Extending an Index-Benchmarking Framework with Non-Invasive Visualization Capability

David Broneske[1], Martin Schäler[1], Alexander Grebhahn[2]

[1] Department of Technical and Business Information Systems
Otto-von-Guericke University Magdeburg
david.broneske@st.ovgu.de, schaeler@iti.cs.uni-magdeburg.de

[2] Brandenburg University of Applied Sciences
grebhahn@fh-brandenburg.de

Abstract: Finding a suitable multi-dimensional index structure for a data-intensive system is not a trivial task. The QuEval framework supports users in finding the best index structure from a list of candidates. Nevertheless, if an index structure shows itself superior to other index structures most oft the times, but fails for one data set, we want to know the reason for this phenomenon. To support an understanding of deficits, a visualization of the partitioning scheme is helpful. Consequently, we propose a visualization component which interacts with QuEval without affecting the performance evaluation. Thus, we use a modern software-engineering approach based on AspectJ to support Digital Engineering of complex solutions.

1 Introduction

Modern forensic investigation makes increasing use of digitally stored evidences. For this purpose, on the crime scene latent fingerprint scans are executed [KFV11]. These fingerprints are stored in a database to find possible suspects. Querying a fingerprint database bears challenges, e.g. given workload [GG98], data dimensionality and used query types [GBS+12]. These challenges are addressed by multi-dimensional index structures. Numerous multi-dimensional index structures are introduced [GG98, Sam05], but the most suitable index structure depends on the current scenario.

To find the most suitable index structure for a given use case, the QuEval framework[1] can be used. QuEval offers an expendable list of index structures, customizable data sets and different query types, too. Working with QuEval, we observed performance issues of index structures with some data sets. Consequently, we have to figure out partitioning deficits under these circumstances. As a consequence, the need for a visual representation arises to examine the partitioning scheme and to derive deficits [GBS+12]. Furthermore, the sequence of operations (e.g. inserts and updates) influence the partitioning of index structures [NT01]. Thus, two different sequences performing the same operations lead to different structures. In summary, such a visualization component shall, for instance help engineers to decide whether identified performance problems are due to limitations of the index structure or may be due to erroneous data gathering procedures.

[1] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/iJudge/index_en.php

The research question, we address is whether we can generalize the visualization of index structures and whether we can automatically extract necessary information. To implement such a visualization component, we have to face two tasks:

(a) a suitable visualization of indexed regions for multi-dimensional data and
(b) the information extraction from QuEval and integration into the framework without remodeling of already implemented index structures.

To summarize, we need to integrate a non-invasive visualization that does not affect non-functional properties of the index structures, such as response time and memory consumption. Consequently, we need a modern software-engineering approach that allows seamless integration and unintegration of the additional functionality.

The paper is structured as follows: In Section 2, we show relevant properties of index structures for our visualization followed by our visualization and software requirements. We show their implementation in Section 4 and 5 and evaluate the implementation of our requirements in Section 6. We finish with related work, conclusion and future work.

2 Impact of Index Structure Categories on the Visualization

In literature, many different index structures are proposed. To give an overview of existing index structures, we categorize them either as tree techniques, optimized sequential search, dimensionality reduction or hashing methods [GG98]. The following paragraphs will give a brief overview of these categories, which will help us to figure out specific requirements on the visualization. For more detailed information, we refer to the given references.

Tree Techniques. Tree techniques partition the space hierarchically to improve the query performance. They partition the space so that one superordinate regions encloses several subordinate regions. This builds a hierarchical tree structure. With this, the complexity should decrease from $\mathcal{O}(n)$ to $\mathcal{O}(log(n))$ for point search. An important multi-dimensional tree technique is the R-Tree [Gut84] as well as its derivates R^+-Tree [SRF87] and R^*-Tree [BKSS90]. In Figure 1, we visualize the SS-Tree [WJ96] using multi-dimensional spheres and the SR-Tree [KS97] whose regions are the intersection between a multi-dimensional sphere and rectangle. However, tree techniques suffer from the curse of dimensionality, which says that at a certain dimensionality (for tree techniques ca. 26), the nearest neighbor query performance of an index structure will be worse than a sequential search [WSB98]. Thus, a visualization of the query execution would be helpful to show these deficits. When visualizing the partitioning scheme, it is important to have a good representation for the hierarchy. Since regions on the same level may overlap, the belonging of subordinate to superordinate have to be encoded in the visualization strategy.

