Dynamic cubing for hierarchical multidimensional data space

Usman Ahmed

N° d'ordre :
2013ISAL0011

Année : 2013

Université de Lyon
Institut National des Sciences Appliquées de Lyon
Laboratoire d'InfoRmatique en Image et Systèmes d'information

THESE

Dynamic Cubing for Hierarchical Multidimensional Data Space

pour obtenir le grade de

Docteur de L'INSA de Lyon

Discipline : Informatique
Ecole Doctorale : InfoMaths

présenté et soutenue publiquement par

Usman AHMED

le 18 février 2013

devant le jury composé de

Ladjel BELLATRECHE ..... Professeur des Universités, ENSMA Poitiers (Rapporteur)
Maryvonne MIQUEL ..... Maître de Conférences, HDR, INSA de Lyon (Co-Directrice)
Jean-Marc PETIT ..... Professeur des Universités, INSA de Lyon
Franck RAVAT ..... Professeur des Universités, Université Toulouse I
Anne TCHOUNIKINE ..... Maître de Conférences, INSA de Lyon (Co-Directrice)
Karine ZEITOUNI .... Professeur des Universités, Université de Versailles Saint-Quentin-en-Yvelines (Rapporteur)
Esteban ZIMANYI ..... Professeur des Universités, Université Libre de Bruxelles

Usman AHMED
PhD, Computer Science, INSA de Lyon

Order No.:
2013ISAL0011

Year: 2013

Université de Lyon
Institut National des Sciences Appliquées de Lyon
Laboratoire d'InfoRmatique en Image et Systèmes d'information

THESIS

Dynamic Cubing for Hierarchical Multidimensional Data Space

submitted to obtain the grade of

Doctor of INSA de Lyon

Discipline: Computer Science
Ecole Doctorale: InfoMaths

presented and publicly defended by

Usman Ahmed

on 18th February, 2013

in front of a jury composed of

Ladjel BELLATRECHE ..... Professeur des Universités, ENSMA Poitiers (Reviewer)
Maryvonne MIQUEL ..... Maître de Conférences, HDR, INSA de Lyon (Co-Supervisor)
Jean-Marc PETIT ..... Professeur des Universités, INSA de Lyon
Franck RAVAT ..... Professeur des Universités, Université Toulouse I
Anne TCHOUNIKINE ..... Maître de Conférences, INSA de Lyon (Co-Supervisor)
Karine ZEITOUNI ..... Professeur des Universités, Université de Versailles Saint-Quentin-en-Yvelines (Reviewer)
Esteban ZIMANYI ..... Professeur des Universités, Université Libre de Bruxelles

INSA Direction de la Recherche - Ecoles Doctorales – Quinquennal 2011-2015

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Insa : R. GOURDON	M. Jean Marc LANCELIN Université de Lyon – Collège Doctoral Bât ESCPE 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72.43 13 95 directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Secrétariat : M.C. HAVGOUDOUKIAN eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60 97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Insa : H. CHARLES	Mme Gudrun BORNETTE CNRS UMR 5023 LEHNA Université Claude Bernard Lyon 1 Bât Forel 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cédex Tél : 04.72.43.12.94 e2m2@biomserv.univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://ww2.ibcp.fr/ediss** Sec : Safia AIT CHALAL Insa : M. LAGARDE	M. Didier REVEL Hôpital Louis Pradel Bâtiment Central 28 Avenue Doyen Lépine 69677 BRON Tél : 04.72.68 49 09 Fax :04 72 35 49 16 Didier.revel@creatis.uni-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** http://infomaths.univ-lyon1.fr	M. Johannes KELLENDONK Université Claude Bernard Lyon 1 INFOMATHS Bâtiment Braconnier 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72. 44.82.94 Fax 04 72 43 16 87 infomaths@univ-lyon1.fr
Matériaux	**MATERIAUX DE LYON** Secrétariat : M. LABOUNE PM : 71.70 Fax : 87.12 Bat. Saint Exupéry Ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cédex Tél : 04.72.43 83 18 Fax 04 72 43 85 28 Jean-yves.buffiere@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** Secrétariat : M. LABOUNE PM : 71.70 Fax : 87.12 Bat. Saint Exupéry mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél :04.72.43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** M. OBADIA Lionel Sec : Viviane POLSINELLI Insa : J.Y. TOUSSAINT	M. OBADIA Lionel Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Tél : 04.78.69.72.76 Fax : 04.37.28.04.48 Lionel.Obadia@univ-lyon2.fr

*ScSo : Histoire, Geographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

.

To my parents, my siblings and all my teachers!

A mes parents, mes frères et sœurs et tous mes enseignants!

میرے والدین، بہن بھائیوں اور تمام اساتذہ کے نام!

Acknowledgements

I would like to express my gratitude for the continued guidance, support and the confidence my supervisors, Dr. Maryvonne Miquel and Dr. Anne Tchounikine, showed in me. Their ideas, suggestion and repetitive corrections played a major part in completion of the research work and this manuscript. For someone like me who was first time ever away from his home and that too about 6000 kms, who did not know the native language and was shy, it would have been impossible to complete this research endeavor without the support of my supervisors. They were always there to support me not only in the research work but also in my daily affairs, right from the day I arrived here in Lyon. For such a tremendous support, I owe them a life time gratitude.

I am thankful to the reviewers, Pr. Ladjel Bellatreche and Pr. Karine Zeitouni, for their time and the important suggestions to improve the quality of the manuscript. I am also thankful to the other members of jury, Pr. Jean-Marc Petit, Pr. Franck Ravat and Pr. Esteban Zimanyi, for accepting to come and evaluate my work.

I would also like to acknowledge the help of my colleagues who were very kind to help me in my work, when I needed, as well as in understanding the French culture and administration. It was thanks to them that we were able to maintain a very healthy environment at the work place and which let me complete my work with full serenity and peace of mind. The encouragement and appreciation I got from the members of the Database Research Group at LIRIS is also very important for me.

Last but not the least, I would like to thank my family members, friends and all my teachers right from my early schooldays who showed confidence in me and made me capable of achieving whatever I have. I am grateful for all your support and encouragement.

Abstract

Data warehouses are being used in many applications since quite a long time. Traditionally, new data in these warehouses is loaded through offline bulk updates which implies that latest data is not always available for analysis. This, however, is not acceptable in many modern applications (such as intelligent building, smart grid etc.) that require the latest data for decision making. These modern applications necessitate real-time fast atomic integration of incoming facts in data warehouse. Moreover, the data defining the analysis dimensions, stored in dimension tables of these warehouses, also needs to be updated in real-time, in case of any change. In this thesis, such real-time data warehouses are defined as dynamic data warehouses. We propose a data model for these dynamic data warehouses and present the concept of Hierarchical Hybrid Multidimensional Data Space (HHMDS) which constitutes of both ordered and non-ordered hierarchical dimensions. The axes of the data space are non-ordered which help their dynamic evolution without any need of reordering. We define a data grouping structure, called Minimum Bounding Space (MBS), that helps efficient data partitioning of data in the space. Various operators, relations and metrics are defined which are used for the optimization of these data partitions and the analogies among classical OLAP concepts and the HHMDS are defined. We propose efficient algorithms to store summarized or detailed data, in form of MBS, in a tree structure called DyTree. Algorithms for OLAP queries over the DyTree are also detailed. The nodes of DyTree, holding MBS with associated aggregated measure values, represent materialized sections of cuboids and tree as a whole is a partially materialized and indexed data cube which is maintained using online atomic incremental updates. We propose a methodology to experimentally evaluate partial data cubing techniques and a prototype implementing this methodology is developed. The prototype lets us experimentally evaluate and simulate the structure and performance of the DyTree against other solutions. An extensive study is conducted using this prototype which shows that the DyTree is an efficient and effective partial data cubing solution for a dynamic data warehousing environment.

Résumé

De nombreuses applications décisionnelles reposent sur des entrepôts de données. Ces entrepôts permettent le stockage de données multidimensionnelles historisées qui sont ensuite analysées grâce à des outils OLAP. Traditionnellement, les nouvelles données dans ces entrepôts sont chargées grâce à des processus d'alimentation réalisant des insertions en bloc, déclenchés périodiquement lorsque l'entrepôt est hors-ligne. Une telle stratégie implique que d'une part les données de l'entrepôt ne sont pas toujours à jour, et que d'autre part le système de décisionnel n'est pas continuellement disponible. Or cette latence n'est pas acceptable dans certaines applications modernes, tels que la surveillance de bâtiments instrumentés dits "intelligents", la gestion des risques environnementaux etc., qui exigent des données les plus récentes possible pour la prise de décision. Ces applications temps réel requièrent l'intégration rapide et atomique des nouveaux faits dans l'entrepôt de données. De plus, ce type d'applications opérant dans des environnements fortement évolutifs, les données définissant les dimensions d'analyse elles-mêmes doivent fréquemment être mises à jour. Dans cette thèse, de tels entrepôts de données sont qualifiés d'entrepôts de données dynamiques. Nous proposons un modèle de données pour ces entrepôts dynamiques et définissons un espace hiérarchique de données appelé Hierarchical Hybrid Multidimensional Data Space (HHMDS). Un HHMDS est constitué indifféremment de dimensions ordonnées et/ou non ordonnées. Les axes de l'espace de données sont non-ordonnés afin de favoriser leur évolution dynamique. Nous définissons une structure de regroupement de données, appelé Minimum Bounding Space (MBS), qui réalise le partitionnement efficace des données dans l'espace. Des opérateurs, relations et métriques sont définis pour permettre l'optimisation de ces partitions. Nous proposons des algorithmes pour stocker efficacement des données agrégées ou détaillées, sous forme de MBS, dans une structure d'arbre appelée le DyTree. Les algorithmes pour requêter le DyTree sont également fournis. Les nœuds du DyTree, contenant les MBS associés à leurs mesures agrégées, représentent des sections matérialisées de cuboïdes, et l'arbre lui-même est un hypercube partiellement matérialisé maintenu en ligne à l'aide des mises à jour incrémentielles. Nous proposons une méthodologie pour évaluer expérimentalement cette technique de matérialisation partielle ainsi qu'un prototype. Le prototype nous

permet d'évaluer la structure et la performance du DyTree par rapport aux autres solutions existantes. L'étude expérimentale montre que le DyTree est une solution efficace pour la matérialisation partielle d'un cube de données dans un environnement dynamique.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Chapter Outline

1 Motivation

Since quite a long time, data warehouses are being used to help decision making in many applications such as applications aimed at inventory forecasting, customer relation management etc. These applications generate huge volume of strategically important data which is stored for eventual use in decision making process. The analysis over the data is usually carried out through online analytical processing (OLAP) which involves the summarization of historical data stored in the data warehouses. Classically, the updates in data warehouses are integrated through periodic (e.g. daily, weekly etc.) offline bulk updates. Such a strategy allows enough time for data cleaning, quality checking and maintenance of materialized views but implies that the latest data is not always available for the decision making process and the data warehouses themselves are not available while the updates are being integrated.

As the time has passed, the usage of data warehouses has increased. These are now being used in much critical applications such as surveillance applications, intelligent building, smart grid, fraud/threat detection, energy management, operational intelligence etc. Latest data in these applications is of paramount importance for making critical decisions. This latest data needs to be analyzed in conjunction with the stored historical data in real-time to make timely and realistic decisions. For example, in an intelligent building application several sensors are installed at various locations of the building. These sensors record various indicators such as temperature, humidity, motion etc. which need to be analyzed in real-time for efficient energy management and/or surveillance. Thus, they must be integrated in the warehouse as soon as they are recorded. Moreover, the number of the sensors used in such application are not static and we may need to add new ones. This requires the addition of new members in analysis dimensions or, in other words, demands the fast evolution of dimensional data in warehouse. Due to such critical requirements and usage, the traditional data warehouses are not suitable and recently some techniques and solutions have been proposed.

In literature, such solutions have been proposed using different terminologies which include near real-time, soft real-time, real-time, right time, useful time, active or zero-latency data warehouses. All these solutions propose to integrate new data more often and while both operational systems and data warehouse are online but with some slightly different constraints.

2 Positioning our Research Work

Our research works resides at the frontiers of real-time data warehouses and evolving data warehouses.

The real-time data warehouses invoke the problems linked to the freshness of data. In other words, they deal with the challenges encountered while making efforts to reduce the latency between the occurrence of a fact in the observed system and its integration in an OLAP system. Among the research works addressing such data warehouses, we very often talk about "near real-time" [Zuters 2011, Bruckner 2002, Jorg 2010], "soft real-time" [Vu 2009, Namgyu 2007], "real-time" [Santos 2008] and "zero-latency" [Nguyen 2003] data warehouses. The research works on these data warehouses focus on continuous fast integration of data, appearing at the transactional system's end, into the warehouse. The focal point of the proposed solutions is finding out the optimized ETL strategies in which the triggering event to start data loading is the occurrence of a transaction at the source system.

The data warehouses called as "right-time" [Thomsen 2008] and "useful-time" [Santos 2009] also invoke the issues related to data freshness but this time from the analysis perspective. The solutions proposed in this context present scheduling strategies for the update of aggregates and cube maintenance on the basis of requirements of queries in OLAP systems ("on-demand" updates).

The common point among real-time and right-time data warehouses is that the dynamic aspect of data warehouse and data cubes is linked to the integration of new facts in the system. Another type of data warehouses, we call evolving data warehouses, deal with the problems of considering the evolution of the data which serves at defining the schema of a data cube (e.g. dimensions, hierarchies, members). Therefore, the dynamic aspect in these data warehouses is related to the addition or change in the definition of new members in the multidimensional data space. These data warehouses are sometimes referred as "temporal" [Chamoni 1999, Malinowski 2008] or "dynamic" [Dayal 1999, Theodoratos 1999] data warehouses. The proposed solutions in this context often include strategies such as time-stamping and/or versioning of data.

In this thesis, we use the term *dynamic data warehouses* to simultaneously take the dynamic aspect of the data in observed system and that of the new members in multidimensional data space into account.

3 Problems and Challenges

The special nature of dynamic data warehouses raises some real challenges for its realization. Among these, one of the principle challenges is to ensure freshness of data. This requires the integration of new transactions in the data warehouse as soon as they occur at the source systems. The data structures (e.g. index, materialized views) maintained for the performance improvement also need to be updated in real-time to make them available for analysis and decision making process. In dynamic data warehouses, these data structures must support online incremental updates without needing any complex computations and system down time. The issue can be addressed by the supporting online atomic facts insertion and by improving the performance of facts insertion, views maintenance and querying algorithms.

Another challenge while dealing with dynamic data warehouses is that as data is loaded through fast atomic insertions, it can not be ordered. As a result, these data warehouses do not operate in an ordered data space which is usually the case for traditional data warehouses. Indeed, the working data space of most data warehouses is composed of both naturally ordered (e.g. time) and non-ordered (sensors, location etc.) dimensions. Since ordered data renders the optimal techniques of data manipulation feasible, the members of naturally non-ordered dimensions are usually assigned numerical ids to introduce an artificial order in them. Such an ordering strategy improves the querying performance of the system, but might require reordering of entire data space, upon insertion of new dimension members, and hence is time consuming. Still, as data in classical data warehouses is loaded through offline ETL operations, this strategy is a reasonable one. Nonetheless, dynamic data warehouses can not afford to have such strategy because it would compromise the system availability. Moreover, the growth in the working data space of dynamic data warehouses is more important than in traditional data warehouses which usually operate in static or slowly changing data space. Therefore, we believe that the naturally non-ordered dimensions in dynamic data warehouses should remain non-ordered which is helpful in supporting dynamic growth of data space. However, the nature of ordered dimensions could be exploited so as to facilitate the manipulation of data (e.g. indexing, partitioning etc.). This makes dynamic data warehouses to operate in a mixed data space that is composed of both ordered and non-ordered dimensions. This raises the need of a specific data model for dynamic data warehouses.

4 Contribution

Our contribution in this research work is on partial cube materialization in a dynamic data warehousing environment. For this purpose, we first focus on the problems of data modeling which requires to consider both naturally ordered and non-ordered hierarchical dimensions for the description of data objects and algebraic operations. The materialization of views is achieved through data grouping structures that are stored and indexed in a tree like structure. The indexing structure operating in such an environment requires itself to be able to integrate the newly coming data dynamically. It should also be able to efficiently respond to OLAP queries which involve aggregation at different levels of granularity. Precisely, over contribution in this research work includes the proposition of:

- a data model for dynamic data warehouses. In our data model, we propose the concept of Hierarchical Hybrid Multidimensional Data Space (HHMDS) which constitutes of both ordered and non-ordered hierarchical dimensions. The axes of the data space are non-ordered which help their dynamic evolution without any need of reordering. We define a data grouping structure, called Minimum Bounding Space (MBS), that helps the efficient data partitioning of data in the space. Various operators, relations and metrics are defined which are used for the optimization of these data partitions and the analogies among classical OLAP concepts and the HHMDS are defined which lets us use the operations of classical data warehousing in HHMDS.

- a data structure called DyTree that indexes the MBS in its nodes and facts in the leaves with corresponding measure and aggregate values. The nodes holding MBS with associated aggregate values represent materialized sections of cuboids and the tree as a whole could be seen as a partially materialized data cube. Therefore, the DyTree is also a cubing structure. The views that are materialized (i.e. the nodes) are constructed at run time and guided by the proposed metrics. Algorithms for atomic insertion of a fact and querying the DyTree are provided.

- a methodology to experimentally evaluate the performance of the DyTree. For this purpose, a workflow has been set up. We outline input and output parameters for this workflow allowing us to assess the performance in different scenarios. Using input parameters, the size and nature of data and queries

sets is varied to see its effect on the output parameters which let us characterize and compare the performance of the trees. The methodology allows us to systematically analyze the DyTree from both quantitative and qualitative perspectives.

- a prototype that implements the experimental evaluation workflow and lets us evaluate the efficiency of the DyTree. The prototype has a range of features which include schema building, data and queries sets generation, trees construction, querying, 3D data and tree visualization and simulation utilities. The prototype helps us validate and evaluate the performance of our solution.

5 Organization of the Thesis

After discussing our motivation and giving a brief introduction about our research work in this chapter, we discuss our research work in more details in the next chapters:

- In chapter 2, a study of existing state-of-the-art solutions that deal with the issues related to ours is presented.
- In chapter 3 we present the data model for HHMDS and provide the details of our data grouping structure and related algebra.
- The structure of the DyTree and algorithms for its construction and querying are presented in chapter 4.
- In chapter 5, we detail our methodology for the experimental evaluation of the DyTree and discuss the obtained results. The details of our prototype implementing the experimental evaluation workflow and allowing the interactive simulation of the DyTree are also presented in this chapter.
- We conclude our discussion in chapter 6 by highlighting some strong and weak points of our solution and give some perspectives and possible directions for the future research works.

Chapter 2

Literature Review

Chapter Outline

1 Introduction

For over a decade now, the BI research community has paid a lot of attention to the works focusing on to reduce the time required to take a business action on the occurrence of some event in the concerned application. The delay between the happening of this event and the business action is called action time and can be considered as comprising of four components [Nguyen 2006]:

- Data latency: time between the occurrence of some event and the data storage in the warehouse,
- Analysis latency: time from the data being available for analysis to the time when the information is generated based on it,
- Decision latency: time between the delivery of information to the selection of some business strategy, and
- Response latency: time required for the implementation of new business strategy or decision.

In this research work, we rather focus our discussion on the first two components which aim for quick availability (achieved through real-time ETL) and fast integration of this data in data structures maintained for efficient analysis (e.g. aggregates, index etc.).

In the following, we present and discuss the existing research works addressing the issues related to these two components. We start our discussion by presenting some of the research works focusing on real-time ETL. We then discuss the data cubing and views materialization techniques which are largely used to pre-calculate and store the aggregates in a data warehouse to gain the performance advantage. Next, we consider techniques and methods that index data in multidimensional data space and contribute to the performance improvement.

2 Real-Time ETL

As discussed in last chapter, various terms (e.g. right-time, soft real time etc.) are used to refer very close concepts. In this section, we regroup these concepts under real-time data warehouses for the sake of simplicity.

Real-time data warehousing has attracted much attention of researchers during the last decade. Several studies have been carried out to deal with the issues in making it

a reality and improve its performance. Among these, some projects (e.g. zero-latency data warehousing [Nguyen 2003, Nguyen 2005, Nguyen 2007], LiveBI [Gupta 2011], DataDepot [Golab 2009a]) have been realized to propose architectural frameworks for these real-time data warehouses with the overall objective of reducing the action time. Here we focus our discussion on the solutions for the improvement of ETL process in real-time data warehouses. We categorize these research works into four categories (sometimes overlapping): process improvement, updates and queries scheduling, concurrency control and performance measurement.

2.1 Process Improvement

In this section, the research works focusing on to improve the efficiency of real-time ETL process are discussed. Among these, [Santos 2009] presents interesting study on data loading techniques in real-time data warehouses and propose a methodology supporting the continuous integration of incoming data while minimizing the impact on online analysis at the user end of the data warehouse. From the experimental study, the authors make an observation that the insertion of new rows in a table with no or very few contents is much less complex and faster than those in large size tables. Based on this observation, they propose a data warehouse loading methodology comprising of four steps: data warehouse schema adaptation, continuous data integration, OLAP queries adaptation and data area packing and re-optimization. In the first step, exact replica of all the tables that are suspected to welcome new insertions is created without any contents, primary key, index and referential integrity constraints. These replicated tables are named temporary tables. Data coming from source systems is inserted into these tables after all the necessary transformations. OLAP queries are adapted so as to take the schema adaptation into consideration. This is achieved by joining the data in temporary and permanent tables. Since, the continuous integration of data into temporary tables would degrade the insertion efficiency of new records after some time, the data in temporary table is integrated into the tables in data warehouse and all the temporary tables are recreated.

[Polyzotis 2007] discusses a particular aspect of continuous data loading into an active data warehouse, i.e. the meshing of stream updates with persistent data available in the data warehouse. The authors propose a join algorithm, called MESHJOIN, that is able to handle the fast arrival rate of the incoming streams and limited memory. The fast arrival rate is handled thanks to a so called tuple-shedding strategies which may cause the loss of some information. Consequently, the answer to queries are only approximate.

In [Thomsen 2008] a middle-ware system, called RiTE (Right-Time ETL), is presented. The data from source system is temporarily stored in a buffer at middle-ware (i.e. RiTE) unless it is committed by the producer or required by the data consumer. The buffer is managed by a so-called catalyst which is also responsible for data movement from the producer to the consumer. The purpose of the catalyst is to enable the fast movement of the data.

[Karakasidis 2005] presents an architectural framework for active data warehousing. The architecture is based on queuing networks where each queue is maintained to perform specific ETL activity.

2.2 Updates and Queries Scheduling

Since the new updates and queries may arrive at a very fast rate in real-time data warehouses, they are required to be scheduled for better performance. Dealing with this issue, [Thiele 2009] provides a dynamic scheduling algorithm for simultaneous updates and queries while assuring the user associated QoS (quality of service) and QoD (quality of data) criteria. The QoS criteria is used to optimize the response time while QoD is used to optimizing data freshness. The workload consists of two components, i.e. query workload and update workload. Each input query is accompanied with user preferences of QoS and QoD. A profit parameter is attached with each update specifying the user benefit if the update is applied. The correlation between queries and updates is determined thanks to data warehouse partitions: if same partitions are accessed by an update u and query q then u and q are correlated. Using this correlation information and user specified QoS and QoD criteria, the schedules are generated such that they are optimal under specified constraints.

The research work was extended in [Thiele 2010] to support the updates coming from several data sources. The authors propose two approaches, named local scheduling and global scheduling. In case of local scheduling, a scheduler is maintained at the site of each data source while for global scheduling only one global scheduler is used. The local schedulers are aware of the workload at their respective ends while the global scheduler is aware of the workload at all the data sources and stages of the data warehouse. A detailed comparative study of the global and scheduling is presented which shows the advantages and drawbacks of the two approaches.

In [Golab 2009b], the authors propose a scheduling algorithm for integration of incoming streams into a data warehouse. The proposed algorithm takes not only the priority of task into account but also the data freshness in base and/or derived tables. The algorithm aims at minimizing the overall staleness of data in the data warehouse.

2.3 Concurrency Control

As discussed earlier, updates are integrated in real-time data warehouses at the same time when it is being used for analysis. In other words, same data structures are accessed by updates and queries at the same time. This could lead to the conflict in access priority and may cause deadlocks. The issue can be resolved using an access control mechanism. This issue in real-time data warehouse is discussed in [Kim 2007]. The authors propose to maintain a global version for operational databases (VODB) and a separate version for data warehouse (VDW). The update transactions are stored in VODB before their integration in VDW while the queries are directly executed at VDW. Update transactions stored at VODB are integrated in VDW thanks to an operation called *publishing*. This integration of update transactions or in other words the view maintenance is achieved in a conflict-free publishing order by the help of so-called publish order graphs (POG). The POG are maintained such that there is no concurrency issue in the execution of the transactions. The authors propose an algorithm with its correctness proof to perform these operations and performance study of the algorithm proves its utility in increasing data freshness, or in other words, reducing the data latency.

2.4 Performance Measurement

To measure the performance of an ETL process, [Simitsis 2009] proposes a suite of quality metrics, referred as QoX. The suite includes both quantitative (e.g. performance, freshness, availability etc.) and qualitative (e.g. maintainability, flexibility etc.) metrics. The metrics from QoX suite are incorporated at all the stages of design process and guide its optimization. The interrelationships and dependencies among the metrics are discussed which lead to the tradeoffs for alternative optimization of the process. The metrics can be applied to any existing ETL framework to measure its performance. The QoX metric suite is employed in LiveBI [Castellanos 2010, Gupta 2011] project to guide the optimization of its ETL process.

3 Data Cubing

To reduce the analysis latency, an extensively used technique is pre-aggregation of data in cubes. The introduction of data cube [Gray 1997] which pre-calculates the aggregated measures for all level and dimension combinations of group-by was the first effort to improve OLAP query response time. This cubing, however, raises

considerable problems related to the complexity of calculation and storage of the data. Many research works have been carried out and various solutions are proposed to optimize the data cube computation and storage efficiency. We discuss some of these research works in the following sections.

3.1 Partial Data Cubing

One of the solution to address the cube storage problem is partial data cubing i.e. to pre-compute only a subset of all possible aggregates (also called views). In literature, numerous research solutions can be found for efficient and strategic selection of such subset of views to materialize in a data warehouse. These solutions provide a reasonable compromise between space usage and query response time. Among these, [Harinarayan 1996] was the first work dealing with the issue which discussed the need of partial view materialization and proposed a greedy algorithm to select a subset of views to materialize aimed at reducing the query response time. The research work was extended by [Gupta 1997b], but still do not consider the update cost of views. [Gupta 1997a] proposes another framework to address the issue and proposes a polynomial time heuristic to optimize query response time. The selection of views is based on AND/OR graphs and the proposed algorithm takes the update cost into account. Authors of [Baralis 1997] propose a technique to reduce the solution space on the basis of the prior information related to the user specified queries. A heuristic based on the size of the candidate views is used to further reduce the solution space. [Zhang 1999] proposes a genetic algorithm for the same problem of views selection while in [Aouiche 2009] a strategy to select a subset of materialized views and an index (from a set of candidate indices) is discussed and a data-mining based approach is employed for the purpose.

3.2 Space Efficient Data Cubing

Apart from partial data cubing, many other research works have also addressed the issue of space consumption and have proposed space efficient storage ways for the purpose. The basic principle of these works is based on avoiding storing unnecessary or redundant information.

In [Wang 2002] the authors propose a structure called Condensed Cube for reducing the size of data cube which in turn reduces the space and computation overheads. The condensed data cube is an uncompressed fully computed data cube and does not require any compression/decompression. The idea is based on identifying *base*

single tuples (BST) which on a subset of dimensions SD is defined as the only tuple in a vertically partitioned (on SD) fact table. This definition implies that the aggregate values of all the cuboids on the subsets of SD, are same as the aggregated value of the BST. Therefore, only BST are physically stored in a cube while the other cuboids are generated (when required) from these without needing much computation. [Feng 2004a] proposes further condensing in Condensed Cube using the prefix sharing technique and propose another structure called PrefixCube.

Quotient Cube [Lakshmanan 2002] is based on idea of partitioning the cube cells in classes such that all the cells in a class have same aggregate value in addition so some filtering criteria (e.g. minimum count). The classes so formed are arranged in lattice and replace the original cube. The classes are generated in such a way that even without keeping the information about all the cells of a data cube, the drill-down/roll-up semantics are preserved. QC-Tree [Lakshmanan 2003] is used to efficiently store the quotient cube. PMC (Partially Materialized Cube) is an extension of QC-Tree for partial views materialization. Less number of views are materialized in PMC while additional information is stored in each node to still be able to compute all the aggregates. This results in better performance for integration of new updates at the cost of querying performance. Range Cube [Feng 2004b] is also a similar solution proposing a compact structure. The idea of Range Cube is based on generating range tries using the correlation among data tuples of the base fact table. A single scan is needed over a fact table to construct the trie. The nodes of the trie hold only non-redundant dimension values and the associated aggregates. Number of nodes in generated tries is based on the correlation of data and has a strong effect on the performance of the Range Cube's computation, memory usage and querying performance.

[Morfonios 2006] proposes a ROLAP cubing method called CURE (Cubing Using a ROLAP Engine) that computes whole data cube over very large data space constituted of hierarchical dimensions. CURE uses an efficient algorithm for partitioning fact table that helps improving the cube computation speed. Efficient cube storage is achieved by removing the dimensional (repetition of dimension values) and aggregational (repetition of aggregate values) redundancy. The work was extended to make online incremental updates possible in [Zhang 2008]. The authors have proposed a solution, named DOLUS (Dynamic Online Updating Solution) that works with CURE. The DOLUS can also be used independently without CURE. The incremental updates in DOLUS, however, are not atomic. The tuples from fact table are stored in a buffered pool where they are sorted before being integrated into cube.

Some research works have discussed the cube computation problem on the basis of the density of data cubes. Among these, [Ross 1997] proposes an algorithm named PartitionedCube for efficient computation of sparse data cubes over a large data space. The idea is to partition a large fact table into smaller tables and create data cubes over these small partitioned data tables. [Yu-cai 2004] also proposes an algorithm for computation of sparse data cubes which exploits the functional dependencies that may exist among different dimensions. The algorithm assumes that each dimension value is an integer between zero and its cardinality which is known in advance. Since levels in hierarchical dimensions can be assumed to be different functional dependent dimensions (e.g. Country and City can be assumed to be two different dimensions where City is functionally dependent on Country), cube with hierarchical dimensions can also be computed.

3.3 Data Cubing in a Dynamic Environment

Dimensions in data warehouse are generally believed to be static or slowly changing [Kimball 1996]. This, however, is not always valid and there exist the data warehousing applications where data in dimension tables evolves over the time, i.e. they operate in a dynamic environment. These data warehouses are also called temporal data warehouses. A large amount of research work dealing with the issues related to temporal data warehouses is available in literature. These research works can be categorized in five categories [Golfarelli 2009], i.e works on: changes in data source (e.g. [Amo 2000, Chen 2001]), data changes in data mart (e.g. [Hurtado 1999, Gupta 1995]), schema changes in data mart (e.g. [Hurtado 1999, Bellahsene 2002, Body 2002, Ravat 2006, Favre 2007]), querying temporal data (e.g. [Wrembel 2006, Mendelzon 2000]) and designing temporal data warehouse (e.g. [Malinowski 2008, Rizzi 2006]). The focus of these research works is mostly on providing models to support evolution in data warehouses including dimension updates, hierarchy updates, levels updates, instances updates, facts updates etc.

The computation of data cube under such an evolving environment is discussed in [Geffner 2000]. The authors propose a cube computation approach for dynamic data warehousing environment, called Dynamic Data Cube (DDC) . A data space of DDC is a multidimensional array in which the cardinality of each dimension equals the size of dimension's domain set. The data space is partitioned thanks to so-called overlay boxes which are completely disjoint hyper-rectangles. Each overlay box stores only a small number of values that represent row sums of the cell values in each dimension. The DDC proposes to store these overlay boxes in a B-Tree based tree structure called

B^c-Tree (Cumulative B-Tree). A separate B^c-Tree is maintained for each set of row sum values. A 2-dimensional overlay box has two set of row sum values each of which is one-dimensional. In case of a d-dimensional overlay box, we will have d groups of row sum values each of which will be (d-1)-dimensional. Therefore, a d-dimensional overlay box is stored in (d-1)-dimensional dynamic data cube , recursively unless we get to d=2. There, the 2-dimensional DDC is stored in two B^c-Trees. The addition of a new dimension value or in other words, a new cell requires the creation of a new root above the existing one in the B^c-Tree. The new root has a size that is twice the size of the replaced root in each dimension. This causes the number of levels in the tree to grow. The tree can be maintained incrementally, hence the dynamic growth of the cube becomes possible. The preceding description makes it clear that a higher dimensional DDC needs complex computations and requires a lot of space. The issue of space efficiency of DDC was later addressed in [Riedewald 2000] without much effecting the query and update costs. This research work consider only numerical dimensions for the calculation of range sums, which are naturally totally ordered. No study was carried out to see the effect of increase in number of dimensions on these approaches.

4 Data Indexing

Data in warehouses is multidimensional in nature. One of the approaches to index multidimensional data is to transform it into one-dimensional data using space filling curves (e.g. z-ordering [Orenstein 1984], Hilbert curve [Faloutsos 1989], Peano curve [Morton 1966], gray ordering [Faloutsos 1988], etc.). The transformed data can then be indexed using one dimensional indexing techniques such as B-Tree [Bayer 1972]. All these space filling curves propose to partition the data space using a grid and then each of the resultant grid cells is labeled with a unique number determining the position of that cell in the introduced total order. The total order is then used while indexing the data lying in the multidimensional data space. While labeling the grid cells, it is assured that there is a high probability of spatial proximity of the data points is being preserved. The clear advantage of using the space filling curves is that they are insensitive to the number of dimensions if the one dimensional keys can be arbitrarily large [Gaede 1998]. However, we do not detail these works focusing on space filling curves and one dimensional indexing techniques here and rather focus on the indexing techniques that propose to index the data directly in a multidimensional data space or can be used to index the data in a warehousing context.

In this section, we first discuss the data indexing techniques for multidimensional databases and then present a study on specific indexing techniques for data warehouses. We discuss the indexing techniques dealing with the data lying in partially or non-ordered data space at the end of this section.

4.1 Data Indexing for Multidimensional Databases

In this section, we discuss the multidimensional data indexing techniques that directly index the data in a multidimensional data space. We classify these indexing techniques into two groups: (1) space partitioning indexing techniques and (2) data partitioning indexing techniques. In space partitioning indexing techniques, partitions of data space are stored in some data structure (mostly a tree structure). The process of partitioning a data space is usually very straight forward and the index always covers the whole data space. This, however, may result in poor space utilization in case of sparse data sets. The data partitioning techniques, on the other hand, do not suffer from the inefficient space utilization. The data partitions do not cover all the data space but as all the existing data is indexed in the data structures, they can easily provide accurate answers to the queries. The process of creating data partitions, however, is more complicated than the space partitions. In the following, we discuss important works from each of these classes.

4.1.1 Space Partitioning Techniques

Among the space partitioning techniques for multidimensional data indexing, quadtree [Finkel 1974] proposed by Finkel and Bentley is one of the initial structures. The initial quadtree was proposed for indexing of point data in 2-dimensional data space, but the idea could be easily extended to the higher dimensional data space. A two dimensional data space in a quadtree implementation is divided into four quadrants or partitions (referred as NW, NE, SW, SE) every time the space needs to be divided which explains the origin of the name "quadtree". In other words, every non-leaf node of a quadtree has four children (2^n in case of an n-dimensional data space). In a quadtree, data points themselves serve as the points of division, therefore the created partitions need not to be of equal size and the tree itself is not guaranteed to be balanced. The authors have also proposed an algorithm to build an optimized structure called optimized quadtree if all the data points are known a priori. Figure 2.1 shows an example point quadtree in 2-dimensional data space.

Many variants (e.g. [Samet 1984, Hunter 1979, Vassilakopoulos 1993, Bai 2006]) of the quadtree were proposed which can also index other complex objects (e.g. lines,

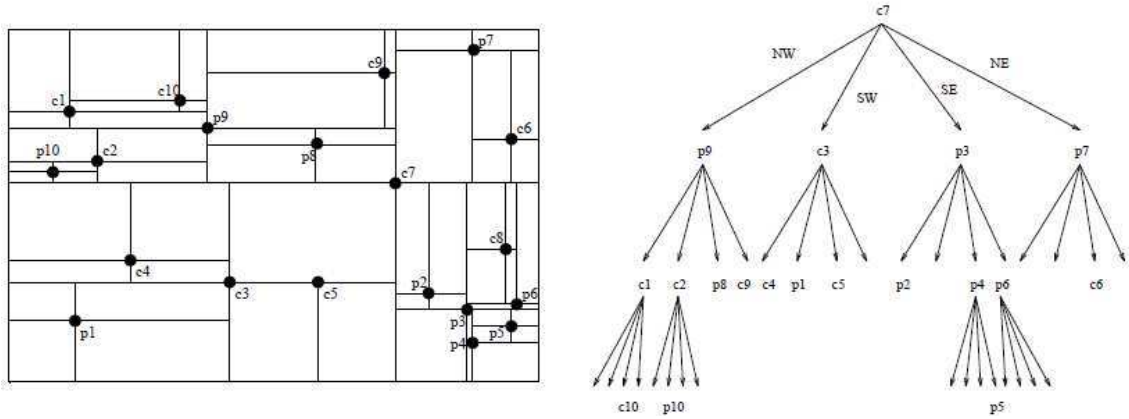


Figure 2.1: An example of point QuadTree [Gaede 1998]

polygons etc.) and/or provide different space decomposition methods. All these methods, however, work on the same basic principle of decomposing the n-dimensional data space in 2^n partitions.

