Cube-Lifecycle Management and Applications

Konstantinos Morfonios*

National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, University Campus, 15784 Athens, Greece
`kmorfo@di.uoa.gr`

Abstract. A common operation involved with the majority of algorithms relevant to On-Line Analytical Processing is aggregation, which can be extremely time-consuming if applied over large datasets. To overcome this drawback, scientists have proposed the precomputation and materialization of a large volume of aggregated data into a structure called data cube. Nevertheless, the construction and usage of the data cube itself has been found very demanding in terms of computational and storage resources. In the thesis summarized here [1] (hereafter called the thesis), we study this problem in depth and propose comprehensive suites of scalable algorithms that perform efficient cube construction, storage, query answering, incremental updating, indexing, and caching. Our extensive experimental evaluation indicates that our solutions are viable even when applied over very large datasets with arbitrary hierarchies. Some key points in our work include the introduction of a novel storage scheme for cubes that is based on the use of row-id references, a new external partitioning algorithm, an efficient construction plan, and the use of on-demand approaches for processes that are too expensive to perform in a given window of time. A unique property of all our algorithms is their compatibility with the relational model, which makes them easy to incorporate into existing relational servers. Finally, as an application in data mining, we study the usage of aggregate queries for solving the problems of feature selection and classification and propose a disk-based, lazy, and accurate solution, which exhibits great potentials in a broad range of applications.

1 Introduction

Modern data analysis "mines" knowledge from data stored in database systems discovering trends useful for decision making. To achieve such knowledge discovery, analysts pose complex queries that extensively use aggregation in order to group together "similarly behaving tuples". The response time of such queries over extremely large fact tables in modern data warehouses can be prohibitive. This inspired Gray et al. [2] to propose the implementation of the data cube. Implementation of the data cube is one of the most important, albeit computationally expensive, processes in On-Line Analytical Processing (OLAP). It

* Dissertation Advisor:Yannis Ioannidis, Professor

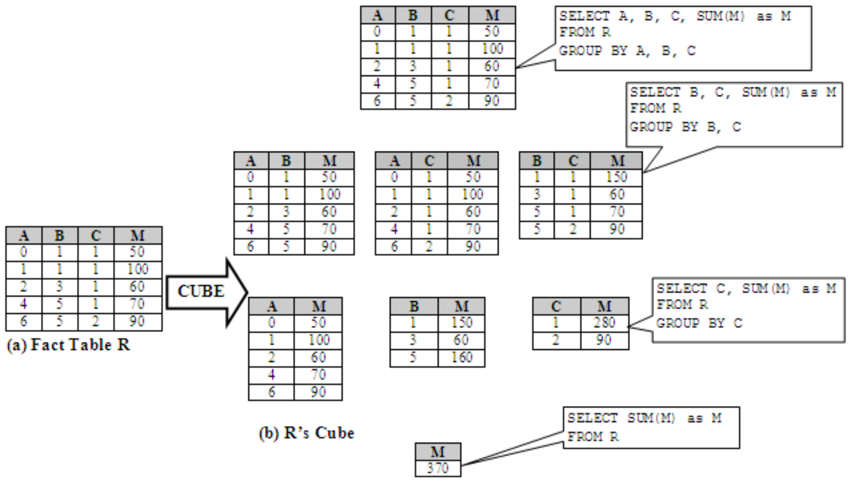
involves the computation and storage of the results of aggregate queries grouping on all possible dimension-attribute combinations over a fact table in a data warehouse. Such precomputation and materialization of (parts of) the cube is critical for improving the response time of OLAP queries and of operators such as roll-up, drill-down, slice-and-dice, and pivot, which use aggregation extensively [2]. Materializing the entire cube is ideal for fast access to aggregated data but may pose considerable costs in computation and maintenance time, as well as in storage space.

In order to overcome this problem and balance the tradeoff between query-response times and cube-resource requirements, implementation of the complete data cube has been studied using various data structures to construct and store the cube. In general, the data-cube implementation algorithms that have been proposed in the literature can be partitioned into four main categories, depending on the format they use in order to compute and store a data cube. On the one hand, Relational-OLAP (ROLAP) [2–5] and Multidimensional-OLAP (MOLAP) [6] methods use materialized views and multidimensional arrays, respectively, focusing mainly on the efficient sharing of computational costs (like sorting or hashing) during cube construction. On the other hand, Graph-Based approaches [7, 8] exploit specialized graphs (that usually take the form of tree-like data structures) in order to compute and store cubes more efficiently. Finally, Approximation-Based methods [9, 10] exploit various in-memory representations (like histograms), borrowed mainly from statistics. In this thesis, we focus on ROLAP methods and do not further study methods that belong to the other categories for the following reasons:

- MOLAP methods are poor performers when data is sparse, which is the case in most real-life applications. Although challenged by some, this has been observed by many researchers [3, 4].
- Graph-Based methods appear to have superior performance for cube construction and storage, but are currently not supported by any widely used product. Hence, they require nontrivial implementation effort, mainly due to the use of specialized data structures and algorithms.
- Approximation-Based methods generate and store approximate results, which are much more difficult to manage at run time compared to precise results generated by ROLAP methods.