Optimized Sequential Search. Since tree-based index structures are affected by the curse of dimensionality, methods using an optimized sequential search are introduced which compare every data point with the query. The optimization is often an approximation of the actual data points to reduce comparison costs. For example, the VA-File [WB97] splits the whole space into cells

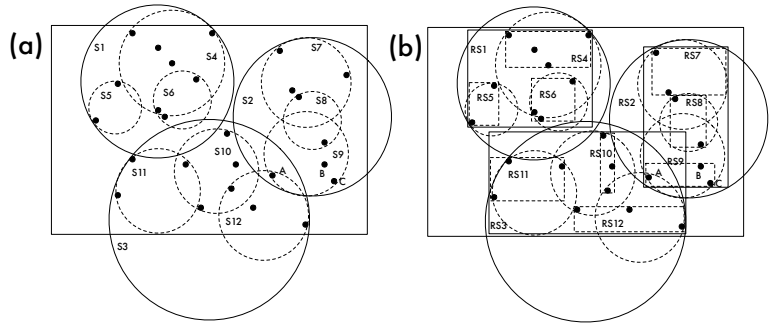

Figure 1: (a) SS-Tree, (b) SR-Tree

and assigns a unique bit string for each. For each data point the bit string of its cell will be stored. As a result, many bit strings of points fit into main memory and disk accesses are reduced. The partitioning should result in almost equally filled regions, nevertheless the data distribution may interfere with this goal. Consequently, visualizing the partitioning is helpful.

Dimensionality Reduction. Another option to avoid the curse of dimensionality is to decrease the dimensionality of the space. This can be done for example using space-filling curves. A space-filling curve is a mathematical function, which is constructed iteratively assigning a one dimensional value to each indexed region. Typical space-filling curves are the Z-Curve [OM84], used in the UB-Tree [Bay97], and the Hilbert-Curve [FR89]. Considering nearest neighbor queries, it is helpful to see neighboring regions in the space-filling curve and whether all of them are evaluated.

Hashing Methods. As a hashing method for multi-dimensional data, Locality Sensitive Hashing (LSH) [IM98] is most often used, because they support nearest neighbor queries. LSH uses locality sensitive functions to set up several hash tables with the constraint that all points in one bucket should, with a high probability, be locally nearer to each other than to any point in another bucket. With this constraint, neighborship relations are preserved. Promising LSH methods are p-stable LSH [DIIM04] whose partitioning looks similar to stairs and the permutation approach [CGFN08]. Since hashing methods use several hash tables, a visualization of several partitionings is necessary.

3 Requirements for a Visualization Component

For our visualization component, we divide the requirements into (a) visual and (b) software requirements. The visualization requirements define properties that help to show the functionality of an index structure, whereas software requirements demand that no runtime or memory consumption overhead occurs from additional visualization implementation.

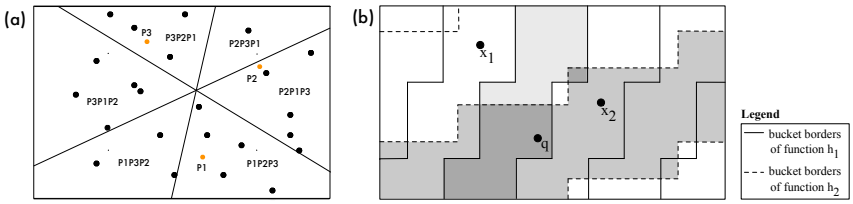

Figure 2: (a) permutation Approach of LSH, (b) p-stable LSH

3.1 Visual Requirements

The visual requirements include visualizing the (a.1) partitioning scheme, (a.2) hierarchies, (a.3) queries and it should be able to (a.4) support different dimensions of data.

