D*: A Data Storage and Retrieval System for Scientific Studies

Ratko Orlandic

Department of Computer Science
University of Illinois at Springfield
One University Plaza, UHB 3100
Springfield, IL 62703, USA
rorla2@uis.edu

Sachin Kulkarni

Department of Computer Science
Illinois Institute of Technology
10 West 31st Street, SB 236
Chicago, IL 60616, USA
kulksac@iit.edu

Abstract

D* is a novel system for data storage and retrieval appropriate for advanced scientific studies, as in high-energy physics, environmental sciences, and astronomy. The design of the D* system is based on certain principles of organizing and accessing multi-dimensional data on storage, whose pursuit requires that the storage system acquire a greater knowledge about the data. This provides a basis for a tighter integration of data storage and data mining technologies. Through the application of innovative retrieval and clustering techniques for high-dimensional data, D* can support high-performance data access and provide data mining applications useful insights into the data that can facilitate subsequent processes of data preparation and data mining. The basic processes of the D* system include data clustering, space partitioning, data loading, and data retrieval based on region queries and similarity searching. D* scales well with increasing data dimensionality and works well on incremental load of data.

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

1 Introduction

In contemporary data mining systems, data is usually stored in flat files and its analysis tends to be sequential. This is because contemporary storage systems do not contribute much to the process of data mining. The prototype system described in this paper, which is called D* (Data STorage And Retrieval), is a useful paradigm one can use to develop future storage systems that contribute to the data mining process. The goal is not to include everything that analysis needs into the storage system, but that in the process of achieving its primary objective (i.e., the fast and scalable access to the data), storage system can assist data analysis.

Laying out data on storage in a way that facilitates data analysis increases the performance and scalability of data mining tasks. Moving analytical capabilities down to the storage system streamlines the data mining process, reduces unnecessary redundancies, and enables hot analysis as new data is arriving. Since storage system can have a more complete view of data, it is in a better position than data mining applications to gain and provide a fast and accurate insight into data distribution. In turn, gaining an accurate insight into data distribution is an essential aspect of many data preprocessing and data mining tasks.

Basic processes of D* include data clustering, space partitioning, data loading, and data retrieval based on region and *k* nearest-neighbor (kNN) queries. To support these processes, the system employs a new multi-dimensional indexing technique, an access method for similarity searching, an efficient and scalable clustering algorithm, and two methods of deriving a space partition. The techniques are designed to operate in high-dimensional spaces without dimensionality reduction.

The D* system, and much of its constituent technology, is based on two design principles, called the principles of clustering and cluster representation. In the existing literature, these principles have been stated implicitly and in ways that are not particularly useful for the design of access methods for data in high-dimensional spaces. In fact, a vast majority of these access methods were not designed with these principles in mind. Because we differentiate the two principles and derive them formally, we believe that our formulations of these principles are accurate statements of important goals around which multi-dimensional access methods and multi-dimensional storage systems should be organized.

An important observation behind the D* system is that, in the pursuit of the principles of clustering and cluster representation, the storage system must acquire a greater knowledge about the data. This means that the storage system must become more intelligent in that it itself must incorporate elements of data mining. This, in turn, provides a basis for a tighter integration of data storage and data mining technologies.

The D* system is not meant to be a real analytical storage system; only a paradigm one could use to develop such systems. D* is not meant to be a data mining application either. It is a prototype system for storing and retrieving multi-dimensional data which also provides useful insights into the data that can facilitate analysis. However, even as it is, D* can be used to perform sophisticated analytical tasks. With more extensive data mining capabilities, which can easily be incorporated into the system, D* can evolve into a genuine data mining system.

For example, in addition to data clustering supported by D*, a simple extension of the system can support instance-based (kNN) data classification. Data pre-processing is another area

where the system can provide significant support. A relatively simple modification of our data-sensitive space partitioning described later in this paper can enable the system to quantify the discriminative power of every data dimension. This, in turn, can be used for efficient attribute relevance ranking and dimensionality reduction. Other extensions of the system can support imputation of missing values, data smoothing, and stratified sampling.

Since this technical document is intended to provide a complete description of the technical aspects of the D* system, we repeat here some information and algorithms already published in [Kul06], [Luk04], [Orl05], or [Orl06]. In the rest of the document, Section 2 summarizes related work. Section 3 formulates the design principles around which the system is organized. Section 4 describes the general architecture of the D* system. Section 5 describes one of the partitioning strategies applied by the system, called "data-blind" space partitioning. Section 6 describes the main aspects of the clustering technique used by the system. Section 7 describes the process of data-sensitive space partitioning. Section 8 gives the algorithms of similarity and region searching. Section 9 presents experimental evidence. Section 10 summarizes the paper and discusses broader application of the technology described in this paper.

2 Related Work

High-dimensional data pose major challenges to many advanced applications, including scientific data analysis. An important aspect of the general problem of data dimensionality is that the performance of traditional multi-dimensional access methods rapidly deteriorates as dimensionality grows. Yet despite a considerable interest in the problem of accessing data in high-dimensional spaces [Boh01], we still know little about the basic parameters governing the retrieval performance in these situations.

Early approaches to the problem of accessing data in high-dimensional spaces focused on the fact that, as the number of dimensions increases, both the size of index entries and the storage overhead of multi-dimensional structures grow accordingly. For example, the TV-tree [Lin95] pursued the idea of selecting a relatively small number of "active" dimensions that may participate in the process of node splitting. By ignoring the rest of the dimensions, the TV-tree reduces the size of the indexing structure, which generally has positive effect on retrieval performance. Many other access methods for high-dimensional data also apply dimensionality reduction [Agg01, Fag03, Rav98, Yu04].

Several high-dimensional indexing techniques arose from the observation that the strategy of partitioning the space may have a profound effect on the retrieval performance in high-dimensional situations [Ber98, Cha99, Kat97, Sac00, Whi96]. For example, to improve the performance in high-dimensional spaces, the Pyramid Technique [Ber98] statically partitions the D-dimensional space into 2D pyramids that meet at the center of the universe. The Hybrid-tree [Cha99] follows a different approach. As long as the splits of index pages do not require downward propagation, the structure uses the space partition of KDB-trees into non-overlapping regions [Rob81]. In order to prevent downward cascading splits, it allows certain amount of overlap (space covered by more than one index region) between the index regions [Cha99], as in the R-tree variants [Bec90, Gut84]. The X-tree [Ber96] tries to eliminate region overlap, whose negative effects are more pronounced in high-dimensional spaces.

The diversity of the proposed techniques suggests that, in high-dimensional spaces, there are numerous parameters of good retrieval performance one can pursue. We approach the design of the D* system in a systematic way, guided by two important design goals: principle of clustering and principle of cluster representation. As we will see in Section 3, the first design principle deals with the grouping of multi-dimensional data in index pages, whereas the second deals with their representation within the indices. The pursuit of these principles can result in significant improvements of retrieval performance [Orl06]. The principles of clustering and cluster representation have been pursued in some contemporary multi-dimensional access methods. However, in the existing literature [Bec90, Cia97], they have not been differentiated. Moreover, they have generally been stated implicitly and in ways that are not particularly useful for the design of access methods for high-dimensional feature spaces.

A vast majority of access methods was not designed with these principles in mind. Most indexing techniques based on bit slicing, vertical partitioning, and dimensionality reduction [deV02, Lin95, Rav98, Wu04] are not concerned with the quality of data clustering. Since typical access methods based on space partitioning [Cha99, Gut84, Rob81] split an index region along a selected dimension when a certain page overfills, the index regions cannot be split along all dimensions of a high-dimensional space. As a result, these methods have many of the same limitations as those that apply dimensionality reduction. M-trees [Cia97, Sex04] adhere to these principles, but their construction relies on expensive distance computations and incurs potentially large region overlap. To improve retrieval performance, new access methods with explicit clustering have emerged [Cha00]. Unfortunately, the clustering techniques used for this purpose do not scale well with a growing data dimensionality.

Similarity searching in high-dimensional spaces is an important research area. Real high-dimensional data are often clustered and data tends to occupy only a small fraction of the space [Blo03]. An appropriate search method must be aware of the locality of data in high dimensions. However, most methods used to find the locality of data rely on dimensionality reduction. Unless a multi-step approach is applied [Sei98], this leads to approximate results.

Due to high computational cost, finding approximate rather than exact solutions is the most popular approach to k-nearest neighbor searching. In order to tackle the “curse of dimensionality”, various approximate solutions based on dimensionality reduction have been proposed [Agg02, Fag03, Gio99]. [Agg02] emphasized the need to distinguish between the localities in the data and introduced a concept of locality sensitive subspace sampling. The concept of locality sensitive hashing (LSH) is developed in [Gio99]. The emphasis in [Agg02] and [Gio99] is on finding the locality of the data in a way that makes the process of dimensionality reduction more “data aware”. The focus is on local rather than global distribution of data.

