High-Dimensional Similarity Search using Data-Sensitive Space Partitioning

Sachin Kulkarni[1] and Ratko Orlandic[2]

[1] Illinois Institute of Technology, Department of Computer Science,
Chicago 60616, USA
kulksac@iit.edu
[2] University of Illinois at Springfield, Computer Science Department,
Springfield 62703, USA
rorla2@uis.edu

Abstract. Nearest neighbor search has a wide variety of applications. Unfortunately, the majority of search methods do not scale well with dimensionality. Recent efforts have been focused on finding better approximate solutions that improve the locality of data using dimensionality reduction. However, it is possible to preserve the locality of data and find exact nearest neighbors in high dimensions without dimensionality reduction. This paper introduces a novel high-performance technique to find exact *k*-nearest neighbors in both low and high dimensional spaces. It relies on a new method for data-sensitive space partitioning based on explicit data clustering, which is introduced in the paper for the first time. This organization supports effective reduction of the search space before accessing secondary storage. Costly Euclidean distance calculations are reduced through efficient processing of a lightweight memory-based filter. The algorithm outperforms sequential scan and the VA-File in high-dimensional situations. Moreover, the results with dynamic loading of data show that the technique works well on dynamic datasets as well.

1 Introduction

Many traditional access methods are ineffective in high-dimensional spaces [10, 19]. Moreover, real high-dimensional data are often correlated or clustered, and the data tends to occupy only a small fraction of the space [5]. An appropriate similarity search method must be aware of the locality of data in high dimensions. However, most methods for finding the locality of data rely on dimensionality reduction. Unless a multi-step approach is applied [16], this leads to approximate results.

The problem of similarity search can be stated as follows: Given a database with N points and a query point q in some metric space, find $k \geq 1$ points closest to q [6]. Effective solutions to this problem find applications in computational geometry, geographic information systems, multimedia databases, data mining, etc. Most of these applications deal with Euclidean multi-dimensional spaces.

In order to tackle the "curse of dimensionality", various approximate solutions based on dimensionality reduction have been proposed [2, 6, 7]. Aggarwal [2] empha-

sized 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 [7]. The emphasis in [2] and [7] is on finding the locality of the data in a way that makes the process of dimensionality reduction more "data aware". The focus here is on local rather than global distribution of data.

Significant effort in finding the exact nearest neighbors has yielded limited success. The SR-Tree [9] 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 [4], the SR-Tree is at par with sequential scan when dimensionality is at least 20. Blott and Weber [5] proposed the VA-File, which applies a filter to the sequential scan using the concept of vector approximations. The bit-encoded approximations provide bounds to guide the elimination of points during the search. The solution is simple and tends to be efficient.

A-tree [15] and i-distance [19] are reported to work well in high dimensions. The A-tree is an index structure that stores virtual bounding rectangles, which approximate minimum bounding rectangles. i-distance uses the concept of space partitioning to separate the data into different regions. The data for each region are transformed into a single-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 searching in high-dimensional spaces is still an intriguing one.

In this paper, 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 algorithm for exact similarity search in high-dimensional spaces. The basic idea is to separate clusters in the dataset and eliminate searching over the empty space, thus improving the retrieval performance. We adopt an explicit density-based clustering using the efficient $\text{GARDEN}_{\text{HD}}$ clustering technique [13], which is different than the sampling-based approach proposed in [19]. We then apply a new space partitioning technique, called DSGP (data-sensitive gamma partitioning), which operates on the compact cluster representation of data produced by $\text{GARDEN}_{\text{HD}}$.

The paper also presents the results of comprehensive experiments showing that our approach can efficiently find exact k nearest neighbors in high dimensions. Moreover, it can work efficiently on dynamically growing data. The algorithm is compared to the sequential scan, the VA-File, and the GammaSLK partitioning and indexing technique without explicit data clustering [12].

The rest of this paper is organized as follows. Section 2 gives the basic design principle underlying the proposed approach and our clustering and partitioning schemes. Section 3 presents the system architecture and briefly reviews the adopted Γ partitioning and $\text{GARDEN}_{\text{HD}}$ clustering. Section 4 introduces the data-sensitive space partitioning. Section 5 introduces the proposed algorithm for similarity search. Section 6 provides experimental evidence. Section 7 concludes the paper.

2 Design Principle

For generality, 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. Then the design principle underlying our approach can be stated as follows: *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.*

The term "relevant dimensions" refers to the fact that multi-dimensional region queries may have "affinity" for certain dimensions, consistently leaving other dimensions unrestricted. However, since exact similarity searching must restrict the search space in all dimensions, 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.

