IEEE Advances in Digital Libraries, ADL 2000, Washington DC

Efficient Similarity Search in Digital Libraries

Christian Böhm Bernhard Braunmüller Hans-Peter Kriegel Matthias Schubert

Computer Science Institute, University of Munich
E-mail: {boehm, braunmue, kriegel, schubera}@dbs.informatik.uni-muenchen.de

Abstract

Digital libraries are a core information technology. When the stored data is complex, e.g. high-resolution images or molecular protein structures, simple query types like the exact match query are hardly applicable. In such environments similarity queries, particularly range queries and k-nearest neighbor queries, turn out to be important query types. Numerous approaches have been proposed for the processing of similarity queries which mainly concentrate on highly dynamic data sets where insertion, update, and deletion operations permanently occur. However, only little effort has been devoted to the case of rather static data sets - a case that frequently occurs in digital libraries. In this paper, we introduce a novel technique for efficient similarity search on top of static or rarely changing data sets. In particular, we propose a special sorting order on the data objects which can be effectively exploited to substantially reduce the total query time of similarity queries. An extensive experimental evaluation with real-world data sets emphasizes the practical impact of our technique.

1. Introduction

In recent years, *digital libraries* have become a core information technology [10, 17]. Among the various important aspects of digital libraries the search for *similar* objects in the huge amount of digitized data has become an essential task. The QBIC system, for instance, contains a large image library which can be effectively searched for similar images by using *similarity queries* [22, 9]. The Brookhaven Protein Data Bank currently provides the atomic coordinates of several thousand proteins [2]. Here, the solution of the important molecular docking problem is supported by applying similarity queries on docking segments [19]. Another example are industrial CAD repositories which can effectively be used to reduce the cost of developing and producing new parts by maximizing the reuse of existing parts [21, 4]. Again, similarity queries are the most important query type in this process. Several other examples exist, e.g. geographical repositories, image collections and medical libraries.

The most common approach for efficient similarity search is to map data objects into some high-dimensional vector space (the so-called *feature space*). Similarity between two data objects is assumed to correspond to the distance of their feature vectors. Thus, searching for similar objects to a given query object is transformed into the problem of finding feature vectors which are *close* to the query feature vector. Popular examples of feature vectors are color histograms [8, 26], shape descriptors [15, 13], Fourier vectors [11, 1] and multi-parametric surface functions [18].

Typically, *range queries* and *k-nearest neighbor (k-nn) queries* are applied for the process of finding feature vectors which are close to a given query feature vector. Range queries retrieve all feature objects within a given radius ε from the query feature vector. For k-nn queries, the user provides a number k and receives the k feature vectors which are closest to the query vector. In general, a similarity query is a CPU and I/O intensive task and the conventional approach to address this problem is to use some multidimensional index structure [25, 12]. Unfortunately, even specialized index structures like the TV-tree [20] or the X-tree [6] often fail to process similarity queries efficiently when the dimensionality d of the feature space is too high ($d > 10$). However, this is a frequently encountered situation since the dimensionality of the feature space directly corresponds to the accuracy of the similarity search. In such environments, scan-based techniques like the VA-file [27] provide the most efficient query processing. These approaches mainly consider the case of highly dynamic environments with frequent insertion, update, and deletion operations. Only little effort has been devoted to the important case of static data sets. Obviously, static data sets provide more optimization potential than highly dynamic data sets. However, this optimization potential is not exploited by techniques which were developed for dynamic environments.

In this paper, we concentrate on the situation of static or rarely changing data sets (where the data set is known in advance) as they frequently occur in digital libraries. In particular, we introduce a novel technique (the *landmark file*) to significantly improve the query performance of scan approaches when storing static data sets. The main idea of the landmark file is to introduce a special order on the feature objects which can be exploited to substantially reduce the total amount of data which has to be scanned during the similarity search process.

The rest of our paper is organized as follows. In section 2, we review the problem of similarity search in the context of index-based and scan-based access methods. We present our approach in section 3 and section 4. We performed an extensive experimental evaluation in order to show the efficiency of our approach. The results are presented in section 5 and section 6 concludes the paper.

2. Similarity search in feature spaces

Range queries and *k*-nn queries are the most important similarity queries [7]. In the following, we formally define both query types.

