

# Efficient Nearest Neighbor Retrieval by Using a Local Approximation Technique - the *Active Vertice* Approach

Sören Balko Ingo Schmitt

Otto-von-Guericke-Universität Magdeburg  
Institut für Technische und Betriebliche Informationssysteme  
Postfach 4120, D-39016 Magdeburg, Germany  
{balko|schmitt}@iti.cs.uni-magdeburg.de

March 2002



## **Abstract**

In this paper we propose the Active Vertice approach for efficient high-dimensional nearest neighbor retrieval. In contrast to existing proposals the Active Vertice approach is classified as a sequential local approximation technique which is not based on the well-known VA-file.

In comparison to the VA-file approach, we obtain a superior retrieval performance on uniform and real data sets for low, mid and very high dimensionalities. In addition, we formally relate the Active Vertice approach to the VA-file fundamentals and analytically confirm the improved retrieval performance.



---

---

# Contents

<b>List of Figures</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Related Work . . . . .	1
1.2 The VA-file . . . . .	3
<b>2 The Active Vertice Approach</b>	<b>5</b>
2.1 Point Approximation . . . . .	5
2.2 Point Insertion . . . . .	7
2.3 Maintenance . . . . .	8
<b>3 Formal Cost Comparison</b>	<b>13</b>
3.1 LB-Values of the AV-tree . . . . .	14
3.2 LB-Values of the VA-file . . . . .	17
<b>4 Experimental Results</b>	<b>21</b>
4.1 Parameterization . . . . .	23
4.2 Performance Comparison . . . . .	23
<b>5 Conclusions</b>	<b>31</b>
<b>Bibliography</b>	<b>33</b>



# List of Figures

1.1	VA-file in $\mathbb{R}^2$ . . . . .	3
1.2	Algorithm <code>retrieveNN</code> . . . . .	4
2.1	Reference Points at Different AV-tree Levels . . . . .	5
2.2	Structure of a Bit Code $B = B^1 \otimes B^2 \otimes B^3$ . . . . .	6
2.3	Algorithm <code>insertPoint</code> . . . . .	7
2.4	AV-tree in $\mathbb{R}^2$ . . . . .	8
2.5	AV-tree Mapping to Prefix-tree . . . . .	9
2.6	Algorithm <code>mapTree</code> . . . . .	10
3.1	Radii at Different AV-tree Levels $t$ . . . . .	16
3.2	Probability as Volume . . . . .	18
3.3	Plotted Values of $t$ . . . . .	20
4.1	Optimal Parameterization ( $d = 360$ ) . . . . .	22
4.2	Parameterization ( $d = 20, \dots, 360$ ) . . . . .	25
4.3	Parameter Profile ( $d = 20, \dots, 360$ ) . . . . .	26
4.4	Performance Comparison ( $d = 9, 16$ ) . . . . .	27
4.5	Performance Comparison ( $d = 32$ ) . . . . .	28
4.6	Performance Comparison ( $d = 100, 200$ ) . . . . .	29
4.7	Performance Savings ( $d = 20, \dots, 360$ ) . . . . .	30



# Chapter 1

## Introduction

In recent years the problem of high-dimensional indexing has attracted much interest in the database community. The problem of nearest neighbor in general can be formulated in a metric space where a set of database objects, a query object and a metric norm are given: *Find the database object which has the smallest distance to the query object according to the metric norm.*

### 1.1 Related Work

[CPZ97] introduced the M-tree to support fast nearest neighbor search in a metric space. An improvement is the Slim-tree proposed in [JTSF00]. Since node overlap directly affects the query performance the Slim-tree is designed to reduce such an overlap.

In [FTJF01] the authors map database objects to points where a metric norm is used to compute spatial distances between database objects and focus points.

In many applications database objects can be directly represented by points in a high-dimensional space. [YF00] introduces a general indexing schema for time sequences when the distance function is any of arbitrary  $L_p$  norms. Range queries using the  $L_\infty$  norm can be efficiently handled by the iMinMax structure proposed in [OTYB00].

Some approaches reuse a one-dimensional index structure, e.g. one of the B-tree family. In the Multi-Index, see [LHC01], along each dimension an index is constructed according to the values of feature points. The iDistance approach proposed in [YOTJ01] builds spherical clusters and uses  $B^+$ -trees to store distances between data points and cluster centers.

Most indexes exist for  $L_2$ -based nearest neighbor search among points in high-dimensional space. The classic data partitioning index is the R-tree [Gut84]. Many further refinements are proposed, among them the X-tree [BKK96] and the SR-tree [KS97].

A general problem of data partitioning indexes, often referred to as “curse of dimensionality”, is discussed in [WSB98]. The authors show, that searching in high-

dimensional indexes is not faster than a simple sequential scan when the number of dimensions is high. The general restriction of data partitioning methods based on convex shapes is discussed in [Sch01]. [BGRS99] introduces the term “contrast” to describe the fact that high-dimensional points tend to have similar distances from each other. [GR00] relates the performance of their proposed P-sphere-tree to a contrast distribution.

Due to the inherent problem of high-dimensional indexing [WSB98] suggests a totally different approach. The authors introduce the VA-file which is an approximation of data points and which is sequentially scanned to search the nearest neighbor efficiently. This approximation approach reduces the sequential scan cost by a constant factor.

Later approaches combine data partitioning and approximation. In the A-tree, see [SYUK00], minimum bounding rectangles (MBR) from a data partition tree are approximated by quantization. In this way, the tree fanout is enlarged. However, the A-tree is still a data partitioning tree which suffers from the curse of dimensionality.

Another approach is the IQ-tree proposed in [BBJ<sup>+</sup>00]. Whereas in the VA-file all points are globally approximated according to one fixed grid the IQ-tree supports a flexible approximation grid reflecting different data distributions. A certain approximation area is localized by a minimum bounding rectangle. This approach dynamically finds the separation between indexing using MBR’s and a sequential scan by use of a cost model. However, due to the deficiencies of MBR indexing in high-dimensional space the separation will be in favor of the sequential scan when the number of dimensions is high. Less MBR’s, however, mean a global, less flexible approximation approaching the costs of the VA-file. Experiments show the superiority of the IQ-tree in comparison to the VA-file up to 16 dimensions which is too small for many applications.

In this work we propose the *Active Vertice* approach. The advantages of this method are:

- it dynamically adapts the approximation accuracy to the data distribution;
- the adaption is independent from a data partitioning by convex regions and, henceforth, does not suffer the dimensionality curse;
- it outperforms the VA-file even in high-dimensional space both for reproducible artificial uniform data and real data.

Our basic idea is to approximate data points by prefix paths through a Quad-tree-like structure. Please note, although concepts of the Quad-tree are used, single data points are approximated and sequentially scanned in order to find nearest neighbors.

Some of the previously discussed spatial access methods together with our method are classified in Table 1.1.

In the following we relate our Active Vertice approach to the VA-file. Therefore, we will recall this approximation schema in the following subsection.

	no approx.	global approx.	local approx.
<i>tree</i>	R*-tree, SR-tree		A-tree
<i>hybrid</i>	X-tree		IQ-tree
<i>scan</i>	lin. scan	VA-file	<b>Active Vertice</b>

Table 1.1: Classification of Spatial Access Methods in Accordance to the Classification Proposed in [SYUK00]

## 1.2 The VA-file

In [WSB98] the *VA-file* approach was presented which totally separates the vector space into non-overlapping approximation boxes by means of a fixed grid. That is, a number of  $b$  bits per dimension specifies the grid by addressing  $2^b$  equally long intervals. In Figure 1.1, a VA-file ( $b = 3$ ) approximating three data points ( $p_1$ ,  $p_2$ , and  $p_3$ ) in  $\mathbb{R}^2$  is depicted. Obviously, in  $d$  dimensions,  $d \cdot b$  bits jointly specify  $2^{db}$  approximation boxes.

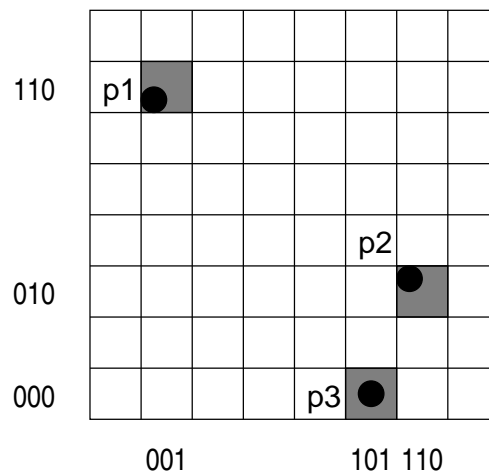


Figure 1.1: VA-file in  $\mathbb{R}^2$

A data point is approximated by the bit code of its containing approximation box. For a number of  $N$  data points, we obtain an array of  $N$  approximation bit codes which can be stored in a flat file which totally requires  $N \cdot d \cdot b$  bits of storage space.

The retrieval process roughly bases on the GEMINI [Fal96] approach. That is, the algorithm comprises two stages. In the first stage (filtering step) the array of approximation bit codes is sequentially processed into a candidate list. In the second step, the data points assigned to the candidates are retrieved from the database and scanned for the nearest neighbor point.