As we show in this thesis, existing ROLAP methods that implement data cubes focus mainly on construction and storage of flat cubes (i.e., cubes constructed over flat datasets). The lifecycle of a data cube, however, does not involve (off-line) construction and storage only, but also query answering and incremental maintenance. Moreover, real-world datasets are not always “flat” but are usually organized in hierarchies, whose nature introduces several complications into all phases of the cube lifecycle, making existing techniques essentially inapplicable in a significant number of real-world applications. To overcome these problems, in this thesis, we develop comprehensive ROLAP solutions that address efficiently all functionality in the lifecycle of a cube and can be implemented easily over existing relational servers. They are families of algorithms developed around novel, purely-ROLAP construction methods that provide fast computation of a fully-materialized cube in compressed form, are incrementally updateable, and exhibit fast query-response times that can be improved by low-cost indexing and caching, even when dealing with very large datasets with arbitrary hierarchies. The efficiency of our methods is demonstrated through comprehensive experiments on both synthetic and real-world datasets, whose results have shown great promise for the performance and scalability potential of the proposed techniques, with respect to both the size and dimensionality of the fact table.

The rest of this paper is organized as follows: In Section 2, we provide a detailed description of the data-cube implementation problem, focusing mainly on ROLAP techniques, and define some basic terminology that we use throughout this paper. Then, in Section 3, we summarize the main contributions of the thesis. Finally, we conclude in Section 4 and describe the directions of our future work.



A	B	C	M
0	1	1	50
1	1	1	100
2	3	1	60
4	5	1	70
6	5	2	90

A	B	C	M
0	1	1	50
1	1	1	100
2	3	1	60
4	5	1	70
6	5	2	90

```
SELECT A, B, C, SUM(M) as M
FROM R
GROUP BY A, B, C
```

A	B	M
0	1	50
1	1	100
2	3	60
4	5	70
6	5	90

A	C	M
0	1	50
1	1	100
2	1	60
4	1	70
6	2	90

B	C	M
1	1	150
3	1	60
5	1	70
5	2	90

```
SELECT B, C, SUM(M) as M
FROM R
GROUP BY B, C
```

A	M
0	50
1	100
2	60
4	70
6	90

B	M
1	150
3	60
5	160

C	M
1	280
2	90

```
SELECT C, SUM(M) as M
FROM R
GROUP BY C
```

M
370

```
SELECT SUM(M) as M
FROM R
```

Fig. 1. Fact table R_1 and its data cube

2 Problem Description

Consider a fact table R_1 (Figure 1a) consisting of three dimensions (A, B, C) and one measure M. Figure 1b illustrates the corresponding cube. Each view that belongs to the cube (also called cube node hereafter) materializes a specific group-by query as shown in Figure 1b. Clearly, assuming that the data in the fact table is flat, i.e., that it is not organized in hierarchies, if D is the number of dimensions of a fact table, the number of all cube nodes is 2^D. (The situation becomes even more challenging in the presence of hierarchies.) The factor 2^D

implies that the cube size is exponentially larger with respect to D than the size of the original data (in the worst case). In typical applications, this is in the order of gigabytes, so development of efficient data-cube implementation algorithms is extremely critical.

A common representation of the data cube that captures the computational dependencies among different group-by queries is the cube lattice [11]. Let "grouping attributes" be the fields of a table that participate in the group-by part of a query. All group-bys computed for the data-cube construction can be partially ordered in a lattice structure [11], which is a directed acyclic graph (DAG), where each node represents a group-by query on the fact table and is connected via a directed edge with every other node that contains exactly one fewer grouping attribute. The source of an edge is called its parent node and the destination of the edge is called its child node. The node whose grouping attributes consist of all dimensions (highest) is called the root of the lattice. The node whose grouping-attribute set is empty (lowest) is called the ALL node. For example, Figure 2 presents the lattice that corresponds to the fact table R_1 (Figure 1a). Nodes in higher lattice levels contain more grouping attributes and are thus more detailed (hold data of finer granularity) than nodes in lower levels, which are more specialized (hold data of coarser granularity). Exploiting the "1-1" mapping between group-by queries and lattice nodes, we call node size the size of the result of the corresponding query.

