Challenges in Finding an Appropriate Multi-Dimensional Index Structure with Respect to Specific Use Cases

Alexander Grebhahn
Institute of Technical and Business Information Systems
University of Magdeburg
grebhahn@st.ovgu.de

David Broneske
Institute of Technical and Business Information Systems
University of Magdeburg
dbronesk@st.ovgu.de

Martin Schäler
Institute of Technical and Business Information Systems
University of Magdeburg
schaeler@ovgu.de

Reimar Schröter
Institute of Technical and Business Information Systems
University of Magdeburg
rschroet@st.ovgu.de

Veit Köppen
Center for Digital Engineering
University of Magdeburg
vkoeppen@ovgu.de

Gunter Saake
Institute of Technical and Business Information Systems
University of Magdeburg
saake@ovgu.de

ABSTRACT

In recent years, index structures for managing multi-dimensional data became increasingly important. Due to heterogeneous systems and specific use cases, it is a complex challenge to find an appropriate index structure for specific problems, such as finding similar fingerprints or micro traces in a database. One aspect that should be considered in general is the dimensionality and the related *curse of dimensionality.*

However, dimensionality of data is just one component that have to be considered. To address the challenges of finding the appropriate index, we motivate the necessity of a *framework* to evaluate indexes for specific use cases. Furthermore, we discuss core components of a framework that supports users in finding the most appropriate index structure for their use case.

Keywords

index structures, evaluation, multi-dimensional data

1. INTRODUCTION

In the last years, data storage and management in computer-aided systems became more advanced, because of an increasing amount of unstructured data being stored. For example, in multimedia databases images or videos are stored and analyzed to find similar data items. A special use case is the Digi-Dak Database Project[1], where multi-dimensional feature vectors of fingerprints and micro traces are stored in a database. To manage these data items, methods are required to handle unstructured data in an appropriate way. It is possible to extract feature vectors from an item to manage the data in a compressed and meaningful way. For managing these feature vectors, multi-dimensional index structures can be used. In general, the question arises, which index structure supports managing data best. Throughout this paper, index structure performance describes suitability with respect to a specific use case. However, we analyze core aspects that have to be considered, if trying to answer this for a specific use case. In order to achieve a reconstructible and valid comparison, we present the idea of a framework that allows the comparison of different index structures in a homogeneous test environment.

This paper is organized as follows: In Section 2, we give a short overview of basic components that have to be considered for evaluating the performance of index structures. Within Section 3, we give an overview of additional challenges, which have to be handled by using index structures in a specific use case. Finally, in Section 4, we present core components that a framework needs for quantitatively evaluation of multi-dimensional index structures with respect to different use cases.

[1]https://omen.cs.uni-magdeburg.de/digi-dak

24th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

2. BASIC CHALLENGES

Querying multi-dimensional data in an efficient way is a complex challenge. Within the last decades, new index structures are proposed and existing once are improved to solve this challenge. Regarding a specific use case, it is not suitable to consider an index structure in isolation. Additionally data properties, used query types, and underlying distance metrics have to be taken into account. In this section, we give a short overview of these four basic challenges.

2.1 Data Properties

Characteristics of data cause main challenges of querying data within a database system. For instance, data dimensionality has to be considered, because existing index structures are generally effected by the curse of dimensionality [7, 23]. As a result, index structures, that are suitable for a small number of dimensions are not necessarily suitable for a larger amount of dimensions. An additional important property is the data distribution, because some index structures are more practicable for clustered data than others.

Furthermore, value domain and the type of the data has to be considered.

2.2 Query Types

Based on the work of Böhm et al. [8], query types can be categorized into two groups: ϵ-similarity queries and Nearest-Neighbor-similarity (NN-similarity) queries. The former describes a query, resulting in a set of data points being situated in a defined ϵ-distance to the query point, whereas the latter results in a data point being the nearest item to the query point. Describing these two groups, the ϵ-similarity and NN-similarity has to be defined.

Definition: ϵ-similarity Query.

