I slept
and dreamt that life was joy.
I awoke and saw that life was service.
I acted and behold, service was joy

(Tagore)

to my grand-parents, my family,
all my angels, for their support

Acknowledgement

This thesis is the result of an intense research activity performed at Department of Electronics, Computer Sciences and Systems (DEIS) of the University of Calabria with the support of a number of people to which I want to express my thankfulness.

My first “thank you” goes to my advisor, prof. D. Saccà, whose personality and deepness has always been the light to follow in this research experience, without him anything would have been the same, the *lighthouse* of this research world! Secondly I’d like to thank Elio Masciari he started to support me when I was just a student, and because of him I’m able to finish this awesome research program.

I’m also very grateful to prof. Luigi Palopoli for his careful guidance as PhD program coordinator.

Furthermore I can not forget all my friends ICAR-CNR institute, which supported me in every situation and helping to make me stronger: thanks to Gianni, Ettore, Francesco, Eugenio, Giuseppe, Agostino, Massimo, they have been so precious, and Alfredo who guided me in my first years of this PhD course.

Foremost, I owe a special thank to my mother and my father, they promote me in all situations and in every difficulties, they always encouraged me to overcome problems and sadness, making me much more motivated and enthusiastic. Thanks to my incomparable and inimitable sister, always a true and real support for everything and to my cousin Michele, my “missing brother”; thanks to my best friends Damiano, Eliana and Serena who have been always kind and encouraging me everytime

A real thanks to all my ‘American’ friends I met at UCLA and most of all to prof. Carlo Zaniolo who made my seven months experience at Computer Science department the best I could ever had.

The latter but not the last, a special thanks to Angela, my little curly angel, who supported, tolerated and endured me with her beautiful smile and love.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivations

How can a manager get out of a data-flooded mire? How can a confused decision maker navigate through a maze? How can an over-bursty stream of data be queried? How can is possible to better show data in a mess cuboid? How can the best moment to invest be found in the Stock Market trends?

The answer to all of these is challenging, last years saw many contributions from many research groups, trying to find the best way to manage huge amount of data, in all scenarios real life can avoid. Data management systems are becoming every day critical for companies and most of them trust innovative techniques to support decision in their business processes.

Decision making support frameworks and all related technologies innovative strategies are rising up introducing improvement and efficiency in querying, analysing and extracting information from data. All of these improvements are providing great perspectives to marketing research initiatives and investigations, network data storage management, massive data querying capabilities and knowledge retrieval from huge amount of raw data.

All these techniques are able to turn data into money; transform information into intelligence; change patterns into profit; and convert relationships into resources. To obtain knowledge, one must understand the patterns that emerge from information. Patterns are not just simple relationships among data; they exist separately from information, as archetypes or standards to which emerging information can be compared so that one may draw inferences and take action. Over the last 40 years, the tools and techniques used to process data and information have continued to evolve from databases (DBs) to data warehousing (DW), to Data Mining. DW applications, as a result, have become business-critical and can deliver even more value from these huge repositories of data.

Although technologies are deeply widespread in many economical and social environment, considerable differences related to '*how*' to manage potentially unbounded data when they arrive with a very high rate with respect to the processing capabilities of system, and the storage space is always limited. It is clear that systems which deal with bursty data need to take into account policies to process incoming data very fast and if needed to shed load when it is too much to be processed. It is straightforward at this point that it is possible to identify two main areas strictly related one another, which try to study new aspects, technologies, algorithms when manipulating *static data* - data that reside in a classical *Database Management System (DBMS) / Data Warehouse (DW)* - and *dynamic data* which are characterized by *data streams* which need to be processed.

These two areas received and receive today as well, great attention from research groups. The basic issue related to these two areas is that techniques to process and analyse static data using classic DBMS and DW systems, might not be applied to do the same kind of job considering data streams. The main motivation is that data streams are not bounded - information flood can be infinite - therefore it is not possible to store such kind of data in a classical DBMS which has a finite - even if large - storage capacity. Moreover data streams are often generated in sensor networks which are usually involved for monitoring activities, i.e. in geographical areas, network traffic, trend analysis etc.

Starting from these assumptions the problem of provide *efficiently* answers to queries which involve massive data, in case of static and dynamic data is challenging, and as stated above, techniques are reasonably different.

Static data are often analysed i.e. when a company (or corporation) needs to provide a realistic evaluation of the next marketing plan, to extract knowledge from them, perform data mining processes like classification, when a support to decision making process is needed.

The latter case is one of the most interesting aspects and it is usually performed adopting the well-known-in-literature on line analysis techniques, also known as *On Line Analytical Process (OLAP)*. OLAP analysis is very efficient particularly when it is needed to process hierarchical data - i.e. date (day-month-year), job position (simple employer - project manager - area manager etc.) - and aggregate ones (i.e. querying using aggregate operators like SUM, AVG, etc), because it provides operators which are particularly efficient to change data visualization in such a way as searched information is much easier to get. About OLAP systems there are some interesting aspects to be considered, like the querying problem, related to how much the answer approximation has to deal with exactness, if time constraints are hard; the way how data are represented and processed and above all how to represent data inside OLAP systems. The last cited point is very challenging and tricky but at the same time it adds great value to decision making support frameworks, in fact it is much easier to analyse data when they are better shown to the user (or application). Also data compression can provide great improvements in query answering process time execution, in fact it is possible to consider different policies that show recent data or data with certain characteristics in order to meet application requirements.

Strictly connected with these aspects, OLAP systems are very critical in privacy issues management. Many literature works have shown that malicious users can easily get sensible information from the datacube exploiting OLAP operators in manipulating data visualization. To this end, techniques aiming to protect sensitive information have been proposed and many of them exploit different aspects related to privacy preservation in OLAP and Data Warehousing in general.

It is clear that static data needs to be managed trying to improve performances in query answering, when queries involve huge amount of data and/or query complexity need too much calculate power to be performed.

Recently researchers are trying to apply benefits of multidimensional analysis of static data to systems dealing with dynamic ones, which have however a considerably different kind of architecture. As already said above, a new class of applications has emerged that requires managing data streams, i.e., data composed of continuous, real-time sequence of items. Moreover, database management systems were originally developed to support business applications. The data in such applications is changed as a result of human-initiated transactions. Similarly data is queried as a result of human-initiated queries. The database management system acts as a passive repository for the data, executing the queries and transactions when these are submitted. However, this model of a database management system as a repository of relatively static data that is queried as a result of human interaction, does not meet the challenges posed by streaming applications.

A data stream is a possibly unbounded sequence of data items. Streaming applications have gained prominence due to both technical and business reasons. Technologically data is now available from a wide variety of monitoring devices, including sensors that are extremely cheap. Data from such devices is potentially unbounded and needs to be processed in real-time.

Streaming applications pose new and interesting challenges for data management systems. Such application domains require queries to be evaluated continuously as opposed to the one time evaluation of a query for traditional applications.

Streaming data sets grow continuously and queries must be evaluated on such unbounded data sets. The monitoring aspect of many streaming applications requires support for reactive capabilities in real-time from data management systems. These, as well as other challenges, require a major rethink of almost all aspects of traditional database management systems to support streaming applications.

Key themes are related to compression of data, i.e. “ancient” data could be less important than “newer” ones, and therefore the challenging trade off between accuracy and exactness in answering queries.

In sum , the motivation of the thesis is to investigate the problem of providing effective support to handling huge amount of data and enabling spreading share and delivery of contents. To achieve this ambitious goal, different issues must to be dealt with such as: methods and algorithms to aggregate raw data using lossy compression (size resilience), amenities to disseminate information by removing privacy concerns in information exchange (privacy resilience), and tools for managing data streams (dynamicity resilience). The thesis does not offer a unified framework to the problem, for a solution is too far to be achieved; it rather gives insights on how to proceed toward such a solution and put some lights to all facets of the problem, by presenting novel results on each of them, which are relevant "per se" and may eventually represent a step forward a more comprehensive ways to manage continuous streams of large amount of data.

1.2 Main Contributions of the Thesis

This thesis is focused on management of data considering both the case of static and dynamic data. It proposes a number of novel contributions aiming at easing query optimization in the fields of OLAP data compression and privacy preservation, and approximate query answering in data streams, also in case of queries defined using a pattern syntax.

OLAP systems have been deeply investigated and analysed with particular regard to techniques related to intelligent and efficient compression and visualization techniques, in order to perform querying process with great performances. In the first case it has been proposed a novel top-down compression technique for data cubes, which takes into account the case in which multiple Hierarchical Range Queries (HRQ), a very useful class of OLAP queries, must be evaluated against the target data cube simultaneously. As result of the study, an innovative multiple-objective OLAP computational paradigm, and a hierarchical multidimensional histogram, has been shown, whose main benefit is meaningfully implementing an intermediate compression of the input data cube able to simultaneously accommodate an even large family of different-in-nature HRQ.

Another very interesting aspect has been taken in consideration, in the same area, it is a semantics-driven compression technique for efficiently supporting advanced OLAP visualization of multidimensional data cubes. One of the main contribution this thesis present consists in the amenity of using the data compression paradigm as a way of visualizing multidimensional OLAP domains, in order to overcome the natural disorientation and refractoriness of human beings in dealing with hyper-spaces. Experimental results conducted on several kind of synthetic data sets clearly confirm the effectiveness and the efficiency of this technique, also in comparison with state-of-the-art proposals.

As stated above, OLAP systems are very vulnerable if they contain sensible information. Literature proposed many works on this, and here it is presented a robust sampling-based framework for privacy preserving OLAP. The most distinctive characteristic of the proposed framework consists in adopting an innovative privacy OLAP notion, which deals with the problem of preserving the privacy of OLAP aggregations rather than the one of data cube cells, like in conventional perturbation-based privacy preserving OLAP techniques. This results in a greater theoretical soundness, and lower computational overheads due to processing massive-in-size data cubes. Also, performances of this privacy preserving OLAP technique is compared with the one of the method Zero-Sum[SLXN06], the state-of-the-art privacy preserving OLAP perturbation-based technique, under several perspectives of analysis. The derived experimental results confirm to us the benefits deriving from adopting our proposed framework for the goal of preserving the privacy of OLAP data cubes.

In Data Stream Management Systems (DSMS) it is not possible to manage data using classical DBMS techniques, because of space bounds and incoming data rate which can be very high. To this end, DSMS introduces new policies and compression approaches which characterizes the whole system architectures. In this thesis is presented the system *SensorGrid*[CFMS04], a Grid-based sensor network data warehouse, which encompasses several metaphors of data compression/approximation and high performance and high reliability computing that are typical of Grid architectures. Experimentation focuses on two main classes of aggregate range queries over sensor readings, namely *(i)* the window queries, which apply a SQL aggregation operator over a fixed window over the reading stream produced by the sensor network, and *(ii)* the continuous queries, which instead consider a "moving" window, and produce as output a stream of answers. Both classes of queries are extremely useful to extract summarized knowledge to be exploited by OLAP-like analysis tools over sensor network data. The experimental results, conducted on several synthetic data sets, clearly confirm the benefits deriving from embedding the data compression/approximation paradigm into Grid-based sensor network data warehouses.

The querying process in DSMS needs to be very fast and concrete, as already said, in order to provide answers as soon as possible. In many cases queries might not have the classical SQL form, but used languages borrow main constructs and syntax. A very exciting field, still not analysed enough is strictly related to optimization of queries execution when using Kleene closure operator. In this case literature does not provide many techniques able to deal with Kleene-* operator, except for [SZZA01a], but it can be sensibly improved with respect to scalability in case of many complex queries at a time. The problem has been centered on designing a framework to efficiently evaluate patterns queries containing general predicate, Kleene closure and predicate functions.

Complex patterns queries involving Kleene closure are finding applications in many areas including financial services, healthcare monitoring, RFID-based inventory management etc.

Many continual predicate queries can be issued against a rapid and bursty data stream. Furthermore query predicates may be comprised of Kleene closure or predicate functions. Previous approaches involving query indexing techniques are not scalable in this environment, therefore to efficiently evaluate continual queries a new framework is required to address these issues.

The proposed framework has been is a hybrid method that uses indexing and string matching techniques to provide *high performance pattern matching over streams (HPSM)* involving general predicates, Kleene closure and functions. While indexing of patterns involving general predicates was well studied, indexing of Kleene closure and functional patterns over streams have unique issues regarding scalability in both performance and size of the index. All these issues are explained in a context of thousands of continuous queries. In particular, such a hybrid framework provides a scalable solution to a pattern matching problem and describe support for indexing functional patterns.

The main literature works chosen to compare against are the one on the *YFilter* (see [yfi02]) and *Suffix Tree* (see [McC76]) approaches. In both cases raw experimentation seems to show that the proposed approach significantly outperforms the existing ones in terms of both storage and time. However this is still an on-going work, therefore all motivations justifying experimentation will be better presented in 7.

1.3 Thesis Organization

The reminder of this thesis is structured as follows.

Chapter 2 presents an overview of data warehouse and OLAP systems. In particular it focus on data representation and compression in OLAP, explaining with the right theoretical details all advantages in using such techniques. Better data visualization helps user/application to easier retrieve knowledge and make querying process faster. Data compression in OLAP datacube is a pretty straightforward techniques that allows to consider multiple aggregate levels in order to quickly obtain approximated query answers keeping a very high accuracy level.

Chapter 3 is centred on security and privacy preservation techniques in OLAP systems. They become very important since malicious users can use statistical inference to get sensitive information from the system. Literature proposed technique are focusing many aspects and it is also explained a new technique which introduce a new *privacy OLAP definition*, based on aggregates and compares experimental result one of the most famous literature work from prof. Ng.

Chapter 4 presents a detailed overview on data stream systems and their main features. It deeply analyse motivations and differences against classical DBMS. Many existing DSMSs provide nice features and challenging goals, trying to enrich the system with functionalities. To this end it describes a DSMS system (*SensorGrid*)and its architecture and shows experimental evaluation of its performances with many different parameters over synthetic and real data sets.

Chapter 5 introduces actual works on a particular aspect of the wide stream mining field: the *frequent trajctory mining*. It proposes a technique that will be better extended to support ordering in frequent patterns research in data stream itemsets.

Chapter 6 deals with the open problem of Kleene closure operator in *pattern queries* when considering data streams. In this case it is to be hoped that if a multiquery environment is set, then an optimized scheduling preprocessing phase would improve performances and the speed in getting the right answer from all pattern queries.

Finally Chapter 7 draws some conclusions and highlights some still open issues that are worth considering in further investigation.

2

Querying and Answering Multidimensional Static Data

2.1 Introduction

When companies are dealing with huge amounts of data, often they need to consider technologies that allow to efficiently manage them, providing fast answer and when it is needed the maximum accuracy possible. Most of these technologies are strictly related to the *datawarehousing* area and therefore all aspects concerning decision support, efficient query answers have to be taken into consideration, mainly when data are *static*, that basically means that they are relying on large storage devices and they do not change within small amount of time.

There are many meanings stating what a *data warehouse* is and what are its main functions and characteristics. From a certain point of view we can assert a data warehouse is a repository of an organization's electronically stored data and it is usually designed to facilitate reporting and analysis. More technically, a data warehouse houses a standardized, consistent, clean and integrated form of data sourced from various operational systems in use in the organization, structured in a way to specifically address the reporting and analytic requirements.

This definition of the data warehouse focuses on data storage. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes *business intelligence* tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve metadata.

2.1.1 Storage of Data: Normalized VS Dimensional Approach

There are two leading approaches to storing data in a data warehouse - the dimensional approach and the normalized approach.

In the *dimensional approach*, transaction data are partitioned into either *facts*, which are generally numeric transaction data, or *dimensions*, which are the reference information that gives context to the facts. For example, a sales transaction can be broken up into facts such as the number of products ordered and the price paid for the products, and into dimensions such as order date, customer name, product number, order ship-to and bill-to locations, and salesperson responsible for receiving the order. A key advantage of a dimensional approach is that the data warehouse is easier for the user to understand and to use. Also, the retrieval of data from the data warehouse tends to operate very quickly.

The main disadvantages of the dimensional approach are: 1) In order to maintain the integrity of facts and dimensions, loading the data warehouse with data from different operational systems is complicated, and 2) It is difficult to modify the data warehouse structure if the organization adopting the dimensional approach changes the way in which it does business.

In the *normalized approach*, the data in the data warehouse are stored following, to a degree, database normalization rules. Tables are grouped together by subject areas that reflect general data categories (e.g., data on customers, products, finance, etc.) The main advantage of this approach is that it is straightforward to add information into the database. A disadvantage of this approach is that, because of the number of tables involved, it can be difficult for users both to 1) join data from different sources into meaningful information and then 2) access the information without a precise understanding of the sources of data and of the data structure of the data warehouse.

These approaches are not mutually exclusive, and of course there are other approaches. Dimensional approaches can involve normalizing data to a degree.

Data warehouses are optimized for speed of data retrieval. Frequently data in data warehouses are denormalised via a dimension-based model. Also, to speed data retrieval, data warehouse data are often stored multiple times - in their most granular form and in summarized forms called aggregates. Data warehouse data are gathered from the operational systems and held in the data warehouse even after the data has been purged from the operational systems.

2.2 On Line Analytical Processing (OLAP)

One of the most important technique to retrieve efficiently data providing good support to business intelligence is the *On Line Analytical Processing (OLAP)*. It is an approach to quickly answer multi-dimensional analytical queries. The typical applications of OLAP are in business reporting for sales, marketing, management reporting, business process management (BPM), budgeting and forecasting, financial reporting and similar areas. The term OLAP was created as a slight modification of the traditional database term *OLTP (Online Transaction Processing)*.

Databases configured for OLAP use a multidimensional data model, allowing for complex analytical and ad-hoc queries with a rapid execution time. They borrow aspects of navigational databases and hierarchical databases that are faster than relational databases.

The output of an OLAP query is typically displayed in a matrix (or pivot) format. The dimensions form the rows and columns of the matrix; the measures form the values.

At the core of any OLAP system is the concept of an *OLAP cube* (also called a *multidimensional cube* or a *hypercube*). It consists of numeric facts called *measures* which are categorized by dimensions. The cube metadata is typically created from a *star schema* or *snowflake schema* of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

Each measure can be thought of as having a set of labels, or meta-data associated with it. A dimension is what describes these labels; it provides information about the measure.

A simple example would be a cube that contains a store's sales as a measure, and Date/Time as a dimension. Each Sale has a Date/Time label that describes more about that sale.

Any number of dimensions can be added to the structure such as Store, Cashier, or Customer by adding a column to the fact table. This allows an analyst to view the measures along any combination of the dimensions.

Three relevant challenges of OLAP systems [CD97, GCB$^+$97] have captured a lot of attention during the last years: *(i)* the data querying problem, which concerns with how data are accessed and queried to support summarized knowledge extraction from massive data cubes; *(ii)* the data modeling problem, which concerns with how data are represented and, thus, processed inside OLAP servers (e.g., during query evaluation); *(iii)* the data visualization problem, which concerns with how data are presented to OLAP users and decision makers in *Data Warehouse* environments. Indeed, research communities have mainly studied and investigated the first two problems, whereas the last one, even if important-with-practical-applications, has been very often neglected.

Approximate Query Answering (AQA) techniques address the first challenge, and can be reasonably considered as one of the most important topics in OLAP research. The main proposal of AQA techniques consists in providing approximate answers to resource-consuming OLAP queries (e.g., range-queries) instead of computing exact answers, as decimal precision is usually negligible in OLAP query and report activities. Due to a relevant interest from the Data Warehousing research community, AQA techniques have been intensively investigated during the last years, also achieving important results. Among the others, histograms (e.g., [APR99a, BCG01, GKTD00a]), wavelets ([VWI98]), and sampling (e.g., [GM98]) are the most successful techniques, and they have also inducted several applications in even different contexts than OLAP. Conceptual data models for OLAP are widely recognized as based on data cube concepts like dimension, hierarchy, level, member, and measure, first introduced by Gray et al. [GCB$^+$97], which inspired various models for multidimensional databases and data cubes (e.g., [HS98, Vas98]). Nevertheless, despite this effort, recently, several papers have put in evidence some formal limitations of accepted conceptual models for OLAP (e.g., [CT98]), or theoretical failures of popular data cube operations, like aggregation functions. Contrarily to data querying and modeling issues, since data presentation models do not properly belong to the well-founded conceptual-logical-physical hierarchy for relational database models (which has also been inherited from multidimensional models), the problem of OLAP data visualization has been studied and investigated only so far (e.g., [GJJ97, MVSV03]). On the other hand, being essentially OLAP a technology to support decision making, thus based on (sensitive) information exploration and browsing, it is easy to understand that, in future years, tools for advanced visualization of multidimensional data cubes will rapidly conquest the OLAP research scene.

2.3 Advanced OLAP Visualization Using Hierarchies

Starting from the fundamentals of data cube compression and OLAP data visualization research issues, it is possible to meaningfully exploit the main results coming from the first one and the goals of the second one in a combined manner, and obtaining a novel technique for supporting advanced OLAP visualization of multidimensional data cubes. The basic motivation of such an approach is realizing that *(i)* compressing data is an (efficient) way of visualizing data, and *(ii)* this intuition is well-founded at large (i.e., for any data-intensive system relying on massive data repositories), and, more specifically, it is particularly targeted to the OLAP context where accessing multidimensional data cubes can become a realistic bottleneck for Data Warehousing systems and applications. Briefly, it is possible to distinguish two steps. The first one consists in generating a two-dimensional OLAP view D from the input multidimensional data cube A by means of an innovative approach that allows us to flatten OLAP dimensions (of A), and, as a consequence, effectively support exploration and browsing activities against A (via D), by overcoming the natural disorientation and refractoriness of human beings in dealing with hyper-spaces.

Particularly, the (two) OLAP dimensions on which D is defined are built from the dimensions of A according to the analysis goals of the target OLAP user/application. The second step consists in generating a bucket-based compressed representation of D named as *Hierarchy-driven Indexed Quad-Tree Summary (H-IQTS)*, and denoted by H-IQTS(D), which meaningfully extends the compression technique for two-dimensional summary data domains presented in [BFSS03], by introducing the amenity of generating semantics-aware buckets, i.e. buckets that follow groups of the OLAP hierarchies of D. In other words, OLAP hierarchies defined on the dimensions of D are used to drive the compression process. The latter step allows us to achieve space efficiency, while, at the same time, supporting approximate query answering and advanced OLAP visualization features. Similarly to [BFSS03], *H-IQTS(D)* is shaped as a quad-tree (thus, each internal bucket in *H-IQTS(D)* has four child sub-buckets), and the information stored in its buckets is still the sum of the items contained within them.

Given an OLAP dimension d_i, and its domain of members $\Psi(d_i)$, each of them denoted by ρ_j, a hierarchy defined on d_i, denoted by $H(d_i)$, can be represented as a general tree (i.e., such that each node of the tree has a number $N \geq 0$ of child nodes) built on top of $\Psi(d_i)$. $H(d_i)$ is usually built according to a bottom-up strategy by *(i)* setting as leaf nodes of $H(d_i)$ members in $\Psi(d_i)$, and *(ii)* iteratively aggregating sets of members in $\Psi(d_i)$ to obtain other (internal) members, each of them denoted by σ_j, (the latter are internal nodes in $H(d_i)$). In turn, internal members in $H(d_i)$ can be further aggregated to form other super-members until a unique aggregation of members is obtained; the latter corresponds to the root node of $H(d_i)$, and it is known-in-literature as the aggregation ALL. Each member in $H(d_i)$ is characterized by a level (of the hierarchy), denoted by L_j; as a consequence, we can define a level L_j in $H(d_i)$ as a collection of members. For each level L_j, the ordering of L_j, denoted by $O(L_j)$, is the one exposed by the OLAP server platform for the target data cube on which $H(di)$ is defined. Given a multidimensional data cube A such that $Dim(A) = d_0, d_1, d_{n-1}$ is the set of dimensions of A and $Hie(A) = \{H(d_0), H(d_1), \cdots, H(d_{n-1})\}$ the set of hierarchies defined on A, the collection of members σ_j at the level $L_j \geq 0$ (note that, when $L_j = 0$, $\sigma_j \equiv \rho_j$) of each hierarchy $H(d_i)$ in $Hie(A)$ univocally refers, in a multidimensional fashion, a certain (OLAP) data cell C_p in A at level L_j (in other words, Cp is the OLAP aggregation of data cells in A at level L_j). Such collection is named as j-level OLAP Metadata (for C_p), and denoted as $J - M(C_p)$.

Given a member σ_j at level L_j of $H(d_i)$ and the set of its child nodes $Child(\sigma_j)$, which are members at level L_{j+1}, we define as the *Left Boundary Member (LBM)* of σ_j the child node of σ_j in $Child(\sigma_j)$ that is the first in the ordering $O(L_{j+1})$. Analogously, we define as the *Right Boundary Member (RBM)* of σ_j the child node of σ_j in $Child(\sigma_j)$ that is the last in the ordering $O(L_{j+1})$.

2.3.1 Dimension Flattening

The OLAP dimension flattening process is the first step for supporting advanced OLAP visualization of multidimensional data cubes. In more detail, dimensions of the input multidimensional data cube A can be flatten into two specialized dimensions called *Visualization Dimensions (VDs)*. VDs support advanced OLAP visualization of A via constructing an ad-hoc two-dimensional OLAP view D defined just on VDs.

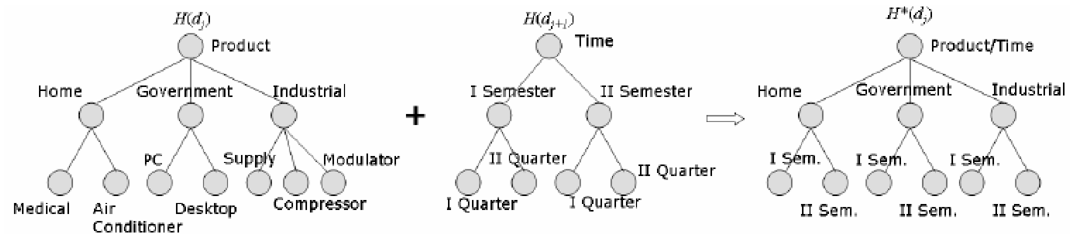


Fig. 2.1. Merging OLAP hierarchies

The process that allows to obtain the two VDs from the dimensions of A works as follows. Letting $Dim(A)$ and $Hie(A)$ be the set of dimensions and the set of hierarchies of A, respectively, each VD is a tuple $vi = \langle d_i, H^*(d_i)\rangle$ such that *(i)* d_i is the dimension selected by the target OLAP user/application, *(ii)* $H^*(d_i)$ is a hierarchy built from meaningfully merging the *original* hierarchy $H(d_i)$ of d_i with the hierarchies of other dimensions in A according to an ordered definition set

$$MD(v_i) = \{\langle HL_i, d_j, Pj\rangle, \langle HL_j, d_{j+1}, P_{j+1}\rangle, \cdots, \langle HL_{j+K-1}, d_{j+K}, P_{j+K}\rangle\}$$

where $K = |MD(v_i)| - 1$. More in detail, for each pair of consecutive tuples $\langle\langle HL_j, d_{j+1}, P_{j+1}\rangle, \langle HL_{j+1}, d_{j+2}, P_{j+2}\rangle\rangle$ in $MD(v_i)$ the subtree of $H(d_{j+2})$ rooted in the root node of $H(d_{j+2})$ and having depth equal to P_{j+2}, denoted by $H_S^{P_{j+2}}(d_{j+2})$, is merged to $H(d_{j+1})$ by appending a clone of it to each member σ_{j+1} at level HL_{j+1}, named as hooking level, in $H(d_{j+1})$. From the described approach, it follows that: *(i)* the ordering of items in $MD(v_i)$ defines the way of building $H^*(d_i)$; *(ii)* the first hierarchy to be processed is just $H(d_i)$. As an example of the flattening process of two OLAP dimensions into a new one, consider 2.1 where the hierarchy $H^*(d_j)$ is obtained by merging $H(d_{j+1})$ to $H(d_j)$ via setting $P_{j+1} = 1$ and $HL_j = 1$.

2.3.2 Hierarchy-Driven Compression of Two-Dimensional OLAP Views

Compressing the two-dimensional OLAP view D (extracted from A according to the OLAP dimension flattening process presented in 2.3.1) is the second step of our proposed technique.

Given D, for each step j of our compression algorithm, we need to

1. greedily select the leaf bucket b of $H-IQTS(D)$ having maximum Sum of the Squared Errors (SSE) [BFSS03],
2. split b in four sub-buckets through investigating, for each dimension d_k of D, levels of the hierarchy $H(d_k)$.

Formally, given the current bucket $b_j = D[l_{j,0} : u_{j,0}][l_{j,1} : u_{j,1}]$ to be split at step j of our compression algorithm, such that $[l_{j,k} : u_{j,k}]$ is the range of b_j on the dimension d_k of D, the problem is finding, for each dimension d_k of D, a splitting position $S_{j,k}$ belonging to $[l_{j,k} : u_{j,k}]$. To this end, for each dimension d_k of D, our splitting strategy aims at *(i)* grouping items into buckets related to the same semantic domain, and *(ii)* maintaining as more balanced as possible the hierarchy $H(d_k)$.

For the sake of simplicity, let us consider the hierarchy-driven compression algorithm for two-dimensional OLAP views through showing how to handle the hierarchy of an OLAP dimension d_k (i.e., how to determine a splitting position $S_{j,k}$ on d_k). Obviously, this technique must be performed for both the dimensions of the target (two-dimensional) OLAP view D, thus obtaining, for each pair of splits at step j (i.e., $S_{j,0}$ and $S_{j,1}$), four two-dimensional buckets to be added to the current partition of D.

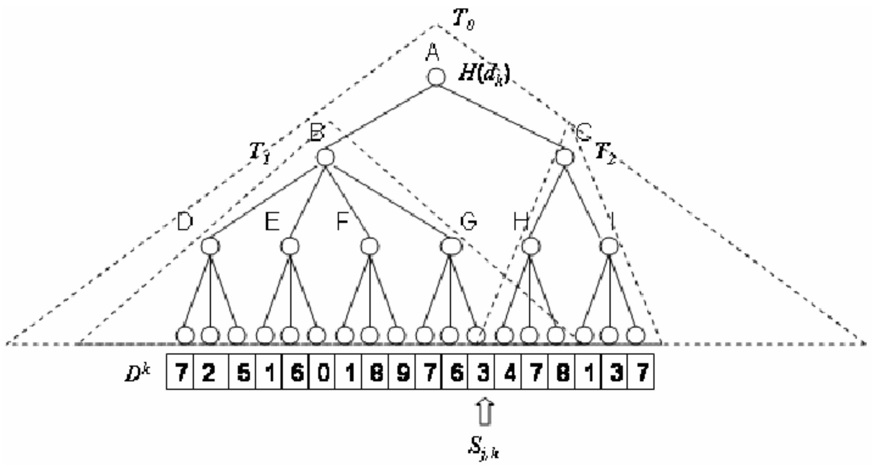


Fig. 2.2. Modeling Splitting Strategy

Let $bj = D[l_{j,0} : u_{j,0}][l_{j,1} : u_{j,1}]$ be the current bucket to be split at step j. Consider the range $[l_{j,k} : u_{j,k}]$ of b_j on the dimension d_k of D. To determine $S_{j,k}$ on $[l_{j,k} : u_{j,k}]$, we denote as $T_{j,k}(l_{j,k} : u_{j,k})$ the sub-tree of $H(d_k)$ whose *(i)* leaf nodes are the members of the sets $0 - M(C_w)$ defined on data cells C_w in $D[l_{j,k} : u_{j,k}]$ (i.e., the one-dimensional bucket obtained by projecting bj with respect to the dimension d_k), and *(ii)* the root node is the (singleton) member of the set $Pk - M(C_r)$ defined on the data cell C_r that is the (singleton) aggregation of $D[l_{j,k} : u_{j,k}]$ at level LP of $H(d_k)$, being P_k the depth of $H(d_k)$ (and also the depth of $T_{j,k}(l_{j,k} : u_{j,k})$). To give an example, consider 2.2, where the one-dimensional OLAP view $D_k = D[0 : |d_k| - 1]$, obtained by projecting D with respect to the dimension d_k, along with the hierarchy $H(d_k)$ are depicted. As shown in 2.2, the tree T_0, properly denoted by $T_{j,k}(0 : 17)$, is related to the whole OLAP view $D_k = D[0 : 17]$, and corresponds to the whole $H(d_k)$. At step j, d_k is split in the position $S_{j,k} = 11$, thus generating the buckets $D[0 : 11]$ and $D[12 : 17]$. In consequence of this, the tree $T1$, properly denoted by $T_{j+1,k}(0 : 11)$, is related to $D[0 : 11]$, whereas the tree $T2$, properly denoted by $T_{j+1,k}(12 : 17)$, is related to $D[12 : 17]$.

Formally, let *(i)* d_k be the dimension of D to be processed; *(ii)* $H(d_k)$ the hierarchy defined on d_k; *(iii)* $b_j = D[l_{j,k} : u_{j,k}]$ the current (one-dimensional) bucket to be split at step j of our algorithm; *(iv)* $T_{j,k}(l_{j,k} : u_{j,k})$ the tree related to b_j. In order to select the splitting position $S_{j,k}$ on $[l_{j,k} : u_{j,k}]$, letting $T^1_{j,k}(l_{j,k} : u_{j,k})$ be the second level of $T_{j,k}(l_{j,k} : u_{j,k})$, we initially consider the data cell C_k in $D[l_{j,k} : u_{j,k}]$ whose indexer is in the middle of $D[l_{j,k} : u_{j,k}]$, denoted by

$$X_{j,D} = \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor$$

It should be noted that processing the second level of $T_{j,k}(l_{j,k} : u_{j,k})$ (i.e., $T_{j,k}(l_{j,k} : u_{j,k})$) derives from the use of the aggregation ALL in OLAP conceptual models, which, in total, introduces an additional level in the general tree modeling an OLAP hierarchy.

Then, starting from ρ_k, being ρ_k the member in the set $0 - M(C_k)$, we go up on $H(d_k)$ until the parent of ρ_k at the level $T^1_{j,k}(l_{j,k} : u_{j,k})$, denoted by σ_k, is reached, $S_{j,k}$ is determined on the basis of the nature of σ_k. If σ_k is the LBM of the root node of $T_{j,k}(l_{j,k} : u_{j,k})$ denoted by $R_{j,k}$, then

$$S_{j,k} = \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor - 1$$

and, as a consequence, the following two (one-dimensional) buckets as child buckets of b_j are obtained:

$$b'_{j+1} = D\left[l_{j,k} : \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor - 1\right]$$

$$b''_{j+1} = D\left[\left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor : u_{j,k}\right]$$

Otherwise if σ_k is the RBM of $R_{j,k}$, then

$$S_{j,k} = \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor$$

and as a consequence

$$b'_{j+1} = D\left[l_{j,k} : \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor\right]$$

$$b''_{j+1} = D\left[\left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor + 1 : u_{j,k}\right]$$

Finally, if σ_k is different from both the LBM and the RBM of $R_{j,k}$, i.e. it follows the LBM of $R_{j,k}$ in the ordering $O(T^1_{j,k}(l_{j,k} : u_{j,k}))$ and precedes the RBM of $R_{j,k}$, then it can be performed a finite number of shift operations on the indexers of $D[l_{j,k} : u_{j,k}]$ starting from the middle indexer $X_{j,D}$ and within the range $\Gamma_{j,k} = \left\lfloor \Gamma^{lo}_{j,k} : \Gamma^{up}_{j,k} \right\rfloor$, such that

$$\Gamma^{lo}_{j,k} = \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor - \left\lfloor \frac{1}{3} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor$$

$$\Gamma^{up}_{j,k} = \left\lfloor \frac{1}{2} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor + \left\lfloor \frac{1}{3} \cdot |D[l_{j,k} : u_{j,k}]| \right\rfloor$$

until a data cell V_k in $D[l_{j,k} : u_{j,k}]]$ such that the corresponding member σ_k at the level $T^1_{j,k}(l_{j,k} : u_{j,k})$ is the LBM or the RBM of $R_{j,k}$, if exists, is found. It should be noted that admitting the specified maximum offset

$$maxOffset = \pm \left\lfloor \frac{1}{3} \cdot \left|D^k[l_{j,k} : u_{j,k}]\right| \right\rfloor$$

with respect to the middle of the current bucket is coherent with the aim of maintaining as more balanced as possible the hierarchy $H(d_k)$, which allows to take advantages from the above-highlighted benefits.