(a.1) Giving the user an understandable picture of an index structure, we want to visualize the partitioning schema step by step when constructing the index structure. With this, users should be able to figure out the functionality of the index structure and understand important algorithms, for example different split algorithms of the R-Tree. Even a comparison of the same index structure with different algorithms can be done with a visual representation. Another advantage is that optimizations can be estimated, when phases are seen where the partitioning results in overfull regions.

(a.2) Another challenge is to represent hierarchies. Visualizing a hierarchical index structure without visual support for different hierarchies, it may be hard to differentiate between different hierarchy levels.

(a.3) Additionally, we want to highlight regions that are accessed when a query is performed. With this visualization, users can identify regions that are frequently evaluated, because they are too large or overfull. Considering hierarchical index structures, we can additionally identify regions that are too near or even overlapping so that multiple paths have to be taken while evaluating a query.

(a.4) An important role is to support different dimensions. Since multi-dimensional data implies a dimensionality of up to 100 [VCPF08], we also have to be able to visualize all of these dimensions and dependencies between them.

3.2 Software Requirements

Considering the implementation, we define the following requirements: (b.1) no adjustments to index interface given and used by QuEval, (b2.) no performance deficits while evaluation, (b.3) a common interface for the extracted visual information.

(b.1) Our visualization component will be integrated into QuEval. QuEval is a Java framework for finding the most suitable index structure for a given use case. To have a comprehensive collection of index structures, an extension of the collection can be done very easily. Users

do only have to implement a few functions to conform to the simple interface. For more information visit our website[2]. When integrating a visualization component, the interface should not be change to assure an easy implementation of index structures.

(b.2) Since QuEval is a tool for evaluating the query performance of an index structure, our visualization should not affect the performance of the index structure. Consequently, we must not produce a computational overhead for our visualization when an evaluation is in progress. Thus, necessary code for extracting visual information should only be present, if we want to visualize an index structure and no performance benchmark between several index structures is running.

(b.3) Another requirement refers to the extracted information, which will be used to visualize the partitioning scheme. To have a suitable representation of the extracted information, we have to find a common interface to describe index structures.

We visualize a summary of our requirements in Table 1. In the following both sections, we will present first solutions for the given problems and primary concepts of our implementation.

	Visual requirement		**Software requirement**
(a.1)	visualizing partitioning scheme	(b.1)	no adjustment to index interface
(a.2)	support hierarchies	(b.2)	no performance deficits
(a.3)	visualizing query evaluation	(b.3)	common interface
(a.4)	visualizing different data dimensions		

Table 1: Visual and software requirements

4 Visual Representation

In this section, we discuss implementation of the visual requirements, we defined before. We visualize a screenshot of our tool visualizing the R-Tree in Figure 3.

(a.4). To support a multi-dimensional representation of the data and partitioning scheme, we use a scatterplot matrix. A scatterplot is the visual representation of a Cartesian space where the data points are arranged. Although 3D-Scatterplots can be extended using color, shape, and size [EDF08], a single scatterplot does not suffice our requirements for visualizing multi-dimensional data. To solve this problem, we use a scatterplot matrix having the dimensions as rows and columns. Each entry represents an own 2D-scatterplot visualizing the data distribution in the corresponding dimensions. We only visualize the upper triangular matrix (cf. 3), because the lower scatterplots are mirrored duplicates. Nevertheless, with increasing dimensionality, we face the problem of an overcrowded display. Nevertheless, we cannot use an approximation of the data, because the partitioning schema relies on a multi-dimensional vector space. Changing the visualization paradigm (a collection can be found here [Kei02]) would destroy the relation between data and the corresponding partitioning scheme of the index structure.

[2] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/iJudge/index_en.php

(a.1), (a.3). To visualize the construction and query execution, we identify *two-dimensional* geometrical shapes the index structures are based on. Most of the index structures use simple geometrical shapes (here: line, rectangle, sphere):

- Space-filling curves and permutation approach of LSH use lines.
- The R-Tree and its derivates as well as the VA-File are based on rectangles.
- Spheres are used in the SS-Tree.

