case in Denmark which has no states). If such non-immediate links occur in the hierarchy, it is called a *non-covering* or *ragged* hierarchy [2,3], Finally, each non-top value has precisely one parent, e.g., a product must belong to exactly one product group. This may not always be desirable, e.g., it would be natural to put skimmed milk into both the "Diet" and "Dairy" product groups. If the hierarchies do not form balanced trees, this affects the so-called *summarizability* of the data, which means that special care must be taken to obtain correct aggregation results [1].

Cross-references

- ► Dimension
- ► Multidimensional Modeling
- ► On-Line Analytical Processing
- ► Summarizability

Recommended Reading

1. Lenz H. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 39–48.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, NY, 1997.

High Dimensional Indexing

Christian Böhm, Claudia Plant
University of Munich, Munich, Germany

Synonyms

Indexing for similarity search

Definition

The term High Dimensional Indexing [6,9] subsumes all techniques for indexing vector spaces addressing problems which are specific in the context of high dimensional data spaces, and all optimization techniques to improve index structures, and the algorithms for various variants of similarity search (nearest neighbor, reverse nearest neighbor queries, range queries, similarity joins etc.) for high dimensional spaces. The well-known Curse of Dimensionality leads to a worsening of the index selectivity with increasing dimensionality of the data space, an effect which already starts at dimensions of 10–15, also depending on the size of the database and the data distribution (clustering, attribute dependencies). During query processing, large parts of conventional hierarchical indexes (e.g., R-tree) need to be randomly accessed, which is by a factor of up to 20 more expensive than sequential reading operations. Therefore, specialized indexing techniques for high dimensional spaces include e.g. ideas to scale up sequential scans, hybrid approaches combining elements of hierarchical and scan-based indexes, dimensionality reduction and data compression.

Historical Background

One of the first indexing techniques which is nowadays wide spread in commercial databases is the B-tree. One dimensional data items, so-called keys, are stored in a hierarchical balanced search tree. Since the height of a B-tree is bounded above by $O(\log N)$, the index provides retrieval in logarithmic time complexity. Approaches to extend the B-tree to higher dimensions include the R-tree [12], which has been originally designed for two dimensional spatial objects (polygons). The R-tree consists of two types of nodes: directory and data pages. The directory pages contain rectangles (or hyper-rectangles for higher dimensional data) which are the minimum bounding rectangles (MBR) for the underlaying sub-trees, all the way down to the data pages at the leaves. The hyper-rectangles may overlap, and also the directory does not need to cover the whole data space. This implies that the search is not guaranteed to be restricted to one single path from the root to a leave as in the B-tree. However, since the R-tree is a balanced search tree and algorithms for tree construction and restructuring are designed to minimize overlap, search operations on low dimensional data can be performed in almost logarithmic time. Various variants of the R-tree, such as the R*-tree have been proposed.

Foundations

The central problem of similarity search in high dimensional spaces is the deterioration of the index selectivity with increasing dimensionality of the data space, an effect which is essentially inherent to any kind of index structure. Conventional hierarchical index structures achieve impressive performance gains over sequential scan on low to medium dimensional data by excluding large parts of the data contained in subtrees which do not need to be visited. Special properties

of high dimensional spaces, often subsumed by the term Curse of Dimensionality cause that conventional hierarchical indexes break down in performance on high dimensional data. With increasing dimensionality, more points are situated at the boundaries of the data space and the distances between points assimilate. During retrieval, large parts of the index have to be accessed. In the extreme case of very high dimensionality ($d \rightarrow \infty$) it is therefore obvious that no indexing technique is able to outperform the simplest processing technique for similarity queries, the sequential scan. This fact has been discussed in the context of cost models and of various indexing methods [3,7,19] and has also solicited a scientific discussion of the usefulness of similarity queries per se [6]. However, when, and to which extent the dimensionality curse occurs, depends on the size of the database and various other parameters such as the statistical distribution of the data, correlations between the single attributes (i.e. whether the complete data space is covered or the data reside in an arbitrarily complex subspace), and clustering (i.e. if there are clearly distinguishable groups of similar objects in the database). Many indexing methods can be successfully applied in moderately high dimensions, and many dedicated indexing methods are able to index data spaces of a dimensionality which is considerably higher than expected. In order to achieve the goal of efficiently indexing (moderately) high dimensional spaces, the well-known proposals to high-dimensional indexing all apply a combination of tricks which can be categorized into the following classes:

1. Dimensionality reduction
2. Data compression
3. Optimized i/o schedules (page size optimization and fast index scan)
4. Hierarchy flattening
5. Optimizing of the shape of page regions
6. Clustering

In the following, the most relevant approaches to High Dimensional Indexing are described in chronological order.

