Storage and Indexing of Relational OLAP Views with Mixed Categorical and Continuous Dimensions

Oliver Baltzer, Andrew Rau-Chaplin, Norbert Zeh
Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
{obaltzer,arc,nzeh}@cs.dal.ca

Journal of Digital Information Management

ABSTRACT: *Due to the widespread adoption of location-based services and other spatial applications, data warehouses that store spatial information are becoming increasingly prevalent. Consequently, it is becoming important to extend the standard OLAP paradigm with features that support spatial analysis and aggregation. While traditional OLAP systems are limited to data characterized by strictly categorical feature dimensions, Spatial OLAP systems must provide support for both categorical and spatial feature dimensions. Such spatial feature dimensions are typically represented by continuous data values. In this paper we propose a technique for representing and indexing relational OLAP views with mixed categorical and continuous data. Our method builds on top of an established mechanism for standard OLAP and exploits characteristic properties of space-filling curves. It allows us to effectively represent and index mixed categorical and continuous data, while dynamically adapting to changes in dimension cardinality during updates. We have implemented the proposed storage and indexing methods and evaluated their build, update, and query times using both synthetic and real datasets. Our experiments show that the proposed methods based on Hilbert curves of dynamic resolutions offers significant performance advantages especially for view updates.*

Categories and Subject Descriptors
H.3.1[Content analysis and indexing]; E.1 [Data Structures]

General Terms
OLAP paradigm, Relational OLCP views, Data types

Keywords: Spatial OLAP, space-filling curves, view indexing, continuous space

Received 24 December 2006; Revised 15 March and 01 May 2007; Accepted 08 May 2007

1. Introduction

It is estimated that 80% of the data stored in data warehouses has some spatial component [MapInfo Corporation, 2006]. We are currently witnessing an explosion in geo-spatial data resulting from applications such as location-aware services [Chen and Kotz, 2000] and RFID tagging [Hahnel et al., 2004]. As data warehouses fill with such data, it becomes important to extend the standard OLAP analysis paradigm with new features that support spatial analysis and aggregation.

Spatial OLAP systems, sometimes referred to as SOLAP systems [Bédard et al., 2001], attempt to enhance the analysis and decision making process by combining features of OLAP with those of Geographic Information Systems (GIS) into a single tool [Microsoft Corporation, 2004, Cohen et al., 2004, KHEOPS Technologoies, 2005, Scotch and Parmanto, 2005a].

A fundamental difference between a traditional OLAP system and a SOLAP system is that in OLAP all feature dimensions are categorical, i.e., have fixed cardinalities such as Color $\in$ {red, blue, green}, while in SOLAP feature dimensions may be either categorical or spatial, where spatial dimensions are typically represented by continuous data values such as latitude = 13.32861° N. The categorical nature of feature dimensions in OLAP is typically exploited in several important ways. Firstly, categorical data stored in a star scheme (See Figure 1) can be normalized in each dimension, which leads to the imposition of a fixed-cardinality grid on the categorical dimensions. Secondly, dimension hierarchies can easily be defined as ranges of values within these grids to support roll-up and drill-down operations. For continuous dimensions, a different approach is necessary.

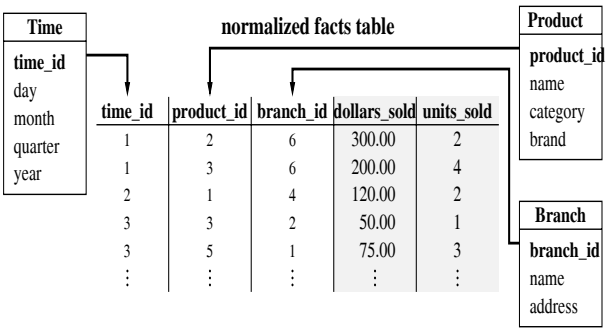

Figure 1. Star schema of a typical data warehouse with facts table, three feature dimensions and two measures. The feature dimensions have been normalized within the facts table in that they are represented as keys into dimension tables.

In adding non-categorical dimensions to OLAP, two key challenges must be addressed: (1) how to define and perform aggregation on spatial/continuous values, and (2) how to represent, index, update, and efficiently query mixed categorical and continuous data. Techniques for the aggregation of continuous spatial and temporal values have been well studied. For a broad survey of these results see [López et al., 2005]. The representation, indexing, and querying of spatial data has also been well studied [Rigaux et al., 2002, Güting, 1994]. However, data in a spatial data warehouse is not exclusively spatial: it is a mix of spatial and categorical attributes. The representation and indexing of such mixed categorical and continuous data in an OLAP setting has received less attention (see Section 2 for a discussion of related work). While efficient multi-dimensional indexing mechanisms for continuous data exist [Guttman, 1988], they are not specifically designed and optimized to meet the requirements of typical OLAP applications such as aggregate queries and hierarchy operations.

An obvious approach to the representation and indexing of mixed categorical and continuous data is to treat the continuous values as if they were values of a categorical space by simply enumerating them. This discretizes the original continuous dimension, allowing the use of established mechanisms for view indexing and query evaluation. However, it also skews the original continuous dimension in a data-dependent manner, and once continuous dimensions have been transformed in this way, view updates that preserve the relative order of records with respect to their original space are not possible without recomputation of the complete view.

In this paper we address the representation and indexing of mixed categorical and continuous data in a Relational OLAP setting. We first describe a general approach that has been shown to be effective for representing and indexing relational tables in a variety of settings (e.g., sequential [Jagadish, 1990, Moon et al., 2001], parallel [Dehne et al., 2003], and P2P [Schmidt and Parashar, 2004]) and then show how this method, based on the use of space-filling curves, can be extended to the mixed categorical/continuous setting.

We have implemented the proposed storage and indexing methods and evaluated their build, update, and query times using both synthetic and real datasets. Our experiments show that the proposed methods based on Hilbert curves of dynamic resolution offers significant performance advantages. In the build phase we observed a speed improvement by a factor of approximately 20-25% over the standard pre-discretization approach. For updates, which can be performed without reordering the entire dataset when using the dynamic-resolution approach, we observed performance improvements of between a factor of 23 for an update batch whose size is 2% of the current view and a factor of 6 for an update batch whose size is 20% of the current view.

As updates to data warehouses are commonly very small (less than 1 or 2%) the performance benefits of our dynamic adaptation approach over pre-discretization is quite significant.

We also compared the query performance of our Hilbert-curve-based index with a standard R-tree drawn from the Spatial Index Library [Hadjieleftheriou, 2006]. Again we observed a significant speed improvement. While the query time on the Hilbert-curve-based index increased only marginally with an increasing number of records and query results were reported in less than 0.2 seconds even for 3M records, the query time of the R-tree index quickly deteriorated for larger numbers of records. Note that some caution is required when interpreting this result. Since we are comparing two independent codebases, it is harder to determine how much of this improvement should be attributed to the improved I/O and cache efficiency of our algorithms and how much is simply due to better coding practices.

The remainder of this paper is organized as follows. Section 2 gives an overview of work related to this paper. Section 3 introduces our basic approach, while Section 4 describes our method for dynamically adapting the resolution of the Hilbert curve and proposes new algorithms for indexing views with both categorical and continuous space dimensions. The construction of the index and its representation in memory is described in Section 5. In Section 5.2 we discuss updates to such views and introduce a merge algorithm that is flexible with respect to dimension cardinality. In Section 6 we describe an initial implementation of our algorithms, whose performance we analyze in Section 6.1. The paper is concluded in Section 7 with a summary and discussion of future work.