Significant effort in finding the exact nearest neighbors has yielded limited success. The SR-Tree [Kat97] uses both a hyper-sphere and hyper-rectangle to represent a region and improves search efficiency over the SS-tree and R-tree. However, as reported in [Bey99], the SR-Tree is at par with sequential scan when dimensionality is at least 20. The VA-File [Blo97, Web98] applies a filter to the sequential scan using the concept of vector approximations. The bit-encoded approximations guide the elimination of points during the search process. A-tree [Sak00] and i-distance [Yu01, Yu04] are reported to work well in high dimensions. The A-tree stores virtual bounding rectangles that approximate minimum bounding rectangles. i-distance uses space partitioning to separate data into different regions. However, the data of each region are transformed into a one-dimensional space in which the similarity is measured.

Our quest is for a solution with an efficient and scalable search, acceptable data-loading time, and the ability to work on incremental loads of data. Despite considerable work done in the area, this formulation of the problem of exact similarity and region searching in high-dimensional spaces is still an intriguing one. We introduce a new way of arranging data on storage to facilitate efficient search. A new space partitioning method is proposed along with a new access method for high-dimensional data. The basic idea is to separate clusters in the data and eliminate searching over empty space.

The work described in this paper is also related to the ongoing efforts on developing scientific data grids [Che01, Fos01, Hos00]. The development of these information-technology frameworks for scientific research is one of the most important undertakings in computer sciences today. The main objective of these endeavors is to enable "geographically dispersed extraction of complex scientific information from very large collections of measured data" [Ave01].

Scientific data grids face difficult problems of scale arising from the large volumes and high dimensionalities of scientific data. To support efficient and scalable retrieval of multi-dimensional scientific data on a designated execution site, the storage and retrieval system should employ a clustered storage organization that maximizes the densities of clusters on storage [Orl03]. The clustering algorithm appropriate for this application should effectively isolate areas with points, reducing the amount of their internal empty space. Furthermore, the clustering method should scale in terms of the number of points and number of dimensions.

The performance of the grid operation has a significant bearing on the success of scientific data grids. Since much of this performance depends on how fast one can find the desired items in the large data repositories, an appropriate storage organization is critical for the success of the data grids. Several research efforts, directly related to the data grid initiative, deal with the problem of managing massive data repositories [Kur01, She02, Sho99]. Yet despite these efforts, the problem of storage organization is not sufficiently addressed.

3 Design Principles

In the following, let us use the term *storage cluster* to denote the spatial region formed by points in a storage unit, which we assume to be an index page. The basic design principle underlying our approach states that: *multi-dimensional data must be grouped on storage in a way that minimizes the extensions of storage clusters along all relevant dimensions and achieves high storage utilization.* We call this the *principle of clustering* data on storage. Formal derivation of this principle appears in [Kul06].

The term "relevant dimensions" refers to the fact that multi-dimensional queries may have "affinity" for certain dimensions, consistently leaving other dimensions unrestricted. However, if necessary, the clustering scheme used for storage organization must treat all data dimensions as equally important. Assuming a multi-dimensional space defined by relevant dimensions, the stated principle implies that the storage organization must maximize the densities of storage clusters both by increasing the number of points in the clusters and by reducing their volumes. To increase the densities of storage clusters, the organization must reduce their internal empty space. For best results, the database system should employ a genuine clustering algorithm for this purpose.

The spaces occupied by storage clusters (in our case, data in index pages) are depicted in the index by structures we call *index regions*. Typically, these index regions are organized into a hierarchy, which is searched in a top-down fashion. An index region is accessed if the query overlaps it. Accessing an index region necessitates searching the contained index cluster(s). To minimize the number of index clusters searched, *index regions must be as tight and square as possible around the enclosed index clusters, and have little empty internal space*. We call these design constraints the *principle of cluster representation*.

The two principles are not independent. The degree to which the principle of cluster representation can be achieved is largely determined by the quality of clustering. On the other hand, without a good quality of cluster representation, the benefits of clustering data on storage are typically lost. However, the adherence to one principle does not imply the adherence to the other. Therefore, these principles are largely orthogonal design goals [Orl06].

We refer to the process of detecting dense areas (*dense cells*) in the space with minimum amounts of empty space as *data space reduction*. In this context, *data clustering* is a process of detecting the largest areas with this property, called *data clusters*. The principles of clustering and cluster representation can be achieved either by data clustering or by data space reduction only. However, a facility to do both is an advantage.

The adherence to the principles of clustering and cluster representation can facilitate various kinds of retrieval by enabling a close-to-optimal assignment of data to pages and a significant reduction of the search space even before the retrieval process hits persistent storage. For effective data space reduction, the clustering method should operate directly in the given space without dimensionality reduction, and it should not be governed by any expectation about the number of clusters. To be useful for storage organization, it must also be very efficient. This set of requirements motivates the design of the $\text{GARDEN}_{\text{HD}}$ clustering algorithm for high-dimensional datasets introduced in [Orl05] and the DSGP data-sensitive space partitioning technique described later in this paper.

4 The Architecture of the D* System

The processes of the D* system are illustrated in Figure 1. The clustering module produces a compact cluster representation of data. Operating on this representation, the partitioning module produces a data-sensitive Γ space partition. More details on Γ partitioning appear in Section 5. The derived partition is maintained by a light-weight in-memory structure, called the *Γ filter*. For the environments where explicit clustering of data is not possible or viable, D* provides an option of deriving the space partition with no insight into the data distribution. This data-blind partitioning produces a fixed number of regions with equal volume.

In the process of data loading, the Γ filter acts like a filter that channels the points of each region in the space partition into a separate KDB-tree index. Together, these indices represent clustered data storage. With the facility for incremental loading, the system can subsequently accept new data points through the existing space partition. The processes of data retrieval include both region and similarity-search queries, which undergo two levels of filtering—one in the memory-resident Γ filter and the other in the selected indices on disk.

In the D* system, Γ filter is used to impose a desirable behavior stated in our principles on traditional multi-dimensional access methods that normally do not exhibit this behavior in

high-dimensional spaces. It provides an inexpensive way of achieving high performance and scalability of multi-dimensional access, enabling effective reuse of existing multi-dimensional access methods. The KDB-tree indexing technique is not necessarily optimal for this environment. The R-tree [Gut84] would yield faster retrieval, but at the expense of slower insertions. In environments with frequent insertions, the later cost can be significant.

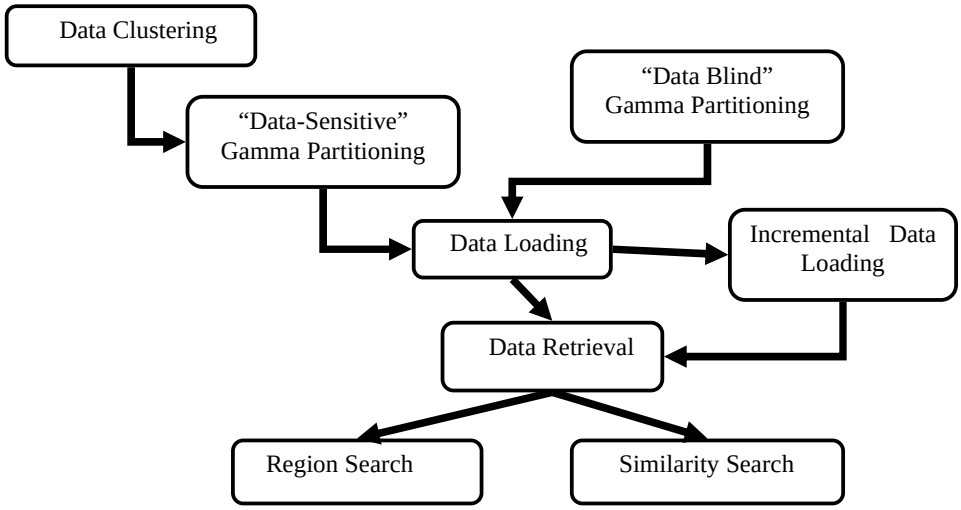


Figure 1. Key processes of the D* system.

Using Γ filter, D* achieves static pre-clustering of data on disk that tends to reduce the volumes of stoareg clusters. Γ partitioning provides the first level of insurance that in high-dimensional spaces most regions are restricted along all dimensions, as required by the principle of clustering. The memory-resident Γ filter helps eliminate from inspection potentially large empty space, as required by the principle of cluster representation. Then, by applying dynamic slicing of Γ regions using KDB-tree indices, the system produces dense index regions consistent with both of our principles.

5 Data-Blind Gamma Partitioning

The Γ partitioning technique, which is illustrated in Figure 2, was first introduced in [Orl02]. A D-dimensional feature space (universe U) is partitioned by several nested hyper-rectangles whose low endpoints lie in the origin of the space. The outermost hyper-rectangle is the space itself. We call these nested hyper-rectangles *partition generators*, or just *generators*. The space inside one generator and outside its immediately enclosed generator, if any, is called *Γ subspace*. We refer to this process of partitioning the space as Γ partitioning because, in a 2-dimensional space, these subspaces resemble the Greek letter Γ.