Two data points p_1 and p_2 are ϵ-similar if and only if $d(p_1, p_2) \leq \epsilon$. The function d defines a similarity measure for two points. In literature, similarity measures are often replaced by distance metrics, which we review in Section 2.3. For finding all points in the data base being ϵ-similar, an ϵ-similarity query is executed. A special case of the ϵ-similarity is represented for $\epsilon = 0$, because this implies two identical points and an exact match is executed [8].

Definition: NN-similarity Query.

The data point p_1 is NN-similar to p_2 with respect to a data base of points DB if and only if $\forall p \in DB, p \neq p_1 : d(p_2, p_1) \leq d(p_2, p)$. For NN-similarity queries, all points in a database are retrieved that are NN-similar to the query point. An extension to the NN-similarity query is presented, when instead of a nearest neighbor, k nearest neighbors have to be retrieved. In this paper, we call the resulting query k-NN query.

Apart from the mentioned similarity range query, window queries are common queries and often called range queries in literature [24]. These window queries are defined by intervals for every queried dimension.

2.3 Distance Metrics

To execute similarity queries, we require a function computing the similarity of two data items. To this end, similarity for equal points is 1 whereas the maximum dissimilarity is expressed by 0. Equivalent information is delivered from distance metrics, whereupon two data items are more similar, the smaller their distance is.

The most common distance metrics are Minkowsky class metrics, also called L_p distance metrics. The distance of two data items x and y is computed by:

$$L_p(x, y) = \Big(\sum_{i=1}^{d} (x_i - y_i)^p\Big)^{1/p}.$$

By choosing different values for p, different representatives of this class are produced. For $p = 2$, the Euclidean distance metric is generated, which dominates common database systems according to Bugatti et al. [9].

Beneath these distance metrics, there are many other metrics, such as Canberra [9] or Dynamical-Partial [16] distance function. In contrast to Minkowsky distance functions, Dynamical-Partial distance metric d_m uses only the m smallest distances for the computation of the distance of data items [16]. As a result, in some specific use cases, it can be a great benefit using the Dynamical-Partial distance metric, because the influence of particular dimensions can deteriorate the distance of data items.

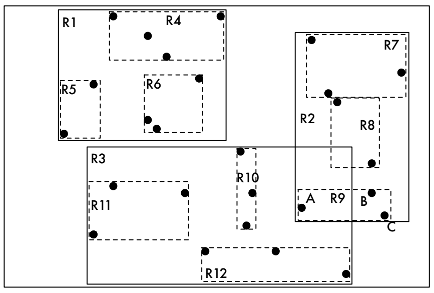


Figure 1: R-Tree with overlapping MBRs.

2.4 Index Structures

Since we aim at providing a comprehensive set of indexes, we want to consider different types of index structures. Thus, we use the classification of Weber et al. [23] to address a broad variety of different approaches. Thus, index structures are classified by partitioning of the data space. Index structures that partition the whole space are called space partitioning methods, whereas data partitioning methods partition the necessary space according to the location of data points [8, 23]. Consequently, there are regions that are not taken into account by performing a query on data partitioning methods.

Alternatively, Andoni and Indyk [2] classify index structures by query results. There are exact index structures that guarantee to retrieve the exact result of a query. Although, this behavior is usually preferred, there are approximation-based index structures, guaranteeing to retrieve points that are similar to the correct result of a query. For instance for k-NN queries, approximation-based index structures provide k near neighbors to the query point instead of all exact nearest neighbors. Hereby, the quality of the retrieved results, called precision, can differ significantly, because approximate index structures aim at improving the query performance by decreasing the precision. Nevertheless, an approximation-based index should hold a threshold, because resulting data would not be useful. In the following sections, we present some representatives of index structures. First, exact index structures, such as R-Tree [11], Pyramid Technique [5], and VA-File [22] are introduced. Subsequently, p-stable Locality Sensitive Hashing [12] as an approximation-based index structure is presented.

2.4.1 Exact Index Structures

Giving an overview of existing index structures, we introduce promising exact index structures in this section. Furthermore, the difference between space partitioning and data partitioning methods is stated by presenting at least one index structure for each category.