In Figure 1.2 this retrieval algorithm is depicted. Please note, that the algorithm was adopted from the VA-file and abstains from using a SAM (e.g. R-tree) in the second stage.  $B_i$  identifies the bit code assigned to a data point  $p_i$  (with  $1 \leq i \leq N$ ),  $q$  denotes the query point, and  $\text{LB}(q, B_i)$  ( $\text{UB}(q, B_i)$ ) computes the lower (upper) bound distance from a query point to the approximation box specified by  $B_i$ . The variable `min` contains

```

function retrieveNN(q: Point): Point
  candidates = {}; nn = p1; min = ||q - nn||2;
  1st stage
  for i = 1 to N do
    if LB(q, Bi) ≤ min then
      if UB(q, Bi) < min then min = UB(q, Bi);
      candidates = candidates ∪ {Bi};
    endif
  od
  sort candidates ascendingly by LB;
  2nd stage
  for each Bi ∈ candidates do
    if LB(q, Bi) ≤ min then
      if ||q - pi||2 < min then
        nn = pi; min = ||q - pi||2;
      endif
    endif
  od
return nn

```

Figure 1.2: Algorithm retrieveNN

the threshold distance between  $q$  and the nearest neighbor data point (`nn`) found so far. That is,  $\|q - \text{nn}\|_2 \leq \text{min}$  holds.

Obviously, there is a trade-off between compression quality (i.e.  $b$ ) and the number of remaining candidates after the filtering step. On the one hand, a better compression quality causes larger bit code representations which require more space for their storage. On the other hand, a smaller list of remaining candidates requires less data points to be fetched from the data base.

In the following, without loss of generality, we assume normalized data points to be processed. That is, all points are elements of the  $d$ -dimensional unit hypercube  $[0, 1]^d$ . Furthermore, we use the Euclidean distance  $L_2$ :

$$\|p_i - p_j\|_2 = \sqrt{\sum_{k=1}^d |p_i[k] - p_j[k]|^2}$$

# Chapter 2

## The Active Vertice Approach

In this chapter, we introduce the *Active Vertice Tree* (AV-tree) as part of a the Active Vertice approach for high-dimensional indexing. We use the term “tree” to express the hierarchical approximation principle. Data point approximations are themselves considered as paths through the AV-tree.

Following the VA-file approach, the AV-tree approximates the locations of high-dimensional data points by using bit codes. However, in contrast to the VA-file we do not utilize a fixed grid which covers the complete vector space. Instead, the AV-tree reuses some ideas of the Quad-tree where the unbalanced structure reflects the actual data distribution.

### 2.1 Point Approximation

As opposed to the (two-dimensional) Quad-tree which comprises quadratic regions, the AV-tree hierarchically localizes data points by means of *reference points*. Reference

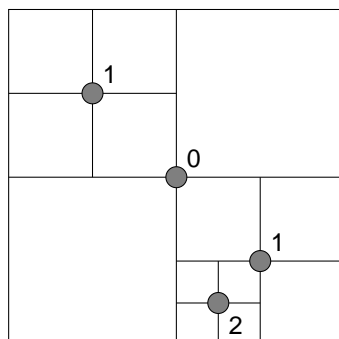


Figure 2.1: Reference Points at Different AV-tree Levels

points represent hypercube centers in a (high-dimensional) Quad-tree. Thus, reference

points are attached to different levels of the AV-tree hierarchy. Figure 2.1 depicts reference points labeled by their tree level.

The precise location of a reference point  $c_i$  is determined by a unique bit code  $B_i$ . A bit code  $B_i$  is a “path” from the root to the reference point  $c_i$  in the corresponding high-dimensional Quad-tree.

By adding one bit per dimension for each succeeding tree level we obtain<sup>1</sup>:

$$B_i = B_i^1 \otimes B_i^2 \otimes \dots \otimes B_i^t$$

Please note, that  $B_i^j$  is the bit code fragment that is added at level  $j$  of the AV-tree. Consequently, the term  $B_i^j[k]$  represents a single bit of  $B_i^j$  at dimension  $k$ . Figure 2.2 exemplifies this situation for  $d = 3$  dimensions and a tree height of  $t = 3$ . To compute

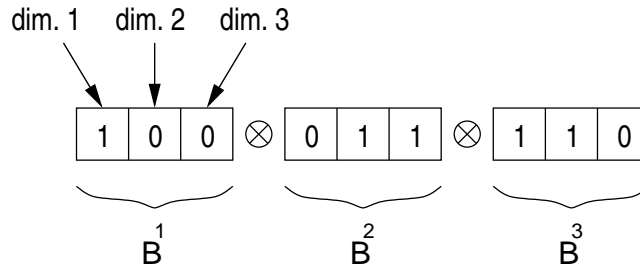


Figure 2.2: Structure of a Bit Code  $B = B^1 \otimes B^2 \otimes B^3$

the  $k$ -th coordinate of the corresponding reference point  $c_i$  we introduce the function `pos` which takes the bit code  $B_i$ , the path length  $t$  (tree level) and the dimension  $k$  as parameters:

$$c_i[k] = \text{pos}(B_i, t, k) = \frac{1}{2^{t+1}} + \sum_{j=1}^t \frac{B_i^j[k]}{2^j}$$

The “root” reference point  $c_{\text{root}}$  which has a path length of  $t = 0$  is, therefore, given by  $c_{\text{root}} = (0.5, \dots, 0.5)$ . Next, we discuss the utilization of a reference point  $c_i$  as approximation region for a data point  $p_i$ . We adapt the VA-file retrieval algorithm (cf. Figure 1.2) to our AV-tree approach. That is, we provide two functions  $\text{LB}(q, B_i)$  and  $\text{UB}(q, B_i)$  which compute the lower and upper bound distance of a query point  $q$  to the approximation region defined by the bit code  $B_i$ .

AV-tree approximation regions are formed by hyperballs around the reference points. That is, each reference point  $c_i$  inherently specifies a hyperball with the radius of  $r$ . Therefore, we obtain the following plain formulas:

$$\begin{aligned} \text{LB}(q, b_i) &= \begin{cases} 0 & \text{if } \|q - c_i\|_2 \leq r \\ \|q - c_i\|_2 - r & \text{otherwise} \end{cases} \\ \text{UB}(q, b_i) &= \|q - c_i\|_2 + r \end{aligned}$$

<sup>1</sup>Here,  $\otimes$  denotes the bit concatenation operator.

Please note, that the reference points  $c_i$  are computed out of their bit codes  $B_i$  by means of the `pos` function:

$$c_i[k] = \text{pos}(B_i, t, k)$$

The radius  $r$  is experimentally determined and globally fixed for all reference points (i.e. spherical approximation regions). In Section 4 we present a guideline to find a good value for  $r$ . Moreover, we present a formal framework to provide an appropriate value for  $r$  in Section 3.

## 2.2 Point Insertion

Subsequently, we sketch the point insertion algorithm. That is, we discuss the assignment of an AV-tree bit code to a data point. In this proposal, we approximate a data point  $p_i$  by a bit code  $B_i$  if  $p_i$  lies inside the spherical region around a suitable reference point  $c_i$ :

$$\|p_i - c_i\|_2 \leq r$$

Figure 2.3 provides the algorithm `insertPoint` which computes the approximation bit code of a data point  $p$ . Starting at the root level, this algorithm comprises two stages. Firstly,

```

function insertPoint( $B$ : bitarray,  $p$ : Point): bitarray
  // 1. compute current reference point  $c$ 
  int  $t = |B|/d$ ;
  Point  $c = (\text{pos}(B, t, 1), \dots, \text{pos}(B, t, d))$ ;
  // 2. check this reference point
  if  $\|c - p\|_2 \leq r$  then
    return  $B$ ;
  endif
  // 3. extend bit code  $B$ 
   $B = B \otimes \text{code}(c, p)$ ;
  return insertPoint( $B, p$ );
end

```

Figure 2.3: Algorithm `insertPoint`

(i)  $p$  is checked against the current reference point  $c$  (computed out of the current bit code  $B$ ). If this check fails, secondly, (ii) the bit code  $B$  is extended by the next subsequent bit code fragment. The function `code` computes the bit code fragment for the data point *relative* to the currently chosen reference point for every dimension:

$$\text{code}(c, p) = \bigotimes_{k=1}^d \begin{cases} 1 & \text{if } p[k] \geq c[k] \\ 0 & \text{otherwise} \end{cases}$$

To insert a point  $p_i$  we start the algorithm with the root reference point  $c_{\text{root}}$ , i.e. we invoke `insertPoint` with a blank bit code:

$$B_i = \text{insertPoint}(\{\}, p_i)$$

The algorithm returns the approximation bit code of  $p_i$ . Figure 2.4 depicts an example

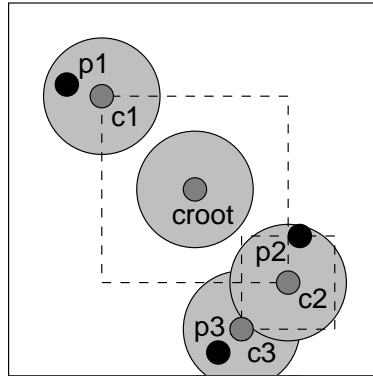


Figure 2.4: AV-tree in  $\mathbb{R}^2$

scenario. Three data points  $p_1$ ,  $p_2$ , and  $p_3$  (black dots,  $\bullet$ ) are approximated by the reference points  $c_1$ ,  $c_2$ , and  $c_3$  (grey dots,  $\bullet$ ) which reside at different levels of the AV-tree. The approximation regions are depicted as grey disks around the reference points. In this way, we obtain the following bit code approximations:

$$\begin{aligned} p_1 &\rightarrow B_1 (01) \\ p_2 &\rightarrow B_2 (10) \\ p_3 &\rightarrow B_3 (10\ 00) \end{aligned}$$

## 2.3 Maintenance

To take advantage of common bit code prefixes we store these bit codes in a Prefix-tree like structure. However, we do not map bit codes  $B_i$  to the Prefix-tree in a bit-wise manner. Rather, each edge reflects a bit code fragment  $B_i^j$  comprising  $d$  bits. In this way, every node corresponds to a unique reference point and every edge marks the transition from one reference point to a subsequent reference point in the AV-tree. Figure 2.5 depicts the utilization of shared prefixes in our example by means of a Prefix-tree. However, this representation has to be mapped to a flat file representation for secondary storage.

