Journal of Computer Science 1 (1): 89-97, 2005
ISSN 1549-3636
© Science Publications, 2005

A Class of Region-preserving Space Transformations for Indexing High-dimensional Data

Ratko Orlandic and Jack Lukaszuk
Department of Computer Science, Illinois Institute of Technology
10 West 31st Street, 236SB, Chicago, IL 60616

Abstract: This study introduces a class of region preserving space transformation (RPST) schemes for accessing high-dimensional data. The access methods in this class differ with respect to their space-partitioning strategies. The study develops two new static partitioning schemes that can split each dimension of the space within linear space complexity. They also support an effective mechanism for handling skewed data in heavily sparse spaces. The techniques are experimentally compared to the Pyramid Technique, which is another example of static partitioning designed for high-dimensional data. On real high-dimensional data, the proposed RPST schemes outperform the Pyramid Technique by a significant margin.

Key words: Database Systems, Access Methods, Space-partitioning Strategy, Data Dimensionality

INTRODUCTION

As data availability has increased, so has the dimensionality of problems to be solved. Objects of multimedia applications are usually mapped to feature vectors indexed by a multi-dimensional database. Since typical feature vectors have numerous components, the resulting data are characterized by very high dimensionality. High-dimensional data also pose major challenges for data-mining algorithms and many scientific applications. For example, the results of high-energy physics experiments are typically depicted as rich parameter spaces of up to 200 dimensions. Similar data spaces appear in many other applications, including environmental studies and astronomy.

Unfortunately, the performance of traditional multi-dimensional access methods [1] rapidly deteriorates as data dimensionality grows. As a result, access methods for high-dimensional data [2-9] continue to attract considerable scientific interest.

It has already been observed that the space-partitioning strategy may have a profound effect on the retrieval performance in high-dimensional situations [2, 4, 10]. This observation gave rise to several retrieval techniques for high-dimensional data. For example, to improve the performance in high-dimensional spaces, the Pyramid Technique [2] statically partitions the d-dimensional space into 2d pyramids that meet at the center of the universe. The Hybrid Tree [4] follows a different approach. As long as the splits of index nodes do not require downward propagation, whose negative consequences are particularly severe in high-dimensional situations, the structure uses the space partition of KDB-trees [11] into non-overlapping regions. However, in order to prevent the downward cascading splits, it allows certain amount of overlap between the index regions [4].

Some point access methods do not perform any partition of the multi-dimensional space. Typical examples are the point-transformation schemes that use a space-filling curve to map points in the d-dimensional space onto one-dimensional index keys [12]. However, in spaces with many dimensions, the complexity of the query transformation tends to be prohibitive. The problem is that space-filling curves distort the neighborhoods in the original space and the distortions become more pronounced as data dimensionality grows. The techniques that apply dimensionality reduction [13,14] are also appropriate only in environments with strongly correlated data.

However, even if a space partition is applied, the contemporary partitioning strategies experience serious problems in high-dimensional situations. For example, to make sure that each axis is partitioned at least once, the traditional partitioning schemes would require 2^d divisions (where, d is data dimensionality) [2], which could be much larger than the number of points. Since a division of an index region is typically performed only when the number of points in the region exceeds certain limit, for all realistic data-set sizes and high data dimensionality, certain dimensions of the index regions are not partitioned at all. As a result, these dimensions do not contribute anything to the selectivity of the structure. As the number of dimensions grows, more sides of the given query window are ignored [2]. Typical partitioning schemes may also suffer from the problems of dead space (indexed space that contains no data objects) [4] and region overlap (space covered by more than one index region) [15].

This study introduces a class of region-preserving space transformation schemes for indexing high-dimensional data. Just like the Pyramid Technique, the structures in this class employ two distinct layers. The higher layer, which statically partitions the space into index regions, maps multi-dimensional points and queries onto their one-dimensional counterparts. The lower layer organizes the resulting index keys into an exact-match retrieval structure, e.g. a B^+-tree [16].

Virtually the only thing that changes across the structures in the class is the space-partitioning strategy. The partitioning schemes developed in this study, called Γ and Θ, allow configurable and overlap-free space partitions, making sure that every axis is partitioned several times. Therefore, each dimension of the space can effectively contribute to the search process. Just like the Pyramid Technique, Γ and Θ avoid another problem discussed in Berchtold [2], which is associated with access methods that strive to partition the space symmetrically (e.g., KDB-trees and R-trees [17]. The problem is that a small region query positioned somewhere in the middle of a high-dimensional space may force the traversal of the entire index.

Even though there are similarities between Γ and Θ partitioning strategies and the Pyramid Technique, the proposed partitioning schemes have significant advantages over the later technique. Since the number and coordinates of index regions do not depend solely on data dimensionality and can be tuned to fit the data distribution, they are much more flexible than the Pyramid Technique. Since individual regions have rectangular shape, the calculations required to identify the regions that must be searched are simpler. Such regions also allow an effective way of dealing with the problem of dead space. In addition, Γ and Θ partitioning schemes reduce the magnitudes of other problems that accompany the Pyramid Technique, which include non-unique key values, false drops, loss of proximity, and the enlargement of queries.