2. Related Work

The problem of indexing multi-dimensional points is not new. A common and often effective approach to this problem is to employ a multi-dimensional R-tree or one of its many variants. The R-tree [Guttman, 1988] is a spatial indexing structure that allows the indexing and efficient querying of points and polygonal shapes. The queries that are supported by the R-tree are rectilinear range queries. Its original design was motivated by the B-tree, and it can be efficiently used as an external-memory data structure. For the application in spatial and spatio-temporal OLAP, a number of extensions [Papadias et al., 2001, Papadias et al., 2002, Tao and Papadias, 2001] of the R-tree have been proposed to facilitate efficient evaluation of queries typical for these domains. Most approaches based on R-trees, however, only focus on continuous space dimensions and don't specifically address the properties of categorical dimensions. To support categorical dimensions, these dimensions are typically transformed into an equivalent representation in continuous space, but this creates problems: The efficient aggregation in OLAP queries over categorical dimensions often crucially relies on these dimensions being integer-valued, and the use of space-filling curves as a locality-preserving mapping from higher-dimensional space into one dimension, discussed next, relies on the existence of at least an implicit integer grid.

Space-filling curves have been used to support the indexing of purely categorical OLAP data [Dehne et al., 2003]. This approach organizes multi-dimensional categorical OLAP views by using the Hilbert space-filling curve to generate a linear ordering of the records in multi-dimensional space and then indexing this linear ordering with a data structure similar to a B-tree. This method exploits that the Hilbert curve strongly preserves spatial locality [Moon et al., 2001].

Previously, Hilbert curves were used by Kamel and Faloutsos [Kamel and Faloutsos, 1994] as a mechanism to enhance the performance of R-trees when used for indexing multi-dimensional data. They used the Hilbert curve to obtain an ordering of the records within each node of the R-tree. This allows the exercise of new strategies for the distribution of records when nodes split and merge during insertion and deletion of records. Using these strategies, the query performance of the R-tree does not deteriorate as drastically as when arbitrarily distributing records between nodes during node splitting and merging.

Later in 2001, Lawder and King proposed a multi-dimensional index that uses a tree structure derived from the construction rules of the Hilbert curve [Lawder and King, 2001]. Each level of this tree corresponds to a level of resolution, or order, of the Hilbert curve and partitions the covered subspace into quadrants. At the root of the tree the entire space is covered. Records are stored at the leaf level of the tree in the subspace quadrants that correspond to the Hilbert values derived from their original coordinates.

None of the previous approaches to indexing multi-dimensional data with the help of Hilbert curves addressed the issue of continuous dimensions. All proposed techniques operate on multi-dimensional grids whose resolutions are known in advance, and records are mapped into cells of these grids. Hence, they require that the cardinality of each dimension is fixed and known in advance. This, however, is not practical in a spatial OLAP environment where multiple continuous dimensions may exists and their attribute

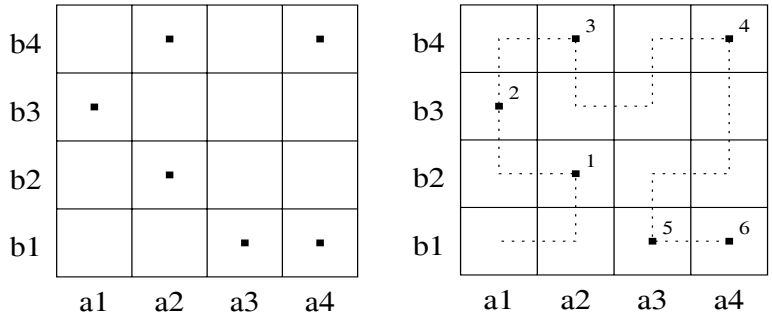

Figure 2. Hilbert curve (a) Grid for which each cell contains at most one record (b) Hilbert curve on grid. Points are numbered in order along the curve

values are not fixed and may change over any number of updates.

A more application-oriented approach to the integration of spatial and categorical data based on building composite systems that integrate existing OLAP and GIS systems has been pursued by both academic [Han et al., 1998, Rivest et al., 2001, Shekhar et al., 2001, Fidalgo et al., 2004] and industrial [KHEOPS Technologoies, 2005, Scotch and Parmanto, 2005b, da Silva et al., 2005, Hernandez et al., 2005] research groups. However, systems such as SOVAT [Scotch and Parmanto, 2005b] or Kheops Technology's JMap [KHEOPS Technologoies, 2005, Rivest et al., 2005] address the integration at a higher level, closer to the end-user, and internal data representation and storage mechanisms manage categorical and spatial data independently by means of traditional categorical data warehouses and spatial databases. While this approach speeds software development, it does not lead to systems that scale well in the face of massive datasets.

3. Our Approach

The central problem we address in this paper is the representation and indexing of mixed categorical and continuous data in a Relational OLAP setting. We first describe a general approach that has been shown to be effective for representing and indexing relational tables in a variety of settings (e.g., sequential [Jagadish, 1990], parallel [Dehne et al., 2003], and P2P [Schmidt and Parashar, 2004]) and then show how this method, based on the use of space-filling curves, can be extended to the mixed categorical/ continuous setting.

The approach has three basic steps:

1. Map the multi-dimensional data into a linear ordering using a locality-preserving space-filling curve, such as a Hilbert curve [Hilbert, 1891].

2. Use the linear order to distribute the data over the available storage. In the sequential setting this may be one or more disks [Jagadish, 1990], in the parallel setting this may be disks across a set of processors [Dehne et al., 2003], and in the P2P setting it may be disks across a collection of peers [Schmidt and Parashar, 2004].

3. Build an index structure on top of the ordered data for efficient query processing. Typically, this is some variant of the R-tree [Guttman, 1988].

The resulting indexing structure is basically an R-tree; however, the use of space-filling curves to fold the multi-dimensional space into a one-dimensional space is particularly well-suited for the even distribution of records over multiple disks, processors, or peers, and it facilitates batch updates of views by allowing to merge record updates into the original view with a single linear scan.

In an OLAP setting, this indexing structure presents a number of advantages:

- The disk layout favours sequential block access and thus reduces the amount of disk I/O and seeks.
- Multi-dimensional data is indexed in such a way that no dimension is favoured over another and the disk layout preserves the locality of the data.
- Extensions to the indexing structure can provide additional support for OLAP-specific operations such as range aggregate or roll-up queries [Papadias et al., 2001].

In this paper, we use the Hilbert curve as the space-filling curve that underlies our method. The Hilbert curve is defined on a *d*-dimensional grid with a side length of 2^k, where *d* denotes the dimension of the view, *c* is the largest cardinality among all dimensions, and $k = \lceil \log_2 c \rceil$ is the resolution or order of the Hilbert curve. This grid covers the entire space of the view so that each grid cell contains at most one record (see Figure 2(a)).