Except for the innermost subspace, each Γ subspace is divided into D non-overlapping rectangular *Γ regions* by means of D-1 hyper-planes lying on the upper boundaries of its inner generator. In Figure 2, these Γ regions are separated by dashed lines. With $m > 0$ generators (including the universe U), the total number of Γ regions is $NG = 1 + (m\text{-}1)\cdot D$. While Γ can accommodate arbitrary nested generators, data-blind partitioning option produces Γ regions of equal volume (data-sensitive space partitioning is described later in this paper).

In the actual representation of Γ partitioning (i.e., in the Γ filter), Γ regions are purely conceptual, i.e. their descriptors need not be stored. To describe a Γ space partition, one needs to store m D-dimensional points, called *generating vectors*, each of which represents the high endpoint of a generator. We assume that the first generating vector in the order is the high endpoint of the universe. These generating vectors are derived using the algorithm for partitioning the space into Γ regions of equal volume given in Figure 3. In order to compute each coordinate of every generating vector, this procedure examines two coordinates of the temporary region G and performs a small number of single-value assignments. The time complexity of the procedure is equivalent to O(m) comparisons of D-dimensional points.

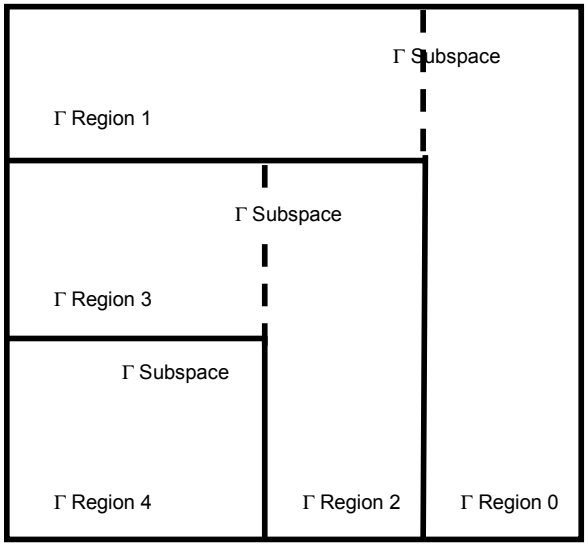


Figure 2. An example of Γ partitioning in a 2-dimensional space.

One can derive the descriptors (the high and low endpoint) of each Γ region from the given list of generating vectors. However, since both point and region (window) queries operate directly on the list of generating vectors, full descriptors of the Γ regions are not required during retrieval. This property of Γ partitioning facilitates fast in-memory processing of point and region queries, equivalent to O(m) comparisons of d-dimensional points, where m is the number of generators. For relatively small m, this amounts to O(1) point comparisons.

Figure 4 gives the algorithm that identifies the Γ region in which a given query point belongs. The algorithm exploits the fact that each Γ region $i \geq 0$ lies above the hyper-plane that separates it from the space in which the subsequent Γ regions $j > i$ lie (*separating hyper-plane* by which the Γ region is carved out from the space). The procedure iterates over the conceptual Γ regions in the order they are carved out from the space. The query point belongs to the first Γ region in the order whose separating hyper-plain lies below the x-th coordinate of the given point, where x is the dimension perpendicular to the separating hyper-plain of the Γ region. This is determined by comparing the coordinates of the point against the corresponding coordinates of the generating vectors. Thus, the time required to locate the Γ region in which a point belongs is O(m) point comparisons.

```
PROCEDURE DataBlindGammaPartitioning
INPUT
      m;                          // number of generators (including the universe U)
      D;                          // data dimensionality
      U [low..high; 0..D-1];      // coordinates of the universe (need not be a unit space)
 OUTPUT
      GVlist [0..m-1; 0..D-1];    // list of generating vectors
BEGIN
      NG := 1+(m-1)·D;            // number of Γ regions
      scale := 1/NG;              // scale for the first Γ region
      G := U;                     // current generator
      GVlist[0] := U[high];
      for i:=1 to m-1 do
            begin
            for j:=0 to D-1 do
                  begin
                  G[high,j] := G[low,j] + (G[high,j] – G[low,j])·(1–scale);
                  scale := scale/(1–scale)      // scale for the next Γ region
                  end
            GVlist[i] := G[high];
            end
END
```

Figure 3. Algorithm for splitting the space into Γ regions of equal volume.

```
PROCEDURE pointQueryFiltering
INPUT
      m;                          // the number of generators
      D;                          // data dimensionality
      Qpoint [0..D-1];            // given query point
      GVlist [0..m-1; 0..D-1];    // list of generating vectors
OUTPUT
      i;                          // number of the Γ region containing the given point
BEGIN
      NG := 1+(m-1)·D;            // number of Γ regions
      i := 0;                     // index of the first Γ region
      j  := (i div D) + 1;        // index of the corresponding generating vector
      x := i mod D;               // dimension along which the Γ region is carved out
      while i < NG-1 and Qpoint[x] < GVlist[j,x] do
         begin
         i := i+1; j := (i div D) + 1; x := i mod D;
         end
END
```

Figure 4. Algorithm that locates Γ region containing a point.

```
PROCEDURE regionQueryFiltering
INPUT
      m;                          // number of generators
      D;                          // data dimensionality
      Qwindow [high..low; 0..D];  // query window
      GVlist [0..m-1; 0..D];      // list of generating vectors
OUTPUT
      NR;                         // number of Γ regions intersecting the query window
      Gregions [0..K-1];          // list of Γ regions intersecting the query
BEGIN
      NG := 1+(m-1)·D;            // number of Γ regions
       k := 0;                    // counts Γ regions intersecting the query
       i := 0;                    // index of the first Γ region
       j := (i div D) + 1;        // index of the corresponding generating vector
       x := i mod D;              // dimension along which the Γ region is carved out
       while i < NG-1 and Qwindow[low,x] < GVlist[j,x] do
          begin
          if Qwindow[high,x] ≥ GVlist[j,x] then
              Gregions[k++] := i;
          i := i+1; j := (i div D) + 1; x := i mod D;
          end
      Gregions[k] := i; NR := k+1;
END
```

Figure 5. Algorithm that locates Γ regions which intersect a query window.

Figure 5 gives the algorithm that locates all Γ regions intersecting a query window, which is similar to the algorithm for point search. The procedure proceeds as if it were looking for the region *i* containing the low endpoint of the query window. For each region $j < i$, if any, it also verifies that its high endpoint lies above the separating hyper-plane of the Γ region *j*, in which case it intersects the query. Since this additional comparison requires only a comparison of two numeric values, the time complexity of the region search is O(*m*) point comparisons.

One can verify the correctness of both the point and region search on the example given in Figure 6. In particular, the point-search algorithm of Figure 4 finds that the high and low endpoints of the given query window lie within the Γ regions 2 and 4, respectively. Similarly, tracing the region-search procedure of Figure 5, one will examine all 5 Γ regions. However, since the high endpoint of the query window lies below the separating planes (in this example, lines) of the Γ regions 0 and 1, these regions do not intersect the query window, and they are eliminated from further inspection.

In the D* system, the Γ filter compactly represents the Γ space partition. During the insertions, for each Γ region, the Γ filter dynamically maintains its *live region*, i.e. the minimum bounding hyper-rectangle enclosing the points in the Γ region. During retrieval, these live regions help eliminate potentially significant empty space, which increases the system's compliance with the principle of cluster representation.

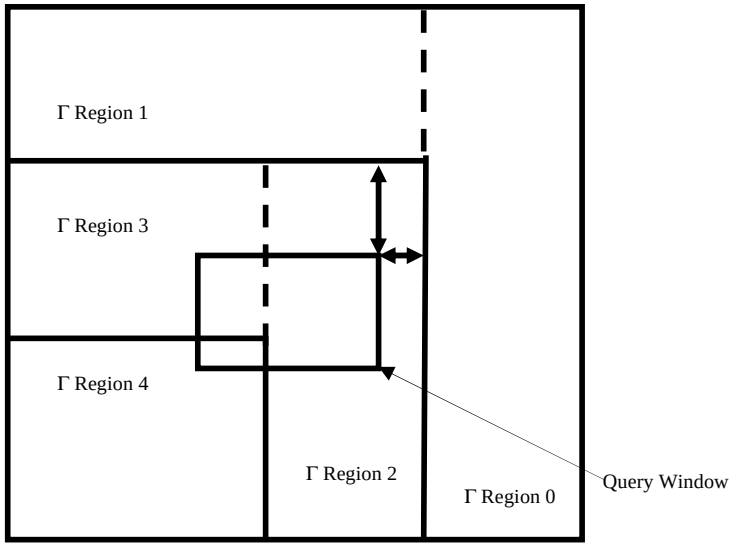


Figure 6. Illustration of a region search on a 2D Γ space partition.