GAMMA AND THETA PARTITIONING

Figure 1a illustrates a Γ partition of a 2-dimensional space. By placing a smaller rectangle in one of the corners of the space (let it be the low right corner for now), we carve out a portion of this space. The remaining subspace takes the form of a Greek Γ from which we derived the name. Since we still want rectilinear subdivision, this remaining Γ subspace must be divided further. The dashed line indicates one possible choice. The inner rectangle can be recursively carved in the same fashion to obtain as many Γ subspaces as desired.

In Fig. 1b, the nested box appears in the origin (low left corner) of the space. This convention will be adopted in the rest of the study. In a 3-dimensional space, a Γ subspace (space inside one and outside its immediately enclosed box) is divided by a 2-dimensional plane lying on one side of the inner box (by Γ subspace, we mean

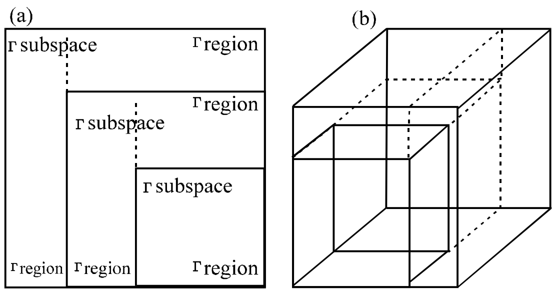


Fig. 1: Examples of Γ Space Partition

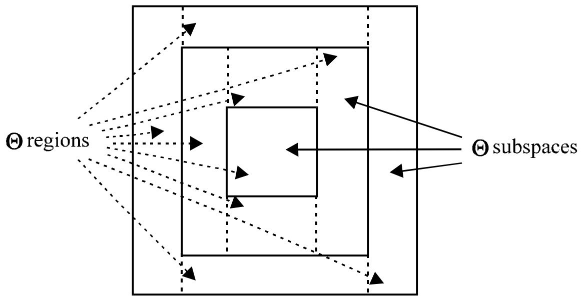


Fig. 2: Example of Θ Space Partition

the space inside one and outside its immediately enclosed box. The resulting rectangular region is called a Γ region. The remaining part of the Γ subspace is further divided into two Γ regions by a 2-dimensional plane that lies on the second side of the inner box. The dividing planes can be selected in accord with any numbering of dimensions.

In general, a d-dimensional universe is statically partitioned by several nested hyper-rectangles (NHRs), which we also call partition generators. Except for the outermost generator, which corresponds to the entire universe, every generator is enclosed by a larger one. The number of generators and their coordinates are selected statically. Except for the innermost subspace, each Γ subspace is further divided into d rectangular Γ regions, by means of d-1 hyper-planes lying on the outer sides of its inner generator. Unless some regions are "trivial", there are exactly $N = 1 + (m - 1) \cdot d$ Γ regions in the space, where m is the number of generators.

Θ partitioning is similar to Γ, except that the low endpoints of the nested generators can appear anywhere in the space, not just in the origin. As illustrated in Fig. 2, this strategy carves out the opposite sides of each generator along individual dimensions, starting from the first dimension and proceeding further in the pre-determined order of dimensions. With m generators, the number of resulting Θ regions is $N = 1 + 2 \cdot (m - 1) \cdot d$, unless some of the regions are trivial. Observe that the Θ partitioning strategy is just a generalization of Γ, in which the nested generators are allowed to float in the space.

Given a set of m generators, the coordinates of regions

```
INPUT:
        m;                                      // number of nested generators
        d;                                      // data dimensionality
        Gen [1..m; low..high; 1..d];   // coordinates of all generators (must be nested and non-trivial)
OUTPUT:
        Reg [1..N; low.high; 1..d];    // coordinates of all regions in the space partition
METHOD:
        Reg[1] := Gen[1];              // Gen[1] is the innermost generator
        k := 2;                                 // temporary variable; counts the index regions
        for i := 2 to m do
                temp1 := temp := Gen[i];        // for each generator, initialize the temporary variables
                for j := 1 to d do
                                // calculate the region on the low end of the current generator along axis j
                        temp[high,j] := Gen[i-1,low,j]; Reg[k] := temp;
                                // NOTE: in a Gamma partition, the region is trivial and will be ignored
                        if Reg[k,low,j] ≠ Reg[k,high,j]          then k := k + 1;
                                // calculate the region on the high end of the current generator along axis j
                        temp1[low,j] := Gen[i-1,high,j]; Reg[k] := temp1;
                                // ignore the calculated region if it is trivial
                        if Reg[k,low,j] ≠ Reg[k, high, j] then k := k + 1;
                                // now, prepare for the next iteration
                        temp1[low,j] := temp[low,j] := Gen[i-1,low,j];
                        temp1[high,j] := temp[high,j] := Gen[i-1,high,j];
                end for
                temp[high,d] := Gen[i-1,low,d]; Reg[k] := temp;
                if ∀c=1..d Reg[k,low,c] ≠ Reg[k,high,c]          then k := k + 1;
                temp1[high,d] := Gen[i-1,high,d]; Reg[k] := temp1;
                if ∀c=1..d Reg[k,low,c] ≠ Reg[k,high,c]          then k := k + 1;
        end for
```

