Real-Time Temporal Data Warehouse Cubing

Usman Ahmed, Anne Tchounikine, Maryvonne Miquel, Sylvie Servigne

Real-Time Temporal Data Warehouse Cubing

Usman Ahmed, Anne Tchounikine, Maryvonne Miquel, and Sylvie Servigne

Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
firstname.lastname@insa-lyon.fr

Abstract. Traditional data warehouses are built in an off-line periodic fashion which makes them less valuable in applications where the most up-to-date data is required. For these applications, data should be incorporated in the warehouse and made available as soon as possible in "Real Time Data Warehouse". In this paper we propose an indexing model named TiC-Tree, in order to simultaneously index and store multidimensional detailed and aggregated data. Our contribution exploits the temporal nature of data and focuses on range and/or group-by queries. We evaluate our proposal with the synthetic data set Star Schema Benchmark and advocate it in comparison with other existing solution.

Keywords: Data warehouse, OLAP, Real Time data, Graph based index.

1 Introduction and Motivation

The maintenance of data warehouse is usually carried out in an off-line fashion, after the facts are inserted via bulk incremental operations. This restriction makes them unsuitable for certain applications such as monitoring systems for natural risk management, traffic surveillance systems etc., that require the data to be always up to date. This type of applications raises the need of what can be called as Real Time Data Warehouses (RTDW), Active Data Warehouses [1] or Zero Latency Data Warehouses [2]. The main challenge is to integrate new data in the cube and make it available as soon as possible. Due to the type of application domain, query response time is of course a decisive key. Temporal OLAP analysis, i.e. drilling and slicing over time dimension, is of major interest. In this work: *First,* we propose a cubing model named TiC-Tree that provides fast update at arrival of each new fact and improves the response time of OLAP range and group-by queries over time dimension. The TiC-Tree indexes and stores detailed and aggregated data in the same structure and favors the grouping of data representing chronologically closed events even if the data delivery is delayed. *T*hen, we implement our proposition and evaluate its performance with the synthetic data set Star Schema Benchmark [3] with slight modification, i.e. the addition of hierachy level *hour* in *Time* dimension. We examine the different cases when the input data set is chronologically ordered and when some of the data arrives with a delay.

P. García Bringas et al. (Eds.): DEXA 2010, Part II, LNCS 6262, pp. 159–167, 2010.

2 Related Works

Cubing operation raises considerable challenges related to the complexity of calculation and storage of the data. An on-going and active research area includes numerous works that aim at defining strategies for selecting subsets of views to be materialized and efficient computation methods for the aggregates [4]. Graph based methods, such as Cubetrees [5] propose an alternative approach exploring hierarchical storage structures for the cube that may eventually be compressed (Dwarf [6], QC-Tree [7]). On the other hand, R-tree and variants [8,9] are widely used to index spatial and/or temporal data. These techniques are based on identification of Minimum Bounding Rectangle (MBR) for space partitioning. However, none of these methods support neither pre-defined dimension hierarchies nor pre-aggregation. Indices proposed in the special context of data warehouses include multidimensional array based methods, bit mapped indices, hierarchical and spatial indexing techniques based methods [10,5]. Among these, the DC-Tree[11] supports both pre-aggregation and pre-defined hierarchies and is based on X-Tree where MBR are replaced by MDS (Minimum Describing Sequence). Indeed, the use of MBR assumes that data is totally ordered in the referenced multidimensional space, while MDS uses the partial ordering of members induced by the dimension hierarchies.

3 Contribution

Time is a peculiar dimension of data warehouses. It is usually the only dimension whose instances grow continuously while all other dimensions are generally either static or slowly changing dimensions. Its members are naturally and totally ordered at each level of hierarchy, however, in context of real-time systems, this chronological order may be disregarded at the time of insertion because of network delays or system downtime. In this paper, we propose to take these features of time into consideration and propose a solution for cubing temporal data warehouses, thanks to a tree structure named TiC-Tree that uses a tailored definition of MDS with special handling for time dimension.