TV-Tree

High dimensional real data often exhibits a much lower intrinsic dimensionality. In this case, principal component analysis leads to a few highly loaded components almost totally covering the variance in the data. The TV-tree [16] exploits the fact that those top ranked components are highly selective dimensions for similarity search, whereas the remaining dimensions are of minor importance. Therefore, only the most selective components are used for pruning during query processing. Since irrelevant sub-trees should be excluded as early as possible, these components, called *active dimensions* are placed at the topmost levels of the index. A region of the TV-tree is described by a sphere in the active dimensions. The remaining dimensions may be active at lower levels of the index or not selective enough for query processing. The authors report a good speed-up in comparison with the R*-tree if the data can be effectively reduced by PCA, Fourier transform etc. On uniform or other real data not amenable to PCA, the X-tree outperforms the TV-tree.

The main contribution to High Dimensional Indexing is the implicit dimensionality reduction. Depending on the depth of the tree, only the first few dimensions are considered in the directory structure. Further, in these considered dimensions, the pages are approximated by bounding spheres which are more suitable for Euclidean queries.

SS-Tree

The SS-tree [20] also uses spheres instead of bounding rectangles as page regions. For efficiency, the spheres are not minimum bounding spheres. Rather, the centroid of the stored points is used as center for the sphere and the radius is chosen such that all objects are included. The region description consists of the centroid and the radius and thus allows efficient pruning. The SS-tree is suitable for all kinds of data distributions and outperforms the R*-tree by a factor of 2, which is mainly due to the fact that spherical page regions are, as mentioned, more suitable to support Euclidean queries. In addition, the algorithms for tree construction and maintenance are highly efficient and also very effective for low to medium dimensional data. On high dimensional data, the SS-tree encounters similar problems as the R-tree family. Compared to the minimum bounding rectangles (MBR) of R-trees, spherical page regions are even more difficult to split in an overlap-free way. This problem is tackled by the SR-tree [15] which uses a combination of a rectangle and a sphere as page region.

X-Tree

In contrast to the TV-tree and the SS-tree, the X-tree [5] is a high-dimensional index structure which is

more closely related to the R-tree, and, in particular, to the R*-tree. It is the first technique which introduces the ideas of flattening the hierarchies of the directory and of enlarging the sizes of directory pages, but in contrast to the techniques described in the next section, this idea was not inspired by a cost analysis but simply from the observation that it can be difficult to split directory nodes (particularly of the higher directory levels) in an overlap-free way. The regions of the child nodes of such directory nodes are axis-parallel rectangles, which have, themselves, been created in a recursive process of splitting. Typically, only the first split in the split history of the child nodes is a good split decision for the parent node. Therefore, each directory node stores this split history of its children and, thus, tries to imitate the splits of its children. If this imitation would result in an unbalanced tree, the split is avoided and, instead, a so-called supernode is created, i.e. a node with an enlarged size and capacity.

Cost Model Based Optimization Techniques

In [3,7], a cost model for vector space index structures with particular emphasis on high dimensional spaces was proposed. The main idea is to estimate the average size of a page region and of a query and to form the Minkowski sum of these two geometric objetcs which has been shown to be a measure of the probability of a page access. To take effects in high-dimensional spaces into account, two concepts were additionally included into the cost model, *the boundary effect* and *the fractal dimension*. The boundary effect means that for high-dimensional spaces typically large parts of a query sphere and of the Minkowski sum are outside the covered data space. The fractal dimension covers the sparsity of the data space. Typically, a high dimensional space is not uniformly and independently in all dimensions covered by data objects. Rather, some of the attributes are dependent on each other, and, therefore, only subspaces of lower dimensionality (not necessarily linear subspaces) are populated. The intrinsic dimensionality of the data can be formalized by the fractal dimension.