As evident from the above explanation, increase in number of dimensions increases the number of created partitions at the decomposition of the data space and so as the fan out of a quadtree node. The fan out of the nodes can be controlled using another prominent space partitioning technique based indexing structure called k-d-tree [Bentley 1975]. The k-d-tree works on a principle of binary search tree and the data space is partitioned (when required) along only one dimension at a time. Each internal node of a k-d-tree has one or two children and each serves as a discriminator to guide the search in the tree. The k-d-tree can index only point data and can not be used to index (without any transformation) complex objects. The k-d-tree is an un-balanced tree structure and it is very sensitive to the order in which the data points are inserted. Like quadtree, many variants [Orenstein 1982, Orenstein 1984, Foley 2005, Zhou 2008] of k-d-tree are also proposed which mainly differ from the original k-d-tree on the basis of the choice of the space decomposition point/axis and some of them can be used to index the complex objects. k-d-tree is basically an in-memory data structure that is not suitable for indexing data in large spatial data space.

As the space partitioning techniques based on space decomposition, they do not take the distribution of data into account and could possibly index large dead spaces which could severely affect the performance of indexing solution.

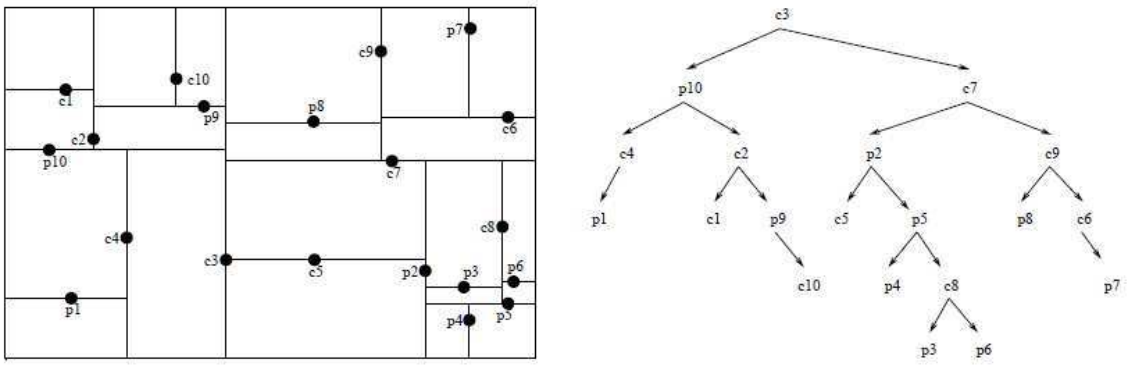


Figure 2.2: An example of point kd-tree [Gaede 1998]

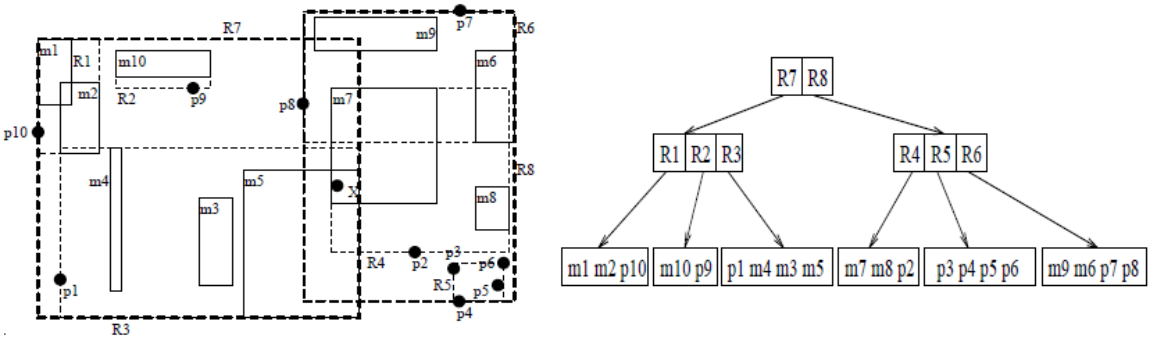


Figure 2.3: An example of R-Tree [Gaede 1998]

4.1.2 Data Partitioning Techniques

The most prominent indexing structure based on data partitioning technique is the R-Tree [Guttman 1984]. R-Tree is a multidimensional generalization of B-Tree and has a balanced structure. The R-Tree indexes the n-dimensional minimum bounding rectangles (MBR) in its nodes which are the data partitions constructed around a set of objects. The leaf nodes of an R-Tree holds the entries consisting of a *reference* pointing to a database object and an MBR bounding the data objects. The non-leaf nodes have similar type of entries where the reference points to a child node while the MBR bounds all entries in the child node.

Figure 2.3 shows an example R-Tree in a 2-dimensional data space where dotted rectangles show the MBR of internal nodes while the solid rectangles are the representation of MBR constructed around the data objects.

The nodes of an R-Tree can have overlapped regions which could degrade the search and consequently the insertion and deletion efficiency of the R-Tree. To overcome such a problem, R^+-tree [Sellis 1987] was proposed. The nodes in R^+-tree are not allowed to overlap using the mechanism of clipping. If two data objects intersect

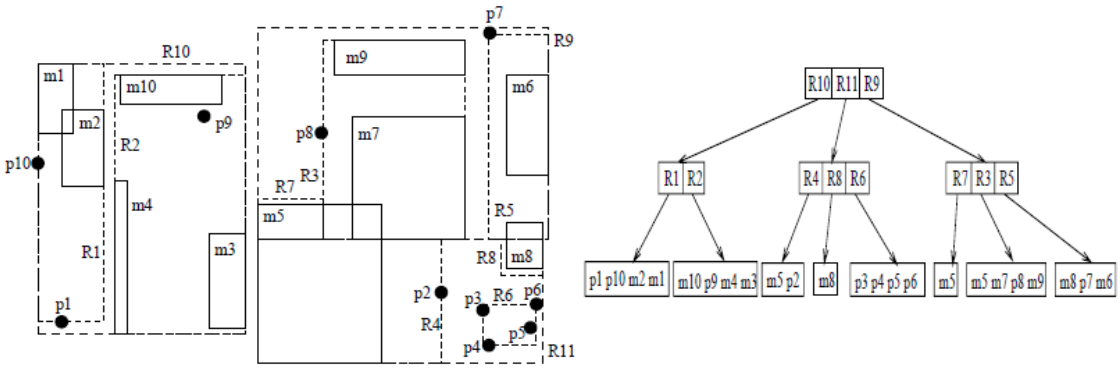


Figure 2.4: An example of R+-Tree [Gaede 1998]

more than one MBR at the same level, they are clipped and stored in more than one leaf nodes. Such a strategy is helpful in improving the efficiency of point queries but a considerable degradation in query response time could be seen in case of range queries. Similarly the insertion of a new object might also require traversing multiple nodes.

Another variant of R-Tree, based on experimental study, was proposed with the name R*-Tree [Beckmann 1990]. The authors carried out an experimental study using different data distributions and identified many weaknesses in the initial R-Tree. The author proposed a *forced reinsert* policy: if a node overflows, it is not split right away. Rather, first some entries (about 30%) of the overflowing node are removed from the entries and reinserted in the tree. In R*-Tree, the insertion of new data objects is carried out while taking other parameters, such as minimal overlap, minimal perimeters and maximum space utilization, into consideration. The authors report a considerable performance improvement (up to 50%) using their proposed solution.

Many other interesting variants [Günther 1989, Kamel 1994, White 1996] with slightly different splitting and/or data partitioning strategy are proposed. However, as [Berchtold 1996] shows, all these indexing techniques are not suitable for high-dimensional data indexing. The increase in number of dimensions of a data space, in fact, has a broad variety effect on data indexing which can not be witnessed unless we experiment with high dimensional data. In literature, such affects are referred as *curse of dimensionality*. As the R-Tree and its most variants were tested for low dimensional data, such effects were not witnessed. One of the reason for witnessing such effects in high dimensional data space is the increase in overlapped area as a result of increase in the number of dimensions which results in a larger amount of dead space and rapidly degrades the insert and querying efficiency of the indexing techniques. To

address the issue, Berchtold et al. have proposed an indexing structure called X-Tree [Berchtold 1996]. The structure is primarily based on R*-Tree but introduces a new splitting algorithm focusing on finding an overlap free split. The authors propose a concept of super nodes which are larger capacity nodes and come into being when no suitable split (controlled through the maximum amount of allowed overlap) is found. The authors also propose to record the split history (in a binary tree) which is then used to find the split axis serving to optimize the splitting process. Many other high dimensional data indexing techniques are also discussed in literature, some of which are discussed in [Böhm 2001].

4.2 Data Indexing for Data Warehouses and OLAP

Data in data warehouses is stored for the purpose of OLAP analysis which involves querying the aggregated data at different levels of granularity characterized by hierarchical dimensions in a data warehouse. The performance of this querying process can be improved by considering such requirements while indexing the data in a warehouse. In this section we discuss the research works taking these constraints of pre-aggregation and hierarchical dimensions into account. We classify these works in four categories: bitmap based indexing technique, tree based indexing techniques, graph based indexing techniques and hash table based indexing techniques. The discussion on the solutions of each of these categories is presented in the following.

4.2.1 Bitmap based Indexing Techniques

Bitmap indexes [O'Neil 1989, O'Neil 1997] are largely used in indexing the data for data warehouses. The original bitmap index is a one-dimensional index and its idea is based on creating bitmaps for each value of an attribute (or dimension) A to index in a table T. A query can be efficiently answered using Boolean operations (such as AND, OR, NOT) on the bitmaps. The main disadvantage of a bitmap index is the size of bitmap index which increases with the increase in size of data sets as well as increase in the cardinalities of the indexed attribute [Wu 1998] and become too large to efficiently deal with in the main memory. To deal with the space consumption problem of bitmap index, different encoding [Wu 1998, Koudas 2000, Rotem 2005, Chan 1998] and compression techniques [Xi 2008, Xiao 2009] are proposed that are able to reduce the size of bitmap indices but with computation overheads. Another index, called bitmap join index (BJI) [O'Neil 1995], is also a well know solution for data warehouse indexing. A BJI speeds up the star-join queries by considerably reducing the volume

of data that needs to be joined. Some studies (e.g. [Aouiche 2005, Bellatreche 2007, Bellatreche 2010]) have also been carried out to select the optimal BJI for a data warehouse. All these bitmap indices, however, are used to index the data at the detailed level of facts and do not have any support for hierarchical dimensions. HOBI [Chmiel 2009] is one of the index that is used to index the data at different hierarchical levels. Separate bitmap indices are used for each level to efficiently support the OLAP queries (see figure 2.5). A query asking for aggregate at higher level of hierarchy needs to consult only the corresponding index which stores direct pointers to the items in fact table. The same authors propose a variant of HOBI, called Time-HOBI [Chmiel 2010], which incorporates the time dimension at each level of hierarchically organized bitmaps and supports the time-window queries more efficiently.

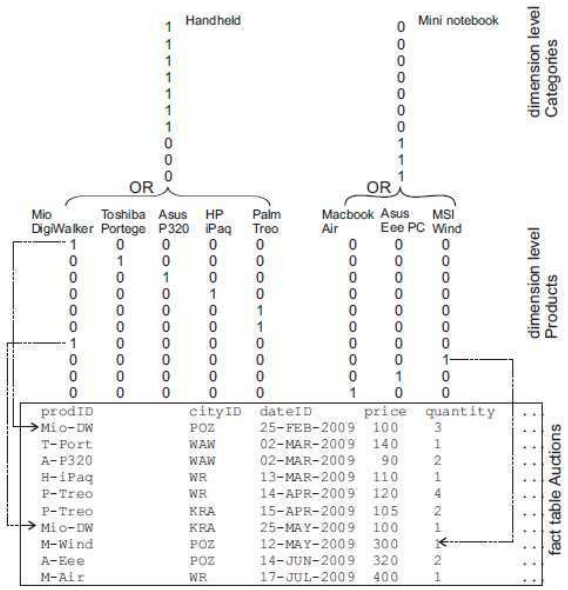
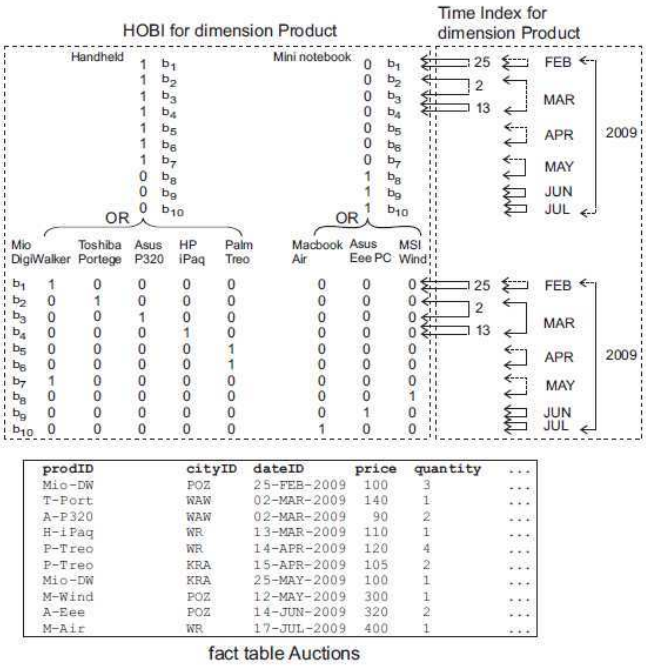


Figure 2.5: An example of HOBI [Chmiel 2010]

4.2.2 Tree based Indexing Techniques

Tree based indexing techniques are heavily used to index the data in various contexts. Many tree based indexing solutions which were originally not proposed for data warehouses, can be used to serve the purpose. For example, some multidimensional variants of B-Tree (e.g. [Govindarajan 2002, Fenk 2000]) can be used to index data in warehouses.

In [Roussopoulos 1997], the authors have proposed an Extended Datacube Model (EDM) to model the data cube in a spatial-like data space and propose to index

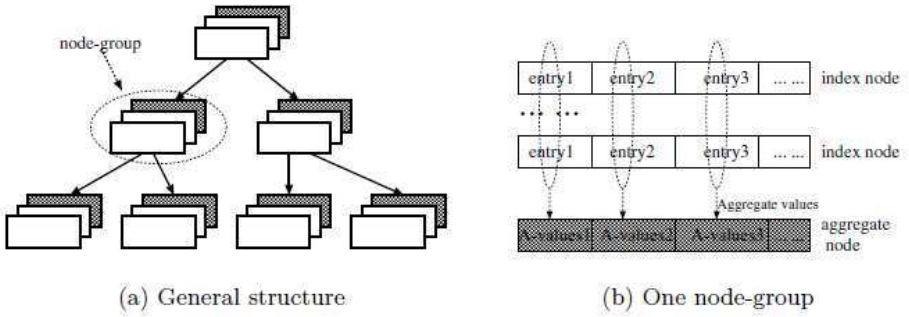


prodID	cityID	dateID	price	quantity	...
Mio-DW	POZ	25-FEB-2009	100	3	...
T-Port	WAW	02-MAR-2009	140	1	...
A-P320	WAW	02-MAR-2009	90	2	...
H-iPaq	WR	13-MAR-2009	110	1	...
P-Treo	WR	14-APR-2009	120	4	...
P-Treo	KRA	15-APR-2009	105	2	...
Mio-DW	KRA	25-MAY-2009	100	1	...
M-Wind	POZ	12-MAY-2009	300	1	...
A-Eee	POZ	14-JUN-2009	320	2	...
M-Air	WR	17-JUL-2009	400	1	...

Figure 2.6: An example of Time-HOBI [Chmiel 2010]

the data using a packed R-Tree [Roussopoulos 1985]. The data cube so indexed is termed as Cubetree. The authors propose the sorting and bulk loading techniques and show that atomic incremental updates of the structure or complete re-computation are not the viable solutions. The same research group in [Kotidis 1998], proposes to store the ROLAP views in Cubetrees. Feng and Makinouchi extend the research work on Cubetree to efficiently support partially-dimensional (involving a subset of dimensions) range queries [Feng 2006] and propose a new structure called Aggregate-Tree (Ag-Tree). The authors propose to maintain a group of nodes for every node of a Cubetree. The Cubetree nodes hold multidimensional MBR while in Ag-Tree a group of nodes each indexing data in one dimension (see figure 2.7). The research work is further investigated and the authors propose to sort the entries in the nodes of Ag-Tree, giving rise to another structure called Ag+-Tree [Feng 2011].

In [Papadias 2001], authors propose an R-Tree based indexing structure for spatial data warehouses, called aggregation tree (aR-Tree). The internal nodes in aR-Tree store, with their MBR, a pre-aggregated value obtained by some aggregation function on all the data object enclosed in the MBR. [You 2006] puts forward a hybrid indexing structure for spatio-temporal data warehouses. The indexing structure is composed of an aR-Tree, indexing the spatial dimension, and hash tables indexing the temporal dimensions. A hash table indexing the data at different temporal levels (e.g. year,



(a) General structure (b) One node-group

Figure 2.7: An example of an Ag-Tree's structure [Feng 2006]

month, day, hour) is linked to every node of the spatial part. The aggregated value associated to the nodes of aR-Tree represent the aggregation for time level ALL for the data enclosed in the MBR of the node.

[Ester 2000] proposes a dynamic indexing technique called DC-Tree that supports atomic incremental maintenance of aggregates in a data warehouse. The data structure is an extension of X-Tree with support of hierarchical dimensions. No total order among the members of dimensions is maintained, which needed the modification in definition of MBR used in X-Tree. For this purpose, the authors propose the concept of Minimum Describing Sequence (MDS) which is a sequence of n sets (n being the number of dimensions) representing the edges of MDS. These MDS are stored in nodes of the DC-Tree as the MBR in X-Tree.

In [Johnson 1996] authors define the concept of cube forest. In the cube forest a tree is defined by a template determining the partial order in which attributes are indexed. The nodes of the tree are the search structures (e.g. B-Tree) indexing different values of an attribute in a cube tree's node. A cube forest could be defined as a union different trees with elimination of redundancies among different attributes. The authors also propose and implemented a solution to incorporate hierarchical dimensions in the cube forest. The template determining the partial order, is crucial to performance and have a strong effect. The updates are carried out in batch. A pruning strategy is proposed to address the issue of space efficiency.

4.2.3 Graph based Indexing Technique

Sismanis et al. propose a compressed directed acyclic graph (DAG) based data structure, named Dwarf [Sismanis 2002] which simultaneously stores and indexes the aggregated data. The storage problem is addressed by removing the prefix and suffix redundancies. A common prefix or suffix is stored only once in a cube helping reduce

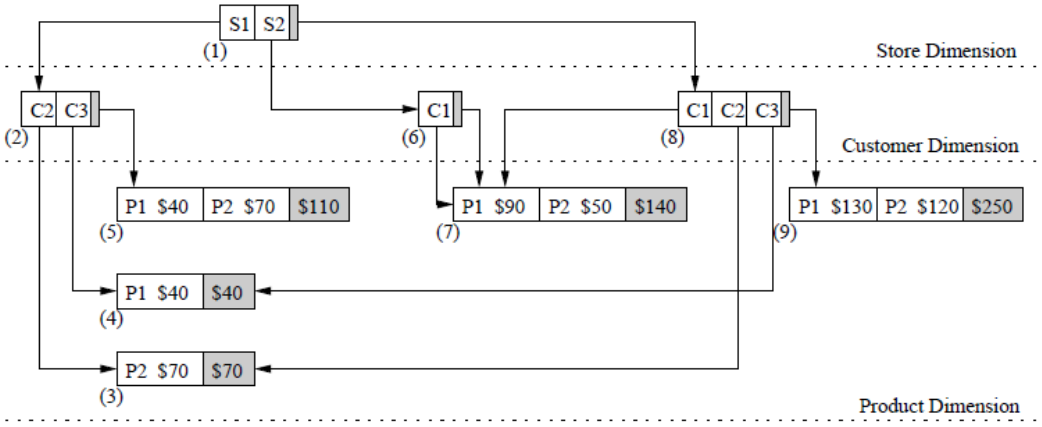


Figure 2.8: An example of a Dwarf's structure [Sismanis 2002]

the storage space (see figure 2.8). Dwarf requires all the dimension attributes to be of integer type, and hence the attributes with other data types need to be mapped to integers. The choice of dimension ordering while indexing the data in a Dwarf structure is very important and could severely effect the performance. The construction of a Dwarf structure requires the fact table to be sorted which renders the atomic incremental update to be impractical. The initial Dwarf structure did not have any explicit support for queries involving aggregation at different levels of hierarchy, the issue later discussed by the researchers in [Sismanis 2003]. These works are efficient for data indexing but do not provide algorithms for atomic incremental updates and do not scale well with increasing number of dimensions.

4.2.4 Hash Table based Indexing Technique

In [Doka 2011], the authors propose a distributed system that uses an adaptive indexing scheme, based on distributed hash tables (DHT), for online queries involving aggregation at different hierarchical levels. The fact table is distributively stored and indexed at different sites over the network, called peers. Initially, a default levels combination is indexed at each peer to effectively answer the queries at that level of aggregation. The statistics (e.g. queries misses) about the queries executed at each peer are stored locally which help the adaptation of the index at some different level of aggregation. The queries concerning the indexed levels can be answered directly at the concerned peer while the queries at different aggregation levels require the global processing for which the queries are flooded in the system. The system supports the approximate queries only and as evident from the obtained results, the precision decreases considerably with increase in number of dimensions.

4.3 Data Indexing in Partially or Non Ordered Data Spaces

Solutions proposed for multidimensional data management (e.g. indexing, partitioning etc.) mostly manage the data in continuous ordered data space [Chen 2009a].

In literature, we find a reasonable amount of research works dealing with the problems in partially or non-ordered data spaces. Among these, [Qian 2003] propose a data indexing technique, named ND-Tree, by mapping the geometrical concepts of continuous ordered data space into a non-ordered discrete data space. The authors propose the concept of discrete minimum bounding rectangles (DMBR) which is an alternative of MBR of R-Tree in a discrete non-ordered data space. The DMBR are stored in a tree structure inspired from R*-Tree. The algebraic metrics and operators (e.g. area, overlap) and algorithms to construct and query the tree are adapted to work in the discrete non-ordered data space.

In [Qian 2006], the same authors propose another indexing technique, called NSP-Tree, for non-ordered data space, this time based on space partitioning technique. The solution is aimed at minimizing the overlap among the nodes which degrade the ND-Tree's performance with increase in overlap. As the NSP-Tree is based on space partitioning strategy, the nodes of NSP-Tree do not overlap each other. The split history, while partitioning the data space, is maintained in an unbalanced binary tree, called SHT. The NSP-Tree indexes only the existing data space which could grow with the arrival of new data items. The NSP-Tree has a balanced tree structure whose non-leaf nodes hold the indexed regions in form of DMBR, a pointer to an SHT and pointer to child nodes. The leaf nodes hold the key of the indexed vector in the data space and a pointer to the indexed object.

Later, [Chen 2009b] propose an indexing technique, called C-ND-Tree, for hybrid data space. The DMBR are replaced by hybrid minimum bounding rectangles (HMBR) and the algebraic metrics and operators are adapted to take the nature (ordered vs non-ordered) of constituting dimensions into account. The strategies to make ordered and non-ordered dimensions comparable are proposed. HMBR are stored in a structure similar to that of the R-Tree and the algorithms are accordingly adapted.

5 Conclusion

Research works presented on real-time ETL have mainly focused on loading and scheduling strategies for incoming tuples and do not discuss the issue cube computation or data indexing.

Partial data cubing and views selection techniques are helpful in improving space and cube computation efficiency but mostly require the knowledge about frequently used queries which is not possible in dynamic data warehouses. Space efficient cube computation techniques provide interesting basis to address our problem but either do not address the issue of atomic incremental maintenance of the cube or do not scale well with the increase in number of dimensions. DDC proposing a solution for data cubing in dynamic environment considers only flat dimensions with members having numerical values.

Multidimensional data indexing techniques proposed in relational/spatial context can be considered for data indexing for warehouses but these do not support high dimensional data indexing. Even those proposed for high dimensional data indexing, support only "flat" dimensions while in data warehousing environment we normally deal with hierarchical dimensions.

Bitmap index is very often used in traditional data warehouses and indexes the data along one flat dimension only. Some variants supporting hierarchical dimensions [Chmiel 2009, Chmiel 2010] are also proposed. The bitmap indexes, however, apart from being space inefficient (the issue that could be addressed by employing encoding and/or compression techniques), have expensive insert/delete operations and therefore, are not suitable for the systems that are frequently updated.

R-Tree based indexing techniques, supporting pre-aggregation or not, are not suitable for high dimensional data indexing as shown in [Berchtold 1996]. The DC-Tree is a suitable solution for atomic incremental updates, supports hierarchical dimensions and pre-aggregation and scales well in high dimensional data space. The DC-Tree is also a good candidate for indexing the data in data warehouses whose data space grows dynamically over the time. However, the DC-Tree lacks a formal data model, does not take advantage of the naturally totally ordered dimensions (if any in the data space) and have a costly split algorithm. The DC-Tree aims at minimizing the overlap among nodes by creating super nodes, but as the number of super nodes grow the performance starts to degrade. C-ND Tree provides an idea of data indexing and modeling in a hybrid data space, however deals with flat dimensions only.

We take inspiration from these above discussed research works and propose a solution for efficient partial data cubing considering the special characteristics of data and a dynamic data warehousing environment, such as multidimensional nature of data, hierarchical dimensions, fast update, dynamic growth of the data space etc.

Chapter 3

Data Model for Hierarchical Hybrid Multidimensional Data Space

In this chapter, we present a multidimensional data model for a dynamic warehouse's data space that is made of both ordered and non-ordered hierarchical dimensions. We establish the analogies among different data warehousing and OLAP concepts and the spatial data space. We provide formal definitions and explain these concepts in terms of OLAP. Relations, operators and metrics are defined to represent, manipulate or compare the multidimensional objects lying in the data space. The concepts of hyperplane and minimum bounding space are introduced which facilitate the grouping and partitioning of data in the space.

Chapter Outline

1 Introduction

As discussed in the previous chapters, the dynamic data warehouses operate in a rapidly changing environment where the working data space keeps growing with time. In such a data warehousing environment, the rearrangement of dimension spaces, while the systems are online, is not possible and, therefore, the data space is composed of both ordered and non-ordered hierarchical dimensions. Description of the objects lying in such a space need a specific data model that can provide the abstraction for them. Moreover, operations, relations and metrics are also needed to compare, characterize and manipulate these objects. The analogies must also be drawn among the concepts related to such a data space and traditional data warehousing environment.

Existing models (e.g. [Li 2004], [Agarwal 2011], [Kuznetsov 2009] etc.) provide abstraction for data warehouses in ordered data space. Some other works (such as [Chen 2009b]) which address the issue of modeling non-ordered data space, do not have support for hierarchical dimensions which are the basis of data warehouse.

In this chapter, we propose a data model that deals with both ordered and non-ordered hierarchical dimensions. This chapter presents the concepts and notations needed to describe such a data space as well as various relations, operators and metrics necessary to perform a multidimensional data analysis in the data space.

2 Illustrating Toy Example

As an illustrating example, we choose the classical application which aims at analyzing the sales of products with respect to time duration and location. In such an application, we consider a very simple data warehouse that is built over a schema with only two dimensions i.e. Location and Time. The locations and time values in the application can be described at different levels of granularity, making them to be hierarchical. These different levels of granularity serve to detail or summarize the sales recorded in the application. In this example we consider that the levels for the dimension location are $ALL_{Location}$, Regions and City while the Time dimension consists of levels ALL_{Time}, Year and Semester. In reality, members of each level could be described by different attributes (e.g. a Region can be described by its ID, name, number of inhabitants, capital etc.) but for the sake of simplicity, we use only name of these members to designate them. The resulting dimension hierarchies for this application are presented in figure 3.1. The numerical value of interest in this

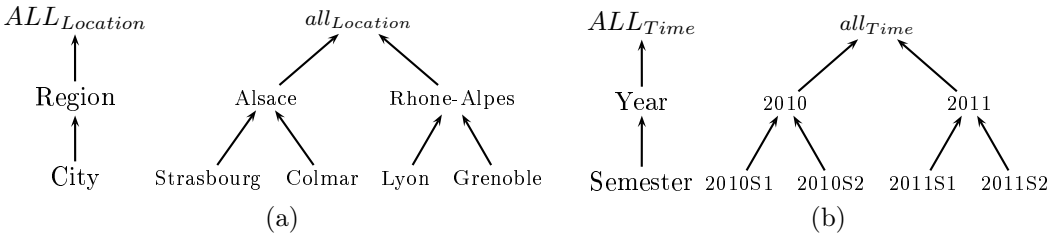


Figure 3.1: Dimension Hierarchies and Instances of dimension (a) Location and (b) Time, for the illustrating toy example.

application is quantity of sales which serves as measure and SUM is used as aggregate function.

Obviously this simplistic example does not explain a real-time scenario, but it is generalizable to other cases. We use such an example just to simplify the explanation and make it easy for the reader to understand our proposal.

3 Data Model

In this section, we detail the concepts related to our proposed multidimensional data space. First, we provide the notations for basic OLAP concepts (e.g. hierarchical dimension, level etc.) and then propose and present other concepts that are necessary to describe and manipulate our data space.

3.1 Hierarchical Dimensions

A dimension is called *hierarchical*, if its domain could be completely specified by different sets categorizing the same domain at different levels of detail and these sets can be arranged in a parent/child relationship among themselves. For example, the domain of the dimension Location could be covered by the sets {Rhône-Alpes, Alsace} and {Strasbourg, Colmar, Lyon, Grenoble} and could be arranged in a parent/child relationship as Region/City. The position of the different domain sets contributing in the hierarchical organization of a dimension is called a *level*. We note l_i^j the j^{th} level of a hierarchical dimension D_i and ALL_i the top most hierarchical level of D_i. Domain set of ALL_i is a singleton $\{all_i\}$.

In figure 3.1, $l^1_{Location}$ corresponds to the lowest level City of dimension Location, $l^2_{Location}$ is Region etc. and $domain$(City)= {Strasbourg, Colmar, Lyon, Grenoble}.

Formally, a *dimension hierarchy* is a directed acyclic graph $\mathcal{H}$ whose nodes are the elements of L_i=$\{l_i^1, l_i^2, ..., l_i^{k-1}, ALL_i\}$ of dimension D_i having k levels of hierarchy and ALL_i is the *sink* (i.e. reachable from every node in $\mathcal{H}$). For $l_i^u, l_i^v \in L_i$, we note $l_i^u \uparrow l_i^v$ if there exists a path from l_i^v to l_i^u in $\mathcal{H}$ where $\uparrow$ is a reflexive, transitive and asymmetric relation.

We define S_{D_i}, the *dimension space* of dimension D_i having k levels as the union of the domain sets of all the levels involved in dimension hierarchy, i.e. $S_{D_i} = \bigcup_{j=1}^{k} domain(l_i^j)$. We write $level(m) = l_i^j$, when $m \in domain(l_i^j)$. An elements of S_{D_i} is called a *member*. For example, $S_{Location}$ ={Colmar, Lyon,..., Rhône-Alpes, Alsace, $all_{Location}$} and $level(Colmar) = City$.

An *instance of a hierarchical dimension* D_i is a directed acyclic graph $\mathcal{I}$ whose nodes are the members of S_{D_i} and all_i is the sink. For m, $n \in S_{D_i}$, we note $m \Uparrow n$ and state "m is reachable from n", if there exists a path from n to m in $\mathcal{I}$. $\Uparrow$ is a reflexive, transitive and asymmetric relation.

Let us note that in this multidimensional model all the dimensions are hierarchical with partial order $\Uparrow$ among the members at different levels of dimension hierarchy. However, the members of the same level are not essentially required to be ordered among themselves. We make the following distinction among these dimensions:

- Dimensions for which the members of the domain sets of the same level are non-ordered. For example, Strasbourg and Grenoble at level City in dimension Location are non-ordered, while Alsace$\Uparrow$Colmar and Rhône-Alpes$\Uparrow$Grenoble etc. In the rest of the dissertation, this type of dimensions is referred simply as non-ordered dimensions.

- Dimensions for which the members of the domain set of the same level are naturally ordered. For example, 2010 $<$ 2011 at level Year in dimension Time and $all_{Time} \Uparrow$ 2010 etc. We refer this type of dimensions as ordered dimensions in the rest of the dissertation.

Thus, we do not impose any artificial order among the members. This strategy facilitates our objective of dynamic maintenance of a dimension space. Consequently, we can add the new members on the fly as and when they appear. In other words, when a fact using some new coordinate arrives, e.g. first sale in the city of Lyon, this triggers the addition of Lyon in the domain of level city of dimension Location. This addition signifies the addition of a new member in the dimension space of Location which results in the dynamic growth of the dimension space. We will see in the

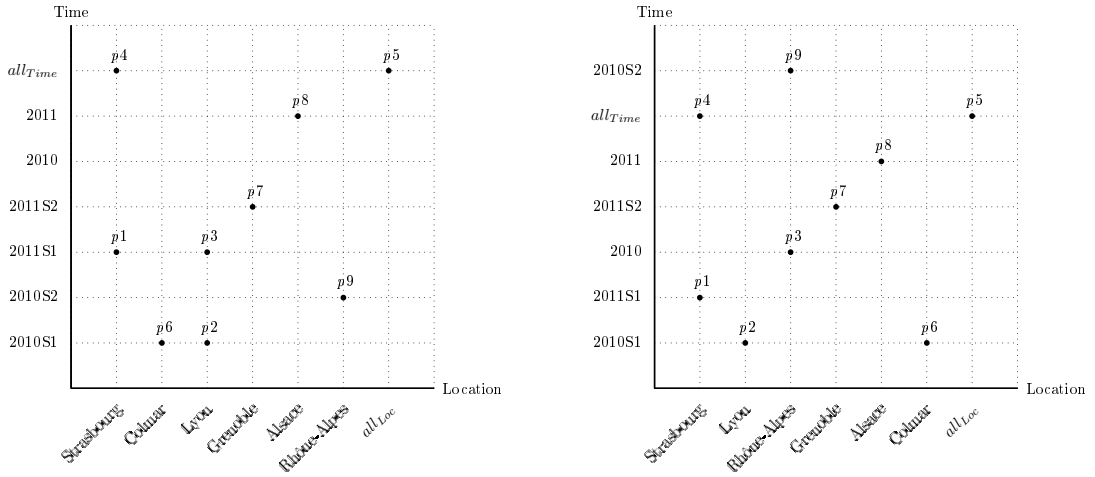


Figure 3.2: Two possible representations of a hierarchical hybrid multidimensional data space.

following that even without imposing an artificial order, we take the natural order (if any) into account for the optimization of data partitions. Otherwise stated, order is an asset and not a limitation; it is ignored while distributing the points in the space but considered while partitioning it.

3.2 Hierarchical Multidimensional Data Space

We have just defined the classical concepts of dimensions, hierarchy and their members and now we define our concept of hierarchical hybrid multidimensional data space. The dimensions serve as the axes of this data space and their members are the coordinates of these axes.

Definition 3.1 *(Hierarchical Hybrid Multidimensional Data Space): A hierarchical hybrid multidimensional data space S is defined as the cartesian product of dimension spaces of all the dimensions that form the basis of the multidimensional model. For a multidimensional model involving n dimensions, $S = S_{D_1} \times S_{D_2} \times ... \times S_{D_n}$ with S_{D_i}, $\forall i\ (1 \leq i \leq n\ )$, being the dimension space of D_i.*

The dimension space is called Hierarchical Hybrid Multidimensional Data Space (HHMDS) because it is made of both ordered and non-ordered hierarchical dimensions. A point p in S is represented by an ordered n-tuple $(x_1, x_2, ..., x_n)$ where $\forall i\ (1 \leq i \leq n\ )$, $x_i \in S_{D_i}$is the i^{th} dimensional coordinate of p.

Figure 3.2a shows a possible representation of a hierarchical multidimensional data space with members of dimensions described in figure 3.1. This, however, is only a possible representation as shown in figure 3.2b depicting the same data space, with dimension members plotted in a different order on the axes. We can see in these two representations that, not only the order on the axes is insignificant, but all the members of a dimension are also plotted on the same axis, regardless of their levels. We also note that the axes are neither continuous nor directional, therefore we do not plot any arrow sign at the end of the axes.

