An Indexing Structure for Dynamic Multidimensional Data in Vector Space*

Elena Mikhaylova, Boris Novikov, and Anton Volokhov

Abstract. The multidimensional $k-NN$ (k nearest neighbors) query problem is relevant to a large variety of database applications, including information retrieval, natural language processing, and data mining. To solve it efficiently, the database needs an indexing structure that provides this kind of search. However, attempts to find an exact solution are hardly feasible in multidimensional space. In this paper, a novel indexing technique for the approximate solution of $k-NN$ problem is described and analyzed. The construction of the indexing tree is based on clustering. Indexing structure is implemented on top of high-performance industrial DBMS.

1 Introduction

Search efficiency is crucial for modern information retrieval. Commonly, search queries retrieve a relatively small portion of information. In this case, indexing search is much more efficient than a full scan. Any given process or stored object can be characterized by a set of features that are usually called attributes. The purpose of any index is quick access to the object based on the values of some of its attributes. In other words, the indices provide effective implementation of associative search.

Multidimensional searching is primarily associated with processing spatial and spatio-temporal data. Another class of applications processing multidimensional data includes systems based on various flavors of a text vector model for methods of data mining, pattern recognition, data compression etc. Data obtained from the application of this class are typically characterized by high dimensionality.

Elena Mikhaylova · Boris Novikov · Anton Volokhov
Saint Petersburg University
e-mail: egmichailova@mail.ru, b.novikov@spbu.ru,
a.v.volokhov@gmail.com

* This research is supported by Russian Foundation for Basic Research, grant 10-07-00156.

T. Morzy et al. (Eds.): Advances in Databases and Information Systems, AISC 186, pp. 185–193.
springerlink.com

In this work we present an approach to approximate $k-NN$ multidimensional searching. Its idea is based on tree-clustering structure. Due to the curse of dimensionality we refused to provide exact $k-NN$ queries and have concentrated on approximate ones.

The remaining part of the paper is organized as follows. Section 2 contains the overview of related work. Section 3 informally outlines the techniques used in our approach. Our algorithms and data structures are presented in section 4, followed by analysis of experiments and results in section 5.

2 Related Work

Years of multidimensional searching evolution led to development of various indexing algorithms. The most respected tree structure to provide multidimensional indexing in generic metric space is M-Tree [5]. To improve the above-mentioned algorithm, a static tree reclustering technique is proposed in bulk loading algorithm [4]. Slim tree [20, 18] is developed to minimize overlap of nodes at the same level. Attempts to partition the initial set of points to disjoint subsets tuned to provide k-NN queries are made in the family of algorithms based on Voronoi diagram: D-Tree [22] treats Voronoi diagram buckets as nodes, and VoR-Tree [15] starts query processing with estimating a near point using R-Tree [10, 11] and then treat this point as a start point for querying Voronoi diagram built on leaf points.

In last decade multidimensional hashing has become a serious competitor to traditional tree approach. Locality-sensitive hashing [7] uses a priori information about a dataset to build a fast and efficient structure for approximate querying. This approach was introduced [12] and well developed [6, 1]. Yet, when dataset is changing rapidly, this idea is not applicable.

In many cases, it is computationally complex not only to find and to prepare vectors (eg. feature extraction of images), but also to calculate the function of similarity (eg. the Levinstein distance, quadric form distance or tree edit distance). Search could be greatly accelerated if it is possible to store a matrix of pairwise distances between objects. This idea was first presented in AESA [21]. To efficiently answer k-NN queries, Ak-LAESA [13] has been suggested. Unfortunately, this solution is not scalable because of the quadric behavior of index size.

In [19] k-NN search problem deals with the double filtering effect of clustering and indexing. The clustering algorithm ensures that the largest cluster fits into main memory and that only clusters closest to a query point need to be searched and hence loaded into main memory. In each cluster data is distributed with ordered-partition tree (OP-tree) main memory resident index, which is efficient for processing k-NN queries.

In [3] divide and conquer methods for computing an approximate k-NN graph are proposed. A hash table is used to avoid repeating distance calculations during the divide and conquer process.

Different aspects of the curse of dimensionality are known to present serious challenges to various machine-learning methods and tasks. In [14] a new aspect of the curse of dimensionality, referred to as hubness, is explored. Origins of this phenomenon, and its connection to k-NN query evaluation are discussed.

High-dimensional clustering is used by some content-based image retrieval systems to partition the data into groups (clusters), which are then indexed to accelerate processing of queries. Recently, the Cluster Pruning approach was proposed in [8] as a simple way to produce such clusters. The evaluation of the algorithm was performed within an image indexing context. The paper [8] discusses the parameters that affect the efficiency of the algorithm, and proposes changes to the basic algorithm to improve performance.

