in Proc. 18th Int. Conf. on Scientific and Statistical Database Management (SSDBM 06), Wien, Austria, 2006

Efficient Query Processing in Arbitrary Subspaces Using Vector Approximations

Hans-Peter Kriegel, Peer Kröger, Matthias Schubert, Ziyue Zhu
Institute for Informatics, University of Munich, Germany
{kriegel,kroegerp,schubert,ziyue}@dbs.ifi.lmu.de

Abstract

In this paper, we introduce the partial vector approximation file, an extension of the well known vector approximation file that is constructed to efficiently answer partial similarity queries in any possible subspace which is not known beforehand. The idea of the partial VA-File is to divide the VA-File into a separate file for each dimension and only load the dimensions that are necessary to answer the query. Thus, the partial VA-File is constructed to improve the query performance for systems that have to cope with a wide variety of previously unknown query subspaces. We propose novel algorithms for partial kNN and ε-range queries based on the new partial VA-File. In our experiments, we demonstrate that our proposed partial VA-File with the novel algorithms improves the average query performance in comparison to the original VA-File when answering partial similarity queries.

1. Introduction

Indexing large sets of high-dimensional feature vectors is an important issue in many database applications. In recent years, many applications evolved that require *partial* similarity queries. A partial similarity query is a similarity query where only a subset of the existing attributes of the feature space is relevant, i.e. the query only specifies a subset of the attributes and is posed against the corresponding projection of the entire feature space. The projection specified for a query is usually not known beforehand and is subject to changes.

Established index structures are designed and optimized for the full dimensional space, i.e. the feature space must be fixed before index construction. However, if only a small subset of the attributes is relevant for the query, the performance of the index structures degenerates because of a very poor pruning power. A solution to this problem might be to use an optimized index structure for each query. However, since the relevant subset of attributes for the query is not known in advance an optimized index structure for each possible attribute subset would be necessary. Obviously, this approach is unrealistic even for a moderate dimensionality. In this paper, we propose such a novel approach to efficiently support partial range queries and partial k-nearest neighbor queries in arbitrary subspaces. In a broad experimental evaluation, we show the applicability of our methods for efficiently answering partial range and partial k-nearest neighbor queries in arbitrary subspaces.

2. Related Work

Spatial index structures (e.g. [7, 2, 4, 8]) are optimized for the entire feature space but are usually not applicable to partial similarity queries because the larger the difference between the dimensionality of the query subspace and the dimensionality of the feature space is, the more the performance of the indexes will decrease.

Tree-striping [3] uses several index structures to index different subspaces of the feature space rather than one full-dimensional index. The choice of the subspaces is data dependent and must be fixed before the indexes are built. Thus, tree striping has similar problems compared to full-dimensional indexes.

Inverted files [6] index each attribute separately using a 1D index such as B+-Trees [5]. A partial similarity query is passed to all 1D indexes that are relevant for the query subspace. The result is aggregated from the corresponding indexes.

The Vector Approximation File (VA-File) [9] is a data structure for efficient similarity search in high-dimensional feature vectors. Instead of a tree like data structure the VA-File uses a simple but efficient filter-refinement approach to save I/O cost. The actual VA-File is a sequential file containing so-called "vector approximations" for each feature vector. These vector approximations are generated by laying a grid of 2^b partitions per dimension over the feature space. Thus, we can approximate the coordinate v_d of a feature vector v in dimension d by the partition p the coordinate is placed in. Since there are only 2^b partitions in each dimension, this partition can be described by b bits. The complete vector approximation of a feature vector has

a size of $d \cdot b$ bits instead of $d \cdot sizeof(v_d)$. Since in modern applications numeric values are usually represented by 32 bit float values or even 64 bit double values and since the number of bits b to encode a partition is about 4 to 8 bits, a vector approximation is between 16 and 4 times smaller than an exact feature vector. Thus, the VA-File, containing the bit encoded vector approximations, is much smaller and much faster to read from disk than the file containing the complete feature vector.

3. The Partial Vector Approximation File

Principally, the VA file is applicable without further modification to answer similarity queries in arbitrary subspaces as well. The only difference is that the distances need to be calculated on the basis of the queried subspace instead of using all dimensions. However, since we always have to load the complete VA-File regardless of the query subspace, large parts of the loaded information may be useless. For example, in the case of a query in a 25 dimensional subspace being part of a 100 dimensional feature space, 75% of the information is loaded from disk but not used to answer the query.