On arrival of a new fact in the decision system, a new point is created in the data space. If any of the dimension member used by the point (i.e. any of its coordinate) and/or its parent is missing on the axes, then it is added: the axes "grow" as and when needed, without any ordering consideration. Therefore, at any point, a dimension space (so as the instance of hierarchical dimension) has the same elements as the corresponding axis of the multidimensional data space. It is important to note that growth in dimension space is only due to the addition of new member to the instance of dimension hierarchy $\mathcal{I}$ which is built dynamically. However the dimension hierarchy $\mathcal{H}$ itself is known beforehand in our model.

One of the objectives of this model is to provide a data grouping strategy, therefore we now need a structure to group the data in the space. Our data grouping structure is inspired by the R-Tree's MBR. However,

- the axes in case R-Tree are ordered while in our case we do not have enough time to order the axes.
- the data space of R-Tree is composed of flat dimensions while in our case, we deal with hierarchical dimensions.

This implies that the data partitions constructed in R-Tree are always at the same level while in our case we need data partitions at different hierarchical levels helping us to store the data at different levels of aggregation. For this purpose, we first define the concept of hyper-planes in the following.

Definition 3.2 *(Multidimensional Hyper-plane): A multidimensional hyper-plane (or simply a hyper-plane) P with n dimensions is defined as $P = domain(l_1^{h_1}) \times domain(l_2^{h_2}) \times ... \times domain(l_n^{h_n})$ where $\forall i$ $(1 \leq i \leq n)$, $1 \leq h_i \leq k_i$ and k_i is the number of levels in dimension D_i.*

Every hyper-plane can be identified by an n-tuple $< l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$. A point $p(x_1, x_2, ..., x_n)$ lies in the hyper-plane $< level(x_1), level(x_2), ..., level(x_n) >$. We can also note that:

- There are $k_1 * k_2 * ... * k_n$ different hyper-planes in a hierarchical multidimensional data space with n dimensions.
- All the hyper-planes in S are organized in a hierarchy with partial order $\geq_P$. Thus given $P < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ and $\mathcal{P}' < l_1^{h_1'}, l_2^{h_2'}, ..., l_n^{h_n'} >$ two multidimensional hyper-planes, we note $P \geq_P P'$ iff $\forall i \, (1 \leq i \leq n)$, $l_i^{h_i} \uparrow l_i^{h_i'}$. We also note that $\geq_P$ is a transitive partial order relation.

The hyper-planes can be seen as mutually exclusive logical subspaces of an HHMDS which represent the spaces determining the aggregation of data at different levels of granularity.

Example 3.1 In figure 3, we can see the following 9 hyper-planes:

$P_1 < City, Semester >$, $P_2 < Region, Semester >$, $P_3 < City, Year >$, $P_4 < Region, Year >$, $P_5 < ALL_{Location}, Semester >$, $P_6 < City, ALL_{Time} >$, $P_7 < ALL_{Location}, Year >$, $P_8 < Region, ALL_{Time} >$, $P_9 < ALL_{Location}, ALL_{Time} >$. Among these hyper-planes, we can see, among others, the following relations: $P_9 \geq_P P_7$, $P_5 \geq_P P_2$ etc.

Definition 3.3 *(Minimum Bounding Space): Let* $\Delta = \{(x_{1,1}, x_{1,2}, ..., x_{1,n}), (x_{2,1}, x_{2,2}, ..., x_{2,n}), ..., (x_{m,1}, x_{m,2}, ..., x_{m,n})\}$ *be a set of* m *points lying in a same* n*-dimensional hyper-plane i.e.* $x_{j,i} \in domain(l_i^{k_i})$ *for* $1 \leq i \leq n$, $1 \leq j \leq m$ *and* k_i *is a level in the hierarchy of dimension* D_i*. A minimum bounding space (MBS) constructed over* Δ*, denoted by* M_Δ*, is defined as:*

$$M_\Delta = \bigcup_{j=1}^{m} x_{j,1} \times \bigcup_{j=1}^{m} x_{j,2} \times ... \times \bigcup_{j=1}^{m} x_{j,n}$$

where each set $\bigcup_{j=1}^{m} x_{j,i}$ *is called the* $i^{th} (1 \leq i \leq n)$ *dimension edge* E_i *of* M_Δ*.*

An MBS allows the grouping of data in the space and represents a section of a cuboid. It is also clear from the definition that an MBS can be constructed over one or more points but lies only in one hyper-plane: MBS can not span multiple hyper-planes. The MBS $M_\Delta = E_1 \times E_2 \times ... \times E_n$, lies in the hyper-plane $< level(e_1), level(e_2), ..., level(e_n) >$ where $e_i \in E_i$. For M_Δ, we note $|M_\Delta| = \prod_{i=1}^{n} |E_i|$ as the maximum number of points the MBS can hold and $||M_\Delta|| = |\Delta|$ as the number of currently existing points in the MBS.

Example 3.2 In figure 3.3, we can observe the following MBS:
$M1 = M_{\{p1, p2, p3, p5\}} = M_{\{(Strasbourg, 2010S1), (Lyon, 2011S1), (Lyon, 2011S2), (Grenoble, 2010S1)\}}$ = {Strasbourg, Lyon, Grenoble}×{2010S1, 2011S1, 2011S2} is an MBS constructed over four points in hyper-plane $< City, Semester >$ with $|M1| = 9$ and $||M1|| = 4$,
$M2 = M_{\{p6, p10, p11\}} = M_{\{(Alsace, 2011S2), (Rhône-Alpes, 2010S2), (Alsace, 2010S2)\}}$ = {Alsace, Rhône-Alpes}×{2010S2, 2011S2} is an MBS constructed over three points in hyper-plane $< Region, Semester >$ with $|M2| = 4$ and $||M2|| = 3$,
$M3 = M_{\{p8, p9\}} = M_{\{(Colmar, all_{Time}), (Grenoble, all_{Time})\}}$ = {Grenoble, Colmar}×$\{all_{Time}\}$ is an MBS constructed over two points in hyper-plane $< City, ALL_{Time} >$ with $|M3| = 2$ and $||M3|| = 2$,
$M4 = M_{\{p4, p7\}} = M_{\{(all_{Location}, 2010), (all_{Location}, 2011)\}} = \{all_{Location}\} \times \{2010, 2011\}$ is an MBS constructed over two points that lie in hyper-plane $< ALL_{Location}, Semester >$ with $|M4| = 2$ and $||M4|| = 2$,
$M5 = M_{\{p2\}} = M_{\{(Lyon, 2011S1)\}}$ = {$Lyon$}×{$2011S1$} is an MBS constructed over a single point in hyper-plane $< City, Semester >$ with $|M5| = 1$ and $||M5|| = 1$,

Definition 3.4 *(Measure and Fact): A numerical value associated to a multidimensional point $p(x_1, x_2, ..., x_n)$ in HHMDS is called measure. We denote this value as $m(p)$. The point p with $m(p)$ is termed as a fact if p lies in the lowest level hyper-plane i.e. $< l_1^1, l_2^1, ..., l_n^1 >$.*

Definition 3.5 *(Aggregate): Let $\Delta = \{p_1, p_2, ..., p_k\}$ be a set of k multidimensional points and M_Δ be an MBS constructed over Δ. A numerical value associated to M_Δ, denoted by $agg(M_\Delta)$, is called aggregate. The value of $agg(M)$ is obtained by applying some aggregate function f (e.g. SUM, MAX, MIN etc.) on the measure values associated the points enclosed in $M_\triangle$ i.e. $agg(M_\Delta) = f(m(p_1), m(p_2), ..., m(p_k))$.*

The measure value in our model can represent both the value associated to a detailed data point (i.e. lying in the lowest level hyper-plane) or some aggregated data point (i.e. lying in some higher level hyper-plane).

Until now, we have defined a hierarchical multidimensional data space, points and MBS. Now, to be able to manipulate these points and MBS, we provide the definitions of some algebraic concepts in the following section.

4 Algebra for HHMDS

In this section, we provide the formal definitions of operators, relations and metrics for HHMDS and explain them with the help of examples. The definition of these

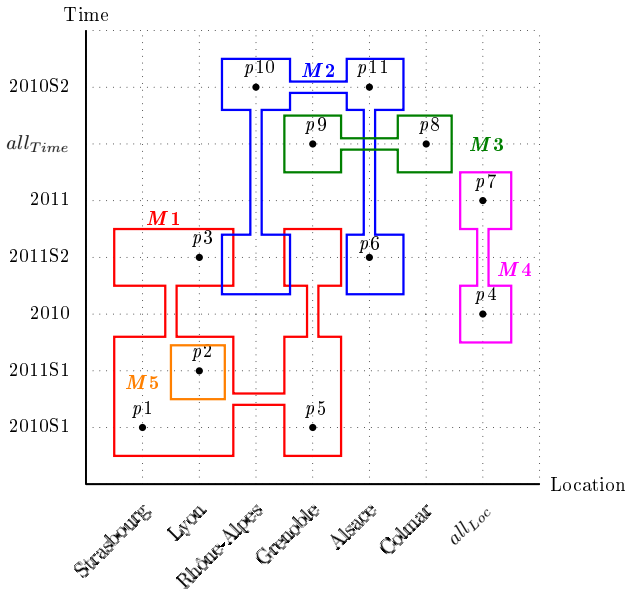


Figure 3.3: Minimum bounding spaces (MBS) in an HHMDS

concepts facilitates the comparison of the objects (points and MBS) lying in the data space as well as the algebraic operations on those objects.

4.1 Operators

The operators defined under this heading are used to provide some algebraic transformations or operations on the objects in HHMDS. We categorize these operators in two categories: operators for dimension members and operators for MBS.

4.1.1 Operators for Dimension Members

In the following we define two operators for the members of the domain sets of different dimensions. These operators allow navigating among the levels of dimension hierarchy.

Definition 3.6 *(Drill-Up): Let $l_i^u, l_i^v \in L_i$ and $l_i^u \uparrow l_i^v$. A drill-up operation $\sigma_{l_i^u}$ is a function:*

$$\begin{array}{rcccl} \sigma_{l_i^u} & : & domain(l_i^v) & \longrightarrow & domain(l_i^u) \\ & & x & \longmapsto & \sigma_{l_i^u}(x) = y \quad such\ that\ y \Uparrow x. \end{array}$$

Definition 3.7 *(Drill-Down): Let $l_i^u, l_i^v \in L_i$ and $l_i^u \uparrow l_i^v$. A drill-down operation $\varrho_{l_i^v}$ is a function:*

$$\begin{aligned} \varrho_{l_i^v} : domain(l_i^u) &\rightarrow \mathrm{P}(domain(l_i^v)) \\ x &\mapsto \varrho_{l_i^v}(x) = \{y \in domain(l_i^v) \mid \forall y,\ x \Uparrow y\} \end{aligned}$$

The drill-up and drill-down operators are traditionally used for the aggregation in classical OLAP model. But, in our model these operators are defined on the members of the instance of dimension hierarchy and allow us navigating from a higher hierarchy level to a lower one or vice versa.

Since the instance of dimension hierarchy evolves over time, the result of a drill-down operation on same member could vary over two different time instances. We also remark that $\varrho_{l_i^v}(\sigma_{l_i^u}(x)) \neq x$.

Example 3.3 For instance, in the data space shown in figure 3:
$\sigma_{Region}(Colmar) = Alsace$, $\sigma_{ALL_{Location}}(Lyon) = all_{Location}$ and $\sigma_{ALL_{Time}}(2011) = all_{Time}$ etc.
$\varrho_{City}(Rhône\text{-}Alpes)$=*{Lyon, Grenoble}* and $\varrho_{Year}(all_{Time})$ = *{2010, 2011}* etc.
$\varrho_{City}(\sigma_{ALL_{Time}}(Lyon)) = \{Strasbourg, Colmar, Lyon, Grenoble\} \neq Lyon$ etc.

4.1.2 Operators for MBS

In the following we define operators for MBS in HHMDS. Among these, translate-up and translate-down are inter-hyper-plane operators while the union is an intra-hyper-plane operator. The inter-hyper-plane operators allow moving an MBS from one hyper-plane to another hyper-plane. These, consequently serve as summarization or detailing operators for the MBS. These operators are also used to make two MBS comparable by describing them at same level of detail. The intra-hyper-plane operator, on the other hand, is used for the merger of two MBS.

Definition 3.8 *(Translate-Up): Let* $P < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ *and* $P' < l_1^{h_1'}, l_2^{h_2'}, ..., l_n^{h_n'} >$ *be two hyper-planes in S such that* $P' \geq_P P$ *and* $M = E_1 \times E_2 \times ... \times E_n$ *be an MBS in P. A translate-up operation* $\lceil_{P'}$ *is defined as:*

$$\begin{aligned} \lceil_{P'} : P &\rightarrow P' \\ M &\mapsto M' = \bigcup_{e_1 \in E_1} \sigma_{l_1^{h_1'}}(e_1) \times \bigcup_{e_2 \in E_2} \sigma_{l_2^{h_2'}}(e_2) \times ... \times \bigcup_{e_n \in E_n} \sigma_{l_n^{h_n'}}(e_n) \end{aligned}$$

A translate-up operation translates an MBS from its current hyper-plane to a target higher level hyper-plane in S. Union of drill-up on the elements of an edge of

the MBS gives us the corresponding edge of MBS in target hyper-plane. The cross product of all the edges is then defined as resultant MBS. This allows us to describe the data enclosed in an MBS at a higher aggregation level or in other words, helps us to summarize the data.

Example 3.4 In figure 3.4, $N1 = \{Strasbourg, Lyon, Grenoble\} \times \{2010S1, 2011S1, 2011S2\}$ holding four points (i.e. $p1$, $p2$, $p3$, $p6$) lies in hyper-plane $< City, Semester >$. To translate this in hyper-plane $< Region, Year >$,we need to do a drill-up on each element of both the edges of $N1$. Since, $\sigma_{Region}(Strasbourg) = Alsace$, $\sigma_{Region}(Lyon) = Rhône-Alpes$, $\sigma_{Region}(Grenoble) = Rhône\text{-}Alpes$, $\sigma_{Year}(2010S1) = 2010$, $\sigma_{Year}(2011S1) = 2011$, and $\sigma_{Year}(2011S2) = 2011$, the sets $\{Rhône\text{-}Alpes, Alsace\}$ *and* $\{2010, 2011\}$ will make the edges of the resultant MBS. Therefore, $\lceil_{<Region, Year>}(N1) = \{Rhône - Alpes, Alsace\} \times \{2010, 2011\} = N2$

Similarly, the translate-up on $N3 = \{Colmar\} \times \{2010, 2011\}$ from its current hyper-plane $< City, Semester >$ to the target hyper-plane $< City, Year >$ produces $N5$ i.e. $\lceil_{<City, Year>}(N3) = \{Colmar\} \times \{2010, 2011\} = N5$ and the translate-up on $N2 = \{Rhône - Alpes, Alsace\} \times \{2010, 2011\}$ from $< Region, Year >$ to the target higher level hyper-plane $< ALL_{Location}, ALL_{Time} >$ produces $N4$ i.e. $\lceil_{<ALL_{Location}, ALL_{Time}>}(N2) = \{all_{Time}\} \times \{all_{Location}\} = N4$ etc.

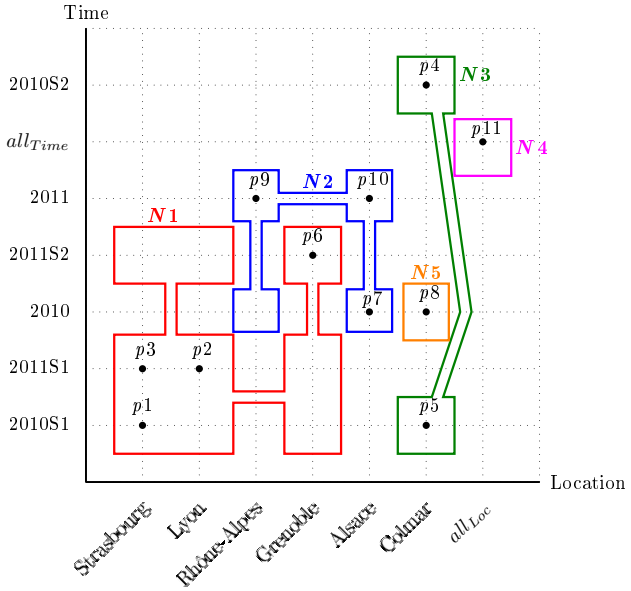


Figure 3.4: Translate-Up operation on MBS

Definition 3.9 *(Translate-Down): Let* $P < l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$ *and* $P' < l_1^{h'_1}, l_2^{h'_2}, ...$

$, l_n^{h_n'} >$ *be two hyper-planes in* S *such that* $P' \geq_P P$ *and* $M' = E_1' \times E_2' \times ... \times E_n'$ *be an MBS in* P'*. A translate-down operation* $\lfloor_P$ *is defined as:*

$$\begin{aligned} \lfloor_P : P' &\rightarrow P \\ M' &\mapsto M = \bigcup_{e_1 \in E_1'} \varrho_{l_1^{h_1}}(e_1) \times \bigcup_{e_2 \in E_2'} \varrho_{l_2^{h_2}}(e_2) \times ... \times \bigcup_{e_n \in E_n'} \varrho_{l_n^{h_n}}(e_n) \end{aligned}$$

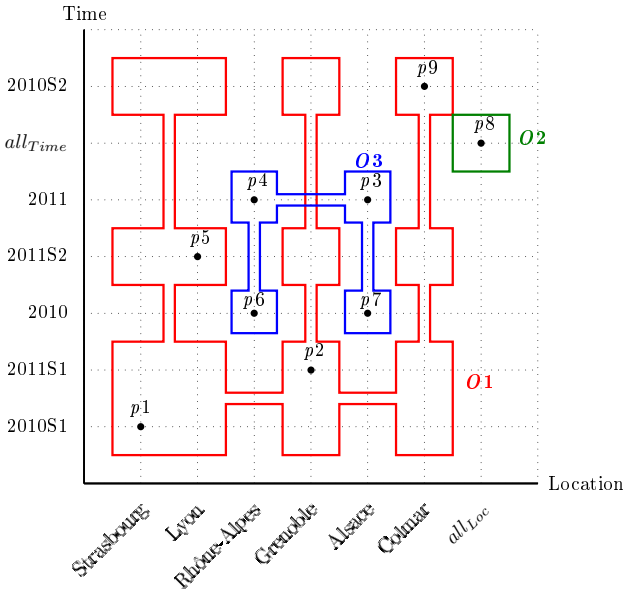


Figure 3.5: Translate-Down operation on MBS

A translate-down operation translates an MBS from its current hyper-plane to a target lower level hyper-plane in S. Union of drill-down on the elements of an edge of the MBS gives us the corresponding edge of MBS in target hyper-plane. The cross product of all the edges is then defined as resultant MBS. Opposite to translate-up, translate-down operator describes the data enclosed in an MBS at a finer level of detail.

We note that $\lfloor_P(\lceil_{P'}(M)) \neq M$ while $\lceil_{P'}(\lfloor_P(M)) = M$. We also note that the translate-up and translate-down operators may produce an MBS that encloses some points that do not actually exist in the HHMDS (e.g. $O1$ in figure 3.5).

Example 3.5 In figure 3.5, $O2 = \{all_{Location}\} \times \{all_{Time}\}$ constructed over an aggregated point $p9$ lies in the hyper-plane $< ALL_{Location}, ALL_{Time} >$. To translate-down this MBS to a target hyper-plane $< Region, Year >$, drill-down operation is required

on all elements of both the edges of $O2$. We get, $\varrho_{Region}(all_{Location})$ ={Rhône-Alpes, Alsace} and $\varrho_{Year}(all_{Time})$ = {2010, 2011}. Therefore, $\lfloor_{<Region, Year>}(O2)$ ={Rhône-Alpes, Alsace}×{2010, 2011} = $O3$

Similarly, $\lfloor_{<City, Semester>}(O3)$ ={Strasbourg, Lyon, Colmar, Grenoble}×{2010S1, 2011S1, 2011S2, 2010S2} = $O1$ is a translate-down on an MBS from $< Region, Year >$ to a target lower level hyper-plane $< City, Semester >$ etc.

Definition 3.10 *(Union): Let $M = E_1 \times E_2 \times ... \times E_n$ and $N = F_1 \times F_2 \times ... \times F_n$ be two MBS in same hyper-plane. The union of M and N, denoted by $M \cup N$ is defined as:*

$$M \cup N = (E_1 \cup F_1) \times (E_2 \cup F_2) \times ... \times (E_n \cup F_n)$$

In other words, the union of two MBS is the cross product of the unions of the corresponding edges. It is important to note that union can only be calculated for the MBS lying in the same hyper-plane.

Example 3.6 In figure 3.6, we have:
$R1 \cup R2$ ={Strasbourg, Lyon, Grenoble, Colmar}×{2010S1, 2011S1, 2011S2},
$R5 \cup R6$ ={Lyon, Colmar}×{2010S1, 2010S2} etc.

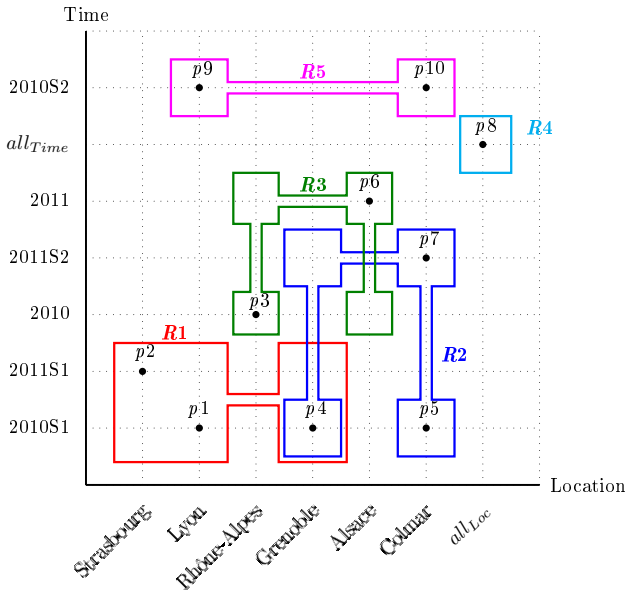


Figure 3.6: MBS to illustrate the metrics and relations defined for HHMDS

4.2 Relations

The partial order in dimension hierarchy of the space constituting dimension allows us to introduce the following relation among different points.

Definition 3.11 *(Covers): For two n-dimensional points* $p(x_1, x_2, ..., x_n)$ *and* $p'(x'_1, x'_2, ..., x'_n)$ *we say* p' *covers* p*, and note* $p' \circledcirc p$*, if* $\forall i\,(1 \leq i \leq n)$, $x'_i \Uparrow x_i$.

A point p' covers another point p if all of its coordinates can be obtained by a drill-up operation applied on those of p. Remember that drill-up is a reflexive operator which means that every point covers at least one point i.e. $p \circledcirc p$.

Example 3.7 In figure 3.6, these relations stand:
$p1$, $p2$, $p4$, $p5$, $p7$, $p9$ *and* $p10$ lie in the lowest level hyper-plane, therefore, they are not comparable and do not cover any point except themselves.
$p_3 \circledcirc p_1$ because $2010 \Uparrow 2010S1$ and *Rhône-Alpes*$\Uparrow$ *Lyon*,
$\neg(p_3 \circledcirc p_{10})$ because $2010 \Uparrow 2010S2$ but $\neg$(*Rhône-Alpes*$\Uparrow$ *Colmar*),
$p_6 \circledcirc p_2$ because $2011 \Uparrow 2011S1$ and $Alsace \Uparrow Strasbourg$,
$\forall p \in S$, $p_8 \circledcirc p$

We define the following two relations which let us find the relative positions of two MBS. These relations are also used in tree construction and querying algorithms described in the next chapter.

Definition 3.12 *(Contains): Let* $M = E_1 \times E_2 \times ... \times E_n$ *and* $N = F_1 \times F_2 \times ... \times F_n$ *be two MBS in an HHMDS. We define relation contains as:*

$$(M \, contains \, N) \, if : \forall p \in N, \exists q \in M \mid q \circledcirc p$$

An MBS M contains another MBS N, if all the points of M are covered by the points of N. Otherwise stated, if M and N are not in the same hyper-plane, M is translated-down (if possible) to the hyper-plane of N and checked if all the edges of N are included in those of the translated-down MBS. Contains is a reflexive, transitive and asymmetric relation over MBS.

Example 3.8 In the following we try to determine if ($R3$ *contains* $R1$) in figure 3.6. $R3 = \{$*Rhône-Alpes*, *Alsace*$\} \times \{2010, 2011\}$ encloses four possible points i.e. (*Rhône-Alpes*, 2010), (*Rhône-Alpes*, 2011), (*Alsace*, 2010), (*Alsace*, 2011) and $R1 = \{Strasbourg$, *Lyon*, *Grenoble*$\} \times \{2010S1, 2011S1\}$ encloses six points that are ($Strasbourg$, $2010S1$), ($Strasbourg$, $2011S1$), (*Lyon*, $2010S1$), (*Lyon*, $2011S1$), (*Grenoble*, $2010S1$) and (*Grenoble*, $2011S1$). Among these, (*Rhône-Alpes*, 2010) covers (*Lyon*, $2010S1$) and (*Grenoble*, $2010S1$), (*Rhône-Alpes*, 2011) covers (*Lyon*, $2011S1$) and (*Grenoble*, $2011S1$),

(*Alsace*, 2010) covers (*Strasbourg*, 2010$S1$) and (*Alsace*, 2011) covers (*Strasbourg*, 2011$S1$). This implies that all the possible points enclosed by $R1$ are covered by those of $R3$. Therefore, ($R3$ *contains* $R1$).
While determining if ($R5$ contains $R6$), we find that (*Colmar*, 2010$S1$) is not covered by any point of $R5$ i.e. (*Colmar*, 2010$S2$) and (*Lyon*, 2010$S2$). Therefore, $\neg(R5\,contains\,R1)$.
Similarly, ($R3$ *contains* $R2$), ($R4$ *contains* $R3$) etc.

Definition 3.13 *(Overlaps): Let $M = E_1 \times E_2 \times ... \times E_n$ and $N = F_1 \times F_2 \times ... \times F_n$ be two MBS in a same hyper-plane P.*

$$(M\,overlaps\,N)\,if: \exists p \in N, \exists q \in M \mid q = p$$

An MBS M overlaps N if there is at least one point enclosed by both M and N.

Example 3.9 In the following we determine if ($R5$ *overlaps* $R6$) in figure 3.6.
$R5 = \{Lyon, Colmar\} \times \{2010S2\}$ encloses two possible points i.e. (*Lyon*, 2010$S2$) and (*Colmar*, 2010$S2$) and $R6 = \{Colmar\} \times \{2010S1, 2010S2\}$ encloses (*Colmar*, *2010S1*) and (*Colmar*, 2010$S2$). Since (*Colmar*, 2010$S2$) is common in both $\mathcal{M}$ and $\mathcal{N}$, ($R5$ *overlaps* $R6$).
Since, $R1 = \{Strasbourg, Lyon, Grenoble\} \times \{2010S1, 2011S1\}$ and $R6 = \{Colmar\} \times \{2010S1, 2010S2\}$ do not have any common point, $\neg(R1\,overlaps\,R6)$
Similarly, ($R1$ *overlaps* $R2$), ($R2$ *overlaps* $R6$) etc.

An MBS overlaps another MBS, if there is at least one common point among the two MBS. The overlap among the MBS lying in different hyper-planes is meaningless and can not be calculated. It is a reflexive, transitive and symmetric relation over MBS.

4.3 Metrics

Since now, we have seen that the edges of an MBS are represented by a set. However, for an ordered dimension (such as time), the edge can obviously be defined by an interval. Let D_t be an ordered dimension in our schema, for example the time dimension, with E_t being the corresponding edge of an MBS M. The time interval covered by E_t is noted as $Interval(E_t) = [\min_{m_i \epsilon E_t}(m_i), \max_{m_i \epsilon E_t}(m_i)]$.

The usage of interval for ordered dimensions to describe the length of an edge is important because it simplifies the calculation of a range of values which, in turn, speeds up the calculations of metrics defined in this section.

The metrics are used to numerically characterize the MBS in the data space. Among the metrics defined under this section, volume and density are calculated for a single MBS while the extension and overlap area are calculated among two MBS. By exploiting these metrics, we optimize the construction of MBS.

Definition 3.14 *(Volume): Let $M = E_1 \times E_2 \times ... \times E_n$ be an MBS in some hyper-plane P in S. The volume(M) is defined as:*

$$volume(M) = |\lfloor_{<l_1^1, l_2^1, ..., l_n^1>}(M)|$$

In order to make the comparison meaningful among different MBS, the volume of an MBS is always calculated with a translate-down at the lowest level hyper-plane i.e. $< l_1^1, l_2^1, ..., l_n^1 >$. The volume of an MBS determines the maximum number of points an MBS can enclose if translated down to the lowest level hyper-plane. Otherwise stated, it is the maximum number of facts, the MBS could possibly group under itself.

Example 3.10 The calculation of $volume(R3)$ in figure 3.6 is as follows:

$$R3 = \{Rhône - Alpes,\ Alsace\} \times \{2010,\ 2011\}$$
$$\lfloor_{<City,\ Semester>}(R3) = \{Strasbourg,\ Lyon,\ Grenoble,\ Colmar\} \times \{2010S1,\ 2010S2,\ 2011S1,\ 2011S2\}$$

Therefore:

$$volume(R3) = 4 * 4 = 16$$

Similarly, $volume(R1) = 3*3 = 9$, $volume(R2) = 2*4 = 8$, $volume(R4) = 4*4 = 3$, $volume(R5) = 2 * 1 = 2$, $volume(R6) = 1 * 2 = 2$ etc.

Definition 3.15 *(Density): Let $M = E_1 \times E_2 \times ... \times E_n$ be an MBS in some hyper-plane P in S. The density of M is defined as:*

$$density(M) = \frac{||\lfloor_{<l_1^1, l_2^1, ..., l_n^1>}(M)||}{volume(M)}$$

The density of an MBS determines the ratio of facts grouped under an MBS to its volume and allows to distinguish between dense and sparse data groups.

Example 3.11 In figure 3.6, we can calculate the $density(R3)$ as follows:
As shown in above example, $\lfloor_{<City,\,Semester>}(R3) = \{Strasbourg, Lyon, Grenoble, Colmar\} \times \{2010S1, 2010S2, 2011S1, 2011S2\}$. Now we can observe that there are seven points ($p1$, $p2$, $p4$, $p5$, $p7$, $p9$, $p10$) in $\lfloor_{<City,\,Semester>}(R3)$ i.e. $||\lfloor_{<l_1^1, l_2^1, ..., l_n^1>}(M)|| = 7$ and as calculated before, $volume(R3) = 16$. Therefore,

$$density(R3) = \frac{7}{16} = 0.437$$

Similarly, $density(R1) = \frac{3}{9} = 0.33$, $density(R2) = \frac{2}{8} = 0.25$, $density(R4) = \frac{7}{16} = 0.437$ $density(R5) = \frac{2}{2} = 1$, $density(R6) = \frac{2}{2} = 1$ etc.

Definition 3.16 *(Extension): Let $M = E_1 \times E_2 \times ... \times E_n$ and $N = F_1 \times F_2 \times ... \times F_n$ be two MBS in a same hyper-plane P and $Interval(E_t) = [t_1, t_2]$ and $Interval(F_t) = [t_3, t_4]$ be the intervals covered by the time dimension edges of M and N respectively. The extension of M to accommodate N, denoted by $extension(M|N)$ is calculated as:*

$$extension(M|N) = \sum_{i=1, i\neq t}^{n} |F_i - E_i| + \delta(E_t, F_t)$$

where

$$\delta(E_t, F_t) = \begin{cases} t_4 - t_2 & t_1 \leq t_3 \; and \; t_2 \leq t_4 \\ t_1 - t_3 & t_1 \geq t_3 \; and \; t_2 \geq t_4 \\ t_1 - t_3 + t_4 - t_2 & t_1 \geq t_3 \; and \; t_2 \leq t_4 \\ 0 & t_1 \leq t_3 \; and \; t_2 \geq t_4 \end{cases}$$

Table 3.1 illustrates the calculation of $\delta(E_t, F_t)$ using Allen's relations.

Extension required to accommodate another MBS can only be calculated, if the two MBS lie in the same hyper-plane. This metric is used to calculate the distance among two MBS. It is a reflexive, transitive and asymmetric metric.

Example 3.12 To calculate $extension(R1|R2)$ in figure 3.6, we can say:
$R1 = E_1 \times E_t \; and \; R2 = F_1 \times F_t$ with $E_1 = \{Strasbourg, Lyon, Grenoble\}$, $E_t = \{2010S1, 2011S1\}$, $F_1 = \{Grenoble, Colmar\}$ *and* $F_t = \{2010S1, 2011S2\}$. Now:

$$\begin{aligned} |F_1 - E_1| &= |\{\text{Colmar}\}| = 1 \\ \delta(E_t, F_t) &= 2010S1 - 2010S1 + 2011S2 - 2011S1 = \; 1 \\ extension(R1|R2) &= 1 + 1 \\ &= 2 \end{aligned}$$

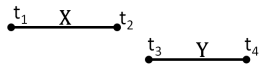
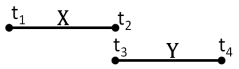
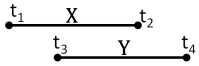
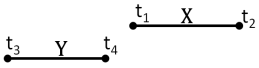
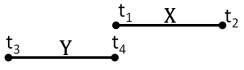
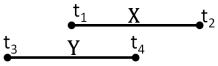
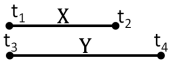
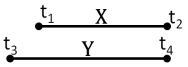
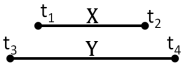
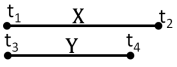
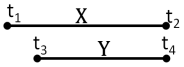
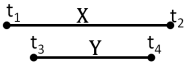
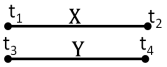
Relation	Illustration	Interpretation	$\delta(E_t, F_t)$	$\beta(E_t, F_t)$
$X < Y$		X takes place before Y	$t_4 - t_2$	-1
$X\, m\, Y$		X meets Y		
$X\, o\, Y$		X overlaps with Y		$t_2 - t_3$
$Y < X$		Y takes place before X	$t_1 - t_3$	-1
$X\, mi\, Y$		Y meets X		
$X\, oi\, Y$		Y overlaps with X		$t_4 - t_1$
$X\, s\, Y$		X starts Y		
$X\, f\, Y$		X finishes Y	$t_1 - t_3$ $+t_4 - t_2$	$t_2 - t_1$
$X\, d\, Y$		X during Y		
$X\, si\, Y$		Y starts X		
$X\, fi\, Y$		Y finishes X	0	$t_4 - t_3$
$X\, di\, Y$		Y during X		
$X = Y$		X is equal to Y		

Table 3.1: Illustration of the calculation of $\delta(E_t, F_t)$ and $\beta(E_t, F_t)$ using Allen's interval algebra. The intervals used for the illustration are $X = Interval(E_t) = [t_1, t_2]$ and $Y = Interval(F_t) = [t_3, t_4]$

Similarly, $extension(R2|R6) = 0$, $extension(R2|R5) = 1$, $extension(R1|R5) = 1$ etc.

Definition 3.17 *(Overlap Area): Let $M = E_1 \times E_2 \times ... \times E_n$ and $N = F_1 \times F_2 \times ... \times F_n$ be two MBS in a same hyper-plane P and $Interval(E_t) = [t_1, t_2]$ and $Interval(F_t) = [t_3, t_4]$ be the intervals covered by the time dimension edges of M and N respectively. The shared area, called overlap area, between M and N, noted as $ovlapArea(M, N)$, is calculated as:*

$$ovlapArea(M,\ N) = \prod_{i=1, i \neq t}^{n} |F_i \cap E_i| * (\beta(E_t,\ F_t) + 1)$$

where

$$\beta(E_t,\ F_t) = \begin{cases} -1 & t_2 < t_3 \ or \ t_4 < t_1 \\ t_2 - t_3 & t_1 \leq t_3 \ and \ t_2 \leq t_4 \\ t_4 - t_1 & t_1 \geq t_3 \ and \ t_2 \geq t_4 \\ t_2 - t_1 & t_1 \geq t_3 \ and \ t_2 \leq t_4 \\ t_4 - t_3 & t_1 \leq t_3 \ and \ t_2 \geq t_4 \end{cases}$$

Table 3.1 illustrates the calculation of $\beta(E_t,\ F_t)$ using Allen's relations.

Overlap area can only be calculated, if the two MBS lie in the same hyper-plane. It is a reflexive, transitive and symmetric metric.

Example 3.13 To calculate $ovlapArea(R1,\ R2)$ in figure 3.6, we can say:
$R1 = E_1 \times E_t \ and \ R2 = F_1 \times F_t$ with $E_1 = \{Strasbourg, Lyon, Grenoble\}$, $E_t = \{2010S1, 2011S1\}$, $F_1 = \{Grenoble, Colmar\}$ *and* $F_t = \{2010S1, 2011S2\}$. Now:

$$\begin{aligned} |F_1 \cap E_1| &= |\{\text{Grenoble}\}| = 1 \\ \beta(E_t,\ F_t) &= 2011S1 - 2010S1 = \ 2 \\ ovlapArea(R1,\ R2) &= 1 * (2 + 1) \\ &= 3 \end{aligned}$$

Similarly, $ovlapArea(R2, R6) = 2$, $ovlapArea(R6, R5) = 1$, $ovlapArea(R2, R5) = 0$ etc.