Fig. 3: Algorithm for Computing Regions in a Γ or Θ Space Partition

```
INPUT:
        m;                                      // number of nested generators
        d;                                      // data dimensionality
OUTPUT:
        Gen [1..m; low..high; 1..d];   // coordinates of all generators
METHOD:
        Gen[m,low] := <0, 0, …, 0>; // low endpoint of the universe (m-th generator)
        Gen[m,high] := <1, 1, …, 1>;           // high endpoint of the universe
        if Gamma partitioning                   // NOTE: all regions (either Gamma or Theta) will be non-trivial
                then    N := 1 + (m - 1) · d;
                else    N := 1 + 2 · (m - 1) · d;
        V := 1;                                 // volume of the universe
        Vr := V / N;                            // volume of each individual region
        for i := m-1 downto 1 do
                temp := Vr / V;
                for j := 1 to d do
                        Gen[i,high,j] := Gen[i+1,high,] - temp · (Gen[i+1,high,j] - Gen[i+1,low,j]);
                        if Gamma partitioning
                                then    Gen[i,low,j] := 0;
                                else    Gen[i,low,j] := Gen[i+1,low,j] + (Gen[i+1,high,j] - Gen[i,high,j]);
                        temp := temp · (Gen[i+1,high,j] - Gen[i+1, low, j]) / (Gen[i,high,j] - Gen[i,low,j]);
                end for
                V := (Gen[i,high,1] - Gen[i,low,1]) · … · (Gen[i,high,d] - Gen[i,low,d]);
        end for
```

Fig. 4: Algorithm for Computing Generators that Induce Γ or Θ Regions of Equal Size

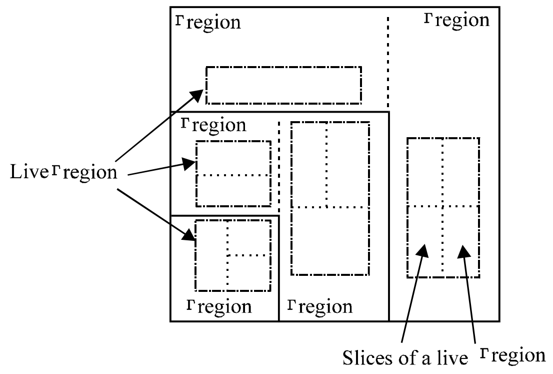


Fig. 5: An Illustration of Live Γ Regions and their Slicing

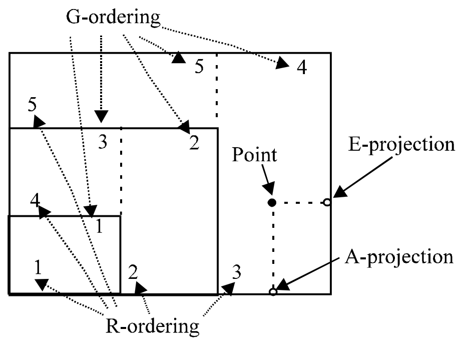


Fig. 6: Alternative Schemes for Γ Regions Numbering and Projection

in a Γ or Θ space partition can be calculated using the generalized algorithm of Fig. 3. Our assumption for the Γ space partition is that the low endpoint of each generator lies in the origin of the space, whose coordinates are 0. The procedure starts from the innermost generator, which becomes a region by itself. Then it goes into a loop which, for every generator, calculates the regions lying in the Γ or Θ subspace formed by this and the immediately enclosed NHR. Depending on whether the given generators induce a Γ or Θ space partition, this will create at most d or 2d regions in the subspace, respectively.

The algorithm is designed so that all trivial index regions are eliminated. By trivial, we mean a rectangular region whose low and high endpoint along a certain dimension are the same. In the d-dimensional space, such a region appears as just a hyper-plane of fewer than d dimensions. Since the points of that region also lie on the boundaries of one or more non-trivial regions, the trivial region can be safely eliminated. Provided the generators are themselves non-trivial as well as nested, the test for trivial regions is fairly simple (the algorithm of Fig. 3).

Figure 4 gives the algorithm calculating the coordinates of generators that induce a space partition into Γ (Θ) regions of equal size. For simplicity, the algorithm assumes a normalized d-dimensional universe $[0,1]^d$. The procedure starts with the outermost generator and, based on the coordinates of the given generator and the calculated volume Vr of a single Γ (Θ) region, computes the coordinates of the enclosed generators in an iterative fashion.

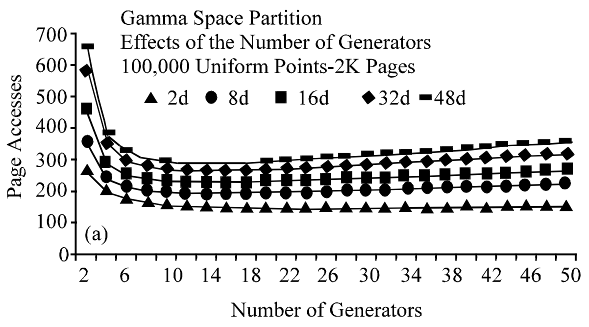
Even though Γ partitioning can be regarded as a special case of Θ, when the space partitions are induced by the algorithm of Fig. 4, Γ and Θ become two different partitioning strategies with distinct properties. As we will later see, Θ is better suited for more uniform data distributions, but Γ tends to be more appropriate for highly skewed data in heavily sparse spaces. Different types of queries may also favor one or the other partitioning strategy.