Definition 1: *Range Query*

Let *DB* denote a set of feature vectors $v \in \Re^d$ and let $dist: \Re^d \times \Re^d \to \Re_0^+$ denote a distance function that measures the (dis-) similarity of two feature vectors $v_1, v_2 \in \Re^d$. Then, for a query feature vector $q \in \Re^d$ and a query range $\varepsilon \in \Re_0^+$, the range query returns the set

$$RQ(q, \varepsilon) = \{v \in DB \mid dist(q, v) \le \varepsilon\}$$

Definition 2: *k-Nearest Neighbor Query*

For a query feature vector $q \in \Re^d$ and a query parameter $k \ge 1$, the *k*-nearest neighbor query returns the set $NN_q(k) \subseteq DB$ that contains *k* feature vectors from *DB*, and for which the following condition holds:

$$\forall v \in NN_q(k), \forall v' \in DB - NN_q(k) : dist(q, v) \le dist(q, v')$$

Note that possibly several feature vectors exist which have the same distance to the query vector as the *k*-th feature vector in the answer set. In this case, the *k*-th feature vector in $NN_q(k)$ is a non-deterministic selection of one of those equally distanced feature vectors. Several distance functions for measuring the (dis-) similarity have been discussed. The Euclidean distance metric L_2, for instance, is one of the most frequently used similarity distance function, e.g. in the area of images, protein structures, CAD objects, or stock data.

Many sophisticated index structures have been proposed for efficiently processing similarity queries, but, only few techniques consider the case of static feature sets. In [24] a compaction technique for "packing" and reducing dead-space on R-trees [14] has been proposed for static pictorial feature sets. Other approaches follow the concept of bulk-loading when the feature set is completely known in advance. The Hilbert R-tree [16], for instance, uses the Hilbert space-filling curve to decompose the feature set into contiguous sequences which are stored in the data pages. In [5] a variant of the well-known Quicksort algorithm is used for a generic bulk loading method. Since index structures like the R-tree have been developed for low-dimensional feature spaces ($d < 5$) they offer only poor query performance for high-dimensional feature sets. In recent years, this problem has been addressed by developing high-dimensional index structures. The TV-tree [20], for example, uses so-called Telescope Vectors, i.e. feature vectors which may be dynamically shortened. The underlying assumption is that only dimensions with high variance are important for the query processing and therefore feature values of dimensions with low variance can be neglected. Another example is the X-tree [6] which uses the concept of directory supernodes. Whenever the split of a directory node would lead to a high overlap of the resulting nodes or to overlap minimal but extremely unbalanced nodes, the overflowing node is transformed into a supernode, i.e. a node with a larger than usual block size.

The main advantage of index-based access methods is the index selectivity during query processing, i.e. only a small fraction of the feature vectors has to be considered. This, however, induces costly random seek operations since the accessed index pages are generally not stored in contiguous disk blocks.

While these high-dimensional indexing techniques perform well for dimensions up to 10, even their performance often degenerates for higher dimensions. The reason for this effect is the so-called *curse of dimensionality*: Most of the measures one could define in a *d*-dimensional vector space, such as volume, area, or perimeter are exponentially dependent on the dimensionality of the space. Due to this effect, scan-based techniques, particularly the VA-file [27], turn out to provide better query performance when accessing high-dimensional feature spaces. The VA-file follows the idea of vector quantization. The feature space is divided into grid cells using α-quantiles in each dimension. Each feature vector is then represented by the address of the grid cell in which the feature vector lies. The main advantage of scan-based techniques is the sequential nature of their I/O operations which results in the absence of expensive random seeks. However, there is no selectivity in the query process involved and therefore all feature vectors have to be considered. Additionally, to the best of our knowledge, no optimization techniques have been proposed for scan-based techniques when the stored feature set is static.

3. The landmark file

As we have discussed in section 2, both query processing paradigms, indexing and sequentially scanning, have advantages and disadvantages. Our solution combines the advantages of both approaches and avoids their disadvantages. It adopts the idea of the scan, to read only from a single, contiguous interval of the file and avoids uncontrolled random seek operations. But our solution does not read and process the complete feature set. Thus, our solution inherits the selectivity from the indexing approach.