We have seen that the calculation of the presented metrics related to MBS is based on the cardinality of the edges in case of non-ordered dimensions while it depends on the length of the intervals in case of ordered dimensions. The consequence of

this distinction is important: suppose, as we considered above, that the only ordered dimension is time. These metrics favor the grouping of temporally closed values together in the same MBS. This grouping, in turn, helps to improve the response time for range queries, which quite often involve the ranges over ordered dimensions. We can easily extend this reasoning for all other ordered dimensions, and can define a formula to calculate the distance among different members, that will allow the grouping of similar or closed points in MBS.

5 Conclusion

In this chapter, we presented a data model that provides an abstraction for a Hierarchical Hybrid Multidimensional Data Space. The model is used to describe OLAP related objects and operations in this spatial-like data space. The proposed data grouping structure is used to partition the data in the space and the metrics are used to optimize these partitions. Our data grouping structure is based on cartesian product of sets of members which is similar to the ideas proposed in other works such as [Ester 2000], [Chen 2009b] and [Aligon 2011].

However, the originality of our data model lies in the non-ordered multilevel axes constituting the dimensions space: indeed each dimension is represented by an axis constituted of the non-ordered set of the members of all levels of the dimension. This gives us the power to represent both detailed and aggregated data in the same data space and then regroup it to create data partitions representing the views at different levels of granularity. It also allows us to dynamically add new members at any hierarchical levels and provides a solution for handling evolving environment and progressive appearance of aggregates in the cube. The proposed algebraic operators then lets us manipulate the multidimensional points and the data partitions by aggregating or detailing them in the data space. Another original aspect of our work is that, in our data model, the dimensions are not required to have any kind of order. However, while calculating the metrics to optimize the MBS, natural order among the members, if any, of a dimension is exploited to regroup close points in the same MBS.

Now, after defining the elements of data model and the algebra, we need a data structure to store these partitions. Algorithms are needed to insert the new fact in this data structure and query it for efficient data retrieval. The details about these algorithms and the data structure are discussed in the next chapter.

Chapter 4

The DyTree

This chapter introduces a data structure called the DyTree. The DyTree operates in Hierarchical Hybrid Multidimensional Data Space and is suitable to store and index the data for dynamic data warehouses. The nodes of a DyTree store an MBS with associated aggregated measure values. The MBS represent the materialized views and are selected thanks to the metrics-based grouping strategy described in the last chapter. The tree as a whole is a representation of partially materialized and indexed data cube. In this chapter, we introduce the structure of the tree as well as the algorithms to construct and query it. The details of algorithms are explained through a running example which facilitates understanding their working.

Chapter Outline

1 Introduction

As discussed in previous chapter, the HHMDS can evolve dynamically without needing any consideration for reordering of the data space. For such a data space, we also proposed data grouping structure and related algebra which allows us to optimize the data partitions and to perform aggregations at different levels of granularity over the data lying in HHMDS.

Now, as the DyTree operates in such data space, it automatically complies to the requirements of dynamic data warehousing. We use MBS proposed in last chapter to group the data points lying in the data space among different partitions. We propose the algorithm to construct and optimize these data partitions using the previously defined metrics. The data partitions are indexed in the proposed tree structure, i.e. the DyTree. As a result, the DyTree simultaneously stores and indexes the detailed and/or aggregated data in HHMDS. The nodes of DyTree are self-constructed chunks of cuboids represented by MBS that are constructed at run-time. The tree as a whole represents a partially materialized data cube that is built dynamically. In the following, we explain the structure and the algorithms to construct and query the DyTree. The working of the provided algorithms is explained through our running example which simplifies the understanding.

2 Structure of the DyTree

In this section we describe the structure of a DyTree. We first discuss the elements that make up a DyTree and then describe the input parameters needed to construct the tree.

2.1 Elements of DyTree

DyTree is a dynamic structure that stores multidimensional points and MBS in HHMDS with associated measure(s) or aggregate values. The tree nodes, therefore, hold a subset of different possible views with pre-calculated aggregate values (materialized views). The DyTree has a structure similar to that of the B-tree and has a balanced tree structure. It indexes and stores the MBS in its internal nodes and facts in the leaves. The insertion order of the facts in the DyTree is not important, which helps dynamic insertions of the facts.

Definition 4.18 *A DyTree node is defined as a tuple node* $< M$, $entrySet$, a_1, a_2, ..., a_k $>$ *where:*

- M is an MBS,
- *entrySet* is a set of pointers to the child entries and
- $a_i\,(1 \leq i \leq k)$ are aggregate values.

We use $size(node.entrySet)$ to note the number of pointers held by the *entrySet* of the *node* at some particular time instance. The nodes are either internal or leaf nodes and can be categorized in following three types:

Directory Node: A node is called a directory node if the maximum value for $size($ *node.entrySet*$)$ is fixed and constant. In other words, a directory node is a fixed and limited capacity node. A directory node represents a fixed sized materialized section of cuboid and helps in improving the efficiency of aggregate queries in DyTree.
For example, $node_3 < M_4, \{f_3, f_6\}, 10 >$ is a directory node of the DyTree shown in figure 4.1a.

Super Node: A node is called a super node if the maximum value for $size($*node.entrySet*$)$ is not defined or unlimited. In other words, a super node is a variable and unlimited capacity node. Like a directory node, a super node also represents a materialized section of cuboid with only difference that they are larger capacity nodes with virtually unlimited size. The objective of having a super node is to avoid having highly overlapped nodes.
For example, $node_4 < M_5, \{f_4, f_5, f_2, f_1\}, 18 >$ is a super node of the DyTree shown in figure 4.1a.

Data Node: A node is called data node, if the maximum $node.entrySet = \phi$ and $M = E_1 \times E_2 \times ... \times E_n$ is an MBS such that M lies in the lowest level hyper-plane i.e. $< l_1^1, l_2^1, ..., l_n^1 >$ and $|E_i| = 1, (1 \leq i \leq n)$. In other words, a data node holds an MBS constructed over a single point in the lowest level hyper-plane of HHMDS with an associated measure value. A data node holds a fact with associated measure value.
An MBS constructed over a single multidimensional point p lying in the lowest level hyper-plane is called with the name of the point , in our examples. For example, $f4 < p4, \phi, 8 >$ is a data node of the DyTree shown in figure 4.1a in which $p4$ is an MBS constructed over $p4$. As the data nodes hold only facts, we use the terms facts and data nodes interchangeably in this chapter.

The root of a DyTree is always a directory node with MBS at level ALL_i for each dimension D_i in space i.e. the root always holds the MBS $all_1 \times all_2 \times ... \times all_n$. The root of the DyTree summarizes the apex of the hypercube. In our implementation, the data nodes in DyTree reside always on disk while the directory and super nodes are kept in main memory.

2.2 Input Parameters for the DyTree

A DyTree is built using two input parameters, i.e. directory node capacity ($DNCAP$) and overlap limit ($OVLAP$). The directory node capacity determines the number of children a directory node can have (i.e. $DNCAP$ = maximum $size(entrySet)$) while the overlap limit determines the maximum amount of shared region that the MBS of two nodes can have in common.

The figure 4.1a shows some parts of a DyTree, built with directory $DNCAP = 3$ and $OVLAP = 0$.

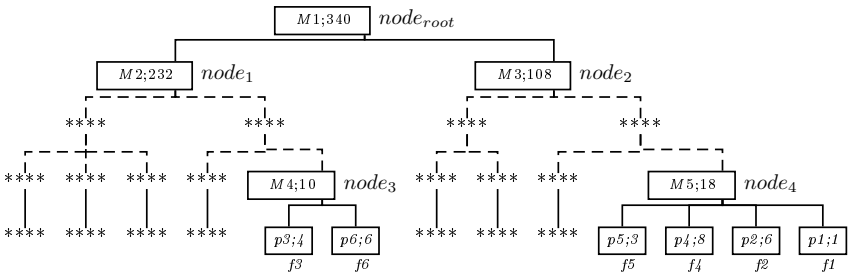
3 Constructing a DyTree

On the occurrence of a new transaction in the system (e.g. sale of an item), the algorithm to insert a new fact in the DyTree is triggered. This results in insertion of a new fact and update of the existing MBS to dynamically construct and maintain the DyTree. In this section, we present and discuss the principle algorithms needed for this purpose.

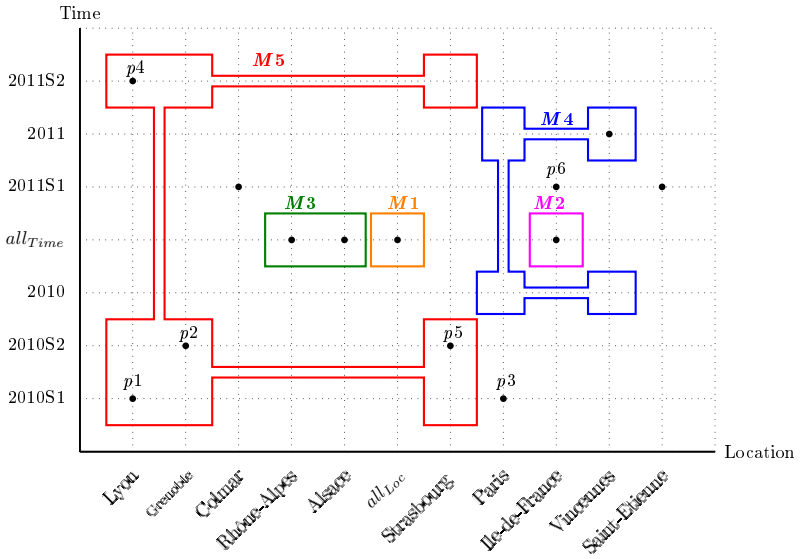
The insert algorithm is used to insert a new fact into a DyTree starting from its root. This algorithm is responsible for finding a suitable place for the incoming fact and placing it at that position in the tree. This algorithm, however, sometimes may need to call another important algorithm called split algorithm. The split algorithm is invoked in case of a directory node's overflow and splits the node into two new directory nodes. The details of these algorithms and a working example are discussed in the following.

3.1 Insert

The insert algorithm starts with packing the new fact which is a new point in HHMDS, into a data node. In this node's MBS, each edge is a singleton and it is made of the corresponding coordinate of the point. The insertion of a fact starts from the root and recursively continues downwards. The insert algorithm (algorithm 4.1) is attached to



(a)



(b)

MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\}\times\{all_{Time}\}$	340
$M2$	{Ile-de-France}$\times\{all_{Time}\}$	232
$M3$	{Rhône-Alpes, Alsace}$\times$ $\{all_{Time}\}$	108
$M4$	{Paris, Vincennes}$\times$\{2010, 2011\}	10
$M5$	{Lyon, Grenoble, Strasbourg}$\times$\{2010S1, 2010S2, 2011S2\}	18
p1	{Lyon}$\times$\{2010S1\}	1
p2	{Grenoble}$\times$\{2010S2\}	6
p3	{Paris}$\times$\{2010S1\}	4
p4	{Lyon}$\times$\{2011S2\}	8
p5	{Strasbourg}$\times$\{2010S2\}	3
p6	{Vincennes}$\times$\{2011S1\}	6

(c)

Figure 4.1: An example DyTree built with $DNCAP = 3$ and $OVLAP = 0$: (a) structure of the DyTree, (b) data space, and (c) MBS and facts.

Algorithm 4.1 Insert algorithm for DyTree's internal (*directory* or *super*) node

```
insert(DyTreeDataNode dataNode){
//insert dataNode into an internal node currentNode

Aggregate currentNode.measure with dataNode.measure
Set targetNode = chooseSubtreeForInsertion()
if(targetNode != NULL) then
  targetNode.insert(dataNode)
else
  if (nodes in currentNode.entrySet are data nodes)
    if (currentNode is a directory node)
      if (size(currentNode.entrySet) <= DNCAP)
        Add dataNode to currentNode.entrySet
      else
        currentNode.split(dataNode)
    else if (currentNode is a super node)
      Add dataNode to currentNode.entrySet
  else
    extend currentNode.MBS to accommodate dataNode.MBS
    Add dataNode to currentNode.entrySet
}
```

a every internal node of the DyTree and starts by updating the aggregate value in the node with the measure associated to the fact, and then chooses a subtree for the further insertion: if the MBS of any of the entries of the node *contains* the data node's MBS, it is chosen as the subtree, otherwise the one needing the minimum extension will be the candidate for further insertion. In this last case, the chosen node's MBS is extended to accommodate the data node's MBS. Then the same insert algorithm for the chosen node or the subtree is called. This process continues until the deepest level internal node is reached i.e. the one holding the data nodes as entries.

If the selected deepest level node is a directory node and it still has not reached its capacity, the data node is added to its entries. In case of overflow, the split algorithm (algorithm 4.2) for this node is called.

If the selected deepest level node is a super node, the data node is simply added to its entries.

3.2 Split

The split algorithm (algorithm 4.2) is invoked by the insert algorithm on occurrence of an overflow in a directory node's MBS. The algorithm starts with the selection of a split dimension and the level for splitting the MBS of the node called *splitting*

node. The dimensions and levels are selected on the basis of either the cardinality of the MBS's edges or the levels of the members of each edge constituting the MBS. In case of selection on the basis of cardinality, the chosen dimensions is one with the highest edge cardinality and the split level is the same as in the corresponding edge of the splitting node's MBS. Otherwise, the dimension corresponding to the highest level is chosen as split dimension and split level is a lower level in the hierarchy of the dimension. Once the split dimension and level are selected, the hyper-plane called *split hyper-plane* for the resultant splitted nodes could be established. The levels of all the dimensions except the split dimension in the split hyper-plane are the same as the dimension levels in the hyper-plane of the splitting node's MBS.

After establishing the split hyper-plane, the MBS of all the entries of the splitting node and that of the split provoking node are translated-up to the split hyper-plane and are stored temporarily in a list called *listNodes*. Two new directory nodes called *splitted node*s are created such that their MBS lie in the split hyper-plane. The two most distant (needing maximum extension) nodes from the listNodes are selected as seeds. One seed is inserted in each of the splitted nodes. For all the remaining nodes in the listNodes, one node is selected such that the difference of required extensions in splitted nodes MBS to accommodate the selected node's MBS is maximum. This criterion selects the node that is the closest to one of the splitted nodes as compared to the other splitted node and makes sure that the MBS are enlarged gradually. The selected node is inserted into either of the splitted nodes on the basis of extension, overlap area and number of existing entries, in order.

After distributing all the nodes in listNodes among the two splitted nodes, if the overlap area between the splitted nodes is more than the predefined limit, an alternate dimension and the corresponding level is chosen. This process continues recursively, unless we find a suitable split. If no suitable split is found, the node is rather adapted to a Super node.

In case of a successful split, the splitted nodes replace the splitting node in the entries of its parents which may start a bottom-up recursive split process. If the splitting node is root, a new root is created and the pointers to the splitted nodes are added to its entries. The splitting of root causes the tree depth to grow.

3.3 Running Example

To better understand the above algorithms, we explain their working through a simple running example. For this running example we use the two dimensional schema with dimensions Location and Time as presented in previous chapter (figure 3.1). However,

Algorithm 4.2 Directory node's split algorithm for DyTree

```
split(DyTreeNode provokingNode){
//splits the splittingNode using provokingNode

do{
  Choose the split dimension and corresponding level
  Set X = hyperplane of splittingNode MBS except along splitting
     dimension where it is set according to the selected level
  Set listNodes = provokingNode union splitingNode.entries
  for each node in listNodes
    TranslateUp node.MBS to X
  Create two directory nodes splitedNode1 and splitedNode2
  Select the two most distant nodes nodeSeed1 and nodeSeed2 from
     listNodes
  Insert nodeSeed1 in splitedNode1 and nodeSeed2 in splitedNode2
  Remove nodeSeed1 and nodeSeed2 form listNodes
  while(listNodes is not empty){
    Set node from listNodes so that extension(node.MBS,
       splitedNode1.MBS)- extension(node.MBS, splitedNode2.MBS) is
       maximum
    Insert node in the node (splitedNode1 or splitedNode2) that
       requires the minimum extension
    In case of tie: prefer the one that induces minimum overlap
       area
    In case of further tie: prefer the one that has less entries}
  if (ovlapArea(splitedNode1.MBS, splitedNode2.MBS) < OVLAP)
    success=true
}while (ovlapArea(splitedNode1.MBS, splitedNode2.MBS) >= OVLAP or
   all dimensions are not processed)
if (success=true)
  if (splitingNode is root)
    splitingNode.entries = {splitedNode1, splitedNode2}
  else
    splitingNode = splitedNode1
    if (splitingNode.parent is a super node or a directory node)
       and (is not full)
      Add splitedNode2 to splitingNode.parent.entrySet
    else
      splitingNode.parent.split(splitedNode2)
else
  Adapt splitingNode to super node
}
```

to better illustrate the working of the algorithms, we use an extended instance of dimension location as shown in figure 4.2. In this example, the facts are inserted one by one into the DyTree which implies the dynamic evolution of dimension tables and the data space. The capacity of a directory node is fixed to 3 and overlap limit is fixed to 0 in this example. These small values are used to simplify the explanation of the DyTree's working. SUM is used as the only aggregate function to construct the DyTree.

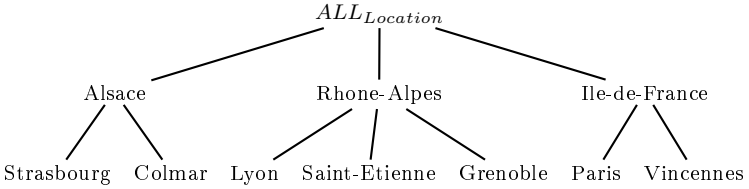
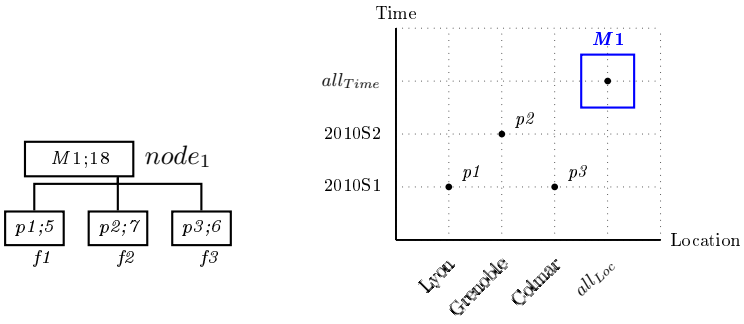


Figure 4.2: The instance of hierarchical dimension Location used in the running example.

In each of the following figures, the sub-figures 'a' show different states of the DyTree, sub-figures 'b' present the evolution of data space and the sub-figures 'c' depicts the description of MBS and facts in the DyTree.

State I (initial state) Figure 4.3 shows the initial state of a DyTree with three facts. As the capacity of a directory node is set to 3, first three facts are easily inserted in the root of the DyTree following the update of its aggregate value. $node_1$ is the root of the tree holding the MBS $M1 = \{all_{Location}\} \times \{all_{Time}\}$.

State II (insertion needing the splitting of root) The insertion of a new incoming fact *f4* in the initial state of DyTree, as shown in figure 4.3, starts from the root. The aggregate value of the root is updated to 19 (18+1). The addition of *f4* in the entries of the root causes the root ($node_1$) to overflow. As a result the N1 needs to be split. As the cardinality of both the edges of $node_1$'s MBS ($\{all_{Location}\}$ and $\{all_{Time}\}$) is 1, the choice of split dimension can not be made according to this criteria. Next, we check the levels of the members of both the edges. The level of the members of both the edges is also equal (i.e. 3) in their respective hierarchies. In this case, we can select any of the two dimensions to be the candidate split dimension. In our example, we select dimension Location to be our candidate split dimension and the chosen split level is 2 or Location.Region (i.e. $level(all_{Location}) - 1$). After this selection process, we can construct our split hyper-plane, which in this example is $< Region, ALL_{Time} >$.



(a) (b)

MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\}\times\{all_{Time}\}$	18
p1	{Lyon}×{2010S1}	5
p2	{Grenoble}×{2010S2}	7
p3	{Colmar}×{2010S1}	6

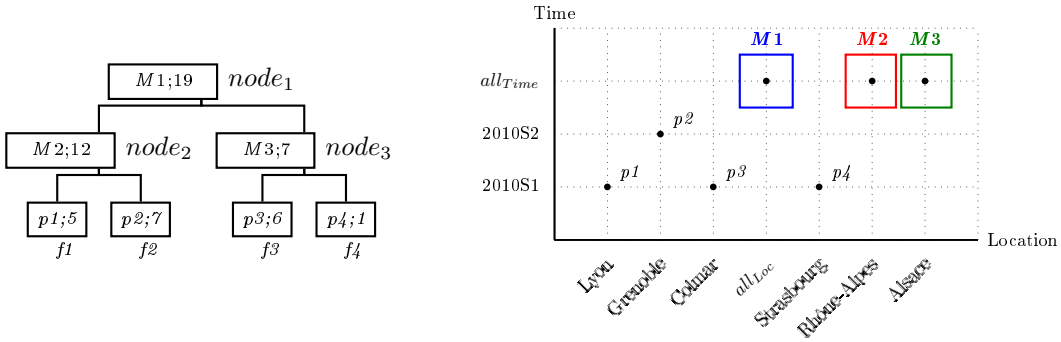
(c)

Figure 4.3: State I: Initial state of DyTree

Once the split dimension and level are selected, two most distant (in terms of required extension) nodes with respect to Location dimension are selected as seeds. It is important to note that in order to calculate the extension, the facts are first translated-up in the split hyper-plane. The distances among different combination of the facts (after the required translate-up operation) in our example are: $extension(f1|f2) = 0$, $extension(f1|f3) = 1$, $extension(f1|f4) = 1$, $extension(f2|f3) = 1$, $extension(f2|f4) = 1$ and $extension(f3|f4) = 0$. Therefore, the two seeds are found to be *f1* and *f3*. We note that as this extension is being calculated among the facts, $exteneions(a|b) = extension(b|a)$.

Next, two new directory nodes,$node_2$ and $node_3$ are created with empty MBS in the split hyper-plane. *f1* is inserted in $node_2$ while *f3* in $node_3$. Therefore, the $node_2$'s MBS ($M2$) becomes $\{Rhône\text{-}Alpes\}\times\{all_{Time}\}$ while the MBS of $node_3$ ($M3$) becomes $\{Alsace\} \times \{all_{Time}\}$. In the next step, each remaining fact is distributed one by one among $node_2$ and $node_3$. We insert *f2* in $node_2$ and *f4* in $node_3$, following the update of their corresponding aggregate values.

As the splitting node is the root of DyTree, it does not disappear, rather all the existing entries of $node_1$ are cleared and the pointers to $node_2$ and $node_3$ are added to its entries. The resultant state of the space and DyTree are depicted in figure 4.4.



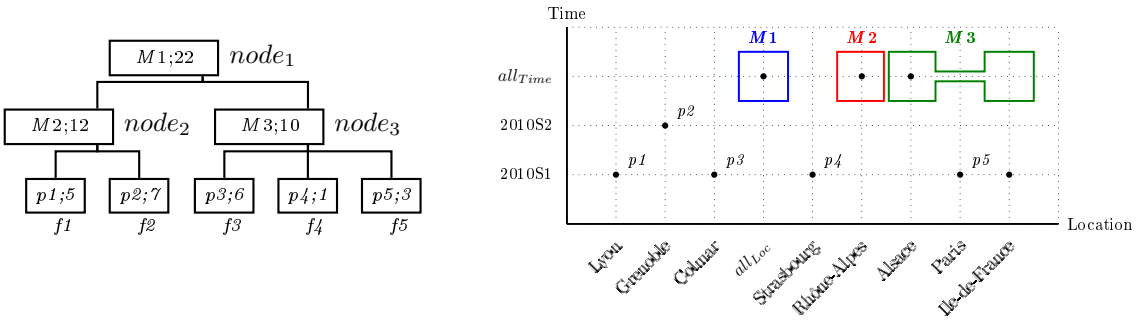
(a) (b)

MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\} \times \{all_{Time}\}$	19
$M2$	{Rhône-Alpes} $\times \{all_{Time}\}$	12
$M3$	{Alsace} $\times \{all_{Time}\}$	7
p1	{Lyon}×{2010S1}	5
p2	{Grenoble}×{2010S2}	7
p3	{Colmar}×{2010S1}	6
p4	{Strasbourg}×{2010S1}	1

(c)

Figure 4.4: State II: Insertion of *f4*.

State III (insertion needing the extension of an MBS) The insertion of *f5* in State II (as shown in figure 4.4) of the DyTree starts from the root whose aggregate value is updated using the measure value of *f5*. The fact's MBS is contained in the root, therefore the condition of containment is checked against all its entries. Since *f5* is not contained in either of the root's entries, extension (after the required translate-up of fact's MBS) of $node_2$'s and $node_3$'s MBS required to accommodate *f5* are calculated. The extension required in both the cases is same. Insertion of *f5* in any of $node_2$ and $node_3$ does not require any overlap and the number of existing entries in both of them is also same. This implies that *f5* can be inserted in either of $node_2$ and $node_3$ after extending the corresponding node's MBS. We choose $node_3$ to accommodate *f5*, therefore its aggregate value is updated and the MBS is extended. The $M3$ now becomes $\{Alsace,\ Ile\text{-}de\text{-}France\} \times \{all_{Time}\}$. Since $node_3$ is the lowest level directory node (i.e. one holding the pointers to data nodes or facts), the pointer of *f5* is added to the entries of $node_3$. The resultant state of DyTree is presented in figure 4.5.



(a) (b)

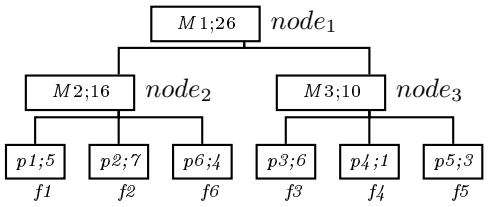
MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\} \times \{all_{Time}\}$	22
$M2$	{Rhône-Alpes} $\times$ $\{all_{Time}\}$	12
$M3$	{Alsace, Ile-de-France} $\times$ $\{all_{Time}\}$	10
p1	{Lyon}$\times${2010S1}	5
p2	{Grenoble}$\times${2010S2}	7
p3	{Colmar}$\times${2010S1}	6
p4	{Strasbourg}$\times${2010S1}	1
p5	{Paris}$\times${2010S2}	3

(c)

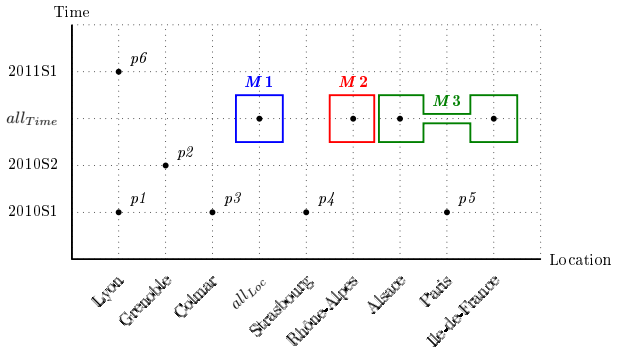
Figure 4.5: State III: Insertion of *f5*.

State IV (insertion without needing any split or extension) For the next update, we insert *f6* in the State III (see figure 4.5) of the DyTree. After the update of the root's aggregate value using the *f6*'s measure, it is checked if the *f6*'s MBS is contained in the MBS of its entries. Since $M2$ contains the MBS of *f6* and $node_2$ holds the data nodes as its entries, *f6* is simply added to the $node_2$'s entries and the aggregate value of $node_2$ is updated. The resultant state of DyTree is presented in figure 4.6.

State V (insertion needing the splitting of a non-root node) Insertion of *f7* in state IV (figure 4.6) of the DyTree causes the $node_3$ to overflow and induces a split. This results in the split of $node_3$ in $node_4$ and $node_5$. The pointer of $node_3$ is replaced by $node_4$ in its parent $node_1$ (i.e. the root) while the pointer of $node_5$ is added to the entries of $node_1$. The DyTree is now composed of four directory nodes and 7 facts, as presented in figure 4.7.



(a)



(b)

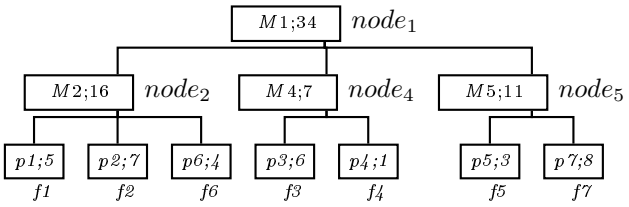
MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\} \times \{all_{Time}\}$	26
$M2$	{Rhône-Alpes} $\times$ $\{all_{Time}\}$	16
$M3$	{Alsace, Ile-de-France} $\times$ $\{all_{Time}\}$	10
p1	{Lyon}$\times${2010S1}	5
p2	{Grenoble}$\times${2010S2}	7
p3	{Colmar}$\times${2010S1}	6
p4	{Strasbourg}$\times${2010S1}	1
p5	{Paris}$\times${2010S2}	3
p6	{Lyon}$\times${2011S1}	4

(c)

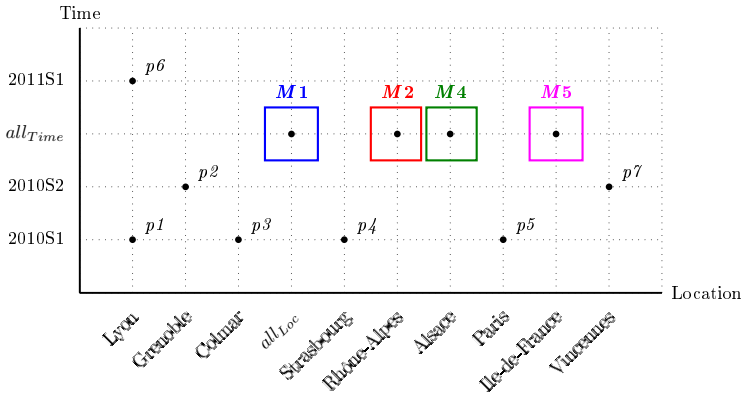
Figure 4.6: State IV: Insertion of a *f6*.

State VI (insertion needing the bottom-up recursive split) Insertion of *f8* in state V (figure 4.7) of DyTree causes the $node_2$ to split in $node_6$ and $node_7$. The pointer of $node_6$ replaces the pointer of $node_2$ in its parent while $node_7$ needs to be added to the parent's entries. This however, causes the parent ($node_1$) to overflow. Therefore, $node_1$ is split creating two new directory nodes $node_8$ and $node_9$. $node_4$, $node_5$, $node_6$ and $node_7$ are distributed among $node_8$ and $node_9$ as explained in the state II. This time distributing entries are directory nodes as opposed to the facts in state II, however this does not changes the procedure of distributing the entries among newly created splitted nodes.

Since the $node_1$ is root, its entries are cleared and the pointers to $node_8$ and $node_9$ are added to the $node_1$'s entries. As a result, the height of the tree grows. The new structure of the DyTree is presented in figure 4.8.



(a)

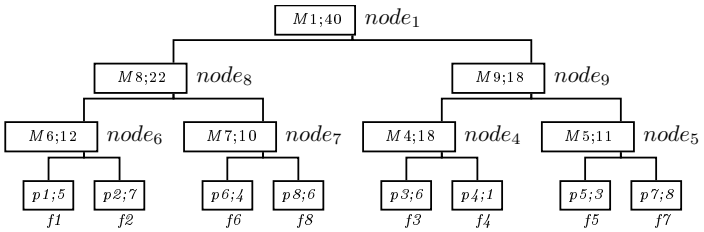


(b)

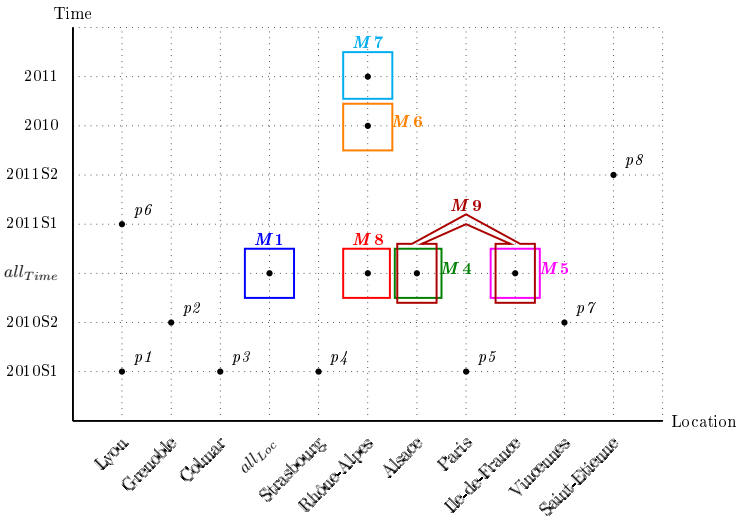
MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\} \times \{all_{Time}\}$	34
$M2$	{Rhône-Alpes} $\times$ $\{all_{Time}\}$	16
$M3$	disappeared	–
$M4$	{Alsace} $\times$ $\{all_{Time}\}$	7
$M5$	{Ile-de-France} $\times$ $\{all_{Time}\}$	11
$p1$	{Lyon}$\times${2010S1}	5
$p2$	{Grenoble}$\times${2010S2}	7
$p3$	{Colmar}$\times${2010S1}	6
$p4$	{Strasbourg}$\times${2010S1}	1
$p5$	{Paris}$\times${2010S2}	3
$p6$	{Lyon}$\times${2011S1}	4
$p7$	{Vincennes}$\times${2010S2}	8

(c)

Figure 4.7: State V: Insertion of a new fact $f7$.



(a)



(b)

MBS	Description	Measure/Aggregate
$M1$	$\{all_{Loc}\} \times \{all_{Time}\}$	40
$M2$	disappeared	–
$M3$	disappeared	–
$M4$	{Alsace} $\times$ $\{all_{Time}\}$	18
$M5$	{Ile-de-France} $\times$ $\{all_{Time}\}$	11
$M6$	{Rhône-Alpes} $\times$ {2010}	12
$M7$	{Rhône-Alpes} $\times$ {2011}	10
$M8$	{Rhône-Alpes} $\times$ $\{all_{Time}\}$	22
$M9$	$M4 \cup M5$ = {Alsace, Ile-de-France} $\times$ $\{all_{Time}\}$	18
p1	{Lyon}$\times${2010S1}	5
p2	{Grenoble}$\times${2010S2}	7
p3	{Colmar}$\times${2010S1}	6
p4	{Strasbourg}$\times${2010S1}	1
p5	{Paris}$\times${2010S2}	3
p6	{Lyon}$\times${2011S1}	4
p7	{Vincennes}$\times${2010S2}	8
p8	{Saint-Etienne}$\times${2011S2}	6

(c)

Figure 4.8: State VI: Insertion of *f8*.

4 Discussion on the DyTree

Now as we have detailed the construction of a DyTree, let us reconsider the DyTree's structure and highlight its principal characteristics.

4.1 Creation of Materialized Views

We have seen that at the beginning, the DyTree has only one directory node enclosing a single point (i.e. $(all_1, all_2, ..., all_n)$) that covers all the other points in the HHMDS. The splitting of the root produces two new directory nodes materializing the data at a lower aggregation level. The splitting of these new nodes then produces more nodes with data materialized at yet another lower aggregation levels. In other words, our splitting strategy splits the nodes along different levels of dimension hierarchy and produces the nodes materializing the data at different aggregation levels starting from higher to the lower ones. The resultant nodes, therefore, materialize either a multidimensional point in any hyper-plane of HHMDS, i.e. representing a detailed or an aggregated point (represented by an MBS all whose edges are singleton) or a range of multidimensional points. The creation of materialized views takes the position of members in dimension hierarchy into account and favors the grouping of the children of the same parents together. In case of ordered dimensions, the ordering among the members of the same level is also taken into account. Such a strategy is adopted to efficiently support the most commonly used aggregate OLAP queries (i.e. point, range etc.). As on splitting of a node, the entries are re-distributed among two new directory nodes, the facts in the tree do not follow their insertion order.

In DyTree, the materialized views, represented by MBS and stored in nodes, are not selected on the basis of the criteria such as cost/benefit analysis [Harinarayan 1996] or prior knowledge of frequently asked queries. The selection is rather guided by the relations and metrics presented in chapter 3. The insertion order of facts may change the structure of tree, i.e. two DyTrees indexing same facts may materialize different sections of cuboids, if the facts are not inserted in the same order in both the DyTrees. During the tree construction process, various MBS appear, vanish and reappear on insertion of new facts.

In our splitting algorithm, unlike hierarchical split algorithm used in DC-Tree, if no suitable splitting dimension is found by going one level down in hierarchy level, we skip that level and allow splitting at further lower levels. For example, If we can not find an overlap free split for $\{all_{Location}\}\times\{all_{Time}\}$ in hyper-planes $< Country, all_{Time} >$ or $< all_{Location}, Year >$, we allow the splitting in hyper-plane $< City, all_{Time} >$or

$< all_{Location}, Semester >$. This may cause the aggregation at level $Country/Year$ to skip but that would re-appear at the time of bottom-up recursive splitting later. This strategy would help considerably reducing the number of super nodes and improving the performance of the DyTree.