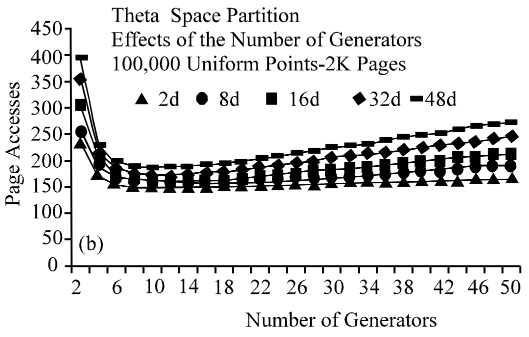
When the data are skewed, large portions of the given space are typically empty (contain no objects). Therefore, the canonical Γ and Θ space partitions as described above would incur a significant amount of dead space. In this regard, they would be no different than the space partition of the Pyramid Technique. However, in contrast to the Pyramid Technique, the rectilinear Γ and Θ partitioning enables a relatively simple way of addressing the problem.

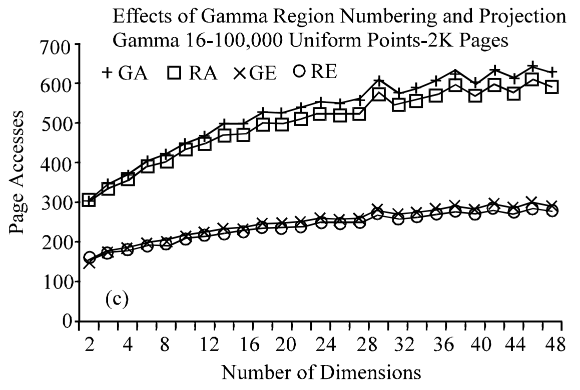
In order to eliminate from inspection a potentially significant amount of dead space, for each Γ or Θ region, one should dynamically maintain the minimum bounding hyper-rectangle enclosing all points that fall in the region. We call this the live region. Depending on the data distribution, one may also want to partition every region along different dimensions into, possibly, several slices. Assuming a static data set, this can be done using a rectilinear division of each live region along certain dimensions to obtain a desired number of slices in proportion to the number of points falling in the Γ or Θ region. Figure 5 illustrates the live regions and their slicing in a Γ partitioned 2-dimensional space. If the data set is dynamic, slicing must be performed on Γ or Θ regions rather than their live portions.

Region-Preserving Space Transformations: A multi-dimensional retrieval structure must be equipped to do more than just the partitioning of space. How the structure maps multi-dimensional data to locations in storage is also an important issue. Since our focus here is on the effects of the partitioning strategy in high-dimensional spaces, we deliberately choose an organization that decouples the space partitioning from the storage aspects of the retrieval scheme. For the purposes of this study, we will ignore the third important aspect of access structures for high-dimensional data, which is data compression [8].

As in the Pyramid Technique, the decoupling of the space partitioning and storage concerns is achieved through a form of region-preserving space transformation (RPST), which maps regions and queries in the multi-dimensional space onto the segments of a linear (one-dimensional) space.







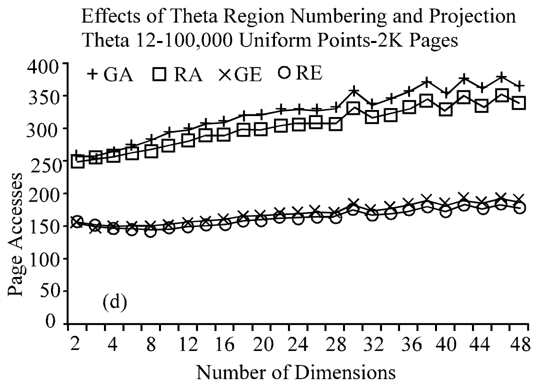
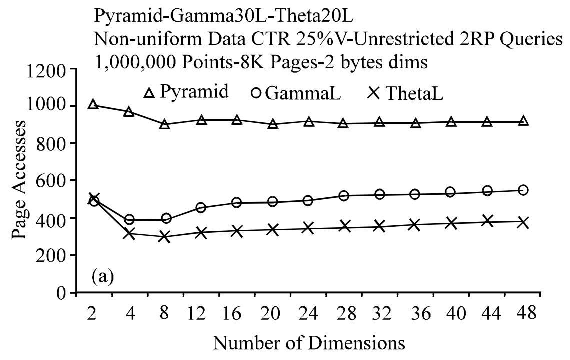
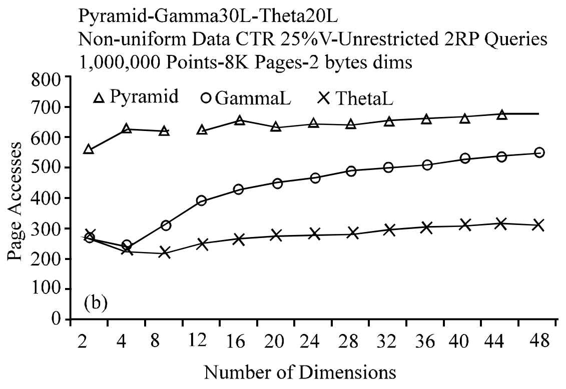
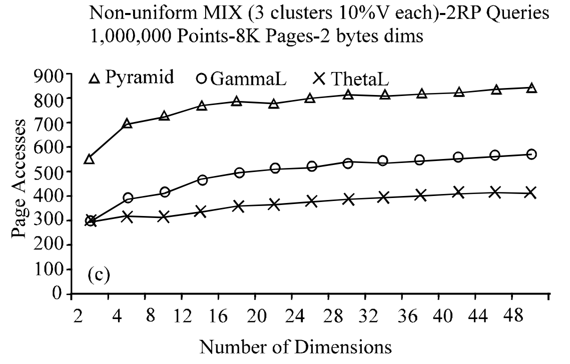