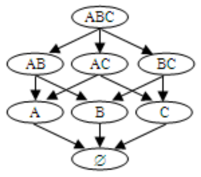


Fig. 2. Example of a cube lattice

In most cases, computation of the data cube is an extremely time-consuming process due to the size of the original fact table and the exponential number of cube nodes with respect to the number of dimensions. Hence, it is common in practice to select a proper subset of the data cube, pre-compute it off-line, and store it for further use. We call this process "data-cube implementation". Over the last years, there has been intense interest in efficient data-cube implementation and a variety of methods has been proposed for this task.

For ROLAP approaches, data-cube implementation consists of data-cube computation and data-cube selection. These, however, are not necessarily two separate procedures applied sequentially. A-priori knowledge of the strategy used for selection may affect the computation procedure and vice-versa. Hence, several methods have been proposed that combine fast computation and efficient selection in one step; we call them integrated methods.

3 Thesis Contribution

Taking into account that existing ROLAP methods exhibit weaknesses in one or more issues related to the lifecycle of data cubes, in this thesis, we propose novel techniques that deal with the problem in a comprehensive fashion, providing efficient solutions for the entire cube lifecycle, including construction and storage, indexing, caching, query answering, and incremental maintenance. We study techniques that are applicable not only on flat data cubes but on datasets that include hierarchies as well. The results of a rather extensive experimental evaluation on both real-world and synthetic datasets are very positive, giving strong indications on the power of our methods and of ROLAP overall.

In more detail, we start with a study of related work and provide a qualitative examination of ROLAP methods. We go beyond a simple review of existing algorithms and contribute to the following issues [12]:

- **Comprehensive Review:** We offer a comprehensive review of existing cubing methods that belong to the ROLAP framework, highlighting interesting concepts from various publications and studying their advantages and disadvantages. To the best of our knowledge, this is the first overview of its kind.
- **Problem Parameterization:** We carefully analyze existing ROLAP cubing methods and identify six orthogonal parameters/dimensions that affect their performance, namely "Traversal of Cube Lattice", "Partitioning of Original Fact Table", "In-Memory Processing Algorithm", "Storage Granularity", "Selection Criterion", and "Lattice Reference". The resulting 6-dimensional parameter space essentially models the data-cube implementation problem.
- **Method Classification:** We place existing techniques at the appropriate points within this parameter space and identify several clusters that these form. These clusters have various interesting properties, whose study leads to the identification of particularly effective values for the space parameters and indicates the potential for devising new, better-performing algorithms overall.

Based on the findings of the aforementioned thread of our work, we investigate new methods and propose a suite of algorithms that exhibit efficiency in all phases of the lifecycle of flat cubes. In particular, we focus on the following issues [13]:

- **Cube Construction:** We incorporate redundancy reduction into the dominant ROLAP cube construction methods and devise three new cubing algorithms, which exhibit considerable reduction in the size of the cube to be stored as well as some minor reduction in its construction time. We study their behavior under large and multi-dimensional datasets and show that the best among them, namely TRS-BUC, is the first ROLAP method that scales well up to at least 25 dimensions.
- **Query Answering:** Under a comprehensive query model, which is broader than the models used in the past for the same purpose, we evaluate novel

algorithms for answering queries on top of the cubes produced by the new methods and demonstrate that the cube resulting from TRS-BUC exhibits significantly better query execution performance compared to all earlier techniques, including those considered as “champions” with respect to construction time and storage space.

- **Incremental Maintenance:** We introduce novel incremental update algorithms and demonstrate that those based on TRS-BUC exhibit again significantly better performance compared to their counterparts. Moreover, they produce a cube identical to the one that would be produced by full reconstruction, i.e., TRS-BUC preserves its compact format unaffected, regardless of the frequency of updates.
- **Indexing and Caching:** We propose indexing and caching schemes and study their effect on both query answering and incremental updating of ROLAP cubes. Our experiments show that TRS-BUC is the only known method that can benefit significantly from such techniques, consuming inexpensive additional resources.