To this end, starting from the middle of $\Gamma_{j,k}$ (which is equal to the one of $D[l_{j,k} : u_{j,k}], X_{j,D})$, it is possible to iteratively consider indexers $I_{j,q}$ within $_{j,k}$ on the basis of the following function:

$$I_{j,q} = \{X_{j,D} if q = 0; I_{j,q-1} + (-1)^q \cdot q if q > 1\}$$

. If such data cell V_k exists, then $S_{j,k}$ is set as equal to the so-determined indexer $I^*_{j,q}$, and, as a consequence, the pairs of buckets are:

$$b_{j+1} = D \left\lfloor l_{j,k} : I^*_{j,q} 1 \right\rfloor$$

$$b'_{j+1} = D \left\lfloor I^*_{j,q} : u_{j,k} \right\rfloor$$

if $I^*_{j,q}$ is the LBM of $R_{j,k}$, or, alternatively, the pairs of buckets area

$$b_{j+1} = D \left\lfloor l_{j,k} : I^*_{j,q} \right\rfloor$$

$$b'_{j+1} = D \left\lfloor I^*_{j,q} + 1 : u_{j,k} \right\rfloor$$

if $I^*_{j,q}$ is the RBM of $R_{j,k}$.

On the contrary, if such data cell V_k does not exist, then no split can be performed on $D[l_{j,k} : u_{j,k}]$, and the splitting is remanded at the next step of the algorithm (i.e., $j + 1$) where the splitting position $S_{j+1,k}$ is determined by processing the third level $T^2_{j+1,k}(l_{j+1,k} : u_{j+1,k})$ of the tree $T_{j+1,k}(l_{j+1,k} : u_{j+1,k})$ (i.e., by decreasing the aggregation level of OLAP data with respect to the previous step). The latter approach is iteratively repeated until a data cell Vk verifying the above condition is found; otherwise, if the leaf level of $T_{j,k}(l_{j,k} : u_{j,k}))$ is reached without finding any admissible splitting point, then $D[l_{j,k} : u_{j,k}]$ is added to the current partition of the OLAP view without being split. This way to do still pursues the aim of obtaining balanced partitions of the input OLAP view.

2.3.3 Experimental Study

It is possible to consider two kinds of experiments. The first one is oriented to probe the data cube compression performance (or, equally, the accuracy) of our technique, whereas the second one is instead oriented to probe the visualization capabilities of this technique in meaningfully supporting advanced OLAP visualization of multidimensional data cubes.

As regards the data layer of our experimental framework, engineered two kinds of synthetic two-dimensional OLAP views (which, in turn, have been extracted from synthetic multidimensional data cubes via a random flattening process): *(i)* the view $D_C(L_1, L_2)$, for which data are uniformly distributed on a given range $[L_1, L_2]$ (i.e., the well-known *Continuous Values Assumption (CVA)* holds), and *(ii)* the view $D_Z(z_{min}, z_{max})$, for which data are distributed according to a Zipf distribution whose parameter z is randomly chosen on a given range $[z_{min}, z_{max}]$.

For the first kind of experiments (i.e., that focused on the accuracy), given a population of synthetic range-SUM queries Q_S, we measure the *Average Relative Error (ARE)* between exact and approximate answers to queries in QS, i.e.:

$$\overline{E}_{rel} = \frac{1}{|Q_S|} \cdot \sum_{k=0}^{|Q|-1} E_{rel}(Q_k)$$

such that for each query Q_k in Q_S:

$$E_{rel}(Q_k) = \frac{|A(Q_k) - \tilde{A}(Q_k)|}{A(Q_k)}$$

where *(i)* $A(Q_k)$ is the exact answer to Q_k, and *(ii)* $\tilde{A}(Q_k)$ is the approximate answer to Q_k. Particularly, fixing a range sizes Δ_k for each dimension d_k of the target synthetic OLAP view D, we generate queries in Q_S through spanning D by means of the seed $\Delta_0 \times \Delta_\ell$ query Q^s.

For the second kind of experiments, inspiration comes from the *Hierarchical Range Queries (HRQ)* introduced by Koudas et al. in [KMS00]. A *HRQ* $Q_H(W_H, P_H)$ can be considered as a full tree such that: *(i)* the depth of such tree is equal to P_H; *(ii)* each internal node N_i has a fan-out degree equal to W_H; *(iii)* each node N_i stores the definition of a (traditional) range-SUM query Q_i; *(iv)* for each node N_i in $Q_H(W_H, P_H)$, there not exists any sibling node N_j of N_i such that $Q_i \cap Q_j <> \emptyset$. Similarly to the previous kind of experiments, for each node N_i in $Q_H(W_H, P_H)$, the population of queries $Q_{S,i}$

to be used as input query set was generated by means of the above-described spanning technique. It should be noted that, due the nature of *HRQs*, the selectivity of seed queries $Q^s_{i,k}$ of nodes N_i at level k of $Q_H(W_H, P_H)$ must decreases as the depth P_k of $Q_H(W_H, P_H)$ increases.

In consequence of this, letting γ be an input parameter and $\|D\|$ the selectivity of the target OLAP view D, it is possible to impose that the selectivity of the seed query of the root node N_0 in $Q_H(W_H, P_H)$, denoted by $\|Q^s_{0,0}\|$, is equal to the $\gamma\%$ of $\|D\|$, and then, for each internal node N_i in $Q_H(W_H, P_H)$ at level k, the seed queries of the child nodes of Ni can be randomly determined by checking the following constraint:

$$\sum_{i=0}^{|(W_{ff})^{k+1}|-1} \|Q^s_{i,k+1}\| \leq \|Q^s_{i,k}\| Q^s_{i,k+1} \cup Q^s_{j,k+1} = \emptyset$$

for each i and j in $[0, |(W_H)^{k+1}|1]$, with $i <> j$, and adopting the criterion of maximizing each $\|Q^s_{i,k}\|$.

Given a *HRB* $Q_H(W_H, P_H)$, it is interesting to measure the *Average Accessed Bucket Number (AABN)*, which models the average number of buckets accessed during the evaluation of $Q_H(W_H, P_H)$, and it is defined as follows:

$$AABN(Q_H(W_H, P_H)) = \sum_{k=0}^{P_H} \frac{1}{(W_H)^k} \cdot \sum_{l=0}^{|(W_{ff})^k|-1} AABN(N_\ell)$$

where, in turn, $AABN(N_\ell)$ is the average number of buckets accessed during the evaluation of the population of queries $Q_{S,l}$ of the node N_ℓ in $Q_H(W_H, P_H)$, i.e.

$$AABN(N_\ell) = \frac{1}{|Q_{S,l}|} \cdot \sum_{k=0}^{|Q_{S,l}|-1} ABN(Q_k)$$

each query Q_k in $Q_{S,l}$, $ABN(Q_k)$ is the number of buckets accessed during the evaluation of Q_k. Summarizing, given a compression technique *T*, *AABN* allows to measure the capabilities of T in supporting advanced OLAP visualization of multidimensional data cubes as the number of buckets accessed can be reasonably considered as a measure of the computational cost needed to extract summarized knowledge, as a sort of measure of the entropy of the overall knowledge extraction process.

Experimental study, compares performances against the following well-known histogram-based techniques for compressing data cubes: *MinSkew* by Acharya et al. [APR99b], *GenHist* by Gunopulos et al. [GKTD00b], and *STHoles* by Bruno et al. [BCG01]. More in detail, having fixed the space budget G (i.e., the storage space available for housing the compressed representation of the input OLAP view), we derived, for each comparison technique, the configuration of the input parameters that the respective authors consider

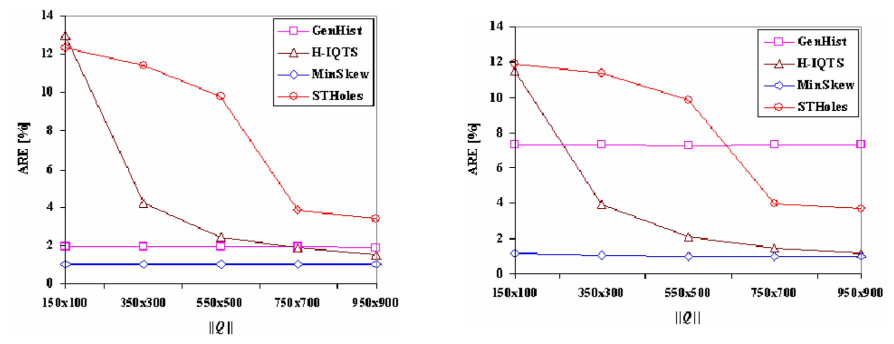


Fig. 2.3. Accuracy metrics w.r.t query selectivity

the best in their papers. This ensures a fair experimental analysis, i.e. an analysis such that each comparison technique provides its best performance.

2.3 shows experimental results for what regards the accuracy of the compression techniques w. r. t. the selectivity of queries in Q_S on the 1000×1000 two-dimensional OLAP views $D_C(25, 70)$ (*left side*) and $D_Z(0.5, 1.5)$ (*right side*), respectively. For all the comparison techniques, letting r be the parametric compression ratio and $size(D)$ the total occupancy of D, we set the space budget G as equal to the $r\%$ of $size(D)$. For instance, $r = 10$ (i.e., G is equal to the 10% of $size(D)$) is widely-recognized as a reasonable setting. 2.4 shows experimental results of the same experiment when ranging r on the interval $[5, 20]$ (i.e., G on the interval $[5, 20]\%$ of $size(D)$), and fixing the selectivity of queries ——Q——; this allows us to measure the scalability of the compression techniques, which is a critical aspect in OLAP systems (e.g., see [8]). Finally, Fig. 5 shows experimental results for what regards the visualization capabilities of the compari- son techniques (according to the guidelines drawn throughout the paper) with respect to the depth of HRQs (i.e., PH) having fan-out degree WH equal to 5 and the parameter equal to 70

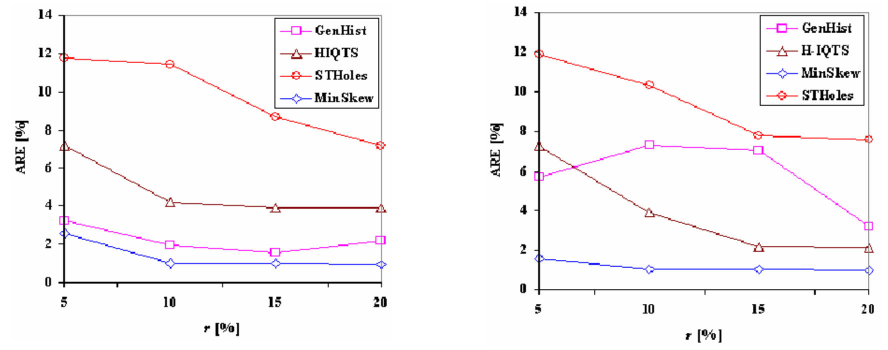


Fig. 2.4. Accuracy metrics w.r.t compression ratio

From 2.3, 2.4 and 2.5, it follows that, with respect to the accuracy metrics, our proposed technique is comparable with *MinSkew*, which represents the best on two-dimensional views (indeed, as well-recognized-in-literature, *MinSkew* presents severe limitations on multidimensional domains); instead, with respect to the visualization metrics, the proposed technique overcomes the comparison techniques, thus confirming its suitability in efficiently supporting advanced OLAP visualization of multidimensional data cubes.

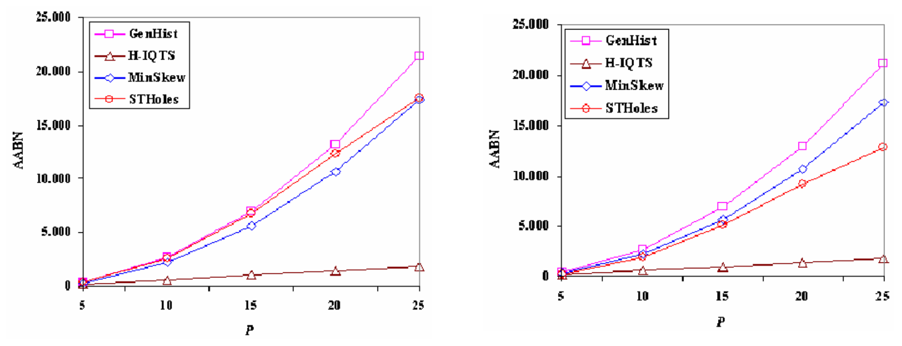


Fig. 2.5. Visualization metrics w.r.t depth of HRQ

2.4 Compressing Data Cubes under Simultaneous Queries

As it has already been seen, conventional data cube compression techniques, such as histograms, are devoted to drive the compression of the input data cube in dependence on one constraint only. Traditionally, this requirement is represented by a given space bound available to house the compressed representation of the data cube, like in conventional approaches. Without loss of generality, this scheme can be classified as adhering to what is usually called the *single-objective data cube compression paradigm*, which defines a class of methodologies wide enough to include most of the data cube compression proposals appeared in literature during the last two decades. This consolidated paradigm has been subsequently made more complex via including novel additional requirements to be considered simultaneously to the main space bound constraint, such as *(i)* compressing data cubes with the additional goal of minimizing the overall query error of a given query-workload (e.g., [BCG01, CW07]), *(ii)* ensuring probabilistic guarantees over the quality of approximate answers evaluated against compressed data cubes (e.g., [CW07]), or *(iii)* mediating on the degree of approximation of the retrieved answers (e.g., [Cuz06b, Cuz06a]).

More problematic issues appear when the data cube compression process must be performed in the presence of multiple constraints, under the *multiple-objective data cube compression paradigm*, which is a novel OLAP computational paradigm not considered by previous research. Basically, according to this novel paradigm, the compressed representation of the input data cube is obtained as that intermediate (compressed) representation which accomplishes, as much as possible, the multiple constraints defined by the input multiple-objective computational scheme. In fact, it is worthy noticing that, in the presence of multiple constraints, it is not possible to obtain a valid-for-all data cube compressed representation (i.e., the compressed representation that simultaneously satisfies all the multiple constraints) so that devising sub-optimal solutions appears to be the most promising strategy for the so-complex computational setting dictated by the multiple-objective application scenario. The idea of introducing multiple-objective computational paradigms in order to deal with complex Database and Data Warehousing research challenges has been considered in few contexts previously, and mostly with respect to requirements defined by multiple queries (i.e., simultaneous queries belonging to different query classes e.g., [Sel88]). Among these initiatives, it is important to recall: *(i)* multiple-query optimization for the view selection and materialization problem [MRSR01], *(ii)* multiple-query based data sources integration [NK01], *(iii)* multi-objective query processing in database systems [BG04] and OLAP [KP03], *(iv)* multi-objective query processing for specialized contexts such as data aggregation [FK06], complex mining tasks [JSA05] and data stream processing [WRGB06a], and, more recently, *(v) skyline query* processing [BKS01], which aims at extracting *Pareto distributions* from relational data tables according to multiple preferences. Contrary to the above-listed research contributions, to the best of our knowledge, there did not exist in literature data cube compression techniques that take into consideration the issue of performing the compression process on the basis of multiple objectives.

Despite theoretical issues, when a multiple-objective OLAP computational paradigm is adopted, one must first set the nature and the typology of multiple goals with respect to which the paradigm has to be implemented. Similarly to the above-listed research experiences, it is ordinary to choose different-in-nature queries as playing the role of multiple objectives to be accommodated during the compression process. In parallel to the nature and typology of multiple goals, a specific query class must be set. The general multiple-objective data cube compression paradigm can be easily customized to any class of OLAP queries, from simple range queries [HAMS97], which are defined as the application of a SQL aggregate operator (e.g., SUM, COUNT, AVG etc) to a given range of multidimensional data, to more complex ones such as iceberg queries [FSGM$^+$98]. Considering this assumption, it is possible to consider the class of *Hierarchical Range Queries (HRQ)*, already introduced as a meaningful extension of those defined by Koudas et al. in [KMS00]. As it will be evident HRQ define very useful tools for extracting hierarchically-shaped summarized knowledge from data warehouse servers, beyond the capabilities of conventional OLAP environments. Definition 1 introduces HRQ.

Definition 2.1. *Given a data cube $L = \langle D, H, M \rangle$, such that (i) D is the set of dimensions of L, (ii) H is the set of hierarchies of L, and (iii) M is the set of measures of L, a Hierarchical Range Query (HRB) Q_H against L is a tuple $Q_H = \langle T, F, \vartheta \rangle$, such that: (i) T is a general tree, (ii) F is a domain of range queries, (iii) ϑ is a mapping function that maps queries in F onto nodes in T, (iv) for each level ℓ in T,*

$$\bigcap_{k=0}^{|queries(l)|1} \|Q_k\| = \emptyset$$

such that $queries(l)$ is the set of range queries at level ℓ of T, and (v) for each node n T such that $depth(n) < depth(T)$

$$\bigcup_{k=0}^{|child(n)|-1} \|Q_k\| \neq \|Q_n\|$$

where: (v.i) $depth(n)$ denotes the depth of n, (v.ii) $depth(T)$ denotes the depth of T, (v.iii) $child(n)$ denotes the set of child nodes of n, (v.iv) Q_k denotes the range query related to the node k, and (v.v) $\|Q\|$ denotes the volume of Q.

It should be noted that, similarly to the multi-level and hierarchical nature of the target data cube L, HRQ are multi-level and hierarchical in nature as well, meaning that, for each level ℓ of T, the set of range queries at level ℓ, denoted by $queries(l)$, must be evaluated against the maximal cuboid of L at level ℓ, denoted by L_ℓ, i.e. the collection of (OLAP) data cells obtained by aggregating data cells of L at the lowest level of detail (i.e., $L_0 \equiv L$) with respect to all the aggregation levels of hierarchies in H at level ℓ. Due to the illustrated properties, it is a matter to note that, given a data cube L and a $HRQQ_H$ having T as structural tree, the answer to Q_H against L, denoted as $A(Q_H)$, is modeled in terms of a general tree Y having the same topology of T and such that each node n stores the answer $A(Q_n)$ to the corresponding range query Q_n. Also, without loss of generality, for the sake of simplicity here we assume that *(i)* hierarchies in H have all the same depth P, and *(ii)* the depth of T is also equal to P.

Depending on the kind of SQL aggregate operator characterizing range queries in F, different classes of HRQ can be obtained. It is quite straightforward to focus on *Hierarchical Range-SUM Queries*, as SUM aggregations are very popular in OLAP, and represent the most common solution for extracting useful knowledge from massive data cubes, while also encompassing the amenity of acting as baseline operators to build more complex OLAP aggregations/functions.

Range-SUM queries have been widely investigated in OLAP. As a consequence, in literature there exist a number of proposals dealing with the issue of answering such queries efficiently via data cube compression techniques (e.g., [HAMS97, Cuz06b]). Basically, histogram-based compression approaches have been proposed, with the appreciable idea of evaluating approximate answers by means of linear interpolation techniques applied to buckets of the histogram via meaningfully exploiting the *Continuous Value Assumption (CVA)* formulated by Colliat in [Col96], which assumes that data are uniformly distributed. Given *(i)* a data cube L, *(ii)* the histogram $Hist(L)$ computed on L, and *(iii)* a range-SUM query Q against L such that Q overlaps the bucket set $B(Q) = b_0, b_1, , b_{W-1}$ of $Hist(L)$, based on CVA the approximate answer to Q, denoted by $\tilde{A}(Q)$, can be obtained as follows:

$$\tilde{A}(Q) = \sum_{w=0}^{W-1} \frac{\|Q \cap b_w\|}{\|b_w\|} \cdot SUM(b_w)$$

where $SUM(b_w)$ denotes the sum of (OLAP) data cells contained in b_w. Moreover, when the CVA fails (i.e., the target data cube is characterized by skewed, i.e. asymmetric e.g., see [Cuz05], distributions), *outliers* contained in L can seriously decrease the accuracy degree of $\tilde{A}(Q)$, as it has been demonstrated in [CW07], and also recognized in [CDD$^+$01]. In this case, in [CW07] the approximate answer to Q is obtained by means of separately considering the contribution given by outliers involved by Q, and summing-up this contribution to the contribution given by classical linear interpolation. This imposes to separately handling outliers, and originates a different formula for $\tilde{A}(Q)$, which, for the sake of simplicity, here can be modeled as:

$$\tilde{A}(Q) = \sum_{w=0}^{W-1} \left[\frac{\|Q \cap b_w\|}{\|b_w\|} \cdot SUM(b_w) + outlier(Q \cap b_w) \right]$$

where $outlier(R)$ is the set of outliers (of L) involved by the region R.

The latter query evaluation scheme gives good performance when a fixed, single (range) query or, better, a workload QWL of queries of the same nature and with similar geometrical characteristics is considered, as argued in [CW07] and in [BCG01]. According to this experimental evidence, the compression process tries to generate buckets such that the final bucket ranges define a partition of the input data cube L that, as much as possible, accommodates all the queries of QWL, thus improving the accuracy of linear interpolation techniques. To this end, the goal is two-fold: *(i)* obtaining buckets defined on uniform data (i.e., data with low variance), and *(ii)* minimizing the geometric difference between buckets and queries, in a global fashion. This provides a query performance better than the one given by computing the histogram without a fixed query-workload, as demonstrated in [CW07]. The same approach could be extended in order to deal with the issue of evaluating a given single $HRQQ_H$ embedding several range queries. One can think of modeling the set of range queries embedded in QH as the target query-workload, and then adopting the same query evaluation scheme described above. Therefore,

Q_H can be straightforwardly evaluated by iteratively evaluating range queries of its nodes, one at time.

It is worthy noticing that, in this case, linear interpolation techniques fail as ranges of queries of HRQ can be very different one from another, at the same level as well as at different levels of the hierarchy of the target data cube L. This makes the previous query evaluation scheme inefficient, and requires *ad-hoc* solutions able to deal with the complexity and the *difficult* nature of HRQ, in a global fashion. According to the proposed solution, the final histogram-based compressed representation of L, denoted by $\tilde{L}$, is obtained as an intermediate representation given by meaningfully partitioning the bucket domain defined by the different query ranges of HRQ at the same level ℓ of L, for each level ℓ of the L hierarchy. In fact, it is a matter to note that, an arbitrary compression of the data cube could easily origin the undesired situation in which some HRQ could take advantage from the compression process (i.e., retrieved approximate answers have a high degree of accuracy), as the final partition fits data and geometric properties of these HRQ, whereas some other HRQ could be disadvantaged (i.e., retrieved approximate answers have a low degree of accuracy), as the final partition does not fit the abovementioned properties for such HRQ. As a consequence, the idea of generating an *intermediate* compressed representation of L makes sense, as, on the average, a fair final representation is obtained (i.e. retrieved approximate answers have acceptable/good accuracy in most cases). On the other hand, it should be noted that this approach has several conceptual points in common with other multiple-query data processing paradigms, such as those focusing on the view selection and materialization problem, where common sub-expressions of target queries are considered in order to obtain efficient solutions.

On a more practical plane, due to the hierarchical nature of cubes and queries, the compressed data cube $\tilde{L}$ is implemented as a hierarchical multidimensional histogram, denoted by $MQ-Hist(L)$, which is obtained by means of a greedy algorithm, namely `computeMultQHist`, that meaningfully exploits the multi-level and hierarchical nature of the input data cube L, and defines a *top-down compression process* able to accommodate the different objectives of multiple, simultaneous HRQ against L.

2.4.1 The Compression Process

Given: *(i)* a n-dimensional data cube L having P hierarchical levels; *(ii)* the set of m HRQ that must be evaluated against L simultaneously, $SHRQ = \{Q_{H_0}, Q_{H_1}, \cdots, Q_{H_{m-1}}\}$ (recall that, for the sake of simplicity, HRQ in SHRQ have the same depth, P, which is also equal to the number of hierarchical levels of L); *(iii)* the space bound B available for housing the compressed representation of L, $\tilde{L}$, which is implemented by the histogram $MQ-Hist(L)$. The multiple-query compression process we propose is accomplished according to the following multi-step approach:

- for each level ℓ of L, such that ℓ belongs to $[0, P-1]$, starting from the bottom level 0 (i.e., according to a *bottom-up strategy*), generate, for each dimension d of L_ℓ, the *ordered union* of range bounds of queries (modeled as multidimensional points in the L_ℓ ($\equiv L$) multidimensional space) at level ℓ of HRQ in SHRQ along d, thus obtaining the so-called *Multiple-Range (MR)* for the dimension d at level ℓ, denoted by $MR_{d,l}$ first step finally generates, for each cuboid L_ℓ of L, the set of n MR, denoted by $MR(L_\ell) = \{MR_{0,l}, MR_{1,l}, \cdots, MR_{n-1,l}\}$;
- for each level ℓ, generate a *Generalized Partition (GP)* of the cuboid L_ℓ at level ℓ, denoted by $G_\ell(L)$, such that buckets in $G_\ell(L)$ are obtained via *(i)* projecting, for each dimension d of L_ℓ, axes along points of MR in $MR(L_\ell)$, and *(ii)* storing the sum of items they contain the collection of GP (one for each level ℓ of L), denoted by $MQ - Coll(L)$, constitutes the *sketch* of $MQ - Hist(L)$, meaning that from $MQ - Coll(L)$ it is possible to finally obtain $MQ - Hist(L)$ according to our greedy algorithm `computeMulQHist`;
- (algorithm `computeMulQHist`) for each level ℓ, obtain from $G_\ell(L)$ of $MQ - Coll(L)$ the so-called *Multiple-Query Partition (MQP)* of the cuboid L_ℓ, denoted by $P_\ell(L)$, via meaningfully merging buckets in $G_\ell(L)$ with the criterion that $P_\ell(L)$ must be able to fit, at level ℓ, all the different query requirements of HRQ in SHRQ;
- return $MQ - Hist(L)$ via hierarchically combining the $PP_\ell(L)$.

From the described approach, it follows that the most important task of this technique is represented by algorithm `computeMulQHist`, whereas the other steps are quite obvious.

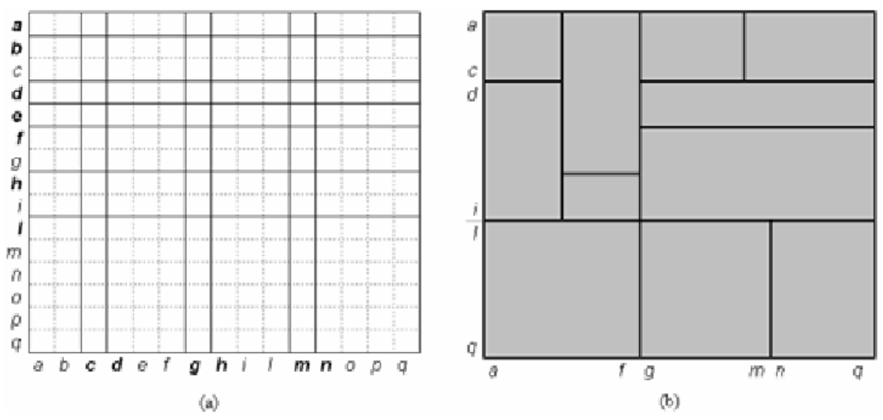


Fig. 2.6. An example of Generalized Partition $G_\ell(L)$ (a) and a possible corresponding $P_\ell(L)$ (b)

First, it is useful to illustrate how MR, GP and MQP are obtained throughout a meaningful example. For the sake of simplicity, consider the following OLAP scenario (see 2.6 (a)): *(i)* a $|d_0| \times |d_1|$ two-dimensional cuboid L_ℓ, such that the domain of d_0 is $Dom(d0) = \{a, b, c, d, e, f, g, h, i, l, m, n, o, p, q\}$, and the domain of d_1 is equal to that of d_0 (i.e., $Dom(d_1) = Dom(d_0)$), being the common-intended lexicographical ordering defined on both of these domains; *(ii)* three range queries, $Q_{i,l}$, $Q_{j,l}$, and $Q_{k,l}$ of distinct HRQ defined on L_ℓ, defined as follows: $Q_{i,l} = \langle [b, d], [c, f] \rangle, Q_{j,l} = \langle [d, e], [d, g] \rangle$, and $Q_{k,l} = \langle [h, i], [m, m] \rangle$. According to the third step of the multiple-query compression process: *(i)* the two MR of L_ℓ are: $MR_{0,l} = \{b, d, e, h, i\}$ and $MR_{1,l} = \{c, d, f, g, m\}$; *(ii)* the GP of L_ℓ, $G_\ell(L)$, is that depicted in 2.6 (a); *(iii)* a possible MQP of L_ℓ, $P_\ell(L)$, is that depicted in 2.6 (b). From 2.6 (a), note that the so-generated buckets of $G_\ell(L)$ are all the buckets that would allow to provide, at level ℓ, approximate answers having the highest accuracy for all HRQ in SHRQ at level ℓ. The same for the other GP of $MQ - Coll(L)$. Obviously, it is not possible to materialize all the buckets of all the GP of $MQ - Coll(L)$, due to the space constraint posed by the input bound B. If this would be the case, it would be easy to finally obtain the histogram $MQ - Hist(L)$ as corresponding to $MQ - Coll(L)$ directly, i.e. as a full materialization of $MQ - Coll(L)$. Being this impossible for reasonable configurations of the input parameters, the adopted strategy is based on obtaining the MQP $P_\ell(L)$ via meaningfully merging buckets in $G_\ell(L)$, thus reducing the overall final size of $P_\ell(L)$ and, as a consequence, the overall final size of $MQ - Hist(L)$, keeping in mind the priority goal of accommodating, as much as possible, the multiple query constraints posed by HRQ in SHRQ. As stated before, this strategy, which is implemented by algorithm `computeMulQHist`, allows us to finally compute $P_\ell(L)$ as a sub-optimal partition of $G_\ell(L)$.

To this end, `computeMulQHist` introduces a *global strategy* and a *local strategy*. The first one deals with the problem of how to explore the overall hierarchical search (bucket) space represented by $MQ - Coll(L)$. The second one deals with the problem of how to explore the search (bucket) space represented by a given $G_\ell(L)$.

First, consider the latter one, which properly realizes the main greedy criterion used to obtain a $P_\ell(L)$ from a given $G_\ell(L)$. The strategy that is inspired by the one adopted by traditional multidimensional histograms (e.g., *MHist* [PI97]), i.e. obtaining final buckets storing as much uniform data as possible via minimizing the skewness among buckets themselves. This, in turn, has beneficial effects on the accuracy of approximate answers computed against the histogram, as widely recognized (e.g., see [Cuz05]). The difference with respect to this approach lies in the fact that traditional histograms operate on the original data cube directly, whereas $MQ - Hist(L)$ is built starting from the bucket space defined by $MQ - Coll(L)$, and, in turn, by each $G_\ell(L)$.

According to these guidelines, in the local computation of `computeMulQHist`, given the GP $G_\ell(L)$, the *most uniform bucket can be greedily selected*, said b_U, among buckets of the overall bucket space of $G_\ell(L)$, and then we explore the neighboring buckets of b_U in search for buckets having a homogeneity close to that of b_U, having fixed a threshold value VU that limits the maximal difference between the homogeneity of b_U and that of its neighboring buckets. To meaningfully support this task, given a bucket b of $G_\ell(L)$, we adopt as homogeneity definition the greatest quadratic distance from the average value of outliers in b, meaning that the less is such a distance, the more is the homogeneity of b. This approach is modeled by the function $unif(b)$, defined as follows:

$$unif(b) = \frac{1}{\max\limits_{i \in Out(b)} \{0, |b[i] - AVG(B)|^2\}}$$

such that *(i)* $Out(b)$ is the set of outliers of b, *(ii)* $b[i]$ is the value of the *i-th* item of b, and *(iii)* $AVG(b)$ is the average value of items in b. Note that the use of the second power to ensure the convergence of the introduced function $unif(\cdot)$.

When a set of neighboring buckets is detected and must be merged in a singleton bucket, said b_M, the criterion is imposed to obtain b_M as a *hyper-rectangular* bucket instead of an arbitrary bucket (i.e., a bucket with irregular shape). This reasonably follows the geometry of arbitrary range queries. In doing this, based on geometrical issues, protruding parts of merged neighboring buckets can be *thrown away* in such a way as to obtain a maximal, internal hyper-rectangular bucket. In turn, the protruding bucket parts are then materialized as new buckets of the GP $G_\ell(L)$, and then the same task is iterated again on the remaining bucket set of $G_\ell(L)$. Comparing with STHoles [4], we observe that [4] proposes a total hierarchical strategy for merging buckets, whereas we propose a planar strategy for merging buckets, but applied to each level of our hierarchical search (bucket) space. Of course, the underlying greedy criterion is different in the two proposals.

For what regards the second aspect of `computeMulQHist` (i.e., the global strategy), the key is to adopt an in- depth visit of $MQ-Coll(L)$ starting from the aggregation ALL (i.e., from the corresponding GP at level P, $G_P(L)$), meaning that, starting from the level P (i.e., the cuboid L_P), when a merged bucket b_M is obtained in the GP $G_\ell(L)$ at level ℓ, the idea is to hierarchically move down to the GP $G_{l+1}(L)$ at level $l + 1$, and consider the collection of buckets of $G_{l+1}(L)$ contained by b_M, and so forth. When the leaf level GP is reached (i.e., the cuboid L_0), we re-start from the aggregation ALL. As said before, the whole process is bounded by the consumption of the storage space B.

Basically, the above-described one defines a top-down approach in the computation of $MQ-Hist(L)$. Without going into details, it should be noted that the in-depth visit approach is in favor of the idea of accommodating a large family of range queries embedded in HRQ rather than range queries referred to a particular level of the logical hierarchy underlying the input data cube L. The rationale of this way to do comes from arguing that, in a typical OLAP scenario, client applications are mainly interested in querying

OLAP data at granularities different from the lowest one [NK01]. To become convinced of this, consider a repository R of sale data (which are particularly suitable to be processed and analyzed by means of OLAP technology) and a data cube L defined on top of R such that L includes, among all, the dimension *Time* with hierarchy H_{Time}. Also, suppose that H_{Time} is organized as follows:

$$H_{Time} : Year \rightarrow Quarter \rightarrow Month \rightarrow Day$$

Even if sale data are available in all the defined temporal granularities (i.e., Year, Quarter, Month, and Day), OLAP client applications typically access and query data at the *Month* or *Quarter* or *Year* granularities mostly [Han05]. This evidence, combined with *(i)* the need of accommodating a large family of queries, and *(ii)* the presence of a bounded storage space (i.e., B), gives raise to the proposed top-down approach.