6 Data Clustering

A frequently cited goal of clustering is to provide insight into data distribution. Unfortunately, contemporary clustering methods are not organized around this goal. They often break disjoint areas with points and merge their pieces into different clusters. Many clustering algorithms require prior knowledge about the number of clusters, which is often hard to determine in advance. However, inaccurate selection of this parameter can also lead to the segmentation of dense areas and aggregation of their parts into different clusters. Frequently cited problems of explicit or implicit dimensionality reduction, which many clustering methods perform or require, is the potential loss of clusters and the distortion of spatial properties and densities of clusters, which can heavily distort the sense of data distribution.

Our clustering technique, called $\text{GARDEN}_{\text{HD}}$ [Orl05], is designed to provide a fast and accurate insight into data distribution in order to facilitate data mining or retrieval. With an appropriately selected density threshold δ, which is the only required input parameter, the method runs in O(N*log*N) time, where N is the number of D-dimensional points in the data set. $\text{GARDEN}_{\text{HD}}$ can operate in high-dimensional spaces without dimensionality reduction.

The algorithm of $\text{GARDEN}_{\text{HD}}$ is given in [Orl05]. Here, we briefly describe the method and summarize its characteristics. $\text{GARDEN}_{\text{HD}}$ is a hybrid of cell- and density-based clustering that operates in two phases. Employing a recursive space partition using a variant of Γ partitioning, the first phase performs an efficient data space reduction, identifying rectangular cells whose density is above the user-defined threshold δ. In the second phase, the adjacent dense cells are merged into larger clusters. It is the application of Γ partitioning that enables the method to operate efficiently in high-dimensional spaces without dimensionality reduction.

Given a density threshold δ, the process of data space reduction starts by scanning the data once in order to compute the live region LRU enclosing all points in the universe. The technique proceeds by partitioning LRU into multiple Γ regions. Within each Γ region, its live

region is identified. Sparse live regions are recursively partitioned into smaller regions using Γ partitioning and live-region identification, until each of the smaller live regions has a density of at least δ, becoming a dense cell. The process produces a compact "signature" of data consisting of the identified dense cells, represented by their low and high endpoints.

The process of data space reduction is illustrated in Figure 7. Figure 7a, in which dark ovals represent areas populated by points, shows the Γ partition of the initial region LRU as well as the live regions within the resulting Γ regions. In Figures 7b and 7c, sparse live regions LR1 and LR3 are partitioned further until all their enclosed dense cells are identified. Whenever a high-density live region is detected (e.g., LR2 in Figure 7a), it is included into the set of dense cells. The resultant dense cells of this example are shown in Figure 7d. Then the adjacent dense cells are merged into larger clusters [Orl05].

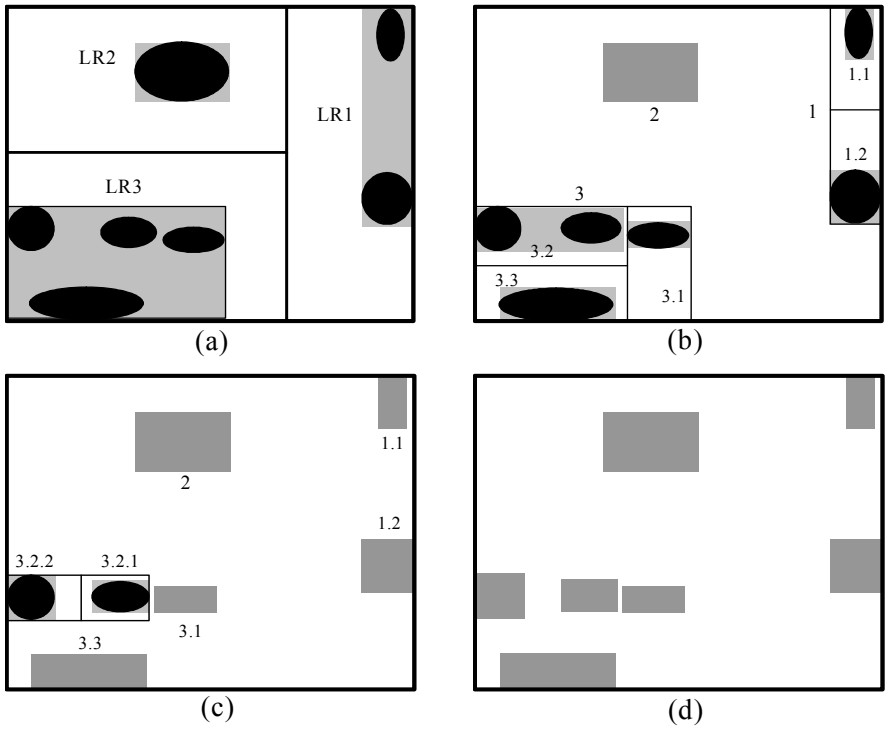


Figure 7. Illustration of GARDEN$_{HD}$ clustering.

Since optimal clustering is prohibitively expensive, all clustering algorithms are heuristic methods. However, GARDEN$_{HD}$ is a high performance clustering method not only because of the heuristics it uses, but also because of the way it pursues the basic question that clustering methods must answer: where do data points lie in the space? Rather than looking where the points are, GARDEN$_{HD}$ looks first for areas where they are not. After eliminating these areas, what is left gives a good idea where the points are [Orl05]. This is a sensible approach from the efficiency point of view because it is much easier to determine whether a region is empty, dense, or sparse than to look for points in any given neighborhood.

By detecting clusters, all clustering methods perform implicit data space reduction. However, in addition to the fact that the reduction is neither effective nor scalable, it regularly does not preserve the essence of data in high-dimensional spaces. In contrast, the compact cluster rep-

resentation of data produced by GARDEN$_{HD}$ can preserve all essential properties of data, including the locations, shapes, and densities of clusters. As a result, many data mining tasks can be performed on this representation alone. Therefore, GARDEN$_{HD}$ can be used not only as a clustering algorithm, but also as a data-reduction method alternative to (or in conjunction with) dimensionality reduction, data compression, and data sampling.

Experiments show that GARDEN$_{HD}$ is not only an efficient and scalable clustering method, but that it effectively isolates densely populated areas in the space [Orl05]. Because of its efficiency, scalability, and the ability to effectively isolate areas with points, GARDEN$_{HD}$ is appropriate for any intelligent system that supports advanced content-based retrieval. By providing a good insight into data distribution, it can enable a close-to-optimal assignment of data to the units of persistent storage. Moreover, the experiments suggest that GARDEN$_{HD}$ can also be used as a general-purpose method for fast and scalable unsupervised learning.

7 Data-Sensitive Gamma Partitioning

The partitioning method, called *DSGP* (*Data-Sensitive Gamma Partitioning*), operates on the cluster representation of data produced by GARDEN$_{HD}$ and generates a space partition in which well-separated clusters appear in different regions of the space. The algorithm runs in time equivalent to $O(C^2)$ comparisons of D-dimensional points, where C is the number of clusters detected by GARDEN$_{HD}$.

Each data cluster is approximated by its minimum bounding hyper-rectangle (MBR), represented by the low and high endpoints. As in data-blind Γ partitioning, DSGP produces static Γ regions. However, the Γ regions are formed around spatial clusters. The number of resulting regions depends on the number of clusters. The objective is to store points of each disjoint cluster into a separate KDB-tree index.

Figure 8 illustrates the steps of the data-sensitive space partitioning strategy. Figure 8a shows four clusters detected by GARDEN$_{HD}$. The DSGP procedure starts by sorting the clusters based on their high endpoints along each dimension. As a result, each dimension is associated with a sorted list of cluster indices. The procedure detects the gaps between clusters as follows. Going from the higher to lower coordinates along each dimension, the low endpoint of each cluster is compared with the high endpoint of the next cluster until a gap is found. A partitioning hyper-plane is drawn in the middle of a detected gap, perpendicular to the dimension with the *minimum-containment region* above the gap. By "minimum-containment region", we mean a Γ region with the smallest number of clusters. The resulting space partition is stored in the Γ filter. The live regions bounding the points of each Γ region in the Γ filter are determined dynamically during initial and incremental data loading.

In Figure 8a, Cluster 1 has been assigned to the first Γ region. Hence, it is eliminated from further consideration. The same procedure is repeated, and Cluster 2 is assigned to another partition along the same dimension (Figure 8b). In the next iteration, a gap is found along the second dimension (Figure 8c). The last cluster is assigned to the remaining Γ region. During data loading, the constructed KBD-tree indices perform implicit partitioning of the respective Γ regions into a collection of index regions, each of which bounds the points in an index page (Figure 8d). Since no point can fall outside the live region of the corresponding Γ region, the KDB-tree index regions are effectively bounded by the corresponding live regions.