The idea of the partial VA-File is to store each dimension i of a vector approximation in a separate file F_i rather than storing all vector approximations in a single file. When answering partial similarity queries, only the necessary dimensions are loaded. Thus, the transfer volume can be reduced to the necessary information. On the other hand, we have to access each dimension separately causing several additional disk accesses. In the following, we will introduce efficient algorithms on the partial VA-File that are capable to considerably speed up partial similarity queries despite of these additional disk accesses.

3.1. Building the Partial VA-File

The partial VA-File consists of d approximation files F_i ($1 \leq i \leq d$), called dimensional A-Files (DA-Files), containing the approximation of the data objects for one particular attribute $a_i \in \mathcal{A}$ only and a sequential file containing the exact feature vectors. We assume that all feature vectors are ordered in the same way in each file. To build the partial VA-File, we start by calculating approximations for the given set of feature vectors. However, instead of using a uniformly distributed grid as the original VA-File does, we adopt the grid to the data distribution by using the algorithm introduced in [1]. As a result, the number of data objects in each partition is approximately the same. We will use the properties of this adaptable grid in the following query algorithms. After constructing our grid, we can now determine the vector approximations in each dimension and store them in the d DA-Files.

3.2. Answering PεRQs

To answer pε-RQs, we introduce a new algorithm that exploits the possibility to read single dimensions. For a given query point q, a range ε and subspace $S \subseteq \mathcal{A}$, we proceed using the following steps:

1. Order Dimensions: Rank the DA-Files with respect to their selectivity for the given query q.
2. Scan DA-Files (Filtering): Load the most selective DA-File and determine all candidate approximations for which the minimal distance to q is smaller than epsilon. Afterwards, load the next DA-File with respect to its ranking and refine the previously determined candidates using the additional dimension. Proceed in the same way until all dimensional VA-Files are processed.
3. Determine Results (Refinement): Determine all candidate approximations that are completely placed within the ε-range of q and add their IDs to the result. Refine the rest of the candidates using the exact representations.

To order the DA-Files in the first step of the algorithm by their selectivity, we use the adaptive grid, which we employed for deriving the vector approximations. Since this grid is built mirroring the distribution of the feature values of the corresponding dimension, the size of a partition indicates the sparseness of data values. A large partition indicates sparsity while a rather small partition indicates a dense area in the corresponding interval of feature values. Based on this observation, we define the *selectivity coefficient* $\sigma_q^\varepsilon(F)$ for DA-File F in the following way. Let $P = \{p_i, .., p_k\}$ be an ordered set of $k = 2^b + 1$ partition borders for the DA-File F. Then, the sparsity coefficient $\sigma_q^\varepsilon(F)$ of an ε-range query to the query point q is determined as following: $\sigma_q^\varepsilon(F) = |\{p_i | p_i \in P \wedge q - \varepsilon \leq p_i \leq q + \varepsilon\}|$.

Intuetively, the selectivity coefficient describes the number of partitions that are intersected by the query sphere. If the dimension is rather sparse in the query range, only a small number or even a single partition is intersected. As a result, the selectivity coefficient is rather small and the dimension yields high pruning power for the given query. Since, the effort for each additional dimension decreases with the number of remaining candidates, a highly selective dimension should be processed quite early. Therefore, we calculate the selectivity coefficient for each DA-File corresponding to any dimension of the query subspace and order the DA-Files ascending by their selectivity.

In the filter step of our algorithm, we now process each dimension in the calculated order. We first of all load the top-ranked DA-File. At the beginning of this step, all objects are potential candidates. Thus, we determine all approximations for which the minimal distance to q, i.e. the mindist, is smaller than ε. Note that the mindist depending on a single dimension is still a best-case estimation for the distance between the query q and the actual data object o

being represented by the approximation a_o. Thus, we can already prune the data objects having a larger mindist with respect to the first dimension. For each candidate, we save the mindist, the maxdist, i.e. the maximal distance to the query object q and its position in the DA-File. For any further iteration, the next DA-File is loaded. However, since we already could prune some of the data objects, we only increase the mindist for the remaining candidates by the values of the current DA-File. Correspondingly, we sum up the maxdist. Remember that we are using squared Euclidian distances and thus, both updates can be easily achieved by single plus-operations. With an increasing number of considered DA-Files, the set of candidates still having a mindist that is smaller than ε is rapidly decreasing. Thus, the number of distance updates decreases with each additional DA-File as well.