2.4.2 Experimental Results

In order to test the effectiveness of the proposed technique, experiments have been designed aiming at probing the data cube compression performance (or, equally, the accuracy) of our technique. Experiments involve several aspects ranging from the data layer to query layer, metrics, and comparison techniques.

Data Layer

As regards the data layer of the experimental framework, two different kinds of data cubes have been engineered. The usage of different classes of data cubes allowed to submit our proposed technique to a comprehensive and “rich” experimental analysis, and, as a consequence, carefully test its performance. Data cube classes considered are the following: *(i)* benchmark data cubes, which allow to test the effectiveness of the technique under the stressing of an in-laboratory-built input, and to evaluate the technique against competitor ones on “well-referred” data sets that have been widely used in similar research experiences; *(ii) real data cubes*, which allow to probe the efficiency of this technique against real-life data sets. Experiments are also conducted on synthetic data cubes, which allow us to completely control the variation of input parameters such as the nature of OLAP data distributions. However, experimental results on benchmark and real-life data cubes, typically, are more probing than synthetic ones. For what regards benchmark data sets, it has been considered the popular benchmark *TPC-H*, which is well-known in the Database and Data Warehousing research communities.

By exploiting data generation routines made available at the benchmark Web site, a two-dimensional benchmark data cubes has been built by means of the well-known *Microsoft Analysis Services 2000* OLAP platform. In more detail, a two-domensional datacube $2,000 \times 2,000$ is built and populated with 4M data cells. For what regards real-life data sets, we considered the popular data set *USCensus1990*, and we built a $1,000 \times 1,000$ two-dimensional data cube, still on top of *Microsoft Analysis Services 2000.*

Query Layer

As regards the input of the experimental study, it has been considered *random populations* of synthetic HRQ with range-SUM queries, modeled in the experimental framework by the set SQH. For these queries, the underlying trees have also been obtained randomly with the constraint of "covering" the different cuboids of the target data cube as much as possible (i.e., obtaining GP having a large number of buckets) at, however, a "reasonable" granularity. *Query Selectivity* $\| \cdot \|$, which for OLAP queries is totally equal to the query volume of the range-SUM queries, is the control parameter used to cost the complexity needed to evaluate queries in SQ_H (e.g., see [Cuz06b]).

Metrics

As regards the outcomes of the experimental study, given a population of synthetic HRQ SQ_H, it has been introduced the *Average Relative Error (ARE)* between exact and approximate answers to queries in SQ_H, defined as follows:

$$\overline{\varepsilon}(SQ_H) = \frac{1}{|SQ_H| - 1} \cdot \sum_{j=0}^{|SQ_H|-1} \overline{\varepsilon}(Q_{H_j})$$

such that *(i)* $\overline{\varepsilon}(Q_{H_j}$ is defined as follows:

$$\overline{\varepsilon}(Q_{H_j} = \frac{1}{P} \cdot \sum_{\ell=0}^{P-1} \left[\frac{1}{|queries(\ell)| - 1} \sum_{k=0}^{|queries(\ell)|-1} \varepsilon(Q_k) \right]$$

and *(ii)* $\varepsilon(Q_k)$ is the *Relative Error (RE)* of the range-SUM query Q_k in Q_{H_j} , defined as follows:

$$\varepsilon(Q_k) = \frac{|A(Q_k) - \tilde{A}(Q_k)|}{A(Q_k)}$$

Comparison Techniques

In the experimental study, the compared performances are related to the proposed technique against the following well-known histogram-based techniques for compressing data cubes: *MinSkew* by Acharya et al. [APR99b], *GenHist* by Gunopulos et al. [GKTD00b], and STHoles by Bruno et al. [BCG01]. This because all these techniques are similar each other, and also represent the state-of-the-art for histogram-based data cube compression research. In more detail, having fixed the space budget B (i.e., the storage space available for housing the compressed data cube), it has been derived, for each comparison technique, the configuration of the input parameters that respective authors consider the best in their papers. This ensures a fair experimental analysis, i.e. an analysis such that each comparison technique provides its best performance. Furthermore, for all the comparison techniques, we set the space budget B as equal to the $r\%$ of $size(L)$, being r the compression ratio and $size(L)$ the total occupancy of the input data cube. As an example, $r = 10\%$ (i.e., B is equal to the 10% of $size(L)$) is widely recognized as a reasonable experimental setting (e.g., see [BCG01]).

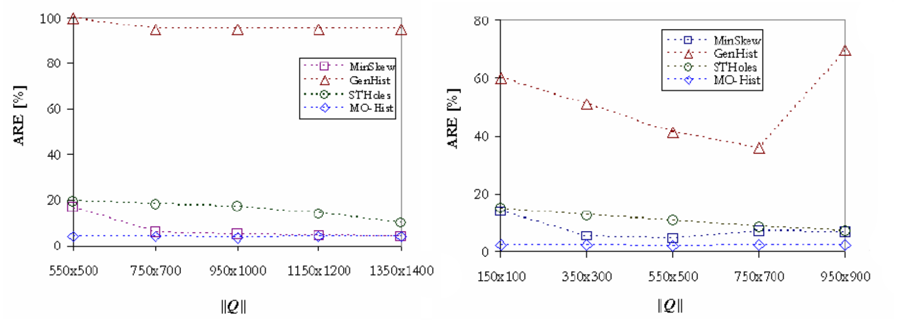


Fig. 2.7. ARE vs query selectivity $\|Q\|$ on the benchmark data cube *TPC-H* (left) and on the real-life data cube *USCensus1990* (right) with $r = 10\%$

2.7 shows experimental results related to the percentage variation of ARE on both benchmark and real-life data cubes with respect to the selectivity of queries in SQ_H. This allows us to measure the quality of compression techniques, i.e. their capability of introducing low query approximation errors. 2.8 shows the results for the same experiment when ranging r on the interval $[5, 20]$ (i.e., B on the interval $[5, 20]\%$ of $size(L)$), and fixing the selectivity of range-SUM queries to $||Q|| = 750 \times 700$ for the benchmark data cube *TPC-H*, and to $\|Q\| = 350 \times 300$ for the real-life data cube *USCensus1990*. This allows to measure the scalability of compression techniques, which is a critical aspect in approximate OLAP query answering engines (e.g., see [Cuz05]). From the analysis of the set of experimental results on two-dimensional benchmark and real-life data cubes, it follows that compression performance of $MO-Hist(L)$ outperforms those of comparison techniques. Also, the proposed technique ensures a better scalability with respect to that of comparison techniques when ranging the size of the storage space B available for housing the compressed representation of the input data cube.

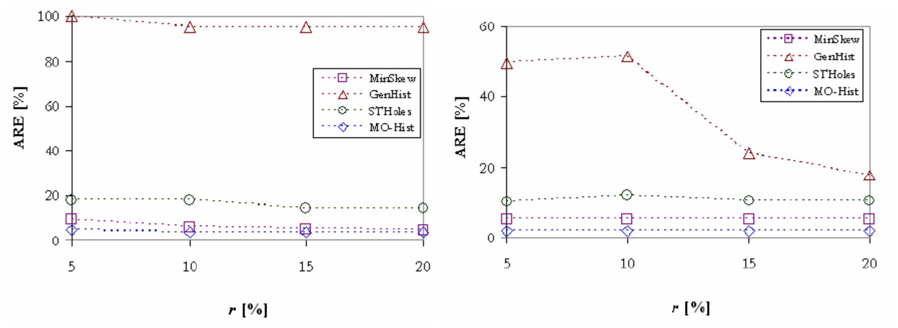


Fig. 2.8. ARE vs compression ratio r on the benchmark data cube *TPC-H* with $\|Q\| = 750 \times 700$ (left) and on the real-life data cube *USCensus1990* with $\|Q\| = 350 \times 300$ (right)

3

Privacy and OLAP

3.1 Introduction

The problem of ensuring the privacy and security of OLAP data cubes ([GCB$^+$97]) arises in several fields ranging from advanced *Data Warehousing (DW)* and *Business Intelligence (BI)* systems to sophisticated *Data Mining (DM)* tools. In DW and BI systems, decision making analysts aim at avoiding that *malicious users* access perceptive ranges of multidimensional data in order to infer *sensitive knowledge*, or attack corporate data cubes via violating user rules, grants and revokes. In DM tools, domain experts aim at avoiding that malicious users infer critical-for-the-task knowledge from authoritative DM results such as frequent item sets, patterns and regularities, clusters, and discovered association rules. In more detail, the former application scenario (i.e., DW and BI systems) deals with both the privacy preservation and the security of data cubes, whereas the latter one (i.e., DM tools) deals with *privacy preserving OLAP* issues solely.

Specifically, *privacy preservation of data cubes* refers to the problem of ensuring the privacy of data cube cells (and, in turn, that of queries defined over collections of data cube cells), i.e. hiding sensitive information and knowledge during data management activities, according to the general guidelines drawn by Sweeney in her seminar paper (k-anonimity), whereas *access control* issues refer to the problem of ensuring the security of data cube cells, i.e. restricting the access of unauthorized users to specific sub-domains of the target data cube, according to well-known concepts studied and assessed in the context of DBMS security.

Nonetheless, it is quite straightforward foreseeing that these two even distinct aspects should be meaningfully integrated in order to ensure both the privacy and security of complex data cubes, i.e. data cubes built on top of complex data/knowledge bases.

During last years, these topics have became of great interest for the Data Warehousing and Databases research communities, due to their exciting theoretical challenges as well as their relevance and practical impact in modern real-life OLAP systems and applications. On a more conceptual plane, theoretical aspects are mainly devoted to study how probability and statistics schemes as well as rule-based models can be applied in order to efficiently solve the above introduced problems. On a more practical plane, researchers and practitioners aim at integrating convenient privacy preserving and security solutions within the core layers of commercial OLAP server platforms.

Basically, to tackle deriving privacy preservation challenges in OLAP, researchers have proposed models and algorithms that can be roughly classified within two main classes: *restriction-based* techniques, and *data perturbation* techniques. First ones propose limiting the number of query kinds that can be posed against the target OLAP server. Second ones propose perturbing data cells by means of random noise at various levels, ranging from schemas to queries. On the other hand, access control solutions in OLAP are mainly inspired by the wide literature developed in the context of controlling accesses to DBMS, and try to adapt such schemes in order to control accesses to OLAP systems.

Handling sensitive data, which falls in privacy preserving issues, is common in many real-life application scenarios. For instance, consider a government agency that collects information about client applications/users for a specific e-government process/task, and then makes this information available for a third-party agency willing to perform market analysis for business purposes. In this case, preserving sensitive data of client applications/users and protecting their utilization from malicious behaviors play a leading role. It should be taken into account that this scenario gets worse in OLAP systems, as the interactive nature of such systems naturally encourages malicious users to retrieve sensitive knowledge by means of inference techniques ([WJW04],[WWJ04]) that, thanks to the wide availability of OLAP tools and operators ([Han05]), can reach an high degree of effectiveness and efficiency.

Theoretical background of privacy preserving issues in OLAP relies on research experiences in the context of statistical databases ([Sho97]), where these issues have been firstly studied. In statistical databases, this problem has been tackled by means of Statistical *Disclosure Control (SDC) techniques* (Domingo-Ferrer, 2002), which propose achieving the *privacy preservation* of data via trade-offing the accuracy and privacy of data. The main idea of such an approach is that of admitting the need for data provisioning while, at the same time, the need for privacy of data. In fact, full data hiding or full data camouflaging are both useless, as well as publishing completely-disclosed data sets. Therefore, balancing accuracy and privacy of data is a reasonable solution to this challenge. In this context, two meaningful measures for evaluating the accuracy and privacy preservation capabilities of an arbitrary method/technique have been introduced. The first one is referred as *Information Loss (IL)*. It allows us to estimate the lost of information (i.e., the *accuracy decrease*) due to a given privacy preserving method/technique. The second one is the *Disclosure Risk (DR)*. It allows to estimate the risk of disclosing sensitive data due to a given privacy preserving method/technique.

[DFK$^+$01] introduces two metrics for probabilistically evaluating IL and DR. Given a numerical attribute A that can assume a value w with probability P_w, such that D_w is the domain of w (i.e., the set of all the values that A can assume), a possible metrics of IL is given by the *Data Utility (DU)*, which is defined as follows:

$$DU(w) = \frac{|D_w|}{\sum_w P_w \cdot (w-1)^2} \tag{3.1}$$

where $|D_w|$ denotes the cardinality of D_w. It should be noted that DU and IL are inversely proportional, i.e. the more is IL the less is DU, and, conversely, the less is IL the more is DU.

Different formulations exist. For instance, [SLXN06] introduce the so-called *accuracy factor* $F_{a,Q}$ of a given query Q against a data cube D, i.e. the relative accuracy decrease of the approximate answer to Q, denoted by $\tilde{A}(Q)$, which is evaluated on the synopsis data cube D obtained from D by means of *perturbation-based techniques* (presented next), with respect to the exact answer to Q, denoted by $\tilde{A}(Q)$, which is evaluated on the original data cube $\tilde{D}$. $F_{a,Q}$ is defined as follows:

$$F_{a,Q} = 2^{-\left|\frac{A(Q)-\tilde{A}(Q)}{A(Q)}\right|} \tag{3.2}$$

With regards to DR, [DFK$^+$01] consider the *probability with respect to the malicious user* that A can assume the value w, denoted by P_w^U, and introduce the following metrics that models DR in terms of the reciprocal of the *information entropy*, as follows:

$$DR(w) = \frac{1}{-\sum_w P_w^U \cdot log(P_w^U)} \tag{3.3}$$

Indeed, being impossible to estimate the value of P_w , as one should know all the information/knowledge held by the malicious user, in ([DFK$^+$01]) the conditional version of 3.3 is proposed as follows:

$$DR(w) = \frac{1}{-\sum_w p(w|u) \cdot log_2 p(w|u)} \tag{3.4}$$

such that $p(w|u)$ denotes the *conditional probability* that the actual value of A is w while the value known by the malicious user is u.

Just like for IL, different formulations for measuring DR exist. [SLXN06] introduce the so-called *privacy factor* $F_{p,D}$ of a given data cube D with respect to the corresponding perturbation-based synopsis data cube D . Fp,D is defined as follows:

$$F_{p,D} = \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{|\tilde{C}_i - C_i|}{|C_i|} \tag{3.5}$$

such that C_i denotes a data cell of the data cube D, and $\tilde{C}_i$ the corresponding perturbed data cell of the synopsis data cube $\tilde{D}$.

According to research results presented in ([DFK$^+$01]), accuracy and privacy of a privacy preserving technique are related and must be traded-off. Intuitively enough, an increase of one of these properties causes a correlated decrease of the other one. Also, it is a matter to notice that having maximum accuracy implies a very high DR while, conversely, minimum accuracy implies minimum DR. On the other hand, accuracy cannot be minimized, and DR cannot be maximized. As we will describe in next Section, most of privacy preserving OLAP techniques of the active literature are based on this terminology, and on the fundamental idea of trading-off accuracy and privacy.

For what regards the background of security issues, actual literature focuses on access control techniques, as discussed above. Access control has a quite long history in DBMS, where the goal is protecting data objects from unauthorized accesses. In this context, an authorization is modeled as a triple: $\langle Object, Subject, +/-Action\rangle$, such that *(i) Object* is the data object to be protected, *(ii) Subject* is the user/application accessing the object *Object*, and *(iii) Action* is the operation that the user/application Subject can or cannot (i.e., +/-) perform on the object *Object*. Typically, read-only operations are controlled, since data updates are indeed allowed to few users/user-groups only.

Because of (very) different data models, the main difference between access control issues in DBMS and OLAP systems is represented by the nature of data objects. In DBMS, an object can be a data table, a query or a record. This allows to achieve very precise authorization mechanisms, as singleton records or partitions of data tables (they may be vertical or horizontal) can be handled. Indeed, it is possible to claim that access control techniques take great advantages from the *flexibility* of DBMS models and schemas. When OLAP systems are considered, models and schemas are characterized by *dimensionality and multi-resolution of data*, and access control mechanisms need to become more sophisticated in order to prevent malicious user attacks. Target data objects are represented by data cubes, which can be generally thought as collections of *cuboids*, one for each combination of hierarchy levels. Also, dimensional hierarchies pose additional challenges, as malicious users can successfully exploit the multi-resolution data models defined by dimensional hierarchies in order to devise very effective *knowledge inference techniques* able to hierarchically browse the structure of the cube with the goal of discovering aggregations computed over sensitive ranges of data. Also, rich OLAP tools and operators ([Han05]), such as *roll-up, drill-down and slice & dice*, represent "sophisticate instruments" in the hands of malicious users, as they can be used to extract sensitive knowledge from a secured cuboid at a given level ℓ starting from disclosed cuboids at levels different from ℓ.

To overcome security breaches like those described above, access control techniques for OLAP data cubes usually apply a restriction-based approach, and limit the set of cuboids that can be accessed by external applications/users. Nevertheless, even in this case a trade-off strategy must be

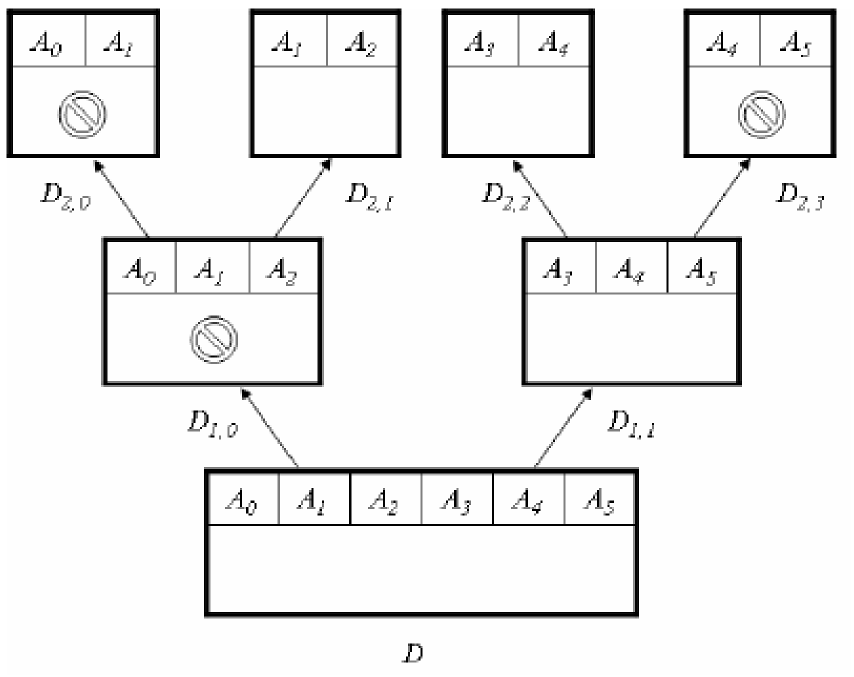


Fig. 3.1. Access control mechanism on a ROLAP data cube

devised, as securing a large number of cuboids can become useless in real-life OLAP scenarios. All considering, as studied by Wang et al. in ([WJW04]; [WWJ04]), the inference problem in OLAP introduces more probing issues rather than precursor scientific areas related to inference issues in statistical databases ([DS83]).

To give an example on a simple access control mechanism in Relational OLAP (ROLAP) data cubes (i.e., data cubes stored in form of tables of a RDBMS ([Han05])), consider 3.1, where a data cube D with the related cuboid lattice is depicted. In this case, the cuboids $D_{1,0}$, $D_{2,0}$, and $D_{2,3}$ are secured to authorized applications/users only and forbidden to unauthorized ones.

3.2 Privacy Preserving Techniques in OLAP

Privacy Preserving OLAP (PPOLAP) ([AST05]) is a specialized case of *Privacy Preserving Data Mining (PPDM)* ([AS00]). While PPDM concerns with the privacy of data during DM activities (e.g., clustering, classification, pattern discovery, association rule discovery etc), PPOLAP deals with the problem of preserving the privacy of data cells of a given data cube during typical

OLAP activities such as performing classical operators (e.g., roll-up and drill-down) or evaluating complex OLAP queries (e.g., *range-* ([HAMS97]), *top-k* ([XHCL06]), and *iceberg* ([FSGM$^+$98]) queries). With respect to PPDM, PPOLAP introduces more *semantics* into the privacy preservation due to its well-known knowledge-intensive tools such as multidimensionality and multi-resolution of data, and hierarchies.

In the following, two kinds of privacy preserving techniques in OLAP are reviewed (i.e., restriction-based and perturbation-based techniques) introduced in the previous Section.

Restriction-based techniques limit the queries that can be posed to the OLAP server in order to preserve the privacy of data cells. This problem is related to the issue of auditing queries in statistical databases, which consists in analyzing the past (answered) queries in order to determine whether these answers can be composed by a malicious user to infer sensitive knowledge in the form of answers to forbidden queries. Therefore, in order to understand which kinds of queries must be forbidden, a restriction-based technique needs to audit queries posed to the target data (e.g., OLAP) server during a given interval of time. Auditing queries in statistical databases is the conceptual and theoretical basis of auditing queries in OLAP systems.

Interesting auditing techniques for queries against statistical databases have been proposed by [DJL79], which introduce a model for auditing average and median queries, and [CO82], which propose a technique for handling the past history of SUM queries in order to reduce the sequence of answered queries to privacy preservation purposes. Also, [CO82] describe how to check the *compromisability* of the underlying statistical database when using the reduced sequence. The proposed auditing technique is called *Audit Expert*.

More recently, few approaches focusing on the problem of auditing techniques for OLAP data cubes and queries appeared. Among all, it is useful to recall: *(i)* the work of [ZZC04], which propose an interesting *information theoretic* approach that simply counts the number of cells already covered to answer previous queries in order to establish if a new query should be answered or not; *(ii)* the work of [MMM06], which introduce a novel notation for auditing range-SUM queries (i.e., an OLAP-like class of queries) against statistical databases making use of *Integer Linear Programming (ILP)* tools for detecting if a new range-sum query can be answered safely.

Perturbation-based techniques add random noise at various levels of the target database, ranging from schemas, like in ([Sch81]), to query answers, like in (Beck, 1980).

[AST05] first propose the notion of PPOLAP. They define a PPOLAP model over data partitioned across multiple clients using a randomization approach on the basis of which *(i)* clients perturb tuples which with they participate to the partition in order to gain row-level privacy, and *(ii)* server is capable of evaluating OLAP queries against perturbed tables via reconstructing original distributions of attributes involved by such queries. In ([AST05]), authors demonstrate that the proposed approach is safe against privacy breaches.

[HZW$^+$05] propose a different approach to preserve the privacy of OLAP data cubes. They argue that hiding parts of data that could cause inference of sensitive cuboids is enough in order to achieve the notion of “secure” data cubes. While a strengthness point of the proposed approach is represented by its simplicity, authors do not provide sufficient experimental analysis to prove in which measure the data hiding phase affects the target OLAP server.

[SLXN06] propose a random data distortion technique, called zero-sum method, for preserving secret information of individual data cells while providing accurate answers to range-queries over original aggregates. Roughly speaking, data distortion consists in iteratively altering the values of individual data cells of the target data cube in such a way as to maintain the marginal sums of data cells along rows and columns of the data cube equal to zero. This ensures the privacy of individual data cells, and the correctness of answers to range-queries.

Due to different, specific motivations, both restriction-based and perturbation-based techniques are ineffective in OLAP. Specifically, restriction-based techniques cannot be applied to OLAP systems since the nature of such systems is intrinsically *interactive*, and based on a wide set of operators and query classes. On the other hand, perturbation-based techniques cannot be applied in OLAP systems since they introduce excessive computational overheads when executed on massive data cubes.

3.3 Security techniques in OLAP

The problem of security control methods has been widely studied in the context of statistical databases, and it has produced a wide and consolidate literature ([AW89]) that, in turn, is inspiring actual initiatives for securing OLAP data cubes, i.e. limiting their access to authorized applications/users only.

As stated in previous sections, the main idea of these approaches is devising access control schemes that establish how applications/users must access multidimensional data on the basis of grants and revokes ([GW76]), roles ([SCFY96]), and authorization rules ([JSSS01]).

Following the above-mentioned pioneristic approaches, some preliminary, sporadic studies in the context of securing data warehouses ([Bha00]) and data cubes ([PP00]) have been appeared in literature subsequently. While these works are clearly in their initial stages, they have inspired most part of actual research effort in the context of access control schemes for OLAP data cubes. ([WJW04]; [WWJ04]) represent the state-of-the-art for access control schemes in OLAP. They propose a novel technique for limiting inference breaches in OLAP systems via detecting cardinality-based sufficient conditions over cuboids, in order to make data cubes safe with respect to malicious users. Specifically, the proposed technique combines access control and inference control techniques ([DS83]), being *(i)* first one based on the hierarchical nature of data cubes in terms of cuboid lattice and multi-resolution of data, and *(ii)* second one based on directly applying restriction to coarser aggregations of data cubes, and then removing remaining inferences that can be still derived.

3.4 A Robust Sampling-Based Technique for Privacy Preservation in OLAP Environment

Beyond effectiveness and efficiency limitations, actual proposals lack of a rigorous theoretical foundation, as they do not consider any so-called privacy *OLAP notion*. In other words, while these proposals focus the attention on the privacy of data cells, they completely neglect to introduce a rigorous notion of privacy in their research. It is a matter of fact to note that, contrary to the trend of actual privacy preserving OLAP proposals, similar initiatives in the context of privacy preserving Databases [WJW04] and Data Mining [AS00] instead introduce a proper notion of privacy for their data processing goals, i.e. they formally define what privacy means in their research. A positive side-effect of such an approach is represented by the amenity of devising properties and theorems on top of the theoretical framework founding on the privacy notion.

Starting from these considerations, it is possible to consider a a robust sampling-based framework for computing privacy preserving data cubes at a provable computational cost. Contrary to state-of-the-art initiatives, in our framework we introduce a meaningful privacy OLAP notion that considers the privacy of OLAP aggregates, and, by adopting this notion as theoretical baseline, we devise a theoretical framework that allows us to nicely treat privacy preservation of data cubes. In particular, due to typical OLAP data cube processing requirements, which usually involve tight computational bounds dictated by enormous sizes and high dimension number, it is useful to consider the constraint B that imposes to adopt the well-known data cube compression paradigm (e.g., [Cuz05], [HZW$^+$05]). Basically, techniques adhering to this paradigm, called *approximate query answering* techniques, propose to compute compressed representations of data cubes in order to mitigate computational overheads deriving from evaluating resource-intensive OLAP queries. According to the proposed methodology, input queries are issued against the compressed data cube instead of the original one, thus obtaining approximate answers whose introduced query error is perfectly tolerable for OLAP analysis goals.

Given an input data cube A and the space bound B, the main goal of the proposed framework is that of computing the sampling-based synopsis data cube A, whose aggregations are obtained via satisfying the so-called privacy constraint. In particular, the privacy constraint requires that approximate answers over the synopsis data cube embed a certain degree of privacy, which is measured by means of a meaningful privacy metrics, and is bounded by a given privacy threshold determined by application-oriented requirements. The metric-based approach for handling privacy of data is well-established in the community, due to its flexibility and nice theoretical properties. To compute the synopsis data cube A, the so-called *privacy grid* has been introduced, which is a grid-based partition of A. The privacy grid allows to meaningfully exploit the multi-resolution nature of OLAP data, and hence obtain an effective information gain during the computation of A. In this respect, in this framework the granularity of the privacy grid (i.e., the size of its elementary cell) is meaningfully chosen as an adequately-small fraction of the selectivity of queries populating typical query-workloads posed against A. Note that this selectivity can be easily gathered thanks to popular active/monitoring components that one can find in conventional OLAP server platforms. Also, since a compression is introduced (due to sampling), the experimental assessment tests the degree of approximation of retrieved answers, as supporting privacy of answers without considering the accuracy of answers is useless. Results clearly show that the framework is able to provide privacy preserving answers that simultaneously retain a good degree of approximation. Therefore, the underlying accuracy constraint is also satisfied, beyond the (main) privacy constraint.

The key idea in [CRS08] is based on a robust sampling-based framework for privacy preserving OLAP, and it provides a comprehensive experimental evaluation of this framework on synthetic data cubes. These data cubes allow us to easily control all the functional characteristics of a cube (e.g., dimension number, size, sparseness coefficient etc), thus leading to the achievement of a reliable and multi-perspective experimental assessment. Performance of the proposed framework is also compared with the one of the method *Zero-Sum* [SLXN06], which can be reasonably considered as the state-of-the-art for perturbation-based privacy preserving OLAP techniques, under several perspectives of analysis that encompass *(i)* the quality of the final synopsis data cube in accomplishing the privacy and accuracy constraints, *(ii)* the effectiveness of the final synopsis data cube in providing retrieved approximate answers having the desired degree of privacy, and, finally, *(iii)* the sensitivity of the final synopsis data cube under the ranging of the space bound constraint, which is a critical parameter of our framework. Specifically, the study of the latter property confirms the robustness of this framework, i.e. its low dependency on configuration parameters. Experimental results underline the benefits due to the privacy preserving OLAP framework, and state that performance of this framework outperforms the one of the comparison method *Zero-Sum.*

3.4.1 Theoretical Model

Using definition introduced in 2 a data cube A is a tuple $\mathcal{A} = \langle \mathcal{D}, \mathcal{L}, \mathcal{H}, \mathcal{M} \rangle$, such that: *(i)* D is the data domain of A containing (OLAP) data cells, which are the elementary aggregations of A computed against the relational data source S; *(ii)* L is the set of dimensions of A, i.e. the functional attributes with respect to which the underlying OLAP analysis is defined (in other words, L is the set of attributes with respect to which relational tuples in S are aggregated); *(iii)* H is the set of hierarchies related to the dimensions of $\mathcal{A}$, i.e. hierarchical representations of the functional attributes shaped in the form of general trees; *(iv)* $\mathcal{M}$ is the set of measures of $\mathcal{A}$, i.e. the attributes of interest for the underlying OLAP analysis (in other words, $\mathcal{M}$ is the set of attributes with respect to which SQL aggregations stored in data cells of $\mathcal{A}$ are computed). Given these definitions, *(i)* $|\mathcal{L}|$ denotes the number of dimensions of $\mathcal{A}$, *(ii)* d denotes a generic dimension of $\mathcal{A}$, *(iii)* $|d|$ the cardinality of d, and *(iv)* $\mathcal{H}(d)$ the hierarchy of d. Finally, for the sake of simplicity, let assume data cubes having a unique measure (i.e., $|\mathcal{M}| = 1$). However, extending schemes, models and algorithms proposed to deal with data cubes having multiple measures (i.e., $|\mathcal{M}| > 1$) is straightforward.

Given an $|\mathcal{L}|$-dimensional data cube $\mathcal{A}$, an *m-dimensional* range-query $\mathcal{Q}$ against $\mathcal{A}$, with $m \leq |\mathcal{L}|$, is a tuple $\mathcal{Q} = \langle R_{k_0}, R_{k_1}, \ldots, R_{k_{m-1}}, A \rangle$, such that: *(i)* R_{k_i} denotes a *contiguous* range defined on the dimension d_{k_i} of $\mathcal{A}$, with k_i belonging to the range $[0, |\mathcal{L}|1]$, and *(ii)* A is a SQL aggregation operator. Applied to $\mathcal{A}$, $\mathcal{Q}$ returns the A-based aggregation computed over the set of data cells in $\mathcal{A}$ contained within the multidimensional sub-domain of $\mathcal{A}$ bounded by ranges $R_{k_0}, R_{k_1}, \ldots, R_{k_{m-1}}$. Range-SUM queries, which return the SUM of the involved data cells, are trendy examples of range queries. In this framework, range-SUM queries are considered as SUM aggregations, which are very popular in OLAP, and efficiently support other SQL aggregations (e.g., COUNT, AVG etc) as well as summarized knowledge extraction from massive amounts of data.

Given a query $\mathcal{Q}$ against a data cube $\mathcal{A}$, the query region of $\mathcal{Q}$, denoted by $R(\mathcal{Q})$, is defined as the sub-domain of $\mathcal{A}$ bounded by ranges $R_{k_0}, R_{k_1}, \ldots, R_{k_{m-1}}$ of $\mathcal{Q}$.

Given an *n-dimensional* data domain $\mathcal{D}$, the *volume* of $\mathcal{D}$, denoted by $\|\mathcal{D}\|$, is defined as follows: $\|\mathcal{D}\| = |d_0| \times |d_1| \times \ldots \times |d_{n-1}|$, such that $|d_i|$ is the cardinality of the dimension d_i of $\mathcal{D}$. This definition can also be extended to a multidimensional data cube $\mathcal{A}$, thus introducing the volume of $\mathcal{A}$, $\mathcal{A}$, and to a multidimensional range query $\mathcal{Q}$, thus introducing the volume of $\mathcal{Q}$, $\mathcal{Q}$. The latter parameter is also recognized-in-literature as the *selectivity* of $\mathcal{Q}$.

Given an $|\mathcal{L}|$-dimensional data cube $\mathcal{A}$, the privacy grid ($\mathcal{P}(\mathcal{A})$ of $\mathcal{A}$ is a tuple $\mathcal{P}(\mathcal{A}) = \langle \Delta\ell_0, \Delta\ell_1, \ldots, \Delta\ell_{|\mathcal{L}|-1} \rangle$ such that $\Delta\ell_k$ is a range partitioning the dimension d_k of $\mathcal{A}$, with k belonging to $[0, |\mathcal{L}|1]$, in a $\Delta\ell_k$-based (one-dimensional) partition. By combining the partitions along all the dimensions of $\mathcal{A}$, it is possible to finally obtain $\mathcal{P}(\mathcal{A})$ as a regular partition of R($\mathcal{A}$) (the multidimensional region associated to $\mathcal{A}$) composed by the so-called *grid regions* $\mathcal{R}_{\mathcal{P}(\mathcal{A}),k} = [\Delta\ell_{0,k}; \Delta\ell_{1,k}; \ldots; \Delta\ell_{|\mathcal{L}|-1,k}]$. Formally, $\mathcal{P}(\mathcal{A})$ can also be defined as a *collection* of (grid) regions, i.e.

$$\mathcal{P}(\mathcal{A}) = \{\mathcal{R}_{\mathcal{P}(\mathcal{A},0}, \mathcal{R}_{\mathcal{P}(\mathcal{A},1}, \ldots, \mathcal{R}_{\mathcal{P}(\mathcal{A},|\mathcal{P}(\mathcal{A}|-1}\}$$

.

As *accuracy metrics* for answers to queries, the relative query error has been used between exact and approximate answers, which is a well-recognized-in-literature measure of quality for approximate query answering techniques in OLAP. Formally, given a query $\mathcal{Q}$, $A(\mathcal{Q})$ is the exact answer to $\mathcal{Q}$ (i.e., the answer to $\mathcal{Q}$ evaluated against the original data cube $\mathcal{A}$), and as $\tilde{A}(\mathcal{Q})$ the approximate answer to $\mathcal{Q}$ (i.e., the answer to $\mathcal{Q}$ evaluated against the synopsis data cube $\mathcal{A}'$).

Therefore, the relative query error $E_Q(\mathcal{Q})$ between $A(\mathcal{Q})$ and $\tilde{A}(\mathcal{Q})$ is defined as follows:

$$E_Q(\mathcal{Q}) = \frac{|A(\mathcal{Q}) - \tilde{A}(\mathcal{Q})|}{A(\mathcal{Q})}$$

Since the challenging problem is related to ensure the privacy preservation of OLAP aggregations, this privacy metrics takes into consideration how sensitive information can be discovered from aggregate data, and tries to contrast this possibility. To this end, it is important to first study how sensitive aggregations can be discovered from the knowledge about exact answers, and metadata about data cubes and queries.