4.2 Dense Data Partitions

Numerous research works (e.g. [Beyer 1999, Yu-cai 2004, Cheung 2001]) have been carried out to efficiently deal with the data cubes by considering the density characteristic of the data sets in concerned applications. In the DyTree, indexed MBS are always dense because these MBS are data partitions constructed over a set of existing data points. The advantage of using such a data partitioning technique rather than a space partitioning one is to avoid indexing possibly large dead spaces in the working data space. Therefore using this strategy in DyTree means that it indexes only the parts of data space where some data points exist. Indexing only the existing data points without covering dead spaces is advantageous in both reducing the size of the index and augmenting the querying performance.

5 Querying the DyTree

In OLAP or data warehousing environment, we come across three types of queries: point queries, range queries and group-by queries. In the following, we discuss these algorithms and explain them through our running example.

5.1 Range Query

The range query algorithm (algorithm 4.3) executes a range query over the DyTree and returns the result of the query. For simplicity, the aggregate function we use to describe the working of the algorithm is SUM.

The input to a range query algorithm is a range query that is transformed into an MBS (called *query_ MBS*) by putting the ranges defined over each dimension in its corresponding edge of the query_MBS. Like insert, the range query algorithm starts with the root. For a node in the DyTree, if the query_MBS *contains* the node's MBS, the aggregate value associated to the node's MBS is added to the result. Otherwise, the two MBS are adapted to the same level by applying translate-up on the MBS that lies in the lower level hyper-plane. Then, if the query_MBS *overlaps* the node's MBS, the same algorithm is recursively called for each of its entries. If both the conditions

Algorithm 4.3 Range (SUM) query algorithm for a DyTree's internal (*directory* or *super*) node

```
rangeQuery(MBS query_MBS){
//calculates result, the aggregated measure of query

result = 0
if contains(query_MBS, currentNode.MBS)
  Aggregate result with currentNode.measure
if (query_MBS and currentNode.MBS are not in same hyper-plane)
  if (query_MBS is in higher level hyperplane than that of
      currentNode.MBS)
    TranslateUp currentNode.MBS to the hyperplane of query_MBS
  else
    TranslateUp query_MBS to the hyperplane of currentNode.MBS
if (query_MBS overlaps currentNode.MBS)
  for (eachnode in currentNode.entrySet)
    node.rangeQuery(query_MBS)
return result
}
```

are false, the subtree is simply ignored for the further inquiry. The algorithm returns the aggregated measure values corresponding to the range query at the end.

5.2 Point Query

Algorithm for point query is similar to the range query's algorithm. The only difference is that of each edge of a query_MBS in point query has only one element (i.e. cardinality of of each edge=1) as opposed to a range query where each edge may have a list of elements (i.e. range of values). The elements in each edge of point query may belong to any level of their respective dimension hierarchies: the queried point may lie in any hyper-plane of the HHMDS. Therefore, the calculation of a point query's result does not necessarily need to query the leaves and could be answered at higher aggregated levels of the DyTree.

5.3 Group-by Query

We consider a group-by query to be a collection of point queries. Therefore, a group-by query algorithm of DyTree simply transforms a group-by query in a set of point queries and then for each point query, above described point query's algorithm is called. The result of a group-by query is returned as a list of the results of the point queries.

5.4 Running Example

To have clear understanding, the above algorithms are explained in this section using our running example. For this purpose, we query the DyTree shown in figure 4.8.

Range Query We consider the query "Find the total amount of sales in Alsace and Rhône-Alpes during 2011 - 2013" with SQL equivalent "SELECT SUM(sales) FROM factTable WHERE Location.Region in ("Alsace", "Rhône-Alpes") and Time.Year BETWEEN 2011 AND 2013" to explain the working of the range query's algorithm:

To answer the above range query, the algorithm starts by transforming the query into an MBS, called query_MBS, such that query_MBS = {"Alsace", "Rhône-Alpes"} $\times$\{2011, 2013\}. After the transformation, it is checked if the query_MBS contains the root's MBS ($M1$). Since the answer is no, second condition is checked: does the query_MBS overlaps $M1$? Since $node_1$'s MBS is in a higher level hyper-plane than the query_MBS, query_MBS first needs to be translated up in the hyper-plane of $M1$ before the determination of overlap condition. After the required translate-up operation, query_MBS overlaps $M1$. In this case, the same query algorithm is invoked for all the entries of the root.

The query_MBS does not contain the $node_8$'s MBS ($M8$) but the overlap between query_MBS and $M8$ is greater than zero, therefore we continue exploring its entries. Among the $node_8$'s entries, query_MBS neither contains nor overlaps the $M6$, therefore this subtree is simply ignored and we continue querying the remaining entries of $node_8$. Since the query_MBS contains $M7$, the $node_7$'s aggregate value is added to the result and the subtree is not explored anymore. The value of result becomes 10. Since there are no more entries of $node_8$ are left for querying, we continue querying the sibling nodes of $node_8$ or in other words, the remaining entries of $node_1$.

Again, the query_MBS does not contain M9 but there is an overlap between M9 and query_MBS, therefore all the entries of $node_9$ are queried against the query_MBS. The overlap between $M4$ and query_MBS is greater than zero while neither of its entries' MBS is contained in query_MBS, therefore no changes are made to the result. Next, $node_5$ is queried against the query_MBS, whose MBS i.e. $M5$ is neither contained in query_MBS nor there is any overlap between query_MBS and $M5$, therefore this subtree is also ignored without making any changes to the result. As all the entries of $node_8$ and so as the $node_1$ are queried, the calculated value of result (i.e. 10) is returned.

Point Query To explain the working of a point query, we consider a point query "Find the total amount of sales in Alsace for the year 2010". This query is in a higher level hyper-plane i.e. an aggregate point query. The query can be re-written in SQL as "SELECT SUM(sales) FROM factTable WHERE Location.Region = "Alsace" and Time.Year= 2010".

The transformation of above query in MBS produces a query_MBS= {"Alsace"}× {2010}. The execution of query starts similar to the range query. The $M8$ is neither contained nor overlaps query_MBS, therefore this subtree is simply ignored. $M9$ and so as the $M4$ overlap query_MBS, therefore the entries of their respective nodes are queried against the query_MBS. Both the entries of dir_4 (i.e. *f3* and *f4*) are contained in query_MBS whose measure values are added up to make the result = 18. As $M5$ is neither contained in query_MBS nor they overlap with each other, dir_5 is ignored. as all the entries of dir_1 are queries, the calculated result = 18 is returned.

Group-by Query A group-by query is of the form "Find the total amount of sales in Alsace by Year" which can be expressed in SQL as "SELECT SUM(sales), year FROM factTable WHERE Location.Region= "Alsace" GROUP BY Time.Year".

Since the data needs to be grouped by Year and in our system it spans three years, the above group-by query is transformed into the following three query_MBS:

- query_MBS1= {"Alsace"}×{2010}
- query_MBS2= {"Alsace"}×{2011}
- query_MBS3= {"Alsace"}×{2012}

Once the query is transformed into the MBS, the point query algorithm is called for each of the above query, described in form of MBS. A list of the result of each individual point query is returned as the result of the group-by query.

6 Conclusion

In this chapter, we have addressed the problem of data indexing in dynamic data warehousing environment and proposed a data structure called DyTree. The DyTree is built dynamically and indexes both detailed data (i.e. facts) and aggregated data (i.e. MBS) in a hierarchical hybrid multidimensional data space. The DyTree does not only indexes the data but also stores it in nodes which are the materialized section of cuboids. For this reason we also consider the DyTree a cubing technique. Thanks

to its data partitioning technique and the proposed metrics, the DyTree nodes are always dense and do not index large dead spaces. After detailing these algorithms, in the next chapters, performance of the proposed algorithms is evaluated with respect to an existing solution.

Chapter 5

Experimental Evaluation

Experimental evaluation of the proposed algorithms is an important part of this thesis. For this purpose, we outline parameters that are found interesting in evaluation of a multidimensional data cubing solution. We use data sets to assess the efficiency and effectiveness of our algorithms from different aspects. The use of the benchmark is important for credible tests while the other data sets are used to determine the suitability for high dimensional data. In literature, we find the DC-Tree to be the closest to our problem, therefore we use it to compare the performance of our solution. The comparison tests presented in this chapter summarize the efficiency of our proposed solution. We have also developed a prototype which lets us know the possible shortcomings in the proposed solution and allows to see it working and better understand it. In this chapter, we present the main functions and features of the implemented DyTree and those necessary for the performance evaluation workflow.

Chapter Outline

1 Methodology

In preceding chapters, we have defined a model for hybrid hierarchical multidimensional data space called HHMDS and the related concepts. We have also proposed DyTree and introduced the algorithms to construct and query the tree. Now in this part of the thesis, our objective is to experimentally assess the efficiency and effectiveness of our solution using our prototype. For this purpose, we devise a careful experimental process to effectively measure and analyze the efficiency of the structure of DyTree and its associated algorithms. These experiments will let us find the answer to following questions:

- **Question 1**: Are the DyTree and associated algorithms efficient?
- **Question 2**: How does the DyTree behave under different circumstances?

1.1 Criteria for Experimental Evaluation

Obviously, the first question is about the performance of the solution. Our criteria for performance evaluation are based on execution time and memory space usage. Here, execution time corresponds to both insertion time of a new fact and query response time of an OLAP query. The prime objective of a dynamic cubing structure is always to minimize the execution time.

Our implementation of the DyTree employs the following storage strategy: internal (directory and super) nodes that hold the calculated aggregates are stored in main memory while the leaves (data nodes) holding the detailed facts are stored on disk. As for the execution time, efforts are made to reduce the memory space usage as well. Therefore, this criteria is also important in performance evaluation of the DyTree.

Question 2 addresses the effectiveness of the solution by observing the behavior of the DyTree. To answer this question, the evaluation criteria depends on the structure and contents of tree and its nodes. These criteria (tree height, width etc.) are expected to affect the performance and may change with the nature of data set or input parameters of tree construction. Our objective by answering this question is to analyze this effect to better understand the behavior of the DyTree which helps tuning and optimizing its performance.

Finally, the criteria we use to respond these questions are:

- Performance metrics to respond question 1: execution time and memory usage.
- Behavioral metrics to respond question 2: structure and contents of the DyTree.

1.2 Outline of the Experimental Evaluation Workflow

The general principle of our experimental evaluation workflow is based on:

- defining a set of configuration parameters to characterize the inputs (data sets, queries sets, algorithm input parameters).
- defining a set of metrics to a analyze and qualify the output (performance metrics) of the DyTree.
- performing experiments to compare:
  - The values of performance metrics obtained using our solution and a reference solution of the state of the art.
  - The values of performance and behavioral metrics obtained by varying the input parameters of the proposed algorithms.
  - The values of performance and behavioral metrics obtained by using the data sets with different characteristics.

Flow diagram presented in figure 5.1 summarizes the overall process of experimental evaluation. For every experiment, a data set is used that is generated on the basis of a use case schema and other data set generation parameters such as size, data density etc. Input parameters for algorithms are set and the tree is constructed and updated. Queries sets are generated with different characteristics such as type of queries. These queries are executed on the tree. Metrics are collected at each step of the process for later analysis. We detail each process and the input/output parameters in the following.

2 Inputs to the Workflow

In this section, we discuss the inputs (as shown in figure 5.1) to our experimental evaluation workflow.

2.1 Data Sets

Our data sets consist of a data warehouse schema, instances of dimensions stored in dimension tables and a fact table.

For experiments, we use two types of data sets. The first type are data data sets whose schema and data generation process is based on a data warehouse version of

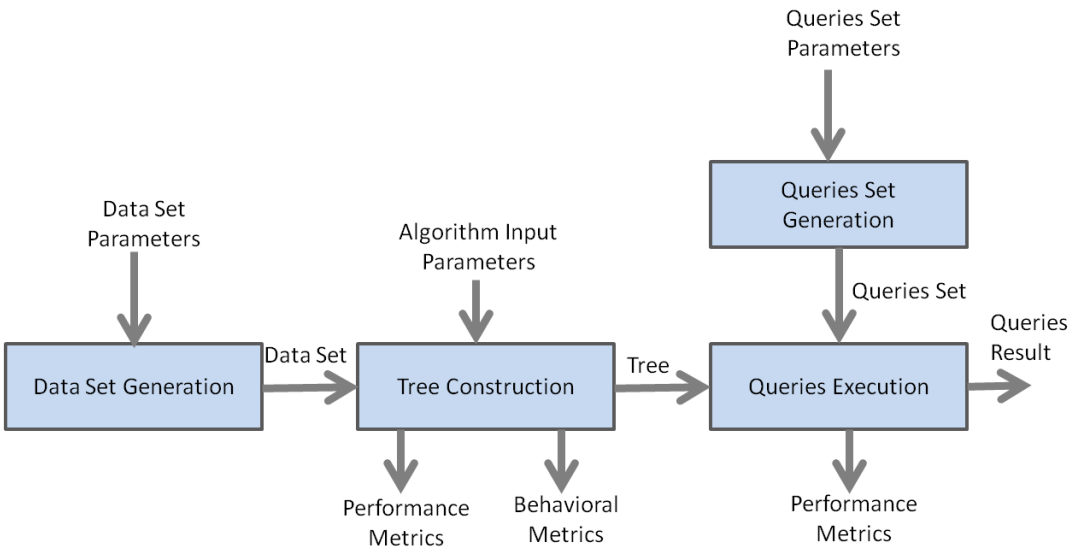


Figure 5.1: Outline of experimental evaluation process

TPC-H benchmark. The second type of data sets are customizable synthetic data sets. For all the data sets a numerical value is used as measure in the fact table.

In the following, we first describe the schema of data sets and then provide the detailed parameters.

2.1.1 Schema of Star Schema Benchmark Data Set

Our data sets of first category are based on star schema benchmark (SSB) [O'Neil 2009]. SSB is a modified version of famous TPC-H benchmark [TPC 2011]. TPC-H was originally proposed for decision making relational database systems while the SSB proposes the modifications to make it suitable for data warehouses. SSB is a widely accepted benchmark and has already been used by many researchers to validate their proposed data warehousing solutions. SSB is based on star schema as shown in figure 5.2. The schema constitutes of four dimension: CUSTOMER, SUPPLIER, PART and DATE while the facts are in table LINEORDER. These dimensions are organized in hierarchy as presented in figure 5.3. The details to generate the instances of each of these dimensions and the fact table (LINEORDER) are provided in [O'Neil 2009]. In table 5.1, we summarize the number of distinct values for important attributes in the tables of SSB.

2.1.2 Schema of Synthetic Data Sets

The data sets of our second category are synthetic data sets.

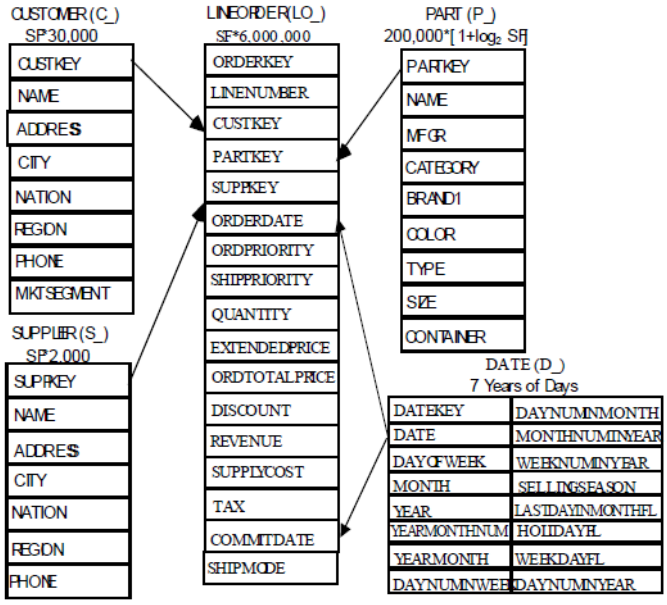


Figure 5.2: Schema used in Star Schema Benchmark

Some of synthetic data sets are generated by varying the number of dimensions. The general data warehouse schema of these data sets is shown in figure 5.4. Each dimension constitutes of between 2 to 5 hierarchy levels. We limit the maximum number of levels to 5 because it is quite reasonable for general data warehouse applications and we do not need to consider more number of levels per dimension. These data sets are listed from # 6 to 10 in table 5.2.

Other synthetics data sets that are based on 3-dimensional schema with each dimension constituted of between 2 to 5 hierarchical levels are also generated. These data sets are numbered 11 - 15 in table 5.2.

2.1.3 Data Set Parameters

The data sets are characterized by two parameters i.e. size and nature of the data sets.

Size of Data Set Size of a data set is an important characteristic of the data set. In our evaluation process, we determine the size of a data set on the basis of following two criteria:

Number of Dimensions This parameter could affect the performance of the proposed algorithms in terms of both execution time and usage of disk space. We propose to vary the number of dimensions from 10 to 30.

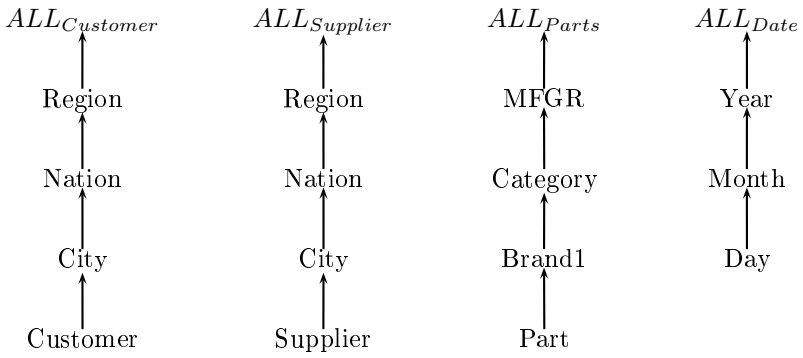


Figure 5.3: Hierarchical organization of dimensions used in Star Schema Benchmark

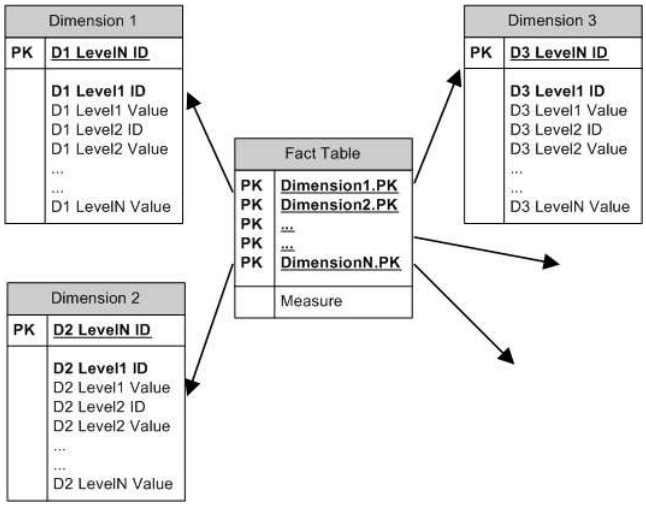


Figure 5.4: Schema of synthetic data sets

Number of Facts The size of a data set could be determined in terms of the number of tuples of fact table. The size of dimension tables in a data warehouse is negligible as compared to the size of the fact table. This is particularly more convenient in case of dynamic data warehousing application where the tuples of a fact table are inserted on tuple by tuple basis and the major interest lies in determining the insertion efficiency as time needed to insert an individual tuple into the data warehouse.

Nature of Data Set Data sets are also characterized according to the nature of input data. We consider following two parameters to vary the nature of our data sets.

		Cardinality
Customer	Region	5
	Nation	25
	City	250
	Customers	30,000
Supplier	Region	5
	Nation	25
	City	250
	Suppliers	2000
Part	MFGR	5
	Category	25
	Brand	1000
	Parts	200,000
Date		Days of 7 Years
LINEORDER		10,000,000

Table 5.1: Cardinality of the instances of different tables involved in SSB

Ordering of Facts Data space of a fact table could be composed of ordered or non-ordered dimensions. In case there is at least one ordered dimension, data could be ordered with respect to one of the ordered dimension. The transaction time, for example, in a system respects the total order among the members of the ordered time dimension and all the facts are ordered according to this transaction time. This order, however, is not surely the insertion order of the facts because it may be compromised due to some technical issue (such as system down time, network failure etc.) while migrating data from source systems to the target data warehouse.

As previously discussed, ordering among the members of the domain sets of ordered dimension is exploited in our solution. The proposed metrics let us group the closed values together in same MBS which, in effect, helps in improving the insertion and query performance of the DyTree. Therefore, it is important to analyze if the out of order insertion of some facts affects this grouping strategy of ours. For this purpose we use multiple data sets (see # 2 - 5 in table 5.2) in which 5%, 10%, 15% and 20% of the facts arrive with delay.

Density The volume of a multidimensional data space represents the number of distinct detailed data points (i.e. possible facts) that could possibly lie in the data space. Let D_1, D_2, ..., D_n be n dimensions of a multidimensional data space. The volume of the data space S can be calculated as:

$$volume(S) = \prod_{i=1}^{n} |domain(l_i^1)|$$

If $||domain(l_1^1) \times domain(l_2^1) \times ... \times domain(l_n^1)||$ is the number of existing data points in the lowest level hyper-plane of S (i.e. the facts in the fact table) then the density of the data set or the S is given by:

$$density(S) = \frac{||domain(l_1^1) \times domain(l_2^1) \times ... \times domain(l_n^1)||}{volume(S)}$$

For instance, consider a multidimensional data space, of a retail store data warehousing application, with three dimensions i.e. customers, products and date. Now, each customer would not buy all the products offered by the store, rather a customer (in general) buys a very small number of the products as compared to the number of products offered by the store. Similarly, the customer does not buy his required products every day. This means that many possible combinations of the customer, product and date are not materialized and hence the collected data set becomes sparse.

For example, we consider that the retail store has only 10 customers, offers 200 products and the data is maintained for only 1000 days. Therefore, the volume of the data space is given by:

$$volume(S) = 10 * 200 * 1000 = 2,000,000$$

Let us suppose that each customer has been to the store on 50 different days and on each visit each of them has bought 20 different products. This implies that existing number of data points in the lowest level hyper-plane of the data space is 10,000 $(10 * 20 * 50)$ and the calculated density would be:

$$density(S) = \frac{10,000}{2,000,000} = 0.005$$

On the contrary, an application aimed at recording the electricity consumption of each customer in the country after every hour, collects a very dense data set.

In this research work, by varying the density of data sets, we aim to analyze if the density of data sets affects the density of the the constructed data partitions i.e. the nodes of the DyTree and the overall performance of the DyTree.

#	Data Set	Category	Size		Nature	
			Dimensions	Tuples	Ordering w.r.t Time Dimension	Density
1	SSB	SSB	4	10,000,000	ordered	$\approx 10^{-7}$
2	SSB-d1		4	10,000,000	5% delay	$\approx 10^{-7}$
3	SSB-d2		4	10,000,000	10% delay	$\approx 10^{-7}$
4	SSB-d3		4	10,000,000	15% delay	$\approx 10^{-7}$
5	SSB-d4		4	10,000,000	20% delay	$\approx 10^{-7}$
6	syn-d10	Dimension Scaling	10	10,000,000	ordered	$\approx 10^{-7}$
7	syn-d15		15	10,000,000	ordered	$\approx 10^{-7}$
8	syn-d20		20	10,000,000	ordered	$\approx 10^{-7}$
9	syn-d25		25	10,000,000	ordered	$\approx 10^{-7}$
10	syn-d30		30	10,000,000	ordered	$\approx 10^{-7}$
11	syn-dns20	Variable Density	3	10,000,000	ordered	0.2
12	syn-dns30		3	10,000,000	ordered	0.3
13	syn-dns40		3	10,000,000	ordered	0.4
14	syn-dns50		3	10,000,000	ordered	0.5
15	syn-dns60		3	10,000,000	ordered	0.6

Table 5.2: Summary of the data sets used in experimental evaluation

2.1.4 Synthesis of Data Sets

In table 5.2, we summarize the characteristics of data sets. These data sets are used as input for different experiments we run to evaluate the performance and behavior of the DyTree.

The data sets prefixed with SSB are based on star schema benchmark, while all other data sets are synthetic data sets. We use data sets # 6 - 10 to see the effect of dimension scaling while the data sets # 11 - 15 are used to analyze the effect of variation in fact tables density on the performance of our solution. All other experiments are done using the data sets based on SSB (see # 1 - 5 in table 5.2) which are aimed at performance evaluation of DyTree in comparison of DC-Tree and to see the effect of delayed insertion of some of the facts in the DyTree.

2.2 Queries Set

We generate three types of queries which are usually used in OLAP analysis i.e. point queries, range queries and group-by queries. All three types of queries are generated randomly using randomly chosen aggregation level.

For a point query, a random multidimensional point is chosen to query it for an aggregated or measure value. This multidimensional point needs not necessarily be in the lowest level hyper-plane of the space, but could rather lie in any hyper-plane of the HHMDS. We represent a point query as an MBS whose all the edges are singleton. For example, a point query "Find the total amount of sales for Mobile phones in November" is represented as $\{all_{Location}\} \times \{Mobile\} \times \{November\}$.

In case of range queries, a hyper-plane in the HHMDS is selected for the range query. A random range of domain values belonging to the domain sets of the levels of each dimension in the chosen hyper-plane are selected. The aggregate value is queried for all the points that lie in the subspace formed by the selected ranges of values for each dimension. A range query is also represented by an MBS but, in this case, the edges may represent a range of values for each dimensions. For example, the query "Find the total amount of sales during the period from October to December in cities of Paris, Lyon and Grenoble" is represented by the MBS $\{Lyon, Paris, Grenoble\} \times \{all_{Product}\} \times \{October, December\}$.

To generate a group-by query, a hierarchical level is randomly selected for a randomly chosen dimension. A group-by is represented by a set of MBS. For example, "Find the total amount of sales for TV sets in by region during the month of November" is represented by a set of MBS i.e. $\{\{Alsace\} \times \{TV\} \times \{November\}, \{Rhône\text{-}Alpes\} \times \{TV\} \times \{November\}, \{Ile\text{-}de\text{-}France\} \times \{TV\} \times \{November\}\}$.

It is obvious from the above examples that a group-by query could be regarded as a set of individual point queries. Therefore, for experimental evaluation, we do not consider the group-by queries any more. For range and group-by queries, we generate the sets of 50 queries for each category. For all these queries, SUM is used as aggregate function. The queries are generated using randomly selected aggregation level as the input parameter. The aggregation level defines the level at which the aggregation of measure is queried.

2.3 Algorithm Input Parameters

Two input parameters are used by tree construction algorithms of the DyTree. These are directory node capacity and overlap limit. We briefly describe these two param-

eters in this subsection.

Directory Node Capacity Directory node capacity $DNCAP$ is a fixed and pre-defined value. It is the maximum number of nodes that can be pointed to by a directory node.

Directory node capacity has a great impact on performance of the tree's algorithms and its behavior. A larger capacity directory node means that a larger number of nodes are indexed sequentially under a directory node. This could degrade the query performance, but on the other hand it minimizes the invocation of nodes splitting algorithm which is a costly algorithm and is called once a directory node overflows. Memory and disk space allocation is also dependent on the directory node capacity because it is allocated, according to the capacity, at the time of creation of every new directory node.

Overlap Limit Like directory node capacity, overlap limit $OVLAP$ is also a fixed pre-defined value. It determines the maximum amount of allowed overlap between the MBS of any two nodes.

Since the overlap limit controls the shared area of two or more MBS, it could require searching algorithm to verify the searching criteria against less/more nodes and consequently affects the performance of both construction and querying algorithms of the DyTree.

3 Outputs of the Workflow

Outputs of our experimental evaluation workflow are the evaluation metrics that are used to analyze the performance of the DyTree:

- with respect to different inputs and
- with respect to the DC-Tree.

These metrics can be grouped into two categories, namely performance metrics and behavioral metrics.

3.1 Performance Metrics

Performance metrics are used to evaluate and analyze the performance efficiency of the proposed solution. For this purpose, we use tree construction time, atomic fact

insertion time, query response time and the memory usage metrics. Definitions of these metrics are provided below.

Atomic Fact Insertion Time Atomic fact insertion time is the time to insert an individual fact in an existing DyTree. As the facts are inserted quite frequently in dynamic systems, an efficient system is one with the minimum insertion time. This metric is recorded in milliseconds.

Tree Construction Time Tree construction time is the time to build a tree from scratch. Since the facts in the DyTree are always inserted one by one, we can alternatively say that the tree construction time is total insertion time to insert a certain number of facts. We use minutes as measuring unit to record the tree construction time.

Query Response Time Query response time is defined as the time taken by the solution to fetch the results corresponding to a given query. We record the query response type of all the queries in milliseconds. An algorithm with minimum query response time is desirable.

Memory Usage To be a suitable solution, we expect our solution to be space efficient. For this purpose, we evaluate the memory usage efficiency of the trees. The memory usage is measured in gigabytes (GB).

3.2 Behavioral Metrics

The behavioral metrics are defined to quantify the different aspects of the DyTree that are found to be interesting in understanding its behavior. These metrics are grouped into two categories: metrics for tree structure and metrics for nodes structure.

3.2.1 Metrics for Tree Structure

We intuitively suppose that the tree structure is affected by the change in directory node capacity and overlap limit. It may also be affected by the nature of data sets. Therefore, the analysis of the tree structure will help us better understand its relation with input parameters and its effect on the performance of the DyTree. Following metrics help us in this analysis. We use figure 5.5 to illustrate these metrics.

Number of Directory Nodes Directory Nodes are fixed capacity nodes that hold the pointers to other nodes in the tree. In the beginning, tree has only one directory node and the new directory nodes are created through nodes splitting process. Therefore, number of directory nodes is an indicator of number of splits occurred and gives an idea about the evolution of the tree. DyTree shown in figure 5.5 has seven ($node_1$-$node_7$) directory nodes.

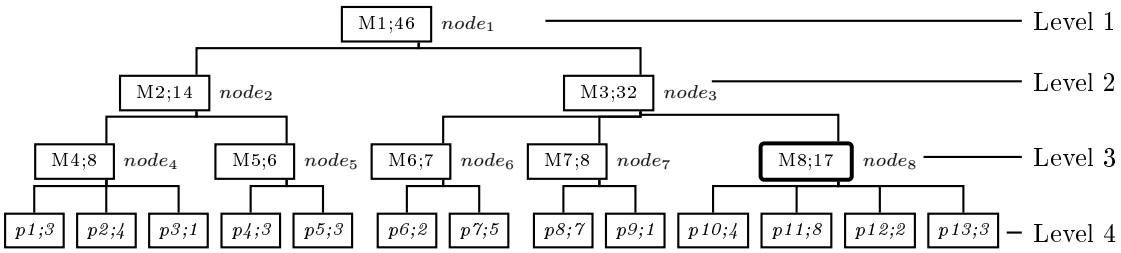
Number of Super Nodes Super nodes are larger capacity nodes that hold pointer to a larger (as compared to the directory nodes) number of other nodes in the tree. Super nodes are created when no suitable split dimension is found at the time of a directory node's splitting. The larger number of super nodes may indicate the wastage of memory and/or disk space and could cause the degradation in efficiency of the search and insert algorithms. Number of super nodes are dependent upon the directory node capacity and overlap limit. Figure 5.5 has only one super node ($node_8$) with four entries.

Tree Height Height of a DyTree is defined as distance between the root node to the deepest level node in the tree. For example, the height of DyTree in figure 5.5 is 4. In DyTree, the height of the tree grows when the root node is split. Therefore, the tree height should be an indicator of how many times the root node is split.

A DyTree with smaller value of tree height needs to sequentially index a greater number of nodes as compared to another DyTree indexing the same data with a higher value of tree height. We also recall that while searching for a node, our insertion and querying algorithms consider all the child entries of an interesting node (i.e. a node that could possibly accommodate the newly coming fact or the one that would participate in the calculation of a query result). This means that locating a particular node in the first DyTree (i.e. with smaller height) would need, on average, a greater number of consultations. Therefore, we expect the DyTree with higher value of tree height to show better performance as compared to the one with smaller value.

Average Tree Levels Width The width of a level is defined as the number of nodes at that particular level in the tree. Average value of the width of individual levels in the tree is known as average tree levels width. We do not consider first and last level for the calculation of this metric because the width of first level is always one and the width of the last level of DyTree is always equal to the number of facts,

and if considered for calculation, these nodes may reduce the usability of the metric. The average tree levels width of DyTree in figure 5.5 is 3.5.



Name	Description	Measure/Aggregate
M1	$\{all_{Loc}\}\times\{all_{Pro}\}\times\{all_{Time}\}$	46
M2	$\{all_{Loc}\}\times\{all_{Pro}\}\times\{Oct\}$	14
M3	$\{all_{Loc}\}\times\{all_{Pro}\}\times$ {Nov}	32
M4	{Alsace, Rhône-Alpes}$\times\{all_{Pro}\}\times$ {Oct}	8
M5	{Ile-de-France}$\times\{all_{Pro}\}\times$ {Oct}	6
M6	{Alsace}$\times\{all_{Pro}\}\times$ {Nov}	7
M7	{Rhône-Alpes}$\times\{all_{Pro}\}\times$ {Nov}	8
M8	{Ile-de-France}$\times\{all_{Pro}\}\times$ {Nov}	17
p1	{Colmar}$\times$ {Sony,}$\times$ {01 Oct}	3
p2	{Lyon}$\times$ {HTC}$\times$ {01 Oct}	4
p3	{Grenoble}$\times$ {S. Vaio}$\times$ {03 Oct)	1
p4	{Paris}$\times$ {Nokia}$\times$ {11 Oct}	3
p5	{Paris}$\times$ {Samsung}$\times$ {13 Oct}	3
p6	{Colmar}$\times$ {Sony}$\times$ {07 Nov}	2
p7	{Mulhouse}$\times$ {Sony}$\times$ {23 Nov}	5
p8	{Lyon}$\times$ {Dell}$\times$ {05 Nov}	7
p9	{Paris}$\times$ {Assus}$\times$ {05 Nov}	1
p10	{Créteil}$\times$ {Panasonic}$\times$ {10 Nov}	4
p11	{Paris}$\times$ {HP}$\times$ {07 Nov}	8
p12	{Vincennes}$\times$ {Sony}$\times$ {17 Nov}	2
p13	{Paris}$\times$ {Apple Mac}$\times$ {03 Nov}	3

Figure 5.5: An example of a DyTree built using a 3-dimensional schema with $DNCAP = 3$ and $OVLAP = 0$

This metric helps us in understanding the affect of different input parameters on the tree. As our hypothesis, we expect that this metric is dependent on directory node capacity. Contrary to the tree height, a DyTree with greater value of average tree levels width indexes a larger number of nodes at the same level or in other words,

more nodes are sequentially indexed under a node than in another DyTree indexing the same data and having a smaller value of average tree levels width. A lower value of average tree levels width, therefore, is expected to result in better performance of the algorithms.

3.2.2 Metrics for Nodes Structure

The internal (directory and super) nodes hold pointer to other child nodes called entries. Memory is allocated for each of these internal nodes according to its predefined capacity at the time of creation while the pointers to the entries are added at run time. Following metrics help us to determine if the indexing through these nodes is efficient or it causes the poor performance of algorithms and wastage of disk space.

Let $node < M, entrySet, a_1, a_2, ..., a_k >$ be a directory node of a DyTree with $DNCAP$ being the directory node capacity.

Node Fill Ratio The fill ratio ($fillratio(node)$) of this nodes can be calculated as:

$$fillratio(node) = \frac{size(node.entrySet)}{DNCAP}$$

The fill ratio of a node determines the extent to which the node is filled. For example, in figure 5.5: $fillratio(node_1) = 2/3$, $fillratio(node_4) = 3/3$ etc.

In our experiments, we record the average value of nodes fill ratio of all the directory nodes in the DyTree. We do not consider data and super nodes for the calculation of average nodes fill ratio, because the data nodes having capacity of 1 are always filled while the super nodes are not fixed in size. A lower value of average nodes fill ratio means that the nodes are not filled up to the capacity and hence indicates the wastage of memory.

Node Leaf Ratio If $node$ indexes $nbFacts$ data nodes under it then the leaf ratio, noted as $leafRatio(node)$, of the node can be calculated as:

$$leafRatio(node) = \frac{nbFacts}{volume(node.M)}$$

The leaf ratio of a node measures the space utilization of the node. It is important to note that a node does not necessarily index all the facts in the data space covered by the points of the node's MBS i.e. $nbFacts \neq ||node_{int}.M||$. Therefore, leaf ratio of a node holding an MBS is not same as the density of the MBS. For example, in

figure 5.5: $leafRatio(node_3) = 8/3120$, $leafRatio(node_5) = 2/806$ etc. The leaf ratio of a data node is always one.

A lower value of a node's leaf ratio means that the node covers a large volume without indexing a sufficient number of data nodes under it. In our experiments, we use the average value of nodes leaf ratio of all the individual internal nodes of the DyTree.

4 Synthesis of the Workflow

We can now detail the outline schema shown in figure 5.1 as in figure 5.6.