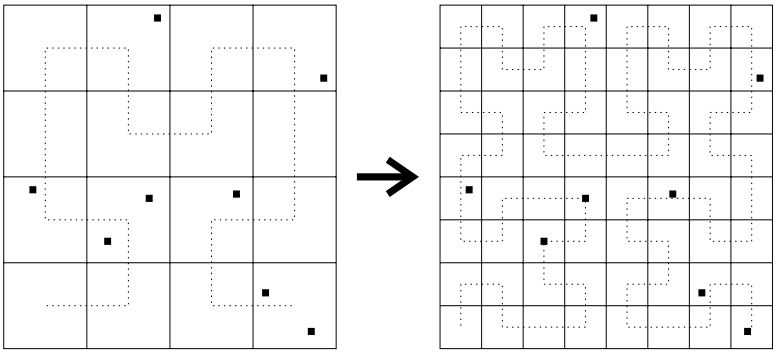

Figure 3. Increasing the resolution of the Hilbert curve preserves the order of records

The Hilbert curve on this grid passes through each record in the view exactly once and thereby generates the Hilbert order of the records (Figure 2(b)). The important property of the Hilbert curve is that it provides a mapping from d-dimensional space to 1-dimensional space that preserves much of the locality of the higher-dimensional space [Moon et al., 2001]. In the following section we show how to index continuous space dimensions without pre-discretization by proposing a technique that dynamically adapts the resolution of the Hilbert curve and determines the order of records locally. Unlike previous approaches that exploit the clustering properties of the Hilbert curve [Jagadish, 1990, Kamel and Faloutsos, 1994, Moon et al., 2001, Dehne et al. , 2003], this new technique is flexible with respect to dimension cardinality in the dataset, as it attempts to optimally utilize the grid space for continuous dimensions; it achieves a significant reduction of the cardinality of continuous dimensions when compared to a pre-discretization approach; and it also allows for the introduction of new records with previously unknown attribute values without the need to recompute the entire view. Introducing a batch of new records into an existing view has a cost proportional to that of sorting the records to be inserted in Hilbert order and linearly scanning the existing view to merge the new records into it.

4. Dynamic Resolution of Hilbert Curves

In this section we describe a method for ordering multi-dimensional records in Hilbert order when some of the dimensions are continuous. The main strength of this approach is that it is dynamic, allowing on-the-fly calculation of the resolution of the Hilbert space for continuous dimensions.

Generating a Hilbert ordering requires a discrete space in which each record is mapped to an associated coordinate and distinct records map to distinct coordinates. The naïve approach to generating a Hilbert ordering over a space that contains continuous dimensions would discretize those dimensions in advance and then generate the Hilbert ordering of the resulting entirely discrete space. This approach, however, exhibits a number of disadvantages that impact its effectiveness:

- The discretization of the continuous space requires multiple expensive sort and scan operations on the original dataset.
- Discretizing each continuous dimension individually causes the resulting space to be skewed, as only the relative order of records, but not the magnitude of their attributes, is preserved.
- The space resulting from the discretization is sparse and not well utilized, as each continuous value is assigned a distinct index, independent of whether or not this is in fact necessary to place each multi-dimensional record into its own grid cell.
- The entire space needs to be rediscretized when an update introduces a new attribute value.

A second approach is to map the continuous space attributes onto a multi-dimensional grid that defines the discrete space. However, predetermining the resolution of this grid requires the computation of pairwise Euclidean distances between records, which is not practical when the number of records is large.

The method proposed here takes a different approach. Based on the recursive definition and the self-similarity of the Hilbert curve, the Hilbert ordering of records can be efficiently computed in continuous space by using an adaptive resolution for the underlying discretization grid. We observe that a Hilbert ordering of records has the property that the relative order of records is preserved when the resolution of the Hilbert curve is increased (see Figure 3). Increasing the resolution of the Hilbert curve, in turn, increases the resolution of the multi-dimensional grid, on which the curve is defined. This property can be used to dynamically adjust the resolution of the grid onto which the continuous space attributes are mapped during the generation of the Hilbert ordering of the records, in order to guarantee that distinct records map to distinct grid cells and thus become comparable in Hilbert order.

The idea behind this approach is to resolve any conflict where two records that are compared map to the same grid cell while sorting the records in Hilbert order. This is done by increasing the resolution of the underlying grid until both records map to distinct grid cells. Due to the self-similarity of the Hilbert curve, changing the resolution of the Hilbert curve does not have an impact on records that have been sorted already. Thus, the resolution can be increased dynamically during the comparison of pairs of records. Furthermore, this technique achieves a better space utilization than a pre-discretization approach because a grid cell is split into smaller cells only if there is in fact a conflict between two records in that cell. Finally, since the grid is dynamically adjusted to the actual multi-dimensional data, the space is not skewed and the relative differences between the continuous values of records are preserved. Figure 3 illustrates how the increase of resolution resolves conflicts between records that are mapped to the same grid cell.

Algorithm 1 Algorithm to sort records in Hilbert order using dynamic resolution adaptation.

Procedure: hilbert-sort
Input: set R of records in no particular order
Output: set R of records in Hilbert order

$\triangleright$ compute initial resolution
1: $k \leftarrow \lceil \log_2 \max\{|D_i| : \forall D_i \in D\} \rceil$
$\triangleright$ call sorting algorithm with **hilbert-compare** as comparison function
2: **sort**$(R, \textbf{hilbert-compare}_k)$

Algorithm 2 Algorithm to determine the order of two records with respect to the Hilbert curve by dynamically adapting the resolution of the Hilbert curve if necessary

Procedure: hilbert-compare
Input: pair of records (r_1, r_2), initial resolution k
Output: -1 if $\mathrm{rank}_k(r_1) < \mathrm{rank}_k(r_2)$, 0 if $r_1 = r_2$ or 1 if $\mathrm{rank}_k(r_1) > \mathrm{rank}_k(r_2)$

```
if r1 = r2 then
    return 0
else
    while rank_k(r1) = rank_k(r2) do
        k ← k + 1 ▷ determine resolution suitable for comparison
    end while
    if rank_k(r1) < rank_k(r2) then
        return -1
    else
        return 1
    end if
end if
```

Algorithm 1 outlines a method to produce a Hilbert ordering of records that are defined in both continuous and discrete space. It uses a standard comparison-based sorting algorithm (e.g. Quicksort or Merge Sort) with a comparison function (Algorithm 2) that determines the relative order of two records on a Hilbert curve at a particular resolution. The relative order is determined by finding the minimum resolution of the Hilbert curve such that both records have a distinct rank with respect to the curve.

The resolution that is found during a comparison is used as the initial resolution for subsequent comparisons and may be increased. That way, the resolution of the grid is either maintained or increased for each comparison. After the sorting process, all records are in Hilbert order, and the resolution determined by the last comparison during the sorting process is the minimal resolution for a grid such that every record of the dataset maps to a distinct cell in that grid.

5. Exploiting Hilbert Order for I/O Efficient Indexing

Once the Hilbert ordering of the records has been determined, the records are sequentially written to disk in a block-wise manner. Each block on disk stores a constant number, B, of records that are consecutive in Hilbert order. Then an indexing structure is built on top of the ordered records that provides features comparable to those of a combination of a B-tree and an R-tree [Dehne et al., 2003].

Figure 4 illustrates the construction of such a tree structure with a specific example. While each intermediate node of the tree is very similar to a node in a conventional B-tree, it is also annotated with the minimum bounding box of the records in its subtree, similar to nodes in an R-tree. This allows for an efficient evaluation of multi-dimensional range queries while exhibiting most of the I/O efficient properties of B-trees. Due to the self-similarity of the Hilbert curve and its property to not favour any specific dimensions, the bounding boxes of the records in the subtrees at each level of the indexing structure are usually fat and usually overlap only very little, if at all. This is crucial for increasing the performance of queries. Figure 4 illustrates the evaluation of a two-dimensional range query on this indexing structure. Note that to answer a query, the tree is traversed in a breadth-first manner, thus limiting the disk accesses to a combination of sequential reads and forward-seek operations and reducing the amount of random access [Eavis, 2003].