To understand the logic behind this principle, let us assume for the moment an idealized system that, for any given query, accesses only those pages on secondary storage containing data items that satisfy the query. Moreover, assume that N data items are divided into $M << N$ pages and that $K << N$ items satisfying the query are randomly distributed among the pages. Then, a well-known Cardenas expression $A = M \cdot (1-(1-1/M)^k)$ gives a good estimate of the number of accessed pages for the given query. Note that the number of pages with useful data is at most $min\{K, M\}$ and, when $K << M$, the above expression can be approximated by $A \approx M \cdot (1-(1-K/M)) = K$.

Eliminating the assumption that data items are randomly distributed among the pages, the number of accessed pages can be estimated by a more general expression, valid for any $K, M \leq N$: $A = K/I$, where $I \geq 1$ is the average number of items that satisfy the query in an accessed page. Since the goal of clustering data on storage is to increase the number of useful items in any page accessed by a typical query, the parameter $H = I/C \leq 1$, where C is the page capacity, is a good measure of the quality of clustering with respect to the given query. Obviously, when $C = 1$ or $H = I/C$, clustered storage organization has the same effect as the organization with randomly distributed data. However, when $C >> 1$ and H is as close to 1 as possible, the performance improvement can be significant—as much as C times fewer page accesses. For this to happen, the storage utilization must also be high.

To develop an appropriate clustering strategy, let us now consider the problem of supporting multi-dimensional region queries, which restrict the ranges of values in one or more dimensions of a D-dimensional unit space $[0,1]^D$. We use the term *storage cluster* to denote a spatial region consisting of points in a storage unit, i.e. a page.

In this context, increasing H implies increasing the probability that each storage cluster with useful data is completely covered by the query. However, since clustering must benefit not one but many different queries, the storage organization must be such that, for every storage cluster S and any query Q, it decreases the probability $P(S \cap Q)$ that S overlaps Q (which would trigger access to the corresponding page), while increasing the probability $P(S \subseteq Q)$ that it is covered by Q. Assuming that S is repre-

sented by its minimum bounding hyper-rectangle, and that all possible positions of S are equally likely, one can easily show that, for data dimensionality D:

$$P(S \cap Q) = \prod_{i=1}^{D} \min\{Q_i + S_i, 1\} \quad \text{and} \quad P(S \subseteq Q) = \prod_{i=1}^{D} \max\{\frac{Q_i - S_i}{1 - S_i}, 0\},$$

where S_i and Q_i are the extensions (lengths) of S and Q, respectively, in an axis i.

Since the extensions of any given query are fixed, the way to reduce $P(S \cap Q)$ and increase $P(S \subseteq Q)$ for an arbitrary query Q is to reduce the lengths S_i of S along all dimensions i restricted by Q (i.e., all i for which $Q_i < 1$). This and the earlier observation about storage utilization lead to the design principles stated earlier.

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 stated design principle can be achieved either by clustering or by data space reduction only. However, a facility to do both is an advantage.

Explicit data clustering with effective data space reduction can facilitate various kinds of retrieval, including similarity searching, 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 (externally defined) 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 [12] and the DSGP data-sensitive space partitioning technique introduced later in this paper.

3 System Architecture

Figure 1 gives the architecture of our system for efficient retrieval of data that scales well with the increasing dimensionality. The process of data clustering produces a compact cluster representation of data. Operating on this representation, the partitioning module produces a data-sensitive Γ space partition (see below). The derived space partition is maintained by a light-weight in-memory structure, called the *Γ filter*.

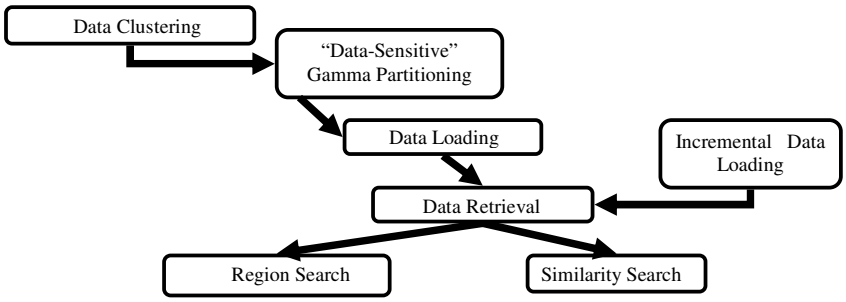

Fig. 1. Key processes of the system.

In the process of data loading, this memory structure 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.