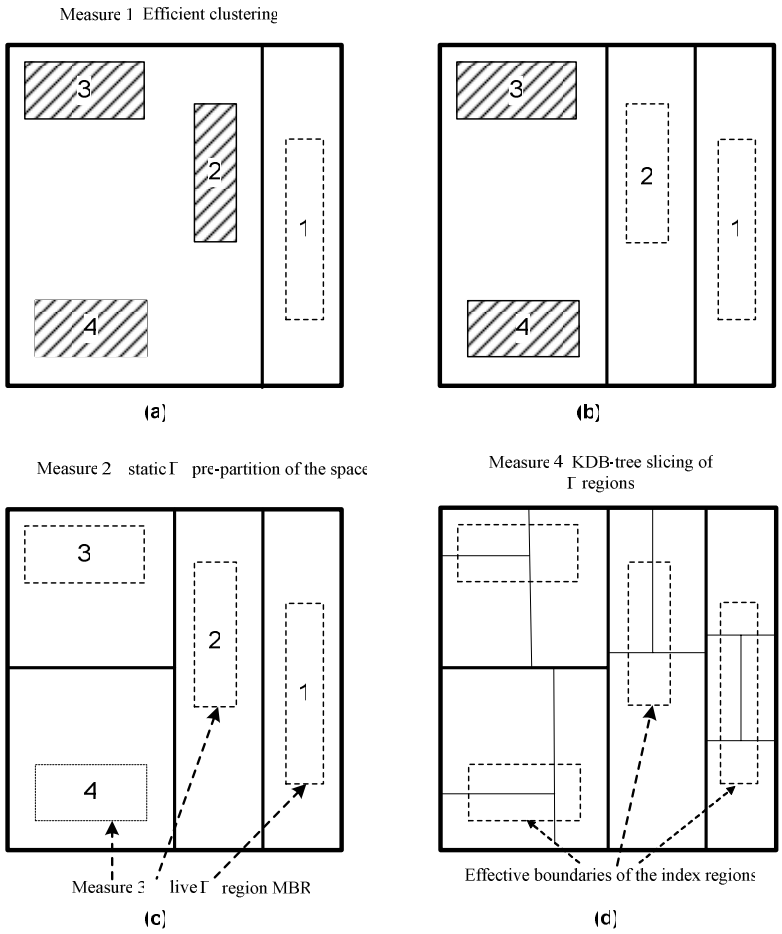


Figure 8. Steps of the DSGP space partitioning.

If multiple clusters appear in the same Γ region, the DSGP procedure performs "slicing" of the Γ region, so that each slice of the Γ region contains only one cluster. The current slicing procedure requires a pair-wise comparison of the given cluster MBRs. It is also possible that no gap can be found during an iteration of this algorithm or during slicing of a Γ region. In such a case, DSGP performs a data-blind Γ partitioning of the space (region) in which the overlapping cluster MBRs appear.

Figure 9 gives the DSGP partitioning algorithm. The live region LRU of the universe helps eliminate zero-extension dimensions. If only one cluster is detected, the space is statically partitioned as explained in Section 5. The indices of the clusters in the ClusterArray are stored in at most D lists. For each dimension *i* with non-zero extension, the cluster MBRs are sorted in the descending order of the *i*-th coordinates of their high endpoints. The algorithm traverses each list until a gap between two consecutive cluster MBRs is found and counts the number of clusters lying above the gap. A Γ region is extrapolated by partitioning the dimension for which the least number of clusters lie above the cut (gap).

```
PROCEDURE DataSensitiveSpacePartitioning
INPUT
  LRU;                                  // Live region of the universe:
  ClusterArray:    Clusters;
OUTPUT
  Regions;                              // array of resulting Gamma regions.
DECLATIONS
  NoClusters                            // number of clusters in ClusterArray;
  ClustersIndex [D][NoClusters];        // cluster index lists for D dimensions
  NoOfEliminatedClust = NoClusters;
  SplitDimension = D;                   // dimension along which a split occurred
BEGIN
  Eliminate zero-length dimensions of LRU;
  if NoClusters = 1 then
      staticGammaPartition (NoGenerators); // Default NoGenerators is currently 3
  else begin
     for all non-zero length dimensions do    // O(D·C·logC), where C is no. clusters
         sortClustersInd (ClustersIndex);     // Sort clusters on high points of cluster MBRs
     while NoClusters > 1 do          // O(D·C^2) worst case, average case is much better
          begin
          for all non-zero length dimensions do
               begin
               Traverse from high end, find the first gap (with the highest coordinate) be-
               tween two consecutive clusters, and record the number of clusters NoC
               traversed above the gap;
               if NoC < NoOfEliminatedClust then
                    Record the order number of the dimension in SplitDimension and set
                    NoOfEliminatedClust := NoC;
               end
          if NoOfEliminatedClust < NoClusters then   // Cut found for partitioning
               Assign the partition point and draw a partition line along the dimension;
          else   begin
               findCutsforOverlap ();              // eliminate a cluster and find a cut
               if noCutAvailable () then staticGammaPartition (NoGenerators);
               return;
               end
          Update Regions array with new partitioned region formed;
          Associate SplitDimension with the new region;
          Eliminate from the clusters array clusters before the gap;     // O(D·C)
          NoClusters ← NoClusters – NoOfEliminatedClust;
          NoOfEliminatedClust = NoClusters;
          if NoClusters = 1 then
               Update Regions array with the remaining region;
          end
     end
END
```

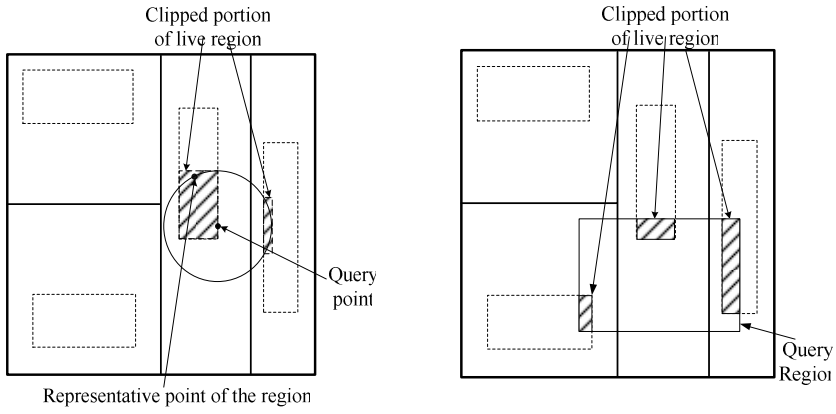
Figure 9. DSGP algorithm.

8 Similarity and Region Search

Through the Γ filter representing the partition produced by either DSGP or data-blind Γ partitioning, the data points are inserted into appropriate KDB-tree indices. As points are inserted, live regions of the Γ region and their slices are dynamically formed. Each inserted point either grows a live region or falls inside it. For a point lying inside a live region, its distance to the geometric center of the live region is calculated. This point becomes a representative of the region if it is closer to the center of the live region than the previous representative, if any. Note that the dynamic computation of representatives takes place after the clustering and partitioning are performed on an early data sample.

Figure 11 gives the algorithm for nearest-neighbor searching, called GammaNN (the *k*-NN algorithm is a simple variant of this) [Kul06]. Since the information about the space partition and live regions is maintained in memory, fast computations help eliminate from inspection a potentially large number of regions without any distance calculations. As noted earlier, fast in-memory computations are due to the way space partitioning is performed. Since the bulk of filtering is done in memory, a very good retrieval performance can be achieved.

Figure 10 depicts the processes of similarity and region searching. The nearest neighbour search in Figure 3a uses a query hyper-sphere with the query point at the center and the distance to its closest region representative as the radius. In this example, the hyper-sphere intersects two live regions whose clipped portions are searched.



(a) (b)

Figure 10. k-Nearest Neighbor and Region Search.