5.1 Answering Range Aggregate Queries

Range aggregate queries are a very common query type in OLAP applications. In particular when spatial information is processed, range-aggregate queries over millions of points may be typical. The indexing structure described above can be used to answer range aggregate queries; however, without extensions it requires retrieval of each individual record within

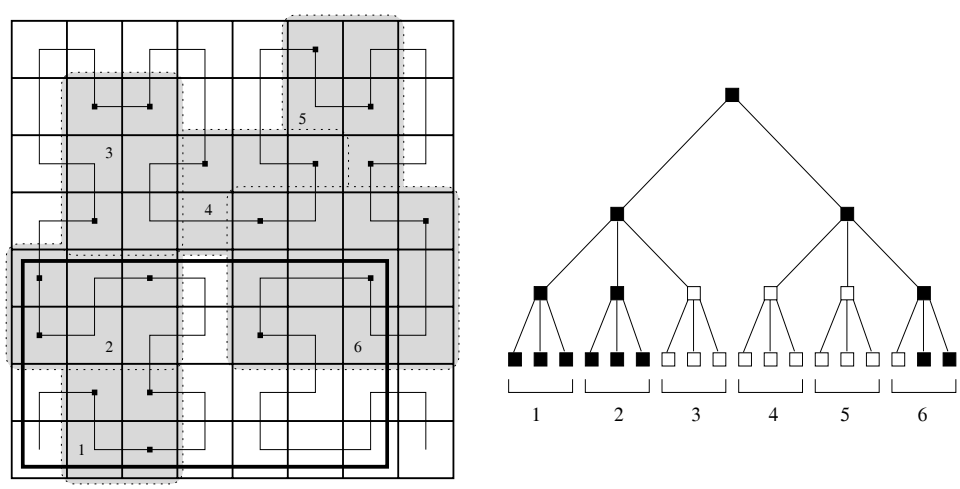

Figure 4. Records in Hilbert order are stored in consecutive blocks on disk and form multidimensional regions in the original space. The evaluation of a range query in breadth-first fashion only traverses part of the tree and results in a combination of sequential read and forward-seek operations at the leaf level.

the query range in order to compute a desired aggregate value To retrieve the required records, the tree is traversed by starting from the root and determining for each child node of a non-leaf node if its bounding box intersects with the query region. If so, all or some records in its subtree may contribute to the final query result, so the traversal continues in the subtree of this node. This traversal is performed in a breadth-first manner such that records at the leaves of the tree can be reported by a combination of sequential read and seek-forward operations. The retrieval of all records contained in a range to compute a single aggregate value is, however, computationally expensive and is often overkill because the records only contribute to the aggregate value and are discarded after this computation.

As proposed in [Papadias et al., 2001], a more efficient approach to answering aggregate range queries is the use of pre-aggregated values. The idea is to annotate non-leaf nodes in the tree with aggregate information derived from the points that are stored in the node's subtree. This supports an improved evaluation of aggregate queries, since a full traversal of the tree down to its leaf level may not be necessary to answer an aggregation query: When the bounding box of a non-leaf node is completely contained in the query region, all records in its subtree contribute to the final query result, and we can use the aggregate stored at the node instead of inspecting all records in its subtree. Thus, the answer to a query can potentially be assembled from a small set of partial results instead of inspecting all points in the query region.

To help answer aggregate OLAP queries efficiently, the aggregate information to be stored at each node is pre-computed while iterating over the children of the node during the bottom-up construction of the index tree. Note that this pre-computation of aggregate information works well for distributive (e.g., COUNT, SUM) and algebraic (e.g., AVERAGE) aggregation functions. Holistic aggregation functions (e.g., MEDIAN), on the other hand, require the retrieval of the actual records enclosed in the query region in order to compute the aggregate value. This can be supported by additionally annotating each non-leaf node with a direct reference to the sequence of leaves in its subtree. When evaluating a query, the records in a node's subtree can be reported immediately once the node has been reached, without traversing the remaining subtree below this node (see Figure 5).

5.2 Updating OLAP Views

A key advantage of using Hilbert curves of dynamic resolution over pre-discretization is that an existing, possibly very large view can be efficiently updated in a batch fashion in time proportional to the cost of sorting the records in the update batch and linearly scanning the existing view. To do so, all records of the update view are sorted, using the same comparison function as in Algorithm 2. The minimum resolution that has been determined while sorting the original records is used as the initial resolution when sorting the update records. Once the update records are in Hilbert order,

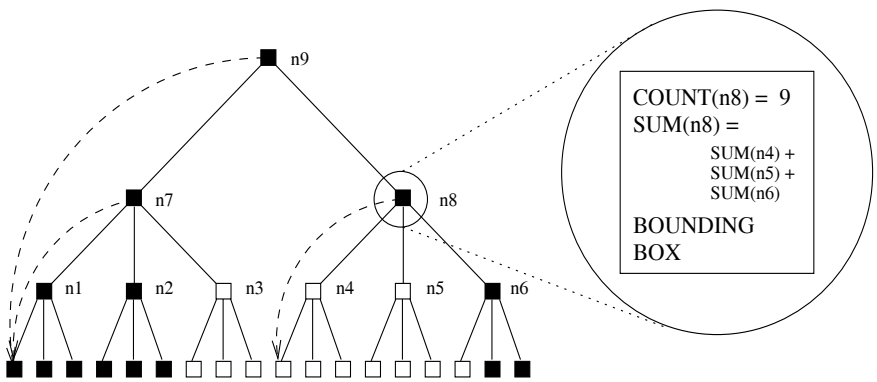

Figure 5. Annotation of non-leaf nodes with aggregate information and references to records at the leaf level

Algorithm 3 Merge algorithm to incorporate update dataset into target dataset.

Procedure: hilbert-merge
Input: stream U of update records in Hilbert order, stream T of existing target records in Hilbert order, resolution k of the Hilbert curve used to order the existing records
Output: merged stream O of records in Hilbert order

```
let u be the first element in U or empty if no such element exists
let t be the first element in T or empty if no such element exists
while u and t are not empty do
   if hilbert-compare(u, t, k) = 0 then
      write u to O
      let t be the next element in T or empty if no such element exists
      let u be the next element in U or empty if no such element exists
   else if hilbert-compare(u, t, k) < 0 then
      write u to O
      let u be the next element in U or empty if no such element exists
   else
      write t to O
      let t be the next element in T or empty if no such element exists
   end if
end while
if u is not empty then
   write the remainder of U to O
else if t is not empty then
   write the remainder of T to O
end if
```

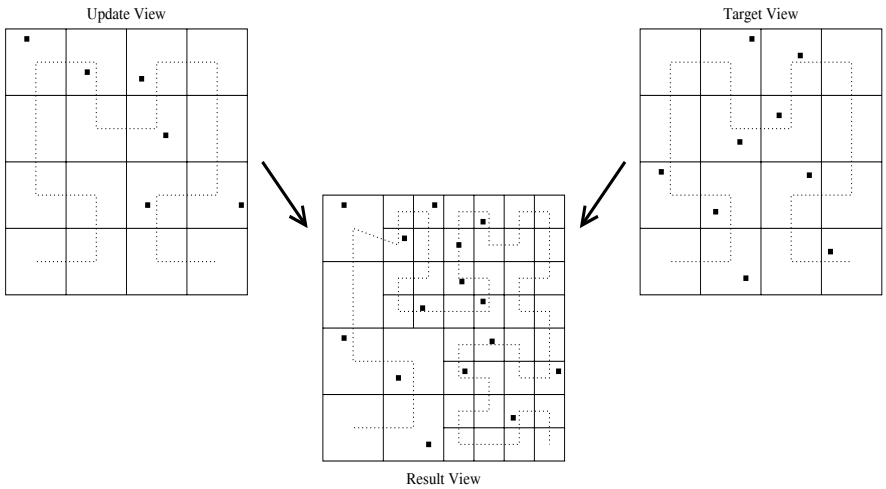

Figure 6. Merging the target view with the update view may result in a dynamic adaptation of the Hilbert curve resolution

they are merged with the original dataset in a single scan over both datasets. As update datasets in an OLAP environment are often only a fraction of the size of the entire data warehouse (typically 1%), the cost of sorting the update view is relatively small compared to rebuilding the entire view.