These geometrical shapes can easily be drawn, if we extract the necessary information. For instance, we only need the end points of the lines to draw it or the end points of the diagonal for a rectangle. To draw a sphere, the center point and the radius have to be known.

Nevertheless, there are some exceptions that are not based on these simple shapes. An index structure that is using multiple geometrical shapes is the SR-Tree. One region is the intersection of a minimum bounding rectangle and a minimum bounding sphere. The visualization of such a complex region is hard to implement without further computations (cf. Figure 1.b), so we decided to draw both shapes. Complex regions are also created when using p-stable LSH, as the region of the hash buckets are similar to stairs and one region cannot be constructed using one simple geometries (cf. Figure 2.(b)).

(a.2). When visualizing hierarchical index structures, we have to support the user in distinguishing between regions of different levels. For this, we provide the individual tree structure, where the user can filter the visualized regions. By hovering over or selecting a node in the tree in the right half of the tool, only the node and its subtree is visualized in the scatterplot matrix.

5 Integration into QuEval

This section addresses our implementation of software requirements, we defined in Section 3.

(b.2). The necessary code for our visualization component is only integrated, when a visualization is performed. An overhead should not be present, when a performance evaluation is done. For this, we use AspectJ to integrate our code into QuEval.

AspectJ is an extension to Java which introduces a set of language constructs to permit aspect-oriented programming in Java [KHH+01]. Aspect-oriented programming aims at implementing crosscutting problems in one programming unit, namely an *aspect*. The code of an aspect is weaved into code, when the aspect is activated. Otherwise, the code will not be taken into account and no computational or storage overhead occurs. The points where aspects add their code will be defined by pointcuts which can extend methods, fields, or classes. A detailed overview of AspectJ can be found in [KHH+01].

(b.1). With AspectJ, changes in the classes are weaved into code when the corresponding module is activated. Thus, the interface stays as simple as it was. Nevertheless, users have to implement suitable AspectJ modules for their index structures, if they want to use the visualization component.

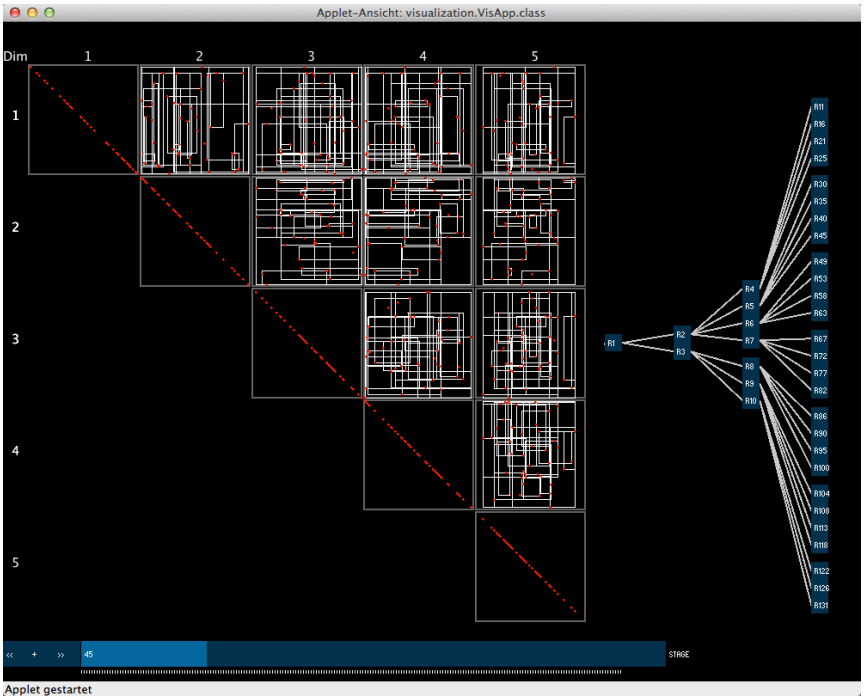

Figure 3: Screenshot of our visualization tool

(b.3). The AspectJ modules should hook in at points, where the partitioning scheme changes or a query is executed and extracts the necessary information. Per stage (state after an insertion or deletion) of the index structure, an array of regions is constructed. We describe a region as a pair of a geometrical object and the parent-ID which encodes the hierarchy. If the index structure is non-hierarchical, the parent-IDs are `NULL`. Consequently, we describe a region with the following interface:

$$(GeoObj, parentID)$$

A geometrical object has an ID, a type, and a location. The simple geometrical shapes (cf. Section 4) can be encoded by two arrays, because lines and rectangles can be described by two two-dimensional points. A sphere consists of one two-dimensional point and the radius of the sphere, so the first entry of the second array denotes the radius while the rest is zero. Consequently, a geometrical object can be encoded as:

$$(ID, Type, MinPoint, MaxPoint)$$

The extracted information can be used to visualize the index structure. To have a reconstructable image of the index structure, the stages are stored and visualized after a full run of evaluation done by QuEval. Then, users can stepwise switch through the stages and see what is happening.

6 Evaluation of Requirements

In this section, we summarize the implementation of our requirements, which is represented in Table 2.

Our visualization component supports hierarchies and queries can be visualized on the partitioning scheme. Nevertheless, there are index structures that are hard to visualize at the current stage, for example p-stable LSH. Furthermore, we visualize multi-dimensional data using a scatterplot matrix, but an increasing number of dimension the user loses the overview.

Considering the software requirements, AspectJ helps us to extend QuEval without performance impact or extension of the index interface. Furthermore, we have presented an interface describing our simple geometrical objects.

	Visual requirement			**Software requirement**	
(a.1)	visualizing partitioning scheme	≈	(b.1)	no adjustment to index interface	✓
(a.2)	support hierarchies	✓	(b.2)	no performance deficits	✓
(a.3)	visualizing query evaluation	✓	(b.3)	common interface	✓
(a.4)	visualizing different data dimensions	≈			

Legend: ≈ - partially implemented; ✓ - completely implemented

Table 2: Implementation of visual and software requirements

We have shown, that it is possible to visualize many index structures. For the presented index structures, it is possible to have a generalized representation which can be used for a visualization. Nevertheless, the programmer of the index structure has to provide the necessary AspectJ modules for each index structure that has to be visualized.

7 Related Work

Related work has already been done by Keim and Kriegel [KK94]. With VisDB, they introduce a system for an exploration of databases. They present their own visualization paradigm using grouping and transformation of the data into a two-dimensional screen to have a compressed view on the data. Furthermore, they visualize queries on the data which can be redefined by users to get a more meaningful query result. In fact, the aim of VisDB is related to our topic. Nevertheless, we cannot transform the data, because the relation between data and the corresponding partitioning scheme would be destroyed.

Another important visualization framework is the scalable framework [KLS00]. Since there is a limited number of objects that can be visualized, the authors propose several techniques, such as Self-Organizing Maps or Magic Eye View to represent high amounts of data points. Such techniques may be helpful to give an overview of scatterplots in the matrix, because the two-dimensional space is displayed in a small pane.

8 Conclusion and Future Work

Visualizing index structures is a promising task to show their functionality and identify disadvantageous partitioning. We have shown, that there are many opportunities to support user's perception considering the partitioning scheme, hierarchies and query execution. The partitioning scheme can be visualized using simple geometric shapes. To extract shapes that have to be visualized, we use AspectJ to weave the necessary code into the index structure. We presented an encoding for the information that will be used in our visualization component.

Further challenges refer to the dimensionality. At the moment, we use scatterplot matrices to visualize the multi-dimensional space. However, with increasing dimensionality, the matrix gets huge and a single scatterplots is barely visualizable. Furthermore, the user is overflooded by numerous scatterplots which makes it hard to make important findings. Another task would be to help the user in comparing the same index structure with different partitioning algorithms or inserting the same data in different orders. Displaying two evaluations side by side would support an understanding of differences in algorithms or insertion orders.