Starting from the knowledge about the target data cube $\mathcal{A}$ (e.g., range sizes, OLAP hierarchies etc), and the knowledge about a given query $\mathcal{Q}$ (i.e., the volume of $\mathcal{Q}$, $\mathcal{Q}$, and the exact answer to $\mathcal{Q}$, A($\mathcal{Q}$)), it is possible to infer knowledge about sensitive ranges of data contained within $R(\mathcal{Q})$. For instance, it is possible to derive the average value of the contribution throughout which each elementary data cell of $\mathcal{A}$ within $R(\mathcal{Q})$ contributes to $A(\mathcal{Q})$. This quantity is named as *singleton aggregation* of $\mathcal{Q}$, denoted by $I(\mathcal{Q})$. $I(\mathcal{Q})$ is defined as follows:

$$I(\mathcal{Q}) = \frac{A(\mathcal{Q})}{\|\mathcal{Q}\|}$$

It is easy to understand that, in turn, starting from the knowledge about $I(\mathcal{Q})$, it is possible to progressively discover aggregations of larger ranges of data within $R(\mathcal{Q})$, rather than those stored within the elementary data cell, thus inferring further knowledge.

Secondly, it is interesting to study how OLAP client applications can discover sensitive aggregations from the knowledge about approximate answers, and, similarly to the previous case, from the knowledge about data cube and query metadata. Starting from the knowledge about the synopsis data cube $\mathcal{A}$, and the knowledge about a given query $\mathcal{Q}$, it is possible to derive an *estimation* on $I(\mathcal{Q})$, denoted by $\tilde{I}\mathcal{Q}$) , as follows:

$$\tilde{I}(\mathcal{Q}) = \frac{\tilde{A}(\mathcal{Q})}{S(\mathcal{Q})}$$

such that $S(\mathcal{Q})$ is the number of samples extracted from R($\mathcal{Q}$) to compute $\mathcal{A}'$ (note that $S(\mathcal{Q}) < \|\mathcal{Q}\|$). The relative difference between $I(\mathcal{Q})$ and $\tilde{I}(\mathcal{Q})$, named as *relative inference error* and denoted by $E_I(\mathcal{Q})$, gives us a metrics for the privacy of $\tilde{A}(\mathcal{Q})$, defined as follows:

$$E_I(\mathcal{Q}) = \frac{|I(\mathcal{Q}) - \tilde{I}(\mathcal{Q})|}{I(\mathcal{Q})}$$

Indeed, while OLAP client applications are aware about the definition and metadata of both the target data cube and queries, the number of samples $S(\mathcal{Q})$ for each query $\mathcal{Q}$ is not disclosed to them. As a consequence, in order to model this facet of the framework, the key idea is to introduce the *user-perceived singleton aggregation*, denoted by $\tilde{I}_U(\mathcal{Q})$, which is the *effective* singleton aggregation perceived by external applications on the basis of the knowledge made available to them. $\tilde{I}_U(\mathcal{Q})$ is defined as follows:

$$\tilde{I}_U(\mathcal{Q}) = \frac{\tilde{A}(\mathcal{Q})}{\|\mathcal{Q}\|}$$

Based on $\tilde{I}_U(\mathcal{Q})$, the definition of the *relative user-perceived inference error* $E_I^U(\mathcal{Q})$, as follows:

$$E_I^U(\mathcal{Q}) = \frac{|I(\mathcal{Q}) - \tilde{I}_U(\mathcal{Q})|}{I(\mathcal{Q})}$$

It is trivial to demonstrate that $\tilde{I}_U(\mathcal{Q})$ provides a better estimation of the singleton aggregation of $\mathcal{Q}$ rather than the one provided by $\tilde{I}\mathcal{Q}$) , as $\tilde{I}_U(\mathcal{Q})$ is evaluated with respect to all the items contained within $R(\mathcal{Q})$ (i.e., $\|\mathcal{Q}\|$), whereas $\tilde{I}\mathcal{Q}$) is evaluated with respect to the number of samples extracted from $R(\mathcal{Q})$ (i.e., $S(\mathcal{Q})$). In other words, $\tilde{I}_U(\mathcal{Q}$) is an *upperbound* for $\tilde{I}(\mathcal{Q})$. Therefore, $\tilde{I}(\mathcal{Q})$ is considered to compute the synopsis data cube, whereas $\tilde{I}_U(\mathcal{Q})$ is considered to model inference issues on the OLAP client application side.

The *privacy OLAP notion* is built upon $I(\mathcal{Q})$. According to this approach, the final goal is *maximizing the relative inference error* $E_I^U(\mathcal{Q})$, as this condition means-in-practice that OLAP client applications retrieve from the synopsis data cube $\mathcal{A}'$ "sampled" singleton aggregations (i.e., $\tilde{I}_U(\mathcal{Q})$ that are very different from the "real" singleton aggregations (i.e., $I(\mathcal{Q})$ of the original data cube $\mathcal{A}$ (i.e., $\tilde{I}_U(\mathcal{Q}) \neq I(\mathcal{Q})$)) This way, the privacy of OLAP aggregations of $\mathcal{A}$ is preserved.

Similarly to related proposals appeared in literature recently [SLXN06], in this framework has been introduced the privacy threshold Φ_I that gives a *lower bound* for the relative user-perceived inference error $E_I^U(\mathcal{Q})$ due to evaluating a given query $\mathcal{Q}$ against the synopsis data cube $\mathcal{A}'$. Therefore, the privacy constraint can formally be modelled as follows: $E_I^U(\mathcal{Q}) \geq \Phi_I$. Above all, Φ_I allows to meaningfully model and treat privacy OLAP issues at a rigorous *mathematical/statistical* plane.

Application-wise, Φ_I is set by OLAP client applications, and the privacy preserving OLAP engine must accomplish this requirement accordingly, while also ensuring the accuracy of approximate answers. The issue of determining how to set this (user-defined) parameter is a non-trivial engagement. Intuitively enough, we set this parameter in terms of a percentage value, as this way its semantics can be immediately captured by OLAP client applications, which indeed do not know the exact values of singleton aggregations. Without going into more details, this approach is similar to the one adopted by a plethora of research experiences in the context of approximate query answering techniques, which make use of a widely-accepted query error threshold (belonging to the interval $[15, 20]\%$) as reference for the accuracy of answers (e.g., see [Cuz05]).

3.4.2 Computing the Privacy Preserving Synopsis Data Cube

Algorithm `computeSPPDataCube` implements the technique above for computing the synopsis data cube $\mathcal{A}'$, given the following input parameters: *(i)* the target data cube $\mathcal{A}$; *(ii)* the space bound $\mathcal{B}$; *(iii)* the integer parameter δ; *(iv)* the privacy threshold Φ_I; *(v)* the typical query-workload QWL on $\mathcal{A}$. Basically, `computeSPPDataCube` is a multi-step algorithm. In next sections each step will be described separately.

The Privacy Grid

The first step of `computeSPPDataCube` consists in computing the privacy grid $\mathcal{P}(\mathcal{A})$ for $\mathcal{A}$. This task finally aims at determining the range $\Delta\ell_k$ for each dimension d_k of $\mathcal{A}$, with k belonging to $[0, |\mathcal{L}|1]$. In turn, this allows to obtain the volume of grid regions $\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}$, $\|\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}\|$ by regularly partitioning $\mathcal{A}$. $\Delta\ell_k$ is determined as an adequately-small fraction of the selectivity of queries in QWL. Let $S_\mathcal{T}$ be the "typical" selectivity of queries in QWL, and $\|\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}\|$ be the volume of $\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}$. If $\|\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}\| \ll S_mathcalT$, then $\mathcal{A}$ can be computed by using the grid region as the elementary reasoning unit, and adopting a resolution level lower than the resolution level of queries against $\mathcal{A}$. This allows to achieve an effective information gain during the computation of $\mathcal{A}'$. In fact, if the grid region $\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}$ is sampled in such a way as to satisfy the privacy constraint, while ensuring the accuracy of approximate answers that involve $\mathcal{R}_{\mathcal{P}(\mathcal{A}),k}$, then the same properties can also be inherited by input queries on $\mathcal{A}$ as well, being the latter queries "defined" on top of grid regions.

On the other hand, it should be noted that adopting the alternative approach of sampling the regions defined by queries in QWL directly, without referring to the reasoning layer of grid regions, would cause that, due to the space bound constraint $\mathcal{B}$, a sub-set of query regions of QWL will be sampled by means of an adequately-wide set of samples, whereas the remaining query regions of QWL will be sampled by means of a lower number of samples (*under-sampling*), or, even, not sampled at all. It is a matter of fact to notice that the latter situation would lead to an "unfair" synopsis data cube $\mathcal{A}'$, i.e. a synopsis data cube such that queries involving some regions of $\mathcal{A}'$ are characterized by low privacy and low accuracy, whereas queries involving other regions of $\mathcal{A}'$ are characterized by high privacy and high accuracy. Contrary to this, the goal is to obtain a "fair" synopsis data cube $\mathcal{A}'$, i.e. a synopsis data cube able to accommodate a large number of queries while satisfying the privacy constraint, and also ensuring the accuracy of retrieved (approximate) answers.

How to determine $S_mathcalT$ from the given query-workload QWL? A reasonable solution consists in selecting $S_mathcalT$ by composing all the most frequent query ranges in QWL. Notice that these ranges can be easily gathered by means of popular active/monitoring components of conventional OLAP server platforms. Overall, this strategy allows to easily obtain a very-reliable "representative" value of selectivity of queries in QWL. However, determining $S_mathcalT$ is an orthogonal aspect for this framework, so that other different strategies can be devised, and straightforwardly integrated within the core layer of the framework.

The Greedy Strategy

The second step of `computeSPPDataCube` embeds a greedy strategy for sampling the input data cube $\mathcal{A}$ in order to obtain the synopsis data cube $\mathcal{A}'$. The greedy choice is dictated by the space bound constraint $\mathcal{B}$ that imposes to compute a "best-effort" synopsis data cube $\mathcal{A}'$, i.e. a synopsis data cube such that simultaneously *(i)* satisfies the privacy constraint, *(ii)* ensures the accuracy of approximate answers, and *(iii)* fits within $\mathcal{B}$. The reasoning unit of the sampling phase of `computeSPPDataCube` is the grid region $\mathcal{R}_{P(A),k}$, meaning that, at each iteration j and until $\mathcal{B}$ is not consumed, `computeSPPDataCube` greedily selects from $\mathcal{P}(\mathcal{A})$ a grid region, denoted by $\mathcal{R}_{P(A),k}$, and extracts from $\mathcal{R}^j_{P(A),k}$ a set of samples, denoted by $S(\mathcal{R}_{P(A),k})$. By iterating the above-illustrated task for each one of the selected grid regions, the final synopsis data cube $\mathcal{A}'$ is obtained.

`computeSPPDataCube` adopts a greedy criterion to select the grid region to be sampled. This criterion considers the properties of data distributions associated to grid regions in $\mathcal{P}(\mathcal{A})$, and *selects the most skewed grid region among the available ones* (i.e., the regions of $\mathcal{P}(\mathcal{A})$ not chosen during previous iterations of the algorithm). Without any loss of generality, given a grid region $\mathcal{R}_{P(A),k}$ in $\mathcal{P}(\mathcal{A})$ the associated data distribution, denoted by $\mathcal{F}(\mathcal{R}_{P(A),k})$, can be reasonably intended as a *multidimensional distribution*, following the nature of regions (which, in turn, are defined on top of multidimensional data cubes). The main idea that underlines this greedy criterion is based on assuming that in order to "describe" a skewed grid region, i.e. a grid region $\mathcal{R}_{P(A),k}$ whose data distribution $\mathcal{F}(\mathcal{R}_{P(A),k})$ is skewed (e.g., distributed according to a *Zipf* distribution with *asymmetric peaks*), it is needed a number of samples greater than the number of samples that are necessary in order to "describe" a uniform grid region, i.e. a grid region $\mathcal{R}_{P(A),k}$ whose data distribution $\mathcal{F}(\mathcal{R}_{P(A),k})$ is Uniform (that is, values of $\mathcal{F}(\mathcal{R}_{P(A),k})$ are regularly distributed around the average value of $\mathcal{F}(\mathcal{R}_{P(A),k})$.

In order to determine if a given data distribution $\mathcal{F}$ is skewed or not, it is possible to adopt a well-established theoretical result of the literature [SO87].

According to [SO87], given a data distribution $\mathcal{F}$, $\mathcal{F}$ is considered as skewed if the *skewness value* of $\mathcal{F}$, denoted by $\gamma_1(\mathcal{F})$, is greater than its standard deviation, denoted by $\sigma(\gamma_1(\mathcal{F}))$, by a factor equal to 2.6 (i.e., $\gamma_1(\mathcal{F}) > 2.6 \cdot \sigma(\gamma_1(\mathcal{F}))$). $\gamma_1(\mathcal{F})$ can be computed as follows [Pap84]:

$$\gamma_1(\mathcal{F} = \frac{(\mu_3(\mathcal{F}))^2}{(\mu_2(\mathcal{F}))^3}$$

such that $\mu_r(\mathcal{F})$ denotes the *r-th central moment* of $\mathcal{F}$, defined as follows [Pap84]:

$$\mu_r(\mathcal{F} = \sum_{k=0}^{q-1}(k-\mu)^{\prime} \cdot \mathcal{F}(k)$$

where q is the number of samples of $\mathcal{F}$ (i.e., data items of $\mathcal{F}$) and μ is the mean value of $\mathcal{F}$. [SO87] also provides with a method for computing the standard deviation of the skewness. According to [SO87], $\sigma(\gamma_1(\mathcal{F}))$ can be computed as follows:

$$\sigma(\gamma_1(\mathcal{F})) = \sqrt{\frac{6}{q}}$$

On the basis of results of [SO87], it is usefult to introduce the so-called *characteristic function* $\Psi(\mathcal{F})$, which allows to determine if a given data distribution $\mathcal{F}$ is skewed ($\Psi(\mathcal{F}) = 1$) or uniform ($\Psi(\mathcal{F}) = 0$). $\Psi(\mathcal{F})$ is defined as follows:

$$\Psi(\mathcal{F}) = \begin{cases} 1 \text{ if } \frac{\sqrt{q}\cdot\left(\sum_{k=0}^{q-1}(k-\mu)^3\cdot\mathcal{F}(k)\right)^2}{\sqrt{6}\cdot\left(\sum_{k=0}^{q-1}(k-\mu)^2\cdot\mathcal{F}(k)\right)^3} > 2.6 \\ 0 \text{ otherwise} \end{cases} \tag{3.6}$$

such that q and μ are the number of data items and the mean value of $\mathcal{F}$, respectively. As regards performance issues, it should be noted that 3.6 can be easily implemented within a software component having low computational cost.

Sampling the Grid Regions

Now it is important to focus the attention on how the set of samples $S(\mathcal{R}^j_{\mathcal{P}(\mathcal{A}),k})$ is extracted from the grid region $\mathcal{R}^j_{\mathcal{P}(\mathcal{A}),k}$ at the iteration j of `computeSPPDataCube`, being $\mathcal{R}^j_{\mathcal{P}(\mathcal{A}),k}$ selected by means of the greedy criterion described above. Recall that sampling the grid region is the baseline operation for computing the final synopsis data cube $\mathcal{A}'$. In particular, as regards the sampling strategy the classical *Uniform sampling* can be adopted, i.e. based on a conventional Uniform generating distribution.

Briefly, this sampling strategy works as follows. Given a one-dimensional data domain $\mathcal{D}$ whose definition interval is: $[I_{min}, I_{max}]$, with $I_{max} > I_{min}$, the *i-th* sample is extracted according to a two-step task: *(i)* random sample an indexer i in $[I_{min}, I_{max}]$ by means of a Uniform distribution defined on the range $[I_{min}, I_{max}]$ (i.e., $i = Unif(I_{min}, I_{max})$); *(ii)* return the sample $\mathcal{D}[i]$. Given an *n-dimensional* data domain $\mathcal{D}$, the *i-th* sample is extracted via iterating the above-illustrated task for each of the n dimensions of $\mathcal{D}$. Also the sampling without duplicates has been used, i.e. at each random extraction is ensured that the sampled indexer has not been picked before.

Given the grid region $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$ at the iteration j of `computeSPPDataCube`, firstly it is needed to consider the corresponding range-SUM query, denoted by $\mathcal{Q}^{j}_{\mathcal{P}(\mathcal{A}),k}$, whose multidimensional range is equal to the range of $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$. Then, on the basis of a metrics-driven approach, given an integer parameter δ, such that $\delta > 0$, it is possible to iteratively sample $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$ by extracting δ-sized sub-sets of samples from $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$ until one of the following two conditions becomes true: *(i) the privacy constraint on* $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$ is satisfied (i.e., $E_I(\mathcal{Q}^{j}_{\mathcal{P}(\mathcal{A}),k}) \geq \Phi_I$), or *(ii)* $\mathcal{B}$ is consumed (i.e., $\mathcal{B} = 0$). It is a matter of fact to note that δ represents the size of a sort of *buffer* used during sampling. This solution avoids excessive computational overheads that instead would be caused if sampling is performed on massive-in-size data cube without buffering. The nature of sampling used in this framework carefully takes into account the nature of OLAP queries considered (i.e., range-SUM queries), and the requirements of the privacy constraint. Let $\mathcal{V}^{j}_{\mathcal{P}(\mathcal{A}),k}$ be the average value of $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$, and $\mathcal{U}^{j}_{\mathcal{P}(\mathcal{A}),k}$ the sub-set of data cells in $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$ whose values are greater than $\mathcal{V}^{j}_{\mathcal{P}(\mathcal{A}),k}$ i.e.

$$\mathcal{U}^{j}_{\mathcal{P}(\mathcal{A}),k} = \{\mathcal{C} \in \mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k} | val(\mathcal{C}) > \mathcal{V}^{j}_{\mathcal{P}(\mathcal{A}),k}\}$$

Then it is possible to apply the Uniform sampling on $\mathcal{U}^{j}_{\mathcal{P}(\mathcal{A}),k}$ rather than $\mathcal{R}^{j}_{\mathcal{P}(\mathcal{A}),k}$. It is easy to understand that this particular sampling strategy naturally allows to obtain an approximate answer to $\mathcal{R}^{j}_{\mathcal{Q}(\mathcal{A}),k}$ having a good degree of approximation, and, *at the same time*, a high degree of privacy (in other words, our sampling strategy is in favor of the satisfaction of the constraint

$$E_I(\mathcal{Q}^{j}_{\mathcal{P}(\mathcal{A}),k}) \geq \Phi_I$$

As above-highlighted, these properties are in turn inherited by input queries on $\mathcal{A}$.

3.4.3 Experimental Results

In order to test performances authors conducted an extensive series of experiments on several classes of synthetic data cubes. Ranging the input parameters (such as dimension number, size, sparseness coefficient etc) is the major benefit coming from using synthetic data cubes instead of real-life ones. As highlighted in previous sections, several perspectives of analysis have been taken into account, and all oriented to test the quality, the effectiveness, and the sensitivity of the proposed technique, respectively. Also, performances of this technique have been compared with the one of the method *Zero-Sum* [SLXN06], the state-of-the-art privacy preserving OLAP perturbation-based technique.

As regards the data layer of our experimental framework, has been considered the case of two-dimensional data cubes, which well covers the goals of a reliable experimental evaluation focused on evaluating privacy preservation capabilities. Indeed, the privacy preserving OLAP technique has been tested on more probing multi-dimensional data cubes, and the observed results are very similar to those experienced on two-dimensional data cubes. In particular, two kinds of two-dimensional (synthetic) data cubes have been considered: *CVA* and *SKEW*. In the first kind of data cubes (i.e., *CVA*), data cells are generated according to a *Uniform* distribution defined on a given range $[U_{min}, U_{max}]$, with $U_{min} < U_{max}$. In other words, for such data cubes the *Continuous Value Assumption (CVA)* holds. CVA assumes that data cells are uniformly distributed over the target domain.

In the second kind of data cubes (i.e., *SKEW*), data cells are generated according to a *Zipf* distribution defined on a given parameter z, with z in $[0, 1]$. In the experimental framework, the parameter $\mathcal{D}$ denotes the kind of generating data distributions. $\mathcal{D}$ can thus assume the following values: *Uniform, Zipf*. For what regards data cube size, considering two-dimensional data cubes, the cardinality of each dimension of the data cube can be denoted by L_0 and L_1, respectively (i.e., $|d_0| = L_0$ and $|d_1| = L_1$). Accordingly, K_0 and K_1 denote the range sizes of grid regions of the privacy grid, respectively. Finally, to obtain close-to-real-life data cubes, the sparseness coefficient s is introduced, which measures the percentage ratio of non-null data cells with respect to the total number of data cells of a given data cube.

Other parameters of the experimental framework are the following: *(i)* B, which models the space bound constraint $\mathcal{B}$; *(ii)* P, which models the privacy threshold Φ_I. Also, for each perspective of analysis captured in the experimental assessment, has been introduced an ad-hoc metrics.

In the quality analysis, we inherit the factors introduced by Sung et al. in [SLXN06] (the method *Zero-Sum*), namely the privacy factor and the accuracy factor. Let *(i)* $\mathcal{A}$ be the input data cube, *(ii)* $\mathcal{A}'$ be the synopsis data cube, *(iii)* $Y\{\mathbf{k}\}$ be data cube cell having $\mathbf{k}$ as multidimensional indexer, with $Y = \mathcal{A}, \mathcal{A}'$, the privacy factor F_P measures the average amount of distorted data cells contained in blocks of $\mathcal{A}'$.

A block in the method *Zero-Sum* is a sub-cube with respect to which marginal sums of perturbed data cells along rows and columns are maintained equal to zero. F_P is defined as follows [SLXN06]:

$$F_P(\mathcal{A}, \mathcal{A}') = \frac{1}{\|\mathcal{A}\|} \cdot \sum_{k=0}^{\|\mathcal{A}\|-1} \frac{|\mathcal{A}'\{\mathbf{k}\} - \mathcal{A}\{\mathbf{k}\}|}{|\mathcal{A}\{\mathbf{k}\}|} \tag{3.7}$$

In other words, F_P provides with a measure on how much good the privacy preservation of $\mathcal{A}'$ is. Being *Zero-Sum* a method oriented to data cells, and the described technique instead based on the privacy OLAP notion, when measuring 3.7 on synopsis data cube, a slight change is needed. First, the concept of block underlying 3.7 is meaningfully changed with the concept of grid region (as, indeed, they are very similar natively). Secondly, if $\mathcal{A}'\{\mathbf{k}\}$ has not been sampled (i.e., $\mathcal{A}'\{\mathbf{k}\} = NULL$), then change $\mathcal{A}'\{\mathbf{k}\}$ with the corresponding singleton aggregation computed with respect to the grid region that contains $\mathcal{A}'\{\mathbf{k}\}$.

The accuracy factor F_A is instead defined in dependence of a given query $\mathcal{Q}$ on the synopsis data cube $\mathcal{A}'$, as follows:

$$F_A(\mathcal{Q}) = 2^{-\frac{|A(\mathcal{Q})-\tilde{A}(\mathcal{Q})|}{|A(\mathcal{Q})|}} \tag{3.8}$$

such that $A(\mathcal{Q})$ is the exact answer to $\mathcal{Q}$ and $\tilde{A}(\mathcal{Q})$ is the approximate answer to $\mathcal{Q}$ (note that 3.8 is very similar to the classical definition). In other words, F_A provides with a measure on how much good the degree of approximation ensured by $\mathcal{A}'$ for a given query $\mathcal{Q}$ is. Since the interest is related to global testing of the accuracy of synopsis data cubes, 3.8 can be extended to an input query-workload QWL, as follows:

$$F_A(QWL) = \frac{1}{|QWL|} \sum_{k=0}^{|QWL|} F_A(\mathcal{Q}_{||} \tag{3.9}$$

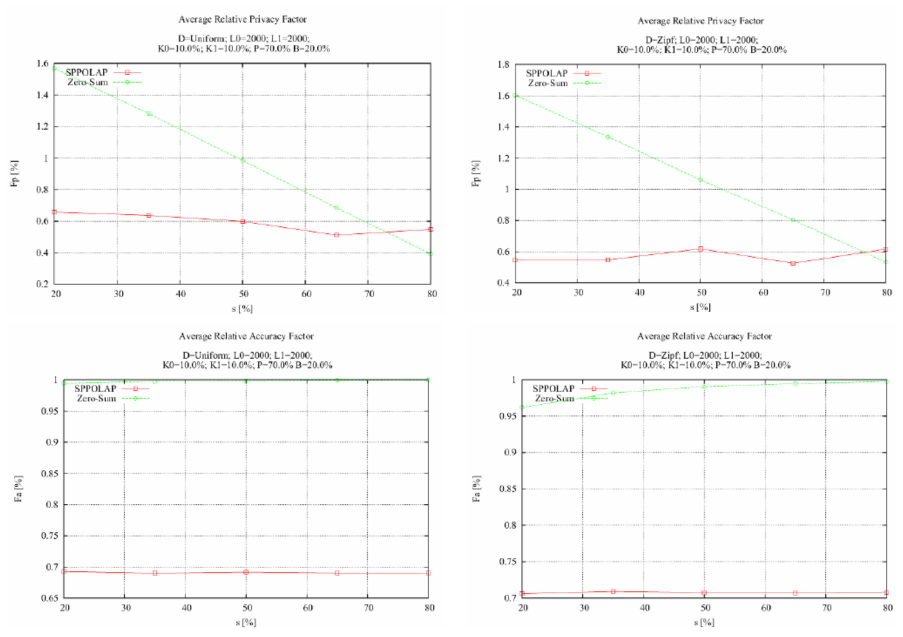
In the quality analysis, QWL is composed by the collection of range-SUM queries corresponding to blocks for the case of the method *Zero-Sum*, and to grid regions for the case of the technique above.

3.2 shows the experimental results of the quality analysis for what regards the privacy factor (3.2 (a) on a CVA data cube and 3.2 (b) on a SKEW data cube) and the accuracy factor (3.2 (c) on a CVA data cube and 3.2 (d) on a SKEW data cube) with respect to the sparseness coefficient s of synthetic data cubes, respectively. The proposed technique is here labeled as SPPOLAP.

In the effectiveness analysis, given a query-workload QWL, experiments show the average relative user-perceived inference error $\overline{E}_I^U(QWL)$ due to evaluating queries in QWL against the synopsis data cube $\mathcal{A}'$. $\overline{E}_I^U(QWL)$ is defined as follows:

$$\overline{E}_I^U(QWL) = \frac{1}{|QWL|} \sum_{k=0}^{|QWL|} E_I^U(\mathcal{Q}_k) \tag{3.10}$$

such that $E_I^U(\mathcal{Q}_k)$ is the relative user-perceived inference error due to evaluating the query $\mathcal{Q}_k$ in QWL against the synopsis data cube $\mathcal{A}'$. In this experimental framework, queries in QWL are synthetically generated as those queries that completely “span” the target synthetic data cube, and having selectivity S equal to a fixed percentage value of the volume of the data cube.



(*c*) (*d*)

Fig. 3.2. Experimental results for the privacy factor ((a) on a CVA data cube, (b) on a SKEW data cube) and the accuracy factor ((c) on a CVA data cube, (d) on a SKEW data cube) w.r.t the sparseness coefficient of synthetic data cubes

3.3 shows the experimental results of the effectiveness analysis on a CVA data cube (3.3 (a)) and a SKEW data cube (3.3 (b)) with respect to the query selectivity S, respectively.

Finally, 3.4 shows the same set of previous experiments (i.e., quality analysis and effectiveness analysis) when ranging the space bound constraint B. This allows to perform the sensitivity analysis of the discussed technique, i.e. studying its dependency on a so-critical parameter like the space bound available to house the synopsis data cube.

From the analysis of the experimental results of the privacy preserving OLAP technique in comparison with the method *Zero-Sum*, it clearly follows that, for what regards the accuracy of synopsis data cubes, the performance of this technique is comparable with the one of *Zero-Sum*. Contrary to this, for what regards the privacy of synopsis data cubes, the performance of the SPPOLAP is significantly better than the one of *Zero-Sum*. This confirms to the relevance of our research contribution. Besides this, this technique introduces computational overheads that are clearly lower than those due to the method *Zero-Sum*, as the latter method is data-cell-oriented and neglects to consider any privacy OLAP notion (like the one introduced in previous sections). Finally, although based on the privacy OLAP notion, this com-

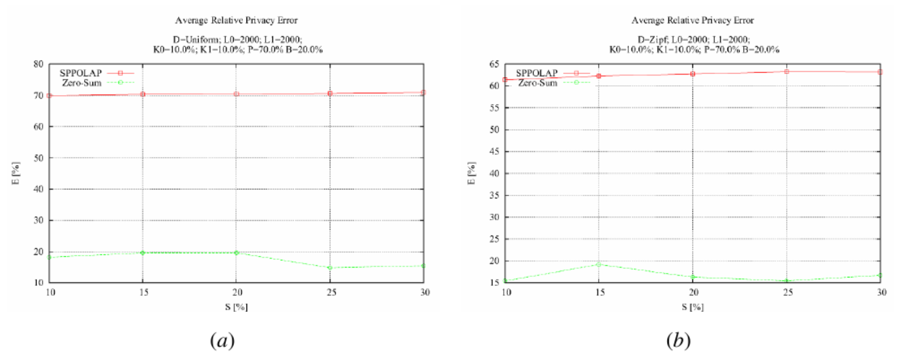


Fig. 3.3. Experimental results of the effectiveness analysis on a CVA data cube (a) and on a SKEW data cube (b) with respect to the selectivity of queries

prehensive experimental evaluation states that this technique is also lowly dependent on the space bound constraint, which is a critical parameter of any data-intensive processing technique in OLAP.

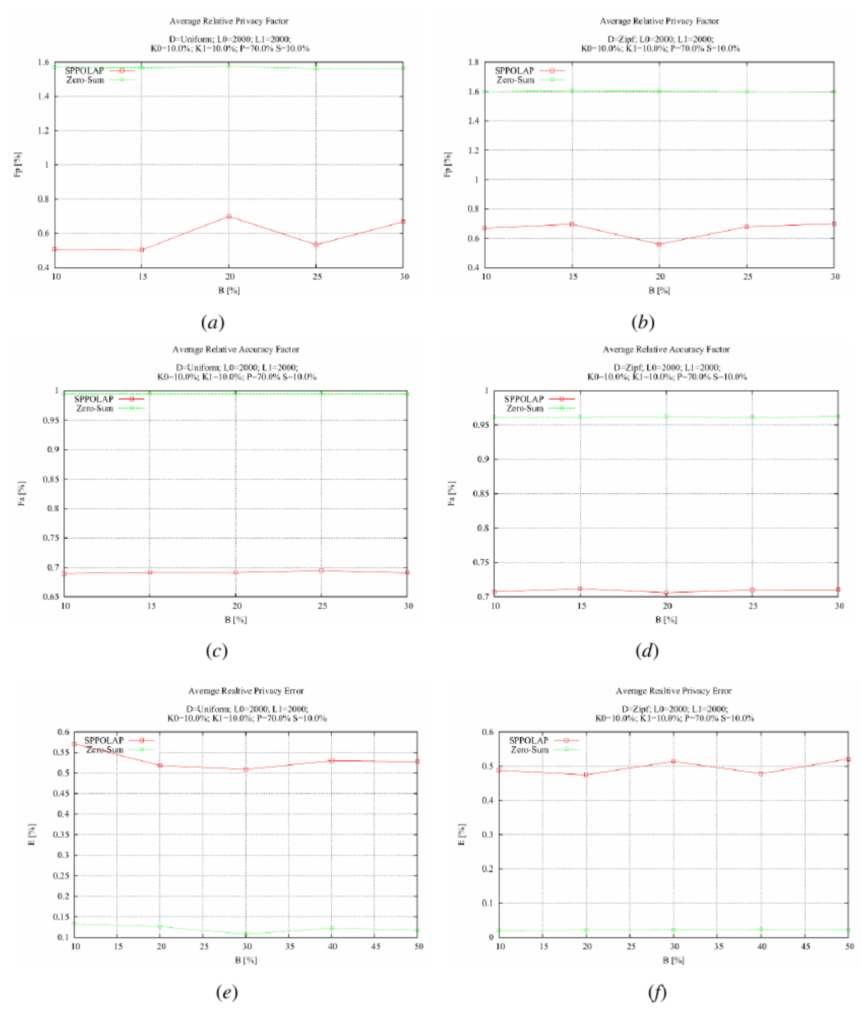


Fig. 3.4. Experimental results of the sensitivity analysis on a CVA data cube ((a), (c), (e)) and on a SKEW data cube ((b), (d), (f)) w.r.t. the space bound

4
A Data Stream Overview

Last years have seen traditional data bases used in applications that require persistent data storage and complex querying processes. However, the past few years have seen rising applications that were managing information in the form of sequence (stream) of data values, as it happens in environment like sensor data, Internet traffic, transaction logs etc. Data streams became very pervasive on the internet therefore some application could be handled by classical DBMS, by first storing information in a DB, and processing it as another application. But today they are too bursty and massive for a *store-now and process-later* approach.

In this scenario *Data Stream Management Systems (DSMS)* became always more popular as they are designed to provide efficient, robust support for online applications and continuous queries.

4.1 Data Streams: Definition and Characteristics

Trying to give an informal definition, it's enough to cite what authors say in [GÖ03a]:

> A *data stream* is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is feasible to locally store a stream in its entirety.

Some of the main characteristics of data streams depend on arrival rates of each data item, the number of attributes, the range of values, the data distribution etc. Moreover sources may introduce typical characteristics into the data stream objects.

Data items can be considered arriving continuously and sequentially, as a matter of fact they are ordered either explicitly by a timestamp or by the values of one or more elements; usually they are processed in the order they arrive. Once one element has been processed it is discarded or archived - it cannot be retrieved easily unless it is explicitly stored in memory, which is usually small relative to the size of the data streams.

After sources generate data streams, they are processed by a Data Stream Management System (DSMS).

The arrival rate among various streams can be very different and irregular according to the application environment.

Data attributes can be of different types like well-structured (i.e. temperature readings), semi-structured (i.e. XML documents) or unstructured (i.e. emails).

Because of the stream nature, network and transmission problems may cause corruption to data item and loss of information.

4.1.1 DBMS and DSMS: similarities and differences

Data Stream Management Systems are designed to process data streams continuously, since a store-now-and-process-later approach will not work due to *response requirements* (real time or quasi real time) and streams are too massive and also bursty. There are many applications similar to those of DBMS, therefore a DSMS is supposed to consider some for of SQL. On the other hand, computing environments of DBMS is quite different from that of DSMS (e.g. persistent queries on transient data vs transient queries on persistent data).

DSMSs have to follow some specific requirements:

- all stream based applications have to be able to process continuously newest arrived data. For instance a query is *long running* and needs to be evaluated repeatedly against new data until the query is terminated;
- exact results are not always needed, therefore approximation is tolerated as long as some critical aspects are satisfied;
- some applications have specific QoS requirements like *response time, precision, allowed memory usage* etc. As these requirements are not independent of each other exists such a kind of *trade-off* among all parameters that need to be balanced by the DSMS based on the application needs;
- resource optimization is a critical aspect, so that it is very important to incorporate mechanisms for capacity planning and optimal resources usages;
- *Complex Event Processing (CEP)*, rule processing and notifications are at the same time very important requirements, in particular for applications that detect dangerous events or conditions and provide actions to catch and manage them (i.e. call emergency services in case of fire etc.).