Fig. 7: The Effects of Various Parameters on the Performance of Γ_s and Θ_s Techniques







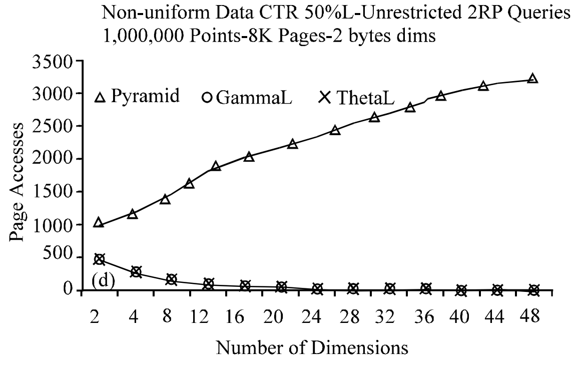


Fig. 8: Observed Performance of the Γ_s, Θ_s and Pyramid Techniques for Simulated Data Distributions

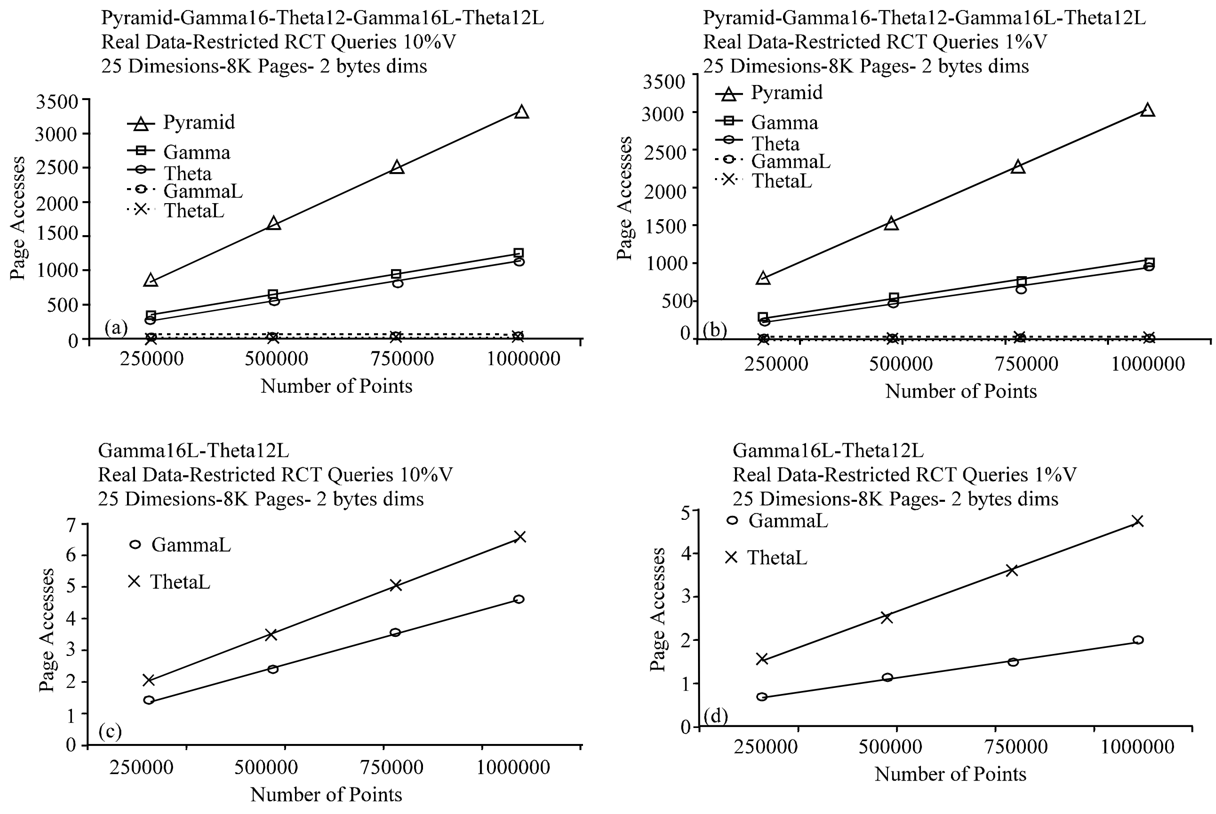


Fig. 9: Observed Performance of the Γ_s, Θ_s and Pyramid Techniques on a Set of Real Data