Unfortunately, these Prefix-trees are very unlikely to be complete. That is, each reference point (Prefix-tree node) may have up to  $2^d$  subsequent reference points, marked by unique bit code fragments. This exponential growth of potential subsequent reference

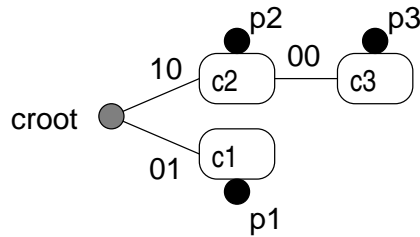


Figure 2.5: AV-tree Mapping to Prefix-tree

points out numbers the number of available data points even for fairly low dimensionalities  $d$ .

Therefore, additional *command codes* are required to map an Prefix-tree to a flat file. In Table 2.1 we introduce four “command codes” which are used in the mapping algorithm. The idea of this mapping is to precede each fixed length bit code fragment

Command Code	Abbreviation	Bit Code
<i>DownCode</i>	DC	00
<i>Up</i>	UP	01
<i>Final</i>	FI	10
<i>TreeUp</i>	TU	11

Table 2.1: Mapping Command Codes

$B_i^j$  with the *DownCode* command. On the one hand, this command marks a succeeding bit code fragment comprising  $d$  bits. On the other hand, it inherently traverses to the subsequent reference point. In our example (cf. Figure 2.5) the sequence DC 10 describes the transition from  $c_{\text{root}}$  to  $c_2$  with the bit code fragment 10.

The command *Up* has a reverse semantics. Thus, it marks the backward transition from a child node to its superordinated node. In contrast, the command *TreeUp* returns to the root node  $c_{\text{root}}$ . The command *Final* marks a data point. Figure 2.6 depicts a simple recursive mapping algorithm. To start the mapping, again, we invoke `mapTree` with  $c_{\text{root}}$ . The resulting flat file `<MAPPING>` has the following syntax, given by an

```

procedure mapTree(c: Point)
  // 1. Final for each assigned data point
  for each data point p assigned to c do
    write FI
  od
  // 2. DownCode for each child reference point
  for each child reference point ci do
    Bi = code(c, ci)
    write DC Bi
    mapTree(ci)
  od
  // 3. Up to return to the superordinated level
  write UP
end

```

Figure 2.6: Algorithm mapTree

(extended) BNF-like notation:

$$\begin{aligned}
 \langle \text{MAPPING} \rangle &::= \langle \text{COMMAND} \rangle, \square \\
 \langle \text{COMMAND} \rangle &::= \langle \text{COMMAND} \rangle, \langle \text{COMMAND} \rangle \\
 \langle \text{COMMAND} \rangle &::= \langle \text{UP} \rangle | \langle \text{TU} \rangle | \langle \text{FI} \rangle | \langle \text{DC} \rangle, \langle \text{CODE} \rangle \\
 \langle \text{CODE} \rangle &::= \{ "0" | "1" \}^d \\
 \langle \text{DC} \rangle &::= "00" \\
 \langle \text{UP} \rangle &::= "01" \\
 \langle \text{FI} \rangle &::= "10" \\
 \langle \text{TU} \rangle &::= "11"
 \end{aligned}$$

Please note, firstly,  $\square$  is the *end-of-file* mark which does not actually appear in the flat file representation. Secondly,  $\{ "0" | "1" \}^d$  describes an arbitrary bit code fragment  $B_i^d$  that comprises exactly  $d$  bits.

To obtain a further reduction of the storage size we can put the following simplification rules into place:

$$\begin{aligned}
 \langle \text{UP} \rangle, \square &\mapsto \square \\
 \langle \text{DC} \rangle, \langle \text{CODE} \rangle, \langle \text{FI} \rangle, \langle \text{UP} \rangle &\mapsto \langle \text{DC} \rangle, \langle \text{CODE} \rangle, \langle \text{UP} \rangle \\
 \{ \langle \text{UP} \rangle \}^t &\mapsto \langle \text{TU} \rangle
 \end{aligned}$$

These rules (i) remove unnecessary *Up*'s from the end, (ii) remove (inherently given) *Final*'s in front of succeeding *Up*'s, and (iii) combine  $t$  (tree height) *Up*'s to one single *TreeUp*.

---

---

In our example scenario (cf. Figures 2.4 and 2.5), we obtain the following AV-tree to Prefix-tree mapping:

(i) Mapping without simplification (24 bit):

**00011001 00101000 00100101**  
1 byte 1 byte 1 byte

(ii) Mapping with simplification (16 bit):

**00011100 10100000**  
1 byte 1 byte

Please note, that the command codes are depicted in **bold face** for clarity only.



# Chapter 3

## Formal Cost Comparison

In this chapter, we provide an analytically funded retrieval cost comparison between the AV-tree and the VA-file approach. The algorithm `retrieveNN` (cf. Figure 1.2) fetches all data point approximations  $B_i$  and processes them into a list of candidate bit codes (candidates) in its first stage. In the second stage, the algorithm traverses the list of corresponding data points to determine the actual nearest neighbor point. Thus, the overall retrieval costs are, made up by two terms:

$$\text{cost} = \text{compression} + \text{selectivity}$$

The value of `compression` reflects the compression ratio in both approaches. That is, it specifies the amount of approximation data to be fetched from the database during the first stage of the retrieval algorithm:

$$\text{compression} = \begin{cases} N \cdot d \cdot b & \text{(VA-file)} \\ N \cdot d \cdot t & \text{(AV-tree)} \end{cases}$$

In this comparison we abstain from the Prefix-tree style mapping of the AV-tree (cf. Section 2.3). In contrast, we store the approximation bit codes  $B_i$  sequentially. Due to the fact, that the Prefix-tree style mapping needs less storage space than the sequential list of bit codes this restriction can me made. For an average tree depth (path length) of  $t$ , we obtain  $d \cdot t$  bits of approximation data for each data point.

The value of `selectivity` determines the amount of data points which are fetched in the second stage of the retrieval algorithm. Its value is upper bounded as follows<sup>1</sup>:

$$\text{selectivity} \leq |\text{candidates}| \cdot d \cdot \text{sizeof}(\text{float})$$

The retrieval of a data point  $p_i$  by an approximation bit code  $B_i$  requires a constant time in both the AV-tree and the VA-file approach. We relate this time to the amount of data  $d \cdot \text{sizeof}(\text{float})$  which is occupied by a single data point. This data point retrieval can, for

---

<sup>1</sup>Here, `float` denotes a 4-byte single precision floating point number (IEEE 754).

example, be implemented by a dynamic hashing technique using the approximation bit codes as hash keys. Alternatively, a constant time can also be achieved by an ordering of the data points to the order of the approximation bit codes.

For a cost comparison, we fix the selectivity in both approaches and investigate the corresponding compression results. Precisely, we equate the selectivity of the AV-tree and the VA-file approach by means of their distinct LB (i.e. *LowerBound*) computations.

We consider an idealized retrieval algorithm where the minimal value of  $\min$  is known in advance. Furthermore, all subsequent considerations are based on artificially created data points with coordinate values uniformly distributed in  $[0, 1]$ . We show, that the compression size of the VA-file exceeds the size of the AV-tree approximation bit codes. That is, we prove that

$$\begin{aligned} & \text{compression}_{\text{VA-file}} \geq \text{compression}_{\text{AV-tree}} \\ \Leftrightarrow & N \cdot d \cdot b \geq N \cdot d \cdot t \\ \Leftrightarrow & b \geq t \end{aligned}$$

holds.

The idea of our following considerations is a formal estimation of  $L_2$ -norm based distances (LB) between query points and approximation regions in the AV-tree and the VA-file. We start our comparison by stating the probability (interval distribution) for  $L_2$ -distances between a query point and the approximation region in one dimension.

### 3.1 LB-Values of the AV-tree

AV-Trees are based on hyperball-shaped approximation regions with a radius of  $r$ . With respect to a query point ( $q$ ) location, we have to consider two cases:

- (i)  $q$  lies inside the approximation hyperball.
- (ii)  $q$  is located outside the hyperball.