DBMS can not satisfy those requirements simply loading data streams and processing queries. It's suitable for providing persistent data management but it can not easily manage transient data. A DBMS is always working on a set/bag of tuples unlike DSMS manages infinite sequences of tuples. The update process is also very different: DBMSs provide updates referring to all tuples but in DSMS only append operation are feasible. In table 4.1 are pointed out main differences between DBMS and DSMS.

DBMS	**DSMS**
Model: **persistent** data	Model: **transient** data
Table: **set/bag** of tuples	**infinite sequence** of tuples
Updates: **all**	Updates: **append only**
Query: **transient**	Query **persistent**
Query Answer: **exact**	Query Answer: often **approximate**
Query Evaluation: **multi-pass**	Query Evaluation: **one-pass**
Operator: **blocking** and **unblocking**	Operator: **unblocking only**
Query Plan: **fixed**	Query Plan: **adaptive**

Table 4.1. Main differences between DBMS and DSMS

Moreover we will see operations over DSMS are well-known, in the sense that they are the operations of relational algebra but the evaluation of these operations is entirely different than in traditional database systems. The main reason derive from the fact that moving from disk-based data to network-bound data, two of the fundamental database assumptions are dropped: firstly, the data is no longer stored on a local disk; as if this were not enough, a stream may in fact be infinite in length, as we have already said. This means that the query processing paradigm employed by database systems needs to be revisited and, in consequence, largely modified to address the new assumptions.

4.2 Issues on Query Execution

The main approaches to query execution and optimization for queries over streaming information sources can be classified into two broad categories the *static approach* and the *adaptive approach*. The first one is applied when the system is relatively static, i.e., the optimize-time environment is not significantly different than the run-time environment. The latter is also expected to remain stable throughout query execution. For instance, a query over a continuous feed of data arriving at a constant rate is a good candidate for static optimization and execution.

The second approach is considered when the environment is completely *dynamic* in the sense that the system cannot make any assumptions on the nature of the inputs, the rates at which they arrive, the selectivity factors of the predicates and so on. The only thing the system can do is adapt to the changing environment.

To give a general idea on how execution is performed, we will concentrate on a powerful subset of relational algebra: *conjunctive queries*, i.e., queries containing projections and conjunctions of selections and joins. We assume that the system employs *push-based execution model*[1], i.e., the tuples of the incoming streams, as well as the operator output tuples, are immediately pushed to subsequent operators for further processing. A very important aspect is also related to the semantics of queries over infinite sources. It is obvious that they have to be modified for certain operators. For instance, a join over infinite sources means that infinite memory is needed to buffer the inputs. We need to a mechanism to extract finite subsets of the inputs. It will later be seen (4.4) this mechanism is based on *sliding windows*.

4.3 Continuous Queries

Queries over continuous data streams have much in common with queries in a traditional database management system. However, there are two important distinctions peculiar to the data stream model. The first distinction is between *one-time queries* and *continuous queries* [TGNO92]. One-time queries (a class that includes traditional DBMS queries) are queries that are evaluated once over a point-intime snapshot of the data set, with the answer returned to the user. Continuous queries, on the other hand, are evaluated continuously as data streams continue to arrive. *Continuous queries* are the more interesting class of data stream queries, and it is to them that we will devotemost of our attention. The answer to a continuous query is produced over time, always reflecting the stream data seen so far. Continuous query answers may be stored and updated as new data arrives, or theymay be produced as data streams themselves. Sometimes one or the other mode is preferred. For example, aggregation queriesmay involve frequent changes to answer tuples, dictating the stored approach,while join queries are monotonic andmay produce rapid, unbounded answers, dictating the stream approach.

The second distinction is between *predefined queries* and *ad-hoc queries*. A predefined query is one that is supplied to the data stream management system before any relevant data has arrived. Predefined queries are generally continuous queries, although scheduled one-time queries can also be predefined. Ad hoc queries, on the other hand, are issued online after the data streams have already begun. Ad hoc queries can be either one-time queries or continuous queries. Ad hoc queries complicate the design of a data stream management system, both because they are not known in advance for the purposes of query optimization, identification of common subexpressions across queries, etc., and more importantly because the correct answer to an ad hoc query may require referencing data elements that have already arrived on the data streams (and potentially have already been discarded).

In [BW01] and [BBD$^+$02] authors demonstrate with simple examples the need of continuous queries and why conventional DBMS technologies are inadequate. They consider also a very simple scenario, based on network traffic

[1] in 4.6 we will better point out the characteristics of this model

management system for large network, to illustrate the differences between querying data streams and traditional stored data sets. Such systems monitor a variety of continuous data streams that may be characterized as unpredictable and arriving at a high rate, including both packet traces and network performance measurements. Typically, current traffic-management tools either rely on a special-purpose system that performs online processing of simple hand-coded continuous queries, or they just log the traffic data and perform periodic offline query processing. Conventional DBMSs are deemed inadequate to provide the kind of online continuous query processing that would be most beneficial in this domain. A data stream system that could provide effective online processing of continuous queries over data streams would allow network operators to install, modify, or remove appropriate monitoring queries to support efficient management of the network resources.

Given a continuous stream of tuples and a single query Q we are interested in answering over the stream. Q is a continuous query that operates continuously as new tuples appear in the stream and suppose we are interested in the exact answer to Q (as opposed to an approximation). Let us further suppose that the data stream is *append only*[2] therefore we can think of the stream as an unbounded append-only database D. Even in this simplest of cases, there are different possible ways to handle Q, with different ramifications:

1. suppose we want to always store and make available the current answer A to Q. Since the database D may be of unbounded size, the size of A also may be unbounded (e.g., if Q is a selection query).
2. suppose instead we choose not to store answer A, but rather to make new tuples in A available when they occur, e.g., as another continuous data stream. Although we no longer need unbounded storage for A, we still may need unbounded storage for keeping track of tuples in the data stream in order to determine new tuples in A (e.g., if Q is a self-join).
3. even if the stream is append-only, there may be updates or deletions to tuples in answer A (e.g., if Q is a group-by query with aggregation). Now, in case above we may need to somehow update and delete tuples in our output data stream, in addition to generating new ones.
4. In the most general scenario, the input data stream also may contain updates or deletions. In this case, typically more of the stream needs to be stored in order to continuously determine the exact answer to Q. One way to address these issues is to restrict the expressiveness (see 4.3.1) of Q and/or impose constraints on characteristics of the data stream so that we can guarantee that the size of Qs answer A is bounded, or that the amount of extra storage needed to continuously compute A is bounded. Another possibility is to relax the requirement that we always provide an exact answer to Q, which relates to the area of *approximate query answering* that will be further investigated.

[2] it has no updates or deletions

To guarantee a syntax semantic uniformity between DSMS and DBMS, researches use simple extensions of the standard SQL (Structured Query Language). In general queries for DSMS can be tested on DBMS, and it could be a practical solution if arrival rates are modest. It's quite clear that the vice versa might not work.

In next sections we will discuss more in detail how to manage operators in data stream query processing and how to guarantee a certain expressiveness level during query definition.

4.3.1 CQ and Blocking Operators

Continuous queries consist of classical relational operators like `select`, `join` and some other `aggregation` operators. Although those similarities, it important to point out operators in DSMS do not assume the stream to be finite. Therefore many complications crop up due to unbounded nature of the input stream, in fact some of these operations and aggregations like `join` or `sort` can not be completed if the entire input data set has not been processed. Those operators *block* the processing phase producing no output until the end of the stream. They are called *blocking operators*. On the other hand there are operators like projection and selection, that work on a single data item at a time and do not block computation, those are named *non-blocking operators*. Traditional SQL aggregates are blocking. In DBMS several query operators are implemented in blocking ways, but they express functions that *are not intrinsically* blocking. DBMS use blocking implementations for many operators and they are not suitable for DSMS, however if it's possible to implement the same operator in non-blocking ways, then it can be used in DSMS. For instance `GROUP BY`, `JOIN` can be implemented in blocking (by *sorting*) and non-blocking (e.g. by hash tables), other operators are intrinsically blocking, therefore it's needed to specify which one is can be represented in a non-blocking manner. This complication produce a significant loss of expressive power. However by [LWZ04] we can point out a theorem:

Theorem 4.1. *Queries can be expressed via nonblocking computations iff they are monotonic w.r.t. the presequence ordering.*

As a sketch of the proof we can observe for additional input the output is larger than the previous one, so we can just add to output!

This theorem generalizes from presequences to sets, i.e. presequences where duplicates are not allowed and order is immaterial.

Traditional aggregate operators (MAX, AVG, etc.) always return a sequence of length one and they are all non-monotonic, and therefore blocking. Continuous `COUNT` and `SUM` are monotonic and non-blocking and thus suitable for continuous queries.

If we then consider a query language L, we can assert that it can express a given set of functions on its input (DB, sequences, data streams). Thus non-monotonic functions are intrinsically nonblocking and they cannot be used on data streams. For continuous queries on data streams, we should disallow blocking (i.e., nonmonotonic) operators and constructs and only allow non-blocking (i.e., monotonic) operators: NB-operators for short (including the query constructs used to express them).

It is useful at this point to introduce the *Non-Blocking-Completeness* property, which gives more generalization to the use of monotonic operators into queries in DSMS:

Definition 4.2 (NB-completeness). *When using only NB-operators, L can express all the monotonic queries expressible in L, then L is said to be* ***NB-complete****.*

As we know Relational Algebra/Calculus, non-recursive Datalog achieve a level of expressive power known as *relational completeness*. These are *NB-complete* iff they can express all their monotonic functions only using their monotonic operators/constructs. Let us now consider DB relations, the monotonicity property is with respect to subset containment, i.e. selection, projection, union, and joins are monotonic but set difference R-S is monotonic on R and antimonotonic on S (it will block on S till it has seen the whole S but not on R).

Using new constructs the expressive power for NB-threshold is higher. Moreover *User defined Aggregates* functions remain a viable extensibility mechanism for continuous queries, but only non blocking aggregates can be used and with UDAs we get *Turing Completeness*[3]. Then NB-Completeness requires expressing all computable monotonic functions, using only non-blocking constructs.

Because of its nonmonotonic operators SQL-2 is not suitable for executing queries over data streams, in fact SQL's lack of expressive power is a major problem for database-centric applications. These are significantly more serious for data streams since only monotonic queries can be used and actually not even all monotonic ones since SQL is not nb-complete, neither embedding SQL code into programming languages, as often happens with Java or C++. In this case is quite useful to access the tuples returned by SQL using a '*Get Next of Cursor*' statement. But the problem is that cursors are a *pull based* mechanism and cannot be used on data streams: the DSMS cannot hold tuples until the programming language request them. On the other hand the DSMS can only deliver its output to the programming language as a stream, that would be fine some simple situations, but not if more complex scenarios, where the core of the work has to be done by the programming language. As a conclusion it is easy to understand that to support applications of any complexity we must have a DSMS with real expressive power as opposed to DBMS that are useful even with a weak query language.

[3] A computational system that can compute every Turing-computable function is called *Turing-complete* (or Turing-powerful). Alternatively, such a system is one that can simulate a universal Turing machine

As a matter of fact embedding continuous query language in programming languages does not work well, but embedding programming languages into continuous query languages works. In particular UDA can be defined using a programming language or SQL itself and moreover with natively defined UDAs, SQL becomes Turing complete and NB-complete because it can express all monotonic functions.

4.3.2 Order

Another complication arises: as it has already been seen in most applications, data streams are viewed as ordered either explicitly by their timestamps or implicitly by the arrival order. Thus queries and Relational Algebra (RA) operators must preserve this order. E.g. union of two or more streams becomes a merge that preserves the order of the incoming streams and the order of their timestamps (if they are present). But order also finds many uses, in fact windows and window operators assume an order. Moreover powerful SQL extensions for pattern search have been proposed for ordered sequences of tuples. It's clear that there's a similarity between order-based construct for DSMS and DBMS.

4.4 Approximate Query Answering

As described in the previous section, when we are limited to a bounded amount of memory, e.g. find the average volume of trades during the last hour, it is not always possible to produce exact answers for data stream queries; however, high-quality approximate answers are often acceptable in lieu of exact answers. *Approximation algorithms* for problems defined over data streams has been a fruitful research area in the algorithms community in recent years. These activities have led to some general techniques for data reduction and synopsis construction, including: sketches, random sampling, histograms, and wavelets. Based on these summarization techniques, there have been many on approximate query answering. However, research problems abound in the area of approximate query answering, with or without streams. Even the basic notion of approximations remains to be investigated in detail for queries involving more than simple aggregation.

4.4.1 Sliding Window

One technique for producing an approximate answer to a data stream query is to evaluate the query not over the entire past history of the data streams, but rather only over *sliding windows* of recent data from the streams. For example, only data from the last week could be considered in producing query answers, with data older than one week being discarded.

When a sliding window is set over a data stream, it is a natural method for approximation that has several attractive properties. One of the best advantages is that the semantics of the approximation are clear, therefore users of the system can be confident that they understand what is given up in producing the approximate answer. At the same time there is no danger that different choices will cause a bad approximation. One of the most important property is that the sliding window emphasizes recent data, which in the majority of real-world applications is more important and relevant than old data. In many applications where monitoring environment and applications is a critical task, it is much more useful to spend resources in analyzing recent data despite older one. For instance, one is trying in real-time to make sense of network traffic patterns, or phone call or transaction records, or scientific sensor data, then in general insights based on the recent past will be more informative and useful than insights based on stale data. In fact, for many such applications, sliding windows are considered not only as an approximation technique but rather as part of the desired query semantics, expressed as part of the users query. There are a variety of research issues in the use of sliding windows over data streams. There is the fundamental issue of how we define timestamps over the streams to facilitate the use of windows.

Data streams are (basically ordered according their timestamps. The meaning of relational algebra operators such as unions and joins is based on timestamps. DSMS are considering timestamps assignment and meaning in different ways. We can distinguish *external timestamps*, *internal timestamps*, *missing timestamps*[4].

Extending SQL or relational algebra to incorporate explicit window specifications is nontrivial. The implementation of sliding window queries and their impact on query optimization is a largely untouched area. In the case where the sliding window is large enough so that the entire contents of the window cannot be buffered in memory, there are also theoretical challenges in designing algorithms that can give approximate answers using only the available memory.

Aggregates are very frequently applied to windows and also joins of data rely on windows. Windows are historical snapshot of a finite portion of a stream at any time point, and two basic types of windows are used widely: time-based (or a physical window) and tuple-based (or a logical window). For instance a time-based window is used when in case of high arrival rate it is only possible to keep tuples arrived in the last hour or something like `[Range N time units, advance M time units]`; and a tuple-based window can be expressed as `[Row N tuples, advance M tuples]`[5]. There is a clear way of moving from time-based to tuple-based windows: *Little's Theorem*[BG92] states that given a process with an average arrival rate of A and a time period

[4] *missing timestamps* are also called *latent*

[5] In this notation we consider: `Range` specifies the size of the first window. How the window advances to form a new window is specified by the `advance` component.

T, then the expected number of arrivals is equal to $A \cdot T$. From the previous equation, a time-based window of length T becomes a tuple-based window of size N.

Generally speaking, a window can be specified for each data stream used by a query and is applied to all the operators of that query. Of course the window characteristics can be different for the same stream in different queries. A window is usually specified as a *sliding* (or rolling) one either as overlapping or as disjoint (or tumbling). An overlapping window shares a portion of the current window with the next one whereas a disjoint one does not. In many SQL extensions proposed in literature, the `FROM` clause often includes the window specification. A sliding overlapping window is shown in Fig. 4.1 along with start and end boundaries.

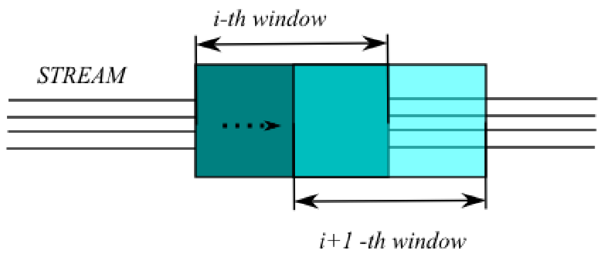


Fig. 4.1. A Sliding Window

4.4.2 Other Approximation Techniques

Sliding windows are not, as we already said, the only way to perform approximation over incoming data streams. They are surely one the most important and widely used. Another class of techniques for producing approximate answers is to give up on processing every data element as it arrives, considering sampling or batch processing techniques to speed up query execution. In any case the primary performance goal is to ensure that space required in processing phase is *small* and if possible independent of the number of streams N (which is unbounded).

Batch and Sample Processing

When a DSMS needs a very fast update operation but the way to compute the answer is quite slow, the natural solution is to process the data in *batches*. Rather than producing a continually up-to-date answer, as long as the data elements arrive, they are buffered and the answer to the query is computed according to a specified time period. Because of answer is not in run-time then this approach is considered to be approximate, i.e., it represents the exact answer at a point in the recent past rather than the exact answer at the present moment. This approach of approximation through batch processing is very attractive because it does not cause any uncertainty about the accuracy of the answer, sacrificing timeliness instead. On the other hand, if the time to compute the answer is fast, it is unuseful to wait all data before computing the answer, because data arrives faster than it can be processed. Therefore, some tuples must be skipped altogether, so that the query is evaluated over a sample of the data stream rather than over the entire data stream. We obtain an approximate answer, but in some cases one can give confidence bounds on the degree of error introduced by the sampling process. However sampling-based approaches cannot give good approximation guarantees and designing sampling-based algorithms that can produce *good* approximate answers is an important and active area of research.

Synopsis Data Structures

The case when both the update and the computation operations are fast is most desirable. For classes of data stream queries where no exact data structure with the desired properties exists, one can often design an approximate data structure that maintains a small synopsis or sketch of the data rather than an exact representation, and therefore is able to keep computation per data element to a minimum. Performing data reduction through synopsis data structures as an alternative to batch processing or sampling is a fruitful research area with particular relevance to the data stream computation model.

4.5 Continuous Query Languages for DSMS

Most of DSMS projects use SQL as a language to write and define continuous queries. There are many good reasons that carry out this choice. First of all many applications span data streams and DB tables, therefore a unique language will facilitate integration and common operations on both management systems; secondly a continuous query language based on SQL will be easier to learn and use, therefore it is not needed to introduce new formalism to describe new syntax and semantic. On the other hand DSMS were *designed for persistent data and transient queries not for persistent queries on transient data* therefore adaptation of SQL and its enabling technology presents many research challenges. As we already said because of all limitations in defining queries due to blocking operators, the lack of expressive power can be reduced considering User-Defined Aggregates (UDAs).

4.5.1 The Power of UDAs

One of the best and useful feaures related to UDAs, is that they allow to concisely express complex applications, with good performance and one of the first system allowing this feature is *ATLAS* [WZ03], a single-user DBMS developed at UCLA. It provides support for SQL with UDAs on top of Berkeley-DB record manager and allows data mining analysis by defining classifiers in very small amounts of code. But most of SQL-2 aggregates are blocking therefore windows (logical, physical, slides, tumbles,) are the most suitable and flexible synopses that solve the blocking problem for aggregates. ESL [LTWZ05] is the first to support the complete integration of UDAs. UDAs are the key to power and extensibility, and thus it can support data mining, XML and also sequences not supported by other DSMS. In this way it is possible to consider just one framework for aggregates and windows, whether they are built-ins or user-defined, and independent on the language used to define them.

4.6 Query Plan: DBMS vs DSMS

The key difference between DBMS and DSMS is the way the system processes the data. DBMSs follow a pull based model as opposed to push based model of DSMS. In fact in traditional DBMS the query plan is a pipelined or iterator-based *pull paradigm*, and it is processed by starting the computation at the root operator by obtaining tuples from each of its child operators. Each of them, recursively, calls on its child operators to get required tuples in order to output computed results to its parent. On the other hand a DSMS can not satisfy the pull paradigm as the operator would be as the operators would be blocked in the case there was no input from one of its child operator temporarily. Also, when there are no inputs at one operator, the processing of that operator needs to be suspended and switched to another operator that has inputs. Hence, to accommodate the input nature of data stream processing, a *push paradigm* is used and the tuples are pushed from leaf operators gradually to the parent operators. In addition, each operator can go with a main memory buffer (or a queue), which stores unprocessed or partially processed data items. The result of an operator is directly sent to the input queues of one or more operators.

Due to the processing differences outlined, a query plan processed by a DSMS consists of a set of non-blocking operators and the queues that connect them. In Fig. 4.2 is shown the query plan of a DBMS with that of a DSMS. Continuous query processing over data streams can be conceptualized as a data flow diagram. In Fig. 4.2-(a) a node (e.g. BOP1) represents the so called non-blocking operator. While directed edge (along with the buffer) between two nodes represents the input and output relationship between those two operators. Each leaf node (e.g. BOP1) has one or more buffers (depending on the operator type) into which the incoming streams are fed. The root node of the plan (e.g., BOP2) produces the results of a CQ and are consumed by an application. With each node (or operator) there is a *synopsis* which corresponds to the resources (primarily main memory) that are needed to perform window-based computations correctly. For some operators such as select, there is no need for a synopsis as no state information need to be maintained; whereas for a symmetric hash join, a synopsis is needed. For the symmetric hash join, the synopsis consists of the hash tables for the two participating relations for the duration of the window. On the right side of the picture, Figure 4.2-(b) represents the widely-used, iterator-based (or pull-based) left-deep tree for processing relational queries. Typically, the intermediate results are pipelined (and not materialized) unless warranted for operations such as sort-merge join and aggregation.

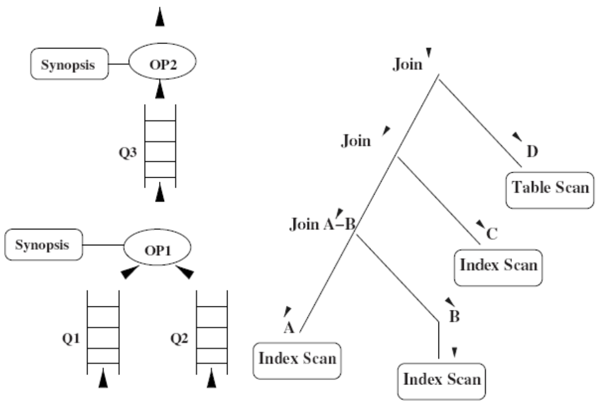


Fig. 4.2. Query Processing in a DSMS vs DBMS

4.7 DSMS Optimization: QoS and Scheduling

At this point it is easy to understand that the DSMS optimization problem is quite different. In DSMS data is stored in memory and execution time is mostly determined by the query graphs and the cost of tuples being processed. But there are many queries competing for resources and thus schedules must be optimized to minimize latency and memory (it is alike tasks scheduling in operating systems). In DBMS it is well known how to obtain execution time savings by selecting operator implementation indexes and join reordering: all of them are measured by page swap counts. Then the scheduling of various query tasks might be left up to the operating system. On the other hand in DSMS data lies in memory and execution time is mostly determined by the query graphs and the costs of the tuples being processed. Because many queries are competing for resources, schedules must be optimized to minimizelatency and memory (like in operating systems).

4.7.1 Approaches for Scheduling Continuous Queries in DSMS

As we have already seen a DSMS is supposed to compute functions over data streams in real-time and in a possibly smooth and continuous manner. Let's consider a (multiple) continuous query processing system that processes data items arriving on input streams. The problem of *resource allocation*, similar to the scheduling of individual operators of a CQ, is critical. In few words, our goal is to find a mechanism such that it establishes which query or operator of a query should be scheduled and when, and it has to compute over the newly arrived data items or partially processed data items.

In a multiple query processing system, different queries can have different QoS requirements. Some of them may favor a real-time response and some may prefer accurate results. Others may be interested in a combination of response time and accuracy or may not even care about QoS metrics at all. From a QoS perspective, a DSMS can allocate resources by choosing scheduling strategies based on individual querys QoS requirements. However, the problem is more complicated because:

- A DSMS may have a limited amount of resources (i.e., CPU cycles and main memory).
- Different objects (queries or operators) have different processing requirements and different capacities in releasing the amount of memory after a tuple is processed by an operator.
- The input pattern of a data stream can be irregular and bursty.

These considerations carry out the significant impact of scheduling strategies on various performance aspects of a DSMS and hence a DSMS can manage different strategies to maximize the available physical resources to handle the bursty nature of data streams.

Let's consider for instance two operators A and B in the system; operator A needs 1 second to process 1 tuple with a size of 2 bytes and outputs 1 tuple with a size of 1 byte (processing rate is 1 tuple/second, and memory release rate is 1 byte/second). However, operator B needs 2 seconds to process 1 tuple with a size of 2 bytes and outputs a tuple with a size of 1 byte (processing rate is 0.5 tuple/second and memory release rate is 0.5 byte/second). If input streams of both operator A and B have an input rate of 1 tuple per second for 5 consecutive seconds and then a pause, a scheduling strategy which schedules A rst if there is any tuple waiting at A, then schedules B, requires a maximal memory of 10 bytes (or 5 tuples) that are waiting at B right after the bursty input period. Another strategy which gives B a higher priority than A requires a maximal memory of 14 bytes (5 tuples at A and 2 tuples at B) right after the bursty input period. If the system only has 12 bytes memory in total, the second scheduling strategy is denitely not the best one. Similarly, a scheduling strategy also has a signicant impact on overall tuple latency and throughput of a DSMS. For this problem, the goal is to develop low overhead scheduling strategies for optimizing main memory usage, tuple latency, and their combination.

4.8 Quality of Service and Load Shedding

The mail goal of scheduling strategies is to allocate resources carefully in such a way that QoS requirements are satisfied for different queries in a DSMS, assuming sufficient total amount of resources. However, a DSMS may be short of resources for processing all registered queries and satisfy their QoS requirements during temporary overload periods due to bursty inputs. Therefore when the input stream exceeds the system capacity it may be infeasible for a scheduling strategy, no matter how good (or optimal) it is, to satisfy QoS requirements of all registered queries in the system. A natural and even the more drastic solution to this problem is to selectively discard some unprocessed or partially processed tuples from the system. It is important to understand that discarding tuples degrade the quality (in terms of accuracy) of query results. However, recall from the characteristics of stream-based applications that many of these applications can tolerate approximate results. The process of gradually discarding some tuples from a DSMS with the goal of minimizing errors introduced towards the nal results is termed *load shedding*. The load shedding process is necessary and important in a DSMS to deal with the bursty nature of its input data streams.

Introducing load shedding in a data stream manager is a challenging problem. Issues that arise for the problem of load shedding are:

- where to discard tuples along the query plan,
- the choice of tuples to discard,
- how much to discard,
- when to discard,
- when to stop discarding

For instance queries can be cut all together, or if possible some tuples might be dropped from certain streams in ways that only reduce the accuracy of the queries, by *random shedding* or *semantic shedding* that drops the less valuable first.

4.9 Complex Event Processing (CEP) and DSMS

A monitoring application needs to continuously process stream data in order to detect interesting events. These interesting events are usually composed to form higher-level events or situations (e.g., re as a composition of sudden increase in temperature accompanied by smoke within a time interval in a small geographic location) to trigger a sequence of predened actions once the situation is detected.

A CEP component is not usually a part of the QoS-aware DSMS architecture and a DSMS is not likely to have this component as part of its functionality. DSMSs have little or no support to express events as the outcome of continuous queries and further compose them to form complex events. In contrast, event processing systems that use the *Event-Condition-Action* (ECA) paradigm have been researched extensively from the situation monitoring viewpoint to detect complex or composite events and to take appropriate actions. Several event specication languages and processing models have been developed, analyzed, and implemented. Researchers have addressed these two topics independently at different periods of time. Many systems claim to be both a stream as well as a complex event processing system further blurring the differences between the two. CEP and DSMS seems to be competing for the same applications and as a matter of fact they use similar acronyms. But they have different technical roots:

- query languages of databases inspired DSMS
- message systems, CORBA

According to David Luckham, author of the book *"The Power of Events"* ([Luc08]) they belong to two different conceptual models:

- An ***event stream*** is a sequence of events linearly ordered by time such as stock market feed.
- An ***event cloud*** is a *Poset of Events*, that is the result of many event generating activities with different sources destinations in an IT system. A cloud might contain many streams.

More in details, event stream processing is focused more on high speed querying of data in streams of events. As a consequence, processing a stream of events in their order of arrival means fast and less memory. Clouds do not assume that events arrive in nice order and users are often interested in events that have complex relationships. It's easy to see that CEP applies to a richer set of business problem, not only event data processing, but also business process management (e.g extracting information from clouds of events created in enterprise IT and business systems. CEP takes more memory and more time.

4.10 DSMS Projects

In last years many research groups presented their works in designing DSMS. Each product have different characteristics. Most of them lost the support from founders and many other new systems are growing. In folloing paragraph we present one of the most important DSMS which opened important research topics and posed bases to most challenging data stream research problems.

STREAM

One of the first and most famous is *STREAM (STanford StREam DatA Manager)* [ABB+04] from the the Stanford University which is no more officially supported. In the STREAM project builds a general-purpose prototype that supports a large class of declarative continuous queries over continuous streams and traditional stored data sets. The STREAM prototype targets environments where streams may be rapid, stream characteristics and query loads may vary over time, and system resources may be limited.

TelegraphCQ

The *Telegraph project*[CCC$^+$03] at UC Berkeley began in early 2000 with the goal of developing an Adaptive Dataflow Architecture for supporting a wide variety of dataintensive, networked applications. The Telegraph concept grew out of earlier projects on adaptive relational query processing aimed at building systems that could adjust their processing on the fly, in response to changes in user needs or to intermittent delays in accessing data across wide-area networks. The basic technologies underlying Telegraph were developed to provide adaptivity to individual dataflow graphs.

Aurora

Aurora[CcC$^+$02] is a prototype system which is designed to better support monitoring applications. this system assume data to come from a variety of data sources such as computer programs that generate values at regular or irregular intervals or hardware sensors. The basic job of Aurora is to process incoming streams in the way defined by an application administrator. Aurora is fundamentally a data-flow system and uses the popular boxes and arrows paradigm found in most process flow and workflow systems. Hence, tuples flow through a loop-free, directed graph of processing operations (i.e., boxes). Ultimately, output streams are presented to applications, which must be programmed to deal with the asynchronous tuples in an output stream. Aurora can also maintain historical storage, primarily in order to support ad-hoc queries.

Gigascope

Gigascope [CJSS03] is a stream database for network applications including trafc analysis, intrusion detection, router conguration analysis, network research, network monitoring, and and performance monitoring and debugging. Gigascope is undergoing installation at many sites within the AT&T network, for detailed monitoring. The Gigascope query language, *GSQL*, is a pure stream query language with SQL-like syntax (being mostly a restriction of SQL). That is, all inputs to a GSQL are streams, and the output is a data stream. The query model used by most of the recently proposed stream database systems is that of a continuous query over a sliding window of the data stream. While this model has some advantages (e.g., presentation of results to the end user) and some areas of best application (e.g. sensor networks), we felt that the continuous query model to be inappropriate for network data analysis. One significant problem is that the continuous query model makes query composibility difficult. The input to a query is one or more data streams, but the output is a (continuously changing) table.

4.11 SensorGrid System

There are many other systems based on stream processing issues and many of them are still in a work in progress state. Although cited systems represent the main works on DSMS, there are many of them working on some different aspect and environment. One of them is the SensorGrid system [CFM$^+$04, CFMS04] which is based on a *grid architecture* which carries out many challenging problems. The key idea is based on providing fast approximate answers to aggregate queries on sensor data streams. It is based on a hierarchical summarization of the data stream embedded into a exible indexing structure, which permits to both access and update compressed data efficiently. The compressed representation of data is updated continuously, as new sensor readings arrive. When the available storage space is not enough to store new data, some space is released by compressing the oldest stored data progressively, so that recent information (which is usually the most relevant to retrieve) is represented with more detail than old one.

In [CKR08] there is a comprehensive experimental evaluation of the SensorGrid query performance for two important classes of OLAP-like queries over sensor readings, namely the window queries [CFM$^+$04] and continuous queries [BW01]. Window queries apply a SQL aggregation operator over a fixed window over the materialized sensor readings (i.e., in an OLAP-like manner). Continuous queries instead consider a moving window, and produce as output a stream of answers. Both classes of queries are extremely useful to extract summarized knowledge to be further exploited by OLAP-like analysis tools over sensor network data. Also, the experimental assessment proposal is conducted on several synthetic data sets, and takes into account several perspectives of performance analysis. The experimental results clearly confirm the benefits deriving from embedding the data compression/approximation paradigm into Grid-based sensor network data warehouses.

SensorGrid system consists of the following entities (see 4.3): *(i) Sensor*: is the basic data source; *(ii) Stream Source*: is the Grid node collecting readings produced by sensors; *(iii) Source Domain*: is a domain of Stream Sources; *(iv) Stream Server*: is the Grid node handling the summarized information stored in a certain Source Domain; *(v) Server Domain*: is a domain of Stream Servers that establish a subscriber/executor snapshot protocol, called *Grid Snapshot Protocol (GSP)*. According to the architecture sketched above, SensorGrid is based on the definition of several domains at different levels of abstraction, and each of these domains is responsible for a particular task. The final goal of such an approach is to obtain a multi-level Grid framework able to effi-

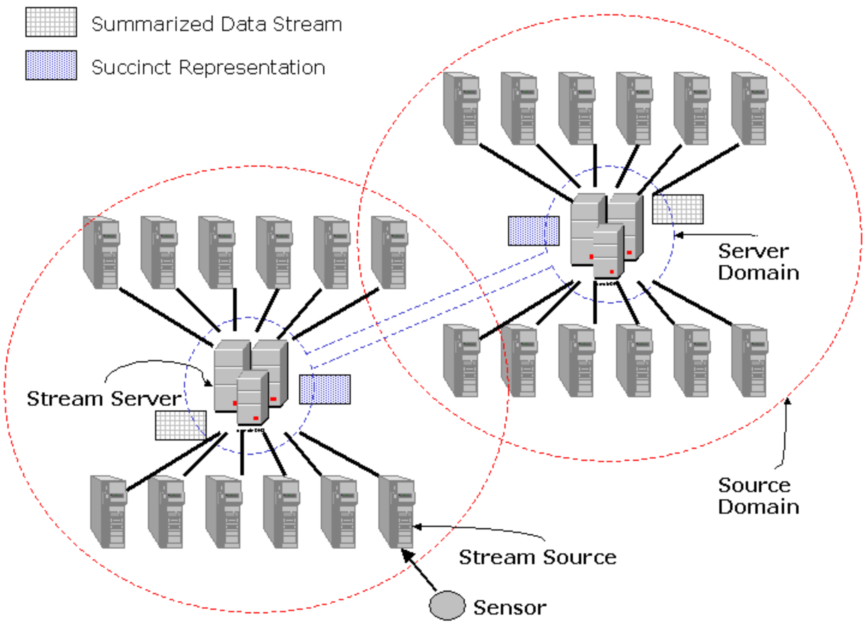
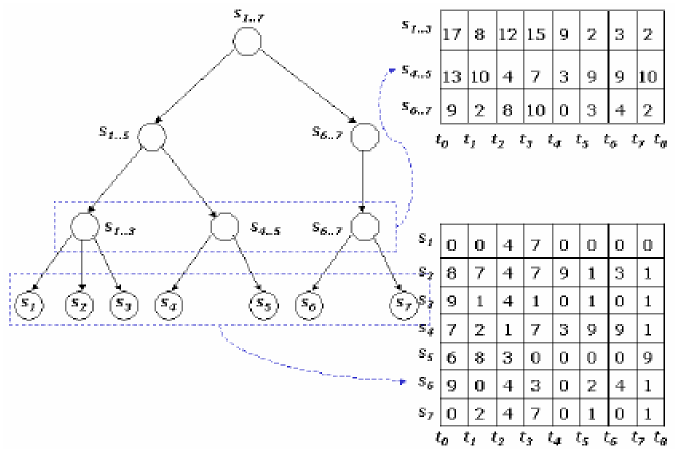


Fig. 4.3. Sensorgrid Overview

ciently manage and analyze readings produced by high dimensional and high performance sensor networks.

