems*, 44(4):817–841, 2008.

E.S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):522–532, 1998.

C. Robson. *Real World Research.* Blackwell, 2002.

W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proc. ICSE*, pages 328–339, Monterey, California, USA, 1987.

P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

C. Sá, C. Soares, A. Jorge, P. Azevedo, and J. Costa. Mining association rules for label ranking. In *Proc. PAKDD*, pages 24–27, Shenzhen, China, 2011.

M. Omolade Saliu and G. Ruhe. Supporting software release planning decisions for evolving systems. In *SEW*, pages 14–26, 2005.

M.O. Saliu and G. Ruhe. Bi-objective release planning for evolving software systems. In *ESEC/SIGSOFT FSE*, pages 105–114, 2007.

C. Sapia. PROMISE: Predicting query behavior to enable predictive caching strategies for OLAP systems. In *Proceedings International Conference on Data Warehousing and Knowledge Discovery*, pages 224–233, London, UK, 2000.

S. Sarawagi. Explaining differences in multidimensional aggregates. In *Proc. VLDB*, pages 42–53, Edinburgh, Scotland, 1999.

S. Sarawagi. I3: Intelligent, interactive inspection of cubes. http://www.cse.iitb.ac.in/ sunita/icube/, 2009.

M. Schneider. Integrated vision of federated data warehouses. In *DISWEB*, Luxemburg, 2006.

K. Schwaber. SCRUM development process. In *Proc. OOPSLA*, 1995.

A. Sen and A. P. Sinha. A comparison of data warehousing methodologies. *Commun. ACM*, 48(3):79–84, 2005.

N.Z. Shor. *Minimization Methods for Non-differentiable Functions.* Springer-Verlag, 1985.

A. Simitsis and P. Vassiliadis. A method for the mapping of conceptual designs to logical blueprints for ETL processes. *Decision Support Systems*, 45(1):22–40, 2008.

T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

I. Sommerville. *Software Engineering.* Pearson Education, 2004.

K. Stefanidis, M. Drosou, and E. Pitoura. "You May Also Like" results in relational databases. In *Proceedings International Workshop on Personalized Access, Profile Management and Context Awareness: Databases*, Lyon, France, 2009.

M. Svahnberg, T. Gorschek, R. Feldt, R. Torkar, S. Bin Saleem, and M. Usman Shafique. A systematic review on strategic release planning models. *Information & Software Technology*, 52(3):237–248, 2010.

A. Szoke. Conceptual scheduling model and optimized release scheduling for agile environments. *Information & Software Technology*, 53:574–591, 2011.

Ashish T., Joydeep S. S., Namit J., Zheng S., Prasad C., Ning Z., Suresh A., Hao L., and Raghotham M. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

T.B. Nguyen T.A.D. Hoang. State of the art and emerging rule-driven perspectives towards service-based business process interoperability. In *Int. Conf. on Comp., and Comm. Tech.*, pages 1–4, Danang City, Vietnam, 2009.

I. Tatarinov and A. Y. Halevy. Efficient query reformulation in peer-data management systems. In *SIGMOD Conference*, pages 539–550, 2004.

B. ten Cate and P.G. Kolaitis. Structural characterizations of schema-mapping languages. *Commun. ACM*, 53(1):101–110, 2010.

Teradata. The active data warehouse: Where agile retailers win by capitalizing on time, 2008.

The Data Warehousing Institute. Pervasive business intelligence techniques and technologies to deploy bi on an enterprise scale, 2008.

ThoughtWorks Studios. Mingle: Agile project management. http://www.thoughtworks-studios.com/, 2011.

R. Torlone. Two approaches to the integration of heterogeneous data warehouses. *Distributed and Parallel Databases*, 23(1):69–97, 2008.

F. S. C. Tseng and C.-W. Chen. Integrating heterogeneous data warehouses using xml technologies. *J. Information Science*, 31(3):209–229, 2005.

A. Vaisman, M. Minuto Espil, and M. Paradela. P2P OLAP: Data model, implementation and case study. *Inf. Syst.*, 34(2):231–257, 2009.

G. van Valkenhoef, T. Tervonen, B. de Brock, and Douwe Postmus. Quantitative release planning in extreme programming. *Information & Software Technology*, 53: 1227–1235, 2011.

A. Veloso, H. Mossri de Almeida, M. André Gonçalves, and W. Meira Jr. Learning to rank at query-time using association rules. In *Proc. SIGIR*, pages 267–274, Singapore, 2008.

J. Vieira, J. Bernardino, and H. Madeira. Efficient compression of text attributes of data warehouse dimensions. In *DaWaK*, pages 356–367, 2005.

R. Wagner and M. Fischer. The string-to-string correction problem. *Journal ACM*, 21 (1):168–173, 1974.

P. Weber and D. Chapman. Location intelligence: An innovative approach to business location decision-making. *T. GIS*, 15(3):309–328, 2011.

Hadoop Wiki. Apache hadoop. http://www.yellowfinbi.com/YFWebsite-Yellowfin-Business-Intelligence-77988, 2012.

X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *Proceedings International Conference on Data Engineering*, pages 964–975, Shanghai, China, 2009.

Q. Yao, A. An, and X. Huang. Finding and analyzing database user sessions. In *Proceedings International Conference on Database Systems for Advanced Applications*, pages 851–862, Beijing, China, 2005.

Yellowfin. Location intelligence. http://www.yellowfinbi.com/Document.i4?DocumentId=102780, 2010.

F. Yu and S. Wang. Compressed data cube for approximate olap query processing. *J. Comput. Sci. Technol.*, 17(5):625–635, 2002.

S. Zhou, A. Zhou, X. Tao, and Y. Hu. Hierarchically distributed data warehouse. In *HPC*, pages 848–853, 2000.