R-Tree.

One of the most important multi-dimensional index structures is the R-Tree [11], introduced by Guttmann in 1984. Since this time, many new index structures are proposed based on the ideas used in the R-Tree. For instance R$^+$-Tree [21], R*-Tree [4], X-Tree [6], A-Tree [19], and SR-Tree [13]. Beside these structures, there are many more index structures which are not mentioned here. For further informations, see Samet [20], giving a comprehensive overview of existing index structures.

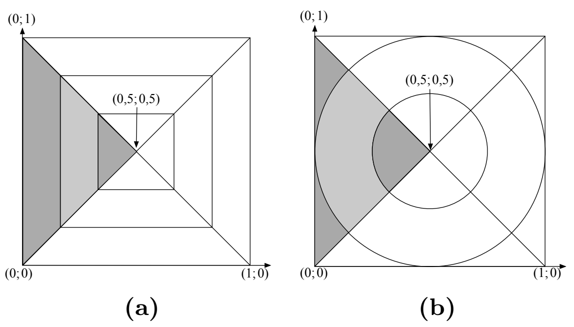
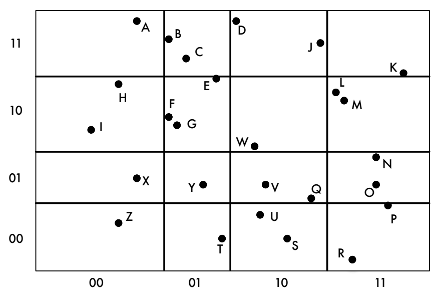


Figure 2: Space partition of a 2-d space by Berchtold et al. [5] (a) and Lee and Kim [15] (b).



Approximation File

A	00 11	N	11 01
B	01 11	O	11 01
C	01 11	P	11 00
D	10 11	Q	10 01
E	01 10	R	11 00
F	01 10	S	10 00
G	01 10	T	01 00
H	00 10	U	10 00
I	00 10	V	10 01
J	10 11	W	10 10
K	11 11	X	00 01
L	11 10	Y	01 01
M	11 10	Z	00 00

Figure 3: Partitioning of the VA-File.

However, the basic idea of these index structures is to administrate points hierarchically in a tree. The R-Tree partitions the data space using minimum bounding rectangles (MBR). A minimum bounding rectangle can be described by two points, being the end of the diagonal of the rectangle. Stepwise, the space is partitioned by MBRs, so that the superordinate MBR encloses all of its subordinate MBRs, as we visualize in Figure 1.

With increasing dimensionality, R-Trees face the challenge of overlapping MBRs. A query rectangle, situated in a region, where two or more MBRs overlap (like the MBR R2 and R3 in Figure 1), forces the R-Tree to follow up two or more different routes in the tree. Thus, the query performance decreases [6]. To overcome this disadvantage other index structures that we mentioned before, are developed.

Pyramid Technique.

An example for an exact space partitioning index structure is the Pyramid Technique, which was introduced by Berchtold et al. [5]. The Pyramid Technique divides an n-dimensional space into $2d$ pyramids [5]. A d dimensional normalized point x is inserted into a pyramid according to the dimension j_{max} with its maximum distance to the center of the data space. Thus, the pyramid number p_i is computed as follows:

$$i = \begin{cases} j_{max} & \text{if } x_{j_{max}} < 0,5 \\ (j_{max} + d) & \text{if } x_{j_{max}} \geq 0,5 \end{cases}$$

Second, for managing the space enclosed by a pyramid, the pyramids are divided in pyramid slices. According to the query types supported by the index structure, the partition of pyramids can be done in different ways. In Figure 2, we present two different possible methods for partitioning a pyramid, for a two dimensional normalized space.

In particular, the partition of Figure 2 (a) is proposed by Berchtold et al. [5] to support range queries. The other partition, shown in Figure 2 (b), is used by the approach of Lee and Kim [15] to support k-NN queries. It is possible to use the partitioning from Berchtold et al. for k-NN queries as well, but not in an efficient way. Anyway, a point is inserted into the slice depending on its distance to the center of the space. To sum up, for supporting different query types in an efficient way, different pyramid partitions are required.