In order to efficiently support approximate aggregate query answering on sensor readings, we adopt a two-dimensional aggregation scheme for representing summarized readings, first proposed in [CFM+04]. Under this scheme, the first dimension, called *sensor dimension*, represents the sensor domain, and the other one, called *temporal dimension*, represents the time. Each cell of the so-obtained two-dimensional array stores the sum of all the readings produced by the corresponding range of sensors during the corresponding time interval. Specifically, due to the aggregation function considered, it is clear that this approach addresses the problem of efficiently evaluating range-SUM aggregate queries [HAMS97], which apply the SQL aggregation operator SUM on a set of selected data. Supporting SUM-based aggregations allows to also manage AVG and COUNT-based aggregations as well, which are very useful to a large number of large-scale scientific applications. SensorGrid further improves the semantics of the sensor dimension. In fact, while the temporal dimension can be aggregated by means of several strategies (e.g., $Minute \rightarrow Hour \rightarrow DayorWeek \rightarrow Month \rightarrow Year$), as time is naturally ordered, the sensor dimension is not naturally ordered and then an enumeration (e.g., based on the absolute sensor identifier) must be introduced. This is the main assumption of the proposal [CFM+04] and in SensorGrid the evolution of the original aggregation scheme is achieved via superimposing an *Aggregation Hierarchy (AH)* on the sensor domain. *AH* is finally implemented as a general tree where leaf nodes represent sensors, and internal nodes represent aggregations of sensors, following a given semantics. *AH* allows us to obtain a *Multi-Level Aggregation Scheme (MAS)* over the whole two-dimensional sensor-time domain of aggregate readings (see 4.4, which is a significant evolution of the previous (single-level) aggregation scheme. In particular, each level of the *AH* involves in a specific aggregation partition of the summarized readings, and the time range is kept unaltered. This amenity allows to take advantages from an OLAP-like multi-resolution manner of representing and querying the examined sensor readings.



$s_{1..3}$	17	8	12	15	9	2	3	2
$s_{4..5}$	13	10	4	7	3	9	9	10
$s_{6..7}$	9	2	8	10	0	3	4	2

s_1	0	0	4	7	0	0	0	0
s_2	8	7	4	7	9	1	3	1
s_3	9	1	4	1	0	1	0	1
s_4	7	2	1	7	3	9	9	1
s_5	6	8	3	0	0	0	0	9
s_6	9	0	4	3	0	2	4	1
s_7	0	2	4	7	0	1	0	1

Fig. 4.4. the Aggregation Hierarchy and the Multi-level Aggregation Scheme

Another important component of SensorGrid is represented by the *Grid Snapshot Protocol (GSP)* (see 4.5). On the basis of this Grid-oriented data exchange protocol, a Stream Server S_i maintains a succinct version of summarized readings stored in another Stream Server S_j, and exploits it at query time. Specifically, in the *GSP* two roles are introduced, namely the *Snapshot Executor* and the *Snapshot Subscriber*. These roles are played by *Stream Servers* that establish the so-called *Snapshot Contract*, on the basis of which the executor is in charge of performing a set of *Snapshot Queries* required by the subscriber. The contract is governed by some pre-fixed rules (e.g., execution frequency, null value management, transmission bulk etc) codified into a Snapshot Policy shared between the contractors. Snapshot queries allow us to obtain near the subscriber the succinct representation of the summarized readings stored near the executor (see 4.5).

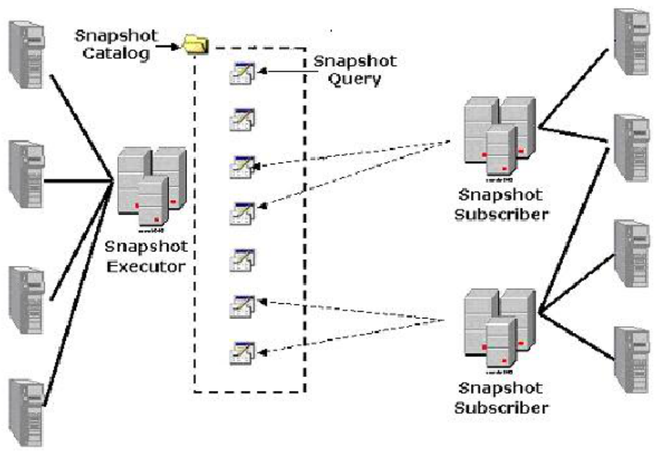


Fig. 4.5. the Grid Snapshot Protocol

GSP is meant to support both data and replica management facilities (e.g., load balancing, fault tolerance etc), but mostly a distributed query evaluation paradigm based on the approximation concept. Under this vision, during the evaluation of a query Q posed to a server S_i and involving the domain of summarized readings handled by another server S_j, S_i can alternatively re-direct Q towards the responsible server S_j, or answer Q by using its local succinct representation of summarized readings stored in S_j, thus providing a less-detailed answer, or decompose Q in a subset of queries $q_0, q_1, \ldots, q_{N-1}$ to be executed in another set of Grid nodes based on their own succinct representations of summarized readings stored in S_j. The above-described query scenario leads to the definition of the socalled *Grid-aware Distributed Query Engine (GaDQE)*. The main functionality carried out by *GaDQE* consists in dynamically determining cost-based optimal distributed query execution plans on top of *(i)* traditional parameters such arrival rate, transmission bandwidth, Grid nodes computational power, load balancing issues etc, and *(ii)* the approximation paradigm, i.e. the amenity of querying a certain Grid node or a set of Grid nodes instead of the original one on the basis of the required approximation degree and summarized reading replicas.

This framework, oriented to streaming data, is general enough to include sensor readings as a specialized instance of more general data streams. Briefly, the proposal [CFM$^+$04] is based on a quad-tree based hierarchical summarization of data streams over fixed time windows, called *Quad-Tree Windows (QTW)*, implemented by means of conventional quad-trees and embedded into a flexible *B-tree* indexing data structure. At the lowest level of aggregation, a QTW represents summarized readings into two-dimensional sensor-time arrays. The whole collection of QTW plus the B-tree index form the so-called *Multi-Resolution Data Stream Summary (MRDS)*. In SensorGrid, Stream Servers store MRDS data structures as summarized representations of readings produced by sensors belonging to their own *Source Domains* . Also, Stream Servers store MRDS data structures as succinct representations of readings produced by sensor belonging to other Source Domains. Each (native) MRDS is progressively compressed meaning that it is updated continuously, as new sensor readings arrive, and, when the available storage space is not enough to host new data into newest QTW, some space is released via compressing the oldest QTW. By adopting this approach, the recent knowledge on the MRDS is represented with more detail than the old one. Note that the recent knowledge is usually more relevant to extract for the context of sensor network applications. 4.6 shows a sketch of the MRDS data structure.

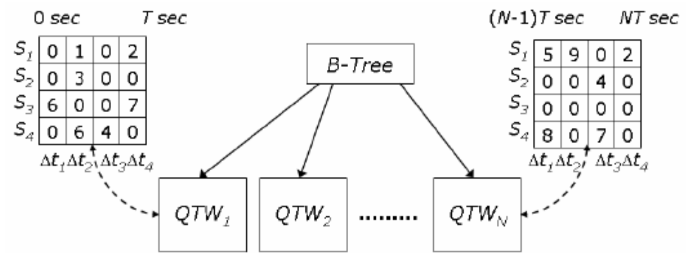
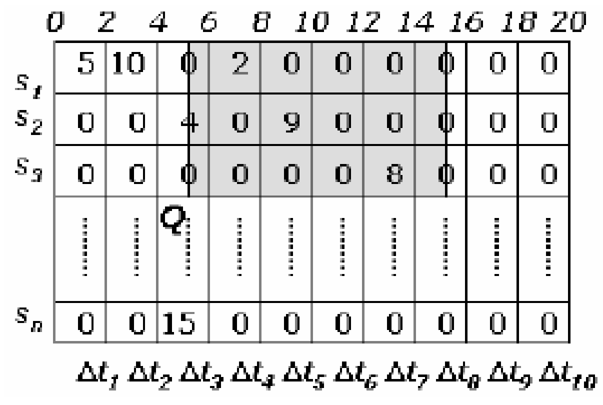


Fig. 4.6. the Multi Resolution Data Stream Summary

Using such a representation, an estimate of the answer to a (general) range-SUM query

$$Q = \langle [S_s : S_e], [ts : te] \rangle$$

over summarized data streams, such that S_s and S_e are the starting and ending stream sources, respectively, and t_s and t_e are the starting and ending timestamps, respectively, can be obtained by summing two contributions. The first one is given by the sum of those ranges completely contained by the query range. The second one is given by the sum of those ranges that are partially contained by the query range. Note that the first of these contributions does not introduce any approximation, whereas the second one is approximate, as the use of the time granularity Δj makes impossible to discriminate the distribution of data streams within the same interval Δt_j. Specifically, the latter contribution can be evaluated by means of well-known linear interpolation techniques, assuming that data distributions inside each range are uniform (see 4.7) the depicted gray box represents the range-SUM query $Q = \langle [S1 : S3], [5 : 15] \rangle$.



0 2 4 6 8 10 12 14 16 18 20
S_1 5 10 0 2 0 0 0 0 0 0
S_2 0 0 4 0 9 0 0 0 0 0
S_3 0 0 0 0 0 0 8 0 0 0
Q
S_n 0 0 15 0 0 0 0 0 0 0
Δt_1 Δt_2 Δt_3 Δt_4 Δt_5 Δt_6 Δt_7 Δt_8 Δt_9 Δt_{10}

Fig. 4.7. Two-dimensional representation and querying of summarized readings

4.11.1 Experimentation

Experimental evaluation of the SensorGrid query performance for both window (range-SUM) queries and continuous (range-SUM) queries on synthetic data sets is further presented. First, note that, since our emphasis is on testing the accuracy of approximation due to the compression paradigm on materialized sensor readings, in this experimentation authors introduce an error metrics that considers the relative difference between the exact answer (i.e., the answer evaluated on the uncompressed sensor readings) and the approximate answer (i.e., the answer evaluated on the compressed sensor readings). Therefore, this imposes to run experiments on the compressed portions of the MRDS, and neglect the uncompressed MRDS portions accordingly. As regards the experimental environment, a Java-based framework has been implemented where sensor sources are modeled by Java threads, and located on different Grid nodes of the experimental architecture.

Continuous (Range-SUM) Queries

A continuous (range-SUM) query QC on summarized sensor readings is defined as the tuple

$$Q_C = \angle\angle[Ss : Se], [ts : te]\rangle, \Delta t_s tep, f_Q\rangle$$

such that *(i)* $[S_s : S_e]$ is a fixed sensor range, *(ii)* $[t_s : t_e]$ is a moving time range, *(iii)* $\Delta t_s tep$ is the step by which the range $[t_s : t_e]$ moves forward, and *(v)* f_Q is the query frequency that establishes the time period by which Q_C is evaluated as a window query on the actual window $\langle[Ss : Se], [ts + tstep \times k :$

$t_e + \Delta t_s tep \times k]$, at the iteration k. This produces in output a stream of answers.

In order to carefully test the query performance of SensorGrid under the stress of a *ranging* input, we considered two kinds of data sets. Data sets considered are the following ones: *synthetic* and *real-life* data sets. Experimental results are similar for both classes of data sets. For space reasons, let us take in consideration results obtained from synthetic data sets. To generate synthetic data sets, customizable data distributions characterizing the nature of (synthetic) sensor readings have been used. In particular, authors introduce three distributions of sensor readings: *Uniform, Gauss, Zipf.* These distributions are together able to capture the most popular cases of sensor reading distributions that one can finds in real-life sensor networks. The benefits deriving from using synthetic data sets mainly rely in the possibility of completely controlling the nature of such data sets, thus better studying how the query performance of the system varies with respect to the variation of characteristics of the input data set.

Error Metrics

We introduce a different error metrics in dependence on the kind of queries considered. For window queries, we consider synthetically-generated populations of range queries over the compressed portion of the MRDS. This portion, which in our model is delimited by the value Tmax on the temporal dimension, can be easily determined during the run of the experimental framework via monitoring when the compression process is activated. Let $\langle [S_0 : S_{N-1}], [t_0 : T_max] \rangle$ denote the compressed portion of the MRDS, and ΔS and ΔT two input parameters that represent a range on the sensor dimension and a range on the temporal dimension, respectively. The population of window queries QP_W as those queries contained by the range $\langle [S0 : SN-1], [t0 : Tmax] \rangle$, and having *(i)* selectivity equal to $\Delta S \times \Delta T$ and *(ii)* left-up corners corresponding with items of the range $\langle [S0 : SN-1], [t0 : Tmax] \rangle$. To establish how much window queries generate, and hence the cardinality of QP_W, denoted by $|QP_W|$, we introduce the sampling factor s that determines a sample of the maximal query population $QP_{W,MAX}$ (for which $s = 100\%$). $QP_{W,MAX}$ is that population that can be defined on the range $\langle [S0 : SN-1], [t0 : Tmax] \rangle$ via considering queries having as left-up corners corresponding to all the items of the range $\langle [S_0 : S_{N-1}], [t_0 : T_max] \rangle$. Upon this model, authors introduce the error metrics given by the *Average Relative Error* ϵ_{QP_W}, defined as follows:

$$\epsilon_{QP_W} = \frac{1}{\mid QP_W \mid} \cdot \sum_{k=0}^{|QP_W|-1} \epsilon(Q_{W,k})$$

such that $\epsilon(Q_{W,k})$ is the relative error due to the approximate evaluation of the window query $Q_{W,k}$. In turn, $\epsilon(Q_{W,k})$ is defined as follows:

$$\epsilon(Q) = \frac{\mid A(Q_{W,k})\tilde{A}(Q_{W,k}) \mid}{A(Q_{W,k})}$$

where $A(Q_{W,k}$ is the exact answer to $Q_{W,k}$, and $\tilde{A}(Q_{W,k})$ is the approximate answer to $Q_{W,k}$.

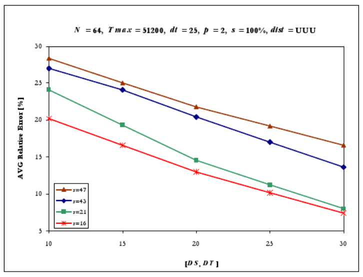
For continuous queries, authors instead fix both the (query) range on the sensor dimension, ΔS, and the (query) range on the temporal dimension, ΔT, and moves forward the $\Delta S \times \Delta T$ window query by steps equal to $\Delta t_s tep$, being the latter one an input parameter. The (approximate) answer is evaluated with frequency fQ. Due to the particular nature of continuous queries, in this case is not appropriate to adopt an averaged value as metrics (like in window queries), but rather it is interesting observing how *distant* is the variation of the approximate answer from the one of the exact one, and the corresponding relative error. Authors also consider two distinct queries simultaneously. The first one is a *small* query, i.e. a query having low selectivity, and the second one is a *large* query, i.e. a query having high selectivity. It is well-recognized that query selectivity heavily impacts on the performance of any query engine.

Experimental Results

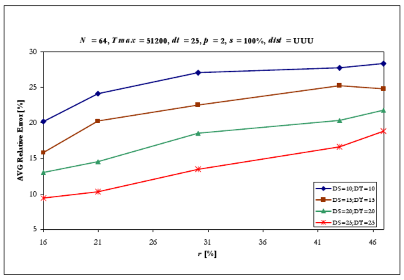
4.8 shows the experimental results when considering window queries as input. 4.9 instead shows the case for continuous queries. In particular, 4.8-(a) shows the variation of the percentage value of $\epsilon_Q P_W$ with respect to the selectivity of (window) queries in QPW on an Uniform (synthetic) data set. 4.8-(b) shows the same experimentation when ranging the compression ratio r. 4.8-(c) and 4.8-(d) show the above-illustrated metrics for a Gaussian (synthetic) data set, whereas 4.8-(e) and 4.8-(f) for a Zipfian (synthetic) data set.

4.9-(a) focuses on the variation of the distance between exact and approximate answers to two *small* and *large* (continuous) queries on an Uniform (synthetic) data set. 4.9-(b) plots the percentage variation of $\epsilon(Q_C)$ due to exact and approximate answers shown in 4.9-(a). Finally, 4.9-(c) and 4.9-(d), and 4.9-(e) and 4.9-(f) show the same experimentation on a Gaussian (synthetic) data set, and a Zipfian (synthetic) data set, respectively. It should be noted that for continuous queries, a different configuration of the experimental parameters has been considered, due to need for *(i)* adequately testing with continuous queries, and *(ii)* capturing the starting time of the related MRDS compression process (this involved Tmax = 75,225 for the Uniform data set, and Tmax = 75,133 for Gaussian and Zipfian data sets).

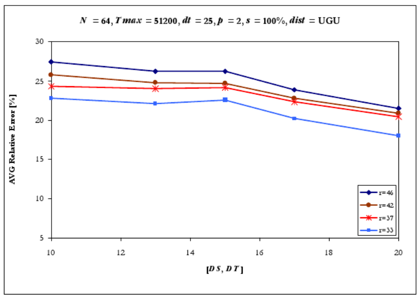
From the analysis of the experimental results presented above, it clearly follows that SensorGrid effectively supports the approximate evaluation of OLAP-like queries over Grids at a provable degree of accuracy and with good performance with respect to the ranging of the storage space available to house the compressed representation of the MRDS. The latter one is a critical parameter of our proposed Grid framework. Also, note that, for populations of window queries, the percentage average query error is mostly between the



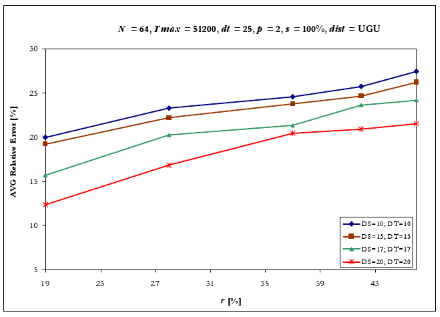
(*a*)



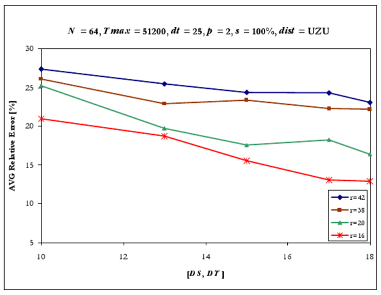
(*b*)



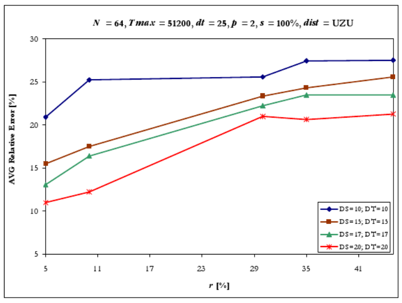
(*c*)



(*d*)



(*e*)



(*f*)

Fig. 4.8. Experimental results for Window Queries

interval $[10, 27]\%$. For what instead regards continuous queries, the percentage query error for low-selective and high-selective queries is mostly between the interval $[5, 20]\%$. These observed values are well-recognized as best performance for any approximate query engine over OLAP data.

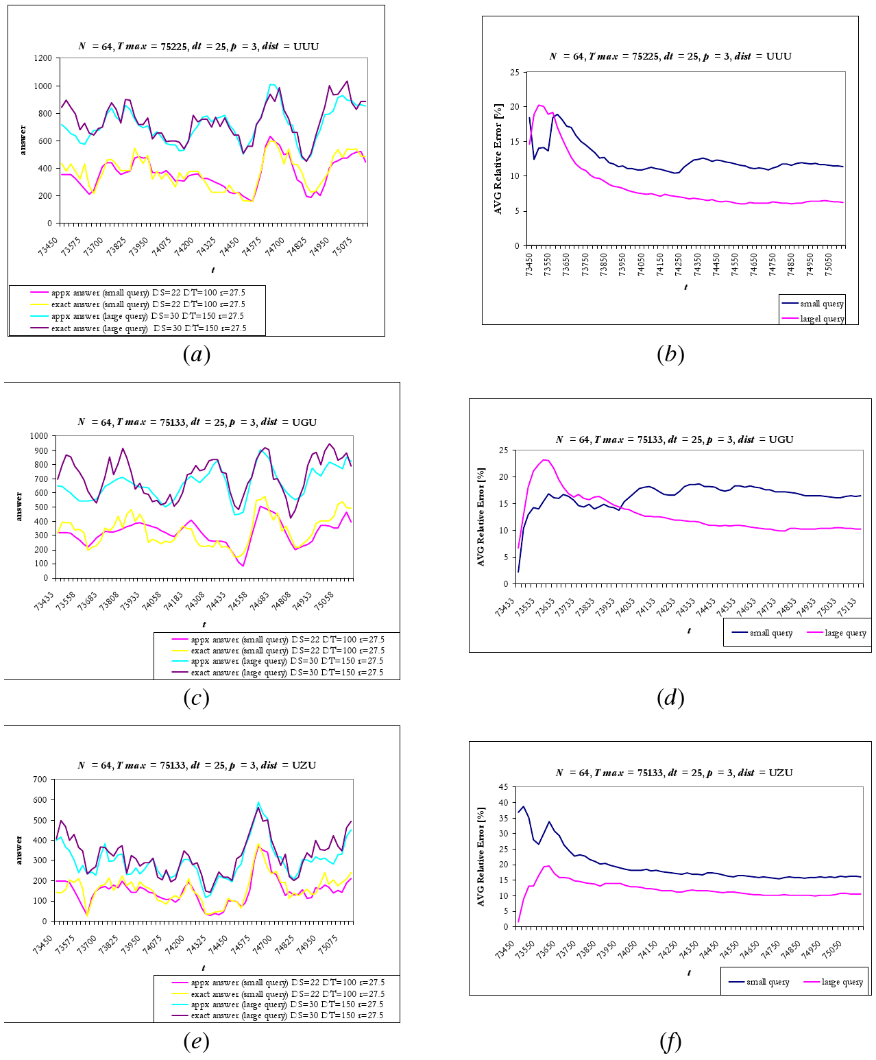


Fig. 4.9. Experimental results for Window Queries

5

Data Stream Mining

5.1 Introduction

A largely untested hypothesis of modern society is that it is important to record data as it may contain valuable information. This occurs in almost all facets of life from supermarket checkouts to the movements of cows in a paddock. To support the hypothesis, engineers and scientists have produced a raft of ingenious schemes and devices from loyalty programs to RFID tags. Little thought however, has gone into how this quantity of data might be analyzed.

Machine learning, the eld for nding ways to automatically extract information from data, was once considered the solution to this problem. Historically it has concentrated on learning from small numbers of examples, because only limited amounts of data were available when the eld emerged. Some very sophisticated algorithms have resulted from the research that can learn highly accurate models from limited training examples. It is commonly assumed that the entire set of training data can be stored in working memory.

More recently the need to process larger amounts of data has motivated the eld of data mining. Ways are investigated to reduce the computation time and memory needed to process large but static data sets. If the data cannot t into memory, it may be necessary to sample a smaller training set. Alternatively, algorithms may resort to temporary external storage, or only process subsets of data at a time. Commonly the goal is to create a learning process that is linear in the number of examples. The essential learning procedure is treated like a scaled up version of classic machine learning, where learning is considered a single, possibly expensive, operationa set of training examples are processed to output a nal static model.

The *data mining* approach may allow larger data sets to be handled, but it still does not address the problem of a continuous supply of data. Typically, a model that was previously induced cannot be updated when new information arrives. Instead, the entire training process must be repeated with the new examples included. There are situations where this limitation is undesirable and is likely to be inefcient.

The *data stream* paradigm has recently emerged in response to the continuous data problem. Algorithms written for data streams can naturally cope with data sizes many times greater than memory, and can extend to challenging real-time applications not previously tackled by machine learning or data mining. The core assumption of data stream processing is that training examples can be briey inspected a single time only, that is, they arrive in a high speed stream, then must be discarded to make room for subsequent examples. The algorithm processing the stream has no control over the order of the examples seen, and must update its model incrementally as each example is inspected. An additional desirable property, the so-called anytime property, requires that the model is ready to be applied at any point between training examples.

Studying purely theoretical advantages of algorithms is certainly useful and enables new developments, but the demands of data streams require this to be followed up with empirical evidence of performance. Therefore, claiming that an algorithm is suitable for data stream scenarios implies that it possesses the necessary practical capabilities.

Data stream classication algorithms require appropriate and complete evaluation practices. The evaluation should allow users to be sure that particular problems can be handled, to quantify improvements to algorithms, and to determine which algorithms are most suitable for their problem.

Measuring data stream classication performance is a three dimensional problem involving processing speed, memory and accuracy. It is not possible to enforce and simultaneously measure all three at the same time so in many approaches it is necessary to x the memory size and then record the other two.

Various memory sizes can be associated with data stream application scenarios so that basic questions can be asked about expected performance of algorithms in a given application scenario.

The recent growth of interest in data stream systems and data stream mining is due to the fact that, in many applications, data must be processed continuously, either because of real time requirements or simply because the stream is too massive for a store-now & process-later approach. However, mining of data streams brings many challenges not encountered in database mining, because of the real-time response requirement and the presence of bursty arrivals and concept shifts (i.e., changes in the statistical properties of data). In order to cope with such challenges, the continuous stream is often divided into windows, as already said in 4.4, thus reducing the size of the data that need to be stored and mined. This allows detecting concept drifts/shifts by monitoring changes between subsequent windows. Even so, association rule mining over such large windows remains a computationally challenging problem requiring algorithms that are faster and lighter than those used on stored data. Thus, algorithms that make multiple scans of the data should be avoided in favor of singlescan, incremental algorithms. In particular, the technique of partitioning large windows into slides (a.k.a. panes) to support incremental computations has proved very valuable in DSMS [LMT$^+$05, BTW$^+$06] and is often exploited in many approaches. The following observation is important: in realworld applications there is an obvious difference between the problem of *(i)* finding new association rules, and *(ii)* verifying the continuous validity of existing rules.

Normally, finding new rules requires both machines and domain experts, since size of the data is too large to be mined by a person and importance of new rules with respect to the application can only be validated by domain experts. In this situation, delays by the mining algorithms in detecting new frequent itemsets are also acceptable, provided that they add little to the typical time required by the domain experts to validate new rules. Thus, in [MTZ08] authors propose an algorithm for incremental mining of frequent itemsets that compares favorably with existing algorithms when real-time response is required. Furthermore, the performance of the proposed algorithm improves when small delays are acceptable.

5.2 Mining Trajectories: Proposal Using Frequent Itemsets Techniques

Frequent pattern mining techniques, are often needed, when there is a need of analyzing trajectories in order to efficiently extract useful information. It has received a great attention since it was originally introduced for transactional data [AS94, HPY00] due to its intriguing challenges. The same problem for trajectory data has been studied in [GNPP07, LCP$^+$07]. In order to better understand the problem of sequential pattern mining the following toy example is quite useful.

Example 5.1. Consider the trajectories depicted in Fig. 5.1 and the simple regioning associated to the search space. it is possible to represent the set of trajectories as $T = \{ABCDE, EDCBA, AGMS, UVWST, YTOJ\}$. It is easy to see that while in the transactional case the first two items would be considered the same, in this case they have to be treated separately.

Based on this representation of trajectory data, it is possible to introduce an approach for mining frequent patterns based on suitable space partitioningindexspace partitioning. Indeed, since trajectory data carry information about actual position and timestamp of a moving object, it is possible to split the search space in regions having the suitable granularity and represent them as symbols. The sequence of regions define the trajectory traveled by a given object. Note that regioning is a common assumption in trajectory data mining [LHLG08, GNPP07] and in this case it is even more suitable since our goal is to extract typical routes for moving objects, in order to identify

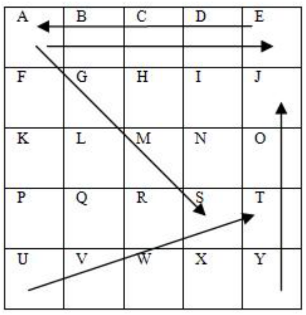


Fig. 5.1. A set of example trajectories

as an example critical junctions in a vehicular system, or as a counterpart, alternative routes when the traffic is heavy. For trajectory representation it is possible to use a novel data structure for representing itemset called *augmented FP-Tree*. The underlying idea is that it is needed to be fast while counting frequent items and due to intrinsic nature of trajectory data (order of appearance of an item is important). In this respect items appearing before and after that timestamp are taken into account separately for each node in the tree (that has a proper timestamp). This allow to maintain the nice features of FP-Trees and also to have the chance to move efficiently forward and backward on the trajectory while searching for frequent itemsets. Then a suitable strategy is used to check incoming trajectories w.r.t. existing patterns based on *pane* comparison in order to avoid inefficiencies due to comparing all the trajectories at the same time. Moreover, matching indexed frequent pattern against moving objects window, is a peculiar feature of this approach since in all the mentioned application scenarios it could happen to deal with a prohibitive number of input streams. This technique can be used to mine frequent patterns and as a nice side effect it allows to identify relevant features among data thus it can be used as a feature selection phase in trajectory classification.

While for transactional data a pattern is a subset of single items, a sequential pattern is an ordered set (i.e., a sequence) of timestamped items. For instance, if the toy database in Example 5.1 were transactional, the frequency of pattern $\{A, B\}$ would be 2, whereas for the same database (if each tuple represented a trajectory) it is clear that the frequencies of sequential patterns $< A, B >$ and $< B, A >$ are considered 2 and 0, respectively.

Trajectory data are usually recorded in variety of different formats, and they can be drawn from a continuous domain. Here is assumed a standard format for each trajectory, as defined next.

Definition 5.2 (Trajectory). *Let P and T denote the set of all possible (spatial) positions and all timestamps, respectively. A trajectory is defined as* *a finite sequence* $s_1, \cdots, s_N$*, where* $N \geq 1$ *and each* s_i *is a pair* (p_i, t_i) *where* $p_i \in P$ *and* $t_i \in T$*.*

It is assumed that P is a set of discrete symbols. For continuous locations, one can partition the space into regions to map the initial locations into discrete symbols. Notice that the way chosen for the assignment of symbols to locations is totally irrelevant for our goal since we are interested in frequent pattern mining. The granularity of the regioning can be decided according to the application requirements. For example, for tracking trucks, the GPS data can be rounded within 15 meters of precision and so on.

A sequential pattern is any sub-sequence of a trajectory. Given a collection of trajectories, it is possible to define the frequency (support) for a sequential pattern as the number (ratio) of its occurrence in the trajectories. Throughout the rest of this paper we use the term 'pattern' to simply refer to 'sequential pattern', unless otherwise is stated. The problem of mining sequential patterns from trajectories, is finding all the patterns whose frequency is larger than or equal to a given minimum support, called α. In general, traditional methods developed for mining transactional (non-sequential) patterns are not applicable to sequential patterns, as they do not differentiate between different items orders in the same pattern.

5.2.1 Defining Regions of Interest

The problem of finding a suitable partitioning for both the search space and the actual trajectory is a core problem when dealing with spatial data. Every technique proposed so far, somehow deals with regioning and several have been proposed such as partitioning of the search space in several regions of interest (*RoI*)[GNPP07] and trajectory partitioning [LHW07].

Principal Component Analysis PCA finds *preferred* directions for data being analyzed and thus it is possible to exploit this information for saving both space and time since the goal is finding *frequent* patterns, so it is likely that less frequently crossed regions are not significant.

5.2.2 Principal Component Analysis

Principal Component Analysis (PCA) [Jol02] finds a linear transformation l which reduces n-dimensional feature vectors to an h-dimensional feature vectors (where $h < n$) in such a way that the information is maximally preserved in terms of minimizing the mean squared error. The PCA also allows rolling back to n-dimensions from the h-dimensional feature vectors, with of course introducing some reconstruction error. The h-column vectors define the basis vectors. The first basis vector is in the direction of maximum variance of the given feature vectors. The remaining basis vectors are mutually orthogonal and, maximize the remaining variances subject to the orthogonal condition. Each basis vector represents a principal axis. These principal axes are those orthonormal ones onto which the remaining variances under projection are maximum. The orthonormal axes are given by the leading eigenvectors (i.e. those with the largest associated eigenvalues [Jol02]) of the measured covariance matrix.

In PCA, the original feature space is characterized by these basis vectors and the number of basis vectors used for characterization is usually less than the dimensionality d of the feature space. The covariance matrix is built for the trajectories by using the multidimensional points contained in each trajectory. Many tools have been implemented for computing PCA such as [Jol02], in particular experiments the fixed-point algorithm in [SP07] has been used.

Before defining the simple and effective regioning strategy it is needed to give an intuitive interpretation of the results of PCA analysis on trajectory data.

Consider the set of trajectories depicted in 5.2.2, running the PCA algorithm on the above set of trajectories identify the preferred directions(the red arrows), i.e. the directions along which the density of trajectories is the highest. Indeed, in 5.2.2 is drawn the first two eigenvalues that offers the best initial assignment for regioning, even if PCA returns all the eigenvalues (thus all the principal directions) with a proper rank that can be used for further refining the initial region assignment.

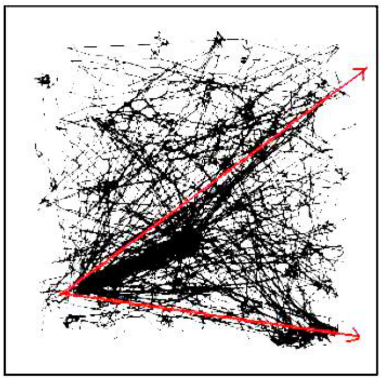


Fig. 5.2. Principal eigenvalues identified by PCA

This information are exploited using a partition strategy that gives more effort to regions along the principal directions. This allows to focus on the preferred regions when computing clusters thus saving space and computational time.

More formally, the results of running PCA on trajectories data is a set of eigenvalues that states the principal directions followed by data. Each eigenvalues define an angle α w.r.t. the principal axes. Depending on the scenario being analyzed it is choosen a suitable level of granularity that defines the size d of each region (it is assumed squared regions). The size d of each region is defined according to the domain being analyzed, for example for cellular nets it could be some meters while for truck movements it could be hundred of meters and for hurricane control it will be of kilometers. In order to define regions of interest we partition the search space in squared regions along the directions set by the eigenvalues so far obtained by PCA and having size d. It could happen that this regioning strategy could also consider regions not parallel to the principal axes thus not covering the whole space. Indeed, this "white" regions, i.e. regions not on the principal directions, could be considered as not interesting regions and thus no more investigated. This strategy, is quite effective in practice since as we point out, unfrequent regions will not affect clustering results (since few trajectories cross that regions), so pruning the search space will increase the efficiency and accuracy of the mining step.