3.1 Multidimensional Model

Let $\mathcal{D}$ be a set of n dimensions $D_1, D_2, \ldots D_n$ of a multidimensional model. Each dimension D_i has a totally ordered set $\mathcal{L}_i$ of levels. A dimension hierarchy can be viewed as a directed acyclic graph of levels. We note $l_i^{k+1} \uparrow l_i^k$ the existing edge between 2 levels $l_i^{k+1}, l_i^k \in \mathcal{L}_i$ and $1 \leq k \leq |\mathcal{L}_i|$. An instance of a dimension D_i is defined by a set of values over $\bigcup domain(l_i^k)$, and a parent/child relation noted $\Uparrow$ between the members of levels such as $l_i^{k+1} \uparrow l_i^k$. A dimension instance can be viewed as a directed acyclic graph of members. If a path exists from n to m in the graph of members, then m is said to be an *ancestor* of n. Each dimension D_i contains a top most level ALL such that $domain(ALL) = \{all\}$.

Example 1. For time dimension D_t we define an ordered set of levels $\mathcal{L}_t$ as: $\mathcal{L}_t$ = {ALL, year, month, day, hour} *and* $|\mathcal{L}_t| = 5$ *and* All $\uparrow$ year $\uparrow$ month $\uparrow$ day $\uparrow$ hour *and* 2008 $\Uparrow$ March 2008 $\Uparrow$ 12 March 2008.

Definition 1. ***(Fact Table)*** *A fact table* $T_F(l_1^1, l_2^1, \ldots, l_n^1)$ *of a multidimensional model is a table with tuples* $< x_1, x_2, \ldots x_n, m_1, m_2, \ldots, m_p >$ *where* $x_i \in domain(l_i^1)$ *i.e. is a member of the lowest level of each dimension hierarchy and* m_j *is a numeric value called measure of the fact.*

Definition 2. ***(Aggregate Table)*** *An aggregate table* $T_A(l_1^{k_1}, l_2^{k_2}, \ldots l_n^{k_n})$ *of a multidimensional model is a table with tuples* $< x'_1, x'_2, \ldots x'_n, a_1, a_2, \ldots, a_p >$ *where* $x'_i \in domain(l_i^{k_i})$ *and* $\exists i | k_i > 1$ *and* a_j *is an aggregate measure value obtained by some aggregation function.*

Table 1. Extracted tuples of the Fact Table with 4 dimensions and one measure value

ID	Customer	Supplier	Part	Time	Quantity
t1	Customer234	Supplier329	Part3432	04:00 05/03/1999	15
t2	Customer103	Supplier1023	Part862	12:00 23/04/2000	60
t3	Customer20	Supplier19	Part1322	17:00 11/07/1999	20
t4	Customer20	Supplier1360	Part1322	02:00 18/11/1999	15
t5	Customer293	Supplier1870	Part94	10:00 03/06/2000	10
t6	Customer293	Supplier329	Part94	13:00 13/02/1999	30
t7	Customer923	Supplier1870	Part647	03:00 13/12/1999	45

Definition 3. ***(MDS)*** *A Minimum Describing Sequence* $M(l_1^{k_1}, l_2^{k_2}, \ldots, l_n^{k_n})$ is a sequence $[S_1, S_2, \ldots, S_n]$ *of n sets where* $S_i \subset domain(l_i^{k_i})$ *and n is the number of dimensions.*

Two MDS $M(l_1^{k_1}, l_2^{k_2}, \ldots, l_n^{k_n})$ and $N(l_1^{k'_1}, l_2^{k'_2}, \ldots, l_n^{k'_n})$ are considered to be at same level if $\forall i = 1, 2, \ldots n\ k_i = k'_i$, i.e. the corresponding sets have their members at the same levels of dimension hierarchies. An MDS can be regarded as a minimal representation of hyper-rectangle of members, in an n-dimensional space.

Example 2. M=[{Europe},{USA},{Part342},{1999}] and L=[{Europe},{USA, France}, {MFGR#5113}, {'Apr99','Jul99'}] are 2 MDS at different levels.

3.2 The TiC Tree

The TiC-Tree stores and indexes MDSs with associated aggregate or detailed values. Leaves of the tree are tuples of the fact table whereas internal nodes are aggregates computed on stored facts. Unlike the DC-Tree, the TiC-Tree processes the temporal data so as to help the grouping of closer values together. We define a set of metrics which are used to build, update and query the tree and allow optimization of MDS distribution in the nodes. These metrics take

the totally ordered nature of time into account and facilitate this grouping. This grouping strategy will facilitate time range and group-by queries. For all non-temporal dimensions, the calculation of these metrics is based on the cardinality of dimension sets in the MDS, while for the temporal one these are calculated on the basis of time duration covered by the MDS.