The computation of  $\text{LB}_{\text{AV}}(q, B)$  reflects both cases:

$$\text{LB}_{\text{AV}}(q, B) = \begin{cases} 0 & \text{if } \|q - c\|_2 \leq r \\ \|q - c\|_2 - r & \text{otherwise} \end{cases}$$

Please note, that  $c$  denotes a reference point (i.e. the center point of an approximation hyperball) specified by the approximation bit code  $B$ . Due to space restrictions we abstain from a complete derivation of  $L_2$ -distances between uniformly distributed points [Sch01]. Instead, we refer to [WB00] where the expecting value of *squared*  $L_2$ -distances between uniformly distributed points is already quantified with:

$$\text{E}(X)_{\|V-V\|_2^2} = \frac{d}{6}$$

Furthermore, we follow the argumentation from [AHK01] by using Slutsky's Theorem<sup>2</sup> and derive an expecting value for  $L_2$ -distances between uniformly distributed points of:

$$E(X)_{\|V-V\|_2} \rightarrow_p \sqrt{\frac{d}{6}}$$

The probability of a query point being located inside (i) the hyperball corresponds to the hyperball volume:

$$P_{AV}(\|q - c\|_2 \leq r) = r^d \frac{\pi^{d/2}}{\Gamma(1 + d/2)}$$

The remaining volume covers the probability for an outside (ii) query point:

$$P_{AV}(\|q - c\|_2 > r) = 1 - P_{AV}(\|q - c\|_2 \leq r)$$

The overall expecting value for  $LB_{AV}$  computes, therefore, as follows:

$$E(X)_{LB_{AV}} = P_{AV}(\|q - c\|_2 > r)(E(X)_{\|V-V\|_2} - r)$$

Due to the fact that the approximation hyperball volume becomes very small in rising dimensionality, we simplify this formula as follows:

$$E(X)_{LB_{AV}} \approx \sqrt{\frac{d}{6}} - r$$

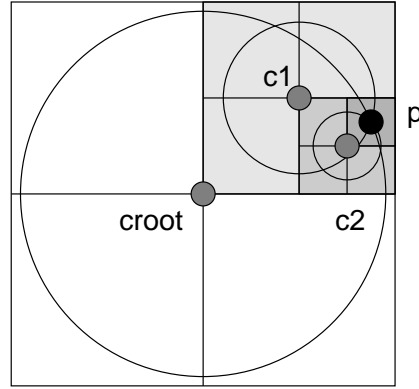
The radius  $r$  depends on the average tree depth  $t$  of the AV-tree. Figure 3.1 depicts three reference points ( $c_{\text{root}}$ ,  $c_1$ , and  $c_2$ ) in an AV-tree. These reference points reside at different levels of the AV-tree (0, 1, and 2, respectively). The circles around the reference points illustrate their distances to a data point  $p$ . These distances correspond to the radius  $r$  of an approximation hyperball around  $c_i$ . In this consideration, we aim to determine the average radius at a certain tree depth. That is, we compute the expecting value for the distance of a reference point  $c_i$  at a certain AV-tree level  $t$  to an arbitrarily located data point  $p$ . Since the reference points are located in the corner of a hyper-rectangle (Figure 3.1, shaded in grey), this problem corresponds to the distance of a uniformly distributed data point to the origin point of this hyper-rectangle.

First of all, the edge lengths  $l$  of these hyper-rectangles must be determined. The function `len` computes the edge length out of the AV-tree level  $t$ :

$$\text{len}(t) = 1/2^{t+1}$$

That is, we know that at tree level  $t$  a data point  $p$  must be located inside a hyper-rectangle with an edge length of  $l = \text{len}(t)$ . The next possible approximation point  $c$  lies in one corner (w.l.o.g. the origin point) of this hyper-rectangle. Please refer to Figure 3.1 where this situation is depicted.

<sup>2</sup>Slutsky's Theorem: Let  $Y_1 \dots Y_d$  be a sequence of random vectors and  $h(\bullet)$  be a continuous function. If  $Y_d \rightarrow_p c$  then  $h(Y_d) \rightarrow_p h(c)$  holds.

Figure 3.1: Radii at Different AV-tree Levels  $t$ 

The interval distribution for the distance between a uniformly distributed (in  $[0 \dots l]$ ) random value (data point coordinate) and 0 (origin coordinate) is:

$$P_{|V-0|}(X < x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x \geq l \\ x/l & \text{otherwise} \end{cases}$$

For the computation of  $L_2$ -distances, the coordinate value differences need to be squared. By replacing  $x$  with  $\sqrt{x}$  we obtain an interval distribution for the squared value differences of:

$$P_{|V-0|^2}(X < x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x \geq l^2 \\ \sqrt{x}/l & \text{otherwise} \end{cases}$$

The density function  $\phi_{|V-0|^2}(x)$  is subsequently computed as follows:

$$\phi_{|V-0|^2}(x) = \frac{dP_{|V-0|^2}(X < x)}{dx}$$

For the range of  $0 \leq x \leq l^2$  we obtain:

$$\phi_{|V-0|^2}(x) = \frac{1}{2l\sqrt{x}}$$

Now, we can compute the expecting value for squared value differences  $E(X)_{|V-0|^2}$  as follows:

$$E(X)_{|V-0|^2} = \int_{-\infty}^{\infty} x \cdot \phi_{|V-0|^2}(x) dx = \int_0^{l^2} \frac{\sqrt{x}}{2l} dx = \frac{l^2}{3}$$

Next, we compute the expecting value for squared  $L_2$ -distances by adding the squared value differences for each dimension:

$$\|p - 0\|_2^2 = (p[1] - 0)^2 + \dots + (p[d] - 0)^2$$

In this context,  $(p[i] - 0)^2$  can be considered as a random variable with an expecting value of  $E(X)_{|V=0|^2}$ . Therefore

$$\begin{aligned} E(X)_{\|V=0\|_2^2} &= E_{|V=0|^2} + \dots + E_{|V=0|^2} \\ &= \frac{d \cdot l^2}{3} \end{aligned}$$

holds. By following the argumentation in [Sch01] we can, again, determine the expecting value for  $L_2$ -distances:

$$E(X)_{\|V=0\|_2} \rightarrow_p l \sqrt{\frac{d}{3}}$$

Therefore, we compute the average radius  $r$  at a certain tree level  $t$  as follows:

$$r = l \sqrt{\frac{d}{3}} = \frac{\sqrt{d/3}}{2^{t+1}}$$

Thus, we can derive the following formula of the expecting value for  $LB_{AV}$  computations:

$$E(X)_{AV} \rightarrow_p \sqrt{\frac{d}{6}} - \frac{\sqrt{d/3}}{2^{t+1}}$$

## 3.2 LB-Values of the VA-file

Now, we compute the expecting value for the lower bound distance in VA-files. In contrast to AV-trees, VA-files approximate data point locations by means of hyper-rectangles. Again, we start our considerations with a statistical analysis of single dimension distances.

In one single dimension, the location of a query point  $q$  is given by the value of  $x_1$  whereas the left boundary of the approximation box is given by  $x_2$ , the edge length of the approximation box is given by:

$$w = 2^{-b}$$

Following our assumption,  $x_1$  is uniformly distributed in  $[0, 1]$ . The left boundary  $x_2$  is uniformly distributed in  $[0, 1 - w]$ . The distance  $x$  between  $x_1$  and the approximation box is determined as follows:

$$x = \begin{cases} x_2 - x_1 & \text{if } x_2 \geq x_1 \\ x_1 - x_2 - w & \text{if } x_1 \geq x_2 + w \\ 0 & \text{otherwise} \end{cases}$$

Please note, that the third case reflects the situation when  $x_1$  lies inside the approximation box. The interval distribution ( $0 \leq x \leq 1 - w$ ) can be determined as follows:

$$P_{|V=I|}(X < x) = 1 - \frac{(1 - w - x)^2}{1 - w}$$

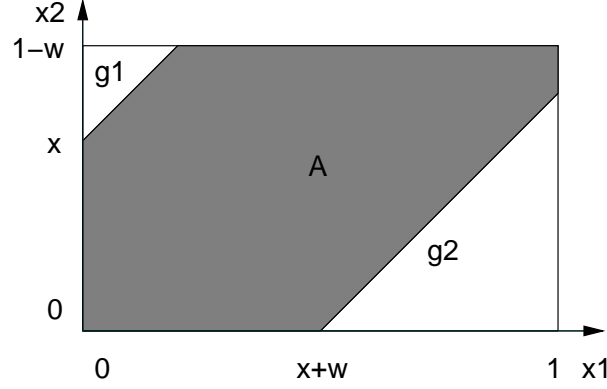


Figure 3.2: Probability as Volume

The probability is 0 and 1 for  $x < 0$  and  $x > 1 - w$ , respectively. For an illustration, please refer to Figure 3.2.

**Proof.** First of all, the overall probability is expressed by the rectangle  $[0, 1] \times [0, 1 - w]$ . For  $x = x_2 - x_1$  (first case,  $x_2 \geq x_1$ ), we obtain the line  $g_1 : x_2 = x_1 + x$ .