Algorithm 3 shows in detail how both sequences are merged. It iterates through the records in both datasets in Hilbert order and repeatedly compares the two current records from both datasets. If both records share the same attribute values, the record from the update dataset is considered an update of the existing record and therefore moved into the output dataset, while the original record is discarded. If the attribute values of both records differ in at least one dimension, the Hilbert ranks of both records at the initial resolution are determined and compared. If one record has a lower Hilbert rank than the other, it is moved to the output dataset and a new current record is fetched from the respective input dataset. In the case when the same Hilbert rank is computed for both records, the resolution has to be increased until the Hilbert ranks differ. Since the locality of records mapped into Hilbert space is preserved when increasing the resolution of the Hilbert curve, records that have been sorted already are not affected by the increase of resolution (see Figure 6). The merging process continues until all records from both datasets have been processed. The output dataset is a set of records in Hilbert order and will contain all records from the update dataset as well as those from the original dataset that have not been updated. Following the merging process, the indexing data structure is rebuilt over the output dataset.

6. Implementation and Experiments

Based on the proposed approach, an initial prototype, called geoCUBE, was implemented and extensively evaluated. The focus of the implementation was on determining the Hilbert order for records with attribute dimensions defined in continuous and discrete space and on evaluating query performance and the construction and batched update time of the proposed indexing structure. The software was written in C and includes implementations of Algorithms 1, 2 and 3 as well as variations of them to explore performance trade-offs.

Computing a point's mapping from n-dimensional discrete space to one-dimensional Hilbert space was implemented based on a modified version of Doug Moore's Hilbert mapping library [Moore, 1998]. The data types that are supported by the implementation are 64-bit integers and 32-bit floating-point numbers. The implementation presented here assumes that the attribute values of each continuous space dimension are normalized to the interval **[0.0,1.0)** and that the attribute values of each discrete space dimension are natural numbers in the interval **[0, c -1]** , with *c* being the cardinality of the dimension.

6.1 Performance Evaluation

The experimental platform was a workstation with one Intel Xeon 1.8GHz processor and 1.5 GB RAM, running FreeBSD 6.2. The compiler used to translate the C programs was part of the GNU Compiler Collection version 3.4.6.

The experiments we conducted evaluate the cost of sorting records into Hilbert order, building the index, evaluating queries, and performing view updates. The running times were measured as wall-clock times in seconds. Both synthetic and "real-world" datasets were used in the evaluation. Synthetic datasets were generated with a uniform distribution so that we could better understand the effects of various dataset parameters on performance. The categorical and continuous dimensions of the synthetic datasets were generated with a cardinality of 64 and 1000 respectively. The real-world datasets were drawn from the HYDRO1k dataset published by the U.S. Geological Survey and, where necessary, were reduced in dimensionality and size through random sampling. The main test dataset was a 6-dimensional dataset composed of two categorical dimensions, with cardinalities 57 and 51, and four continuous dimensions with cardinalities 956, 1000, 802 and 288. It was used in each of the following experiments unless explicitly stated otherwise.

Throughout the experiments, the impact of the sizes of the datasets on the running times of the various algorithms was investigated. Additionally the benefit of proposed optimizations was determined, and the traditional method [Jagadish, 1990, Moon et al., 2001, Dehne et al., 2003] of pre-discretizing continuous space into discrete space was compared to the new dynamic approach. Unless stated otherwise, all experiments were performed with the most beneficial

optimization enabled. In particular, each record was annotated with its last computed Hilbert rank.

For completeness, the evaluation also contains a performance comparison of the geoCUBE prototype to a conventional R-tree implementation drawn from the Spatial Index Library [Hadjieleftheriou, 2006].

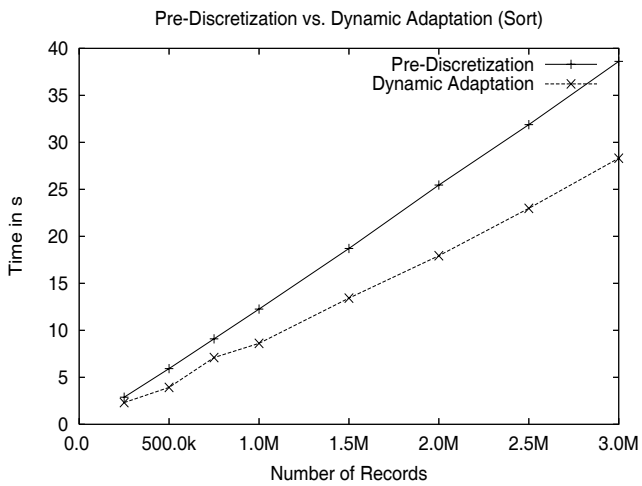

Figure 7. Overhead of pre-discretization compared to dynamic resolution adaptation

6.1.1 Dynamic adaptation versus pre-discretization of continuous dimensions

Figure 7 shows a direct comparison of the dynamic adaptation of the Hilbert curve resolution versus the traditional method of pre-discretization. The top curve represents the time required to pre-discretize and sort the synthetic dataset into Hilbert order. The pre-discretization involves the sorting of each continuous space dimension and the subsequent enumeration of their unique values, resulting in only categorical dimensions. The dataset is then sorted in Hilbert order with a resolution derived from the cardinalities of the dimensions. The bottom curve represents the time required for sorting the same synthetic dataset into Hilbert order using our dynamic resolution adaptation approach. As one can see, dynamic adaptation of the Hilbert curve performs better than the pre-discretization approach. In particular for 3 million records, the pre-discretization approach requires approximately 38 seconds to pre-discretize and sort the dataset, while the dynamic approach takes only 28 seconds. This corresponds to a speed improvement of about 26%, which we also observed for most other datasets we tested. We believe that this better performance is the result of avoiding the expensive process of identifying distinct attribute values.