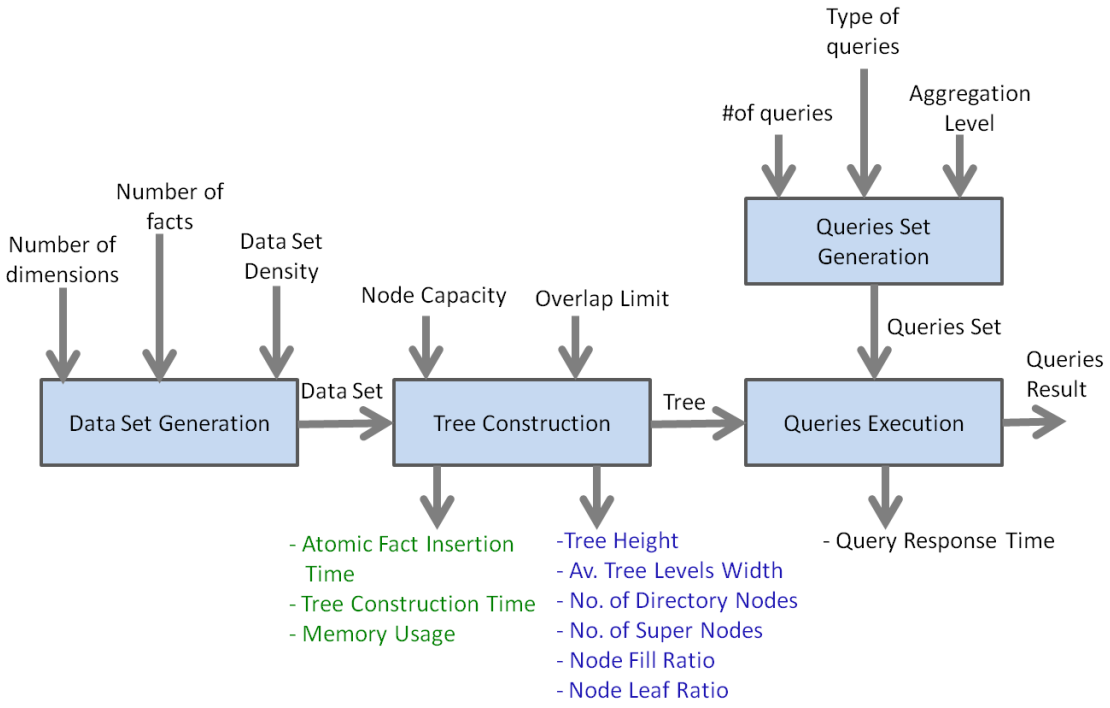


Figure 5.6: Detailed outline of experimental evaluation process

5 Experimental Results and Discussion

All the experiments presented here are carried out on a 2.67 GHz Intel Core 2 Duo machine with 14 GB of RAM, 500 GB of DELL MD32xxi disk storage running Microsoft Windows 2008 R2 operating system.

Results of various experiments are presented in this section. First, we compare the performance of DyTree against the DC-Tree, and then we present the study of

effects varying the input parameters, outline in experimental evaluation workflow, on the DyTree's behavior. In this chapter, we do not present the trivial results and discuss only the interesting ones.

5.1 Performance Comparison with the DC-Tree

The performance of the DyTree is evaluated on the basis of tree construction, atomic fact insertion and query response time. We also consider the memory usage efficiency to measure the performance.

The data sets used for this performance evaluation process are presented in table 5.2.

5.1.1 Atomic Facts Insertion and Tree Construction

We first discuss the results for atomic insertion of facts (update of existing DyTree) and then present the results for the construction of trees from scratch. To asses the space efficiency of trees, memory usage metrics is used.

The trees are constructed using an overlap limit of 0 and the directory node capacity is fixed at 15 while using SSB, and 35 while using syn-dns40 data set.

Atomic Fact Insertion Time Figure 5.7 illustrates the comparison results of atomic fact insertion time in DC-Tree and DyTree. The results presented in the figure 5.7a are obtained using SSB (see table 5.2) while those presented in figure 5.7b are obtained using syn-dns40. The insertion time is presented in milliseconds. In both these cases, DyTree shows better performance than the DC-Tree. We observe that initially, with smaller tree size, the difference is not very important but as the size of the tree grows, the difference in insertion time becomes more and more visible.

We also observe that the improvement in insertion time of DyTree presented in 5.7b is more than that in 5.7a. The reason for this difference is explained by the fact that the syn-dns40 is a higher density data set or in other words, has the dimension tables with low cardinality values as opposed to the SSB. This decreases the probability of successful nodes splitting and hence increases the number of super nodes. As the super nodes sequentially index a larger number of nodes under itself, finding a suitable directory or super node to accommodate the incoming fact takes longer. Therefore the more efficient splitting algorithm of DyTree shows more improvement in the case where the probability of a successful split is less i.e. when using syn-dns40.

Tree Construction Time The tree construction time is the cumulative time required for atomic insertions of multiple facts in an empty tree. We study this insertion time to take the effect of all splits into account. The splitting algorithm is found to be the costliest algorithm used in the atomic insertion of a fact in a DC or DyTree. Therefore to take the effect of all the splits into account, we present the obtained results for this metric in figure 5.7. The recorded cumulative insertion time is recorded in minutes.

Similar to the results of single record insertion, DyTree outperforms the DC-Tree in both the cases, i.e. when using SSB (figure 5.7c) and when using the syn-dns40 (figure 5.7d).

Memory Usage As discussed before, the data nodes are stored on disk while the directory and super nodes are kept in memory for both the DC-Tree and DyTree. Since the data nodes represent the facts used as input, the disk space usage for the trees will always be the same. Therefore, to analyze the space usage efficiency, we study the comparison of memory usage only. This memory usage is the effect of directory and super nodes present in a DC-Tree or DyTree which are the result of nodes splitting.

The results for the comparison of memory usage are presented in figure 5.7. The results show that the difference in memory usage of DC-Tree and DyTree is neither much important when the input data set is SSB (figure 5.7e) nor in case of syn-dns40 (5.7f).

5.1.2 Query Response Time

In data warehousing environment, the data retrieval is performed through OLAP queries. Therefore, the assessment of our proposed solution for query response time of these OLAP queries is essential. As discussed earlier, a group-by query can be regarded as a set of point queries, therefore, we discuss the results only for point and range queries in the following. The results are obtained by executing the queries of the queries sets described in section 2.2 on DC-Tree and DyTree materializing different aggregates for 1, 2 up to 10 million facts. The query response time is recorded as the average query response time of all the queries present in a query set.

The queried trees are constructed using the parameters discussed in section 5.1.1.

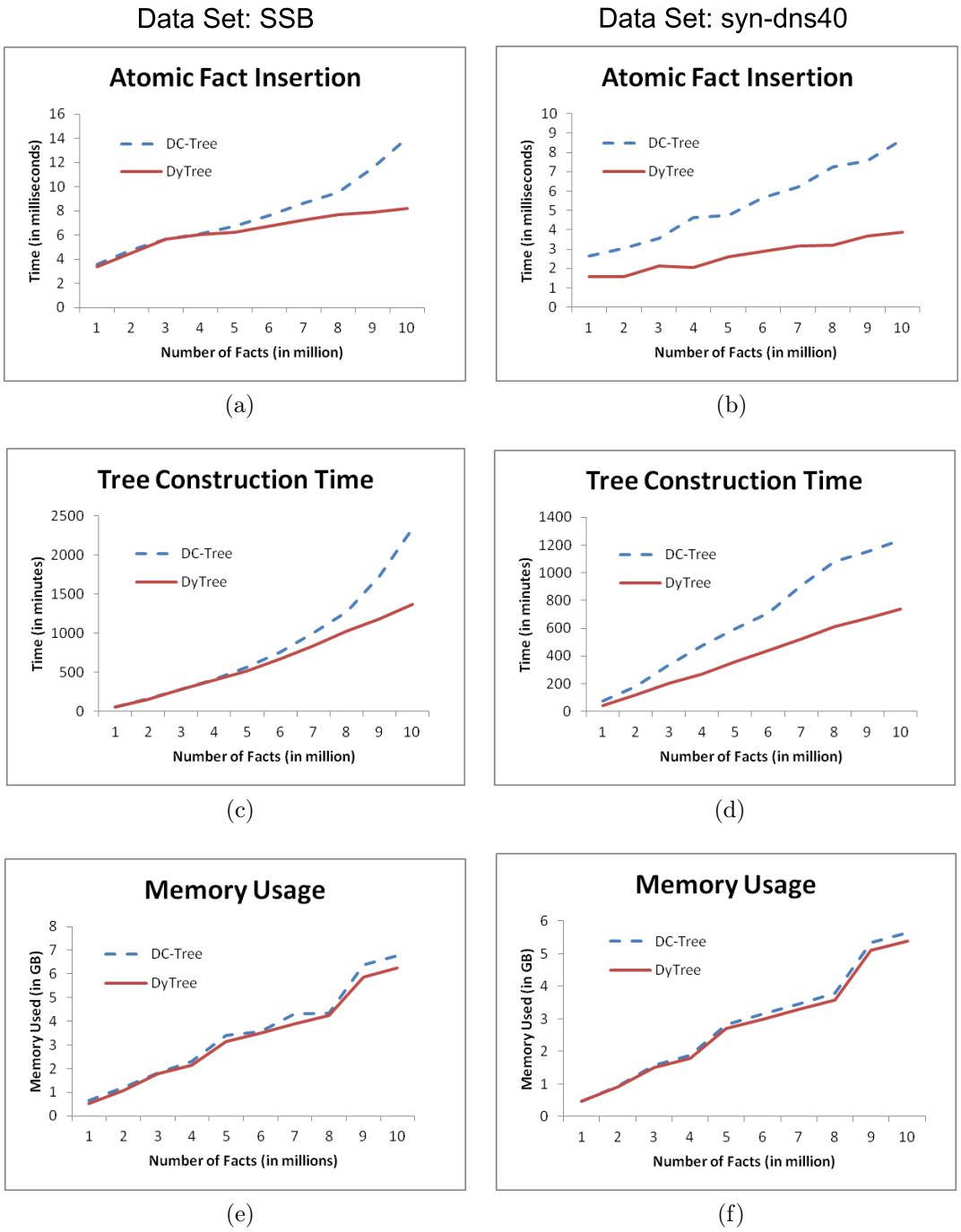


Figure 5.7: Comparison of, single data record insertion time on (a) SSB (# 1 in table 5.2) (b) syn-dns40 (# 13 in table 5.2); tree construction time on (c) SSB (d) syn-dns40; memory usage on (e) SSB (f) syn-dns40

Point Queries Figure 5.8 shows the comparison of query response time of point queries on DC-Tree and DyTree constructed using two different data sets i.e. SSB (figure 5.8a) and syn-dns40 (5.8b).

We observe that the query response time of DyTree is less than the DC-Tree's. The difference in the performance of both solutions increases with the increase in the size of the partially materialized data cubes represented by DC-Tree and DyTree. This difference is due to our strategy of favoring the grouping of temporally closed values together in the same nodes. We recall that our point queries do not necessarily lie in the lowest level hyper-plane but could lie in any of the hyper-planes of the HHMDS. Therefore, these point queries may also involve aggregation and hence the grouping of temporally closed values together in a node plays an important role in the betterment of query response time of a point query on DyTree.

As the super nodes are larger capacity nodes and index a larger number of nodes sequentially under itself, its retrieval performance is not as good as that of a directory node. Since our optimized splitting algorithm produces a smaller number of super nodes, it is also a contributing factor to this difference. Moreover, as the size of the input data set grows, the number and size of the super nodes in DC-Tree also grow. Hence the difference is increased with the size of data set.

We observe that the difference is much more important when the input data set is based on the low cardinality dimension tables (figure 5.8b). This is due to the lower successful splitting probability in case of low cardinality dimension tables, which produces a larger number of super nodes and the difference in the query response time of the two cubing techniques becomes more important.

Range Queries The results obtained for query response time of range queries on SSB and syn-dns40 are presented in figures 5.8c and 5.8d respectively. In both these cases as well, we witness the better performance of DyTree than the DC-Tree's. The explanation of this improvement is similar to the explanation of the improvement in case of point queries. However, to have a better understanding of the different contributing factors, we present, in figure 5.9, the results of query response time of range queries in two different cases: when the range of values is defined only on one (1) ordered dimension "Time" (figure 5.9a and 5.9b), (2) non-ordered dimension "X" (figure 5.9c and 5.9d).

Figure 5.9a and 5.9b show that the query response time of DyTree is always better than the DC-Tree's and the difference in the response time of the two techniques

increases with the size of the queried tree. This difference is mainly due to our grouping strategy of keeping temporally closed values together.

On the other hand, the results presented in 5.9c and 5.9d show that initially the DC-Tree performed better but with the increase in the size of the trees, DyTree starts showing better performance. This phenomenon is due to the fact that initially there would be no need to create super nodes while with increase in the size of input data set, the super nodes emerge and their sizes grow. As the DyTree avoids the creation of super nodes by providing an improved split algorithm, it shows better performance than that of the DC-Tree with increasing size of the input data set.

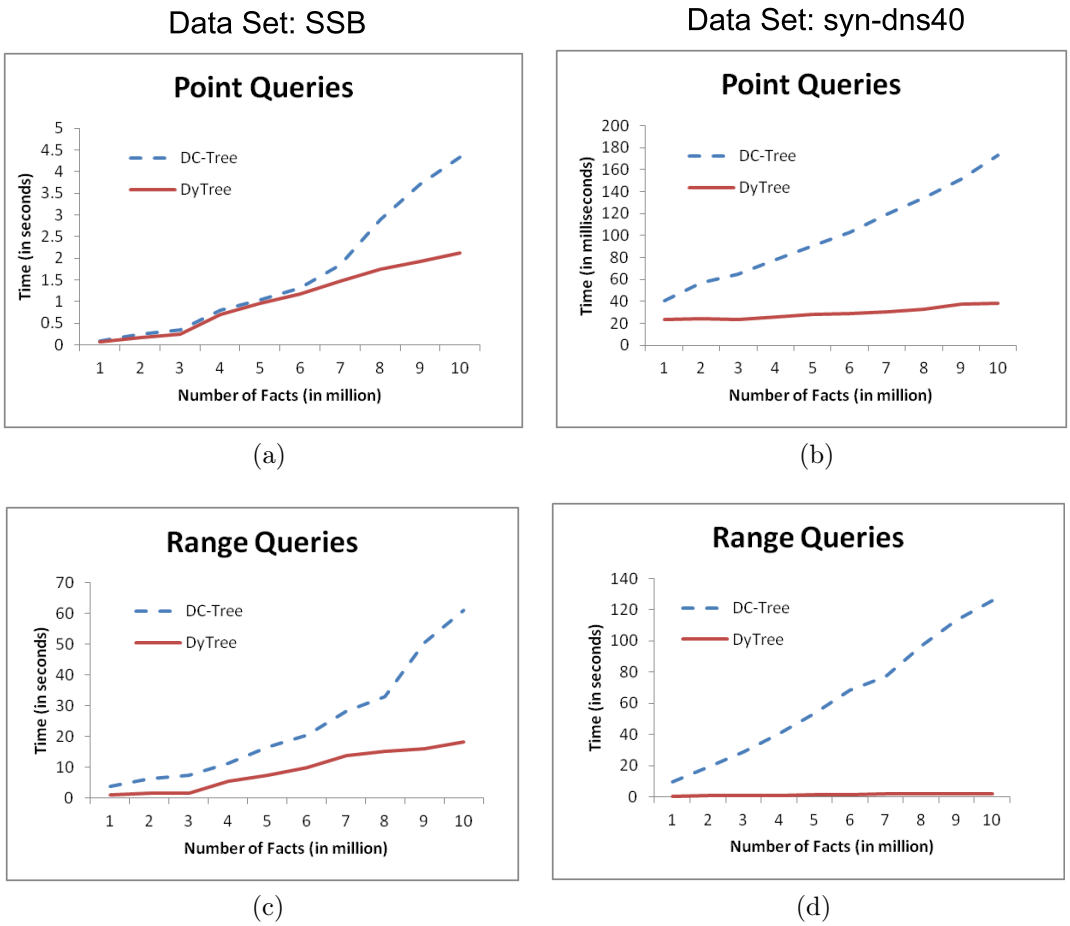


Figure 5.8: Comparison of query response time of (a) point queries using SSB (# 1 in table 5.2), (b) point queries using syn-dns40 (# 13 in table 5.2), (c) range queries using SSB, (d) range queries using syn-dns40

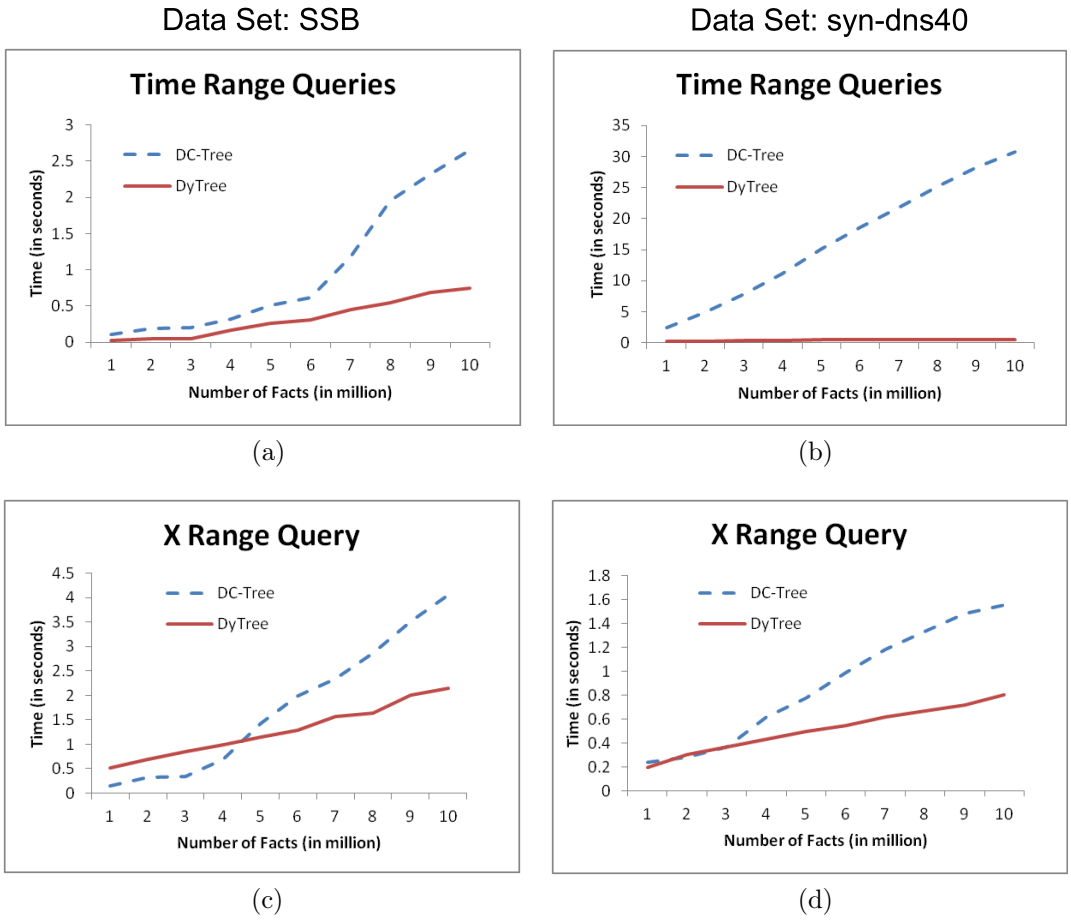


Figure 5.9: Comparison of queries response time of range queries when the range is defined on only one (a, b) temporal dimension and (c, d) non-temporal dimension

5.2 Effect of Varying Algorithm Input Parameters

In this section we see the effect of varying the values of algorithm input parameters i.e. overlap limit and directory nodes capacity on the performance of the DyTree.

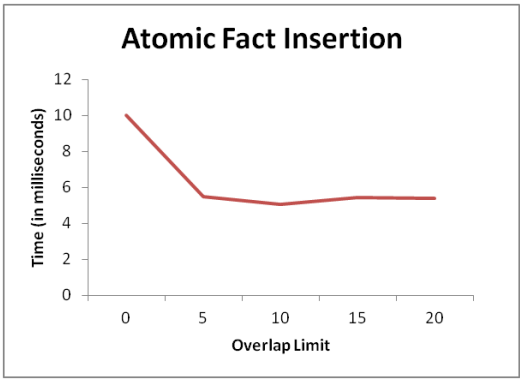
5.2.1 Effect of Varying Overlap Limit

As discussed earlier, overlap limit is an input parameter for the construction of a DyTree. This parameter allows internal nodes to share the same region data space and increases the probability of finding a suitable split. To understand the effect of this parameter on the performance of DyTree, we perform an experimental study by varying the overlap limit of a DyTree and discuss some of the obtained experimental results in the following. These experiments are conducted using SSB data set and the directory node capacity is fixed at 15.

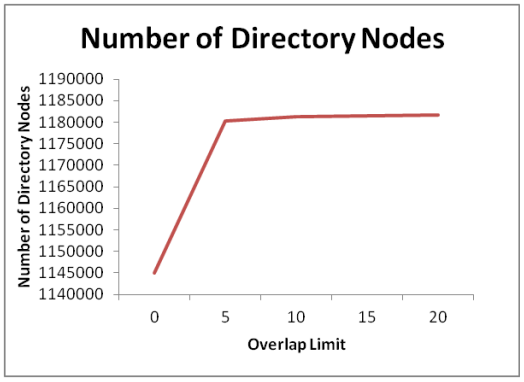
Atomic Fact Insertion Time As evident from figure 5.10a, the atomic insertion time of a single fact decreases rapidly by increasing the overlap limit from 0 to 5. This phenomenon is observed because allowing an overlap among two nodes increases the probability of finding a suitable split and this suitable split can be found easily in early iterations which reduces the average insertion time of a single fact. Further increasing the overlap limit also decreases the insertion time but very slightly. This can be justified that the overlap limit of 5 already produces the most of the possible optimization in insertion time.

The difference can also be explained by a very significant increase in number of directory (figure 5.10b) while decrease in number of super (figure 5.10c) nodes. We observe that initially both the number of directory and super nodes change rapidly by increasing the overlap limit from 0 to 5 but then become quite stable for further increase. This effect is exactly the same as on the atomic fact insertion insertion time. In fact, the decrease in number of super nodes, which sequentially index quite a large number of nodes, reduce the number of required consultations to locate a node to accommodate the incoming fact which consequently reduces the insertion time.

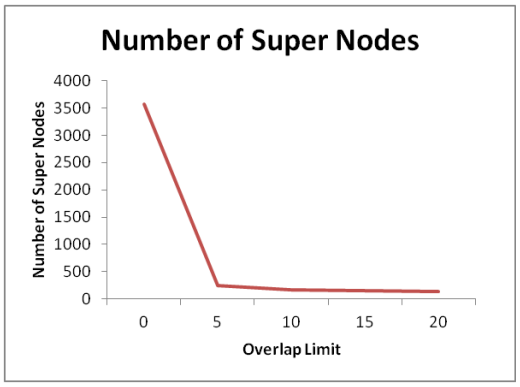
Query Response Time The effect of variation of overlap limit on query response time of a DyTree is presented in figure 5.10. We witness that the effect on the query response time of both point (figure 5.10d) and range (figure 5.10e) queries is negative i.e. the query response time increases with increase in overlap. This effect is due to the fact that allowing an overlap allows two nodes to share the same data space and



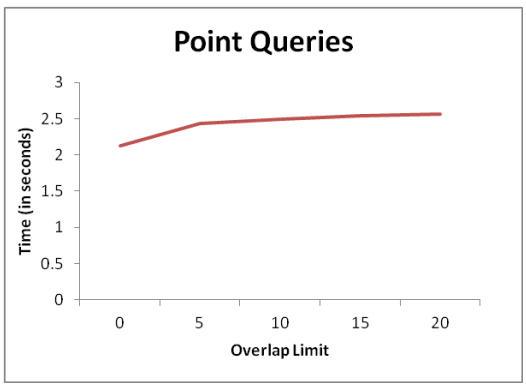
(a)



(b)



(c)

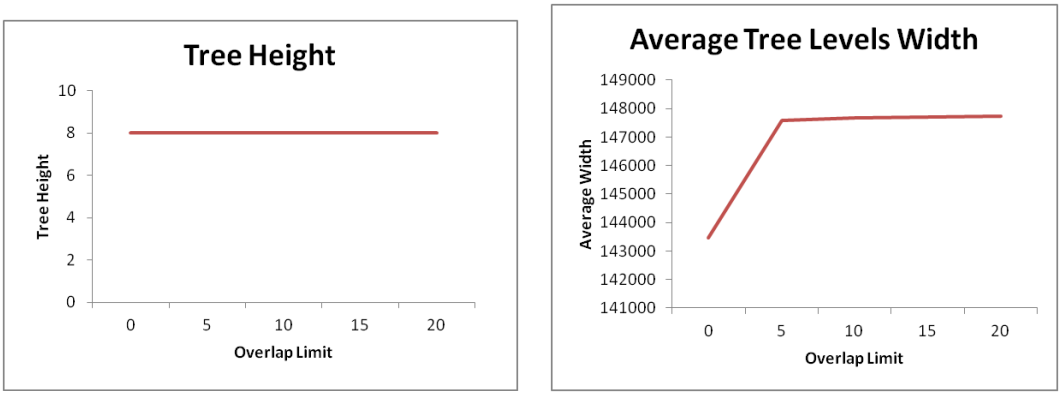


(d)

(e)

Figure 5.10: Effect of varying the overlap limit on, (a) insertion time of a single fact; number of (b) directory nodes (c) super nodes; query response time of (d) point queries and (e) range queries. The experiments are conducted using SSB data set (# 1 in table 5.2)

when queried, a query needs to look for the possible answer along multiple paths of the tree.



(a) (b)

Figure 5.11: Effect of varying the overlap limit on (a) tree height and (b) average tree levels width

The figure 5.11 showing the tree height and average tree levels width of the DyTree can also be used to explain the reason of better querying performance of the DyTree. We observe that the tree height is not affected (see figure 5.11a) while the average tree levels width is increased by increasing the overlap limit (as shown in figure 5.11b). The higher value of average tree levels width means that greater number of entries are sequentially indexed under the nodes at same levels. As our querying algorithm requires verifying querying criteria against all the entries indexed under any visited internal node, higher value of average tree levels width means that greater number of consultations required. This, in effect, causes the query response time to increase.

From the above discussion on the experimental results of the performance of DyTree obtained by varying the overlap limit, we can conclude that allowing an overlap among the nodes may decrease the insertion time, but at the same time causes the query response time of a DyTree to grow.

5.2.2 Effect of Varying Directory Nodes Capacity

A higher value of directory node capacity allows a larger number of nodes to be indexed under a directory node and minimizes the need of a node split. The experimental results to study the effect of the variation of this parameter on the performance of DyTree are discussed below. The overlap limit for this experimental study is fixed at 0.

Atomic Fact Insertion Time Increasing the directory node capacity from 15, initially decreases the atomic insertion time of an individual fact as shown in figure 5.12a, but if we keep increasing the directory node capacity, the insertion time start to rise again. The initial decrease in insertion time is due to the larger capacity of directory nodes, which avoids the need of nodes splitting: the costliest part of the insertion process. However, if we keep increasing the directory nodes capacity, they would sequentially hold a larger number of pointers to other nodes. This sequential indexing of pointers would need more time to find a suitable node in DyTree to accommodate a newly incoming fact and hence increases the insertion time.

Figure 5.12 can also be used to explain this effect. In fact, a rapid decrease in the number of super nodes initially reduces the atomic fact insertion time, but as the capacity of directory node keeps increasing, the directory nodes themselves start sequentially indexing a larger number of nodes and hence the significant increase in the insertion time is visible.

Query Response Time The effect of the variation of directory nodes capacity is summarized in figure 5.12. The query response time of both point (figure 5.12e) and range (figure 5.12f) queries initially decreases and then starts rising again with directory nodes capacity.

The directory nodes capacity of 15 with an overlap limit of 0 has a large number of directory and super nodes. Initially with the increase in directory nodes capacity, the query response time falls because the number of directory and super nodes decreases and the size of directory nodes remain reasonable. This makes the querying the DyTree easier and efficient. But with further increase in directory nodes capacity, the directory nodes start growing which sequentially hold the pointers to a larger number of other nodes. This makes the querying algorithm to look into a larger number of nodes pointed to by a node, and in turn, decreases the querying efficiency of DyTree for both point and range queries.

Memory Usage The memory usage of a DyTree is found to increase linearly with increase in directory nodes capacity (see figure 5.12b). We witness such an effect because the larger capacity directory nodes occupy more space in the memory. But as they would not be filled to their full capacity (see figure 5.13a and 5.13b), a relatively larger amount of memory space is wasted.

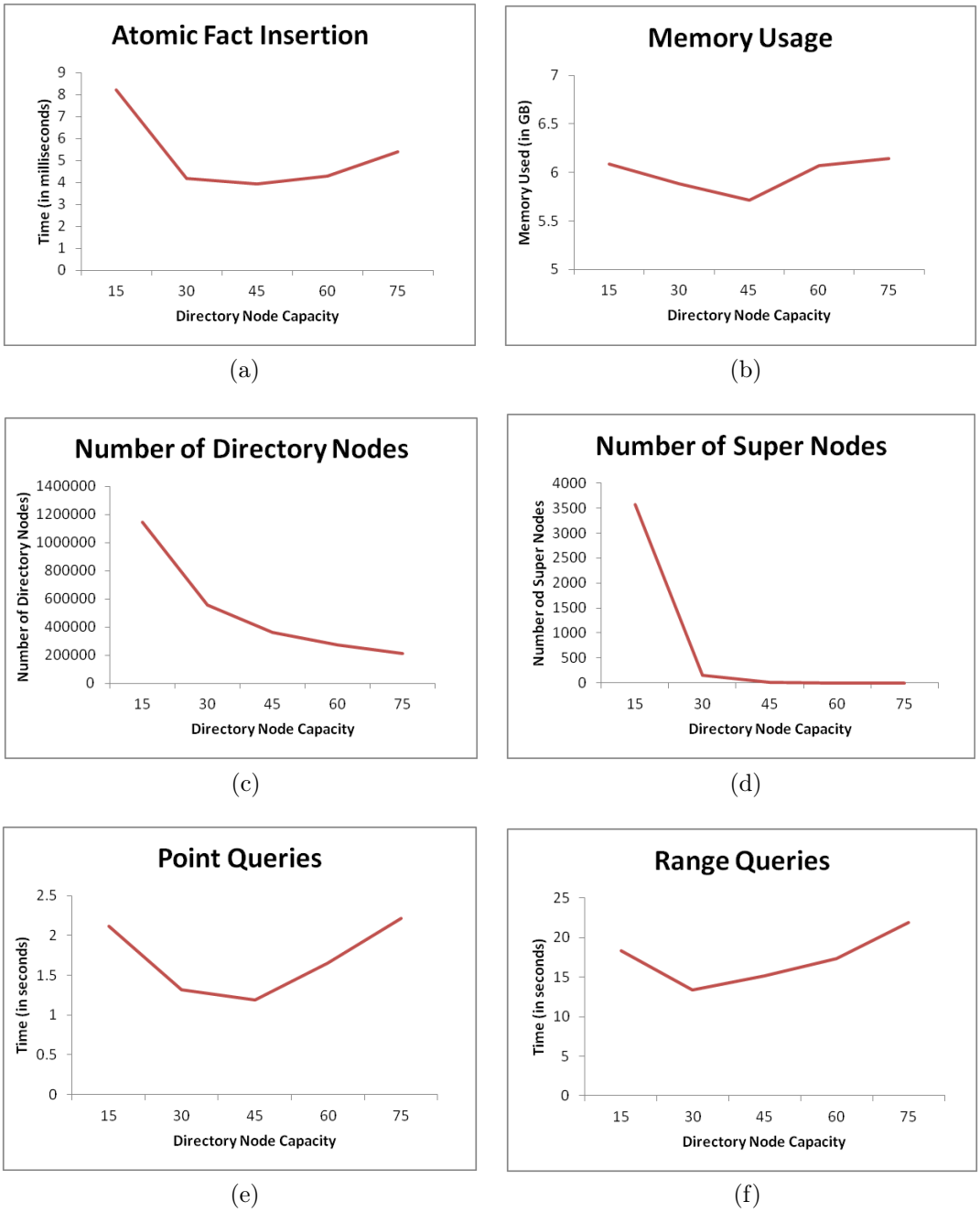


Figure 5.12: Effect of varying the directory node capacity on, (a) insertion time of a single fact; (b) memory usage; query response time of (c)point queries, (d) range queries. The experiments are conducted using SSB data set (# 1 in table 5.2).

The discussion on above results shows that the directory nodes capacity has an important effect on the performance of a DyTree. Neither a very small nor a very big value for the directory nodes capacity is suitable for an efficient DyTree.

5.3 Scaling in High Dimensional Data Space

For the results discussed above, we used a synthetic data set with a 10 dimensional schema and an SSB data set which is based on a four dimensional schema. Both of these data sets are based on low dimensional schema while in reality, a data warehouse may involve high dimensional data. This high number of dimensions could severely affect the performance of an indexing or cubing solution [Berchtold 1996]. Therefore, in order to asses this affect on the performance of DyTree, we discuss some experimental results obtained by varying the number of dimension of input data sets. We use syn-d10, syn-d15, ..., syn-d30 (# 6 - 10) presented in table 5.2 as the input data sets for this experimental study. The presented results are obtained on the DyTree materializing 10,000,000 facts.

The directory node capacity for these experiments is fixed at 35 while the overlap limit is 0.

Atomic Fact Insertion Time The effect of dimension scaling on atomic insertion time of a single record is depicted in figure 5.14a. The figure shows that the insertion time of a single fact is slightly affected by the increase in number of dimensions. The insertion time for a single record on a data set with 30-dimensional schema and 10 million tuples is found to be 5 milliseconds approximately, which is reasonable for a dynamic data warehousing environment.

Query Response Time The effect of dimension scaling on the query response time of a DyTree is visible in the figure 5.14. Figure 5.14c shows the effect on query response time of point queries while in figure 5.14d, the effect on the query response time of range queries is presented. The figure show that the query response time increases very slightly in case of point queries while in case of range queries, this effect is more prominent.

We recall that the range of values to be queried are defined for all the dimensions in our range queries. Therefore in case of a 10-dimensional schema we have ranges defined on 10 different dimensions while in a 30-dimensional schema we have the ranges defined on 30 dimensions. This means that a range query on a data set with high dimensional schema will have to verify more conditions than that on a data set

with low dimensional schema. This causes the query response time of range queries to increase with the increase in number of dimensions. This increase, however, is linear which is justified and acceptable.

Memory Usage The memory usage of a DyTree, as evident from the figure 5.14b, increases linearly with the increase in number of dimensions. This increase is easily understandable because for a high dimensional data, the MBS of directory and super nodes need to store more information which, in turn, utilizes more memory.

From the above discussion, we can conclude that the DyTree's performance does not degrade with increase in number of dimensions.

5.4 Effect of Varying the Data Sets Density

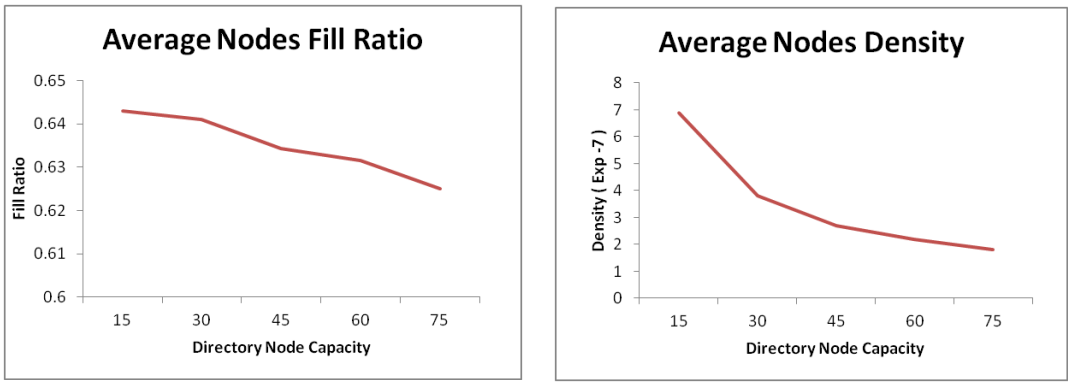
The data sets in real-world can be very different based on their density depending upon the nature of the applications they are being used in. Therefore studying the effect of variable density on the performance of an indexing solution is important. For this purpose, we discuss the effect of using dense data sets on our result in this section. We use the data sets with variable densities (i.e. syn-dns20, syn-dns20, ..., syn-dns60 presented in table 5.2) and discuss the obtained results.

The directory node capacity for these experiments is fixed at 35 while the overlap limit is 0.

Atomic Fact Insertion Time The effect of variation in fact table's density on atomic fact insertion time is depicted in figure 5.15a. The figure shows that the insertion time slightly increases with the increase in the density of fact table. This is due to the fact that increasing density reduces the chances of finding a suitable split which causes the nodes to split more often and as the splitting algorithm is the most expensive part of insertion algorithm, it causes the fact insertion time to grow.