Once the live regions that overlap the query hyper-sphere or query rectangle (window) are determined, they are clipped against the hyper-sphere or the query window, respectively. The KDB-tree corresponding to an overlapping live region is then queried with the appropriate clip of the query window. In the case of a k-nearest neighbor searching, the query hyper-sphere dynamically shrinks as the new nearest neighbors are detected. The points returned by the queried KDB-tree indices are compared with the query point to construct the resulting list of *k*-nearest neighbors. For a region search, illustrated in Figure 10b, the points returned by the KDB-tree indices represent the result set.

```
PROCEDURE NearestNeighborSearch
INPUT
    Q;                  // query point:
    NoRegions;          // number of Gamma regions
    Regions;            // list of Gamma regions
OUTPUT
    Result.Point;       // nearest neighbor
    Result.Distance;    // distance to the nearest neighbor
DECLARATIONS
    Slice;              // a slice of a region (region can have one or more slices)
    Slice.LR;           // live region of the given slice
    TempResult;         // contains a temporary NN and the distance to it
    Distance := ∞, Dist; // temporary distances
    Qclip;              // query window (clip) by which an index is searched
BEGIN                   // find closest representative and "construct" the sphere
 for i:=1 to NoRegions do
   if sphereIntersectsGammaRegion (Q, Distance, Region[i]) then
      if Region[i].Cardinality > 0    // in a data-blind partition, region can be empty
         for j:=1 to Region[i].NoSlices do
            if sphereIntersectsLiveRegion (Q, Distance, Region[i].Slice[j].LR)
               begin
               MarkSlice (Region[i].Slice[j]);   // mark slice for later inspection
               if Dist := calculateDistance (Slice[j].Repr, Q) < Distance then
                  begin Slice := Region[i].Slice[j]; Distance := Dist; end
               end
 Qclip := constructSearchWindow (Q, Distance, Slice.LR); // construct query clip
 Result := searchIndex (i, Qclip);     // search index and return the temporary NN
 Distance := Result.Distance;      // examine points in other slices, shrinking the sphere
 for each other Slice in the list of marked slices do
   if sphereIntersectsLiveRegion (Q, Distance, Slice.LR) then
      begin             // since the sphere is shrinking, we had to test again for overlap
      Qclip := constructSearchWindow (Q, Distance, Slice.LR);
      TempResult :=searchIndex (i, Qclip);
      if TempResult.Distance < Distance then
         begin Distance := TempResult.Distance; Result := TempResult; end
      end
END
```

Figure 11. GammaNN algorithm for nearest-neighbor searching.

Figure 12 gives the algorithm of region search. The input to the algorithm is the query window. First, the procedure determines which Γ regions intersect the query window. This is done by the procedure *regionSearchFiltering* of Figure 5. If an intersecting region has cardinality greater than zero, the respective KDB-tree is queried with the clipped portion of the query window that intersects the corresponding live region.

```
PROCEDURE RegionSearch
INPUT
        Qwindow;        // query window
        NoRegions;      // number of regions integer
OUTPUT
        Result;             // list of points satisfying the query
DECLARATIONS
        Qclip;              // query window (clip)
        Region;             // list of Gamma regions
BEGIN                   // find all regions intersecting the hyper-sphere
        regionSearchFiltering (m, D, Qwindow, Gamma filter); // currently, m is set to 3
        for i:=1 to NoRegions do
           begin
           if windowIntersectsGR (Qwindow, Region[i]) then
              if Region[i].Cardinality > 0    // cardinality 0 is possible in data-blind partition
                 for j:=1 to Region[i].NoSlices do
                    if windowIntersectsLR (Qwindow, Region[i].Slice[j].LiveRegion) then
                       begin
                       Qclip := getQclip (Qwindow, Region[i].Slice[j].LiveRegion);
                       Result := Result ∪ searchIndex (i, Qclip);
                       end
           end
        return Result;
END
```

Figure 12. Region search algorithm.

9 Experimental Results

The experiments were performed on simulated and real data on a PC configuration with a 3.6 GHz CPU, 3GB RAM, and 280GB disk. In all structures, the page size was 8K bytes. We assumed a normalized D-dimensional space $[0,1]^D$. Each coordinate of a point was packed in 2 bytes. The GammaNN implementations with and without explicit clustering are referred to here as 'data aware' and 'data blind' algorithms, respectively. The static Γ partitioning of the data blind GammaNN was obtained assuming 3 generators, decided based on a number of experiments. In the synthetic data of up to 100 dimensions, the points are distributed across 11 clusters—one in the center and 10 in random corners of the space. The real data is a 54-dimensional forest cover type ("covtype") set obtained from the UCI machine learning repository (http://kdd.ics.uci.edu/databases/covertype/covertype.data.html).

9.1 Structure building and data loading time

Figure 13 gives the pre-processing time for two versions of GammaNN and the VA-File. For the data-blind algorithm, this time includes the time of space partitioning, I/O (reading the data), and the time for data loading (i.e., the construction of indices during insertion). The data-aware algorithm includes the clustering time in addition. Observe that the pre-processing time of this algorithm is heavily dominated by the construction of KDB-tree indices, whereas $\text{GARDEN}_{\text{HD}}$ clustering is very fast.

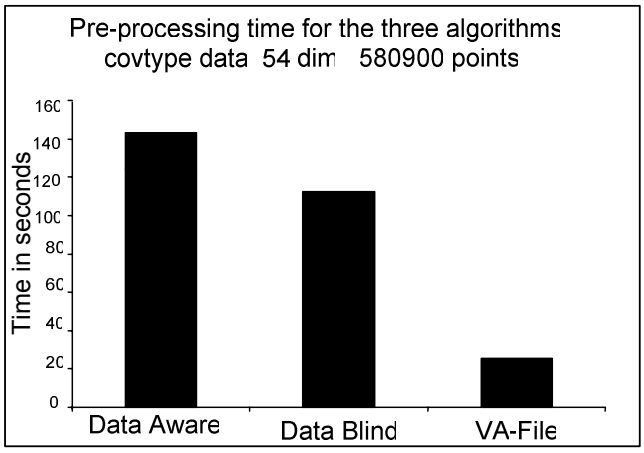


Figure 13. Preprocessing time including the time for data loading.

For the VA-File technique, the pre-processing time includes the time to generate the VA-File. Since this time is dominated by the calculation of approximation values and requires no insertion of points into any data structure, a faster pre-processing for VA-File is expected. However, since the pre-processing time is usually amortized over a large number of queries, it is much less consequential than the search time.

9.2 Similarity Search

Figure 14 shows the results on 100,000 synthetic data points as their dimensionality increases from 10 to 100. The data-aware algorithm is more than eight times faster than sequential scan and six times faster than the VA-File. The data-aware method and the VA-File incur almost the same number of page accesses to the data. However, this is because we counted only accesses to index or data pages, respectively. In other words, no page access was counted for the processing of the Γ filter or the VA-File, which favors the latter technique. If the VA-File were maintained on disk, the VA-File technique would incur many more page accesses.

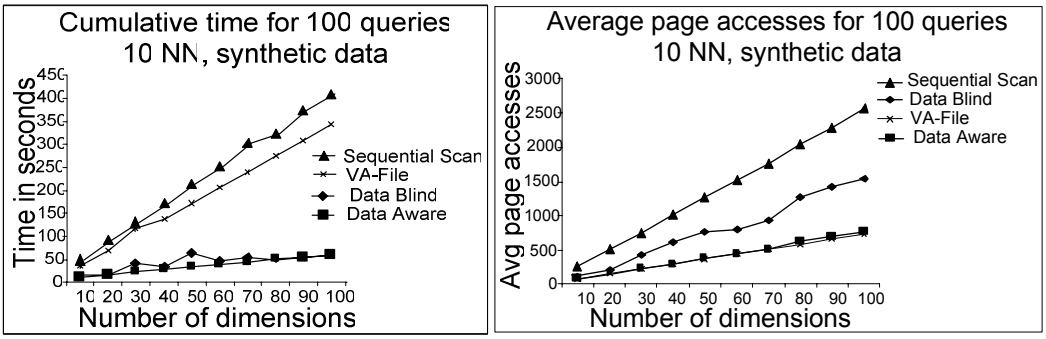


Figure 14. Synthetic distribution with query distribution same as data, 10 NN.

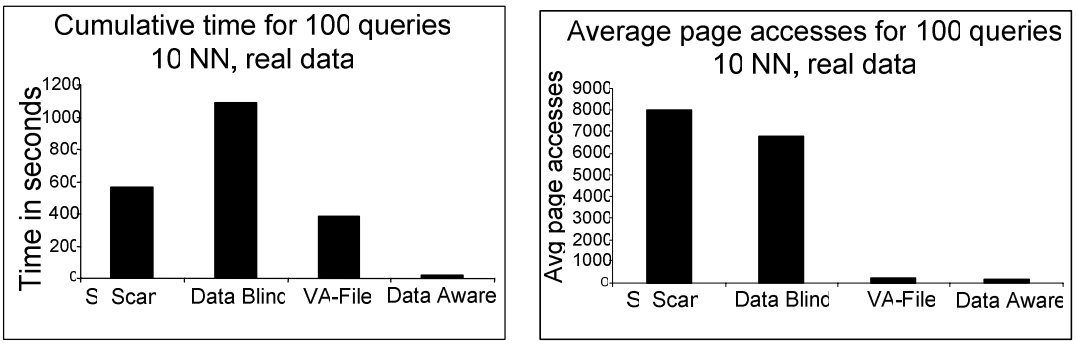


Figure 15. Real data with 100 queries selected from the real data file, 10 NN.