The KDB-tree indexing technique is not necessarily optimal for this environment. The R-tree would yield faster retrieval, but at the expense of slower insertions. In environments with frequent insertions, the later cost is not insignificant.

3.1 Gamma Partitioning

Γ space partitioning was first introduced in [11]. A *D*-dimensional space 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*. Except for the innermost subspace, each Γ subspace is further divided into at most *D* hyper-rectangular regions, called *Γ regions*, by means of (*D*–1)-dimensional hyper-planes, each of which lies on an upper boundary of the inner generator. Beginning with the outermost generator, these Γ regions are carved out from the base region one by one (see Figure 2c below). Each coordinate of the high endpoint of a generator gives the position of the hyper-plane that separates a Γ region from the space in which the subsequent Γ regions lie. Note that, if *G* is the number of generators and *D* the number of dimensions, the total number of created regions is at most $1+(G-1)\cdot D$.

In our system, Γ space partition is compactly represented by the Γ filter. 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.

3.2 GARDEN$_{HD}$ Clustering

GARDEN$_{HD}$ [13] is designed to provide a fast and accurate insight into data distribution in order to facilitate data mining or retrieval. This clustering method efficiently and effectively separates disjoint areas with points, which is the primary reason we selected it for this application. With an appropriately selected density threshold, which is the only input parameter, GARDEN$_{HD}$ runs in O(Nlog$N$) time [13], where N is the number of *D*-dimensional points in the dataset.

GARDEN$_{HD}$ 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 [13], the first phase performs an efficient data space reduction, identifying rectangular cells whose density is above the user-defined threshold. In the second phase, the adjacent dense cells are merged into larger clusters. It is the application of Γ partitioning that enables the algorithm to efficiently cluster data in high-dimensional spaces without dimensionality reduction.

4 Data Sensitive Gamma Partitioning - DSGP

The partitioning method, called *DSGP* (*Data-Sensitive Gamma Partitioning*), uses 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. DSGP runs in time equivalent to $O(L^2)$ comparisons of D-dimensional points, where L 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 into regions of equal volume [11], DSGP produces static Γ regions, but 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.

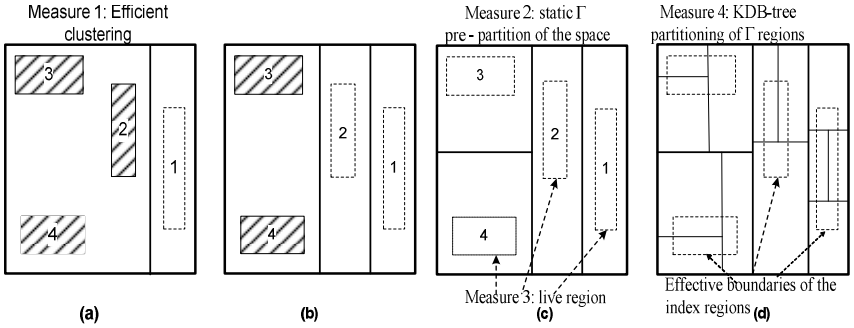

Fig. 2. Steps of the DSGP space partitioning.

Figure 2 illustrates the steps of the data-sensitive space partitioning strategy. Figure 2a 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 the "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 2a, 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 2b). In the next iteration, a gap is found along the second dimension (Figure 2c). 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 2d). 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.

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 [12] of the space (region) in which the overlapping cluster MBRs appear.

5 Similarity Search

Through the constructed Γ filter representing the partition produced by DSGP, data points are inserted into appropriate KDB-tree indices. As points are inserted, live regions of the Γ regions 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 is performed on an early data sample.

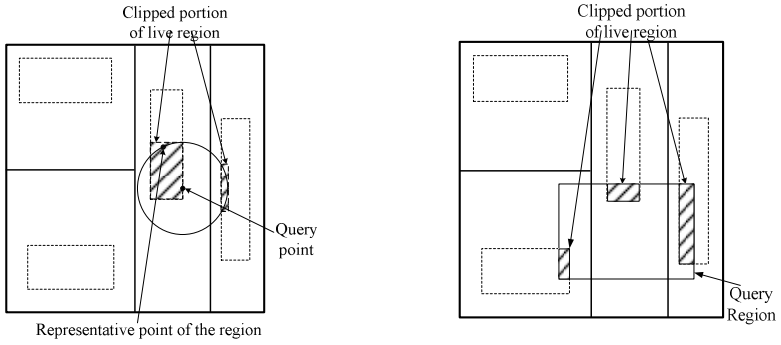

Fig. 3. *k*-nearest neighbor and region search.