[16] presents an adaptive Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) technique for high-dimensional indexing. The MMDR technique discovers elliptical clusters for more effective dimensionality reduction by using only the low-dimensional subspaces, data points in the different axis systems are indexed using a single B+-tree. The technique is highly scalable in terms of data size and dimensionality. It is also dynamic and adaptive to insertions. An extensive performance study was conducted using both real and synthetic datasets, and the results show that the proposed technique not only achieves higher precision, but also enables queries to be processed efficiently.

In paper [9] a new clustering method is proposed: to group together only points that are close to each other. The remaining points are stored separately, not clustered. Such a structure is obtained unbalanced. In order to create the indexing structure, in [2], the distances between the selected set of pivots and the data objects is computed, sorted and nearest distances are stored in separated tables. For search at first query is compared with pivots.

In resent research [17] general problems of metric access methods are discussed. Relevance of metric generalization in questioned. Idea of using more data-specific approaches to indexing is proposed.

3 The Approach and Rationale

Due to survey [17], most common metric in this case is Lp. The goal of this work is to construct dynamic bottom-up built tree for approximate nearest-neighbor search objects in high-dimensional vector space Lp. To build the indexing structure we use tree based paradigm with data points stored in leafs and routing points in non-leaf nodes. Parent points (centroid) are means of all its children. Tree is balanced by the construction. Volume of node is a parameter of algorithm and depends on operation system page size.

As we refused to use the metric approach, we don't have upper bound for k-approximate nearest-neighbor queries points, retrieved as nearest neighbors, are allowed to be arbitrarily far from the query point. According to this approach we choose precision (number of actual $k-NN$ in retrieved set divided by the value of k) as our main metric.

The idea of the algorithm is not to split overcrowded node, but to recluster the whole level of the tree, distinguishing new node in this level. This technique is believed to catch the inherent structure of initial dataset. As a result we can obtain more compact nodes. So the node size is a parameter of the algorithm. However, inserting is performed in traditional technique $1-NN$ query is implemented and new point is added to retrieved leaf node.

The algorithm is developed with keeping in mind the idea of dynamic tuning: since in every non-leaf node we store only total number of its leaf children, the default approach is to descend only to the nearest leaf during searching. However, the algorithm implies the possibility of tuning the number of nodes to descend to perform better querying. In this case we choose total number of leaf nodes to be examinated and descend to several nodes, if number of children in nearest node is less than chosen threshold.

In order to evaluate the characteristics and performance of this structure, we have implemented the model over the data provided industrial relational DBMS. Perhaps this decision affects the efficiency but also ensures quick implementation, suitable for use in the prototype, for example, in an experimental system for content based image retrieval. That is, the developed system allows finding images on a number of characteristics similar to the one desired.

4 Algorithm Description

4.1 Data Structure

For indexing structure we used 4 tables. First table is a Table of points. In order the table structure was independent from the space dimension, each vector is represented in the table as a set of rows - one row for each vector attribute. Each table row has the *Point_id*, *Attribute_id* and *Attribute_value*. The table of centroids stores mass centers for every cluster and has the same structure as Table of points.

The third table is used to store the tree structure. Each row has *Point_id* and *Child_id*.

Also we use the temporary tables for mass centers and tree structure. The rows have *Point_id*, *Attribute_id*, *Attribute_value* and *Iteration*.

4.2 Index Structure Parameters

The proposed algorithm has parameters. The parameters might affect the efficiency of the algorithm. Parameters are:

- distance measure,
- maximum number of points in one node maxNodeSize,
- balancing factor to prevent frequent re-clustering balancingFactor

In our experiments distance was calculated using the $L2$ metric , but algorithm works in any vector space.

We can choose maximum number of points in one tree node - *maxNodeSize*. In our experiment *maxNodeSize* was changed from 10 to 100. This number depends on the space dimensionality.

In order to avoid frequent reorganization of the tree, we dont fill all clusters to *maxNodeSize* by re-clusterization. Number of points in each node cant exceed two thirds of the *maxNodeSize*.

4.3 Index Structure Construction

Indexing algorithm starts with an empty set of indexed points and one tree node, considered as root node. When new point arrives, we recursively descent to node, that is nearest to arrived point. When the leaf node is reached, new point is added to corresponding node in tree structure. After that, algorithm rises to the root node, recalculating all centroid points to hold keep them being mean of their children.

When node overflows, we get all children its parent's node brothers and perform k-means clustering with following number of centers:

$$k = \max\left(1, \frac{n}{k} \times maxnodesize \times balancingFactor - k\right)$$

where n number of obtained, k number of brothers.

If constructed cluster is overflowed, we recursively recluster upper level of tree structure. When root node is reached, the tree is pulled up.