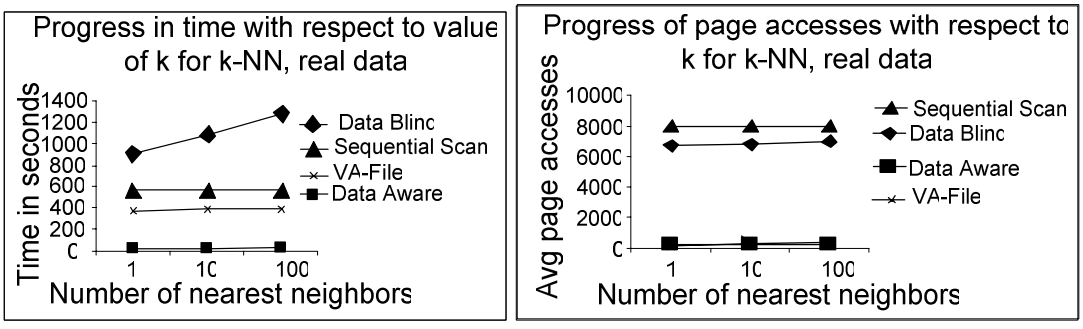


Figure 16. Progress as the number of nearest neighbors increases on real data.

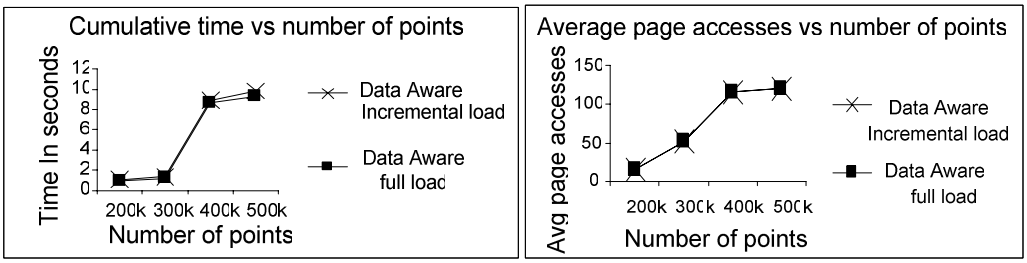


Figure 17. Time and average page accesses for incremental load on real data.

Figure 15 shows that the data-aware algorithm is significantly faster than sequential scan and the VA-File. For this experiment, 580,900 points were loaded into each structure, and the remaining 100 points in the data set were used as query points. Also noteworthy is that the data-aware algorithm accesses only about 3% of all data points.

Figure 16 shows the changes in the performance of different methods with respect to the increasing value of k in k-NN searching. Except for the data-blind algorithm, all methods have a stable performance as k grows up to 100.

Figure 17 shows the performance of the data-aware algorithm on incremental loading of the real data set. The clustering and space partitioning were performed on the first 100,000 points in the set, which were loaded into the structure. The results of 10-NN queries were recorded after subsequent incremental loads of 100,000 points. No re-clustering was performed after an incremental load. However, as described earlier, the live regions and their respective representatives were dynamically modified during the incremental loads. The equivalent results of the same algorithm but without incremental loading (in Figure 17, referred to as "data aware, full load"), i.e. after clustering the entire subset of the data, are used as benchmarks.

One can observe from Figure 17 that the data-aware GammaNN algorithm results in almost the same number of page accesses and the query execution times with or without incremental loading of data. This suggests that GammaNN reacts well to incremental loads. As in this case, in many practical environments, it will require no re-clustering of data even after many incremental loads. This is particularly important for scientific applications, which regularly obtain data through incremental loads.

9.3 Region Search

The region search algorithm was tested on the same two data sets: synthetic (center-corners) and the real covtype dataset (see Figures 18 and 19, respectively). The volume of generated queries was 1% of the total space. The timing results on the synthetic data set show that the data-blind and data-aware algorithms are much faster than than sequential scan. In terms of page accesses, data-sensitive approach has much better performance than the other two methods. The results on the covtype data set also reveal significantly better performance of the data-sensitive algorithm.

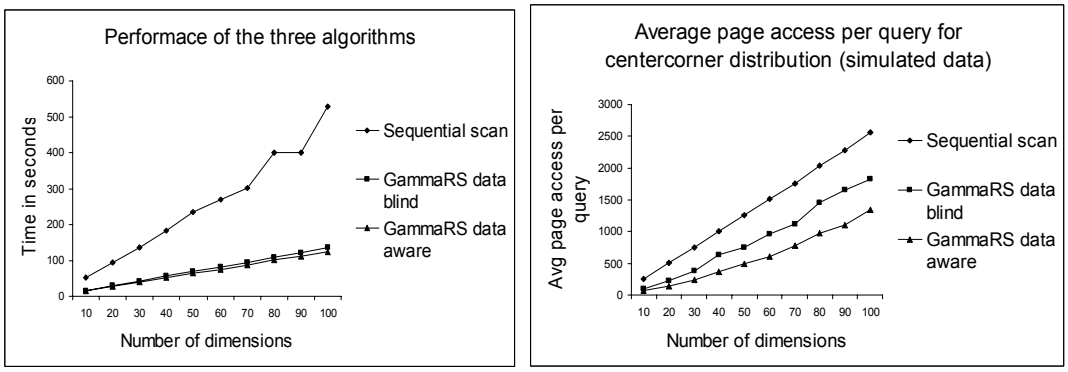


Figure 18. Time and average page access/query for 1% volume query.

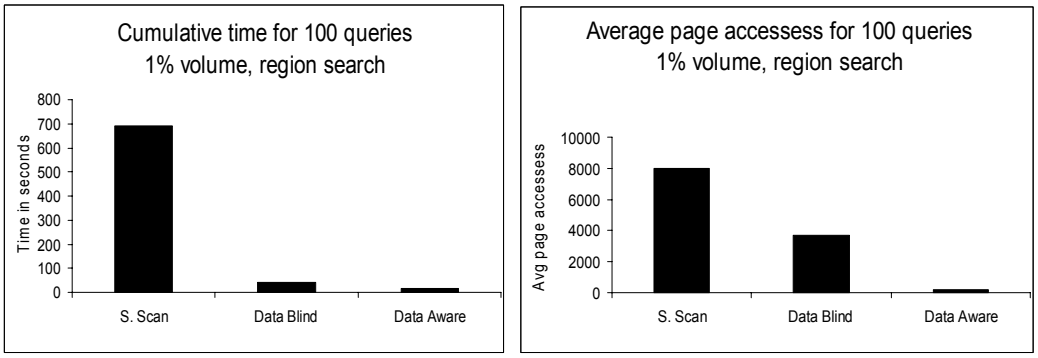


Figure 19. Time and average page access per query for real data, 1% volume query.

10 Discussion

In this paper, we described a new system for storage and retrieval in high dimensionalities, called D*. The system employs explicit data clustering using a new density-based clustering method and a new data-sensitive space partitioning method in order to preserve the locality of data and reduce the volumes of clusters on storage. The application of a memory-based filter further improves the performance of similarity and region searching.

The comparison of the data-sensitive and data-blind approach clearly highlights the importance of clustering data on storage for efficient similarity search. Our approach can support exact similarity search while accessing only a small fraction of data. The algorithm is very efficient in high dimensionalities and performs better than sequential scan and the VA-File technique. The performance remains good even after incremental loads of dynamically growing data sets without re-clustering.

The high retrieval performance is due to the system's adherence to the design principles of Section 3. By detecting dense areas in the space, the clustering facility determines the spatial proximity of data. Data-sensitive space partitioning enables a static pre-clustering of data on storage according to their spatial proximity. Storing the points of every region into a separate KDB-tree index enables a dynamic sub-clustering of data into index pages that correspond to relatively small and dense index regions, as required by the principle of clustering.

The application of the memory-resident filter is important for several reasons. The structure dynamically channels incoming data into the appropriate indices. It dramatically reduces the number of costly distance computations. With the dynamically maintained live regions, it also reduces the amount of searching over empty space, enabling a potentially significant reduction of search space before accessing the storage. With simple extensions, the system can facilitate various analytical tasks, such as data classification and data pre-processing. In our future work, we plan to incorporate some of these facilities.

Perhaps in no other systems is the interplay between data storage and data mining more transparent than in scientific data grids. Their services like request planning and data replication are designed to increase the utilization of resources and the availability of data. However, the issues of data storage and organization have received relatively little consideration, even though they are critically important for these environments. This is because, when the size of a data collection exceeds certain limits, the capabilities of any given site in a distributed and replicated environment such as a data grid are determined not only by its computational and data resources but also by the way data is organized on storage.

For example, a simple temporal ordering of data on storage is adequate for certain tasks that access data in their temporal order, e.g. full data replication. However, when the repository is queried on the intrinsic properties of data, the items that satisfy the query appear to be randomly distributed across the storage. Therefore, a typical query must access many storage units, and only a small fraction of the retrieved data is relevant. In scientific data grids, this not only adversely affects the performance of analytical processing, but also has the effect of "blinding" the process of request planning. The latter process selects the execution site for the given program based in part on an estimate as to how much data will be accessed.