Figure 3 depicts the processes of nearest neighbor 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 and requires the region searches only for the overlapping clipped portions of the live regions. Figure 3b is included to emphasise that region search can be performed in a similar way. Figure 4 gives the algorithm for nearest-neighbor searching, called GammaNN (the *k*-NN algorithm is a simple variant of this). A similar procedure can be used for region search as well.

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 *k*-nearest neighbor searching, the query hyper-sphere dynamically shrinks as the new nearest neighbors

are detected. The points returned by the interrogated KDB-tree indices are compared with the query point to construct the resulting list of *k*-nearest neighbors. For a region search, the points returned by the KDB-tree indices represent the result set.

```
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
Local:
    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 distance
    Qclip;              // query window (clip) by which an index is searched

BEGIN GammaNN
                        // 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
                        // examine points in the "closest" slice
 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 for overlap again
      Qclip ← constructSearchWindow (Q, Distance, Slice.LR);
      TempResult ←searchIndex (i, Qclip);
      if TempResult.Distance < Distance then
         begin Distance ← TempResult.Distance; Result ←TempResult; end
      end
END GammaNN
```

Fig. 4. GammaNN algorithm for nearest-neighbor searching.

6 Experimental Evidence

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' [12] 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[1].

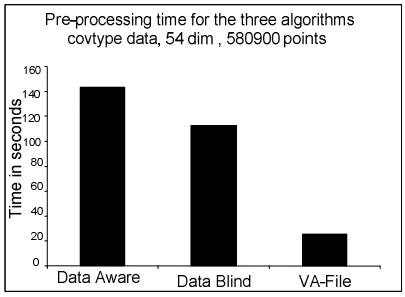

Fig. 5. Preprocessing time including the time for data loading.

Figure 5 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 plus insertion of data). 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 fast.

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 [17] and requires no insertion of points into any data structure, 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.

Figure 6 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, and the relative differences with respect to GammaNN variants would be closer to those observed for processing times.

[1] http://kdd.ics.uci.edu/databases/covertype/covertype.data.html.

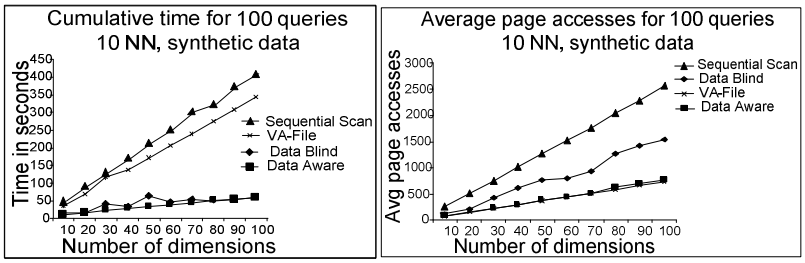

Fig. 6: Synthetic data with query distribution same as data, 10 NN.

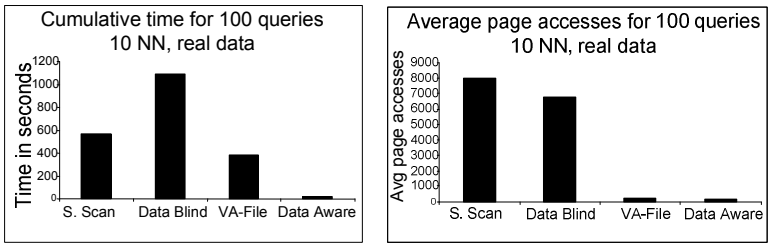

Fig. 7: Real data with 100 queries selected from the real data file, 10 NN.

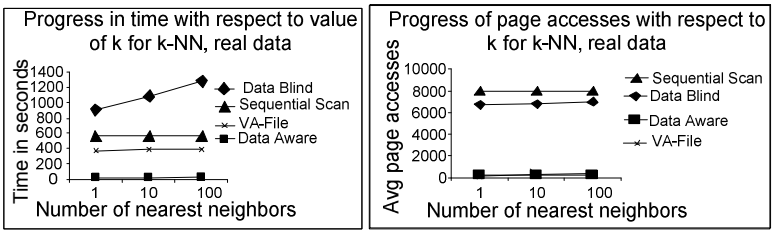

Fig. 8: Progress as the number of nearest neighbors increases on real data.