Initial values for k-means clustering are set to previous node centers. The point, that overflowed cluster become the one of new centers, the others are taken randomly from effected points. K-means algorithm is chosen according to the fact that centroids in tree are already similar to k-means centers every centroid contain points, that more similar to this centroid, than to the others. In this case we can assume, that nodes don't need many k-means iterations to return values of new centroids. On the other hand, k-means doesn't keep in mind inherent database properties and with some assumptions has exactly Voronoi diagram in a result. Algorithmic complexity of constructing the indexing structure equals

$$O(n \times \log^2 n)$$

4.4 Search

The search algorithsm can be described with the following steps:

1. Start from the top level.
2. Compute distances between points of current level (inside current node) and query point, retrieve list of points, ordered by distance to the query point

3. Pick points from the top of the list, until total number of point's leafs is bigger than number of points we want to examine.
4. Descend to chosen nodes and repeat $2-3$ until leaf nodes are reached.
5. Get all points from the leaf nodes.
6. Sort obtained points and retrieve first k points, k is defined by query type.

Algorithmic complexity of searching equals:

$$O(d \times \log n)$$

where d dimensionality of initial dataset, n cardinality of initial dataset.

5 Experiment and Analysis

The algorithm described above, was analyzed on two different datasets.

The first one is 41-dimensional representation of image characteristics with cardinality of 30000. The second one is 100-dimensional synthetic dataset with values from Gaussian distribution. Cardinality of second dataset is also 30000 vectors.

The method was implemented within the DBMS Oracle. Tables were constructed and procedures were written. Maximum node size was set to 10/50/100.

Table 1 Experimental results (1% examined)

	Image representation dataset			Gaussian dataset	
maxNodeSize	10	50	100	50	100
$1-NN$ query	21%	28%	85%	20%	31%
$10-NN$ query	27%	45%	32%	3%	5%
$100-NN$ query	45%	38%	18%	5%	3%

Table 2 Experimental results (10% examined)

	Image representation dataset		Gaussian dataset	
maxNodeSize	50	100	50	100
$1-NN$ query	56%	92%	42%	50%
$10-NN$ query	60%	47%	20%	23%
$100-NN$ query	61%	35%	20%	21%

Each experiment was performed twice. First time we set the total number of leaf points to be examined to 1% of initial dataset, and second time to 10%. Results for the fast one-percent scan are shown on the figure 1 and the result for more precise ten-percent scan are on the figure 2.

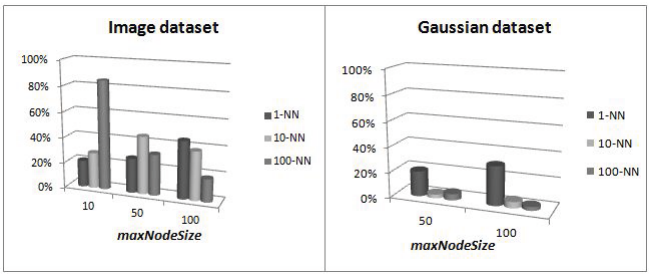

Fig. 1 The results for scanning 1% of the dataset

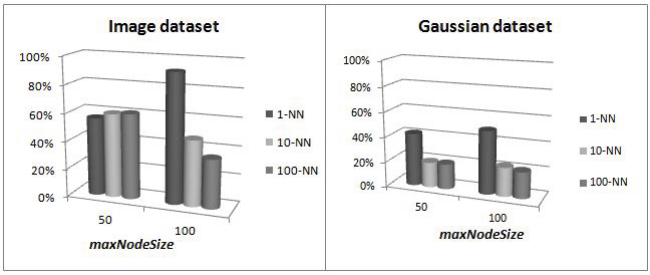

Fig. 2 The results for scanning 10% of the dataset

6 Conclusion

We present the algorithm that can deal with rapidly changing data in high-dimensional vector space. Experiments show high precision on not very high dimensional space. Precision in high-dimensional space strongly depends on size of a tree node: when nodes size is approximately equal to the value of k in $k-NN$ query, search algorithm visits only a few number of nodes and retrieves only a few actual nearest neighbors.