Clustering data on storage can generate two orders of magnitude fewer accesses to the storage than the temporal layout of data. Unfortunately, clustering large data collections would require considerable computational and storage resources as well as periodic re-clustering of the entire repository. Sometimes, clustering data on a subset of dimensions is what the application needs. However, in other situations, this can lead to suboptimal performance as the values of the remaining dimensions are disseminated across storage in a virtually random fashion. Further complicating the matter, the access patterns change over time. Therefore, any given storage organization is likely to become inappropriate at one time or another.

To address these problems, data repository should be replicated. However, since different types of processing may require radically different storage organizations, the layout of data on different replica sites should be different. A dynamically clustered storage organization that can take the advantage of any number of dimensions, possibly all dimensions of data, would tremendously facilitate this goal.

The technology described in this paper is designed with this goal in mind. It enables a mix of static and dynamic decisions designed to achieve full benefits of clustering data on storage at reasonable costs. As in the D* system, a static partition of the space can be used to organize data into *a priori* clusters corresponding to different regions in the space. For best results, the space partition can be derived after clustering an early sample of data. Even though the space partition is static, the process of sub-clustering is performed dynamically while the data is

arriving. Depending on the type of storage, each *a priori* cluster of data (data points in a region of the space partition) can be maintained in a separate multi-dimensional index on disk (as in the D* system) or a group of files on tertiary storage that are periodically re-clustered. Since this *a posteriori* clustering operates on a relatively small subset of the data, it can be performed in environments with limited resources. Our future plans include adaptations of the technology described in this paper for tertiary-storage environments.

References

[Agg01] C.C. Aggarwal, "On the Effects of Dimensionality Reduction on High Dimensional Similarity Search," Proc. 20th ACM PODS Symposium on Principles of Database Systems, 256-266, 2001.

[Agg02] C.C. Aggarwal, "Hierarchical Subspace Sampling: A Unified Framework for High Dimensional Data Reduction, Selectivity Estimation and Nearest Neighbor Search," Proc. ACM SIGMOD Conf., 452-463, 2002.

[Ave01] P. Avery and I. Foster, "The GriPhyN Project: Towards Petascale Virtual Data Grids," GriPhyN-14, 2001. http://www.griphyn.org

[Bey99] K.S. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, "When is `Nearest Neighbor' Meaningful?," Proc. 7th Int. Conf. on Database Theory, 217-235, 1999.

[Blo97] S. Blott and R. Weber, "A Simple Vector-Approximation File for Similarity Search in High-Dimensional Vector Spaces," Technical report, Esprit Project Hermes (#9141), 1997.

[Boh01] C. Bohm, S. Berchtold and D.A. Keim."Searching in High-Dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases," ACM Computing Surveys, 33(3): 322-373, 2001.

[Ber98] S. Berchtold, C. Bohm and H.P. Kriegel, "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," Proc. ACM SIGMOD Conf., 142-153, 1998.

[Ber96] S. Berchtold, D.A. Keim and H.P. Kriegel. "The X-tree: An Index Structure for High-Dimensional Data," Proc. of 22nd VLDB, Conf., 28-39, 1996.

[Bec90] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD, 322-331, 1990.

[Cha99] K. Chakrabarti and S. Mehrotra. "The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces," Proc. 15th ICDE Conf.. 440-447, 1999.

[Cha00] K. Chakrabarti and S. Mehrotra. "Local dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces," Proc. 26th VLDB Conf., 89-100, 2000.

[Che01] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," Journal of Network and Computer Applications, 23:187-200, 2001.

[Cia97] P. Ciaccia, M. Patella, and P. Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces," Proc. 23rd VLDB Conf., 1997.

[Fag03] R. Fagin, R. Kumar, D. Shivakumar: "Efficient Similarity Search and Classification via Rank Aggregation," Proc. ACM SIGMOD Conf., 301-312, 2003.

[Fos01] I. Foster, C. Kesselman and S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organization,” Int. Journal of High Performance Computing Applications, 15(3):200-222, 2001.

[Gio99] A. Gionis, P. Indyk and R. Motwani, “Similarity Search in High Dimension via Hashing,” Proc. 25th VLDB Conf., 518-529, 1999.

[Gut84] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching,” Proc. ACM SIGMOD Conf., 47--54, 1984.

[Hos00] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger, “Data Management in an International Data Grid Project,” Proc. First IEEE/ACM Int. Workshop on Grid Computing, 2000.

[Kat97] N. Katayama and S. Stoh. "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," SIGMOD Record 26(2):369-380, 1997.

[Kul06] S. Kulkarni and R. Orlandic, “High-Dimensional Similarity Search using Data-Sensitive Space Partitioning,” Proc. 17th Int. Conf. on Database and Expert Systems DEXA, 2006. (to appear)

[Kur01] T. Kurc, U. Catalyurek, C. Chang, A. Sussman and J. Saltz, “Visualization of Large Datasets with the Active Data Repository,” IEEE Computer Graphics and Applications, 21(4):24-33, 2001.

[Lin95] K. Lin, H. Jagadish, and C. Faloutsos, "The TV-tree: An Index Structure for High-Dimensional Data," VLDB Journal. 3 (1995): 517-542.

[Luk04] J. Lukaszuk and R. Orlandic, "Efficient High-Dimensional Indexing by Superimposing Space-Partitioning Schemes," Int. Database Engineering and Applications Symposium IDEAS, 257-264, 2004.

[Orl03] R. Orlandic, “Effective Management of Hierarchical Storage Using Two Levels of Data Clustering,” Proc. 20th IEEE / 11th NASA Goddard Conf. on Mass Storage Systems and Technologies, 270-279, 2003.

[Orl05] R. Orlandic, Y. Lai and W.G. Yee, “Clustering High-Dimensional Data Using an Efficient and Effective Data Space Reduction, Proc. ACM Conf. on Information and Knowledge Management CIKM’05, 201-208, 2005.

[Orl06] R. Orlandic, J. Lukaszuk, S. Kulkarni, and W.G. Yee, “On Two Principles of Designing Multi-Dimensional Access Methods,” Technical Document, 2006. Available at www.cs.iit.edu/~egalite

[Orl02] R. Orlandic, J. Lukaszuk and C. Swietlik, “The Design of a Retrieval Technique for High-Dimensional Data on Tertiary Storage,” SIGMOD Record, 31(2):15-21, 2002.

[Rav98] K.V. Ravi-Kanth, D. Agrawal and A. Singh. “Dimensionality Reduction for Similarity Search in Dynamic Databases,” Proc. ACM SIGMOD Conf., 166-176, 1998.

[Rob81] J.T. Robinson, “The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes,” Proc. ACM SIGMOD Conf., 10-18, 1981.

[Sak00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. "The A-tree: An Index Structure for High-dimensional Spaces Using Relative Approximation," Proc. 26th VLDB Conf., 516-526, 2000.

[Sei98] T. Seidl and H.P. Kriegel, “Optimal Multi-Step k-Nearest Neighbor Search,” Proc. ACM SIGMOD Conf., 154-165, 1998.

[Sex04] A.P. Sexton and R. Swinbank, “Bulk Loading the M-Tree to Enhance Query Performance,” Proc. 21st British National Conference on Databases BNCOD, 190-202, 2004.

[She02] X. Shen and A. Choudhary, “A Distributed Multi-Storage I/O System for High Performance Data Intensive Computing,” Proc. IEEE Int. Symposium on Cluster Computing and the Grid CCGrid, 2002.

[Sho99] A. Shoshani, L.M. Bernardo, H. Nordberg, D. Rotem and A. Sim, “Multidimensional Indexing and Query Coordination for Tertiary Storage Management,” Proc. 11th Int. Conf. on Scientific and Statistical Database Management SSDBM, 214-225, 1999.

[deV02] A.P. de Vries, N. Mamoulis, N. Nes and M.L. Kersten. “Efficient k-NN Search on Vertically Decomposed Data,” Proceedings of ACM SIGMOD International Conference on Management of Data. 322-333, 2002.

[Web98] R. Weber, H.J. Schek and S. Blott, “A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces,” Proc. 24th VLDB Conf., 194-205, 1998.

[Whi96] D.A. White and R. Jain. "Similarity Indexing with the SS-tree," Proc. 12th ICDE Conf., 516-523, 1996.

[Wu04] K. Wu, E.J. Otoo and A. Shoshani. “On the Performance of Bitmap Indices for High Cardinality Attributes,” Proc. 30th VLDB Conf., 24-35, 2004.

[Yu04] C. Yu, S. Bressan, B.C. Ooi and K.-L. Tan, “Querying High-Dimensional Data in Single-Dimensional Space,” VLDB Journal 13(2):105-119, 2004.

[Yu01] C. Yu, B.C. Ooi, K.-L. Tan and H.V. Jagadish, “Indexing the Distance: An Efficient Method to KNN Processing,” Proc. 26th VLDB Conf., 421-430, 2001.