For static data sets, we can achieve this goal by keeping the feature set in a sort order according to an appropriate

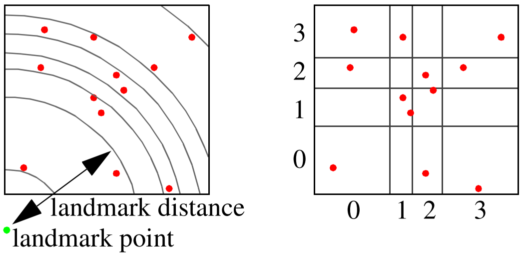

Figure 1. Landmark approx. (l.) and quantization (r.)

sorting key. For each query, the query processing algorithm determines the lower and upper bound of the sorting key and performs a sequential scan of the corresponding interval of the feature file. To choose the sorting key of the feature vectors, there are several possibilities, such as the projection to a single dimension. To assess the quality of a sorting key, recall the general objective: a substantial reduction of the number of scanned feature vectors. The interval into which the query is translated has to contain few feature vectors. In other words, the sorting key must distinguish as much as possible between different feature vectors. Obviously, the projection to a single dimension could fail to reach such a high distinguishing power. Therefore, our sorting key considers all relevant dimensions. A general approach is to use the Euclidean distance between a selected reference point and the feature vectors. If, for instance, the origin is selected as reference point, the sorting key corresponds to the sum of the squared dimension values. We will show in section 4, how an optimal reference point can be chosen.

3.1 Structure of the landmark file

In the following we show how the idea of sorting the feature vectors can be applied to establish selectivity in scan-based query processing. A point of the feature space (not necessarily contained in the data set) is chosen as reference point. This point is called the *landmark point* (cf. figure 1 left). The feature vectors are sorted in ascending distance to the landmark point (the sorting key is called the *landmark distance*).

We keep three representations of the feature vector file. In the *exact representation*, we store the full geometry of all feature vectors, i.e. the floating point values of the dimensions. In the *compressed representation*, we adopt the idea of the VA-file and store a quantized version of all feature vectors, i.e. the address (number) of the grid cell in which the corresponding feature vector lies (typically a few bits per dimension, cf. figure 1 right). The compressed representation requires substantially less space on secondary storage, and, therefore, the I/O cost compared to the exact representation is approximately reduced by the *compression factor*. On the other hand, the position of a feature vector is not exactly known such that a single access to the exact representation may be necessary.

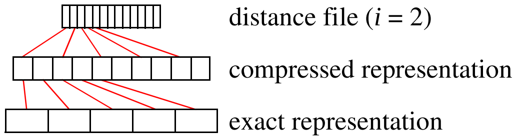

Figure 2. The landmark file

The third representation of the feature vector file is the *distance file*. This file does not contain information about each feature vector of the data set, but only for one vector out of i where i is a user-provided parameter, the *chunk parameter*. The landmark distance of each i-th vector of the ordered feature set is stored. This distance file partitions the feature space into spherical shells which contain a constant number i of feature vectors. The *landmark shells* for $i = 2$ are depicted in figure 1 left. There are no explicit pointers or links between the different representations of the files. As the sort order of the feature vectors is identical in all three representations, and since the length of each feature vector representation is constant, this reference is implicitly given by the position of the feature vector in the file (cf. figure 2).

Every query processing algorithm first considers the distance file and determines the landmark shells which potentially include query results. The corresponding part of the file with the compressed representation is scanned and candidates are collected. For every candidate which cannot be definitely classified as a query result by the compressed representation, a lookup to the exact representation is performed.

To process rare insertions, deletions, and updates we propose to store new or updated points in a separate file which is scanned sequentially without any optimizations for query processing. Whenever this overflow area becomes too large, the complete landmark file is reconstructed from scratch.

3.2 Query processing using landmarks

When the region of the query is exactly known in advance (such as in range queries), then query processing is straightforward:

- The distance file is loaded.
- Using the distance file, those shells are determined which are intersected by the query (cf. figure 3).
- For the intersected shells, the compressed representation is scanned.
- If necessary, the exact representation is accessed for

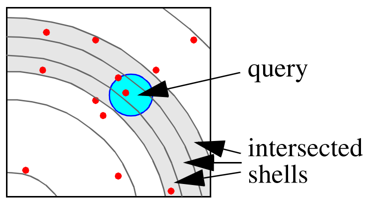

Figure 3. Range queries on the landmark file

each feature vector for which a decision cannot be made using the compressed representation only.