VA-File.

In 1997, Weber and Blott [22] introduce the VA-File to overcome the curse of dimensionality. The VA-File is an improved sequential scan, because Weber et al. noticed a degeneration of most index structures to a sequential scan, if the dimensionality of data points exceeds a certain limit [23]. Hence, the authors propose to accelerate the sequential scan by using vector approximation.

The VA-File divides each dimension of the space into 2^b equally filled cells, where b is an user defined amount of bits per dimension. Each cell is labeled with a unique bit string, being the concatenation of the corresponding bit strings for every dimension. For every point, the bit string of the cell is stored, which the point is inserted into. Thus, the VA-File uses two lists: an approximation file that stores the bit string of the cells for every point and a vector file with the vector data for each point. An exemplary space partitioning and the corresponding approximation file can be seen in Figure 3.

Generally, the query algorithm of the VA-File traverses the whole approximation file to collect suitable candidates for the query result at first. After that, exact comparisons between the vector data of the candidates and the query are performed.

The approximation technique of the VA-File helps to reduce hard-disk accesses, because small bit strings can be kept in main memory. Even if the whole approximation file does not fit into the main memory, the sequential examination of the approximations reduces disk access costs compared to random accesses to many data items [22]. Another advantage is, in contrast to the Pyramid Technique, the availability of different algorithms to efficiently support all query types being executable on a sequential scan without adaption of the space partitioning of the VA-File.

2.4.2 Approximation-based Index Structures

Typical representatives for an approximation-based index structure are based on hash schemes. Apart from common hashing algorithms, scattering inserted data points over the amount of buckets is not applicable for similarity queries. Consequently, there is a need for hash functions, causing collisions when hashing locally near situated points. This challenge is handled by Locality Sensitive Hashing (LSH). The aim of LSH is to map the key to a one dimensional hash value. Thus, all comparisons are made on the hash value instead of a high dimensional key. Supporting nearest neighbor queries, LSH uses $(P1, P2, r, cr)$-sensitive functions h to compute the hash value. These functions h have to fulfill the following constraints [12]:

For every dataset in a d-dimensional space $p, q \in \mathcal{R}^d$:

1. $if\ ||p - q|| \leq r,\ then\ Pr[h(p) = h(q)] > P1$
2. $if\ ||p - q|| \geq cr,\ then\ Pr[h(p) = h(q)] < P2$

The first constraint demands that the probability for two points to be hashed into the same bucket has to be larger than $P1$ if their distance is smaller than r. Whereas, if their distance is bigger than cr, the probability should be smaller than $P2$. In order to be an useful locality sensitive function, $P1$ should be much bigger than $P2$.

Improving the precision of the index structure, usually several hash tables with different hash functions are used. Consequently, the need for $(P1, P2, r, cr)$-sensitive functions is obvious. A promising family of hash functions is used in p-stable LSH.

p-stable LSH.

The approach of p-stable LSH is based on p-stable distributions. A distribution $\mathcal{D}$ is p-stable for $p \geq 0$ if for any n the real numbers $v_1, ..., v_n$ and i.i.d. random variables $X_1, ..., X_n$ with distribution $\mathcal{D}$, the following constraint is fulfilled:

$$\sum_{i=1}^{n}(v_i X_i) \sim \Big(\sum_{i=1}^{n}(\|v_i\|^p)\Big)^{1/p} X,$$

$\sim$ means the operands have the same distribution and X is a random variable from the distribution $\mathcal{D}$ [18].

Using d random variables from $\mathcal{D}$ to form a d-dimensional Vector $\vec{a}$, the scalar of vector $\vec{a}$ and the data point $\vec{v}$ result in a random variable with distribution $\Big(\sum_{i=1}^{d}(\|v_i\|^p)\Big)^{1/p} X$ [12]. Several of these scalar products with different vectors can be used to estimate $\|\vec{v}\|_p$ (the L_p distance metric). The corresponding distributions are:

- The Cauchy Distribution $\mathcal{D}_{\mathcal{C}}(0, 1)$, defined by the density function $c(x) = \frac{1}{(\pi(1+x^2))}$ is 1-stable and can be used to estimate the Manhattan distance metric.
- The Gaussian (Normal) Distribution $\mathcal{D}_{\mathcal{G}}(0, 1)$, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is 2-stable and can be used to estimate the Euclidean distance metric.