In the second case ( $x_1 \geq x_2 + w$ ), we can derive  $g_2 : x_2 = x_1 - w - x$ . Both lines are depicted in Figure 3.2. The area  $A$  is then computed as follows:

$$A = (1 - w) - (1 - w - x)^2$$

To obtain the probability, we have to relate  $A$  to the overall area:

$$P_{|V-I|}(X < x) = \frac{A}{1 - w} = 1 - \frac{(1 - w - x)^2}{1 - w}$$

□

Again, we have to consider the squared  $L_2$ -distances. We replace  $x$  with  $\sqrt{x}$  and obtain the following interval distribution for  $0 \leq x \leq (1 - w)^2$ :

$$P_{|V-I|^2} = 1 - \frac{(1 - w - \sqrt{x})^2}{1 - w}$$

The probability is 0 and 1 for  $x < 0$  and  $x > (1 - w)^2$ , respectively. The density function is derived as follows:

$$\phi_{|V-I|^2} = \frac{dP_{|V-I|^2}}{dx}$$

For  $0 \leq x \leq (1 - w)^2$  we obtain:

$$\phi_{|V-I|^2} = \frac{1 - \sqrt{x} - w}{\sqrt{x}(1 - w)}$$

Finally, we can compute the expecting value by solving the following integral:

$$\begin{aligned} E(X)_{|V-I|^2} &= \int_{-\infty}^{\infty} x \cdot \sqrt{x}(1 - \sqrt{x} - w) dx \\ &= \frac{(1-w)^3}{6} \end{aligned}$$

Following our previous argumentation we obtain these formulas for the (squared)  $L_2$  distances:

$$\begin{aligned} E(X)_{\|V-I\|_2^2} &= \frac{d}{6}(1-w)^3 \\ E(X)_{\|V-I\|_2} &\rightarrow_p \sqrt{\frac{d}{6}(1-w)^3} \end{aligned}$$

That is, we obtain an expecting value for lower bound (LB) computations in the VA-file of:

$$E(X)_{\text{LB}_{\text{VA}}} \rightarrow_p \sqrt{\frac{d}{6}(1-2^{-b})^3}$$

Finally, we can relate the expecting values for lower bound (LB) computations of the AV-tree and the VA-file as follows:

$$\begin{aligned} E(X)_{\text{LB}_{\text{VA}}} &= E(X)_{\text{LB}_{\text{AV}}} \\ \sqrt{\frac{d}{6}\left(1 - \frac{1}{2^b}\right)^3} &= \sqrt{\frac{d}{6} - \frac{\sqrt{d/3}}{2^{t+1}}} \end{aligned}$$

By relating the number  $b$  of bits per dimension (VA-file) to the average tree level  $t$  (AV-tree), we obtain:

$$t = -\log_2 \left[ 1 - \sqrt{\left(1 - \frac{1}{2^b}\right)^3} \right] - \frac{1}{2}$$

In Figure 3.3 the resulting values of  $t$  are depicted for  $b = 1, \dots, 10$ . Approximately,

$$b \approx t + 1$$

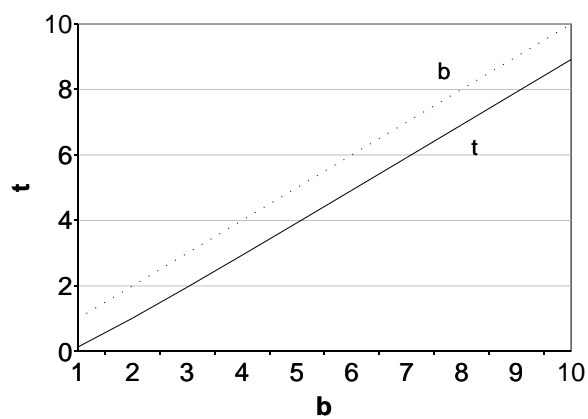
holds. For example, for a dimensionality  $d = 50$  and an average tree level of  $t = 2$  we obtain a number of bits per dimension of  $b = 3$ . That is, the approximation bit code  $B_i$  for a data point  $p_i$  takes:

$$|B_i| = \begin{cases} d \cdot t = 100 \text{ bit} & \text{AV-tree} \\ d \cdot b = 150 \text{ bit} & \text{VA-file} \end{cases}$$

Thus, we can state lower compression costs for the AV-tree approach while maintaining the same selectivity:

$$\text{compression}_{\text{VA-file}} > \text{compression}_{\text{AV-tree}}$$

Vice versa, if we choose an equal compression size in both approaches, we obtain a better selectivity in the AV-tree approach.

Figure 3.3: Plotted Values of  $t$

# Chapter 4

## Experimental Results

In this chapter, we present the results from our experimental studies. These comprise optimal parameterization and performance studies. In particular, we experimentally compare the Active Vertice approach with a VA-file implementation on various data sets and dimensionalities. We base our experiments on both artificial and real-world feature data. Precisely, that is:

- (i) **UNIFORM**: This artificial data set consists of data points whose coordinates are uniformly distributed in  $[0, 1]$ . ( $d = 20, \dots, 360$ )
- (ii) **COREL**: These data sets originate from feature data of the Corel image collection. This data is freely available from the UCI KDD archive<sup>1</sup> and comprises feature vectors based on color histogram, color histogram layout, color moments, and co-occurrence texture extraction algorithms. ( $d = 9, 16, 32$ )
- (iii) **ADAC**: This data set comprises color histograms of car photos available on the ADAC<sup>2</sup> 2001 automotive image CD-ROM. The images were transformed to the HSV color model. Subsequently, color histograms were extracted with varying dimensionalities. ( $d = 20, \dots, 360$ )

All data was scaled to the interval  $[0, 1]$  in each dimension. Thus, we obtain data points  $p_i$  with  $p_i \in [0, 1]^d$ . The nearest neighbor queries were performed with data points  $p_i$  and query points  $q_i$  which originate from a joint data set (**UNIFORM**, **COREL**, or **ADAC**), i.e. are subject to the same data distribution.

For the performance studies, we abstain from time measurements on particular hardware and database configurations. In contrast, we determine the average amount of data which has to be fetched from a database during the retrieval process. That is, the length of bit code compressions used in the AV-tree (cf. Section 2.3) and the VA-file is quantified as the size of the compressed index (in bit). Data point coordinates are considered as single precision float numbers which require 32 bit (4 byte) of storage space.

---

<sup>1</sup>[kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html](http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html)

<sup>2</sup>Allgemeiner Deutscher Automobil Club (engl. *General German Automobile Club*)

In Section 2.3 we introduced a mapping of the AV-tree to a flat file representation. This

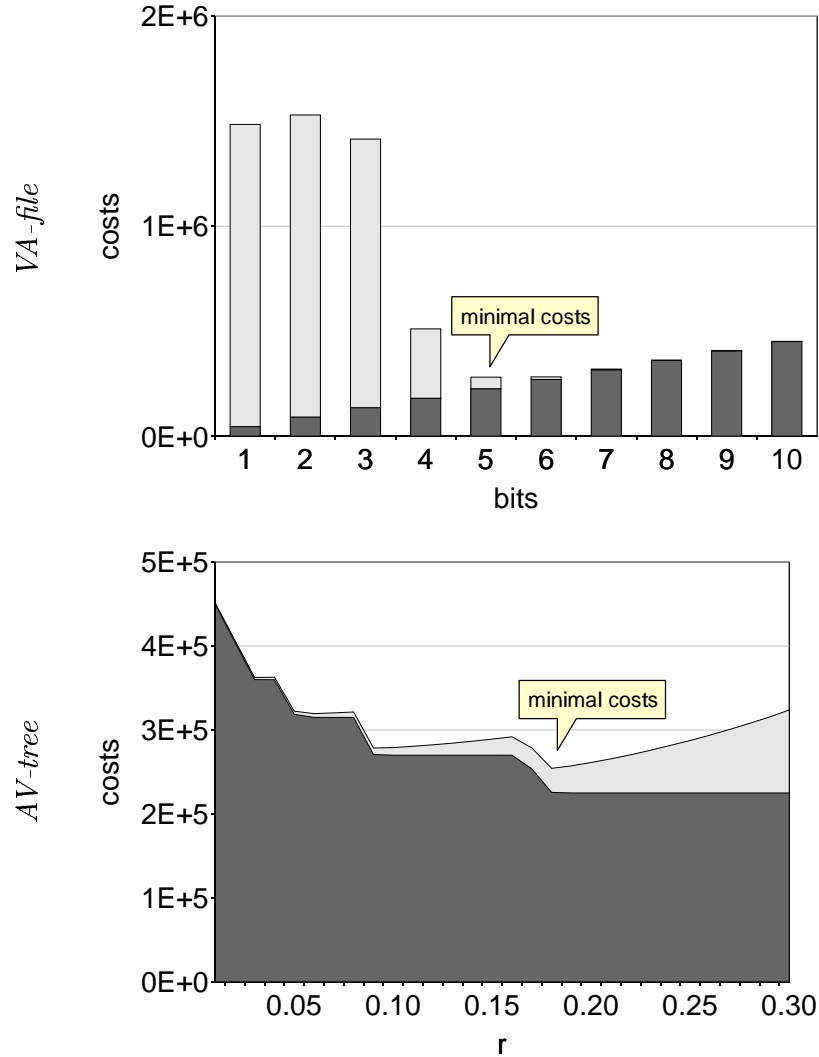


Figure 4.1: Optimal Parameterization ( $d = 360$ )

flat file can sequentially be read from disk pages on the secondary storage. Therefore, we can directly relate the amount of data to a number of page accesses. In this way, we achieve independency from a certain platform and do not have to consider side effects from other software components (e.g. operating system swap cycles).