For (1-) nearest neighbor queries, the query region is not known in advance. Such queries are usually evaluated by increasing the radius of the query sphere until the nearest neighbor is covered by the query sphere. We adapt this algorithm for the landmark file (cf. figure 4):

- The distance file is loaded.
- Using the distance file, the shell containing the query vector is determined. This is the start shell.
- Beginning with the start shell, the unprocessed shell which is closest to the query vector is scanned successively. The scanned part of the compressed representation is extended, alternating at its upper and lower end. Whenever a feature vector is found which is closer than the current candidate, it is stored in a variable.
- This step is repeated until the distance between the next shell and the query vector is larger than the distance to the current candidate.

It is straightforward to extend this algorithm for *k*-nn search: Instead of a single candidate, the algorithm has to maintain a list with *k* candidates, and the distance to the last candidate is used as the stopping condition. For both query types, the algorithm must perform lookups to the exact representation in tie situations.

3.3 Parameter optimization

The chunk parameter *i*, which determines the size of the blocks for scanning in the nearest neighbor algorithm, is obviously a critical parameter. If it is chosen too small, the compressed representation is scanned in too small portions, which induces non-negligible I/O overhead. In contrast, if it is chosen too large, many compressed feature vectors may be scanned unnecessarily. To minimize the query processing time, the parameter *i* must be optimized.

Optimization problems like this are often solved by cost models such as [3]. In this special case, however, the estimation of the cost may be difficult because the intersection volumes between sphere shells and the query sphere must be computed. Therefore, we base our optimization on statistical information from previous runs of the algorithm or from tentative runs on a sample.

The information we are gathering is the means μ and the standard deviation σ of the number a of feature vectors with

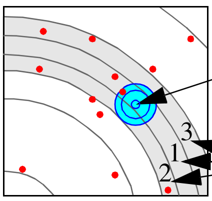

Figure 4. Nearest neighbor queries

a landmark distance in the range from $L(q) - r$ to $L(q) + r$, where $L(q)$ is the landmark distance of the query vector and r is the distance of the nearest neighbor of q. The number a corresponds to the number of vectors (= the number of shells) which must be processed by a query when $i = 1$.

For a known parameter a, we obtain the following cost function:

$$t(i, a) = (a + i) \cdot t_{tr} + \left(\frac{a}{i} + 1\right) \cdot (t_{pos} + t_{lat}) \qquad (1)$$

where t_{tr} is the transfer time per compressed feature vector, t_{pos} is the positioning time and t_{lat} is the latency time (rotational delay) of the disk drive. The first term of $t(i,a)$ indicates, that a compressed feature vectors are scanned during query processing, with an additional overhead of one block (i compressed feature vectors), because, on the average, half a block too much is scanned at the lower and the upper end of the scanned interval, respectively. In the second term of $t(i,a)$, the fraction $a/i + 1$ represents the number of separate I/O requests during query processing, for each of which a seek operation with arm positioning and rotational delay is required. This cost $t(i,a)$ can be minimized by setting the derivative to 0:

$$\frac{\partial}{\partial i} t(i, a) = 0 \Rightarrow i_{opt} = \pm\sqrt{a \cdot \frac{t_{pos} + t_{lat}}{t_{tr}}} \qquad (2)$$

Only the positive solution is valid, and it corresponds to a minimum, because the second derivative is (constantly) positive.

This solution optimizes the chunk parameter i for a known parameter a. However, this parameter is not constant for all queries. Therefore, we have to optimize i for a variety of different parameters a occurring in different queries. It turned out in our experiments that a is normally distributed. Under this assumption we can extend our model to optimize i for an a which is normally distributed with means μ and standard deviation σ:

$$\tilde{t}(i) = \int_0^{\infty} \left(\frac{1}{\sigma\sqrt{2\pi}} \mathbf{e}^{\frac{-(a-\mu)^2}{2\sigma^2}} \cdot t(i, a) \right) \mathbf{d}a \qquad (3)$$

This formula assigns to each cost $t(i,a)$ the probability of the corresponding a. The average over all possible a can be determined by integration with a ranging from 0 to infinity. In eq. (3), i can again be optimized by setting the derivative $\partial\tilde{t}(i)/\partial i$ to zero:

$$i_{opt} = \sqrt{\mu \cdot \frac{t_{pos} + t_{lat}}{t_{tr}}} \qquad (4)$$

The result is independent of the variance σ^2 and equal to the result of eq. (2) for a known a, where a is replaced by the means μ.