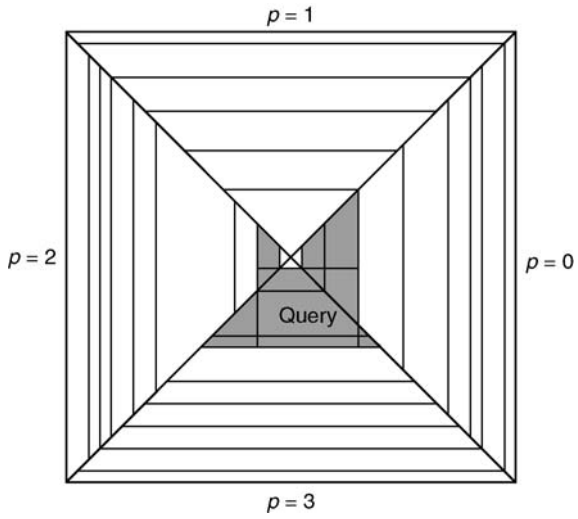
It is important to optimize various parameters of an index using a cost model in order to make it competitive. For instance, in [8], it was proposed to optimize the page size of data pages of the index according to the cost model. To do this automatically is particularly important because data sets with a large intrinsic dimensionality cause scanning of large parts of the index. If small page sizes (such as 4 KB) are used in this case, the random accesses cause a large I/O load, and an unoptimized index is much slower than sequential data processing. In contrast, if the intrinsic dimensionality is small, then large page sizes lead to unnecessarily large reading operations. Carefully optimized page sizes outperform sequential scanning and non-optimized indexes for most of the data sets and tie with the competitors in the worst case. The dynamic page size optimization allows the automatic adaptation of the page size even if the data distribution of the data set changes over time. Moreover, it is possible to use different page sizes for different parts of the index structure, if the index stores groups of data with different characteristics.

Another example of the successful application of a cost model is tree striping [4], where the data objects are vertically decomposed into sub-vectors which are stored in separate search trees. The dimensionality of the sub-vectors (and, inversely, the number of index structures to store them) is an optimization task which is important, because the unnecessary decomposition in too many, small sub-vectors causes a large overhead in the final merging of the results, whereas no decomposition or an insufficient decomposition the query processing inside a single index is too expensive due to the curse of dimensionality. Again, a cost model can decide an optimal dimensionality.

Pyramid Technique

The Pyramid Technique [1] is a *one-dimensional embedding technique,* that means, it transforms the high-dimensional objects into a single-dimensional space which can be efficiently indexed using B-trees, for instance. In contrast to most previous embedding techniques which are based on space-filling curves, the Pyramid Technique does not rely on a *recursive* schema of space partitioning. In contrast, the data space is partitioned into $2 \cdot d$ hyper-pyramids which share the origin of the data space (which can be chosen as the center point of the data set) as top point and have each an individual $(d - 1)$-dimensional basis area (cf. the four pyramids in Fig. 1 $p = 0..3$). The pyramids are systematically numbered which forms the first part of the embedding key (a natural number p). The second part is the distance (with respect to the maximum metric) from the origin (a positive real number r). The embedding key can be formed as an ordered pair $k = (p, r)$, or, equivalently, if the maximum of all r-values (r_{max}) is known, one single embedding key

$k' = r_{max} \cdot p + r$ can be formed. In both cases, a d-dimensional range query can be translated into a set of search intervals on the search keys. The number of intervals is at most $2 \cdot d$ because the query object can at most have one intersection with each of the pyramids. Since nearest neighbor queries can be transformed into range queries (which requires a set of at most two one-dimensional ranking processes per pyramid) it is also possible to evaluate nearest neighbor queries. The Pyramid Technique is in general not limited to a particular metric, but the schema of space partitioning makes it particularly suited for queries using the maximum metric. The Pyramid Technique has inspired a number of other techniques such as the Onion Technique [10] or Concentric Hyperspaces [11] and many others which focus on different metrics including the Euclidean metric.



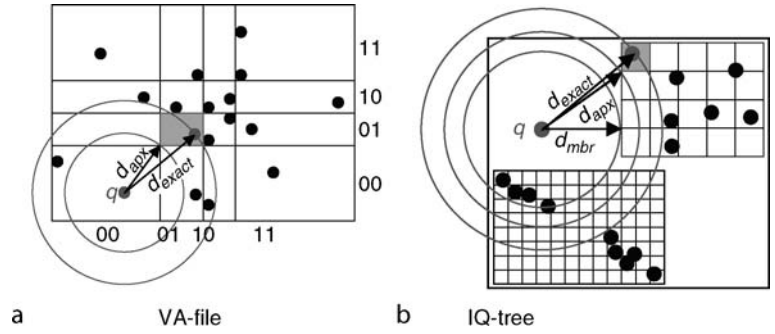
High Dimensional Indexing. Figure 1. Pyramid technique.