Moving on to hierarchical datasets, we show that the nature of hierarchies introduces several complications in all phases of the cube lifecycle that render existing ROLAP techniques (including those built around TRS-BUC) impractical. To overcome this drawback, we propose an extension of TRS-BUC, called CURE [14], and revisit all its surrounding algorithms related to cube usage. CURE contributes a novel lattice traversal scheme, an optimized data partitioning method, and a suite of relational storage schemes for all forms of redundancy. The last two are useful to “flat” datasets as well, but they are mostly necessary in the presence of hierarchies. In more detail:

- **Lattice Traversal with Dimension Hierarchies:** To the best of our knowledge, CURE is essentially the first comprehensive ROLAP solution capable of constructing a complete cube not only at the leaf level of each dimension hierarchy, but also at all higher levels, precomputing group-by queries at all granularities. To achieve this, CURE uses an efficient way of traversing an extended lattice that includes dimension hierarchy levels (first proposed elsewhere [11]), which enables great cost sharing of sorting operations through pipelining.
- **External Partitioning:** We propose an efficient algorithm for partitioning fact tables that store hierarchical data of any size into memory-fitting segments, while computing a very small subset of the cube using inexpensive additional resources. Exploiting this early-computed data, CURE accelerates the construction of the final cube significantly, making it feasible even when the original fact table is extremely large. Existing techniques partition data according to values in a single dimension and require that segments of tuples with the same value in this dimension fit in memory. As shown in this thesis, however, this is not always possible in cases that include hierarchies, due to small domain sizes at coarse granularities.
- **Efficient Storage:** Unlike previous ROLAP methods that rely only on avoiding redundant-tuple storage for cube size reduction, we further study

alternative schemes for storing nonredundant data efficiently as well. To the best of our knowledge, CURE is the only ROLAP method that condenses the cube both by rejecting all kinds of redundancy and by further exploiting appropriate data representations.

– **Query Answering:** We develop a straightforward algorithm for answering arbitrary queries using data materialized in an (unindexed) CURE cube and show that its practicability is limited in real-world applications that typically involve selective queries over large datasets. To overcome this, we investigate the effect of indexing on CURE cubes and propose an efficient extension of the original algorithm that is based on low-cost indexes. We show that indexing the entire cube, which is potentially very expensive in hierarchical cubes, is not necessary; indexing only the original fact table is enough, primarily because of the particular storage format of CURE cubes.
– **Query Optimization:** We examine customized query optimization policies to identify when using an index is beneficial and to indicate which index combination to use for a given query, based on cost estimations.
– **Incremental Maintenance:** We study different approaches for the incremental maintenance of CURE cubes and conclude that common eager tactics that refresh a cube periodically during a dedicated window of time are prohibitively expensive, due to the storage format of CURE and the nature of hierarchies. Alternatively, we propose a novel lazy method that only performs some lightweight operations during the update window and updates data on-line, when necessary, during query processing. Interestingly, the additional query cost is marginal making the lazy approach the method of choice. To the best of our knowledge, this is the first time a lazy method has been applied to a cube-related method. Finally, we propose a hybrid combination of the eager and the lazy method, which is very promising under certain conditions.

Having found efficient algorithms for materializing and using a large number of aggregate views, we study an application of aggregation on data-mining techniques, including classification and feature selection. In this thread of our work, we propose LOCUS, a lazy classifier implemented exclusively with aggregate queries expressed in standard SQL. To the best of our knowledge, LOCUS is the first disk-based lazy classifier with all the following properties [15]:

– **Classification Accuracy:** It exhibits good classification accuracy, which improves as training sets become larger. This can be justified theoretically based on its convergence to the optimal Bayes classifier, which minimizes the classification error probability. The same is also verified experimentally in comparison to Decision Trees, a very popular and accurate existing classifier.
– **Disk-Based Implementation:** It is database-friendly and, to the best of our knowledge, the only lazy method that uses a small and constant number of highly selective range queries in order to classify unknown objects. Such queries actually need to access a very small part of the underlying database and have been well studied in the database literature. They can be expressed

in standard SQL and existing query optimizers guarantee their fast response times with the use of traditional indexes, e.g., B$^+$-Trees.
- **Feature Selection:** Its classification accuracy can be efficiently used as a promising criterion for feature selection as well, in the sense that features are selected based on the accuracy that the classifier achieves when applied on them.
- **Parallelization:** It can be efficiently parallelized with essentially unlimited scalability.

4 Conclusions and Future Work

In this thesis, we presented a comprehensive study of efficient (mostly ROLAP) algorithms related to data cubes. We started with a thorough review of existing algorithms for efficient data-cube implementation in a ROLAP environment and identified six orthogonal parameters: "Traversal of Cube Lattice", "Partitioning of Original Fact Table", "In-Memory Processing Algorithm", "Storage Granularity", "Selection Criterion", and "Lattice Reference". We placed the existing algorithms at the appropriate points within the problem space based on their properties. We observed that the algorithms form clusters, whose study led to the identification of particularly effective values for the space parameters.