However to resolve this problem we can increase the number of nodes examined on each level: experiments show that scanning only 10% of the database results in significant increase of precision.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM 51(1), 117–122 (2008), http://doi.acm.org/10.1145/1327452.1327494, doi:10.1145/1327452.1327494

2. Barton, S., Gouet-Brunet, V., Rukoz, M.: Large scale disk-based metric indexing structure for approximate information retrieval by content. In: Proceedings of the 1st Workshop on New Trends in Similarity Search, NTSS 2011, pp. 2–7. ACM, New York (2011), http://doi.acm.org/10.1145/1966865.1966869, doi:10.1145/1966865.1966869
3. Chen, J., Fang, H.R., Saad, Y.: Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. J. Mach. Learn. Res. 10, 1989–2012 (2009), http://dl.acm.org/citation.cfm?id=1577069.1755852
4. Ciaccia, P., Patella, M.: Bulk loading the m-tree. In: Proceedings of the 9th Australasian Database Conference, ADC 1998, pp. 15–26 (1998)
5. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB 1997, pp. 426–435. Morgan Kaufmann Publishers Inc., San Francisco (1997), http://dl.acm.org/citation.cfm?id=645923.671005
6. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG 2004, pp. 253–262. ACM, New York (2004), http://doi.acm.org/10.1145/997817.997857, doi:10.1145/997817.997857
7. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases, VLDB 1999, pp. 518–529. Morgan Kaufmann Publishers Inc., San Francisco (1999), http://dl.acm.org/citation.cfm?id=645925.671516
8. Gudmundsson, G.T., Jónsson, B.T., Amsaleg, L.: A large-scale performance study of cluster-based high-dimensional indexing. In: Proceedings of the International Workshop on Very-Large-Scale Multimedia Corpus, Mining and Retrieval, VLS-MCMR 2010, pp. 31–36. ACM, New York (2010), http://doi.acm.org/10.1145/1878137.1878145, doi:10.1145/1878137.1878145
9. Günnemann, S., Kremer, H., Lenhard, D., Seidl, T.: Subspace clustering for indexing high dimensional data: a main memory index based on local reductions and individual multi-representations. In: Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT 2011, pp. 237–248. ACM, New York (2011), http://doi.acm.org/10.1145/1951365.1951395, doi:10.1145/1951365.1951395
10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD 1984, pp. 47–57. ACM, New York (1984), http://doi.acm.org/10.1145/602259.602266, doi:10.1145/602259.602266
11. Guttman, A.: R-trees: a dynamic index structure for spatial searching. SIGMOD Rec. 14(2), 47–57 (1984), http://doi.acm.org/10.1145/971697.602266, doi:10.1145/971697.602266
12. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998), http://doi.acm.org/10.1145/276698.276876, doi:10.1145/276698.276876
13. Moreno-Seco, F., Milcó, L., Oncina, J.: A modification of the laesa algorithm for approximated k-nn classification. Pattern Recogn. Lett. 24(1-3), 47–53 (2003), http://dx.doi.org/10.1016/S0167-8655(02)00187-3, doi:10.1016/S0167-8655(02)00187-3

14. Radovanović, M., Nanopoulos, A., Ivanović, M.: Hubs in space: Popular nearest neighbors in high-dimensional data. J. Mach. Learn. Res. 11, 2487–2531 (2010), http://dl.acm.org/citation.cfm?id=1756006.1953015
15. Sharifzadeh, M., Shahabi, C.: Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. Proc. VLDB Endow. 3(1-2), 1231–1242 (2010), http://dl.acm.org/citation.cfm?id=1920841.1920994
16. Shen, H.T., Zhou, X., Zhou, A.: An adaptive and dynamic dimensionality reduction method for high-dimensional indexing. The VLDB Journal 16(2), 219–234 (2007), http://dx.doi.org/10.1007/s00778-005-0167-3, doi:10.1007/s00778-005-0167-3
17. Skopal, T.: Where are you heading, metric access methods?: a provocative survey. In: Proceedings of the Third International Conference on SImilarity Search and Applications, SISAP 2010, pp. 13–21. ACM, New York (2010), http://doi.acm.org/10.1145/1862344.1862347, doi:10.1145/1862344.1862347
18. Skopal, T., Pokorný, J., Krátký, M., Snášel, V.: Revisiting M-Tree Building Principles. In: Kalinichenko, L.A., Manthey, R., Thalheim, B., Wloka, U. (eds.) ADBIS 2003. LNCS, vol. 2798, pp. 148–162. Springer, Heidelberg (2003)
19. Thomasian, A., Zhang, L.: Persistent clustered main memory index for accelerating k-nn queries on high dimensional datasets. Multimedia Tools Appl. 38(2), 253–270 (2008), http://dx.doi.org/10.1007/s11042-007-0179-7, doi:10.1007/s11042-007-0179-7
20. Traina Jr., C., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 51–65. Springer, Heidelberg (2000), http://dl.acm.org/citation.cfm?id=645339.650146
21. Vidal, E.: New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa). Pattern Recognition Letters 15(1), 1–7 (1994)
22. Xu, J., Zheng, B., Lee, W.C., Lun Lee, D.: The d-tree: An index structure for planar point queries in location-based wireless services. IEEE Trans. on Knowl. and Data Eng. 16(12), 1526–1542 (2004), http://dx.doi.org/10.1109/TKDE.2004.97, doi:10.1109/TKDE.2004.97