4. Determining the landmark point

In this section, we show how a suitable landmark point can be chosen in order to further optimize query processing using the landmark file. Figure 5 shows a feature set which is skewed from northwest to southeast. On the left side, we choose the landmark point L_1 which is close to the origin. Considering the landmark distances of the query point, we recognize a very low variance. Even for a minimal chunk parameter $i = 1$, the landmark shells which are intersected by the depicted query, cover almost the complete feature set (up to 2 points) On the right side of figure 5, in contrast, the landmark point L_2 is chosen such that it is somewhere on the axis of the major skew direction of the feature set in the southeastern corner of the feature space. We recognize a high variance of the landmark distances and the landmark shells intersected by the query region cover only few points for reasonable values of i (for the minimum chunk size i=1 the query region would only cover one vector, the actual nearest neighbor). Obviously, the variance of the landmark distances is inherited from the variance of the feature set: Whenever the landmark has a position on an axis of high variance of the feature set, the variance of the landmark distances is high.

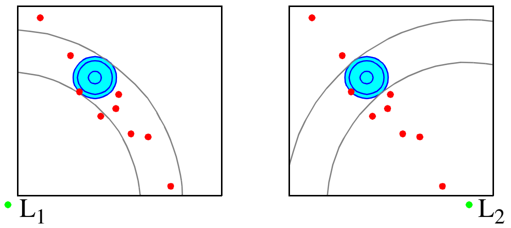

Figure 5. A good (L_2) and a bad (L_1) landmark point

This is captured mathematically by the concept of the principal component analysis (PCA) which finds the direction, where a feature set exhibits the highest variance [23]. For PCA, the means vector $\bar{x} = 1/N \sum_{0 \le i \le N} x_i$ and the covariance matrix $C = 1/N \sum_{0 \le i \le N} (x_i - \bar{x})(x_i - \bar{x})^T$ are determined, and C is diagonalized by some orthonormal matrix Φ, i.e. $\Phi^T C \Phi = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$. The matrix Φ can be determined by singular value decomposition [23], the λ_i are the eigenvalues and the column vectors of $\Phi = [\varphi_1, \ldots, \varphi_d]$ are the eigenvectors of C. The eigenvector φ_i corresponding to the largest eigenvalue λ_i indicates the direction of the highest variance of the feature set. Therefore, the landmark point should be chosen from the straight line $g = \bar{x} + \nu \cdot \varphi_i$. To maximize the variance of the landmark distances, the landmark point should not be chosen inside the feature space.

5. Experimental evaluation

In order to demonstrate the practical relevance of our technique we performed an extensive experimental evaluation using the following real-world feature sets:

- *CAD repository*: 16-dimensional Fourier points corresponding to contours of 1,200,000 industrial parts used in the S3-System [4].
- *Image Collection*: 64-dimensional color histograms corresponding to 112,000 images from TV-snapshots.

The CAD repository we use is a collection of contours of industrial parts (e.g. clips holding cables) from a supplier for automobile companies. The repository is used to reduce the cost of developing and producing new parts by maximizing the reuse of existing parts. This is achieved by applying similarity queries, in particular k-nn queries. The repository can be seen as static since there are no update or deletion operations and very few insertion operations (less than 5 per month). The image collection is comparable to other image collections, e.g. the *Corel image collection*. Each image was reduced to 64 colors and the numbers of pixels per color in an image build the 64-dimensional feature vector. Color histograms are successfully used for similarity search in image libraries, e.g. [22].

In the experimental evaluation, we compare our technique to the VA-file which has been shown to offer excellent performance for high-dimensional feature sets (cf. section 2). We focus on k-nn queries since they have generally more practical impact. The reason is that from a user's point of view range queries are often difficult to apply when a reasonable value for ε is hard to determine. Additionally, the total query time of a k-nn query is an upper bound for a range query with an ε value that is equal to the distance of the k-th nearest neighbor. Recall, the landmark file has no additional seek cost for range queries since it scans the intersected shells in one pass and does not alternate the scanning direction as in the case of k-nn queries. The overall overhead for performing k-nn queries compared to range queries is therefore generally higher for the landmark file than for the VA-file. Thus, we can safely assume that the performance gain of the landmark file compared to the VA-file is even higher