Based on the findings of the first phase of our research, we incorporated redundancy reduction into the best existing pure ROLAP methods for cube implementation and proposed TRS-BUC and a suite of novel algorithms built around it that deal with all aspects of cube usage, including efficient construction, storage, query answering, incremental updating, indexing, and caching. To the best of our knowledge, this has been essentially the first such comprehensive approach to the problem in the ROLAP context, treating all the above aspects in an independent fashion.

Furthermore, we studied ROLAP cubing in the presence of hierarchies and presented CURE, a novel ROLAP cubing method that addresses all challenges imposed by the nature of hierarchies and constructs complete data cubes over very large datasets with arbitrary hierarchies. CURE introduced an efficient execution plan suitable for hierarchical cube construction and revisited external-partitioning and size-reduction methods, which are complicated due to the existence of hierarchies. The effectiveness of CURE has been demonstrated through experiments on both real-world and synthetic datasets (including the APB-1 benchmark in its highest density), which have given very promising results with respect to the potential of CURE overall. Moreover, we developed efficient algorithms for query processing and incremental updating over CURE cubes in the presence of hierarchies, including some lazy policies that were never applied on cubes before. Interestingly, our solutions are ROLAP compatible, matching the design goals of CURE.

Finally, we applied ideas borrowed from data cubing on data mining and proposed LOCUS, an accurate and efficient disk-based lazy classifier that is data-scalable and can be implemented using aggregate queries expressed in standard

SQL. We showed that in most cases LOCUS exhibits high classification accuracy, which improves as training sets become larger, based on its convergence to the optimal Bayes. Furthermore, we used its classification accuracy as an efficient and reliable criterion for feature selection and proposed parallelizing both the classification and feature-selection processes, essentially achieving unlimited scalability. Overall, the results are very promising with respect to the potential of LOCUS as the basis for feature selection and classification, especially over large or inherently complex datasets.

In the future, we plan to compare CURE directly with Dwarf [8] and QC-Tree [7], prominent cubing methods that use specialized graph-like data structures. We expect this comparison to reveal the fundamental strengths and weaknesses of the two underlying techniques. Furthermore, we are interested in applying a CURE-like algorithm for the construction of skyline cubes.

Finally, with respect to data mining, we plan to further explore our feature selection method in the direction of identifying multiple reliable nodes, to implement a parallel version of LOCUS, and to study its applicability in regression problems (possibly replacing the count aggregate function with average).

Although cubing is already more than 10 years old, it remains an exciting problem with several fundamental aspects as well as applications still remaining in the dark and waiting to be investigated.

References

1. Morfonios, K.: Cube-lifecycle management and applications. (Ph. D. Thesis. 2007)
2. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: Proc. of International Conference on Data Engineering (ICDE). (1996) 152–159
3. Beyer, K.S., Ramakrishnan, R.: Bottom-up computation of sparse and iceberg cubes. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (1999) 359–370
4. Ross, K.A., Srivastava, D.: Fast computation of sparse datacubes. In: Proc. of Very Large Data Bases (VLDB). (1997) 116–125
5. Wang, W., Lu, H., Feng, J., Yu, J.X.: Condensed cube: An efficient approach to reducing data cube size. In: Proc. of International Conference on Data Engineering (ICDE). (2002) 155–165
6. Zhao, Y., Deshpande, P., Naughton, J.F.: An array-based algorithm for simultaneous multidimensional aggregates. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (1997) 159–170
7. Lakshmanan, L.V.S., Pei, J., Zhao, Y.: Qc-trees: An efficient summary structure for semantic olap. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (2003) 64–75
8. Sismanis, Y., Deligiannakis, A., Roussopoulos, N., Kotidis, Y.: Dwarf: shrinking the petacube. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (2002) 464–475
9. Gunopulos, D., Kollios, G., Tsotras, V.J., Domeniconi, C.: Approximating multidimensional aggregate range queries over real attributes. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (2000) 463–474

10. Vitter, J.S., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (1999) 193–204
11. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: Proc. of ACM Special Interest Group on Management of Data (SIGMOD). (1996) 205–216
12. Morfonios, K., Konakas, S., Ioannidis, Y., Kotsis, N.: Rolap implementations of the data cube. (submitted)
13. Morfonios, K., Ioannidis, Y.E.: Supporting the data cube lifecycle: The power of rolap. (to appear). In: VLDBJ. (2006)
14. Morfonios, K., Ioannidis, Y.E.: Cure for cubes: Cubing using a rolap engine. In: Proc. of Very Large Data Bases (VLDB). (2006) 379–390
15. Morfonios, K., Ioannidis, Y.E.: Locus: Lazy optimal classification of unlimited scalability. In: HDMS. (2006) 80–89