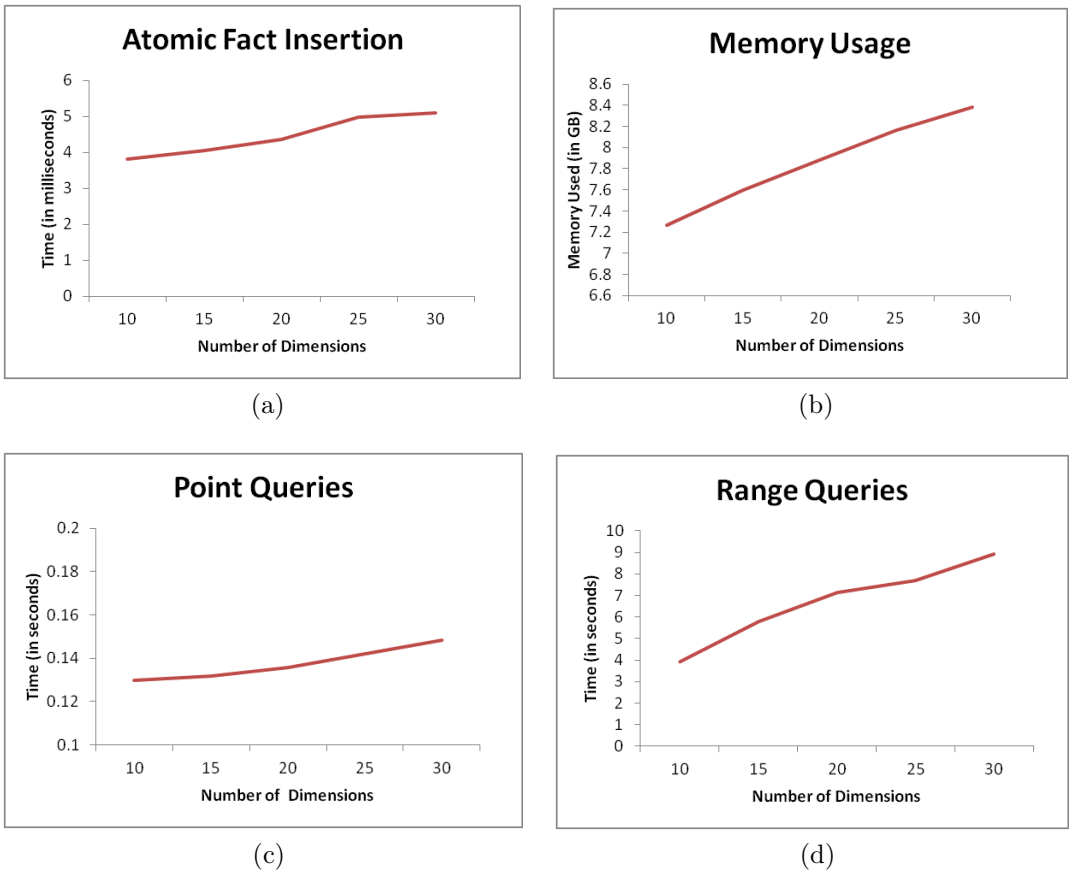
Query Response Time Figure 5.15 presents the effect of variation in density of a fact table on the query response time of the DyTree. Figure 5.15b shows the effect on query response time of point queries while in figure 5.15c, the effect on the query response time of range queries is presented.

We see a slight increase in the query response time of both types of queries. This increase is apparently due to the greater number of queries created as a result of the more splits.



(a) (b)

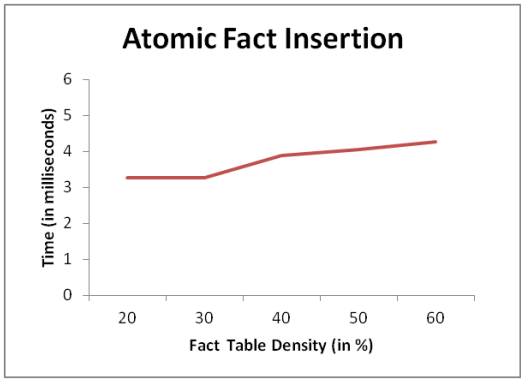
Figure 5.13: Effect of varying the directory node capacity on (a) average nodes fill ratio and (b) average nodes density



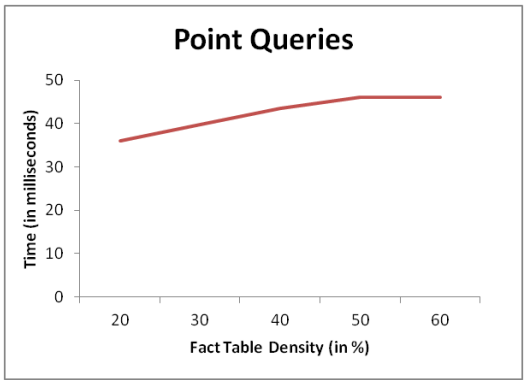
(a) (b)

(c) (d)

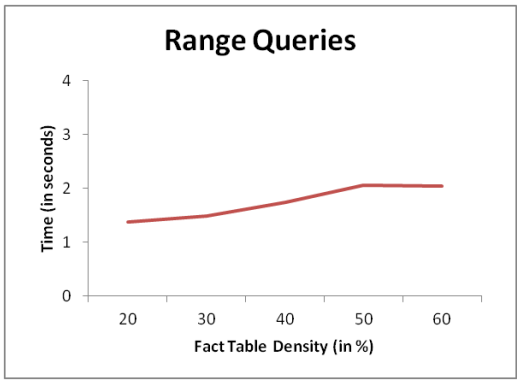
Figure 5.14: Effect of dimension scaling on, (a) atomic insertion of a fact; (b) memory usage; query response time of (c) point and (d) range queries. The experiments are conducted using data sets #6-10 described in table 5.2.



(a)



(b)



(c)

Figure 5.15: Effect of variation in fact table's density on, (a) atomic insertion of a fact, and query response time of (b) point (c) range queries; The experiments are conducted using data sets #11-15 described in table 5.2.

From the above discussion, we can conclude that the DyTree's performance changes slightly with the variation in the input fact tables' density but does not degrade considerably.

5.5 Effect of Delayed Insertion

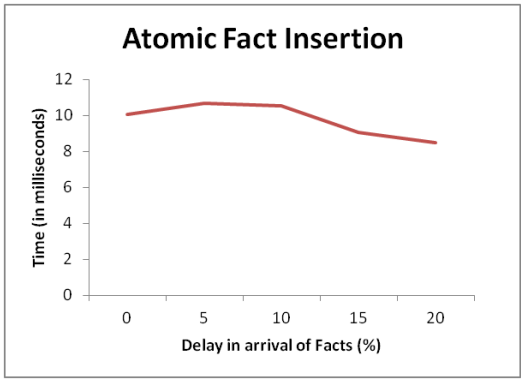
As previously discussed, the facts in a real-time data warehouse do not always arrive in chronological time order. Some of the facts may experience some delay due to the malfunctioning of some source system or due to some network problem. Since the DyTree exploits the temporal order among the facts and favors the grouping of temporally close values together in the tree nodes, we would like to understand if this delayed or out of order insertion of some of the facts affects the performance of DyTree or not. For this purpose, the DyTree is constructed using five different data sets with 10 million facts each. All the facts in the first data set follow chronological time order. In second data set, 5% of the total facts arrive with a delay, in third data set 10% of facts, in fourth 15% and in fifth data set 20 % of the facts arrive with a random delay.

The directory nodes capacity in these experiments is fixed at 15 while the overlap limit is 0.

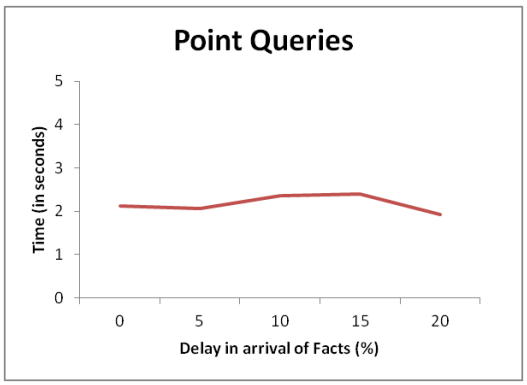
Atomic Fact Insertion Time Figure 5.16a shows that how the insertion time of a single fact is affected by the delayed insertion of some of the facts. We observe that this delay has only a slight effect on the atomic insertion of a fact in DyTree.

Query Response Time Figures 5.16b and 5.16c prove that the delayed insertion of some of the facts has only a slight random effect on the query response time of a DyTree and does not deteriorate the query performance of DyTree.

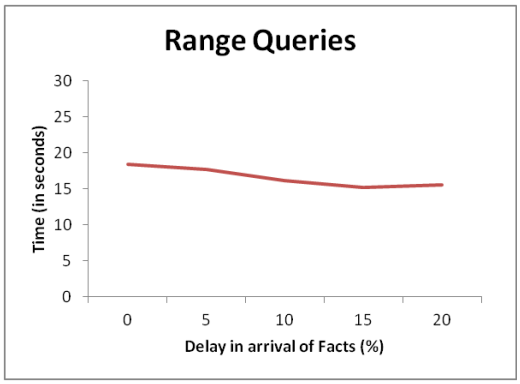
The above results show that our strategy of keeping temporally closed values together in nodes is not much affected by the delayed insertion and as a result the performance of DyTree does not experience any considerable deterioration.



(a)



(b)



(c)

Figure 5.16: Effect of delayed insertion on, (a) atomic insertion of a fact, and query response time of (b) point (c) range queries. The experiments are conducted using data sets #1-5 described in table 5.2.

6 The Prototype

After explaining the performance evaluation workflow and experimental results, we now describe our prototype developed for testing and evaluation. In this prototype we have implemented the proposed algorithms which let us not only see them working but also compare their performance and efficiency with existing solutions. For this purpose, we have also implemented the algorithms proposed in [Ester 2000] under the name of DC-Tree.

The development of the prototype is carried out on 64-bit windows platform. Microsoft Visual C# 2010 is used as the main programming language.

This prototype provides a complete set of functions and features ranging from data generation to data analysis. The data generation features are provided to generate various synthetic data sets while cube construction and data analysis features allow us

to study the behavior and compare the performance of our proposed algorithms. We give the details of the these functions and features of our prototype in the following.

6.1 Data Warehouse Schema Building and Usage

Every database or data warehouse is built over a schema. Therefore, it is mandatory for every software solution concerning these applications to be able to generate or use an existing schema. Our prototype provides an easy to use graphical user interface (figure 5.17) to build a new data warehouse schema from the scratch. The GUI allows the user to provide the names of the dimensions and their nature of being ordered (by checking the "Temporal Dimension" checkbox) or non-ordered. For each dimension, user provides the number and names of levels in each dimension hierarchy. Each level has an associated table whose schema is also defined through the GUI. The tables schema defining interface lets the user to define the names, data types and nature (i.e. primary key, foreign key or value) of the columns in the table. The generated schema can either be directly used as schema for the input data sets or be exported to an XML file for later use.

The prototype also provides the feature to use an existing schema stored in an XML file. This XML file could either be the one exported through our prototype or a manually created file that is built on the well defined specifications. The input XML file must define the participating dimensions, hierarchical levels in each dimensions and the structure of the tables for each hierarchical level.

6.2 Data Set Generation

Our prototype is capable of generating different dimension and fact tables respecting the input data warehouse schema (figure 5.18). The features to generate different data sets having different densities and following different data distributions are also provided. For a fact table generation, density or number of tuples is provided as input. A fact table with uniformly distributed data can be generated automatically while data with biased distribution needs to be generated manually by generating small clusters of data and then combining them to get a large data set having a biased distribution or clustered data.

6.3 Queries Set Generation

OLAP queries (e.g. point, range and group-by) can be generated by grace of queries generation feature. A point query is a query over single multidimensional data point

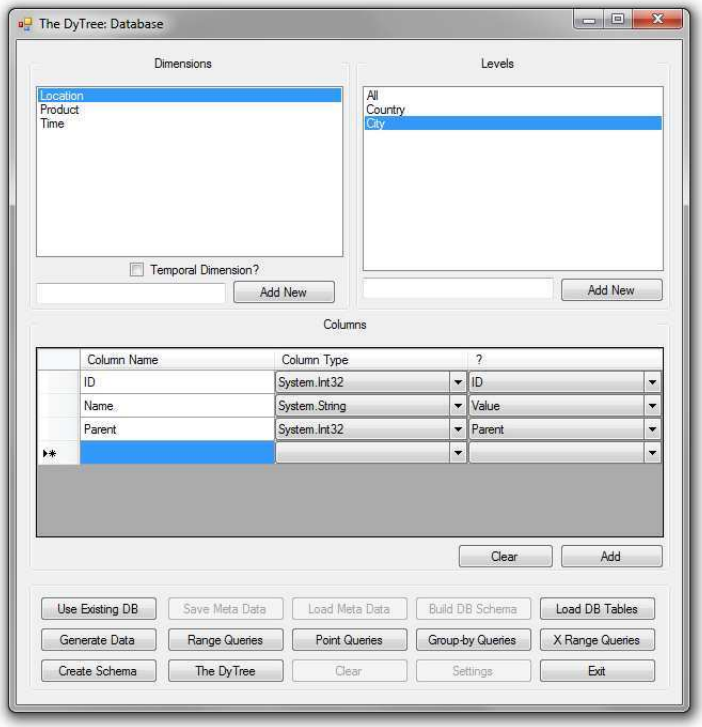


Figure 5.17: Schema definition

that could be either a point in a lowest level plane (i.e. a tuple of the fact table) or a point in some higher level plane in hybrid hierarchical multidimensional data space (HHMDS). A range query involves aggregation over different ranges of values from domain sets of different dimensions. This could be understood as a query over set of multidimensional points in HHMDS. A group-by query on the other hand groups different multidimensional points according to some attribute.

Number of queries to generate and the type of queries serve as the inputs. The range of values for each dimension that would participate in queries generation can also be selected. The queries are generated randomly and they may involve aggregation of the measures for different ranges and at different levels of hierarchy.

The schema, data set and queries set can also be generate automatically in one go by providing input parameters shown in figure 5.20. This utility lets us save time and generate completely random schema and data sets under specified constraints.

6.4 Tree Construction and Visualization

Both DC-Tree and DyTree can be constructed using "directory node capacity" and "overlap limit" as input parameters. Data could be inserted either from a comma separated (.csv) file or through the graphical user interface. The input from a .csv

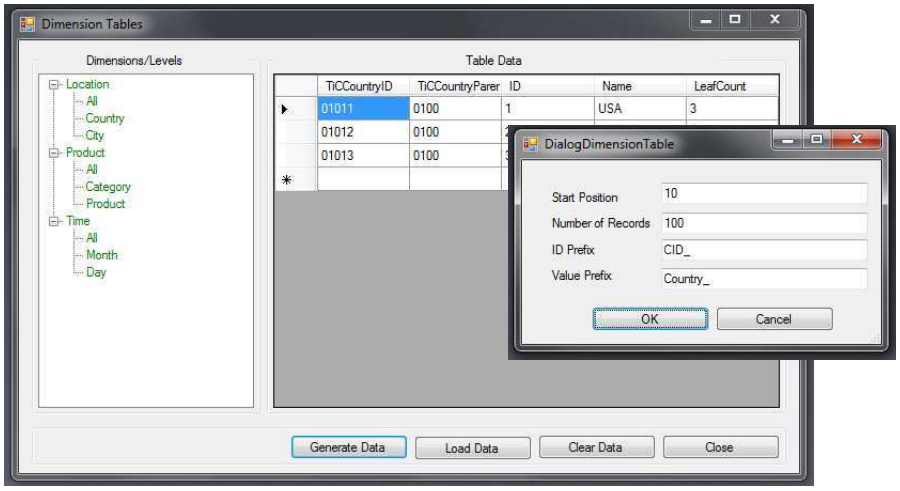


Figure 5.18: Loading or automatic generation of dimension tables data

file is important for running the long experiments using stored data sets while the GUI based loading helps understanding the working of insert and split algorithm, discussed in previous chapter. While loading the data in trees, the interface shows the progress through a progress bar.

A graphical representation of the state of the trees (as shown in figure 5.19) is provided and different statistics like tree construction time, tree depth, nodes density, disk space usage etc. can also be consulted. We provide the feature to save the trees in a binary file for future use as well as in a text file that could be used for offline detailed study of the trees.

6.5 Querying

The prototype allows all three types of queries over both DC-Tree and Dy-Tree. Like data loading, queries could also be executed either using stored queries sets or through graphical user interface. For the execution of every queries set, we also record the execution time and number of nodes visited in order to answer the queries. The graphical user interface, as shown in figure 5.21, provides an easy-to-use interface for multidimensional analysis of the data stored in the DC-Tree or the DyTree.

6.6 Data and Views Visualization

Our prototype offers a graphical user interface (see figure 5.22) to visualize the data in a three dimensional data space. This tool is helpful in viewing the distribution and density of data. If the data lies in a higher (more than 3) dimensional data space, any three of the dimensions could be selected to visualize the data.

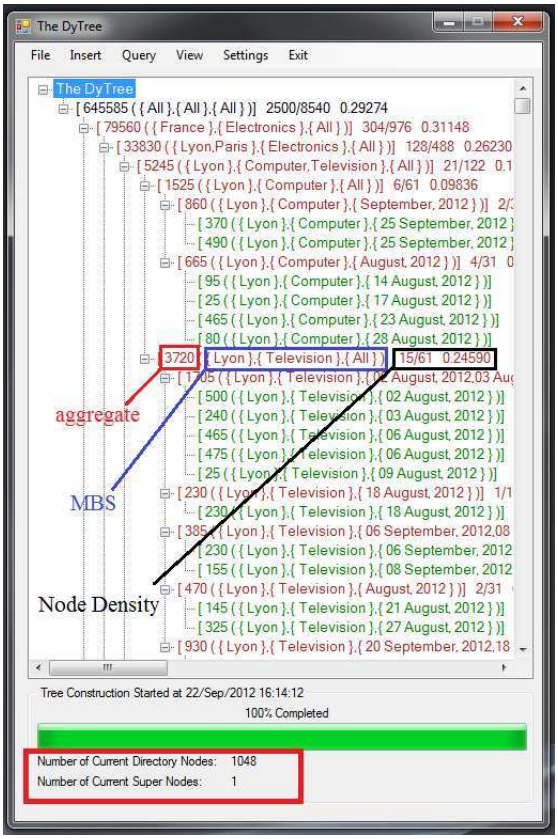


Figure 5.19: The tree construction and visualization interface

This tool also lets us the visualize the trees' nodes, representing the materialized views, in the data space. For this purpose, we can select a node we are interested to visualize from the GUI presented in figure 5.19 and see its contents in the data space. Consequently, we can identify the dense or sparse data nodes and regions in the trees and the data space. This helps us compare different materialized views, characterize them and eventually optimize them.

6.7 Running a Set of Experiments

We have also developed a special feature to run complete set of experiments. Data and queries sets are input to this feature, which constructs the trees and executes the queries at different states of the tree (see figure 5.23). Metrics concerning the construction and queries are recorded continuously throughout the process. These recorded metrics then serve as the input for comparison and trees analysis. This feature helps us to launch the time consuming heavy experiments automatically, without the need of intermittent human interventions which may be difficult at times. The

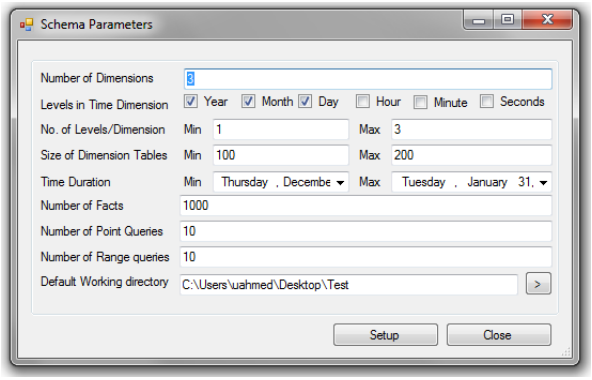


Figure 5.20: Automatic schema, data set and queries set generation

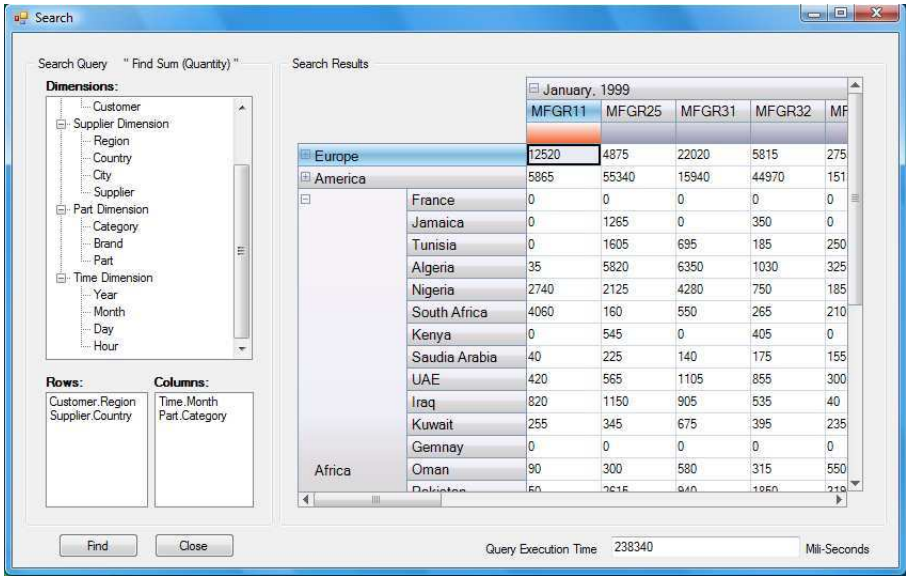


Figure 5.21: Querying a DyTree

logs of test runs are also recorded which help in spotting the cause of anomaly or error, if any, during the process.

7 Conclusion

We used a methodology to evaluate the performance of our proposal. For this purpose, a framework encompassing various evaluation criteria has been proposed. The framework guides us about the important input data sets features that should be considered as well as the metrics we should be looking for to characterize the performance of the DyTree/DC-Tree.

The obtained experimental results show that the DyTree outperforms the DC-Tree in fact insertion speed and query response time while exhibits similar memory

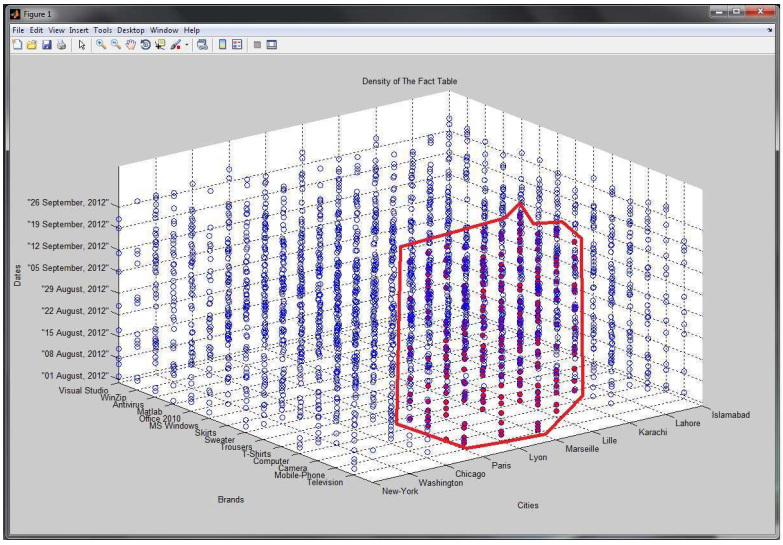


Figure 5.22: Interface to visualize data and tree nodes of a DyTree

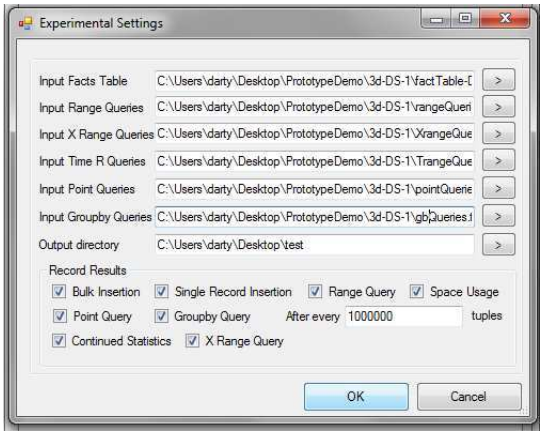


Figure 5.23: Parameters to start automatic experimentation

utilization efficiency. The DyTree scales well with increase in number of dimensions, i.e. its performance does not degrade with increasing the number of dimensions. The delayed insertion of some of the facts, which is quite common in real-time data warehouses, does not severely affect its performance either. The results obtained using very sparse ($density(S) = 10^{-7}$, see table 5.2) and dense (up to $density(S) = 0.6$, see table 5.2) sets show that the DyTree is viable for various types of applications, irrespective of the densities of the data sets they generate. The experiments conducted by varying the values of the algorithm input parameters let us understand the behavior of the DyTree as well as serve to experimentally optimize and tune its performance.

The prototype allows many features for testing and simulation of DyTree and its comparison with the DC-Tree. The experimental evaluation workflow is implemented.

Our prototype supports data visualization for a 3-dimensional data space and any 3 dimensions of a higher dimensional data can be selected to visualize data and tree nodes. As discussed, the current version of the prototype supports the automatic (parametrized) generation of uniformly distributed data only while data with biased distributions could be generated manually.

Chapter 6

General Conclusion

Chapter Outline

1 Contribution Summary

In this thesis, we addressed the issue of partial cube materialization in a dynamic data warehousing environment. We proposed the concept of Hierarchical Hybrid Multidimensional Data Space (HHMDS) that constitutes of both ordered and non-ordered hierarchical dimensions. We argued that a dynamic data warehouse operates in such a data space and introduced a multidimensional data model providing a useful abstraction of data objects lying in this data space. We proposed a data grouping structure, called MBS, that groups the data objects lying in the data space and proposed algebraic relations, operators and metrics allowing manipulate these objects. The proposed operators allow us to summarize or detail the data at different levels of granularity.

Using the proposed algebra, we put forward a data partitioning strategy and algorithms to store the MBS with their associated aggregate values, representing the indexed materialized sections of cuboids. The MBS are stored in a tree based data structure called the DyTree. The DyTree is built dynamically, i.e. through atomic incremental maintenance. The DyTree simultaneously stores and indexes detailed and aggregated data, therefore it is an indexing and cubing structure at same time. As not whole data cube is pre-computed, it is regarded as partially materialized data cube in which the views selection and optimization is based on the proposed metrics and relations. We have proposed efficient insert, split and querying algorithms using our metrics. The DyTree is based on data partitioning strategy so it naturally avoids indexing large data space while the splitting algorithm is designed to minimize the overlap among data partitions. This minimization of overlap, in turn, helps improving the insertion and querying performance the DyTree.

We put forward a methodology to compare the performance of the DyTree with an existing solution and to analyze its performance and behavior in different scenarios. We designed a workflow for this purpose and outlined important input parameters and output metrics allowing us to understand and characterize the performance of our solution. The extensive experiments suite is used first to compare the performance of the DyTree with that of the DC-Tree using different data sets and then to understand the effect of changing scenarios. The obtained experimental results show that the DyTree outperforms the DC-Tree in all cases, except the memory usage where it shows similar performance. The DyTree performance does not degrade by increasing the number of dimensions. The performance is not much affected by changing the density or the insertion order of the facts. The experimental study to understand the

behavior of the tree is useful to explain the performance of the solution as well as to optimize it.

We implemented our experimental evaluation workflow in a prototype which lets us evaluate the efficiency and effectiveness of the DyTree. The prototype provides interesting features including data/queries sets generation, trees construction, experimental evaluation and simulation.

2 Discussion

After having summarized our contribution, we discuss some strong points and limitations of our solution in this section.

2.1 Strong Points

We highlight and discuss some of the strong points of our contribution under this heading.

Distinction of Ordered and Non-ordered Dimension One of the strong for our solution is that our data space constitutes of both ordered and non-ordered dimensions. The members of all the dimensions are treated in a similar manner and their insertion order is not important. It is only at the time of metrics calculation that the ordered dimensions are treated distinctly than those which are not. The non-ordering of dimension members gives us the advantage of dynamic maintenance of data space while considering the natural order among the members of the ordered dimensions in metrics facilitates the data manipulation (e.g. data indexing, partitioning, range/group-by etc.). These advantages are also verified through our experimental study.

Creation of Dense MBS Since the DyTree's construction is based on MBS which are data grouping structure, it falls under the multidimensional data indexing techniques based on data partitioning. Therefore, it inherently avoids indexing large dead spaces. Apart from this, the MBS are created through hierarchical splitting triggered by the number of pointers held by a node. Therefore, if the facts are clustered in a data space, we will have more MBS for the clustered regions and these MBS will be dense. The optimization of MBS is achieved by the help of the proposed metrics.

2.2 Limitations

The proposed solution has some limitations and we discuss some of these below.

Schema Evolution and Versioning As evident from the discussion through out this dissertation, our research work lies at the intersection real-time and temporal data warehouses. However, we consider only one aspect of temporal data warehouses that is the addition of new data in dimension tables. Other types of evolution supported in temporal data warehouses that could be useful in the concerned applications, are not discussed in this research work. For example, the movement of a sensor would require the update of the existing sensors location in the dimension table.

Data Quality In traditional data warehouses, data coming from source systems can be stored at intermediate operational data stores where it can be checked for quality and consistency using other existing data items or master data. In our case, however, we consider that data coming from source systems is directly integrated into the DyTree without any quality or consistency checks. This might incur consistency problems in the stored data and needs to be addressed using either some efficient intermediate layer which performs quality checks without causing much delay in the insertion of facts or by some on-the-fly quality mechanism which could be integrated in the insertion algorithm.

Selection of Views Creation of materialized views (i.e. MBS) and their selection in our solution is guided by the metrics, eventual order among dimension members and the insertion order of facts. Therefore, we do not have complete control on the selection of views to materialize and at any time instance, it is not possible to deduce which views are materialized and stored in the DyTree without complete manual simulation of the scenario.

3 Enhancements and Extensions

The DyTree and our research work still has some room for improvement and needs more efforts. Among the improvements we can foresee, some are short term that we are willing to attack in near future while the others can be termed as long term which need more in depth analysis and research efforts.

3.1 Current Perspectives

In this section we discuss the current perspectives that can be addressed in near future.

3.1.1 Theoretical Cost Model

In this research work, we have evaluated the performance of the DyTree using extensive experimental study but a theoretical cost model is still lacking. A theoretical cost model will help us to determine the time and space complexity of our algorithms and establish the cost of their best, average and worst case scenarios. Theoretical results can be compared with the experimental ones which would consequently allow us to better understand, tune and optimize the performance of the algorithms. We would like to provide correctness, completeness proofs of our algorithms. Theoretical determination of optimized values of input parameters, $DNCAP$ and $OVLAP$, also needs more work.

3.1.2 Experiments with Real-World Data Sets

Though the experimental methodology and data sets used for the experimental evaluation of DyTree were carefully designed and a benchmark (SSB) was also used for the purpose, yet we believe that our solution requires more tests. The SSB is based on synthetic data set and was designed for traditional data warehouses while the synthetic data sets hardly simulate a real-world scenario. Therefore, we believe that the experimentation with a real-world data set would be more credible and convincing and would like to that in near future.

3.1.3 The Prototype Enhancement

We would also like to upgrade our prototype, which for the moment allows automatic generation of uniformly distributed data only. We would like to able to generate data sets following different mathematical distributions which will then be used for the experimental evaluation. We are also working to upgrade the data and tree visualization feature of our prototype which currently supports visualizing only detailed data in an ordered representation of 3-dimensional data space. We would like to make our prototype more interactive and robust.

3.1.4 Towards a Benchmark

The methodology employed to evaluate our experiments considers various scenarios that are important for partial data cubing or indexing technique in data warehouses. We outlined interesting input and output parameters for the purpose. So for we used it to experimentally evaluate our own solution and the DC-Tree and it was accordingly adapted but in near future we would like to generalize it to evaluate the effectiveness and efficiency of selected views by various techniques which would serve as a benchmark in this context.

3.2 Future Directions

This section provides a discussion on some of the possible future directions for the continuation of this research work.

3.2.1 Support For Irregular Dimension Hierarchies

Our current multidimensional data model considers dimensions with simple hierarchies only while in literature, many research works arguing the importance of complex or irregular hierarchies [Malinowski 2004, Mansmann 2006] in an OLAP system can be found. To make our model deal with such dimension hierarchies needs more research efforts but will make the model more practical and usable.

3.2.2 Considering other Types of Ordering

In this research work, we proposed the idea of considering the nature of dimensions for data modeling and creating/manipulating data partitions. We used only one ordered dimension i.e. time and favored the grouping of temporally closed values together and the utility of such an approach in improving the performance of temporal range queries is shown through experiments. This idea can be applied to other ordered dimensions as well to exploit other types of closeness such spatial proximity. In that case spatially closed object can be grouped together in same views to enhance the performance of spatial range queries. Similarly, the indexing of semantically closed values together in same node can be favored in case of textual data.

3.2.3 Favoring the Appearance of Certain MBS

As discussed earlier, in current DyTree, the materialized views represented through MBS are constructed at run time and depend upon the proposed metrics and the

insertion order of the facts. We believe that the performance of its algorithms can be improved by favoring the appearance of certain MBS on the basis of following criteria.

Frequently Used Queries Unlike the views selection strategy employed in the DyTree, many research works related to partial cube materialization provide solutions on the basis of prior knowledge of used queries. Since the creation of views are aimed at improving the query's performance only, such a strategy of selection of views is a reasonable one. However, as the DyTree operates in a dynamic environment, the queries to be used are not known a priori. Still, it is possible to have the knowledge of type of some frequently used queries. If there is any kind of such knowledge, it would be useful for the selection of views. Therefore, we believe that using this knowledge, in addition to the current DyTree's strategy, to favor the materialization of particular views is a possible and interesting future work.

Correlation among Dimensions As we know, dimension in a data warehouse can sometimes be correlated. This correlation information has been used by some researchers for efficient cube computation (e.g. [Yu-cai 2004, Feng 2004b]). Such correlation information (if any) can easily be used while constructing and optimizing the MBS stored in DyTree to group correlated values together and improve the query performance. For example, in a sales application if it can be deduced that the sales of warm clothes in Europe is much more significant than in Africa, then the early appearance of warm clothes in an MBS with Europe can be favored. Similarly, certain sparse MBS can also be made to appear if it could be deduced from the existing pattern that the upcoming facts would come under it. For example, if it is known in an intelligent building application that at certain location (say L1) the change in temperature would be recorded more often than the others, then the node holding an MBS with location L1 can be created even if it is sparse. Next temperature readings coming from L1, will directly be indexed under that node without needing any split unless it overflows. This will help reducing the splitting of existing MBS and consequently improve the insertion efficiency of the DyTree.

4 Final Words

The importance of dynamic data warehouses is increasing day by day. These data warehouses require the real-time integration of new data in warehouse and necessitate the incremental update of aggregates and materialized views maintained for the

purpose of reducing the analysis latency. Our research work on this issue leads to a formal data model supporting dynamic updates and a partial cubing structure with efficient algorithms to incrementally build and query it. Many interesting research directions follow from this work and can be considered in future.

