Hierarchical Simplex Approximation - Towards Improved Index Compression for Efficient Nearest-Neighbor Retrieval

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

June 2002

Abstract

In this paper, we propose a novel approximation technique for efficient high-dimensional nearest neighbor retrieval: the *Simplex* approach. That is, we advance the idea of hierarchical local approximation introduced by our *Active Vertice* approach. By utilizing simplex-shaped cluster geometries we aim at (i) sustaining short approximation bit codes, and (ii) taking advantage out of shared bit code prefixes.

This paper introduces foundations of regular simplexes and explains the ideas of hierarchical simplex approximation. Moreover, we present an efficient algorithm for the computation of the *LowerBound*/*UpperBound* parameters along with their side problems. Following, we discuss data point approximation by regular simplexes and present an efficient insertion algorithm.

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we introduce the *Simplex* approach. That is, we briefly sketch the idea of hierarchical Simplex approximation for efficient high-dimensional Nearest Neighbor retrieval and relate it to our well known Active Vertice approach. Secondly, we provide a short survey of related work in this area.

1.1 Hierarchical Simplex Approximation

In this paper, we aim at improving our previous approximation approach for high-dimensional nearest neighbor retrieval: the *Active Vertice* proposal [Bal02, BS02, BSS02], cf. Figure 1.1. Unlike other approximation approaches (e.g. the *VA-file* [WSB98,

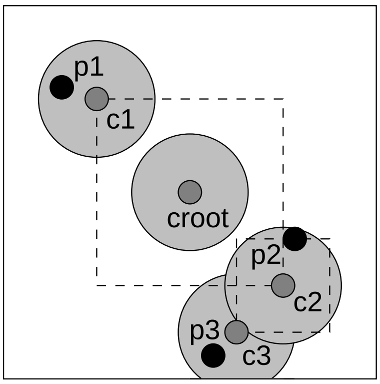

Figure 1.1: The Active Vertice Approach

WB97]), the Active Vertice approach bases on a hierarchical (i.e. *Quad-tree* [Sam84] like) partition of the vector space. In this way, we obtain a local data point approximation which superiorly adapts to actual data point distributions than global approximation techniques like the VA-file. In [BS02] we compared the Active Vertice approach and the

VA-file in terms of their retrieval performance. We proved the advantage of the Active Vertice approach both, experimentally, and analytically.

Besides its hierarchical local approximation, the Active Vertice approach seeks to take advantage out of shared approximation bit code prefixes. Figure 1.2 depicts this situation. The approximation bit codes of the data points p_1, p_2, and p_3 (cf. Figure 1.1)

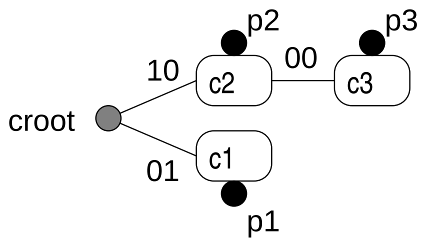

Figure 1.2: Approximation Bit Codes in Active Vertice Tree

are inserted into a tree-like structure. Since each level in the approximation hierarchy corresponds to a hypercube with 2^d vertices, the Active Vertice approach requires d bits to uniquely identify one of these vertices. In this example scenario, p_2 and p_3 have a shared bit code (10) in the first level of their approximation hierarchy. Our proposed storage method takes advantage out of shared bit code prefixes.

However, since the number of hypercube vertices grows exponentially (2^d) as the dimensionality d increases, it becomes increasingly unlikely to have shared bit code prefixes. Therefore, the approximation bit codes are singularly stored for high dimensionalities.

The idea of the Simplex approach is to maintain the advantages of hierarchical, local approximation and to overcome the exponential growth of vertices by using an alternative approximation shape. This approximation shape must clearly not be hypercubic.

Therefore, we propose to utilize *regular simplexes* (simplex, hyper-polyhedra) as approximation shapes. A simplex is the generalization of a tetrahedron to a dimensionality of d[1]. An n-simplex in $\mathbb{R}^d$ is bordered by $(n-2), \ldots, 0$-dimensional faces (boundary simplexes). In fact, there are n 0-faces (vertices), $\frac{(n-1)n}{2}$ 1-faces (edges), and $\binom{n}{i+1}$ i-faces.