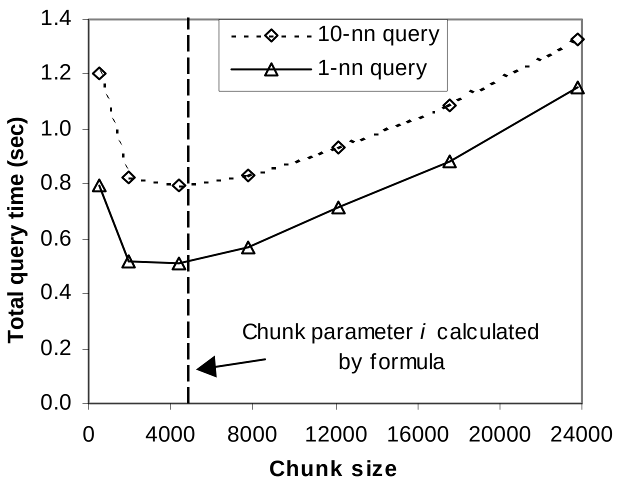

Figure 6. Optimal chunk parameter *i*, image collection

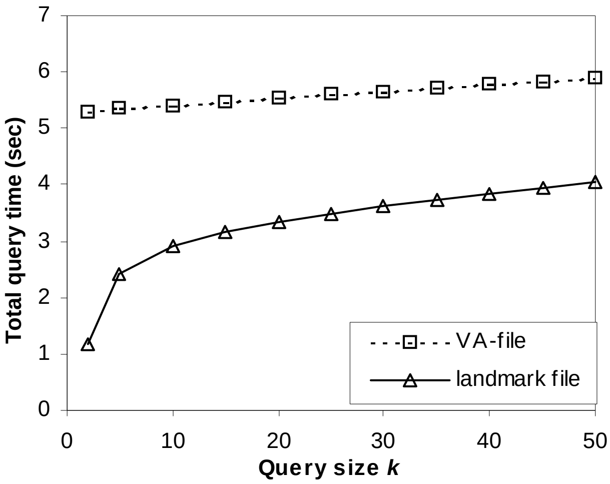

Figure 7. Query size on the CAD repository

for range queries. We performed all experiments on a HP C160 under HP-UX 10.20. The implementation language for our technique and the VA-file was C++.

In our first experiment, we evaluate our solution for the important problem of optimizing the chunk parameter i (cf. section 3.3). We performed 1-nn queries and 10-nn queries and varied the chunk size from 500 to 24,000 points using the image collection. The total query times with respect to the chunk size are depicted in figure 6. Obviously, there exists an optimum for the chunk size for both query parameters which lies at approximately $i_{opt} = 4350$. Applying our formula derived in section 3.3, we find our chunk parameter $i = 4380$ almost perfectly matching. We observed the same effect when using the CAD repository (not depicted). Therefore, we applied our formula for the choice of the chunk parameter i in all following experiments.

Next, we investigate the performance of the landmark file and the VA-file with respect to the query parameter k. We increased k from 1 to 50 and measured the total query times. Figure 7 depicts the results for the CAD repository. We observe that the landmark file clearly outperforms the VA-file for all values of k. For small parameters, e.g. for $k = 1$, the landmark file exhibits only 22% of the query time of the VA-file. When k increases, the performance gain of the landmark file decreases. The reason is that with increasing query size, the distance to the k-th nearest neighbor increases and thus more and more landmark shells are intersected. However, even for high values, e.g. for $k = 50$, our approach saves 31% of the query time compared to the VA-file.

We performed the same experiment on the image collection (cf. figure 8). Again, our approach substantially outperforms the VA-file for all values of k and saves between 45% and 73% of the total query time of the VA-file.

As we have discussed in section 4, the selection of the landmark point is a critical task. In order to evaluate our solution for this problem, we additionally performed k-nn queries on the image collection for randomly selected landmark points. In particular, we measured the total query times of the landmark file using 10 random landmark points and by calculating the means we determined the average total query time as depicted in figure 8 (denoted as "lm file (random lm. pt.)"). We observe that the query performance of the landmark file substantially increases when using the formally derived landmark point compared to using a random landmark point. For $k = 10$, for instance, the total query time of the landmark file using a random landmark point is 2.22 times the total query time using our formally derived landmark point. Another important observation is the fact that even with a random landmark point the landmark file consistently outperforms the VA-file.