Throughout this section we use some notational conventions. Unless explicitly stated differently, the distinct experimental results appear in the diagrams with the shapes from Table 4.1. Furthermore, we use the sampling method proposed in [LS01] for our experiments. In this way, we ran 1000 queries with distinct query points for each measurement reading and computed an average value. The retrieval testbed was implemented in Java interacting with an Oracle database on top of a Linux workstation.

	UNIFORM	COREL, ADAC
AV-tree	◆	⌵
VA-file	■	⌵

Table 4.1: Notational Conventions

## 4.1 Parameterization

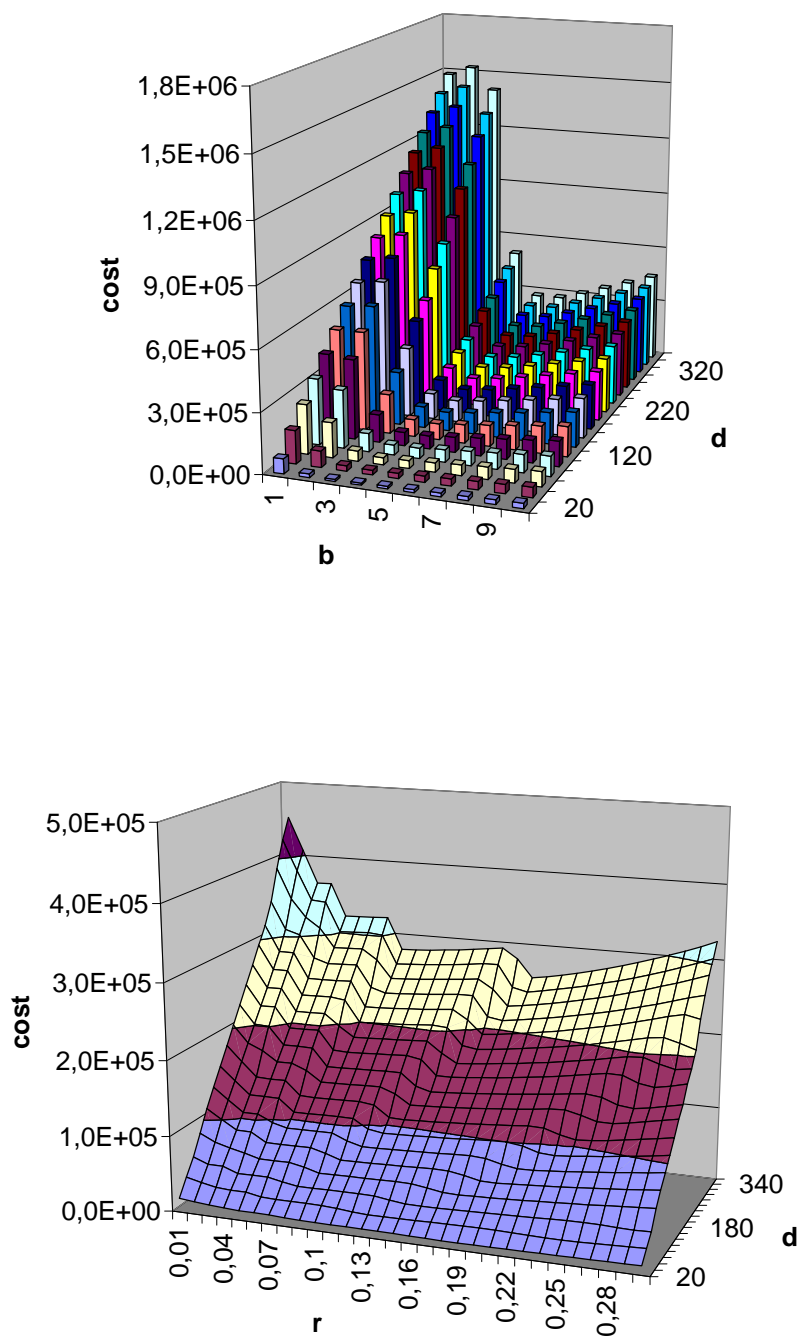
Both, the AV-tree and the VA-file approach require some parameterization in order to yield an optimal retrieval performance. That is, we need to provide (i) the radius  $r$  of the approximation hyperballs (AV-tree) and (ii) the number  $b$  of bits per dimension (VA-file) for the point insertion and the nearest neighbor retrieval algorithms. These parameters reflect the trade-off between a good compression (i.e. small index size) and a high selectivity (i.e. small candidate list). Both values (cf. Figure 1.2) make up the overall retrieval cost. Figure 4.1 depicts the composite retrieval cost using an AV-tree and a VA-file with the **UNIFORM** data set for a dimensionality of  $d = 360$ . The parameters  $r$  and  $b$  were scaled between  $0.01, \dots, 0.3$  and  $1, \dots, 10$ , respectively. Please note that the dark grey areas depict the index size, thus reflecting the term **compression** (cf. Section 3). The bright grey areas reflect **selectivity**, i.e. the amount of data occupied by the number of remaining candidates. The parameters  $(r, b)$  which yield the best (i.e. minimal) overall retrieval cost are marked by yellow arrows. The radius parameter has an indirect impact on the average AV-tree level  $t$ . That is, within certain ranges (e.g.  $r \in [0.10, 0.15]$ )  $t$  remains stable which is reflected by a constant **compression** term. In turn, we obtain a typical pattern for the compression size which abruptly increases at certain threshold values of  $r$ . The selectivity deteriorates constantly with a rising radius.

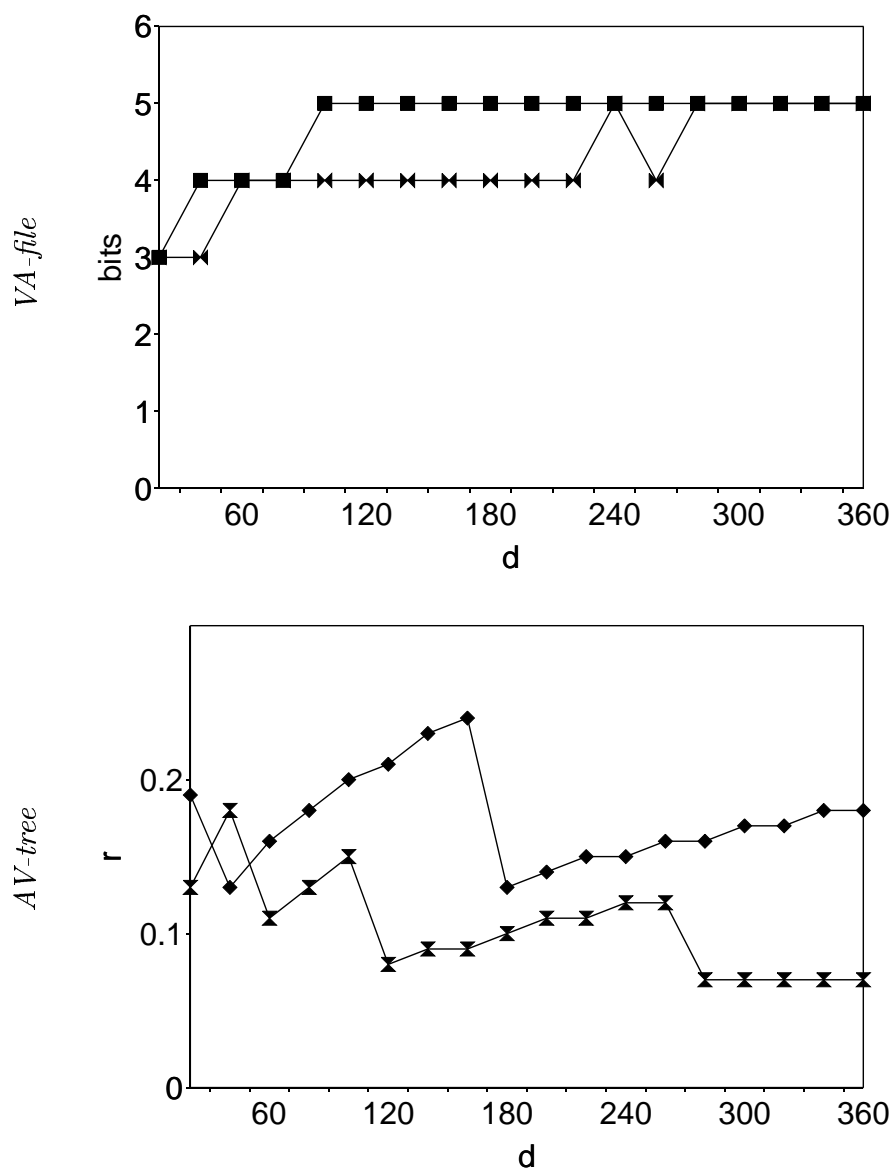
Figure 4.2 (Page 25) depicts the influence of the external parameters on the overall retrieval costs for a large range of dimensionalities ( $d = 20, \dots, 360$ ). In Figure 4.3 (Page 26) the resulting *optimal* parameters  $(r, b)$  are depicted for both **UNIFORM** and real data sets. Again, we observe an oscillating optimal value of  $r$ . The particular values of  $d$  where  $r$  drops (e.g. **UNIFORM**:  $d = 40, d = 180$ ) mark the transition to a higher AV-tree level. Generally, the radius  $r$  shrinks with an increasing average AV-tree level. This is due to the fact, that the average distance from a reference points  $c_i$  to its approximated data point  $p_i$  reduces (cf. Figure 3.1). Within one AV-tree level (e.g. **UNIFORM**:  $d = 40, \dots, 160$ )  $r$  increases which is confirmed by the formal results from Section 3.1. Furthermore, the optimal value of  $r$  tends to be smaller for real data sets which indicates a higher data density in comparison to a **UNIFORM** data set.