Let M and N be two MDS with sequence $[S_1, S_2, \ldots, S_n]$ and $[T_1, T_2, \ldots, T_n]$ respectively. For time dimension D_t, we note:

$$int(S_t) = [\min_{m_i \in S_t}(m_i), \max_{m_i \in S_t}(m_i)] \ .$$

Definition 4. ***(Contains)*** *M contains N is true if* $\forall i \neq t : T_i \subset S_i$ or $\forall n \in T_i, \exists m \in S_i \mid m is ancestor of n$ and $for i = t : int(T_t) \subset int(S_t) or \forall n \in T_t, \exists m \in S_t \mid m$ *is ancestor of* n.

MDS M contains another MDS N if: (1) for all non-temporal dimensions, all the sequence sets of N are included in those of M, or made of children of the members of the sequence sets of M, and (2) the time interval covered by N is contained in the interval covered by M.

Example 3. Let M=[{France},{USA},{Part342},{2001,2002}]; N=[{USA},{USA},{Part342},{2000,2002}]; O=[{Europe},{USA},{Part342},{2001,2005} then:

¬(M *contains* N) and (O *contains* M)

Definition 5. ***(Overlap)*** *The overlap of M and N denoted by overlap(M, N) is defined for 2 MDS at the same level of hierarchy, as:*

$$overlap(M, N) = \begin{cases} 0 & \text{if } int(S_t) \cap int(T_t) = \emptyset \\ \prod_{i=1, i \neq t}^{n} |S_i \cap T_i| & \text{else} \end{cases}$$

The overlap determines the volume of intersection between two MDSs. The overlap between two MDSs can be calculated, if and only if the MDSs are at same level.

Example 4. overlap(M,N) = 0*1*1=0 ; overlap(M,O) : Cannot be calculated.

Definition 6. ***(Extension)*** *The extension of M to accommodate N denoted by extension(M|N) is defined for 2 MDS at the same level of hierarchy, as:* $extension(M|N) = \sum_{i=1, i \neq t}^{n} |N_i - M_i| + df_l + df_u$ where:

$$df_l = \begin{cases} 0 \text{ if } \min_{m_i \in S_t}(m_i) - \min_{n_i \in T_t}(n_i) < 0 \\ \min_{m_i \in S_t}(m_i) - \min_{n_i \in T_t}(n_i) \quad \text{else} \end{cases} \text{ and } df_u = \begin{cases} 0 \text{ if } \max_{n_i \in T_t}(n_i) - \max_{m_i \in S_t}(m_i) < 0 \\ \max_{n_i \in T_t}(n_i) - \max_{m_i \in S_t}(m_i) \quad \text{else} \end{cases}$$

Extension of an MDS determines the enlargement needed in order to accommodate the newly coming MDS. Like *overlap*, *extension* also requires MDSs to be at same levels.

Example 5. extension(M|N)=1+0+0+1=2.

Tree Elements. TiC-Tree is composed of three different types of nodes, i.e. Data Nodes, Directory Nodes and Super Nodes. Let M be a MDS $M(l_1^{k_1}, l_2^{k_2}, \ldots, l_n^{k_n})$ of sequence $[S_1, S_2, \ldots, S_n]$.

Definition 7. ***(Data Node)*** *A data node N_{data} of the TiC-Tree is a tuple$<$ $M, a_1, a_2, \ldots a_p >$where $|S_i| = 1$ and $k_i = 1, \forall i = 1, 2 \ldots n$ and each a_j is a measure value of the node.*

Definition 8. ***(Directory Node)*** *A directory node N_{dir} of the TiC-Tree is a tuple$<$ $M, \mathcal{E}, a_1, a_2, \ldots a_p >$,$\mathcal{E}$ is a set of pointers to other nodes whose MDS are contained in the MDS M. Size of $\mathcal{E}$ is limited and constant and determines the capacity of the node. Each a_j represents an aggregate measure value of the node.*

Definition 9. ***(Super Node)*** *A super node N_{super} of the TiC-Tree is a tuple$<$ $M, \mathcal{F}, a_1, a_2, \ldots a_p >$where $\mathcal{F}$ is a set of pointers to other nodes whose MDSs are contained in the MDS M, and size of $\mathcal{F}$ is unlimited. Each a_j represents an aggregate measure value of the node.*

Figure 1 shows the TiC-Tree constructed with directory node capacity of 3. The leaves represent data nodes, while the rest are directory nodes. A data node encapsulates an MDS together with an associated measure value and represents a tuple of the fact table. A directory node, on the other hand, encapsulates an MDS and associated aggregate measure values of its children. This is also to be noted that the *root* of the TiC-Tree is always a directory node with MDS at level ALL for all the dimensions.