VA-File

As an alternative to tree based indexing techniques, the VA-file (Vector Approximation File [6]) has been designed to speed up the linear scan. The basic idea is to store compact approximations of the high dimensional feature vectors in the so-called *approximation file* which is small enough to enable a fast linear scan during query processing. The approximations are derived by dividing each dimension of the data space into equally populated intervals. Thus, the data space is divided into hyper-rectangular cells. Each cell is allocated by a bit string composed of a fixed number of bits per dimension. As approximation for all contained points, this bit string is stored in the approximation file. The principle of vector quantization is visualized in Fig. 2a. In this example, two bits are assigned to each dimension. While the approximation file is scanned, upper and lower bounds on the distances from the query to the approximations d_{apx} are derived and refined. As a result, only very few approximations have to be further checked for answer candidates, which requires random accesses. Extensions to the VA-file e.g., include the parallel VA-file [18] originally designed for parallel nearest neighbor search in large image collections and the kernel-VA-file [13] for complex, kernel supported distance metrics. Due to global quantization, the VA-file is especially suitable for uniformly distributed data. For clustered data, hierarchical techniques show superior performance.

IQ-Tree

With the IQ-tree [2], the idea of quantizing the data using a grid has been integrated into a hierarchical indexing method. In the IQ-tree, every data page has two versions, one which contains the data points in a compressed way, and one which contains the exact



High Dimensional Indexing. Figure 2. Schematic view of vector quantization.

information. In contrast to the VA-file, each page is quantized independently, and this independent quantization gives the structure its name. The quantization grid is not based on quantiles but is a regular grid partition of the rectangular page region. In the IQ-tree, it was shown that quantiles are not necessary because the page regions already serve a sufficient adaptation to the actual data distribution, and storing individual quantiles for each page would result in a prohibitive storage overhead. In contrast to the VA-file, the resolution of the grid is not determined experimentally but dynamically optimized using a cost model. In the IQ-tree, the actual directory is non-hierarchic and contains only one level. Taken together, the IQ-tree consists of three levels, the directory level, the compressed data level and the uncompressed data level. The vector quantization principle is illustrated in Fig. 2b. When considering a query object q and an arbitrary database object, there are two lower bounding approximations of the exact distance d_{exact}, the distance to the minimum bounding rectangle d_{mbr} which can be determined from the directory and the distance to the grid cell (d_{apx}) which can be derived from the compressed data level.

Apart from the idea of independent quantization, the IQ tree contains the idea of the Fast Index Scan (FIS). After scanning of the directory, it can be decided for every page, whether it is certainly needed, certainly excluded, or has some probability to be needed for a given nearest neighbor query. The probability can again be estimated using a cost model. From this probability, a query processing algorithm can derive optimal schedules of pages, i.e., pages which have neighboring disk addresses, can be called in together in a single I/O operation if they have both high probabilities. Depending on data and query characteristics, the pages of the compressed data level can either be accessed by random accesses or in a more sequential style. Another related approach, also designed to allow a flexible adaptation of the height of the directory to the current data distribution is the A-tree [17].

iDistance

iDistance [21] combines a one-dimensional embedding technique with clustering. The embedding is obtained by expressing the similarity ratios in high dimensional space in terms of distances to reference points which can be stored in a single dimensional index. More precisely, the data space is first split into partitions. As a second step, a reference point is selected for each partition. For all data objects the distances to their reference points are stored in a B^+-tree. The performance of the index strongly depends on an appropriate partitioning and on the strategy how to select the reference points. In the original paper, the authors proposed two variants of partitioning: The straightforward approach of equal data space partitioning is especially suitable for uniformly distributed data. For clustered data the partitions can be determined by an arbitrary clustering algorithm, a simple sampling based method is proposed in the paper. Often it is favorable to select the centroid of a partition as reference point, however selecting an edge point may help to reduce overlap.

The most important strategies to cope with the problems of high dimensional spaces can be subsumed by clustering and mapping to one dimensional space. During query processing, the maximum distance between the query point and the points contained in each partition is used for pruning. The single dimensional space of distances can be very efficiently searched supported by the B^+-tree. However, in high dimensional data, the distances to reference points are often not selective and there are no clusters in the full dimensional space such that large parts of the index need to be accessed. But many real world data sets exhibit more selective subspaces. Therefore in [14] a subspace clustering step to identify local correlations in the data is applied before indexing.