Generalizing this idea, we derive an entire class of RPST retrieval schemes. Virtually the only thing that distinguishes individual structures in this class is the space-partitioning strategy. In the rest of the study, we restrict our attention to the Pyramid Technique and two other RPST schemes, called the Γ_S and Θ_S Techniques, which are based on Γ and Θ partitioning.

Conceptually, each RPST scheme employs two distinct layers. The higher layer statically partitions the given space into a certain number of index regions (or slices, if region slicing is applied), whose descriptors are organized into a list maintained in main memory. This layer also performs an explicit transformation of points and queries onto their one-dimensional counterparts. The lower layer organizes the resulting key values into a regular B^+-tree structure. The index is searched using the one-dimensional segments generated by the query transformation.

As in the Pyramid Technique, the points of every index region (slice) are projected onto a selected dimension (projection axis). In the Pyramid Technique, the central line of the pyramid connecting its top with the center of its base serves as the projection axis. In Γ_S and Θ_S, the projection axis is one of the sides of the given region (slice). In all cases, the position of a point in the linear space is determined by the unique number of the region (slice) containing the point and the projection of the point along the selected axis. The two numbers form an index key, which is inserted into the B^+-tree along with the original multi-dimensional point. Note that the index may implicitly partition every region along the projection axis into possibly several segments, each of which corresponds to a leaf page of the underlying B^+-tree.

The numbering of regions and the choice of the projection axis may influence the performance of an RPST scheme. Assuming the Γ space partition, potentially viable alternatives are illustrated in Figure 6. Analogous numbering and projection schemes can be used with Θ partition. Figure 6 assumes that x is the first and y the second dimension.

With respect to the numbering of index regions, one can distinguish generator-wise ordering (G-ordering), in which all regions of a generator have consecutive numbers, from region-wise ordering (R-ordering), in which the corresponding regions of all generators have consecutive numbers. In Figure 6, the numbers appearing in the lower left corners of the Γ regions correspond to the R-ordering scheme. With regard to the choice of the projection axis for each region, one can distinguish edge-wise projection (E-projection), in which the points are projected onto the first longest edge of the region, from axis-wise projection (A-projection), in which the points are projected onto the dimension whose division resulted in the creation of the given region. If live regions and their slicing are applied, each slice can be projected in an analogous fashion. All slices of a single region, if any, are assigned consecutive numbers.

The search procedure must first determine the regions (or perhaps slices, if slicing is applied) that overlap the query. For each such region (slice), the procedure must construct the interval of the query window that overlaps the region (slice) along its projection axis. These intervals are used to search the underlying B^+-tree. The visited leaf pages represent the segments of the projected regions (slices) that overlap the query.

The above logic can be implemented in two ways. The first option uses the standard B^+-tree interfaces FetchKey and GetNext, and requires no modification of the B^+-tree code. For each interval of the linear space produced by the query transformation, the procedure performs a range search through the B^+-tree using the low and high endpoint of the interval as the fetch and stop point, respectively. However, since processing a single query may require multiple accesses to the same index page, this arrangement leads to more page accesses than necessary.

The second implementation option solves the later problem, but requires a modification of the existing B^+-tree code. The goal is to process each query in the manner of the KDB-tree and R-tree variants, making sure that no index page is accessed more than once. The procedure scans the root page with all intervals produced by the query transformation, identifying all pages at the level below that need to be accessed. If these are interior pages, they are processed in the same way. Whenever a leaf page is accessed, the procedure selects all resident points that fall within the given query. The rest of the procedure involves some simple optimizations designed to reduce the computational overhead of scanning an index page.

The RPST schemes have one important advantage in high-dimensional spaces. Since the transformation produces fairly short index keys whose size is fixed for all dimensionalities, the underlying B^+-tree has few pages in the interior levels. This, in turn, contributes to the overall reduction of page accesses per query. Unfortunately, the clear separation of space-partitioning and storage concerns does not come without certain problems. As a result of the static space partition, the dynamic changes in the volume and distribution of data may require a re-configuration of the space and the rebuilding of the B^+-tree. Moreover, the transformation of data and queries onto their one-dimensional counterparts incurs a loss of information that can result in an increased number of page accesses at the leaf level of the B^+-tree.

However, Γ_S and Θ_S have several advantages over the Pyramid Technique. Since the number and coordinates of NHRs are independent of data dimensionality and can be selected in accord with the actual data distribution, Γ and Θ partitioning strategies are more flexible than the Pyramid Technique. They also allow an effective way of dealing with the problem of dead space. While these schemes do not completely eliminate other problems of the Pyramid Technique, with the appropriate space partitions, the magnitudes of these problems are significantly reduced. In particular, Γ and Θ space partitions create many smaller index regions, which minimize implicit enlargement of queries that is rather severe with the Pyramid Technique.