In the final refinement step, we test all remaining candidate approximations if their maxdist is larger than ε. If this is the case, the approximation is completely placed within the query sphere and the data object already belongs to the query result. Otherwise, the approximation could still represent a feature vector lying outside of the query sphere. Thus, we have to read the exact feature vector from disk and compare it to the query vector.

3.3. Answering PkNNQs

The second partial similarity query, we want to support, are pkNN queries. Note that this problem is more complicated than partial ε-range queries because the pruning power of a single dimension is very limited at the beginning of the algorithm. Most algorithms for kNN queries start with an infinite upper bound for the distance of the k-nearest neighbor to the query object q and then successively decrease this upper bound after reading some objects or approximations. For an ordinary VA-File, the k-nearest maxdist of any read approximation to q is an upper bound for the actual distance of the k-nearest neighbor and thus, this distance can be used to prune other approximations. For the partial VA-File this is a problem because we cannot determine a meaningful maxdist for a complete approximation on the basis of a single dimension. To estimate a maxdist without considering all dimensions, we have to assume a worst-case distance for all missing representations which is the distance of q to the farther rim of the data space in each dimension. Obviously, if the dimensions for which we have to use these worst-case estimates, are too numerous, the number of candidates is usually too big to fit into the main memory.

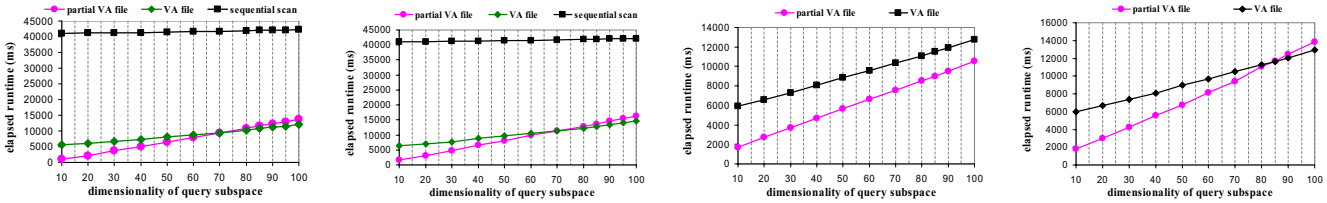
Thus, our algorithm starts with reading the first $2d/3$ DA-Files for a query over a d-dimensional subspace. In our experiments, it turned out that using an approximation in two thirds of the dimensions and estimating the remaining third, was already quite selective. A reason for this effect is that we again sort the dimensions by their potential pruning power. However, in the case of kNN queries the sorting criterion is not based on the selectivity of the dimension, instead we use the distance of the query object q to the farther rim of the data space, i.e. the worst case distance of the result. Thus, by ordering the dimensions by the worst case estimates they are contributing to the current pruning distance, we assure that dimensions that would add large worst-case estimates are read earlier or are contained within the first $2d/3$ dimensions that are read anyway.

Our kNN algorithm starts with reading the first $2d/3$ DA-Files. Afterwards, we determine the mindist and the maxdist of each approximation to the query object q. For the mindist, we simply have to add up the squared distances in each of the $2d/3$ dimensions. To calculate the maxdists, we have to additionally add the worst-case distance for each of the dimensions that were not processed yet. To determine the current pruning distance, we employ a priority queue storing the k smallest maxdists found so far. As in the kNN algorithm for the ordinary VA-File, an object can now be pruned if its mindist is larger than the k smallest maxdist found so far. After this initialization phase, we load one additional DA-File. For the remaining candidates, we now increase the mindist and maxdist using the approximations stored in the read DA-File. Additionally, we have to decrease the maxdist by the worst-case distance that was used to estimate the distance in this dimension. Let us note that the priority queue containing the k smallest maxdists is completely rebuilt for each dimension. The filter step is complete if the number of remaining candidates is small enough that refining these candidates would cause less I/O costs than reading the other DA-Files. If this is not the case, the filter steps ends after each DA-File of any query dimension was processed. In the case of pkNN queries this method works quite well because the set of candidates is usually very small for this kind of query.