The size of the regions can be choosen a priori since for the scenario being analyzed this choice offer really good performances, however a possible improvement could be to consider the region size as a parameter that can be set in an unsupervised way exploiting information about density of trajectories.

5.2.3 The Trajectory Regioning algorithm

Once obtained the set of regions $REG = \{R_1, R_2, \cdots, R_m\}$ as explained in previous sections we assign to each region a symbol in a given alphabet Σ.

As mentioned above a trajectory Tr_i is a sequence of multidimensional points $Tr_i = p_1, p_2, \cdots, p_n$ where n is the trajectory length. In this section we will define a strategy for partitioning the trajectory as a list of regions, thus encoding the sequence of points as a smaller and meaningful sequence of regions.

Given a trajectory Tr_i, we define $Tr'_i = encode(Tr_i)$ its encoded version. An encoded trajectory Tr'_i is a sequence regions $Tr'_i = R_{i,1}, R_{i,2}, \cdots, R_{i,n}$. The encoding is performed by substituting each point p_i with the region $R_{i,j}$ it belongs to (here we use index i to denote that the region R_j appears in trajectory i), indeed many points could fall in a given region, in this case we can either consider that region multiple times or consider it only once and assign as overall timestamp the one assigned to the last point belonging to the trajectory and falling in that region (the choice will depend on the clustering strategy we choose as will be clear in next sections). 1 implements the regioning strategy.

Function $PCAFAst$ run PCA algorithm on the original trajectory (each row in the initial matrix is composed by the sequence of points in each trajectory) using the algorithm in [SP07] and then choose the region dimension exploiting the information about data density along the principal axes. The output is a set of annotated regions and "white" regions.

Function *locate* simply returns for a given point p_i the region in R_{PCA} it belongs to, while function *append* add the region to the current encoded trajectory. In more detail, function *append* has to check if the last added region

Algorithm 1 PCA regioning pseudo code

```
INPUT: a set trajectory T represented as sequences of points
OUTPUT:a set of trajectory T' represented as sequences of regions
VARS:
$R_{PCA}$: the region partition obtained by PCA
$r_{add}$: a candidate region for appending in the output trajectory
$R_{PCA} = PCAFAst(T)$;
$T' = \emptyset$;
for each $(Tr_i \subseteq T)$ do
  $Tr'_i = \emptyset$;
  for each $(p_i \subseteq Tr_i)$ do
    $r_{add} = locate(p_i, R_{PCA})$;
    $Tr'_i = append(Tr'_i, r_{add})$;
  end for
  $T' = T' \bigcup Tr'_i$;
end for
return $T'$;
```

in the encoded trajectory is the same of the one being currently considered (r_{add}), in that case simply update its timestamp, otherwise append r_{add} to the encoded trajectory. The encoding phase allows us to obtain a set of trajectories that could be mined using the algorithms that will be exploited in next sections.

This regioning is a lossless operation since even if we transform a 2-D space in a 1-D one we are interested in clustering data, i.e. grouping data according to a suitable distance measure that in this case as will be clear in next section is not affected by regioning.

5.3 Mining Frequent Pattern

SWIM [MTZ08] exploits the well-known $fp - tree$ [HPY00] data structure, which allows a compact representation of transactions. SWIM splits the window into several slides and then for each slide S, inserts the transactions in a separate $fp - tree$. Then, it computes the frequent itemsets in this small slide using any of existing static-data frequent miners (e.g. FP-growth [HPY00]). Since slides are mined separately and the occurrence of each pattern should be counted over all slides (to determine the total frequency), counting becomes a major bottleneck of the algorithm. The counting in SWIM is thus performed using a separate fast algorithm, which is based on *conditional counting* (=verification), called verifier. The verifier internally uses another $fp - tree$ to store the patterns that need to be counted. This latter tree is called a Pattern Tree (PT). Then, counting is performed via conditionalizing both the pattern tree the $fp - tree$ of the slide in parallel. More details can be found in [MTZ08].

While SWIM was originally designed to discover only frequent (transactional) patterns, maybe it can be easily extended to sequential patterns as well. The plan is on modifying SWIM algorithms based on the following observation. The main difference between mining transactional patterns and sequential ones in a tree-based approach is due to the importance of the order of the single items in each subset. In other words, if looking for a sequential pattern, say $< a, b >$ and see an ordered transaction [1], say $< c, b, a >$, the data structure must be able to realize that the requested order is not satisfied.

To solve this issue, an "augmentation" to the traditional $fp - tree$ data structure has been proposed, as shown in 5.3, since the normal $fp - tree$ is not able to capture the order between the items in the original transactions.

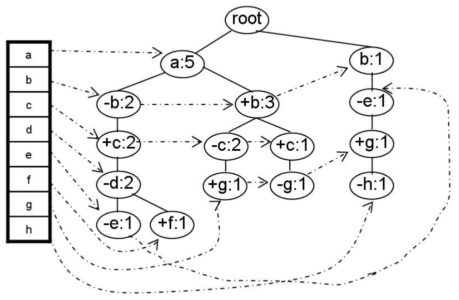


Fig. 5.3. Augmented $fp - tree$ to support sequential patterns.

Note that a naive approach of reserving a different path for each different order of the same set of items, will not be applicable, as it results in an exponential explosion of the tree. This explosion is due to loosing compactness of the tree (i.e., overlapping of the same single items) which is the main feature of the $fp - tree$ data structure. However, our proposal, shown in figure 6, will double the number of the single items, by introducing negative and positive versions for every original item. As a running example, consider the kid database of 5.3. For example, item b will be replaced with $-b$ and $+b$, in order to differentiate between the predecessor and successor items. In 5.3 all the itemsets that include a and then b will be stored under $< a, +b >$ and those that include b and then a will be stored under $< a, -b >$. Note that this schema will result a significant improvement on memory, compared to the naive scheme in which each order of the items produces a unique path in the tree. Denoting the average transaction length with L, the naive approach can generate up to $L!$ different paths for the same set of single items, according to their appearance order. But this scheme can only take up to at most 2^L different paths, as each item in the transaction has only two states

[1] For instance, the items within the same transaction could be ordered based on their timestamps.

compared to its parent: either it precedes its immediate parent or it comes afterwards (respectively, annotated with $-$ and $+$). To further clarify the advantage of many-to-one representation of the sequential transactions, note that both $< b, a, d, f, c >$ and $< b, d, f, c, a >$ will be stored under the path of $a, -b, +c, -d, +f$.

Consequently, the search algorithm for normal (=transactional) patterns can be adapted to sequential patterns by annotating the single items. Having a search algorithm for sequential patterns in the augmented $fp - tree$, the other operations (except mining) will be naturally adapted as well, including conditionalization, verification.

TID	**Single items (ordered by their timestamps)**
T_1	$e\ d\ b\ c\ a$
T_2	$b\ a\ d\ f\ c$
T_3	$g\ a\ b\ c$
T_4	$a\ c\ g\ b$
T_5	$c\ a\ b$
T_6	$h\ e\ b\ g$

Fig. 5.4. A kid database of sequence transactions.

6

Pattern Queries

6.1 Introduction

Query language extensions that allow users to search for patterns in sequences have long been recognized as important by database researches who made seminal contributions toward solving this problem [SLR94, SS95, Ses98, RDR^{+}98, PP99, SZZA01a, SZZA01b, SZZA04]. The need for such extensions has been recently recognized in many and diverse application areas, such as financial services services [SZZA01b], RFID-based inventory management [LWL07, BWL^{+}07], click stream analysis [SZZA01b]. publish-subscribe [SZZA01b], and electronic health system [HH05]. In fact, this application pull is so strong that vendors and DSMS startup companies are proposing new SQL standards for pattern queries [ZWCC07]. The pattern SQL in [ZWCC07] is largely based on the Kleene-* query construct introduced in [SZZA01a].

While Kleene-* query language [SZZA04, ZWCC07, ADGI08] have been shown to be applicable to both databases and data streams, the queries expressed in these languages will execute poorly unless sophisticated query optimization techniques are used. Ideally the goal is to have a general optimization model that defines a wide spectrum of logically equivalent executions to which different cost estimates can be assigned depending on the specific computational environment. Relational algebra provided the basis for such optimization models for the traditional queries of relational DBMS; but new models are needed here since they must deal more complex computations and ordered sets of tuples rather than the unordered ones of database relations. In fact optimization models are needed in such a way that they can adapt to the many possible environments under which these queries will be executed. Indeed any combination from the following pairs represents a possible execution environment:individual queries or query flocks, serial execution or parallel processing, data bases or data streams. In reality, good optimization techniques have already been proposed for specific environments. For instance, a generalization of the Knuth-Morris-Pratt algorithm was proven in [KJP77] to be very effective in minimizing backtracking on individual queries

- whereby the average performance was improved by two orders of magnitude. Also significant speed-ups on the execution of multiple concurrent queries on data streams were reported in [ADGI08] using FSA-oriented models. These specialized techniques must be integrated and generalized to the different execution environments described above, and re-targeted to different optimization objectives (e.g. minimization of overall execution time for DBMS, and minimization of response-time or memory in DSMS).

6.1.1 Match oriented approaches

Baeza-Yates and Gonnet [BYG96] studied how to index regular expressions by using a trie [MM90, McC76, Ukk95]. The *regex* search can be performed directly on a trie and thus given a small problem size this approach is very effective, however given a large set of patterns this approach suffers due to both space and time complexity. Aho-Corasick (AC) [AC75] generalizes KMP in order to handle sets of strings. New ideas presented by AC include keyword trees, failure functions and output links. By having multiple patterns, the naive extension of KMP would have a time complexity of $O(n + zm)$ where z is the number of patterns, however what AC did was to provide an algorithm that runs in $O(n+m+k)$ where k is the number of occurrence of some pattern P_i in text T.

In Aho-Corasick a keyword tree is constructed which can be done in $O(n)$ where n is the total length of all patterns. A naive approach would simply use this tree and align each character of the text against this tree, however this would produce a $O(nm)$ running time. Aho-Corasick uses KMP ideas to speed this up.

Matching text has many useful applications, nevertheless, XML also is widely used and requires no less research into pattern matching. [yfi02] introduces YFilter which is a way to build an NFA to match XML documents. The basic idea is we merge queries together by exploiting commonality among path queries and combine them into a single NFA. The authors specify four basic pieces (/a, //a, /* and //*) which represent possible parts of the path expression. By representing these pieces as separate state machines, the authors use these building blocks to combine them into a single machine. The resulting machine consists of states, where each state stores the following information:

1. ID of the state
2. Type information
3. Hash table for transitions [symbol — ID]
4. For accepting states, ID list of queries

The authors show that their YFilter approach is in fact faster than XFilter and the hybrid approach. YFilter remains one of the dominant works in XML document parsing.

There is a large body of literature that deals with matching regex on string using a finite state automata (FSA) [BS86, Brz64, HMU06]. Usually a *Deterministic Finite Automaton* is constructed for a particular pattern (regular expression) and then it is used to match a particular text. This constructed DFA can be saved and re-used for comparisons against a similar *regex*. In our case however, due to the large size of the number of patterns, the scaling issue becomes a problem and without good parallel implementation the performance suffers.

6.1.2 Index oriented approaches

There have also been methods that deal with indexing of intervals.

Traditional indexing techniques rely on multidimensional indexes such as the R-tree which minimizes the number of insertions, however because the performance deteriorates exponentially as the number of dimensions grows these approaches are not efficient to process large set of queries.

Authors in [LM92] describe a checkpoint and indexing scheme that is used in temporal database environment. The generalized data stream index is developed that can efficiently process a subclass of joins qualified with a snapshot operator.

Authors in [WBTG06] present a query index based on a decision tree. The index presented is applicable to queries consisting of conjunction relationship between predicates. The authors study methods of how to select a dividing attributes during tree construction which is crucial to index performance. Two dividing attribute selection algorithms are presented: *Information Gain (IG)* and *Estimated Time Cost (ETC)*. The experiments presented suggest that tree built using ETC is more efficient, however given a dynamic nature of queries, this approach would not be applicable.

FREE algorithm is proposed in [CR02] FREE is a pre-built index to speed up regular expression matching on large text databases. By using careful indexing techniques based on *selectivity* speed-ups of two orders of magnitude were observed. Also a proposed multi-gram index provides size reduction of orders of magnitude without degradation in performance. The FREE algorithm, however, works by indexing on the underlying data and not on the regular expression itself.

Hashed multiple lists were proposed in [LJLR08] to speed up continuous query evaluation in data stream environment. The proposed technique decides the level of node based on a binary tree and uses a hash table to speed up the search. Because a hash-table is used, the approach was able to search linearly when number of index objects increases.

Very small body of work exists regarding the indexing of Kleene closure. The work is mostly comprised of languages such as SASE+ [DIG07] and SQL-TS [SZZA04] and application specific indexing methods such as query indexing of RFID data streams.

SASE+ proposes a compact language that can be used to define a wide variety of Kleene closure patterns. The authors rigorously studied language semantics and analyzed the expressive power of SASE+, SQL-TS and Cayuga. Also it was shown that these languages belong to several standard complexity classes.

Authors in [PHB07] present a technique for indexing complex continuous queries. It was observed that due to large storage and build time requirements of Kleene closure an aggregate transformation technique to index a group of segment was necessary. The aggregated data developed is representative of the system and therefore only a small fraction of all objects (segments) needs to be stored. A new tree, $KDB_{AT} - Tree$ was designed to support aggregate transformation.

All of the methods presented do not deal with indexing and evaluation of arbitrary patterns consisting of predicates with Kleene closure, general functions in a streaming environment which is the focus of this paper.

In next section is discussed one of the most speculative objective that focus on discovering patterns represented by Kleene-* expressions. Many successful methods of knowledge discovery are based on the simple idea of learning a descriptive model from a set of training examples, and then using that model to predict the class to which new instances belong to. Current learners use predictive models based on different representations, such as decision tree, probability vectors, and neural nets.

6.2 Efficient Pattern Matching over Streams

With a rapid development of the internet more and more data stream applications have emerged in recent years. *Continuous queries (CQs)* are an important type of queries that are issued on a data stream. CQs are registered in a *Data Stream Management System (DSMS)* in advance and are executed on a continuously arriving data stream. An important type of CQ that is widely used is a filtering query. A *filtering query (FQ)* is a condition on a stream of data that is checked on each arriving data tuple. If FQ condition is evaluated to *true* we say that an FQ is *satisfied.* A satisfied filtering query will trigger an action in a specified application system. FQs are used in data stream applications involving sensor network (RFIDs), financial applications, network intrusion detection and so on.

An FQ is a query defined on a data stream and characterized by a relation R $R(a_1 : \omega_1, a_2 : \omega_2, \ldots, a_m : \omega_m)$ where ω_i is the domain of attribute a_i, and m is the number of attributes. Thus an FQ is defined as a conjunction of m predicates, where predicate i can depend on any attribute a_i, $FQ = P(a_1) \wedge P(a_2) \wedge ... \wedge P(a_m)$.

6.2.1 Importance of Kleene-* in continuous queries

The amount of data in databases grows at a tremendous rate which makes simple tasks such as matching difficult. Matching of a regular expression (i.e. finding all occurrences of a particular string in text) in this large set of data therefore presents a challenging task. It was noted that in order to execute a regular expression matching algorithm on a large text database of size 100 GB may take several days [CR02]. The goal to reach is a matching problem given a streaming environment and a large number of queries (patterns).

Of particular importance are Kleene closure patterns which can be used to extract an unbounded number of objects from a data stream. The extracted objects can represent events which are of growing importance in emerging applications including financial services, RFID-based inventory management, click stream analysis, electronic health record system etc.

While indexing of patterns involving general predicates was well studied, indexing of and functional patterns over streams have unique issues regarding scalability in both performance and size of the index

Several notable languages exist that support Kleene closure including SQL-TS, MATCH-RECOGNIZE and SASE+. MATCH-RECOGNIZE comes from a 2007 ANSI standard proposal to add new SQL functionality for finding patterns defined as regular expressions over sequence of rows via a MATCH_RECOGNIZE clause. The new syntax allows pattern definition via a PATTERN component which is used to specify the regular expression. Some of the operations supported in the PATTERN component are: * (0 or more rows), + (1 or more rows), ? (0 or 1 row). In addition, aggregate operators can be used on group variables like AVG(A) where A is a group variable. In addition columns of the matched rows are also accessible via D.price > A.price.

SQL-TS (*Simple Query Language for Time Series*) is another main proposal for SQL language extension to support pattern queries. SQL-TS, besides proposing a set of new language constructs also discusses optimizations of pattern queries. An extension of the well-known Knuth, Morris, and Pratt (KMP)[KJP77] algorithm is introduced in order to optimize pattern queries.

6.2.2 Real-world example

In financial services area, a stock broker may want to retrieve stock transaction that caused the price of a stock to increase steadily which resulted in a steady increase of the trading volume. In a retail store equipped with RFID capability, a store manager would like to capture events that signal shoplifting activity which are caused by abnormal inventory depletion. Example 6.1 display an SQL-TS query to find the maximal periods in which the price of a stock fell by more than 50%, and return the stock name and these period. Efficient evaluation of these types of queries serve as motivation of this paper.

Example 6.1.

```
SELECT X.name, X.date AS start_date,
            Z.previous.date AS end_date
FROM quote
            CLUSTER BY name
            SEQUENCE BY date
            AS (X, *Y, Z)
WHERE Y.price < Y.previous.price
      AND z.previous.price < 0.5 * X.price
```

6.2.3 KMP and OPT

Next is briefly discussed KMP and OPT approaches which are successful string matching algorithms.

Removing inequalities, functional predicates and Kleene closure from query in example 6.1 it is possible to invoke algorithm by Knuth, Morris and Pratt (KMP) which provides a solution of proven optimality for this query. A KMP algorithm takes a sequence pattern of length $m : P_1, P_2, \ldots, P_m$ and text T of length $n : t_1, t_2, \ldots, t_n$ and finds all occurrences of P in T. The algorithm starts from left to right and compares successive characters until the first mismatch occurs. At each step the i^{th} element in the text is compared with the j^{th} element in the pattern. i and j are kept increasing until a mismatch occurs.

When mismatched, a naive algorithm would reset j to 1 and i to the next position from where matching began. Instead, however, KMP avoids backtracking by utilizing knowledge acquired by the fact that some characters might have matched. Thus, KMP is an algorithm that exploits the relationship between elements in the pattern to never have to backtrack on the text.

Unfortunately, KMP algorithm only works when qualifications in the query are equalities with constants. An *Optimized Pattern Search* (OPS)[SZZA01a] algorithm is proposed to extend KMP to be directly applicable to SQL-TS queries. Thus, OPS is applicable to more general conditions that often occur in time-series queries including Kleene closure. OPT, similar to KMP, exploits relationship between elements in the pattern in order to infer shifts of the pattern that can possibly succeed and shifts which need not be checked.

6.2.4 Brief overview and contributions

In order to take proper actions in many applications including financial streams a large number of range queries or filtering queries must be created and continuously executed against the running stream. For example in sensor network stream applications a large number of range queries must be created to monitor temperatures, flows of traffic and other readings.

Thus the focus is on scalability problem of evaluating thousands of queries containing Kleene closure and general predicates over data streams. One approach to accomplish this task is to have a query index. On each incoming tuple, an index is searched for matching queries.

Conceptually this is a simple task, however designing a memory efficient, fast index in a rapid streaming environment is quite a challenge due to the memory and performance scalability issues. Thus indexing and string matching approaches can be combined to overcome these challenges.

The solution proposed is perhaps the easiest and most intuitive, nevertheless it produces promising results. Even in this simple solution to the problem we have a large number of interesting choices to make. Considering N queries similar to those in example 6.1, it is possible to see the challenge of this problem due to the number of queries, general predicates and functions as well as Kleene closure. A full indexing approach would produce a large memory-print and would not scale well in performance. Thus, this simple example paves way to many important questions: *How do build an index on these queries? How do we determine query order execution to achieve best performance? How to we index Kleene closure and general predicate functions?*

A simple answer to these questions would be to store entire patterns in a multidimensional tree with all possible permutations for both functions and Kleene closure. This approach however is clearly unrealistic due to hefty requirements for both build time, storage and maintenance of such an index.

This approach assumes a large number of patterns. Furthermore assume a representation of patterns as queries similar to those defined in SQL-TS [SZZA01a] where each element of the pattern can be defined as a constraint consisting of general predicates or functions such as $x > previous$ where *previous* is a value of the last tuple. The proposal is related to a hybrid approach which combines an indexing method with a string matching approach. Our algorithm indexes first K elements of each pattern using a multidimensional tree or a multidimensional hash table, thus given an input $i_1, i_2, \dots, i_k$ it is possible to eliminate irrelevant queries that do not match. Remaining queries are matched using an existing string matching approach such as KMP. The algorithm seamlessly shares information between these seemingly very different phases.

6.2.5 The Problem

In this section is provided the general architecture of the algorithm. Its goal is to fast process multiple pattern queries in a data stream environment. The trivial solution is to visit each pattern predicate and test whether it satisfies a given input i, which requires $O(m)$ time, where m is the number of predicates in the pattern. However, by observing the fact that a *mismatch* is more common than a *match* it is possible to use an indexing technique to eliminate queries that will never much. This task is not trivial due to the need to index not only *regex*-like queries but queries that contain general predicates and functions. As far as anyone knows there exists no literature which addresses pattern matching given that definition.

For discussion purposes, assume that a large number of queries Q_1, ..., Q_n is available. Each query consists of a sequence of states, and say that a simple condition, such as $var > C$, is checked at each state. It is important to note that C can be an interval or an equality as well as a general function such as $x > previous$ where *previous* is the value of the last tuple.

Figure 6.1 represents a set of queries consisting of inequality predicates. Q_{ij} represents a predicate j of query i. Given a set of K inputs we want to output which queries are evaluated to true by these K inputs.

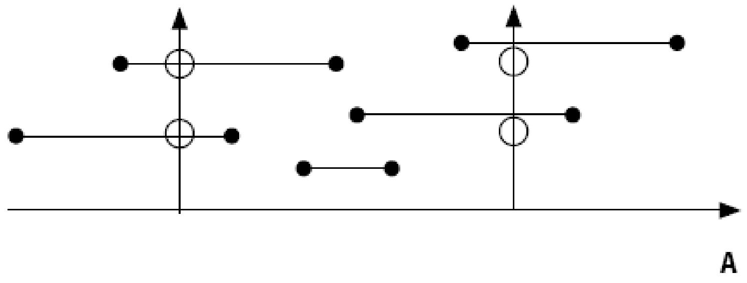


Fig. 6.1. Query evaluation

Given a stream environment, matching must be performed quickly and should scale well with the number of queries. Given a continuous, bursty and fast arrival of tuples our framework must be fast and also have a small memory footprint.

It is interesting to deal with Kleene closure, as the size of the index grows exponentially when Kleene closure are used. As discussed later, indexing of Kleene closure requires generation of up to $O(2^{K-1})$ more patterns, however this is not a problem in the current case. When matching Kleene closure patterns a strategy of *maximal match* is employed where for each input i it always tries to match a Kleene closure element first.

As discussed in the introduction example 6.1, functional predicates such as $Y.price < Y.previous.price$ must also be efficiently supported. The problem with functional predicates is that the range of possible values is unknown a-priori and this makes indexing of these functions challenging.

The core is to efficiently organize query predicates in an index structure that quickly points to the states of the queries that are satisfied by the input values. The key idea is to construct our index on the first K predicates of each query. Later it will be shown that the optimal value of K which is extremely small relative to the length and number of queries.

It is important to note that any indexing technique can be used, thus this approach can be thought of as a *Black Box* approach where any appropriate algorithm can be applied. Experiments are run focusing on basic Hash Table index and Tree index to run. As an example, given an input value A, B and $K = 2$ then index would tell that:

1. The input value A satisfies state 1 of Q_1 and Q_3, input value B satisfies state 2 of Q_2 and Q_3 etc.
2. Therefore given these two inputs Q_3 is returned as a *candidate match* query.

Thus, by combining results returned on K input values, it is possible to determine the queries that are satisfied (or might be satisfied) in their first K states. Ideally, only a few of the initial N queries are satisfied by the K input values, however it is interesting to address the case when it is not possible to eliminate most of the queries based on initial K inputs by employing query *strength*, which is a query ordering technique, as discussed later. After the initial indexing phase, fulfilment the check of the validity of the *candidate queries* by using the standard matching algorithms such as KMP or OPT. Any matching algorithm can in fact be used, thus the matching phase can be considered as another *Black Box* in the proposed algorithm.

The optimal value of K to be used depends on the nature of the queries and their number, N. For $N = 1$, $K = 0$ it falls back to the standard approaches. As shown later K depends on the average number of states in the queries, and never exceed that.

6.2.6 Architecture overview

A high level overview of the discussed approach is depicted in figure 6.2. Our system architecture consists of four main components: 1) a query execution stage, 2) an indexing phase 3) a filtering phase and a 4) matching engine. The user of the system interacts with the query execution phase through which the queries are submitted and matched queries are retrieved. We briefly go over some of these components now and discuss them in detail later.

Query execution and indexing can be thought of as a light pre-processing stages that are meant to eliminate as many pattern queries as possible as described above. *Candidate queries* are passed onto the *Matching Engine for validation*. Based on the results from Matching Engine a *filter* is employed in order to eliminate those queries returned by the index that will never match based on the results from the Matching Engine. *filter* step is also responsible for generating query execution plan which is based on query ordering.

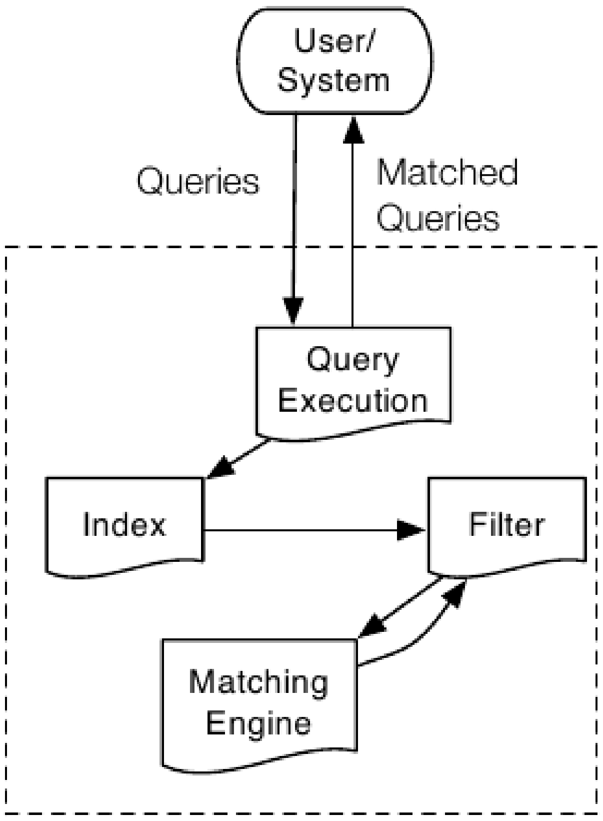


Fig. 6.2. System Architecture

6.2.7 Query ordering

The order of query execution is an important issue, and it will play a major role in query performance. A well known problem, *MQP (Multiple-Query Processing)* exists which deals with the order of query execution.

Definition 6.2. *MQP: Given n sets of access plans $P_1, P_2, \ldots, P_3$ with $P_i = P_{i1}, P_{i2}, \ldots, P_{ik}$ being the set of possible plans for processing Q_i, $1 \leq i \leq n$, and a constant K, is there a global access plan GP such that the cost of GP is less than equal to K?*

Based on the definition 6.2, if MPQ is NP-complete then our query order optimization problem is NP-hard. Authors in [SG90] prove that MPQ is indeed NP-complete, by first noting that MQP belongs to NP since a nondeterministic algorithm can pick 1 query plan and check whether the cost of all merged local plans (to produce one global plan) exceeds that of K. The authors then show that MQP is NP-complete by transforming 3SAT to MQP.

Thus, a heuristic algorithm to optimize query execution is provided. Given N queries $Q_1, Q_2, \ldots, Q_n$ these are ordered based on *query strength.*

Query strength is computed by observing the number of implications Q_i has between other queries.

Definition 6.3. *Implication: Predicate p_i implies predicate p_j iff p_i and p_j are some selection predicates on attribute A of some relation R and $Q_{p_i} \neq Q_{p_j}$ and it is the case that for any input i the result of evaluating a predicate p_i is a superset of the result of evaluating p_j.*

Definition 6.3 is used to find *negative implications* where a mismatch on a predicate p_j would imply a mismatch on a predicate p_i. Note that predicates are identical if p_i implies p_j and p_j implies p_i. It is possible now to define implication between queries.

Definition 6.4. *Query implication: Query q_i implies query q_j iff for a specified number of predicates U the number of predicate implications in q_i is greater than that of in q_j.*

Query strength is more formally defined later, but to get an intuition consider example 6.5.

Example 6.5. Consider a set of queries:
Q_1: EMP.age $\leq$ 40, EMP.ID $>$ 30, EMP.dept-id $<$ 4
Q_2: EMP.age $\leq$ 30, EMP.ID $>$ 20, EMP.dept-id $<$ 5
Q_3: EMP.age $\leq$ 20, EMP.ID $>$ 10, EMP.dept-id $<$ 6

One way to order these queries based on the first predicate (when $U = 1$) is Q_3, Q_2, Q_1. It is important to note that probability of a mismatch is assumed to be greater, thus *negative implication* is used to order queries, and the order when $U = 1$ would be reversed.

6.2.8 The index

Once the queries are received from the user they are indexed, but only considering the first K states (predicates) of each query.

Furthermore it is clear that it is inefficient to pass queries from indexing phase to matching phase without any order because of possible implications between queries which can be leveraged. By using the implications it is possible to further eliminate queries that will not match in the *Matching Engine.*

We refer to the *query order* of the queries as *query strength.* Query strength stems from the observation that there are more mismatches then matches when evaluating queries. Therefore by first evaluating queries that would help *eliminate* other queries if mismatch occurs would provide a more efficient execution model. *query strength* of Q_i denotes the number of queries that can be eliminated if Q_i mismatches on predicate p_i. Therefore order queries are ordered based on their *query strength.*

6.2.9 Query Execution Manager

Query Execution Manager reads a list of queries provided by the user or by the system and returns matching queries. 6.3 shows a more detailed workflow of the execution manager. The main process consists of three main stages: *(i)* reading of queries and constructing an implication graph of U states of each query, *(ii)* computing the strength of queries by measuring the in-edges of each node and finally *(iii)* ranking of the queries based on their strength.

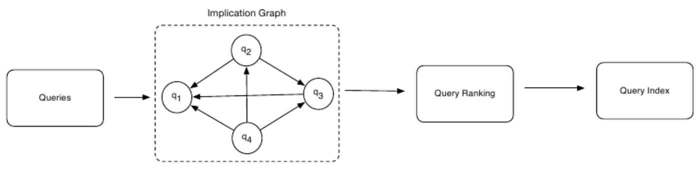


Fig. 6.3. Workflow of the execution manager

The execution system of this approach is appealing for several reasons:

1. It is easily implementable in conventional database systems
2. Parameter U and ranking of queries based on *query strength* have direct impact on performance of the queries.

The following example is useful to better illustrate some of the ideas presented above:

Example 6.6. Consider a large number of queries similar to example 6.1. We can make 2 observations:

1. We can vary K or the number of predicates in query Q which we will index. By varying K we can eliminate a certain number of queries given K input values. The remaining *candidate queries* are matched using standard string matching approaches (e.g. KMP). Later is established the fact that *optimal* K occurs where time to index is equivalent to time to validate *candidate queries*.
2. After indexing stage, there can be a substantial number of queries left, especially if N was initially large. We can also observe that the order of queries that are passed onto the Matching Engine matters. We will show how we can leverage this query order by using *query strength*.

Index structures

As stated before, we use a Tree index and a Hash index to implement the indexing stage of our algorithm. As also noted, any indexing technique can be used that fulfils the requirement of the user. The chosen index will index K states of N queries in hopes of eliminating most of N queries upon arrival of input i. After this initial elimination, *candidate queries* are verified using standard matching approaches.

Here for a tree index is used a multidimensional *interval tree*. An interval tree is an ordered tree data structure to hold intervals. Specifically, it allows one to efficiently find all intervals that overlap with any given interval or point. It is often used for windowing queries, for example, to find all roads on a computerized map inside a rectangular viewport, or to find all visible elements inside a three-dimensional scene.The interval tree employs a red-black n-node tree and therefore the height of this tree is $O(logn)$. Interval-tree therefore takes $O(logn)$ time to find an overlapping node.

In order to build a hash index we use a straight forward multidimensional hashing approach for testing. For better performance, approach such as in [LJLR08] can be used which proposes a combination of the binary tree and a hash table to deliver dynamic hash table support.

Insertions and deletions are performed by utilizing existing approaches. Given a tree index T and a new pattern query Q_{new}, Q_{new} can be inserted into T in $O(logn)$ time.

Given a hash index, upon arrival of a new query in our tests algorithm simply maps the new query to the hash. In approaches suggested by [MW94] upon insertion or deletion of a query from a hash table it performs additional adjustment of the nodes for efficient retrieval later. Given an input i searches can be performed on a tree in $O(lgN)$ time where N is the number of queries in the tree. Hash performance is dependent on the number of intervals (buckets) in the hash. Evaluations are performed where the number of buckets varies and generally achieve better performance than using a tree index.

Overview of the matching engine

As mentioned previously a KMP/OPT approach is employed to carry out the validation procedure of validating *candidate matches.*

1. Input: A set of candidate queries from *Index stage*
2. Output: Matched Queries

In Figure 6.2 *Matching Engine* is responsible for analysing which of the candidate queries that was generated as the output from *Index* actually is a match. This phase uses standard approaches to accomplish this task. This phase also sends information to the *filter* phase in order to notify *Indexing Stage* of any queries that will never match on input $i+K$ where i is the current input and K is a value that *Indexing Stage* will use to *eliminate* irrelevant queries based on the first K states.

Once the execution of *Matching Engine* is complete, the next time K inputs arrive and once again irrelevant queries are eliminated. *Matching Engine* however could have eliminated some of the queries which might be returned by *Indexing Stage*, therefore it is needed to intersect the set of queries which are returned by *Indexing Stage* with those returned by *Matching Engine* efficiently. *Filter* is also responsible for query ordering as defined above. Experiments section confirms that given large number of queries returned by the index, query ordering speed up evaluation.

6.2.10 Indexing of Klene Closure and functions

It needed now to determine which is the optimal value for K, given N number of queries. First we need to define what we mean by optimal K. Because K refers to the length of the sub-pattern which is indexed, the following idea is valid:

Definition 6.7. *Optimal K: Given a pattern P of length M, an optimal K is an integer such that the time it takes to index $P[0 \dots K]$ using an index stage is equal to the time it takes to match $P[K+1 \dots M]$ using the Matching Engine.*

In the following is proved that optimal K will give us optimal matching performance Intuitively it is possible to say that if K is smaller, than it takes longer to match, if K is larger, then it takes longer to index.

Based on these considerations the optimal value can be determined experimentally. 6.4 presents time complexity for naive and indexing approach. The time for indexing approach is cumulative, and as we can see, according to definition 6.7, the optimal value for K (or the number of states that need to be indexed by this indexing approach is $C * lg_2(N)$, where experimentally it has been determined that C is 0.5.