Instead of estimating a distance metric, the scalar product with vectors from p-stable distributions can be used to compute hash values of the data points, because the scalar product maps the vectors to a one dimensional space. Furthermore, the result of the scalar product has the same distribution as the L_p distance metric, which guarantees the $(P1, P2, r, cr)$-sensitiveness.

3. ADVANCED CHALLENGES

After giving a short introduction to general challenges of indexing multi-dimensional data, in this section we provide existing challenges of evaluating the performance of index structures for a specific use case. For giving an overview of possible challenges when evaluating index structures, we group the challenges into three groups.

3.1 Parameter of the Index Structures

Some index structures have specific parameters for tuning their performance. Thus, when evaluating the performance of index structures, these parameter have to be considered as well. For index structures given in Section 2.4, these parameter are: the minimum and the maximum number of points within a MBR for the R-Tree, the number of slices a pyramid is divided in, for the Pyramid Technique, and the length of the bit vector for the VA-File. The parameters of the approximation-based index structure p-stable LSH presented in Section 2.4 are number of hash functions and width of hash buckets.

Thus, we have to assume, that these parameters have an impact on the performance of index structures. Therefore, it is necessary to analyze suitable parameter values when trying to identify an appropriate index structure for a given use case. However, there are some problems considering an appropriate value of some parameters. For example, the vectors used for p-stable LSH are randomly chosen from p-stable distributions. As a result of this random component it is possible that the performance and precision results of the same index structure created with different seeds of the random component can differ very much, within the same use case. This is problematic when trying to quantitatively evaluate the index structure.

3.2 Workload and used Queries

Although, two different applications can deal with the same data, they can have a different workload. For that reason, they can differ in requirements of index structures. The workload of an application depends on the used query types. Yet, it is obvious, not to use the Pyramid Technique presented from Berchtold et al. [5] for performing a k-NN query, but the version presented by Lee and Kim [15], because it is optimized for this query type.

For defining the workload of a database system we use a definition inspired by Ahmad et al. [1]. As a result, the workload is defined by the percentage of the query types used and the amount of concurrent requests performed.

3.3 DBMS Environment

In addition to use cases and workload, the test environment has an impact on the performance of each index structure. As already mentioned, the VA-File is optimized for database systems, storing data items on a disk and not in main memory. Evaluations of VA-File and sequential scan, result in different conclusions according to an evaluation with an in-memory database or a database storing items on the disk. Consequently, it is necessary to consider the underlying storage management of the database system as well.

Beside the storage management of a database, the amount of main memory and the CPU performance are other impact factors to the performance.

4. TOWARDS A FRAMEWORK

Since we aim at providing a comprehensive library of use cases and suitable indexes, we motivate a framework to give users the possibility to evaluate own use cases with different index structures. In this paper, we summarize key aspects of a framework that supports four groups. In Figure 4, we give an overview of these four groups.

4.1 Extensibility

First, the framework has to be extensible w.r.t. four key aspect, we present in Section 2. In other words, for an user, it has to be possible to implement, integrate, and evaluate own index structures. Furthermore, it has to be possible to extend the framework and existing index structures with additional distance metrics and also other query types. Finally, it has to be possible to integrate existing data in the framework and to create data with specific properties like a

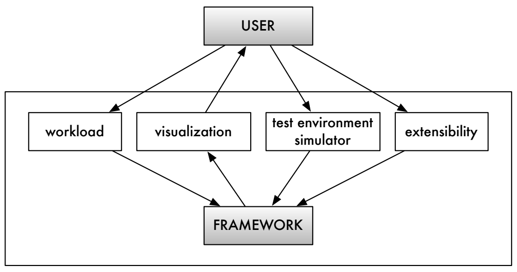


Figure 4: Structure of the Framework.

specific data distribution. Thus, creating data distributions is not trivial, an interface has to be created for importing existing data sets and communicating with systems like R, see for instance [14].