Tree Maintenance. In this section we illustrate the algorithms necessary for TiC tree maintenance. As the data warehouses do not generally have delete or update queries, we discuss only Insert and Search algorithms. Due to the lack of space, we donot give split and group-by query algorithms here. However we explain their working through the running example.

Insert. At the beginning, the tree has only one directory node called *root*. On arrival of a new fact, the data is packed into a data node and inserted into the *root*. The insert algorithm (see Algorithm 1) starts with updating the aggregate measure value of the node, and then searches for the node for subsequent insertion among its entries. This search is based on the metrics *Contain*, *Overlap* and *Extension* in order. This process continues recursively unless a directory or super node with entries of type "Data Node" is reached. Once the node for insertion is located, the data node is added to its entries (following the extension of the MDS of the directory/super node, if required). In case the directory node capacity is reached, an overflow occurs which induces the split of the directory node. This split may cause the parent node to overflow which will result in another split.

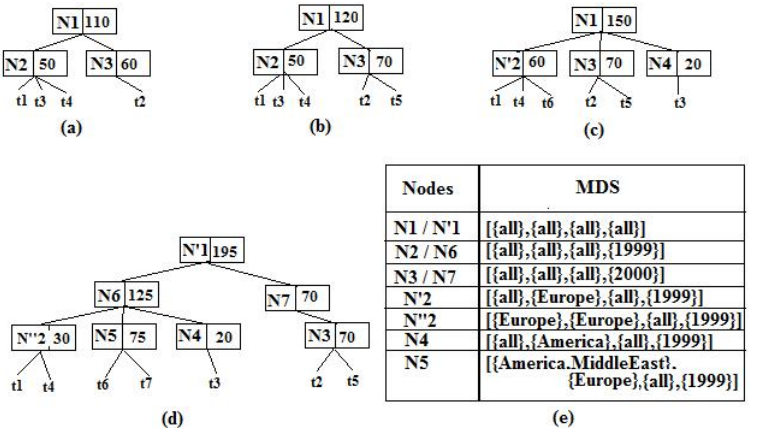
The split method of a directory node starts with selection of a split dimension and split level that are chosen on the basis of hierarchy level and cardinality of dimensions in the MDS. For example, in the MDS [{Europe}, {USA, France}, {MFGR#13}, {2000}], hierarchy level of Supplier and Part dimensions is 3 while

Algorithm 1. Insert Algorithm

```
TiCTDirectoryNode::Insert(TiCTDataNode *dataNode) {
/*Insert dtataNode into a Directory node "thisNode" */
update Aggregate Measure Value of thisNode;
followNode = ChooseSubTree(dataNode);
if (followNode != thisNode) then followNode.Insert(dataNode);
else {
  if (thisNode's Entries Type is "DataNode") then
    if (Number of thisNode's Entries < Maximum Entries Allowed) then add dataNode into the Entries
    else thisNode.Split(dataNode);
  else extend thisNode's MDS to accommodate dataNode and add it into the Entries}}
```

it's value is 4 for Customer and Time dimensions (see SSB [3]). First, the decision is taken on the basis of hierarchy level and one with the highest value is selected (in this example Customer and Time). In case of tie, the dimension with the higher cardinality is chosen as split dimension, which in our example is Supplier. After selecting the split dimension, hierarchy level of MDSs of all entries of the node and the node to split are adapted to splitLevel, two directory nodes are created and the entries are distributed in these newly created nodes on the basis of the metrics. If the overlap of the two MDSs exceeds the predefined limit, a new dimension is chosen for split. This process continues until a suitable split dimension is found; otherwise the overflowing node is adapted to a super node.