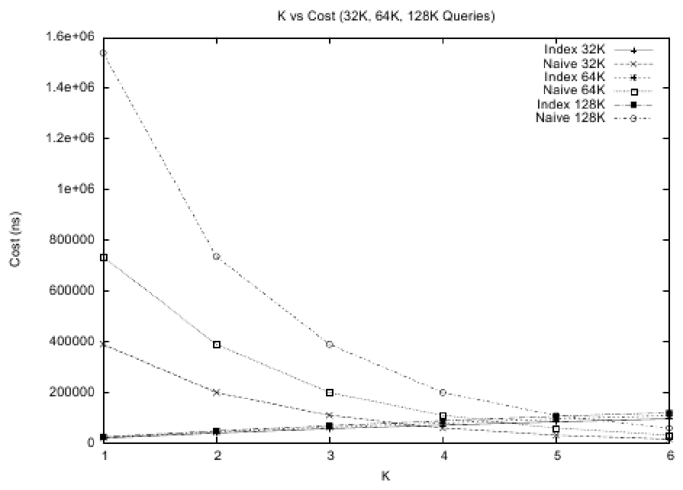


Fig. 6.4. Optimal Search for K (32K Queries)

Using a hashing approach experiments shows that in order for the hashing approach to make sense, about $\frac{1}{16}$-$\frac{1}{32}$ partitions need to be made of equal size.

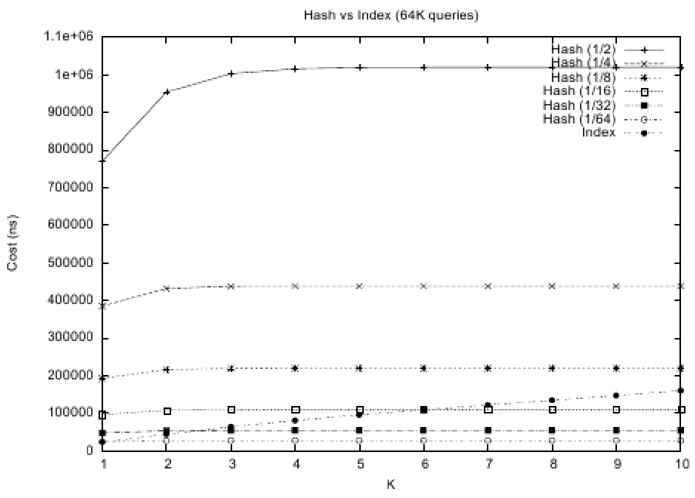


Fig. 6.5. Hash VS index graph showing costs according to different hashes

6.2.11 Dealing with stars

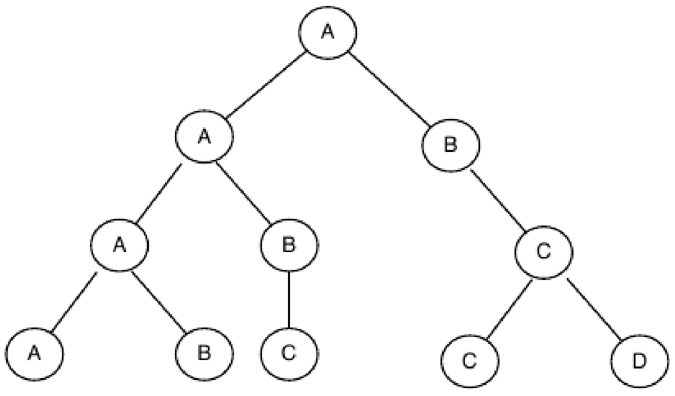


Fig. 6.6. Regular Expression Permutation

Knowing that K is relatively small, an approach is needed to simply enumerate all possibilities of the Kleene closure pattern and index them. Consider a regular expression R $A^{+}BC^{+}D$. In 6.6 are represent all possible expressions of R using a binary tree. Consider $K = 4$. As already known, the maximum possible number of permutation of pattern P with Kleene closure $+$ is $O(2^{K-1})$ where K is the height of the tree. Therefore if M is the number of queries with Kleene closures an overhead added to do indexing is $O(M2^{K-1})$. Because K is generally small, it is possible to index this amount of patterns.

Below there are two figures that show memory (6.8) and execution speed (Figure 6.7) of indexing in the presence of Kleene closure.

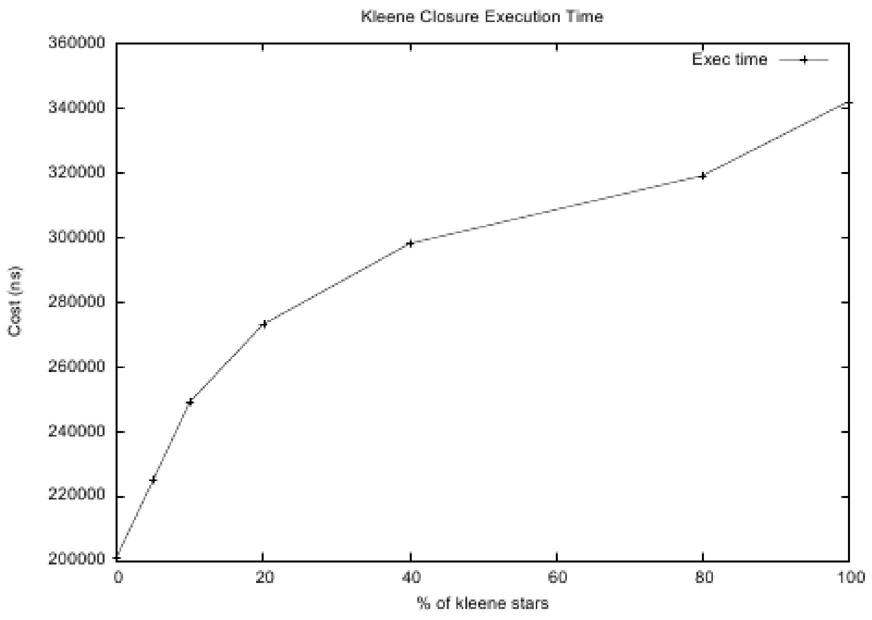


Fig. 6.7. Indexing speed and Kleene closure

It is clear that in most cases the overhead shown is acceptable.

6.2.12 Dealing with functions

As stated before, this algorithm also is capable of indexing general functions in the form $y > Previous$ or $y > previous + 10$. The approach considered is similar to that when indexing Kleene closure. The idea is to enumerate all possibilities, however it is a good point to further minimize the number of permutations by considering implications among functional predicates of query Q_i and previously matched states of query Q_i. Once again, because K is relatively small the overhead is limited.

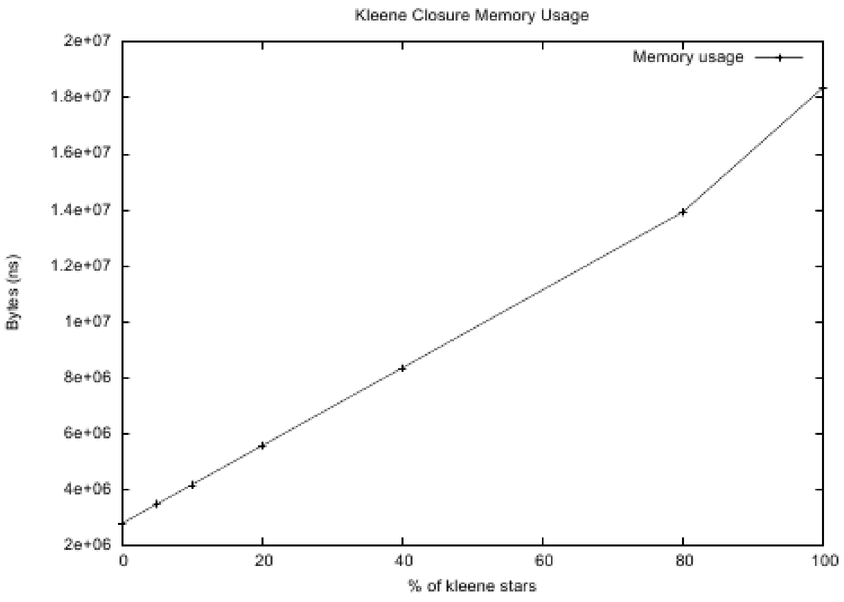


Fig. 6.8. Memory usage of Kleene closure

An example of minimizing the number of functional predicates is presented in figure 6.9.

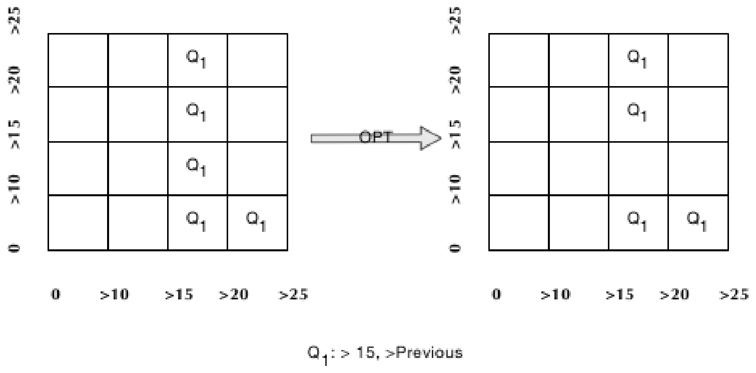


Fig. 6.9. Minimization of the number of functional predicates

6.2.13 Query strength

It i suseful to discuss about the meaning of *query strength* and the role it has on the whole algorithm.

Definition 6.8 (Query order). *: an order of a given set of queries represents an order of query evaluations given input i. This order is represented as acyclic graph P = (V, A, L) where (V, A and L represent sets of verticies, arcs and vertex labels repsectivley) defined as follows.*

1. *For every query Q_i we introduce a vertex v_i.*
2. *If Q_i implies Q_j we introduce an arc from v_i to v_j.*
3. *DFS traversal is done and queries are ordered by the number (in decreasing order) of incoming arcs (hence negative implications).*

Definition 6.9 (Query Strength). *: using definition 6.8 query strength of Q_i is greater than of Q_j if Q_i appears before Q_j in query order.*

We can thus summarize the initialization of the index in algorithm 2. The running time of the indexing phase is based on the underlying index structure and we evaluate its performance in the experiments section.

Algorithm 2 Initialize

```
1: Given $N$ queries, expand first $K$ states containing Kleene closure or functional
   predicates
2: Compute query strength of resulting queries
3: Initialize the Tree index or Hash Index with $K * N$ expanded predicates
4: Given $K$ inputs, evaluate them using an index and return candidate queries to
   the filter stage.
5: Based on |candidate queries| applying ordering by strength to these queries
6: In the filter stage, filter out queries that were determined to be unnecessary in
   the Matching Engine
```

After *Indexing* and *filter* stages, remaining *candidate queries* are passed to the *Matching Engine*. As stated previously, the *Matching Engine* is responsible for validating the *candidate queries* on the remaining set of inputs. The *Matching Engine* also updates the *filter* state with a set of queries that are possible to be executed on the next input.

As noted before, the running time of this phase is based on the running time of the matching algorithm. Algorithm 3 summarizes our matching framework.

6.2.14 Experiments

The experimentation part is not coompleted yet and it will be conducted considering synthetic data and enforcing the comparison against other techniques.

Algorithm 3 Matching

```
1. Input: Query Set $Q$ over data stream $S$ and window $W$. $Q_i$ is a pattern query
   which consists of any general predicates.
2. Output: Remaining patterns after matching $K$ states of each pattern.

1: For each query received from *filter* stage
2: invoke KMP, OPT etc..
3: Upon a mismatch for query $Q_i$, update *filter* with a next possible starting position for query $Q_i$
4: Upon a match, return query.
```

Actually synthetic data-set queries have been generated with pre-specified number of Kleene closure and predicates. Queries generated are based on two main distributions, namely only unique queries, and queries with specified amount of overlap (potentially duplicate queries). The framework has been implemented in Java and had run tests on an Intel Core duo 2.53Ghz with 4GB of RAM.

As with every experiment it is important to select a fair set of queries to execute, however due to the lack of a standardized benchmark suites for this field of study synthetically generated data have been considered. Therefore it should be noted that the results presented are a potential of improvement of the technique and not necessarily a representative of a real-life query execution performance.

The query that have been considered are similar in nature to those posed by bio-informatics researchers. 6.2.14 specifies what kind of parameters have been set and which queries which have been used.

List of Queries
$Q_1 : ABC$ **WHERE** $A > x, B > y, C > Z$
$Q_2 : ABC$ **WHERE** $A > x, B > y, C > previous$
$Q_1 : AB^*C$ **WHERE** $A > x, B > y, C > Z$

Table 6.1. Typical query on which it is possible to run experiments

Measurements

Interesting aspects can be relevant when evaluating performaces of the proposed technique against others. It is possible to enumerate the main that in the future will have a considerable attention:

1. *Index size and Construction time K*: it is related to the size of the index as a function of the number of queries; tests will compare the Tree index, Hash index and YFilter approach.

Synthetic Query Parameter List		
Parameter	Range	Description
Q	1000 to 500000	Number of queries
W	0 to 1	Probability of a wild-card "*" occurring in a query
Distinct	True or False	Generate unique or random predicates
P	0 to 20	Number of predicates per query

Table 6.2. Parameters for synthetic query generation

2. *Execution time as a function of queries*: hash index, tree index and YFilter are compared.
3. *Effect on execution time* when ordering queries based on their *strength parameter*: only tree index is evaluated.

General performaces will be evaluated in two main ways with respect to the concept of *unique-query* and *non-unique-query* environment.

Performance (Unique queries)

Considering both the case of unique-query and non-unique queries, running the experiments against the YFilter, the nested elements are ignored. As the number of queries increases, the Hash index and Tree index will perform better than YFilter. The intuition behind the result is that this approach 1) Prunes queries quicker 2) Utilizes strength ordering and 3) Removes queries from index result which will not be satisfied ($filter$ stage).

6.2.15 Effects of query strength

Also very intersting will be the behaviour of this algorithm when measure the effect of ordering the queries by their query strength. The intuition is that as number of queries pruned decreases the performance increases when using query ordering by strength.

7

Conclusions

7.1 Summary

With the rapid development of technologies and IT applications, the need of manage efficiently data have become critical for all companies which were dealing with huge amount of data. To this purpose, researchers have proposed many techniques and algorithms which are able to optimize or improve performances of systems in challenging situations which were typical in IT environment.

While data are stored in a general Data warehouse or DBMS the problem is strictly related to manage massive data in efficient ways and at the same time user or application can easier retrieve information when data are represented in a particular way, exploiting main characteristics of OLAP systems, like hierarchical representation of data. In the case that data contains sensible information, it is needed to make sure the user accesses the system with the right privileges. Malicious users can retrieve with some inference operations information that are not devoted to, and they must be avoided in such a kind of activities. Privacy preservation in OLAP can be considered *cell-based* or *aggregate-based*. Considering the latter case, it is possible to define a whole framework which gain a good point in trade-offing between accuracy and privacy in answering queries over a OLAP datacube.

On the other side, when incoming data are like a stream, and it not possible to store everything into a DBMS, it is needed to adopt different policies to manage such a kind of data. Data Stream Management System provide *ad hoc* features and characteristics to efficiently query and analyse data streams, when particular memorization techniques are taken into account.

The problem of query incoming data streams can be very challenging if the number of queries is very big and a quick answer is needed. In this case an intelligent query scheduler might try to reduce the workload size, exploiting queries properties. Pattern queries are still very difficult to manage efficiently and frameworks which extend the main pattern matching algorithm are presented in many works. It is a still open problem that will be further investigated.

7.2 Conclusions and Further Research

This thesis is concluding by mentioning final results obtained and some future developments on further research.

The novel multi-objective data cube compression paradigm, which deals with the problem of compressing data cubes according to multiple constraints rather than only one like in traditional schemes, orients future work towards the following two main directions: *(i)* It is possible to extend the multiple-query data cube compression framework to deal with more complex data warehouse schemas such as multi-measure data cubes and fact constellations, beyond conventional data cubes like those investigated; *(ii)* extending the multiple-query data cube compression framework as to deal with OLAP aggregations different than SUM.

Same kind of analysis can be done with respect to advanced OLAP visualization techniques of multidimensional data cubes which meaningfully exploits the data compression paradigm to overcome the natural disorientation and refractoriness of human beings in dealing with hyper-spaces. In this direction, the OLAP dimension flattening process and the amenity of computing semantics-aware buckets are, to the best of our knowledge, innovative contributions to the state-of-the-art OLAP research.

Privacy preserving OLAP framework knowledge has further confirmed the benefits deriving from adopting the privacy notion and all technical aspects for the goal of preserving the privacy of OLAP data cubes, also in comparison with the method Zero-Sum, the state-of-the-art privacy preserving OLAP perturbation-based technique. It would be interesting to extend the privacy OLAP notion towards complex OLAP aggregations, beyond conventional ones (e.g., SUM, COUNT, AVG).

Data stream systems literature reported many works on how to manage incoming data. Techniques and methods for managing them are various and goal-dependent. The SensorGrid system, founds on the data compression/approximation paradigm. The experimental evaluation has been conducted on several synthetic data sets, and with regard to both window and continuous OLAP-like queries, which are very useful to extract summarized knowledge from large amounts of materialized readings stored in Grid nodes. A good point to continue is the capability of dealing with multidimensional sensor readings, which arise in several novel application contexts and the one of supporting innovative classes of meaningful queries based on complex predicates over sensor readings.

The problem of preserving order when trying to find frequent patterns inside a trajectoy is a very interesting problem that has led many ideas on how to extend already known algorithms defining a data structure called *augmented FP-Tree* in order to efficiently mine frequent sequential pattern among data. Managing more complex patterns is a good proposal for next research topics which are stimulating new ideas

When dealing with the problem of optimization of query execution in a multi-pattern-query environment, the design of a pattern matching algorithm which supports Kleene-closures, is able to work on streams and multiple patterns. System architecture aims at extend the KMP [KJP77] algorithm and its extension to Kleene closure on pattern queries [SZZA01a]. Considering preliminary results it seems to be clear that there a performance guarantee of the algorithm and also this approach is faster than previous approaches. By combining the indexing approach with a string matching approach it is possible to create a much more efficient query matching framework. Very important is at the same time the framework for ordering queries based on the number of implications among queries. The problem with KMP algorithm is that it requires compilation stage to generate the `next` arrays and does not work with Kleene closure. Finally it seems that using intelligent scheduling properties it might be possible to get very good results also in presence of Kleene-* and therefore generalize this kind of problem.

References

[ABB$^+$04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004.

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[ACTY09] Pankaj K. Agarwal, Siu-Wing Cheng, Yufei Tao, and Ke Yi. Indexing uncertain data. In *PODS '09: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 137–146, New York, NY, USA, 2009. ACM.

[ADGI08] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160, New York, NY, USA, 2008. ACM.

[APR99a] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 1999. ACM.

[APR99b] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 1999. ACM.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. *SIGMOD Rec.*, 29(2):439–450, 2000.

[AST05] Rakesh Agrawal, Ramakrishnan Srikant, and Dilys Thomas. Privacy preserving olap. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 251–262, New York, NY, USA, 2005. ACM.

[AW89] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[BBD$^+$02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[BC07] Angela Bonifati and Alfredo Cuzzocrea. Xppx: A lightweight framework for privacy preserving p2p xml databases in very large publish-subscribe systems. In *EC-Web*, pages 21–34, 2007.

[BCG01] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes: a multidimensional workload-aware histogram. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):211–222, 2001.

[BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[BFSS03] Francesco Buccafurri, Filippo Furfaro, Domenico Sacca, and Cristina Sirangelo. A quad-tree based multiresolution approach for two-dimensional summary data. In *SSDBM '03: Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 127–140, Washington, DC, USA, 2003. IEEE Computer Society.

[BG92] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, second edition, 1992.

[BG04] Wolf-Tilo Balke and Ulrich Güntzer. Multi-objective query processing for database systems. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 936–947. VLDB Endowment, 2004.

[Bha00] Bharat K. Bhargava. Security in data warehousing. In *DaWaK 2000: Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, pages 287–289, London, UK, 2000. Springer-Verlag.

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[BS86] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(1):117–126, 1986.

[BTW$^+$06] Yijian Bai, Hetal Thakkar, Haixun Wang, Chang Luo, and Carlo Zaniolo. A data stream language and system designed for power and extensibility. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 337–346, New York, NY, USA, 2006. ACM.

[BW01] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.

[BWL$^+$07] Yijian Bai, Fusheng Wang, Peiya Liu, Carlo Zaniolo, and Shaorong Liu. Rfid data processing with a data stream query language. In *ICDE*, pages 1184–1193, 2007.

[BYG96] Ricardo A. Baeza-Yates and Gaston H. Gonnet. Fast text searching for regular expressions or automaton searching on tries. *J. ACM*, 43(6):915–936, 1996.

[CcC$^+$02] Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB*, pages 215–226, 2002.

[CCC$^+$03] Sirish Chandrasekaran, Sirish Ch, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. Telegraphcq: Continuous dataflow processing for an uncertan world, 2003.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1):65–74, March 1997.

[CDD$^+$01] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proceedings of the 17th International Conference on Data Engineering*, pages 534–542, Washington, DC, USA, 2001. IEEE Computer Society.

[CFG$^+$05] Alfredo Cuzzocrea, Filippo Furfaro, Sergio Greco, Elio Masciari, Giuseppe M. Mazzeo, and Domenico Sacca. A distributed system for answering range queries on sensor network data. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 369–373, Washington, DC, USA, 2005. IEEE Computer Society.

[CFM$^+$04] Alfredo Cuzzocrea, Filippo Furfaro, Elio Masciari, Domenico Saccà, and Cristina Sirangelo. Approximate query answering on sensor network data streams. 2004.

[CFMS04] Alfredo Cuzzocrea, Filippo Furfaro, Giuseppe M. Mazzeo, and Domenico Sacca. A grid framework for approximate aggregate query answering on summarized sensor network readings. In *In Proceedings of the 1st International Workshop on Grid Computing and its Application to Data Analysis, LNCS*, pages 144–153, 2004.

[CJ09] Sharma Chakravarthy and Qingchun Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing.* Springer Publishing Company, Incorporated, 2009.

[CJSS03] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651, New York, NY, USA, 2003. ACM.

[CKR08] Alfredo Cuzzocrea, Abhas Kumar, and Vincenzo Russo. Experimenting the query performance of a grid-based sensor network data warehouse. In *Globe*, pages 105–119, 2008.

[CKV$^+$02] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, 2002.

[CO82] F. Y. Chin and G. Ozsoyoglu. Auditing and inference control in statistical databases. *IEEE Trans. Softw. Eng.*, 8(6):574–582, 1982.

[Col96] George Colliat. Olap, relational, and multidimensional database systems. *SIGMOD Rec.*, 25(3):64–69, 1996.

[CR02] Junghoo Cho and Sridhar Rajagopalan. A fast regular expression indexing engine. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 419, Washington, DC, USA, 2002. IEEE Computer Society.

[CR08] Alfredo Cuzzocrea and Vincenzo Russo. Compressing data cubes in the presence of simultaneous multiple olap queries. In *SEBD*, pages 422–429, 2008.

[CRS08] Alfredo Cuzzocrea, Vincenzo Russo, and Domenico Saccà. A robust sampling-based framework for privacy preserving olap. In *DaWaK*, pages 97–114, 2008.

[CRSS07] Alfredo Cuzzocrea, Vincenzo Russo, Domenico Saccà, and Paolo Serafino. Advanced olap visualization of multidimensional data cubes: A semantics-driven compression approach. In *SEBD*, pages 365–372, 2007.

[CSA05] N. Chaudhry, K. Shaw, and M. Abdelguerfi. *Stream Data Management (Advances in Database Systems)*. Springer, 1 edition, April 2005.

[CT98] Luca Cabibbo and Riccardo Torlone. From a procedural to a visual query language for olap. *Scientific and Statistical Database Management, International Conference on*, 0:74, 1998.

[Cuz05] Alfredo Cuzzocrea. Overcoming limitations of approximate query answering in olap. In *IDEAS '05: Proceedings of the 9th International Database Engineering & Application Symposium*, pages 200–209, Washington, DC, USA, 2005. IEEE Computer Society.

[Cuz06a] Alfredo Cuzzocrea. Accuracy control in compressed multidimensional data cubes for quality of answer-based olap tools. In *SSDBM '06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 301–310, Washington, DC, USA, 2006. IEEE Computer Society.

[Cuz06b] Alfredo Cuzzocrea. Improving range-sum query evaluation on data cubes via polynomial approximation. *Data Knowl. Eng.*, 56(2):85–121, 2006.

[CW07] Alfredo Cuzzocrea and Wei Wang. Approximate range-sum query answering on data cubes with probabilistic guarantees. *J. Intell. Inf. Syst.*, 28(2):161–197, 2007.

[DF02] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases, From Theory to Practice*, volume 2316 of *Lecture Notes in Computer Science*. Springer, 2002.

[DFK$^+$01] G. Duncan, S. Fienberg, R. Krishnan, R. Padman, and S. Roehrig. Disclosure limitation methods and information loss for tabular data. In *Confidentiality, Disclosure, and Data Access: Theory and Practical Applications for Statistical Agencies*, 2001.

[DGGR02] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*

'02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pages 61–72, New York, NY, USA, 2002. ACM.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

[DIG07] Yanlei Diao, Neil Immerman, and Daniel Gyllstrom. Sase+: An agile language for kleene closure over event streams, 2007.

[DJL79] David Dobkin, Anita K. Jones, and Richard J. Lipton. Secure databases: protection against user influence. *ACM Trans. Database Syst.*, 4(1):97–106, 1979.

[DS83] D. E. Denning and J. Schlorer. Inference controls for statistical databases. *Computer*, 16(7):69–82, 1983.

[FK06] Jianchun Fan and Subbarao Kambhampati. Multi-objective query processing for data aggregation, 2006.

[FSGM$^+$98] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 299–310, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[GCB$^+$97] 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. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[GJJ97] Michael Gebhardt, Matthias Jarke, and Stephan Jacobs. A toolkit for negotiation support interfaces to multi-dimensional data. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 348–356, New York, NY, USA, 1997. ACM.

[GKS01] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 2001. ACM.

[GKTD00a] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. *SIGMOD Rec.*, 29(2):463–474, 2000.

[GKTD00b] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. *SIGMOD Rec.*, 29(2):463–474, 2000.

[GM98] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 331–342, New York, NY, USA, 1998. ACM.

[GNPP07] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339, New York, NY, USA, 2007. ACM.

[GÖ03a] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[GÖ03b] Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *VLDB*, pages 500–511, 2003.

[GW76] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Trans. Database Syst.*, 1(3):242–255, 1976.

[HAMS97] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in olap data cubes. *SIGMOD Rec.*, 26(2):73–88, 1997.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HFAE03] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *VLDB*, pages 297–308, 2003.

[HH05] Lilian Harada and Yuuji Hotta. Order checking in a cpoe using event analyzer. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 549–555, New York, NY, USA, 2005. ACM.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2000. ACM.

[HS98] Mohand-Said Hacid and Ulrike Sattler. Modeling multidimensional databases: A formal object-centered approach. In *In: Proc. of the 6th European Conference on Information Systems (ECIS*, 1998.

[HZW$^+$05] Ming Hua, Shouzhi Zhang, Wei Wang, Haofeng Zhou, and Baile Shi. Fmc: An approach for privacy preserving olap. In *DaWaK*, volume 3589, 2005.

[Jol02] I. T. Jolliffe. *Principal Component Analysis.* Springer, second edition, October 2002.

[JSA05] Ruoming Jin, Kaushik Sinha, and Gagan Agrawal. Simultaneous optimization of complex mining tasks with a knowledgeable cache. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 600–605, New York, NY, USA, 2005. ACM.

[JSSS01] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[KJP77] Donald E. Knuth, Jr, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[KMS00] Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Optimal histograms for hierarchical range queries. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 196–204. ACM, 2000.

[KP03] Panos Kalnis and Dimitris Papadias. Multi-query optimization for on-line analytical processing. *Inf. Syst.*, 28(5):457–473, 2003.

[KS03] Nick Koudas and Divesh Srivastava. Data stream query processing: a tutorial. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 1149–1149. VLDB Endowment, 2003.

[LCP$^+$07] Yunhao Liu, Lei Chen, Jian Pei, Qiuxia Chen, and Yiyang Zhao. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 37–46, Washington, DC, USA, 2007. IEEE Computer Society.

[LHLG08] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proc. VLDB Endow.*, 1(1):1081–1094, 2008.

[LHW07] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604, New York, NY, USA, 2007. ACM.

[LJLR08] Dong Gyu Lee, Young Jin Jung, Young Wook Lee, and Keun Ho Ryu. Hashed multiple lists: A stream filter for processing continuous query with multiple attributes in geosensor networks. In *CITWORKSHOPS '08: Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, pages 104–109, Washington, DC, USA, 2008. IEEE Computer Society.

[LM92] T. Y. Cliff Leung and Richard R. Muntz. Temporal query processing and optimization in multiprocessor database machines. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 383–394, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[LMT$^+$05] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 34(1):39–44, 2005.

[LTWZ05] Chang Luo, Hetal Thakkar, Haixun Wang, and Carlo Zaniolo. A native extension of sql for mining data streams. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 873–875, New York, NY, USA, 2005. ACM.

[Luc08] David Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In *RuleML*, page 3, 2008.

[LWL07] Shaorong Liu, Fusheng Wang, and Peiya Liu. Integrated data modeling for querying physical objects in rfid-enabled pervasive computing. In *MDM '07: Proceedings of the 2007 International Conference on Mobile Data Management*, pages 140–145, Washington, DC, USA, 2007. IEEE Computer Society.

[LWZ04] Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Query languages and data models for database sequences and data streams. In *VLDB*, pages 492–503, 2004.

[McC76] Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[MM90] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[MMM06] Francesco M. Malvestuto, Mauro Mezzini, and Marina Moscarini. Auditing sum-queries to make a statistical database secure. *ACM Trans. Inf. Syst. Secur.*, 9(1):31–60, 2006.

[MRSR01] Hoshi Mistry, Prasan Roy, S. Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. *SIGMOD Rec.*, 30(2):307–318, 2001.

[MTZ08] Barzan Mozafari, Hetal Thakkar, and Carlo Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 179–188, Washington, DC, USA, 2008. IEEE Computer Society.

[MVSV03] Andreas S. Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, and Yannis Vassiliou. Cpm: A cube presentation model for olap. In *DaWaK*, pages 4–13, 2003.

[MW94] Udi Manber and Sun Wu. Glimpse: a tool to search through entire file systems. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, pages 4–4, Berkeley, CA, USA, 1994. USENIX Association.

[NK01] Zaiqing Nie and Subbarao Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 223–230, New York, NY, USA, 2001. ACM.

[Pap84] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* Mc-Graw Hill, 1984.

[PHB07] Jaekwan Park, Bonghee Hong, and Chaehoon Ban. A continuous query index for processing queries on rfid data stream. In *RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 138–145, Washington, DC, USA, 2007. IEEE Computer Society.

[PI97] Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 486–495, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[PP99] Chang-Shing Perng and Douglas Stott Parker, Jr. Sql/lpp: A time series extension of sql based on limited patience patterns. In *DEXA '99: Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pages 218–227, London, UK, 1999. Springer-Verlag.

[PP00] Torsten Priebe and Günther Pernul. Towards olap security design — survey and research issues. In *DOLAP '00: Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP*, pages 33–40, New York, NY, USA, 2000. ACM.

[RDR$^+$98] Raghu Ramakrishnan, Donko Donjerkovic, Arvind Ranganathan, Kevin S. Beyer, and Muralidhar Krishnaprasad. Srql: Sorted relational query language. In *SSDBM '98: Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 84–95, Washington, DC, USA, 1998. IEEE Computer Society.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch81] Jan Schlörer. Security of statistical databases: multidimensional transformation. *ACM Trans. Database Syst.*, 6(1):95–112, 1981.

[Sel88] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[Ses98] Praveen Seshadri. Predator: a resource for database research. *SIGMOD Rec.*, 27(1):16–20, 1998.

[SG90] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Trans. on Knowl. and Data Eng.*, 2(2):262–266, 1990.

[Sho97] Arie Shoshani. Olap and statistical databases: similarities and differences. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 185–196, New York, NY, USA, 1997. ACM.

[SLR94] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. Sequence query processing. *SIGMOD Rec.*, 23(2):430–441, 1994.

[SLXN06] Y. Sung, Yao Liu, Hui Xiong, and A. Ng. Privacy preservation for data cubes. *Knowl. Inf. Syst.*, 9(1):38–61, 2006.

[SO87] Alan Stuart and J. Keith Ord. *Kendall's advanced theory of statistics.* Griffin, London, 5. ed. edition, 1987.

[SP07] Alok Sharma and Kuldip K. Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recogn. Lett.*, 28(10):1151–1155, 2007.

[SS95] Praveen Seshadri and Arun N. Swami. Generalized partial indexes. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 420–427, Washington, DC, USA, 1995. IEEE Computer Society.

[Swe02] Latanya Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[SZZA01a] Reza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Optimization of sequence queries in database systems. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 71–81, New York, NY, USA, 2001. ACM.

[SZZA01b] Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, and Jafar Adibi. A sequential pattern query language for supporting instant data mining for e-services. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 653–656, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[SZZA04] Reza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Expressing and optimizing sequence queries in database systems. *ACM Trans. Database Syst.*, 29(2):282–318, 2004.

[TGNO92] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. In *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 321–330, New York, NY, USA, 1992. ACM.

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[Vas98] Panos Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *SSDBM '98: Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 53–62, Washington, DC, USA, 1998. IEEE Computer Society.

[VWI98] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data cube approximation and histograms via wavelets. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 96–104, New York, NY, USA, 1998. ACM.

[WBTG06] Ying Wang, Shuo Bai, Jianlong Tan, and Li Guo. Efficient filtering query indexing in data stream. In *WISE Workshops*, pages 1–12, 2006.

[WJW04] Lingyu Wang, Sushil Jajodia, and Duminda Wijesekera. Securing olap data cubes against privacy breaches. *Security and Privacy, IEEE Symposium on*, 0:161, 2004.

[WRGB06a] Song Wang, Elke Rundensteiner, Samrat Ganguly, and Sudeept Bhatnagar. State-slice: new paradigm of multi-query optimization of window-based stream queries. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 619–630. VLDB Endowment, 2006.

[WRGB06b] Song Wang, Elke A. Rundensteiner, Samrat Ganguly, and Sudeept Bhatnagar. State-slice: New paradigm of multi-query optimization of window-based stream queries. In *VLDB*, pages 619–630, 2006.

[WWJ04] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. Cardinality-based inference control in data cubes. *J. Comput. Secur.*, 12(5):655–692, 2004.

[WZ03] Haixun Wang and Carlo Zaniolo. Atlas: A native extension of sql for data mining. In *SDM*, 2003.

[XHCL06] Dong Xin, Jiawei Han, Hong Cheng, and Xiaolei Li. Answering top-k queries with multi-dimensional selections: the ranking cube approach. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 463–474. VLDB Endowment, 2006.

[yfi02] Yfilter: Efficient and scalable filtering of xml documents. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 341, Washington, DC, USA, 2002. IEEE Computer Society.

[ZWCC07] Fred Zemke, Andrew Witwoski, Mitch Cherniak, and Latha Colby. Pattern matching in sequences of rows. 2007.

[ZZC04] Nan Zhang, Wei Zhao, and Jianer Chen. Cardinality-based inference control in olap systems: an information theoretic approach. In *DOLAP '04: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 59–64, New York, NY, USA, 2004. ACM.