Nomenclature

a_i	i^{th} aggregate value associate to a DyTree node
$agg(M_\Delta)$	aggregate value associated to an MBS M_Δ
$M\, contains\, N$	relation contains among the two MBS
$density(S)$	density of fact table or data set
$density(M)$	density of an MBS
$leafRatio(node)$	Leaf ratio of a DyTree node
D_i	i^{th} dimension
$\delta(E_t,\, F_t)$	difference between the two time intervals i.e. E_t and F_t
$DNCAP$	numerical value representing the capacity of a directory node (Constant)
$domain(l)$	domain set of a level l
$\varrho_{l_i^v}(m)$	drill-down on a member m of dimension from its current level i to level v
$\sigma_{l_i^u}(m)$	drill-up on a member of dimension from its current level i to level u
E_i	i^{th} edge of an MBS
F_i	i^{th} edge of an MBS
$entrySet$	set of pointers of a DyTree node holding the pointers to the child entries
$fillratio(node)$	fill ratio of a DyTree node

f	an aggregate function such as SUM, AVERAGE etc.
$\mathcal{H}$	a graph representing a dimension hierarchy
$\mathcal{I}$	a graph representing the instance of a hierarchical dimension
$Interval(E_t)$	Interval represented by the time dimension edge E_t of an MBS
l_i^j	j^{th} hierachical level of i^{th} dimension.
ALL_i	the top most level of i^{th} dimension
all_i	the unique member of the top most level of i^{th} dimension
$level(m)$	level of a member m of a dimension space
L_i	set of the levels of the i^{th} dimension
M	a mniimum bounding space (MBS) m
N_i, O_i, R_i	MBS used in examples
M_Δ	an MBS M constructed over a set of points Δ
$m(p)$	measure value associated to a point p
$node$	a DyTree node
$p' \circledcirc p$	relation covers among two multidimensional points
$P_1 \geq_P P_2$	ordere relation among the hyper-planes P_1 and P_2
$l_i^u \uparrow l_i^v$	order relation among two levels of a dimension hierarchy
$m \Uparrow n$	order relation among the members of a dimension D_i
$OVLAP$	numerical value representing the overlap limit among two nodes (Constant)
$ovlapArea(M, N)$	the overlapped/shared area between two MBS M and N
$M\, overlaps\, N$	relation representing an MBS M overlaps another MBS N
P	hyper-plane
$< l_1^{h_1}, l_2^{h_2}, ..., l_n^{h_n} >$	Notation for a hyper-plane

$p(x_1, x_2, ..., x_n)$	a multidimensional point p
Δ	set of multidimensional points
$size(entries)$	current number of pointers to child entries
S	data space
S_{D_i}	dimension space of the i^{th} dimension
$\lfloor_P(M)$	translate-down on an MBS M from its current hyper-plane to the hyper-plane P
$\lceil_P(M)$	translate-up on an MBS M from its current hyper-plane to the hyper-plane P
$volume(M)$	volume of an MBS M
$volume(S)$	volume of data space
BJI	bitmap join index
DDC	dynamic data cube
DOLUS	dynamic online updating solution
HHMDS	hierarchical hybrid multidimensional data space
HOBI	hierarchially organized bitmap index
MBS	minimum bounding space
MDS	minimum describing space
OLAP	online analytical processing
QoD	quality of data
QoS	quality of service
RiTE	right-time ETL
ROLAP	relational online analytical processing

References

[Agarwal 2011] Deepak Agarwal and Bee-Chung Chen. *Latent OLAP: data cubes over latent variables.* In Proceedings of the 2011 international conference on Management of data, SIGMOD '11, pages 877–888, 2011.

[Aligon 2011] Julien Aligon, Patrick Marcel and Elsa Negre. *Résumés et interrogations de logs de requêtes OLAP.* In 11ème Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances, EGC'11, pages 239–250, Brest, France, 2011.

[Amo 2000] Sandra De Amo and Mirian Halfeld Ferrari Alves. *Efficient Maintenance of Temporal Data Warehouses.* In Proceedings of the 2000 International Symposium on Database Engineering & Applications, IDEAS '00, pages 188–196, Washington, DC, USA, 2000. IEEE Computer Society.

[Aouiche 2005] Kamel Aouiche, Jérôme Darmont, Omar Boussaïd and Fadila Bentayeb. *Automatic selection of bitmap join indexes in data warehouses.* In Proceedings of the 7th international conference on Data Warehousing and Knowledge Discovery, DaWaK'05, pages 64–73, 2005.

[Aouiche 2009] Kamel Aouiche and Jérôme Darmont. *Data mining-based materialized view and index selection in data warehouses.* Journal of Intelligent and Information Systems, vol. 33, no. 1, pages 65–93, August 2009.

[Bai 2006] Yun Bai, Yanyan Guo, Xiaofeng Meng, Tao Wan and Karine Zeitouni. *Efficient dynamic traffic navigation with hierarchical aggregation tree.* In Proceedings of the 8th Asia-Pacific Web conference on Frontiers of WWW Research and Development, APWeb'06, pages 751–758, Berlin, Heidelberg, 2006. Springer-Verlag.

[Baralis 1997] Elena Baralis, Stefano Paraboschi and Ernest Teniente. *Materialized Views Selection in a Multidimensional Database*. In Proceedings of 23rd International Conference on Very Large Data Bases, VLDB '97, pages 156–165, 1997.

[Bayer 1972] Rudolf Bayer and Edward M. McCreight. *Organization and maintenance of large ordered indexes*. Acta Informatica, vol. 1, pages 173–189, 1972.

[Beckmann 1990] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider and Bernhard Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*. SIGMOD Record, vol. 19, no. 2, pages 322–331, 1990.

[Bellahsene 2002] Zohra Bellahsene. *Schema evolution in data warehouses*. Knowledge and Information Systems, vol. 4, no. 3, pages 283–304, July 2002.

[Bellatreche 2007] Ladjel Bellatreche, Rokia Missaoui, Hamid Necir and Habiba Drias. *Selection and Pruning Algorithms for Bitmap Index Selection Problem Using Data Mining*. In Il Song, Johann Eder and Tho Nguyen, editeurs, Data Warehousing and Knowledge Discovery, volume 4654 of *Lecture Notes in Computer Science*, pages 221–230. Springer Berlin / Heidelberg, 2007.

[Bellatreche 2010] Ladjel Bellatreche and Kamel Boukhalfa. *Yet another algorithms for selecting bitmap join indexes*. In Proceedings of the 12th international conference on Data warehousing and knowledge discovery, DaWaK'10, pages 105–116, Berlin, Heidelberg, 2010. Springer-Verlag.

[Bentley 1975] Jon Louis Bentley. *Multidimensional binary search trees used for associative searching*. Commun. ACM, vol. 18, no. 9, pages 509–517, September 1975.

[Berchtold 1996] Stefan Berchtold, Daniel A. Keim and Hans-Peter Kriegel. *The X-tree : An Index Structure for High-Dimensional Data*. In Proceedings of 22nd International Conference on Very Large Data Bases, VLDB '96, pages 28–39, 1996.

[Beyer 1999] Kevin Beyer and Raghu Ramakrishnan. *Bottom-up computation of sparse and Iceberg CUBE*. SIGMOD Record, vol. 28, no. 2, pages 359–370, June 1999.

[Body 2002] Mathurin Body, Maryvonne Miquel, Yvan Bédard and Anne Tchounikine. *A multidimensional and multiversion structure for OLAP applications*. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, DOLAP '02, pages 1–6, New York, NY, USA, 2002. ACM.

[Böhm 2001] Christian Böhm, Stefan Berchtold and Daniel A. Keim. *Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases*. ACM Comput. Surv., vol. 33, no. 3, pages 322–373, September 2001.

[Bruckner 2002] Robert Bruckner, Beate List and Josef Schiefer. *Striving towards Near Real-Time Data Integration for Data Warehouses*. In Data Warehousing and Knowledge Discovery, volume 2454 of *Lecture Notes in Computer Science*, pages 173–182. Springer Berlin / Heidelberg, 2002.

[Castellanos 2010] Malu Castellanos, Umeshwar Dayal and Meichun Hsu. *Live Business Intelligence for the Real-Time Enterprise*. In From Active Data Management to Event-Based Systems and More, Lecture Notes in Computer Science. 2010.

[Chamoni 1999] Peter Chamoni and Steffen Stock. *Temporal Structures in Data Warehousing*. In Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery, DaWaK '99, pages 353–358, London, UK, UK, 1999. Springer-Verlag.

[Chan 1998] Chee-Yong Chan and Yannis E. Ioannidis. *Bitmap index design and evaluation*. SIGMOD Record, vol. 27, no. 2, pages 355–366, June 1998.

[Chen 2001] Jun Chen, Xin Zhang, Songting Chen, Andreas Koeller and Elke A. Rundensteiner. *DyDa: data warehouse maintenance in fully concurrent environments*. SIGMOD Rec., vol. 30, no. 2, pages 619–, May 2001.

[Chen 2009a] Changqing Chen. *Indexing of Multidimensional Discrete Data Spaces and Hybrid Extensions*. PhD thesis, Michigan State University, East Lansing, Michigan, USA, 2009.

[Chen 2009b] Changqing Chen, Sakti Pramanik, Qiang Zhu, Watve Alok and Gang Qian. *The C-ND Tree: a multidimensional index for hybrid continuous and non-ordered discrete data spaces*. In Proceedings of the 12th International

Conference on Extending Database Technology: Advances in Database Technology, EDBT '09, pages 462–471, 2009.

[Cheung 2001] David W. Cheung, Bo Zhou, Ben Kao, Hu Kan and Sau Dan Lee. *Towards the building of a dense-region-based OLAP system*. Data Knowledge and Engineering, vol. 36, no. 1, pages 1–27, January 2001.

[Chmiel 2009] Jan Chmiel, Tadeusz Morzy and Robert Wrembel. *HOBI: Hierarchically Organized Bitmap Index for Indexing Dimensional Data*. In Data Warehousing and Knowledge Discovery, Lecture Notes in Computer Science. 2009.

[Chmiel 2010] Jan Chmiel, Tadeusz Morzy and Robert Wrembel. *Time-HOBI: indexing dimension hierarchies by means of hierarchically organized bitmaps*. In Proceedings of the ACM 13th international workshop on Data warehousing and OLAP, DOLAP '10, pages 69–76, 2010.

[Dayal 1999] Umeshwar Dayal, Qiming Chen and Meichun Hsu. *Dynamic Data Warehousing*. In Mukesh Mohania and A Tjoa, editeurs, DataWarehousing and Knowledge Discovery, volume 1676 of *Lecture Notes in Computer Science*, pages 798–798. Springer Berlin / Heidelberg, 1999.

[Doka 2011] Katerina Doka, Dimitrios Tsoumakos and Nectarios Koziris. *Online querying of d-dimensional hierarchies*. J. Parallel Distributed Computing, vol. 71, no. 3, pages 424–437, March 2011.

[Ester 2000] Martin Ester, Jorn Kohlhammer and Hans-Peter Kriegel. *The DC-tree: A Fully Dynamic Index Structure for Data Warehouses*. In Proceedings of the 16th International Conference on Data Engineering (ICDE), pages 379–388, 2000.

[Faloutsos 1988] Christos N. Faloutsos. *Gray Codes for Partial Match and Range Queries*. IEEE Transactions on Software Engineering, vol. 14, no. 10, pages 1381–1393, oct 1988.

[Faloutsos 1989] C. Faloutsos and S. Roseman. *Fractals for secondary key retrieval*. In Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '89, pages 247–252, 1989.

[Favre 2007] Cécile Favre. *Évolution de schémas dans les entrepôts de données : mise à jour de hiérarchies de dimension pour la personnalisation des analyses*. PhD thesis, Université Lyon II, Lyon, France, 2007.

[Feng 2004a] Jianlin Feng, Qiong Fang and Hulin Ding. *PrefixCube: prefix-sharing condensed data cube*. In Proceedings of the 7th ACM international workshop on Data warehousing and OLAP, DOLAP '04, pages 38–47, 2004.

[Feng 2004b] Ying Feng, Divyakant Agrawal, Amr El Abbadi and Ahmed Metwally. *Range CUBE: Efficient Cube Computation by Exploiting Data Correlation*. In Proceedings of the 20th International Conference on Data Engineering, ICDE '04, pages 658–, 2004.

[Feng 2006] Yaokai Feng and Akifumi Makinouchi. *Ag-Tree: A Novel Structure for Range Queries in Data Warehouse Environments*. In Mong Li Lee, Kian-Lee Tan and Vilas Wuwongse, editeurs, Database Systems for Advanced Applications, volume 3882 of *Lecture Notes in Computer Science*, pages 498–512. Springer Berlin / Heidelberg, 2006.

[Feng 2011] MicYaokai Feng and Akifumi Makinouchi. *Ag+ tree: an Index Structure for Range-aggregation Queries in Data Warehousing Environment*. International Journal of Database Theory and Applications, vol. 4, no. 2, pages 51–54, 2011.

[Fenk 2000] Robert Fenk, Akihiko Kawakami, Volker Markl, Rudolf Bayer and Shunji Osaki. *Bulk Loading a Data Warehouse Built Upon a UB-Tree*. In Proceedings of the 2000 International Symposium on Database Engineering & Applications, IDEAS '00, pages 179–187, 2000.

[Finkel 1974] R. A. Finkel and J. L. Bentley. *Quad trees: a data structure for retrieval on composite keys*. Acta Informatica, vol. 4, pages 1–9, 1974.

[Foley 2005] Tim Foley and Jeremy Sugerman. *KD-tree acceleration structures for a GPU raytracer*. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '05, pages 15–22, 2005.

[Gaede 1998] Volker Gaede and Oliver Günther. *Multidimensional access methods*. ACM Computing Surveys, vol. 30, no. 2, pages 170–231, June 1998.

[Geffner 2000] Steven Geffner, Divyakant Agrawal and Amr El Abbadi. *The Dynamic Data Cube*. In Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '00, pages 237–253, London, UK, UK, 2000. Springer-Verlag.

[Golab 2009a] Lukasz Golab, Theodore Johnson, J. Spencer Seidel and Vladislav Shkapenyuk. *Stream warehousing with DataDepot*. In Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD '09, pages 847–854, 2009.

[Golab 2009b] Lukasz Golab, Theodore Johnson and Vladislav Shkapenyuk. *Scheduling Updates in a Real-Time Stream Warehouse*. In Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09, pages 1207–1210, 2009.

[Golfarelli 2009] Matteo Golfarelli and Stefano Rizzi. *A Survey on Temporal Data Warehousing*. International Journal of Data Warehousing and Mining, vol. 5, pages 1–17, 2009.

[Govindarajan 2002] Sathish Govindarajan, Pankaj K. Agarwal and Lars Arge. *CRB-Tree: An Efficient Indexing Scheme for Range-Aggregate Queries*. In Proceedings of the 9th International Conference on Database Theory, ICDT '03, pages 143–157, 2002.

[Gray 1997] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow and Hamid Pirahesh. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*. Data Mining and Knowledge Discovery, vol. 1, 1997.

[Günther 1989] Oliver Günther. *The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases*. In Proceedings of the 5th International Conference on Data Engineering, ICDE '89, pages 598–605, 1989.

[Gupta 1995] Ashish Gupta, Inderpal S. Mumick and Kenneth A. Ross. *Adapting materialized views after redefinitions*. SIGMOD Records, vol. 24, no. 2, pages 211–222, May 1995.

[Gupta 1997a] Himanshu Gupta. *Selection of Views to Materialize in a Data Warehouse*. In Foto N. Afrati and Phokion G. Kolaitis, editeurs, Proceeding of the

6th International Conference on Database Theory, volume 1186 of *ICDT '97*, pages 98–112. Springer, 1997.

[Gupta 1997b] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman and Jeffrey D. Ullman. *Index Selection for OLAP*. In Proceedings of the Thirteenth International Conference on Data Engineering, ICDE '97, pages 208–219, 1997.

[Gupta 2011] Chetan Gupta, Umeshwar Dayal, Song Wang and Abhay Mehta. *Live BI: A Framework for Real Time Operations Management*. In Databases in Networked Information Systems, Lecture Notes in Computer Science. 2011.

[Guttman 1984] Antonin Guttman. *R-Trees: a dynamic index structure for spatial searching*. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data, SIGMOD '84, pages 47–57, 1984.

[Harinarayan 1996] Venky Harinarayan, Anand Rajaraman and Jeffrey D. Ullman. *Implementing Data Cubes Efficiently*. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD '96, pages 205–216. ACM Press, 1996.

[Hunter 1979] Gregory M. Hunter and Kenneth Steiglitz. *Operations on Images Using Quad Trees*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. PAMI-1, no. 2, pages 145 –153, april 1979.

[Hurtado 1999] Carlos A. Hurtado, Alberto O. Mendelzon and Alejandro A Vaisman. *Maintaining Data Cubes under Dimension Updates*. In Proceedings of the 15th International Conference on Data Engineering, ICDE '99, pages 346–, Washington, DC, USA, 1999. IEEE Computer Society.

[Johnson 1996] Theodore Johnson and Dennis Shasha. *Hierarchically Split Cube Forests for Decision Support: description and tuned design*. Rapport technique, Department of Computer Science, New York University, New York, USA, 1996.

[Jorg 2010] Thomas Jorg and Stefan Dessloch. *Near Real-Time Data Warehousing Using State-of-the-Art ETL Tools*. In Enabling Real-Time Business Intelligence, volume 41 of *Lecture Notes in Business Information Processing*, pages 100–117. Springer Berlin Heidelberg, 2010.

[Kamel 1994] Ibrahim Kamel and Christos Faloutsos. *Hilbert R-tree: An Improved R-tree using Fractals.* In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pages 500–509, 1994.

[Karakasidis 2005] Alexandros Karakasidis, Panos Vassiliadis and Evaggelia Pitoura. *ETL queues for active data warehousing.* In Proceedings of the 2nd international workshop on Information quality in information systems, IQIS '05, pages 28–39, 2005.

[Kim 2007] Namgyu Kim and Songchun Moon. *Concurrent View Maintenance Scheme for Soft Real-time Data Warehouse Systems.* Journal of Information Science and Engineering, pages 725–741, 2007.

[Kimball 1996] Ralph Kimball. The data warehouse toolkit: practical techniques for building dimensional data warehouses. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[Kotidis 1998] Yannis Kotidis and Nick Roussopoulos. *An alternative storage organization for ROLAP aggregate views based on cubetrees.* SIGMOD Record, vol. 27, no. 2, pages 249–258, June 1998.

[Koudas 2000] Nick Koudas. *Space efficient bitmap indexing.* In Proceedings of the ninth international conference on Information and knowledge management, CIKM '00, pages 194–201, 2000.

[Kuznetsov 2009] S. Kuznetsov and Yu. Kudryavtsev. *A mathematical model of the OLAP cubes.* Programming and Computer Software, vol. 35, pages 257–265, 2009.

[Lakshmanan 2002] Laks V. S. Lakshmanan, Jian Pei and Jiawei Han. *Quotient Cube: how to summarize the semantics of a data cube.* In VLDB '02: Proc. of the 28th Int. Conf. on Very Large Data Bases, pages 778–789. VLDB Endowment, 2002.

[Lakshmanan 2003] Laks V. S. Lakshmanan, Jian Pei and Yan Zhao. *QC-Trees: an efficient summary structure for semantic OLAP.* In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, SIGMOD '03, pages 64–75, 2003.

[Li 2004] Xiaolei Li, Jiawei Han and Hector Gonzalez. *High-dimensional OLAP: a minimal cubing approach*. In Proceedings of the 30th international conference on Very Large Data Bases - Volume 30, VLDB '04, pages 528–539, 2004.

[Malinowski 2004] Elzbieta Malinowski and Esteban Zimányi. *OLAP Hierarchies: A Conceptual Perspective*. In Anne Persson and Janis Stirna, editeurs, Advanced Information Systems Engineering, volume 3084 of *Lecture Notes in Computer Science*, pages 19–35. Springer Berlin / Heidelberg, 2004.

[Malinowski 2008] Elzbieta Malinowski and Esteban Zimányi. *A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models*. Data Knowledge & Engineering, vol. 64, no. 1, pages 101–133, January 2008.

[Mansmann 2006] Svetlana Mansmann and Marc H. Scholl. *Extending visual OLAP for handling irregular dimensional hierarchies*. In Proceedings of the 8th international conference on Data Warehousing and Knowledge Discovery, DaWaK'06, pages 95–105, Berlin, Heidelberg, 2006. Springer-Verlag.

[Mendelzon 2000] Alberto O. Mendelzon and Alejandro A. Vaisman. *Temporal Queries in OLAP*. In Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00, pages 242–253, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Morfonios 2006] Konstantinos Morfonios and Yannis Ioannidis. *CURE for cubes: cubing using a ROLAP engine*. In Proceedings of the 32nd international conference on Very large data bases, VLDB '06, pages 379–390, 2006.

[Morton 1966] G.M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. Rapport technique, IBM Ltd., Ottawa, Ontario, Canada, 1966.

[Namgyu 2007] Kim Namgyu and Moon Songchun. *Concurrent view maintenance scheme for soft real-time data warehouse systems*. Journal of information science and engineering, vol. 23, no. 3, pages 723–739, 2007. eng.

[Nguyen 2003] Tho Manh Nguyen and A Min Tjoa. *Zero-Latency data warehousing for hetrogeneous data sources and continuous data streams*. In Proceedings of the 5th International Conference on Information Integration and Web-based Applications Services, iiWAS '03, 2003.

[Nguyen 2005] Tho Manh Nguyen, Peter Brezany, A. Min Tjoa and Edgar Weippl. *Toward a Grid-Based Zero-Latency Data Warehousing Implementation for Continuous Data Streams Processing.* International Journal of Data Warehousing and Mining, IJDWM, vol. 1, no. 4, pages 22–55, 2005.

[Nguyen 2006] Tho Manh Nguyen and A Min Tjoa. *Zero-latency data warehousing (ZLDWH): the state-of-the-art and experimental implementation approaches.* In The 4th IEEE International Conference On Computer Sciences Researc; Innovation and Vision for the Future, RIVF '06, pages 167–176. IEEE, 2006.

[Nguyen 2007] Tho Manh Nguyen, Josef Schiefer and A. Min Tjoa. *ZELESSA: an enabler for real-time sensing, analysing and acting on continuous event streams.* International Journal of Business Intelligence and Data Mining, vol. 2, no. 1, pages 105–141, March 2007.

[O'Neil 1989] Patrick E. O'Neil. *Model 204 Architecture and Performance.* In Proceedings of the 2nd International Workshop on High Performance Transaction Systems, pages 40–59, 1989.

[O'Neil 1995] Patrick O'Neil and Goetz Graefe. *Multi-table joins through bitmapped join indices.* SIGMOD Record, vol. 24, no. 3, pages 8–11, September 1995.

[O'Neil 1997] Patrick O'Neil and Dallan Quass. *Improved query performance with variant indexes.* In Proceedings of the 1997 ACM SIGMOD international conference on Management of data, SIGMOD '97, pages 38–49, 1997.

[O'Neil 2009] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen and Stephen Revilak. In Raghunath Nambiar and Meikel Poess, editeurs, Performance Evaluation and Benchmarking, chapitre The Star Schema Benchmark and Augmented Fact Table Indexing, pages 237–252. Springer-Verlag, Berlin, Heidelberg, 2009.

[Orenstein 1982] Jack A. Orenstein. *Multidimensional Tries Used for Associative Searching.* Information Processing Letters, no. 4, pages 150–157, 1982.

[Orenstein 1984] Jack A. Orenstein and Tim H. Merrett. *A class of data structures for associative searching.* In Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, PODS '84, pages 181–190, 1984.

[Papadias 2001] Dimitris Papadias, Panos Kalnis, Jun Zhang and Yufei Tao. *Efficient OLAP Operations in Spatial Data Warehouses.* Proceedings of the 7th

International Symposium on Advances in Spatial and Temporal Databases, pages 443–459, 2001.

[Polyzotis 2007] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis andAlkis Simitsis and Nils-Erik Frantzell. *Supporting Streaming Updates in an Active Data Warehouse*. In Proceesdings of 23rd International Conference on Data Engineering, ICDE '07, 2007.

[Qian 2003] Gang Qian, Qiang Zhu, Qiang Xue and Sakti Pramanik. *The ND-tree: a dynamic indexing technique for multidimensional non-ordered discrete data spaces*. In Proceedings of the 29th international conference on Very large data bases - Volume 29, VLDB '03, pages 620–631, 2003.

[Qian 2006] Gang Qian, Qiang Zhu, Qiang Xue and Sakti Pramanik. *A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces*. ACM Transactions on Information Systems, vol. 24, no. 1, pages 79–110, January 2006.

[Ravat 2006] Franck Ravat, Olivier Teste and Gilles Zurfluh. *A multiversion-based multidimensional model*. In Proceedings of the 8th international conference on Data Warehousing and Knowledge Discovery, DaWaK'06, pages 65–74, Berlin, Heidelberg, 2006. Springer-Verlag.

[Riedewald 2000] Mirek Riedewald, Divyakant Agrawal, Amr El Abbadi and Renato Pajarola. *Space-Efficient Data Cubes for Dynamic Environments*. In Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery, DaWaK 2000, pages 24–33, London, UK, UK, 2000. Springer-Verlag.

[Rizzi 2006] Stefano Rizzi and Matteo Golfarelli. *What time is it in the data warehouse?* In Proceedings of the 8th international conference on Data Warehousing and Knowledge Discovery, DaWaK'06, pages 134–144, Berlin, Heidelberg, 2006. Springer-Verlag.

[Ross 1997] Kenneth A. Ross and Divesh Srivastava. *Fast Computation of Sparse Datacubes*. In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, pages 116–125, 1997.

[Rotem 2005] Doron Rotem, Kurt Stockinger and Kesheng Wu. *Optimizing candidate check costs for bitmap indices*. In Proceedings of the 14th ACM international conference on Information and knowledge management, CIKM '05, pages 648–655, New York, NY, USA, 2005. ACM.

[Roussopoulos 1985] Nick Roussopoulos and Daniel Leifker. *Direct spatial search on pictorial databases using packed R-trees*. SIGMOD Record, vol. 14, no. 4, pages 17–31, May 1985.

[Roussopoulos 1997] Nick Roussopoulos, Yannis Kotidis and Mema Roussopoulos. *Cubetree: organization of bulk and incremental updates on the data cube*. In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, SIGMOD '97, pages 89–99, NY, USA, 1997. ACM.

[Samet 1984] Hanan Samet and Robert E. Webber. *On Encoding Boundaries with Quadtrees*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 6, no. 3, pages 365–369, March 1984.

[Santos 2008] Ricardo Jorge Santos and Jorge Bernardino. *Real-time data warehouse loading methodology*. In Proceedings of the 12th International Database Engineering and Application Symposium, IDEAS '08, pages 49–58, New York, NY, USA, 2008. ACM.

[Santos 2009] Ricardo Jorge Santos and Jorge Bernardino. *Optimizing data warehouse loading procedures for enabling useful-time data warehousing*. In Proceedings of the 2009 International Database Engineering & Applications Symposium, IDEAS '09, pages 292–299, 2009.

[Sellis 1987] Timos K. Sellis, Nick Roussopoulos and Christos Faloutsos. *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects*. In Proceedings of the 13th International Conference on Very Large Data Bases, VLDB '87, pages 507–518, 1987.

[Simitsis 2009] Alkis Simitsis, Kevin Wilkinson, Malu Castellanos and Umeshwar Dayal. *QoX-driven ETL design: reducing the cost of ETL consulting engagements*. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, SIGMOD '09, pages 953–960, 2009.

[Sismanis 2002] Yannis Sismanis, Antonios Deligiannakis and Yannis Roussopoulos Nickand Kotidis. *Dwarf: shrinking the PetaCube*. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of data, SIGMOD '02, pages 464–475, New York, NY, USA, 2002. ACM.

[Sismanis 2003] Yannis Sismanis, Antonios Deligiannakis, Yannis Kotidis and Nick Roussopoulos. *Hierarchical Dwarfs for the rollup cube*. In Proceedings of the 6th ACM Intenrational workshop on Data Warehousing and OLAP, DOLAP '03, NY, USA, 2003.

[Theodoratos 1999] Dimitri Theodoratos and Timos K. Sellis. *Dynamic Data Warehouse Design*. In Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery, DaWaK '99, pages 1–10, London, UK, UK, 1999. Springer-Verlag.

[Thiele 2009] Maik Thiele, Andreas Bader and Wolfgang Lehner. *Multi-objective scheduling for real-time data warehouses*. Computer Science - Research and Development, vol. 24, pages 137–151, 2009.

[Thiele 2010] Maik Thiele and Wolfgang Lehner. *Evaluation of Load Scheduling Strategies for Real-Time Data Warehouse Environments*. In Enabling Real-Time Business Intelligence, volume 41 of *Lecture Notes in Business Information Processing*, pages 84–99. Springer Berlin Heidelberg, 2010.

[Thomsen 2008] Christian Thomsen, Torben Bach Pedersen and Wolfgang Lehner. *RiTE: Providing On-Demand Data for Right-Time Data Warehousing*. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE '08, pages 456–465, 2008.

[TPC 2011] TPC. *TPC Benchmarks*. `http://www.tpc.org/information/benchmarks.asp`, 2001 - 2011. Accessed: 03/16/2012.

[Vassilakopoulos 1993] M Vassilakopoulos, Y Manolopoulos and K Economou. *Overlapping quadtrees for the representation of similar images*. Image and Vision Computing, vol. 11, no. 5, pages 257 – 262, 1993.

[Vu 2009] Nguyen Hoang Vu and Vivekanand Gopalkrishnan. *Epsilon Equitable Partition: On Scheduling Data Loading and View Maintenance in Soft Real-time Data Warehouses*. In Sanjay Chawla, Kamalakar Karlapalem and Vikram

Pudi, editeurs, Proceedings of 15th International Conference on Management of Data, COMAD '09. Computer Society of India, 2009.

[Wang 2002] Wei Wang, Hongjun Lu, Jianlin Feng and Jeffrey Xu Yu. *Condensed Cube: An Efficient Approach to Reducing Data Cube Size*. In Proceedings of 18th International Conference on Data Engineering, ICDE '02, 2002.

[White 1996] David A. White and Ramesh Jain. *Similarity Indexing with the SS-tree*. In Proceedings of the 12th International Conference on Data Engineering, ICDE '96, pages 516–523, Washington, DC, USA, 1996. IEEE Computer Society.

[Wrembel 2006] Robert Wrembel and Tadeusz Morzy. *Managing and querying versions of multiversion data warehouse*. In Proceedings of the 10th international conference on Advances in Database Technology, EDBT'06, pages 1121–1124, Berlin, Heidelberg, 2006. Springer-Verlag.

[Wu 1998] Ming-Chuan Wu and Alejandro P. Buchmann. *Encoded Bitmap Indexing for Data Warehouses*. In Proceedings of the 14th International Conference on Data Engineering, ICDE '98, pages 220–230, 1998.

[Xi 2008] Jianqing Xi, Fuqiang Chen and Pingjian Zhang. *A New Bitmap Index and a New Data Cube Compression Technology*. In Proceedings of the 8th international conference on Computational Science and Its Applications, Part II, ICCSA '08, pages 1218–1228, 2008.

[Xiao 2009] Weiji Xiao and Jianqing Xi. *CCBitmaps: A Space-Time Efficient Index Structure for OLAP*. In Proceedings of the 5th International Conference on Advanced Data Mining and Applications, ADMA '09, pages 729–735, 2009.

[You 2006] Byeong-Seob You, Dong-Wook Lee, Sang-Hun Eo, Jae-Dong Lee and Hae-Young Bae. *Hybrid index for spatio-temporal OLAP operations*. pages 110–118, 2006.

[Yu-cai 2004] Feng Yu-cai, Chen Chang-qing, Feng Jian-lin and Xiang Long-gang. *Fast computation of sparse data cubes with constraints*. Wuhan University Journal of Natural Sciences, vol. 9, pages 167–172, 2004.

[Zhang 1999] Chuan Zhang and Jian Yang. *Genetic Algorithm for Materialized View Selection in Data Warehouse Environments.* In Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery, DaWaK '99, pages 116–125, 1999.

[Zhang 2008] Lei Zhang and Xiao-Guang Hong. *Dynamic On-Line Updating Solution for CURE Cubes.* In Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on, volume 5, pages 396 –400, oct. 2008.

[Zhou 2008] Kun Zhou, Qiming Hou, Rui Wang and Baining Guo. *Real-time KD-tree construction on graphics hardware.* ACM Transactions on Graphics, vol. 27, no. 5, pages 126:1–126:11, December 2008.

[Zuters 2011] Janis Zuters. *Near Real-Time Data Warehousing with Multi-stage Trickle and Flip.* In Perspectives in Business Informatics Research, volume 90 of *Lecture Notes in Business Information Processing*, pages 73–82. Springer Berlin Heidelberg, 2011.

Last Name : Ahmed **Defense Data** : 18/02/2013

First Name : Usman

Title : Dynamic cubing for hierarchical multidimensional data space

Nature : PhD **Order No.** : 2013ISAL0011

Ecole Doctorale : InfoMaths

Speciality : Computer Science

Abstract :

Data warehouses are being used in many applications since quite a long time. Traditionally, new data in these warehouses is loaded through offline bulk updates which implies that latest data is not always available for analysis. This, however, is not acceptable in many modern applications (such as intelligent building, smart grid etc.) that require the latest data for decision making. These modern applications necessitate real-time fast atomic integration of incoming facts in data warehouse. Moreover, the data defining the analysis dimensions, stored in dimension tables of these warehouses, also needs to be updated in real-time, in case of any change. In this thesis, such real-time data warehouses are defined as dynamic data warehouses. We propose a data model for these dynamic data warehouses and present the concept of Hierarchical Hybrid Multidimensional Data Space (HHMDS) which constitutes of both ordered and non-ordered hierarchical dimensions. The axes of the data space are non-ordered which help their dynamic evolution without any need of reordering. We define a data grouping structure, called Minimum Bounding Space (MBS), that helps efficient data partitioning of data in the space. Various operators, relations and metrics are defined which are used for the optimization of these data partitions and the analogies among classical OLAP concepts and the HHMDS are defined. We propose efficient algorithms to store summarized or detailed data, in form of MBS, in a tree structure called DyTree. Algorithms for OLAP queries over the DyTree are also detailed. The nodes of DyTree, holding MBS with associated aggregated measure values, represent materialized sections of cuboids and tree as a whole is a partially materialized and indexed data cube which is maintained using online atomic incremental updates. We propose a methodology to experimentally evaluate partial data cubing techniques and a prototype implementing this methodology is developed. The prototype lets us experimentally evaluate and simulate the structure and performance of the DyTree against other solutions. An extensive study is conducted using this prototype which shows that the DyTree is an efficient and effective partial data cubing solution for a dynamic data warehousing environment.

Keywords : OLAP, Partial Views Materialization, Multidimensional data indexing, Data Cube,

Real-Time Data Warehouse

Research Laboratory : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)

PhD Supervisors : Maryvonne Miquel, Anne Tchounikine

President of Jury :

Composition of Jury : Ladjel Bellatreche (Reviewer), Maryvonne Miquel (Co-supervisor),
Jean-Marc Petit (Examiner), Franck Ravat (Examiner),
Anne Tchounikine (Co-supervisor), Karine Zeitouni (Reviewer),
Estaban Zimányi (Examiner)

NOM : AHMED **Data de Soutenance** : 18/02/2013

Prénom : Usman

Intitulé : Dynamic cubing for hierarchical multidimensional data space

Nature : Doctorat **N° D'ordre** : 2013ISAL0011

Ecole Doctorale : InfoMaths

Spécialité : Informatique

Résumé :

De nombreuses applications décisionnelles reposent sur des entrepôts de données. Ces entrepôts permettent le stockage de données multidimensionnelles historisées qui sont ensuite analysées grâce à des outils OLAP. Traditionnellement, les nouvelles données dans ces entrepôts sont chargées grâce à des processus d'alimentation réalisant des insertions en bloc, déclenchés périodiquement lorsque l'entrepôt est hors-ligne. Une telle stratégie implique que d'une part les données de l'entrepôt ne sont pas toujours à jour, et que d'autre part le système de décisionnel n'est pas continuellement disponible. Or cette latence n'est pas acceptable dans certaines applications modernes, tels que la surveillance de bâtiments instrumentés dits "intelligents", la gestion des risques environnementaux etc., qui exigent des données les plus récentes possible pour la prise de décision. Ces applications temps réel requièrent l'intégration rapide et atomique des nouveaux faits dans l'entrepôt de données. De plus, ce type d'applications opérant dans des environnements fortement évolutifs, les données définissant les dimensions d'analyse elles-mêmes doivent fréquemment être mises à jour. Dans cette thèse, de tels entrepôts de données sont qualifiés d'entrepôts de données dynamiques. Nous proposons un modèle de données pour ces entrepôts dynamiques et définissons un espace hiérarchique de données appelé Hierarchical Hybrid Multidimensional Data Space (HHMDS). Un HHMDS est constitué indifféremment de dimensions ordonnées et/ou non ordonnées. Les axes de l'espace de données sont non-ordonnés afin de favoriser leur évolution dynamique. Nous définissons une structure de regroupement de données, appelé Minimum Bounding Space (MBS), qui réalise le partitionnement efficace des données dans l'espace. Des opérateurs, relations et métriques sont définis pour permettre l'optimisation de ces partitions. Nous proposons des algorithmes pour stocker efficacement des données agrégées ou détaillées, sous forme de MBS, dans une structure d'arbre appelée le DyTree. Les algorithmes pour requêter le DyTree sont également fournis. Les nœuds du DyTree, contenant les MBS associés à leurs mesures agrégées, représentent des sections matérialisées de cuboïdes, et l'arbre lui-même est un hypercube partiellement matérialisé maintenu en ligne à l'aide des mises à jour incrémentielles. Nous proposons une méthodologie pour évaluer expérimentalement cette technique de matérialisation partielle ainsi qu'un prototype. Le prototype nous permet d'évaluer la structure et la performance du DyTree par rapport aux autres solutions existantes. L'étude expérimentale montre que le DyTree est une solution efficace pour la matérialisation partielle d'un cube de données dans un environnement dynamique.

MOTS-CLES : OLAP, Matérialisation partielle, Index multidimensionnel, Cube de données,

Entrepôt de données temps-réel

Laboratoire de Recherche : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)

Directeur(s) de Thèse : Maryvonne MIQUEL, Anne TCHOUNIKINE

Président de Jury :

Composition du Jury : Ladjel BELLATRECHE (Rapporteur), Maryvonne MIQUEL (Co-dir. de thèse), Jean-Marc PETIT (Examinateur), Franck RAVAT (Examinateur), Anne TCHOUNIKINE (Co-dir. de thèse), Karine ZEITOUNI (Rapporteur), Estaban ZIMANYI (Examinateur)