As a running example, the insertion of t5 (see Table 1) in figure 1(a), starts with the packing of t5 into a new data node and update of N1's aggregate measure. N3 is chosen for following insertion which requires zero extension as compared to 1 for N2. The data node is added to N3 and its aggregate measure



Nodes	MDS
N1 / N'1	[{all},{all},{all},{all}]
N2 / N6	[{all},{all},{all},{1999}]
N3 / N7	[{all},{all},{all},{2000}]
N'2	[{all},{Europe},{all},{1999}]
N''2	[{Europe},{Europe},{all},{1999}]
N4	[{all},{America},{all},{1999}]
N5	[{America,MiddleEast}. {Europe},{all},{1999}]

(e)

Fig. 1. TiC-Tree Updation: Insertion of tuples (a) t1, t2, t3 and t4 (b) t5 (c) t6 (d) t7 (e) MDSs corresponding the nodes

Algorithm 2. Range Query Algorithm

```
int TiCTNode::RangeQuery(range_MDS){
/*Query a node "thisNode" against a rangeMDS*/
result=0;
foreach dimension
  if(rangeMDS and thisNode's MDS are not at same level) then adapt MDS with
lower level to the one with higher level;
if (Overlap(rangeMDS,thisNode.MDS)>0) then {
  if (thisNode's MDS is contained in rangeMDS) then re-
sult+=thisNode.AggregateMeasureValue;
  else if (thisNode.NodeType == "SuperNode" or "DirectoryNode")
    foreach "entry" in thisNode, result+=entry.RangeQuery(range_MDs); } return
result; }
```

is updated. Subsequent insertion of t6 causes a split of N2 and produces the resulting tree structure shown in figure 1(c). Insertion of t7 results in the splitting of N'2 and consequently N1 that results in the final tree shown in figure 1(d).

Query. A range query is represented by an MDS that we call rangeMDS. For example, the query "Number of Parts sold tothe customers of Europe during the years 2000 to 2002" is represented by the rangeMDS [{Europe},{all},{all},{2000 ,2001,2002}]. The algorithm (see Algorithm 2) for range query takes a rangeMDS as input and queries the node, starting from the root. If the node's MDS is contained in the range_MDS, the node's aggregate value is added to the result. Else if node's MDS and rangeMDS overlap and the node is not a Data Node, then same algorithm is run recursively for all entries of the node.

For example, let us query the TiC-Tree in fig. 1(d) against above rangeMDS. The root's MDS is not contained in rangeMDS and overlap between them is greater than zero. Therefore, the algorithm continues searching the entries of N'1. MDSs of both the entries are not contained in rangeMDS and the overlap between the MDS of N6 and rangeMDS is also zero. However, N7's MDS and rangeMDS have overlap value greater than zero, and so as N3. The MDS of t2 is contained in the range MDS, so its measure value is added to the result while t5's MDS is not contained in the rangeMDS. As the leaves of the tree are reached, and no overlapping part of the tree is left, the method returns the result i.e. 60. The algorithm for group by query is almost the same, the only difference is in the input MDS. Group by query's MDS contains only one attribute per dimension at a time, e.g. "Number of Parts sold by region where region in (Europe, America)" is translated to group-by MDSs [{Europe},{all},{all},{all}] and [{America},{all},{all},{all}].

4 Experimentation and Results

In order to evaluate the performance, we developed the TiC-Tree as well as the DC-Tree. A 'csv' file containing the tuples of the fact table serves as input for

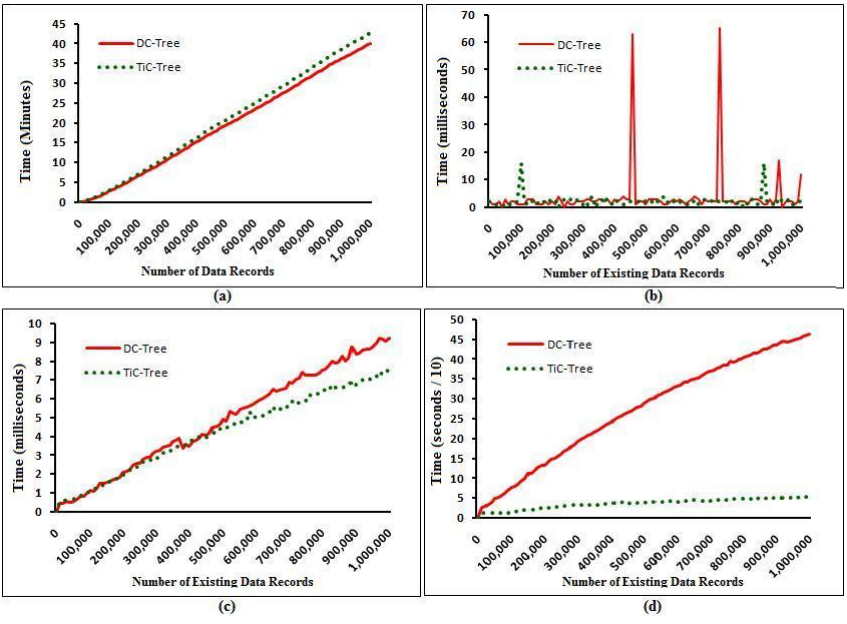


Fig. 2. Response time for: (a) Bulk Insertion (b) Single Record Insertion (c) Range Query (d) Group-by Query; on chronologically ordered data