In contrast to a hypercube, a *complete* regular simplex has exactly $d+1$ vertices. The length of the bit code to uniquely address one vertice of this $(d+1)$-simplex is, thus, upper bounded as follows:

$$|\text{bit code}| = \lceil \log_2(d+1) \rceil$$

Furthermore, we define an n-simplex s as a set of vertices v_i, i.e.:

$$s = \{v_1, \ldots, v_n\}$$

where

$$\forall i, j, k \text{ with } 1 \leq i, j, k \leq n \text{ and } i \neq j \text{ and } i \neq k : \; \|v_i - v_j\|_2 = \|v_i - v_k\|_2$$

[1] C.f. http://mathword.wolfram.com/Simplex.html

holds[2]. That is, all points in a regular simplex have the same distance among each other. Please note, that this definition holds for n-simplexes with $n \geq 3$, only. By definition, any set s with $1 \leq |s| \leq 2$ and $\forall v \in s : v \in \mathbb{R}^d$ with $d \in \mathbb{N}$ is also a regular simplex.

Unlike the Active Vertice approach, the Simplex approach bases on simplex-shaped approximation regions which are defined by a non-disjoint, complete hierarchical data space partition. Starting with a "root" simplex, each simplex has $d+1$ subsequent "child" simplexes (subsimplexes) which are assigned to the simplex vertices. In Figure 1.3 the

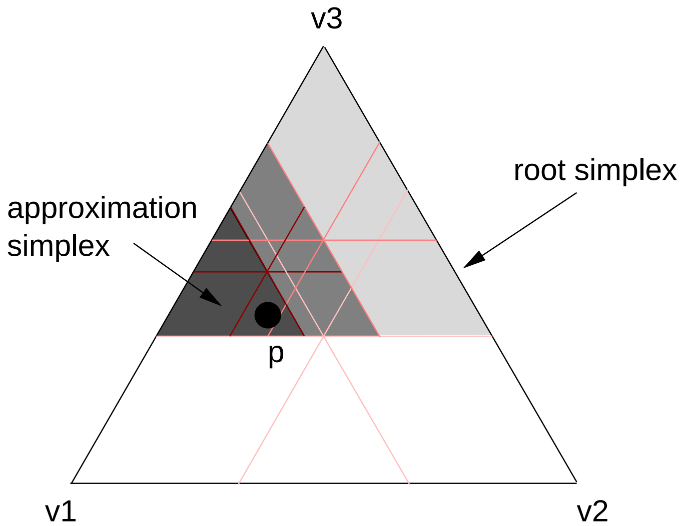

Figure 1.3: Hierarchical Simplex Approximation

root simplex in $\mathbb{R}^2$ is defined as follows:

$$s_{\text{root}} = \{v_1, v_2, v_3\}$$

The data point (*here:* p) shall be approximated by a child simplex at a hierarchy level of $t = 3$. Since each child simplex may uniquely be assigned to a vertice of the superordinated simplex, we may identify a child simplex by a unique bit code which labels the vertice. In our example, we specify the following bit codes:

- v_1: 00
- v_2: 01
- v_3: 10

In this way, an approximation simplex which lies at a certain hierarchy level t may be represented as a list of t bit codes. In our example, the approximation simplex of p has the following bit code sequence:

$$p \rightarrow 10, 00, 00$$

[2]Please note, that $\| \bullet \|_2$ denotes the Euclidean (L_2) norm.

Please note, that the subsequent child simplexes are depicted in different shades of grey. In Chapter 4 we discuss the maintenance of our indexing method more thoroughly.

1.2 Related Work

In this Section, we give a brief overview on related work concerning approximation techniques. We present both, external and own approaches for efficient high-dimensional nearest neighbor retrieval.