Finally, we investigate the performance of our approach with respect to the size N of the feature set. We used the CAD repository and increased the number N of feature vectors from 100,000 to 1,200,000. We measured the total query time of 1-nn, 10-nn, and 50-nn queries for both access methods. The results are presented in figure 9. For small values of N, the landmark file and the VA-file perform almost equally. When the size of the feature set increases, however, this situation changes drastically. For $N = 400{,}000$ and $k = 10$, for instance, the landmark file already saves 33% of the total query time compared to the VA-file and this ratio increases up to 46% for $N = 1{,}200{,}000$. For $k = 1$ and the complete CAD repository, the landmark file even saves 78% of the total query time of the VA-file, which corresponds to a speed-up factor of 4.53.

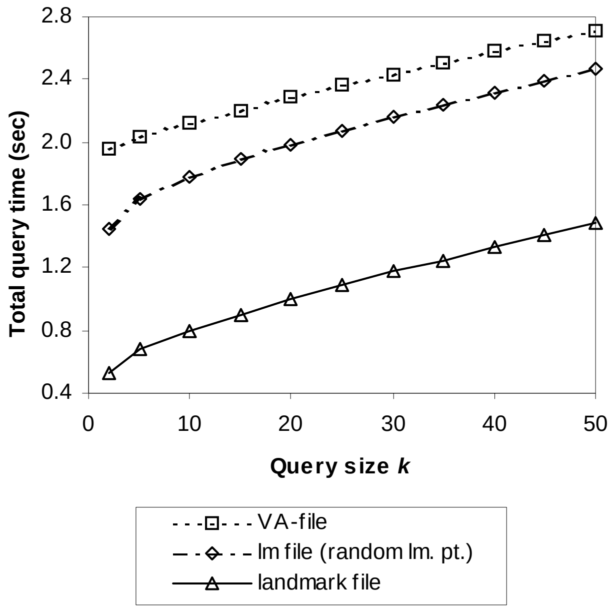

Figure 8. Query size on the image collection

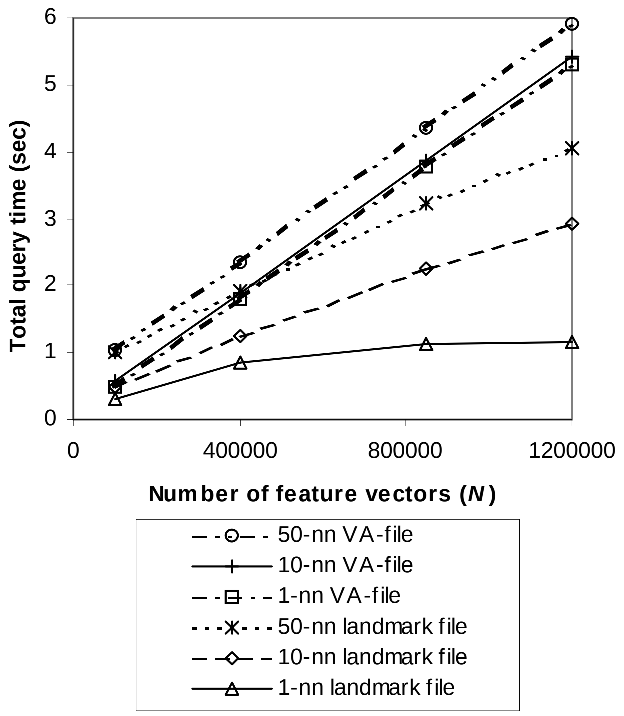

Figure 9. Size of the CAD repository

6. Conclusions

Similarity queries, particularly *range queries* and *k-nearest neighbor* queries, are important query types in digital libraries. In this paper, we have studied the problem of efficiently processing similarity queries in static or rarely changing high-dimensional data sets as they frequently occur in digital libraries. We propose a new technique, the *landmark file*, which uses a special sorting order on the feature objects to efficiently process similarity queries. In an extensive experimental evaluation we demonstrated that our approach consistently outperforms the VA-file which is the most efficient scan-based access method for high-dimensional data sets proposed so far. For future work, we plan to extend our technique to parallel and distributed environments.

7. References

[1] Agrawal R., Lin K.-I., Sawhney H. S., Shim K.: *'Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases',* Proc. Int. Conf. on Very Large Data Bases, 1995.

[2] Bernstein F. C., Koetzle T. F., Williams G. J., Meyer E. F., Brice M. D., Rodgers J. R., Kennard O., Shimanovichi T., Tasumi M.: *'The Protein Data Bank: a Computer-based Archival File for Macromolecular Structures',* Journal of Molecular Biology, Vol. 112, 1977, 535-542.