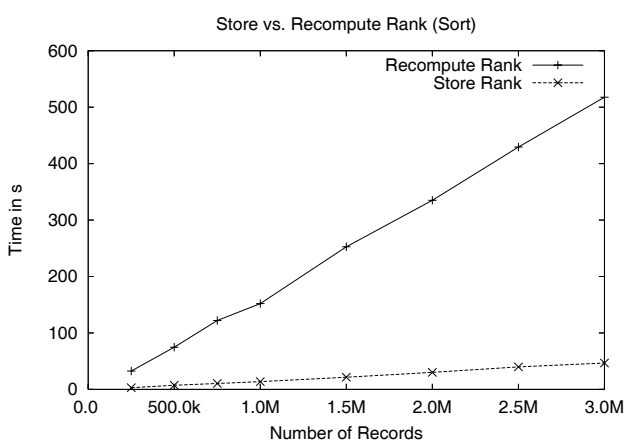

Figure 8. Recomputing each record's Hilbert rank for each comparison versus storing the last computed Hilbert rank of each record

Before computing the Hilbert order of records by using an adaptive increment of the Hilbert curve resolution, the base resolution, k_{base}, that is required to map all discrete space dimensions into Hilbert space must be computed from their cardinalities. In the following, this base resolution is initialized to $k_{base} = \log_2 c_{max}$ with $c_{max} = \max|D_i|$ if there exists at least one discrete dimension, or $k_{base} = 0$ if only continuous space dimensions are defined.

To compare two records based on their Hilbert ranks, the ranks of both records have to be computed at the same resolution. With $O(n \log n)$ comparisons to sort *n* records, recomputing the Hilbert rank of a record for each comparison it is involved in is computationally very expensive. To limit the amount of recomputation, each record can be annotated with its last computed rank and the resolution for which this rank is valid. Even though the resolution may increase, causing the Hilbert rank of the record to be recomputed, this will affect only records that are not in their final position yet and those for which the rank has not been computed yet. Also, instead of globally increasing the resolution for all subsequent comparisons, the resolution is only increased locally for each comparison if necessary. This avoids the computation of Hilbert ranks at an unnecessarily high resolution for other records. In practice, storing the last computed Hilbert rank reduces the number of rank calculations considerably and improves the overall sort time. The main drawback of storing the computed rank along with each record is a substantial increase in the space requirements. In the best case, the overhead, in bits, of storing the Hilbert rank at a given resolution *k* for *d* dimensions is, $\sum_{=1}^{d} \min\{k, [\log_2 |D_i|]\}$ where $|D_i|$ is the cardinality of dimension D_i. However, this is only true when continuous dimensions are pre-discretized.

For the dynamic mapping of continuous dimension values and the preservation of their spatial relationships, k may become significantly larger than $\lceil \log_2 |D| \rceil$ as the minimum difference of normalized values of two records is the decimal precision *p* of the continuous dimension (e.g. $p = 10^{-3}$). In this case the overhead can be quantified as ,

$$\sum_{i=1}^{z} \min\{k, \lceil \log_2 |Z_i| \rceil\} + \sum_{i=1}^{r} k$$

where *z* is the number of categorical dimensions *Z*, *r* the number continuous dimensions and $k \leq [\log_2 1/p]$.

Figure 8 shows the impact of annotating each record with its last computed Hilbert rank compared to recomputing the record's Hilbert rank during each comparison. The dataset used for this experiment was the HYDRO1k dataset. As can be clearly seen, the optimization has a very significant effect on the overall running time and results in an order of magnitude performance gain. Specifically, for 3 million records, the annotation approach takes 46 seconds, while the recomputation approach takes 517 seconds; this represents a performance improvement by a factor of 11.2.

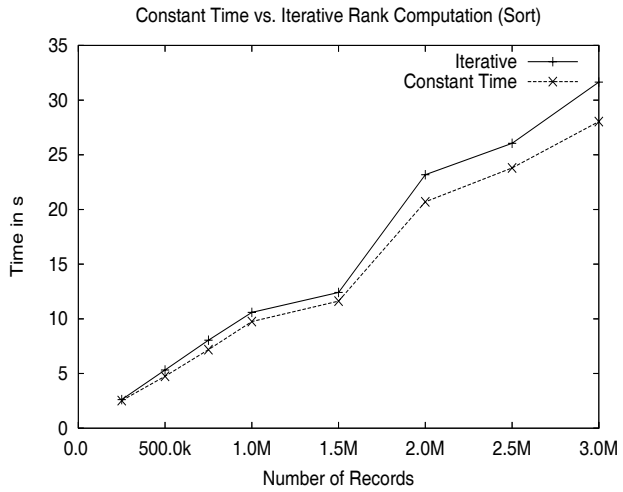

Figure 8. Recomputing each record's Hilbert rank for each comparison versus storing the last computed Hilbert rank of each record

6.1.3 Optimization 2: Iterative versus constant time resolution determination

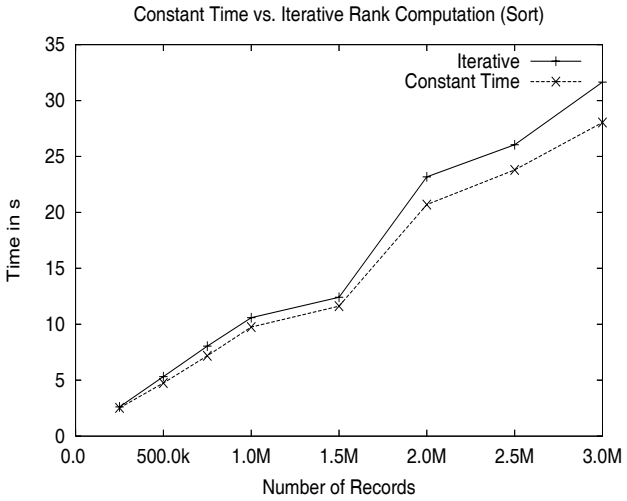

Figure 9. Iterative versus constant-time resolution determination

Another possible bottleneck in the proposed comparison function is the iterative determination of the grid resolution that guarantees that no two records share the same rank. In the worst case, this requires many iterations, and consequently recomputations of Hilbert ranks for the records that are compared, if the two records are sufficiently close to each other in continuous space. It would be desirable to determine the necessary minimal resolution with a constant amount of computation for any two records. The approach presented here takes constant time to determine a resolution at which both records do not share the same rank. This is achieved by computing the Euclidean distance between both records and determining the resolution at which the diagonal of a grid cell is shorter than the distance between the records. This guarantees that both records are mapped to distinct cells, but the determined resolution may not be minimal, as there may be a lower resolution that already maps both records into different cells. The disadvantage of obtaining a resolution that is not minimal is an additional space requirement and a slight increase in computation required to determine the Hilbert rank at this resolution.

Figure 9 illustrates the effects that the method of resolution determination has on the overall running time of the algorithm. For datasets with only continuous space dimensions, theon-the-fly determination of the resolution is essential, since no initial resolution can be computed. The curves presented in Figure 9 were obtained by experiments with the HYDRO1k dataset. The top curve represents the performance of the iterative method. Note that this method always finds the minimal resolution necessary to compute distinct Hilbert ranks for all records. The bottom curve shows the performance of the constant-time method. This method performs approximately 10% better than the iterative method. In some cases, however, the determined resolution of the Hilbert curve may not be minimal, causing the computation of each Hilbert rank to become more expensive. Also, as records may be arbitrarily close to each other in the original space, the constant-time approach may determine a resolution that is significantly higher than the minimum resolution determined by the iterative approach. Consequently, there is a trade-off between the performance gain of the constant-time approach and the likelihood that it will exceed the available storage to represent Hilbert ranks.