4.2 Adaptability to Different Workload

In real world applications, the workload differs quite much. On the one hand, there are use cases that use only read transactions. On the other hand, the workload can consist of read and write transactions. Thus, the performance results of workloads can differ very much. Hence, an interface is needed for importing workloads from existing systems. A further requirement, is to support standardized benchmarks, e.g. the TPC-H Benchmark[2].

Beside the queries used, the desired precision of the query results have to be defined by the user. Thus, if approximate results are allowed, the user has to define the accuracy of the results. Nevertheless, the precision depends on the data properties, the given distance metric, the used queries and the parameters of the index structure as well.

4.3 Test Environment Simulator

Existing index structures are created with respect to different optimization criteria. As already mentioned, the VA-File is optimized for reducing disk accesses. Consequently, another criteria our framework has to consider is the environment the tests are located in. Thus, within the framework a parameter has to exist, for setting whether the test is for an in-memory database or if a disk access is needed for accessing the data. Due to an assumption, that the access time of data differs very much considering Hard-Disk-Drives and Solid-State-Disks, the framework should have a component for virtualizing the disk access. With this component it is possible to perform tests on one system while simulating an access delay of another system. In addition to this storage device simulator, a simulator for all hardware components is required to give an useful hint about the best performing index structure.

Additionally, within the framework a parameter has to exist for defining the values of some index structure parameters such as the maximum number of data items of leaves of the R-Tree.

4.4 Visualization

In case the index structure has to struggle with a specific data distribution or query type, it can be useful to visualize the space partitioning of the index structure. With this visualization, further hypothesis can be drawn on the benefits or pit-falls of the chosen index structure. For instance, the user is able to follow the split of MBRs in the R-Tree and can easily identify overlapping regions while the tree is being constructed. Another aspect, being worth to visualize, is a statistic on query performance. These statistics help to analyze the performance of different index structures for a given workload or an index structure under different workloads. Apart form the query performance, other interesting values may be worth visualizing. The time spent on constructing the index structure is important for systems with many delete and update queries, because a reconstruction of the index is sometimes necessary when a certain threshold of changed data is reached. Furthermore, when using an approximate index structure, the precision of executed queries and the overall precision of the index structure is worth visualizing, because it has an impact on the suitability of an index structure for a special use case.

[2]http://www.tpc.org/tpch/

4.5 Working with the Framework

Finally, our framework shall help finding the most suitable index structures for a given use case. For this, the expected workload has to be known. These parameters include supported query types, exact or approximate results, data dimensionality and distribution, amount of data, the delay of the data access, and the environment. By finding suitable index structures for the given parameters, there are index structures that do not have to be taken into account, because they do not support certain query types or work approximately although the result is restricted to be exact. After excluding unsuitable index structures, the remaining index structures are evaluated under the given workload. By reviewing the performance results, the user can choose the suitable index structure for her use case.

5. RELATED WORK

In the last decades, many new index structures are created [5, 10, 22]. In addition, existing index structures are improved for supporting new query types [15] or to increase performance [6]. However, within the presented evaluation of these index structures only a small set of existing index structures is considered. For example, within Berchtold et al. [5], the Pyramid Technique is evaluated against X-Tree, Hilbert R-Tree, and sequential scan. Therefore, it is problematic to identify, which is the most appropriate index structure for a given problem. Additionally, different performance evaluations are done in different environments with different data characteristics. So, it is problematic to generalize the results of an evaluation.

For giving a comparison of the performance of multi-dimensional index structures, there already exists some frameworks, like the GiST [3] framework or the MESSIF [3] framework. In contrast to the framework we present here, these frameworks have some additional constraints. For example, the GiST framework only focuses on trees, hence no other multi-dimensional index structures such as the VA-File or the Pyramid Technique are considered, while the MESSIF framework only focuses on metric data. Another framework limiting the available index structures is introduced by Muja et al. [17]. The aim of this framework is to optimize parameters of approximate index structures in order to match the required precision under given data distributions.