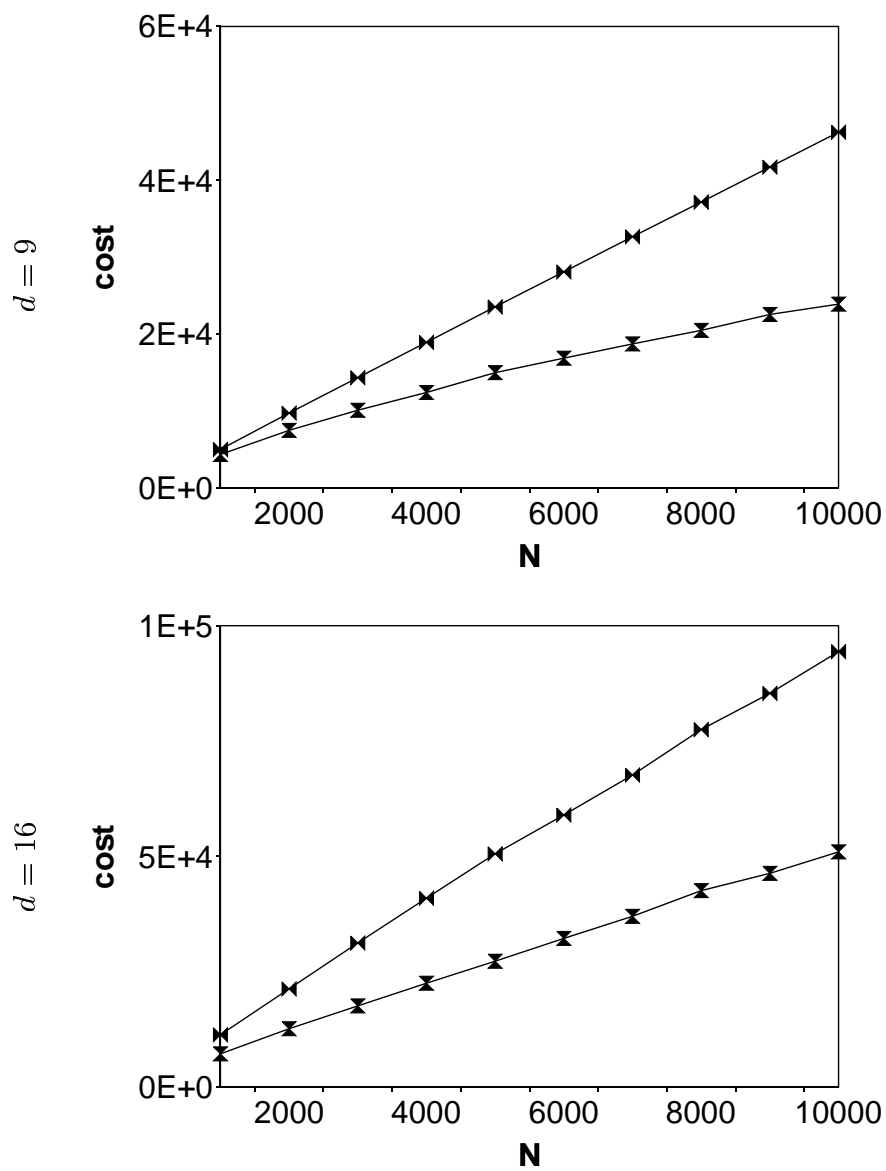
## 4.2 Performance Comparison

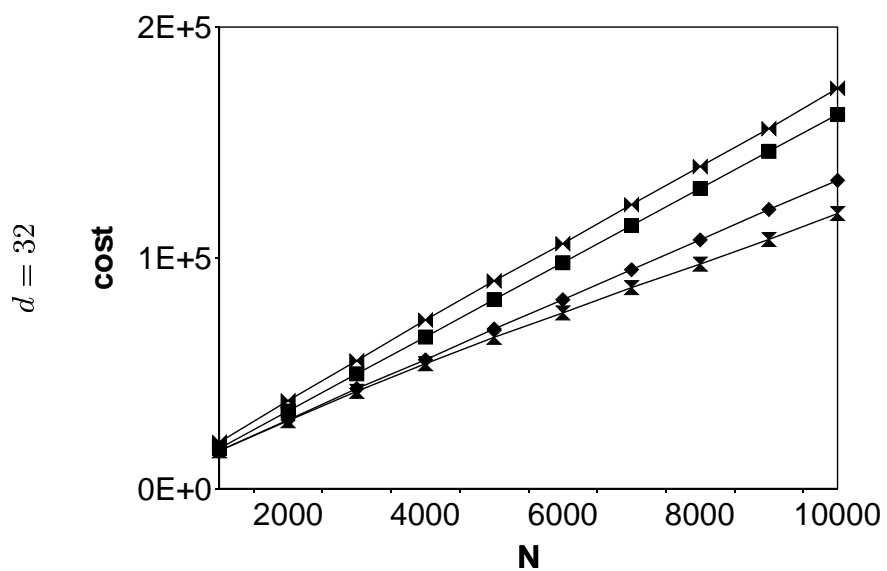
In this section, we directly relate the AV-tree proposal to the VA-file approach in terms of their retrieval performance. We inspect both, real and uniform data sets for various

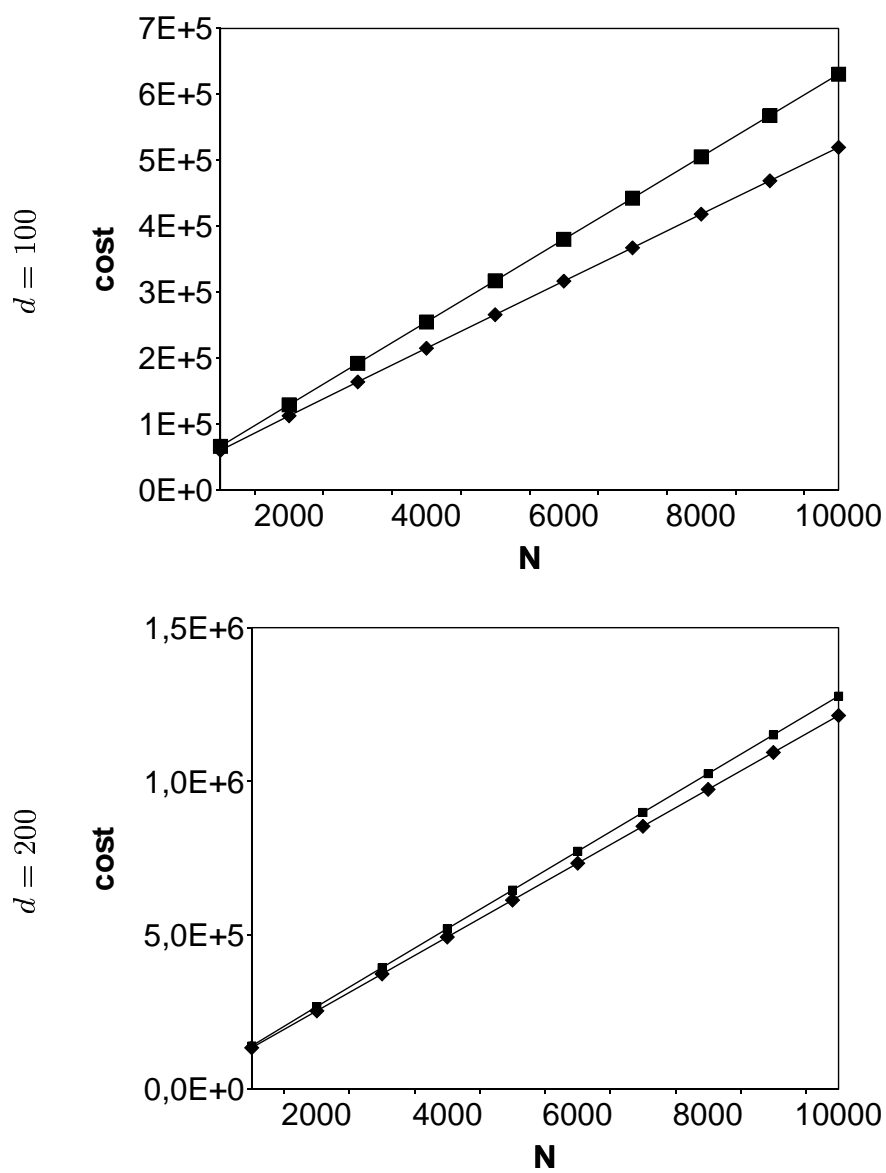
dimensionalities. In Figures 4.4 (Page 27) and 4.5 (Page 28) we performed nearest neighbor retrievals on the **COREL** feature data sets using an AV-tree and a VA-file implementation. We ran this performance evaluation with varying numbers of data points, i.e.  $N = (1, \dots, 10) \cdot 10^3$ . Within this low and mid dimensionalities ( $d = 9, 16, 32$ ) the AV-tree yields a remarkably superior retrieval performance compared to the VA-file performance. In Figure 4.5 (Page 28) we accompanied the real data experiments with the performance measurements on **UNIFORM** data ( $d = 32$ ). Although the performance difference is smaller than for real data, the AV-tree still clearly outperforms the VA-file. Figure 4.6 (Page 29) depicts the retrieval costs for very high dimensionalities ( $d = 100, 200$ ) based on the **UNIFORM** data set. Again, we observe a superiority of the AV-tree approach in terms of its retrieval performance. Finally, we recorded the performance gains (i.e. retrieval cost savings) by using the AV-tree instead of the VA-file in Figure 4.7 (Page 30). That is, we applied the AV-tree and VA-file approaches on a small sample ( $N = 1000$ ) of the **UNIFORM** and **ADAC** data sets. Furthermore, we measured the performance differences in a wide dimensionality spectrum of  $d = 20, \dots, 360$ . Figure 4.7 depicts increasing cost savings in rising dimensionalities (► uniform data, ◀ real data). The local extremes, again, reflect the transition to a higher AV-tree level (cf. Figure 4.3 on Page 26).