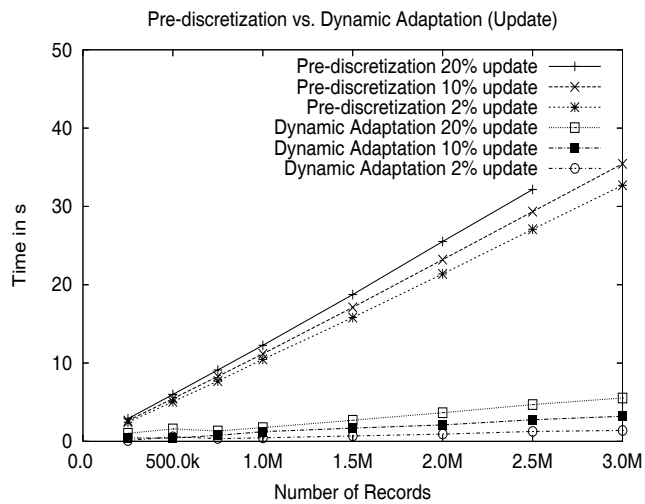

Figure 10. Batch update time for different update sizes.

6.1.4 Update Time

The implementation of the update mechanism follows the description in Algorithm 3. First the update dataset is sorted into Hilbert order using, as a starting resolution, the minimum resolution that was determined for the original dataset. The next step then merges both, the original and the update dataset, by sequentially scanning through them and repeatedly comparing the next records from both datasets. If both records are equivalent, only the update record is written to the output dataset; otherwise, the comparison function used for sorting both datasets is used to determine the order in which the records should be written to the output dataset. As in the sorting step, if the order of two distinct records cannot be determined at a given resolution, the resolution of the Hilbert curve is dynamically increased until both records map to distinct ranks on the Hilbert curve.

The main advantage of this approach is that only the updates need to be sorted, while a linear scan of the updated dataset is sufficient. In contrast, when applying updates that introduce new attribute values, the pre-discretization approach requires to newly discretize the entire merged dataset.

Figure 10 shows the times required to obtain the Hilbert order of the updated dataset for updates with 2%, 10% and 20% of the sizes of the original dataset. The graph compares the update performance of both the pre-discretizationof the entire updated dataset, its total update time is composed of the time required to sort the update into Hilbert order and the time required to sequentially merge the original dataset with the update dataset. Hence, our dynamic approach and our

dynamic adaptation approach. Since the dynamic adaptation approach does not require the resorting adaptation approach performs updates significantly faster in comparison with the pre-discretization approach, which requires the sorting of the entire updated dataset. In particular, we observe an improvement between a factor of 23 for the 2% update and a factor of 6 for the 20% update. As updates to data warehouses are commonly very small (less than 1%) the performance benefits of our dynamic adaptation approach over pre-discretization, especially for small updates, is very significant.

6.1.5 Index Construction

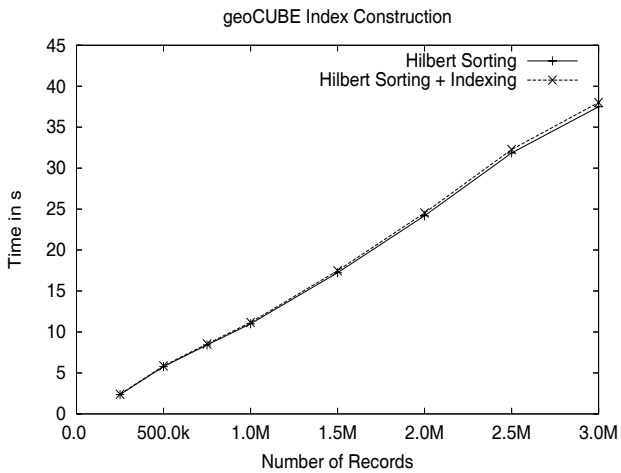

Figure 11. Time to construct the index including sorting of the dataset for the geoCUBE index.

The construction of the indexing structure is implemented as a bottom-up approach starting with the sorted data at the leaf level of the tree. At each non-leaf level, nodes at this level have a fan-out of *f* children, such that nodes from the level immediately below are combined into groups of size *f*. For each of these groups, the corresponding parent node in the tree is annotated with the group's minimum bounding box. The nodes of each level of the tree are stored as consecutive segments in memory to minimize the effect of fragmentation. This method to construct the index turned out to be extremely efficient, accounting for about 1% of the total time required to build the geoCUBE for a given dataset, as shown in Figure 11.

6.1.6 Range Query Performance

Figure 12 shows the average range query time on the HYDRO1k dataset over 1000 experiments using our geoCUBE index and an R-tree index drawn from the Spatial Index Library [Hadjieleftheriou, 2006]. The queries were constructed as hypercuboids from pairs of records, randomlysampled from the dataset and involving all dimensions of the dataset. This graph clearly shows a superior query performance of the geoCUBE index compared to this R-tree implementation. The geoCUBE query time increases only marginally with an increasing number of records, and query results are reported in less than 0.2 seconds even for 3M records. The query time of the R-tree index, on the other hand, quickly deteriorates for larger numbers of records, resulting in query times of more than 1.5 seconds for a single query. Note that some caution is required when interpreting these results. Since we are comparing two independent codebases, it is harder to determine how much of this improvement should be attributed to the improved I/O and cache efficiency of our algorithms and how much is due to improved coding practices.

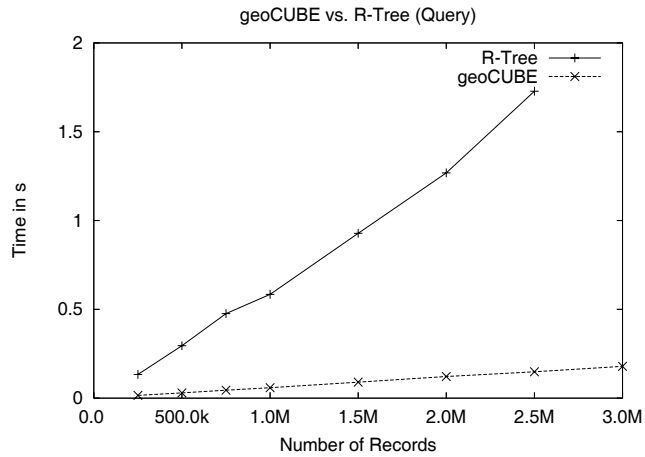

Figure 12. Query performance of geoCUBE vs. R-tree for real world data

7. Conclusion and Future Work

In this paper we have proposed a new technique for representing and indexing relational OLAP views with mixed categorical and continuous dimensions. Our approach is flexible with respect to dimension cardinality and thus allows for the indexing of continuous space dimensions while building on top of established mechanisms for index construction and querying. Our contribution is significant as it integrates the representation of mixed categorical/ continuous data at the storage level, thereby forming the basis for efficient Spatial OLAP systems that can handle massive amounts of data. Such systems are gaining in importance with the increasing amount of spatial data that is collected. Our experimental evaluation shows the practical benefits of the proposed approach.

We are currently exploring extensions of our approach to support the indexing of spatial objects with extent as well as the use of compressed Hilbert indices [Hamilton, 2006]. We are also exploring how to efficiently support specific OLAP operations such as roll-up and drill-down on spatial dimensions.

References

[1] Bédard, Y., Merrett, T., Han, J (2001). Research Monographs in GIS, chapter Fundamentals of Spatial Data Warehousing for Geographic Knowledge Discovery *In*: Geographic Data Mining and Knowledge Discovery, pages 53–73. Taylor & Francis.

[2] Chen, G., Kotz, D (2000). A survey of context-aware mobile computing research. Technical report, Dartmouth College, Hanover, NH, USA.