[3]http://gist.cs.berkeley.edu/

6. CONCLUSION

In this paper, we provide an overview of existing challenges in finding an appropriate index for multi-dimensional data for a specific use case. First, we explain distance metrics and common query types that have to be considered. Second, the parameters of the index structures can have an impact on the performance of an index structure. Third, for users, it has to be possible to define own workload pattern and the environment, the application is located in.

For supporting these characteristics of real-world use cases we present requirements of a framework we intend to develope. Our framework has to support four key aspects. Namely, it has to be extensible, support different workload patterns, virtualize different use case environments, and contain a visualization component for improving user experiences.

7. ACKNOWLEDGMENTS

The work in this paper has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Programme under Contract No. FKZ:13N10817 and FKZ:13N10818. Additionally we want to thank Sandro Schulze for giving us useful comments.

8. REFERENCES

[1] M. Ahmad, A. Aboulnaga, and S. Babu. Query interactions in database workloads. In *Proc. Int'l. Workshop on Testing Database Systems*, DBTest, pages 11:1–11:6. ACM, 2009.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[3] M. Batko, D. Novak, and P. Zezula. Messif: Metric similarity search implementation framework. In *Proc. Conf. on Digital Libraries (DELOS)*, 2007.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. Int'l. Conf. on Mgmt. of Data (SIGMOD)*, pages 322–331. ACM, 1990.

[5] S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid-Technique: Towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27:142–153, 1998.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high-dimensional data. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 28–39. Morgan Kaufmann Publishers Inc., 1996.

[7] C. Böhm. *Efficiently Indexing High-Dimensional Data Spaces.* PhD thesis, Ludwig-Maximilians-Universität München, 1998.

[8] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33:322–373, 2001.

[9] P. H. Bugatti, A. J. M. Traina, and C. Traina, Jr. Assessing the best integration between distance-function and image-feature to answer similarity queries. In *Proc. ACM Symp. on Applied Computing (SAC)*, pages 1225–1230. ACM, 2008.

[10] E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(9):1647–1658, 2008.

[11] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *SIGMOD'84, Proc. of Annual Meeting*, pages 47–57, 1984.

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. Symp. on Theory of Compu. (STOC)*. ACM, 1998.

[13] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. Int'l. Conf. on Mgmt. of Data (SIGMOD)*, pages 369–380. ACM, 1997.

[14] V. Köppen. *Improving the Quality of Indicator Systems by MoSi – Methodology and Evaluation.* PhD thesis, Freie Universität Berlin, 2008.

[15] D.-H. Lee and H.-J. Kim. An efficient technique for nearest-neighbor query processing on the SPY-TEC. *Trans. on Knowl. and Data Eng. (TKDE)*, 15:1472–1486, 2003.

[16] B. Li, E. Chang, and Y. Wu. Discovery of a perceptual distance function for measuring image similarity. *Multimedia Systems*, 8(6):512–522, Apr. 2003.

[17] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. Int'l. Conf. on Computer Vision Theory and Applications (VISAPP)*, pages 331–340, 2009.

[18] J. P. Nolan. *Stable distributions: Models for heavy tailed data.* Springer-Verlag, 2009.

[19] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 516–526. Morgan Kaufmann Publishers Inc., 2000.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann Publishers Inc., 2005.

[21] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 507–518. Morgan Kaufmann Publishers Inc., 1987.

[22] R. Weber and S. Blott. An approximation-based data structure for similarity search. Technical Report ESPRIT project, no. 9141, 1997.

[23] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 194–205. Morgan Kaufmann Publishers Inc., 1998.

[24] R. Zhang, B. C. Ooi, and K.-L. Tan. Making the pyramid technique robust to query types and workloads. In *Proc. Int'l. Conf. on Data Engineering (ICDE)*, pages 313–324. IEEE Computer Society, 2004.