EXPERIMENTAL RESULTS

An extensive set of experiments was performed to compare the retrieval performance of the Γ, Θ_S and Pyramid Techniques in different scenarios. The experiments were conducted over both simulated and real data sets of different size and data dimensionality, and for different types of queries. Except for the space-partitioning strategy, the three techniques were implemented in the same way. Each leaf-level entry of the underlying B^+-tree contained a fixed-size key value and the multi-dimensional point. The interior entries represented <key, pointer> pairs, where pointer was a 4-byte value indicating a lower-level node. In order to guard against the possibility of identical key values whose number exceeds the page capacity, to each index key, we appended a unique 4-byte ID of the data point. The experiments were conducted on three Pentium PCs, each with a single (either 1GHz or 1.5GHz) CPU and a SCSI disk drive.

Configuring the GammaS and ThetaS Structures: The first set of experiments was conducted to observe the effects of various parameters on the retrieval performance of Γ_S and Θ_S Techniques. In these experiments, every coordinate of a point was represented as a 4-byte integer and the page size was set to 2K bytes. Data dimensionality was varied between 2 and 48. Each space was configured using the algorithm of Figure 4, which induced Γ or Θ regions of equal size. Then, each d-dimensional structure with 100,000 uniformly distributed points was searched with 1,000 region queries. The query generation derived each side of a query window from a pair of two random points (2RP queries).

Figures 7a and b show the effects of the number of generators on the performance of Γ_S and Θ_S Techniques, respectively. In all cases, the worst performance was observed with only 2 generators. Overall, the best performance of Γ_S and Θ_S was observed with about 16 and 12 generators, respectively.

Since Θ partitioning strategy creates almost twice as many index regions as Γ, the Θ_S technique is more sensitive to the variances in the number of partition generators than Γ_S. However, as page capacity increased, both Γ_S and Θ_S became less sensitive to the actual configuration of space.

Figures 7c and d show the performance of the two techniques with alternative region numbering and projection schemes. In the Fig. 7, G and R stand for G- and R-ordering, while E and A stand for E- and A-projection, respectively (Fig 6). Thus, GA denotes a technique with generator-wise numbering and axis-wise projection, whereas RE denotes a technique with region-wise numbering and edge-wise projection. While the region numbering had little impact on the performance of either technique (in the experiments, R-ordering was somewhat better than G-ordering), the impact of the projection scheme was significant. In high-dimensional spaces, the performance of the two techniques with the edge-wise projection was about twice as good as with the axis-wise projection. In other experiments presented in this study, we adopted R-ordering and edge-wise projection for both the Γ_S and Θ_S Technique.

Synthetic Data: Figure 8 compares the performance of the Γ_S, Θ_S and Pyramid Techniques for four synthetic data distributions and 1,000 2RP queries. Each d-dimensional structure with 8K pages had 1,000,000 points with 2-byte coordinates. As before, dimensionality was varied between 2 and 48. The Γ and Θ space partitions with live regions and their slicing were induced with 30 and 20 generators, respectively. The number Nr of slices for each Γ or Θ region r was calculated as Nr = max{1, $\lfloor n_r/n_a \rfloor$, where n_r and n_a were the number of points falling in the region r and the average number of points per region, respectively (Nr = 1 means no slicing of the region r).

Figures 8a-c show the observed average number of accessed pages per 2RP query for some “mildly” skewed data distributions consisting of: (a) one cluster (fixed 25% volume of the space) placed in the center of the universe; (b) one cluster (fixed 25% volume) placed in the origin; and (c) three clusters (10% volume each) placed in the origin, center, and the corner of the space that contains the highest value along the first axis. In each scenario, Γ_S and Θ_S outperformed the Pyramid Technique.

In the above scenarios, the impact of live Γ and Θ regions was relatively minor. But, Fig. 8d demonstrates their effectiveness for a heavily skewed data distribution. All points were placed in a hyper-cube centered in the middle of the space, whose each side was restricted to exactly 50% of the corresponding side of the universe. Relative to the entire space, the volume of the hyper-cube reduced rapidly as data dimensionality increased. As a result, the Pyramid Technique incurred a significant amount of dead space.

Real Data: Perhaps the most instructive are the results of our experiments with real data. The data set represented a table with 1,000,000 records, which was extrapolated from a database of a local company. As before, the page size was 8K bytes.

In these experiments, we measured the performance of the Γ_S and Θ_S Techniques both with and without live regions and their slicing. Γ and Θ space partitions were obtained using the algorithm of Figures 4 with 16 and 12 generators, respectively. No attempt was made to optimize the space partitions for the given data distribution. The performance of every structure (the average number of accessed pages per query) was measured for data sets with 1/4M, 2/4M, 3/4M and 1M points and for two types of queries (1,000 queries of each type) with randomly chosen center. The queries were relatively small and restricted to at most 10% and 1% of the total space, respectively.