these applications. All the tests were carried out on a system with 2.0 GHz Processor and 2GB RAM and evaluate the performance on the basis of insertion and query response time. The data warehouse schema for experimentation and performance evaluation is based on the Star Schema Benchmark. We use a fact table with 10,000 to 1,000,000 tuples, ordered chronologically and another data sets where some of the facts (5%, 10%,...) arrive out of time order, with the intention of studying the effect of these postponement. We execute a set of 100 range and group-by queries, for different levels (selected randomly) of dimension hierarchy on each state of the index resulting from the above insertion, average query execution time beeing recorded as result.

Figure 2(a and b) shows the comparitive results for bulk and single record insertion time, respectively, in DC-Tree and the TiC-Tree. In these cases, both indices show almost similar performance. Figure 2(c and d) summarizes the results of query response time for queries. In both cases TiC-Tree exhibits better performance than the DC-Tree. All the driven tests show that delay affects the insertion and query time only to a small extent and does not have any considerable effect on the performance of the index.

5 Conclusion

In this research work, we propose an index structure to store detailed and aggregated multidimensional data. In real time temporal data warehouses, members

of time dimension grow rapidly over time, and we believe the TiC-Tree can be a solution for handling this special case. We propose to use and redefine Minimum Describing Sequences, in order to take advantage of special nature of time dimension and special nature of temporal OLAP queries. We propose to keep temporally close values together in tree nodes in order to facilitate range searches and group-by. Performance evaluation tests show a significant improvement for range and group-by queries, as compared to the reference index. TiC-Tree provides an idea of considering the nature of dimensions for data indexing, in our case, the total ordering of temporal data. The TiC-Tree shows performance improvement in query response time but it is still unable to deal with the costly split algorithm that will need futher investigation.

References

1. Polyzotis, N., Skiadopoulos, S., Alkis Simitsis, P.V., Frantzell, N.E.: Supporting Streaming Updates in an Active Data Warehouse. In: Proc. of the 23rd Int. Conf. on Data Engineering (2007)
2. Tho, M.N., Tjoa, A.M.: Zero-latency data warehousing for hetrogeneous data sources and continuous data streams. In: Proc. of the Fifth Int. Conf. on Information Integration and Web-based Applications Services (2003)
3. O'Neil, P.E., O'Neil, E.J., Chen, X., Revilak, S.: The Star Schema Benchmark and Augmented Fact Table Indexing. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 237–252. Springer, Heidelberg (2009)
4. Gupta, H.: Selection of Views to Materialize in a Data Warehouse. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186. Springer, Heidelberg (1996)
5. Roussopoulos, N., Kotidis, Y., Roussopoulos, M.: Cubetree: organization of and bulk incremental updates on the data cube. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 89–99. ACM, New York (1997)
6. Sismanis, Y., Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Hierarchical dwarfs for the rollup cube. In: Proc. of the 6th ACM Int. Workshop on Datawarehousing and OLAP, NY, USA (2003)
7. Lakshmanan, L.V.S., Pei, J., Han, J.: Quotient cube: how to summarize the semantics of a data cube. In: Proc. of the 28th Int. Conf. on Very Large Data Bases, VLDB Endowment, pp. 778–789 (2002)
8. Tao, Y., Papadias, D.: Efficient Historical R-Trees. In: Proc. of the 13th Int. Conf. on Scientific and Statistical Database Management, Washington, DC, USA (2001)
9. Berchtold, S., Keim, D.A., Kriegel, H.P.: The X-tree: An Index Structure for High-Dimensional Data. In: Proc. of 22th Int. Conf. on Very Large Data Bases, Mumbai (Bombay), India, pp. 28–39 (1996)
10. Papadias, D., Tao, Y., Kalnis, P., Zhang, J.: Indexing spatio-temporal data warehouses. In: Proc. of the 18th Int. Conf. on Data Engineering (2002)
11. Ester, M., Kohlhammer, J., Kriegel, H.P.: The DC-tree: A Fully Dynamic Index Structure for Data Warehouses. In: Proc. of the 16th Int. Conf. on Data Engineering, pp. 379–388 (2000)