1.2.1 External Approaches

The approach which is most referred to is the *Vector Approximation* (VA-) file approach [WSB98, WB97] (cf. Figure 1.4). The main contribution of the VA-file is to separate each

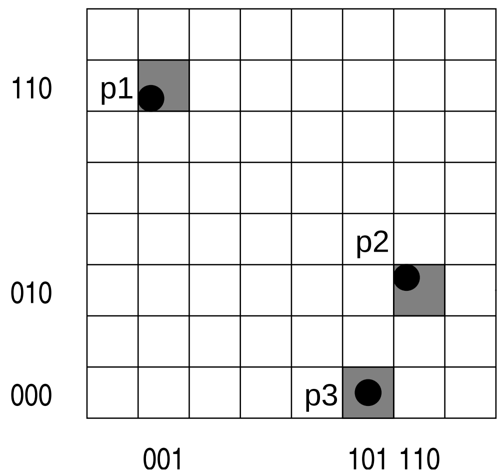

Figure 1.4: VA-File in $\mathbb{R}^2$

dimension i of the data space into 2^{b_i} intervals. These intervals are chosen to ensure a uniform distribution of the data points to the intervals. Each of these intervals is labeled with a unique bit code B_i with b_i bits:

$$|B_i| = b_i$$

In the unit cube, each combination of one interval from each dimension marks a hyper-rectangular region in the data space. This hypercube is considered as the approximation region labeled by the bit code B which is a concatenation of the bit codes B_i from the single dimensions[3]:

$$B = B_1 \otimes B_2 \otimes \ldots \otimes B_d$$

[3]Please note, that $\otimes$ denotes the concatenation operator.

For all d dimensions this adds up to a length

$$|B| = \sum_{i=1}^{d} b_i$$

bits for this bit code. In our example in $\mathbb{R}^2$ (Figure 1.4) we use three bits in both dimensions. That is

$$b_1 = b_2 = 3$$

holds. Therefore, we have $2^3 = 8$ intervals in each dimensions. For the sake of simplicity, we used equally sized intervals in this example. The tree data points p_1, p_2, and p_3 are located in the approximation regions with the following bit codes:

$$\begin{aligned} p_1 &= 001\ 110 \\ p_2 &= 110\ 010 \\ p_3 &= 101\ 000 \end{aligned}$$

In terms of the Nearest-Neighbor retrieval, the VA-file approach roughly bases on the GEMINI approach [Fal96]. That is, the approximation bit codes are stored in a flat file. During the retrieval process, this file is completely scanned and the LowerBound and UpperBound distances from the query point q to each of the approximation hypercubes are computed. Correspondingly, a list of candidate bit codes is processed out of the complete set of bit codes. As a thumb rule, we may state that the higher the compression ratio (i.e. the shorter the approximation bit codes), the more candidates have to be considered in the second stage of the retrieval process.

Precisely, the second stage processes the candidate list (typically ascendingly ordered by the LowerBound parameter) one by one and retrieves the data point(s) p_i which are assigned to the bit codes. By computing the actual distances from q to p_i, the actual nearest neighbor data point is determined.

The advantage of the VA-file is primarily its simplistic implementation. Furthermore, is has actually proven to avoid the well-known "curse of dimensionality" in high-dimensional nearest neighbor retrieval. However, the VA-file proposal has some drawbacks, as well. For instance, for non-uniform data distributions a preprocessing step must determine the interval boundaries in the single dimensions to ensure a uniform distribution of the data points to the intervals. Furthermore, (hyper-)rectangular approximation shapes are not ideal. Even more, a fixed grid (which is implied by the VA-file approximation principle) may not be appropriate for all data distributions.

Recently, a number of VA-file descendants seek to improve the retrieval performance, i.e. yield a higher compression while maintaining a good selectivity in the first (filtering) stage of the retrieval algorithm.

For example, the IQ-tree [BBJ+00] uses a tree of VA-file like grids with different granularity. The authors report a superiority of their approach for dimensionalities up to $d = 16$ and mainly real data sets. The IQ-tree turns out not to be an improvement for higher dimensionalities and arbitrary data sets (e.g. uniformly distributed data points).