Key Applications

High dimensional indexing is important for similarity search systems in various application areas such as multimedia, CAD, systems biology, medical image analysis, time sequence analysis and many others. Complex objects are typically transformed into vectors of a high-dimensional space (feature vectors), and the similarity search thereby translates into a range or nearest neighbor query on the feature vectors. High-dimensional feature vectors are also required for more advanced data analysis tasks such as cluster analysis or classification.

Cross-references

- ► Curse of Dimensionality
- ► Dimensionality Reduction
- ► Indexing Metric Spaces
- ► Nearest Neighbor Query

Recommended Reading

1. Berchtold S., Böhm C., and Kriegel H.-P. The pyramid-technique: towards breaking the curse of dimensionality. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 142–153.
2. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., and Sander J. Independent quantization: an index compression technique for high-dimensional data spaces. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 577–588.
3. Berchtold S., Böhm C., Keim D.A., and Kriegel H.-P. A cost model for nearest neighbor search in high-dimensional data space. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 78–86.
4. Berchtold S., Böhm C., Keim D.A., Kriegel H.-P., and Xu X. Optimal multidimensional query processing using tree striping. In Proc. 2nd Int. Conf. Data Warehousing and Knowledge Discovery, 2000, pp. 244–257.
5. Berchtold S., Keim D.A., and Kriegel H.-P. The x-tree : an index structure for high-dimensional data. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 28–39.
6. Beyer K.S., Goldstein J., Ramakrishnan R., and Shaft U. When is "nearest neighbor" meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.
7. Böhm C. A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst., 25(2):129–178, 2000.
8. Böhm C. and Kriegel H.-P. Dynamically optimizing high-dimensional index structures. In Advances in Database Technology, Proc. 7th Int Conf on Extending Database Technology, 2000, pp. 36–50.
9. Böhm C., Berchtold S., and Keim D.A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
10. Chang Y.-C., Bergman L.D., Castelli V., Li C.-S., Lo M.-L., and Smith J.R. The onion technique: indexing for linear optimization queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 391–402.
11. Ferhatosmanoglu H., Agrawal D., and Abbadi A.E. Concentric hyperspaces and disk allocation for fast parallel range searching. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 608–615.
12. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
13. Heisterkamp D.R. and Peng J. Kernel vector approximation files for relevance feedback retrieval in large image databases.. Multimed. Tools Appl., 26(2):175–189, 2005.
14. Jin H., Ooi B.C., Shen H.T., Yu C., and Zhou A. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 87–98.
15. Katayama N. and Satoh S. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 369–380.
16. Lin K.-I., Jagadish H.V., and Faloutsos C. The tv-tree: an index structure for high-dimensional data. VLDB J., 3(4):517–542, 1994.
17. Sakurai Y., Yoshikawa M., Uemura S., and Kojima H. The A-tree: an index structure for high-dimensional spaces using relative approximation. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 516–526.
18. Weber R., Böhm K., and Schek H.-J. Interactive-time similarity search for large image collections using parallel VA-files. In Proc. 4th European Conf. Research and Advanced Tech. for Digital Libraries. Springer, 2000, pp. 83–92.
19. Weber R., Schek H.-J., and Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 194–205.
20. White D.A. and Jain R. Similarity indexing with the ss-tree. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 516–523.
21. Yu C., Ooi B.C., Tan K.-L., and Jagadish H.V. Indexing the distance: an efficient method to KNN processing. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 421–430.

High-Dimensional Clustering

► Document Clustering

Higher-Order Entity-Relationship Model

► Extended Entity-Relationship Model

Histogram

QING ZHANG
CSIRO ICT Centre, Herston, QLD, Australia

Definition

Given a relation R and an attribute X of R, the domain D of X is the set of all possible values of X, and a finite set $V(\subseteq D)$ denotes the distinct values of X in an instance r of R. Let V be ordered, that is: $V = \{v_i : 1 \leq i \leq n\}$, where $v_i < v_j$ if $i < j$. The instance r of R restricted to X is denoted by T, and can be represented as: $T = \{(v_1, f_1), \cdots (v_n, f_n)\}$. In T, each v_i is distinct and is called a *value* of T; and f_i is the occurrence of v_i in T and is called the *frequency* of v_i, T is called the *data distribution*. A *histogram* on data distribution T is constructed by the following two steps.

1. Partitioning the values of T into $\beta(\geq 1)$ disjoint intervals (called buckets) – $\{B_i : 1 \leq i \leq \beta\}$, such that each value in B_i is smaller than that in B_i if $i < j$.