After the filter phase is complete, the remaining candidates are ordered by their mindist. The k-nearest neighbors that are found so far are stored in a priority queue. The queue is constructed to place the object having the k smallest distances to q on the top position. After refining the first q objects, the top element of this priority queue can be used to determine whether refining additional candidates is still necessary. If the smallest mindist of all remaining candidates is greater or equal to the exact distance on top of the priority queue, the objects stored in the queue are the k-nearest neighbors and we can terminate the algorithm.

4. Evaluation

We evaluated the performance of the partial VA-File on a workstation featuring a 1.7 GHz CPU and 2 GB RAM.



(a) DS2, selectivity: 0.01%

(b) DS2, selectivity: 0.05%

(c) DS2, $k = 1$

(d) DS2, $k = 10$

Figure 1. Runtime w.r.t. dimensionality of query subspace on DS2.

We used a disk with a transfer rate of 45 MB/s, a seek time of 4 ms and a latency delay of 2 ms. We used two equally distributed synthetic data sets, DS1 (50D) and DS2 (100D), both containing 1 million tuples and a real world data set DS3 containing 112,361 tuples representing 64D color histograms of TV snapshots.

When evaluating partial range queries with selectivity values of 0.01% and 0.05% we observed that the partial VA-File as well as the original VA-File clearly outperfom the sequential scan on both data sets. In addition, it turned out that the lower the number of relevant attributes of the query, the higher is the performance advantage of the partial VA-File over the original VA-File. The break-even point is at around 70% relevant query attributes, i.e. for a query that specifies less than 70% of the features to be relevant, our partial VA-File yields a performance advantage over existing methods. Sample results on DS2 are shown in Figure 1.

In case of partial k-nearest neighbor queries we observed that for low values of k the partial VA-File is clearly better than the original VA-File even for full-dimensional queries. If the value of k increases, the performance of the partial VA-File slightly decreases. However, for $k = 10$, the break-even point for which the partial VA-file performs better than the original VA-file was observed at about 80%, i.e. if the dimensionality of the query subspace is at most 80% of the entire feature space, the partial VA-file outperforms the compared methods.

The results of our experiments on the real world dataset DS3 confirm the observations made on the synthetic datasets. We again observe that the sequential scan is clearly outperformed by the VA-File variants.

5. Conclusion

In this paper, we presented the partial VA-File, an index structure that is constructed to speed up similarity queries in arbitrary, *a priori* unknown subspaces of high dimensional feature spaces. We presented novel algorithms for partial kNN queries and partial range queries which have to load only the dimensions that are needed for a particular query and thus save a considerable amount of I/O time when querying low dimensional subspaces. In our experiments, we demonstrate that the partial VA-File is able to achieve a better average query performance over all possible subspaces compared to the original VA file and the sequential scan.

References

[1] R. Agrawal and A. Swami. A one-pass space-efficient algorithm for finding quantiles. IBM Almaden Research Center 650 Harry Road, San Jose, CA 95120, 1995.

[2] N. Beckmann, H.-P. Kriegel, B. Seeger, and R. Schneider. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'90), Atlantic City, NJ*, 1990.

[3] S. Berchtold, C. Böhm, D. A. Keim, H.-P. Kriegel, and X. Xu. "Optimal Multidimensional Query Processing Using Tree Striping". In *Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK 2000), Greenwich, U.K.*, 2000.

[4] S. Berchtold, D. A. Keim, and H.-P. Kriegel. "The X-Tree: An Index Structure for High-Dimensional Data". In *Proc. 22nd Int. Conf. on Very Large Databases (VLDB'96), Mumbai (Bombay), India*, 1996.

[5] D. Comer. "The Ubiquitous B-Tree". *ACM Computing Surveys*, 11(2):121–137, 1979.

[6] A. F. Cárdenas. "Analysis and Performance of Inverted Database Structures". *Communications of the ACM*, 18(5):253–263, 1975.

[7] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In *Proc. ACM SIGMOD Int. Conf on Management of Data (SIGMOD'84), Boston, MA*, 1984.

[8] N. Katayama and S. Satoh. "The SR-tree: An Index Structure High Dimensional Nearest Neighbor Queries". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'97), Tucson, AZ*, 1997.

[9] R. Weber, H.-J. Schek, and S. Blott. "Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces". In *Proc. 24th Int. Conf. on Very Large Databases (VLDB'98), New York, NY*, 1998.