Figures 9a and b show the results. Even without live regions and slicing, Γ_S and Θ_S outperformed the Pyramid Technique. However, with live regions and slicing, the performance improvements over the Pyramid Technique were much more significant. Since the performance curves of Γ_S and Θ_S with live regions and slicing appear to lie on the horizontal axes of the graphs, Figures 9c and 9d show how these curves actually look like.

DISCUSSION

In this study, two novel partitioning strategies, called Γ and Θ were developed. Each of these strategies applies an asymmetric subdivision of individual dimensions, making sure that every axis is divided several times. As a result, every dimension of data can effectively contribute to the search process. Just like the Pyramid Technique, these strategies avoid both the exhaustive search and region overlap. However, unlike the Pyramid Technique, the Γ and Θ partitioning strategies allow highly configurable partitions of space that can fit the actual data distribution. They also enable effective ways of dealing with the other problems of the Pyramid Technique, which include the loss of proximity, the enlarged queries, and dead space.

The proposed partitioning strategies were used to develop two new region-preserving space transformation schemes for indexing high-dimensional data, called the Γ_S and Θ_S Techniques. By reusing an exact-match indexing mechanism along with its concurrency and recovery features, Γ_S and Θ_S enable relatively simple integration of advanced multi-dimensional capabilities in complex transactional environments.

The experimental evidence, gathered on both simulated and real data sets, demonstrates the superiority of Γ_S and Θ_S over the Pyramid Technique, which also uses static partitioning. As data distribution becomes more skewed, the performance improvements over the Pyramid Technique become more pronounced. Θ partitioning is generally better than Γ for more uniform data distributions and when queries tend to clutter around the middle of the space. Γ partitioning tends to be more appropriate in special scenarios that frequently appear in practice.

The proposed partitioning strategies also have direct application in supporting similarity (k-nearest neighbor) searches. Bit-sliced indexes, such as the VA-file [9], are often used for this purpose. However, these indexes employ a grid-like space partition into rectangular cells, whose number grows exponentially with data dimensionality. Since the Γ and Θ partitioning strategies require a limited number of divisions to split each axes multiple times, they can enable a faster and more accurate process of similarity searching with a more compact indexing structure than the VA-file. Other applications of Γ and Θ partitioning include retrieval of data on tertiary storage [18] and clustering large sets of high-dimensional data. In the later context, they can replace typical grid-like space partitions frequently used in contemporary clustering algorithms.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. IIS-0312266

REFERENCES

1. Gaede, V. and O. Gunther, 1998. Multidimensional access methods. ACM Computing Surveys, 30: 170-231.
2. Berchtold, S., C. Bohm and H.P. Kriegel, 1998. The pyramid-technique: Towards breaking the curse of dimensionality. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp: 142-153.
3. Boehm, C., S. Berchtold and D.A. Keim, 2001. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Computing Surveys, 33: 322-373.
4. Chakrabarti, K. and S. Mehrotra, 1999. The hybrid tree: an index structure for high dimensional feature spaces. Proc. 15th Intl. Conf. on Data Engineering, pp: 440-447.
5. Fagin, R., R. Kumar and D. Sivakumar, 2003. Efficient similarity search and classification via rank aggregation. Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp: 301-312.
6. Lin, K., H. Jagadish and C. Faloutsos, 1995. The TV-tree: An index structure for high-dimensional data. VLDB J., 3: 517-542.
7. Sakurai, Y., M. Yoshikawa, S. Uemura and H. Kojima, 2000. The A-tree: An index structure for high-dimensional spaces using relative approximation. Proc. 26th Intl. Conf. on Very Large Data Bases, pp: 516-526.
8. Weber, R., H.J. Schek and S. Blott, 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. Proc. 24th Intl. Conf. on Very Large Data Bases, pp: 194-205.
9. Orlandic, R., and B. Yu, 2002. A retrieval technique for high-dimensional data and partially specified queries. Data and Knowledge Engineering, 42: 1-21.
10. Robinson, J.T., 1981. The K-D-B tree: A search structure for large multidimensional dynamic indexes. Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp: 10-18.
11. Ramsak, F., V. Markl, R. Fenk, M. Zirkel, K. Elhardt and R. Bayer, 2000. Integrating the UB-tree into a database system kernel. Proc. 26th Intl. Conf. on Very Large Data Bases, pp: 263-272.
12. Aggarwal, C.C., 2001. On the effects of dimensionality reduction on high dimensional similarity search. Proc. 20th ACM Symposium on Principles of Database Systems PODS'2001, pp: 256-266.
13. Ravi Kanth, K.V., D. Agrawal and A. Singh, 1998. Dimensionality reduction for similarity search in dynamic databases. Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp: 166-176.
14. Berchtold, S., D.A. Keim and H.P. Kriegel, 1996. The X-tree: an index structure for high-dimensional data. Proc. 22nd Intl. Conf. on Very Large Data Bases, pp: 28-39.
15. Comer, D., 1979. The ubiquitous B-tree. ACM Computing Surveys, 11: 121-137.
16. Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, pp: 47-54.
17. Orlandic, R., J. Lukaszuk and C. Swietlik, 2002. The design of a retrieval technique for high-dimensional data on tertiary storage. SIGMOD Record, 31: 15-21.