Acknowlegements

The work in this paper has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract No. FKZ: 13N10817 and FKZ: 13N10816.

References

[Bay97] Rudolf Bayer. The universal B-Tree for multidimensional indexing: General concepts. In *Proc. Int'l Conf. on Worldwide Computing and its Application (WWCA)*, pages 198–209. Springer-Verlag, 1997.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. Int'l Conf. on Management of Data (SIGMOD)*, pages 322–331. ACM, 1990.

[CGFN08] Edgar Chavez Gonzalez, Karina Figueroa, and Gonzalo Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Patt. Anal. and Machine Intell. (TPAMI)*, 30(9):1647–1658, 2008.

[DIIM04] Mayur Datar, Piotr Indyk, Nicole Immorlica, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. Symp. on Comput. Geometry (SCG)*, number 3, pages 253–262. ACM, 2004.

[EDF08] Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Trans. Visualization and Computer Graphics (TVCG)*, 14:1141–1148, 2008.

[FR89] Christos Faloutsos and Shari Roseman. Fractals for secondary key retrieval. In *Proc. Symp. Principles of Database Systems (PODS)*, pages 247–252. ACM, 1989.

[GBS+12] Alexander Grebhahn, David Broneske, Martin Schäler, Reimar Schröter, Veit Köppen, and

Gunter Saake. Challenges in finding an appropriate multi-dimensional index structure with respect to specific use cases. In *Grundlagen von Datenbanken Workshop*, pages 77–82, 2012.

[GG98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[Gut84] Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. Int'l. Conf. on Management of Data (SIGMOD)*, pages 47–57. ACM, 1984.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. Symp. on Theory of Computing (STOC)*. ACM, 1998.

[Kei02] Daniel A. Keim. Information visualization and visual data mining. *IEEE Trans. Visualization and Computer Graphics (TVCG)*, 8(1):1–8, January 2002.

[KFV11] Tobias Kiertscher, Robert Fischer, and Claus Vielhauer. Latent fingerprint detection using a spectral texture feature. In *Proc. ACM workshop on Multimedia and Security (MMSec)*, pages 27–32. ACM, 2011.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *Proc. Europ. Conf. on Object-Oriented Programming (ECOOP)*, pages 327–353. Springer-Verlag, 2001.

[KK94] Daniel A. Keim and Hans-Peter Kriegel. VisDB: Database exploration using multidimensional visualization. *IEEE Comput. Graph. Appl.*, 14(5):40–49, September 1994.

[KLS00] Matthias Kreuseler, Norma Lopez, and Heidrun Schumann. A scalable framework for information visualization. In *Proc. Symp. on Information Vizualization (INFOVIS)*, pages 27–37, 2000.

[KS97] Norio Katayama and Shin'ichi Satoh. The SR-Tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. Int'l. Conf. on Management of Data (SIGMOD)*, pages 369–380. ACM, 1997.

[NT01] Moni Naor and Vanessa Teague. Anti-persistence: History independent data structures. In *Proc. ACM Symp. on Theory of computing (STOC)*, pages 492–501. ACM Press, 2001.

[OM84] Jack A. Orenstein and Tim H. Merrett. A class of data structures for associative searching. In *Proc. Symp. on Principles of Database Systems (PODS)*, pages 181–190. ACM, 1984.

[Sam05] Hanan Samet. *Foundations of multidimensional and metric data structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[SRF87] Timos K. Sellis, Nick Roussopoulos, and Christos 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.

[VCPF08] Eduardo Valle, Matthieu Cord, and Sylvie Philipp-Foliguet. High-dimensional descriptor indexing for large multimedia databases. In *Proc. Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 739–748. ACM, 2008.

[WB97] Roger Weber and Stephen Blott. An approximation-based data structure for similarity search. Technical report, ETH Zürich, 1997.

[WJ96] David A. White and Ramesh Jain. Similarity indexing with the SS-Tree. In *Proc. Int'l Conf. on Data Engineering (ICDE)*, pages 516–523. IEEE Computer Society, 1996.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen 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.