[3] Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *'A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces',* Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 78-86.

[4] Berchtold S., Kriegel H.-P.: *'S3: Similarity Search in CAD Database Systems',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 564-567.

[5] Böhm C., Kriegel H.-P.: *'Efficient Bulk Loading of Large High-Dimensional Indexes'*, Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 1999, pp. 251-260.

[6] Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data',* Proc. Int. Conf. on Very Large Data Bases, 1996, pp. 28-39.

[7] Böhm C.: *'Efficient Indexing High-Dimensional Data Spaces',* Ph.D. thesis, University of Munich, 1998.

[8] Faloutsos C., Barber R., Flickner M., Hafner J., et al.: *'Efficient and Effective Querying by Image Content',* Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.

[9] Flickner M., Sawhney H., Niblack W., Ashley J., Huang Q., Dom B., Gorkani M., Hafner J., Lee D., Petkovic D., Steele D., Yanker P.: *'Query by image and video content: The QBIC system'*, Computer, Vol. 28, September 1995, pp. 23-32.

[10] Fox E. A., Akscyn R. M., Furuta R. K., Leggett J. J.: *'Digital Libraries-Introduction to the Special Section'*, Communications of the ACM, 38(4), 1995, pp. 22-28.

[11] Faloutsos C., Ranganathan M., Manolopoulos Y.: *'Fast Subsequence Matching in Time-Series Databases',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 419-429.

[12] Gaede V., Günther O.:*'Multidimensional Access Methods'*, ACM Computing Surveys, Vol. 30, No. 2, 1998, pp.170-231.

[13] Gary J. E., Mehrotra R.: *'Similar Shape Retrieval using a Structural Feature Index',* Information Systems, Vol. 18, No. 7, 1993.

[14] Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, 47-57.

[15] Jagadish H. V.: *'A Retrieval Technique for Similar Shapes',* Proc. ACM SIGMOD Int. Conf. on Managem. of Data, 1991, 208-217.

[16] Kamel, Faloutsos: *'Hilbert R-tree: An Improved R-tree using Fractals'*. Proc. Int. Conf. on Very Large Data Bases, 1994, 500-509.

[17] Klavans J.: *'Data Bases in Digital Libraries: Where Computer Science and Information Management Meet'*, Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1998, 224-226.

[18] Kriegel H.-P., Seidl T. : *'Approximation-Based Similarity Search for 3-D Surface Segments'*, GeoInformatica Int. Journal on Advances of Computer Science for Geographic Information Systems, 1998, Vol.2, No.2, pp.113-147.

[19] Kriegel H.-P., Schmidt T., Seidl T.: *'3D Similarity Search by Shape Approximation'*, Proc. 5th Int. Symposium on Large Spatial Databases, 1997, pp. 11-28.

[20] Lin K., Jagadish H. V., Faloutsos C.: *'The TV-tree: An Index Structure for High-Dimensional Data',* Journal of Very Large Data Bases, Vol. 3, pp. 517-542, 1994.

[21] Mehrotra R., Gary J. E.: *'Feature-Based Retrieval of Similar Shapes',* Proc. Int. Conf. on Data Engineering, 1993, pp. 108-115.

[22] Niblack W., Barber R., Equitz W., Flickner M., Glasmann E., Petkovic D., Yanker P., Faloutsos C., Taubin G.: *'The QBIC Project: Querying Images by Content Using Color, Texture, and Shape',* Proc. Int. Symposium on Electronic Imaging, 1993.

[23] Press H. W., Teukolsky A. S., Vetterling T. W., Flannery P. B.: *'Numerical Recipes in C : The Art of Scientific Computing'*, Cambridge Univ. Press, 2nd ed., 1995.

[24] Roussopoulos N., Leifker D.: *'Direct Spatial Search on Pictorial Databases Using Packed R-Trees'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp.17-31

[25] Samet H.: '*The Design and Analysis of Spatial Data Structures*', Addison-Wesley, 1989.

[26] Seidl T., Kriegel H.-P.: '*Efficient User-Adaptable Similarity Search in Large Multimedia Databases',* Proc. Int. Conf. on Very Large Data Bases, 1997, pp. 506-515.

[27] Weber R., Schek H.-J., Blott S.: '*A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces*', Proc. Int. Conf. on Very Large Data Bases, 1998, 194-205.