[3] Cohen, T. L., Baitty, J. R., Plamer, R. G., Adams, K. T., Weyl, J. A (2004). Health Resources and Services Administration Geospatial Data Warehouse. *In:* ESRI 2004 International Users Conference.

[4] da Silva, J., Times, V. C., Fidalgo, R. N., Barros, R. S. (2005). Web Technologies Research and Development - APWeb 2005, chapter Providing Geographic-Multidimensional Decision Support over the Web, pages 477–488. Springer Berlin.

[5] Dehne, F., Eavis, T., Rau-Chaplin, A (2003). Parallel multi-dimensional ROLAP indexing. *In:* ccGRID: 3rd International Symposium on Cluster Computing and the Grid, page 86.

[6] Eavis, T (2003). Parallel relational OLAP. PhD thesis, Dalhousie University.

[7] Fidalgo, R. N., Times, V. C., Silva, J., Souza, F. F. (2004). Data Warehousing and Knowledge Discovery, chapter GeoDWFrame: A Framework for Guiding the Design of Geographical Dimensional Schemas, p. 26–37. Springer Berlin.

[8] Güting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal* 3 (4) 357–399.

[9] Guttman, A (1988). R-trees: a dynamic index structure for spatial searching. Readings in database systems, pages 599–609.

[10] Hadjieleftheriou, M (2006). Spatial Index Library. http://u-foria.org/marioh/spatialindex

[11] Hahnel, D., Burgard, W., Fox, D., Fishkin, K. P., Philipose, M (2004). Mapping and localization with RFID technology. *In:* International Conference on Robotics and Automation, 2004. IEEE.

[12] Hamilton, C (2006). Compact Hilbert indices. Technical Report CS-2006-07, Dalhousie University, Faculty of Computer Science.

[13] Han, J., Stefanovic, N., Koperski, K (1998). Selective Materialization: An Efficient Method for Spatial Data Cube Construction. *In:* PAKDD '98: Proceedings of the Second Pacific-Asia Conference on Research and Development in Knowledge Discovery and Data Mining, p. 144–158, London, UK. Springer-Verlag.

[14] Hernandez, V., Voss A., Gohring, W (2005). Sustainable Decision Support by the Use of Multi-Level and Multi-Criteria Spatial Analysis on the Nicaragua Development Gateway, From Pharaohs to Geoinformatics. *In:* Proceedings of the F´ed´eration Internationale des G´eomètres Working Week 2005 and GSDI-8, p. 16–21.

[15] Hilbert, D (1891). Über die stetige Abbildung einer Linie auf ein Flächenstück. Mathematische Annalen, 38:459–460.

[16] Jagadish, H. V (1990). Linear clustering of objects with multiple attributes. *In:* SIGMOD '90: ACM SIGMOD International Conference on Management of Data, p. 332–342, New York, NY, USA. ACM Press.

[17] Kamel, I., Faloutsos, C. (1994). Hilbert R-tree: An improved R-tree using fractals. *In:* Proceedings ofthe Twentieth International Conference on Very Large Databases, p. 500–509, Santiago, Chile.

[18] KHEOPS Technologoies (2005). JMap Spatial OLAP – Innovative technology to support intuitive and interactive exploration and analysis of spatio-temporal multidimensional data. White paper.

[19] Lawder, J. K., King, P. J. H. (2001). Querying multi-dimensional data indexed using the Hilbert space-filling curve. SIGMOD Record, 30 (1) 19–24.

[20] López, I. F. V., Snodgrass, R. T., Moon, B. (2005). Spatiotemporal Aggregate Computation: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, 17 (2) 271–286.

[21] MapInfo Corporation (2006). Location Intelligence: The New Geography of Business. whitepaper, Business Week Research Services.

[22] Microsoft Corporation (2004). U.S. Government Agency Stores Aerial Imagery in New 25-Terabyte Warehouse Growing Warehouse. Case study.

[23] Moon, B., Jagadish, H., Faloutsos, C., Saltz, J. H. (2001). Analysis of the clustering properties of the Hilbert space-filling curve, *Knowledge and Data Engineering*, 13 (1) 124–141.

[24] Moore, D (1998). C Hilbert mapping library. http://computation.pa.msu.edu/O/F90/SFC/hilbert.c

[25] Papadias, D., Kalnis, P., Zhang, J., Tao, Y (2001). Efficient OLAP Operations in Spatial Data Warehouses. *In:* SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, p. 443–459, London, UK. Springer-Verlag.

[26] Papadias, D., Tao, Y., Kalnis, P., Zhang, J. (2002). Indexing Spatio-Temporal Data Warehouses. In Proceedings 18th International Conference on Data Engineering, pages 166–175. IEEE.

[27] Rigaux, P., Scholl, M., Voisard, A (2002). Spatial Databases with Application to GIS. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[28] Rivest, S., Bédard, Y., Marchand, P (2001). Toward Better Support for Spatial Decision Making: Defining the Characteristics of Spatial On-Line Analytical Processing (SOLAP). Geomatica, 55 (4) 539 – 555.

[29] Rivest, S., Bédard, Y., Proulx, M.-J., Nadeau, M., Hubert, F., Pastor, J (2005). SOLAP technology: Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data. *ISPRS Journal of Photogrammetry & Remote Sensing,* 60:17–33.

[30] Schmidt, C., Parashar, M (2004). Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, 08 (3) 19–26.

[31] Scotch, M., Parmanto, B. (2005a). Development of SOVAT: A numerical-spatial decision support system for community health assessment research. *International Journal of Medical Informatics*.

[32] Scotch, M., Parmanto, B. (2005b). Sovat: Spatial olap visualization and analysis tool. *In:* HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 6, page 142.2, Washington, DC, USA. IEEE Computer Society.

[33] Shekhar, S., Lu, C., Tan, X., Chawla, S., Vatsavai, R. (2001). Map Cube: A Visualization Tool for Spatial Data Warehouses. *In:* Miller, H. and Han, J., editors, Geographic Data Mining and Knowledge Discovery, p. 74–109. Taylor & Francis London.

[34] Tao, Y., Papadias, D (2001). Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, p. 431–440, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Oliver Baltzer received his Dipl.-Ing. in Computer Engineering from Fachhochschule für Technik und Wirtschaft Berlin and his M.Sc. in Network Centred Computing from Reading University in 2002. He is currently a Ph.D. student at the Faculty of Computer Science at Dalhousie University. His research interests are location intelligence and spatial data warehousing with focus on high-performance spatial OLAP.

Professor Rau-Chaplin received a M.C.S. and Ph.D. from Carleton University in Ottawa Canada in 1990 and 1993, respectively. From 1993 to 1994 he was a Postdoctoral Fellow at DIMACS - a National Science Foundation center run by Princeton University, Rutgers, and AT&T Bell labs. In 1994 he joined the Technical University of Nova Scotia and is currently a Professor in the Faculty of Computer Science at Dalhousie University. He is interested in the application of parallel computing to a wide range of problem domains including data warehousing & OLAP, bioinformatics, computational biochemistry, geographic information systems, and computational geometry/CAD.

Norbert Zeh received his Dipl.-Inf. from Friedrich-Schiller-Universität Jena in 1998 and his PhD in Computer Science from Carleton University in 2002. After a short postdoc at the Department of Computer Science of Duke University in 2002, he joined the Faculty of Computer Science at Dalhousie University in 2003, where he currently holds Assistant Professor and Tier II Canada Research Chair positions.