Figure 4.2: Parameterization ( $d = 20, \dots, 360$ )

Figure 4.3: Parameter Profile ( $d = 20, \dots, 360$ )

Figure 4.4: Performance Comparison ( $d = 9, 16$ )

Figure 4.5: Performance Comparison ( $d = 32$ )

Figure 4.6: Performance Comparison ( $d = 100, 200$ )

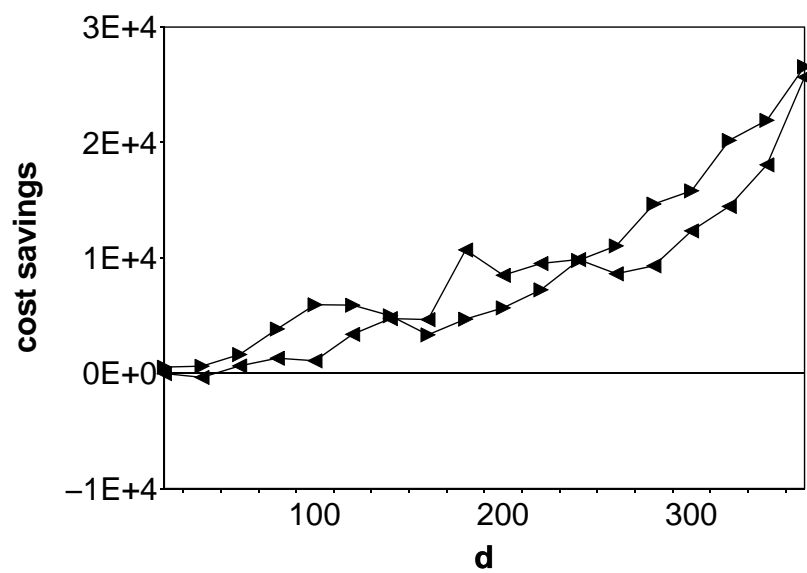


Figure 4.7: Performance Savings ( $d = 20, \dots, 360$ )

# Chapter 5

## Conclusions

We introduced the Active Vertice approach as an approximation technique for high-dimensional nearest neighbor retrieval. Unlike other recent approaches it is not based on the VA-file but, reuses basic fundamentals from the Quad-tree to achieve a relative approximation. We presented the structure of the Active Vertice approximation and discussed insertion and maintenance considerations.

By formally relating our approach to the VA-file in terms of their selectivity, we analytically proved the superiority of the Active Vertice approach by means of statistical considerations. Various experiments, both for an appropriate parameterization and for an experimental comparison with the VA-file confirm the previous analytical results. That is, independently from data distribution and dimensionality the Active Vertice approach turns out to be a superior approach for high-dimensional nearest neighbor retrieval.

Ongoing work will (i) be centered on a formal refinement of the parameterization framework, and (ii) an investigation of alternative geometries for an appropriate allocation of the reference point to improve the AV-tree fanout.



---

---

## Bibliography

- [AHK01] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Database Theory — ICDT'01, Proc. of the 8th Int. Conf., London, January 2001*, Incs, Vol. 1973, pages 420–434. Springer-Verlag, Berlin, 2001.
- [BBJ<sup>+</sup>00] Stefan Berchtold, Christian Böhm, H.V. Jagadish, Hans-Peter Kriegel, and Jörg Sander. Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering, ICDE'00, February 28– March 3, 2000, San Diego CA*, pages 577–588, IEEE Computer Society Press, 2000.
- [BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is Nearest Neighbor Meaningful. In C. Beeri and P. Buneman, editors, *Database Theory — ICDT'99, Proc. of the 7th Int. Conf., Jerusalem, Israel, January 1999*, Incs, Vol. 1540, pages 217–235. Springer-Verlag, Berlin, 1999.
- [BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22nd Int. Conf. on Very Large Data Bases, VLDB'96, Bombay, India, September 3–6, 1996*, pages 28–39. Morgan Kaufmann Publishers, San Francisco, CA, 1996.
- [CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jausfeld, editors, *VLDB 1997, Proceedings of 23th International Conference on Very Large Data Bases, August 25-29, 2000, Athens, Greece*, pages 426–435, Morgan Kaufmann, 1997.
- [Fal96] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1996.
- [FTJF01] R. F. S. Filho, A. Traina, C. Traina Jr., and C. Faloutsos. Similarity Search without Tears: the OMNI-Family of All-Purpose Access Methods. In *Proc. of*

- the 17th IEEE Int. Conf. on Data Engineering, ICDE'01, Heidelberg, Germany, April 2001*, pages 623–630. IEEE Computer Society Press, Los Alamitos, CA, 2001.
- [GR00] Jonathan Goldstein and Raghu Ramakrishnan. Contrast Plots and P-Sphere Trees: Space vs. Time in Nearest Neighbor Searches. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 429–440, Morgan Kaufmann, 2000.
- [Gut84] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In B. Yormark, editor, *Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of Data, Boston, NJ*, ACM SIGMOD Record, Vol. 14, No. 2, pages 47–57. ACM Press, New York, 1984.
- [JTSF00] Caetano Traina Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, Lecture Notes in Computer Science, Vol. 1777, pages 51–65, Springer, 2000.
- [KS97] Norio Katayama and Shin'ichi Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Joan Peckham, editor, *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data, Tucson, Arizona*, ACM SIGMOD Record, Vol. 26(2), pages 369–380, ACM Press, May 1997.
- [LHC01] C.-C. Liu, A. J. L. Hsu, and A. L. P. Chen. Efficient Near Neighbor Searching Using Multi-Indexes for Content-Based Multimedia Data Retrieval. *Multimedia Tools and Applications*, 13(3):235–254, 2001.
- [LS01] C. A. Lang and A. K. Singh. Modeling High-Dimensional Index Structures using Sampling. In Walid G. Aref, editor, *Proc. of the 2001 ACM SIGMOD Int. Conf. on Management of Data, Santa Barbara, CA, USA*, ACM SIGMOD Record, Vol. 30(2), ACM Press, May 2001.
- [OTYB00] B.C. Ooi, K.-L. Tan, C. Yu, and S. Bressan. Indexing the Edges – A simple and yet efficient approach to high-dimensional indexing. In *Proc. of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Dallas, Texas, USA*, ACM Press, May 2000.

- 
- 
- [Sch01] I. Schmitt. Nearest Neighbor Search in High Dimensional Space by Using Convex Hulls. Preprint No. 6, Fakultät für Informatik, Universität Magdeburg, 2001.
- [SYUK00] Y. Sakurai, M. Yoshikawa, Shunsuke Uemura, and H. Kojima. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 516–526, Morgan Kaufmann, 2000.
- [WB00] Roger Weber and Klemens Böhm. Trading quality for time with nearest neighbor search. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, Lecture Notes in Computer Science, Vol. 1777, pages 21–35, Springer, 2000.
- [WSB98] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *Proc. of the 24th Int. Conf. on Very Large Data Bases, VLDB'98, New York City, August 24–27, 1998*, pages 194–205. Morgan Kaufmann Publishers, San Francisco, CA, 1998.
- [YF00] Byoung-Kee Yi and Christos Faloutsos. Fast Time Sequence Indexing for Arbitrary  $\mathcal{L}_p$  Norms. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 385–394, Morgan Kaufmann, 2000.
- [YOTJ01] C. Yu, B. C. Ooi, K.-L. Tan, and H.V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proc. of the 27th Int. Conf. on Very Large Data Bases, VLDB'01, Roma, Italy, September 11–14, 2001*, pages 421–430, Morgan Kaufmann Publishers, 2001.