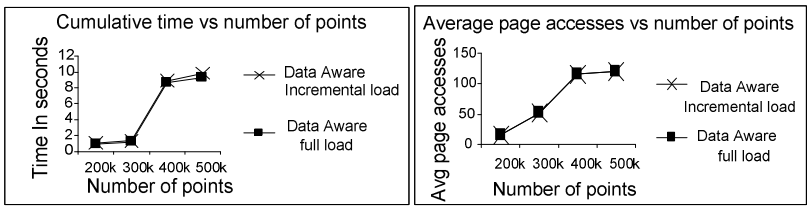

Fig. 9: Time and average page accesses for incremental load on real data.

Figure 7 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 8 shows the changes in 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 9 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 9, 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 9 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.

7 Discussion and Conclusions

In this paper, we proposed a new technique for exact similarity searching in high dimensionalities, called GammaNN. The GammaNN technique 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 data clusters on storage. The application of a memory-based filter with live regions further improves the performance of similarity 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 performance of GammaNN similarity searching is mainly due to the structure's adherence to the design principle stated in Section 2. By detecting dense areas in the space, the data clustering facility determines the spatial proximity of data. The 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 enables a dynamic sub-clustering of data into index pages corresponding to relatively small and dense regions in the space, as required by our design principle.

The application of the memory-resident filter is important for several reasons. It dynamically channels incoming data into 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.

In our future work, we plan to incorporate R-trees into the system and provide a facility for handling data with missing values.

Acknowledgment

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

References

1. Aggarwal, C.C.: On the effects of dimensionality reduction on high dimensional similarity search, Proc. 20th PODS Conf., (2001) 256–266
2. Aggarwal, C.C.: Hierarchical subspace sampling: A unified framework for high dimensional data reduction, selectivity estimation and nearest neighbor search, Proc. ACM SIGMOD Conf., (2002) 452-463
3. Berchtold, S., Ertl, B., Keim, D., Kriegel, H.P., Seidl, T.: Fast nearest neighbor search in high-dimensional space, Proc. 14th ICDE Int. Conf. on Data Engineering, (1998) 209-218.
4. Beyer, K.S., Goldstein, J., Ramakrishnan, R. and Shaft, U.: When is `nearest neighbor' meaningful?, Proc. 7th Int. Conf. on Database Theory, (1999) 217-235
5. Blott, S., Weber, R., A simple Vector-Approximation file for similarity search in high-dimensional vector spaces. Technical report, Esprit Project Hermes (no. 9141), (1997)
6. Fagin, R., Kumar, R., Shivakumar, D.: Efficient similarity search and classification via rank aggregation, Proc. Proc. ACM SIGMOD Conf., (2003) 301-312
7. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimension via hashing, Proc. 25th VLDB Conf., (1999) 518-529
8. Hinneburg, A., Aggarwal, C.C., Keim, D.A.: What is nearest neighbor in high dimensional spaces?, Proc. 26th VLDB Conf., (2000) 506-515
9. Katayama, N., Satoh, S.: The SR-tree: An index structure for high-dimensional nearest neighbor queries, SIGMOD Record 26(2): (1997) 369-380
10. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. Math. Statist, Prob. 1: (1967) 281–297
11. Orlandic, R., Lukaszuk, J., Swietlik, C.: The design of a retrieval technique for high-dimensional data on tertiary storage, SIGMOD Record 31(2): (2002) 15–21
12. Orlandic, R., Lukaszuk, J.: Efficient high-dimensional indexing by superimposing space-partitioning schemes, Proc. 8th International Database Engineering & Applications Symposium IDEAS'04, (2004) 257-264
13. Orlandic, R., Lai, Y., Yee, W.G.: Clustering high-dimensional data using an efficient and effective data space reduction, Proc. ACM Conference on Information and Knowledge Management CIKM'05, (2005) 201-208
14. Robinson, J.T.: The K-D-B-Tree: A search structure for large multidimensional dynamic Indexes**,** Proc. ACM SIGMOD Conf., (1981) 10-18
15. Sakurai, Y., Yoshikawa, M., Uemura, S., Kojima, H.: The A-tree: An index structure for high-dimensional spaces using relative approximation, Proc. 26th VLDB Conf., (2000) 516-526
16. Seidl, T., Kriegel, H.P.: Optimal multi-Step k-nearest neighbor search. Proc. ACM SIGMOD Conf., (1998) 154-165
17. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity search methods in high-dimensional spaces, Proc. 24th VLDB Conf., (1998) 194-205
18. Weber, R., Zezula, P.: The theory and practice of similarity searches in high dimensional data spaces (extended abstract), 4th DELOS Workshop, 1997
19. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the distance: An efficient method to KNN processing, Proc. 26th VLDB Conf., (2001) 421-430