The A-tree [SYUK00] pursues a similar idea. By means of so called *virtual bounding rectangles* (VBRs) the data space is, again, scattered into a hierarchy of approximations with different granularity. The authors compare their A-tree with the VA-file and report a superiority of their approach for some obscure real data sets and dimensionalities up to $d = 64$. Again, for uniform data sets, there is no improvement in comparison to the VA-file. Even more, as there is no generally approved real data set, the results of their experiments can not be confirmed.

Finally, in [FTAA00] another improvement of the VA-file is presented. The authors argue to adapt the approximation grid to the actual data distribution. They apply plenty preprocessing steps (e.g. Karhunen Loeve transform, optimal interval quantization) to yield a superior approximation (tighter bounds). In case of their example dataset, this approach actually outperforms the VA-file. However, again, it is not clear if this approach can be adopted for arbitrary data sets. Also, the preprocessing is expected to be very costly for large data sets.

1.2.2 Own Work

We started our research by an investigation of distances between uniformly distributed data points and approximation errors for convex cluster geometries in [Sch01]. As a result, we obtained a statistical model for the expecting value and standard deviation for distances between data points:

$$\begin{aligned} \mathrm{E}(X)_{\|V-V\|_2} &\to_p \sqrt{\frac{d}{6}} \\ \sigma_{\|V-V\|_2} &\to_p \text{const} \approx 0.24 \end{aligned}$$

Based on these results, we proposed the *R-Shell* cluster in [BS01b]. The R-Shell cluster is a concave variant of a well known convex cluster shape adapted from the SR-tree [KS97]. By excluding large empty regions from the cluster shape, this geometry seeks to obtain tighter bounds (i.e. a lower approximation error).

In [BS01a] we improved the idea of concave cluster geometries with the *Active Vertice* cluster. We could experimentally show both a progress in our approaches as well as a superiority in comparison to convex cluster shapes. Finally, we broadened the idea of a concave cluster geometry towards approximation techniques by means of the Active Vertice approach [Bal02, BS02, BSS02] which was already sketched in Section 1.1.

Chapter 2

Technical Foundations

In this chapter, we derive formulas to compute the radius[1] and height of regular simplexes. Based on this, we formally depict the principles of hierarchical simplex approximation.

2.1 Radius and Height of Regular Simplexes

In this section, we derive simple formulas to compute

- the radius r and
- the height h

of a regular n-simplex s in $\mathbb{R}^d$ with a given edge length of l. These parameters are defined as follows. The radius r denotes the distance of the center point

$$c = (y_1, \ldots, y_d)$$

to an arbitrary vertice point v_i, i.e. the radius of the enclosing hyperball.

The height h denotes the distance of an arbitrary vertice point v_i to the opposite boundary hyperplane of s (cf. Figure 2.1). A regular simplex may have $d+1$ vertices, at most. That is, $1 \leq n \leq d+1$ holds.

To simplify our following considerations we specify a *canonical* regular n-simplex $s = \{v_1, \ldots, v_n\}$ in $\mathbb{R}^d$ (with $n \leq d$) by the vertice points v_i as follows:

$$v_i = (\underbrace{0, \ldots, 0}_{i-1}, x, \underbrace{0, \ldots, 0}_{d-i})$$

[1] Radius of smallest enclosing hyperball.

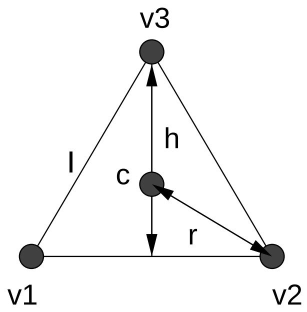

Figure 2.1: Parameter of a 3-Simplex

In this way, we can specify up to d simplex vertices as scaled unit vectors[2] $x \cdot \mathfrak{e}_i$ The distance between two different vertice points v_i, v_j $(i \neq j)$ (without loss of generality: v_1 *and* v_2) is given by the edge length l. Therefore, we obtain:

$$l = \|v_1 - v_2\|_2 = \|(x, 0, \ldots, 0) - (0, x, 0, \ldots, 0)\|_2$$

This boils down to $l = x \cdot \sqrt{2}$, i.e.

$$x = l/\sqrt{2}$$

holds for the coordinate of x. For the computation of the center point c, we can state two observations:

(i) The center point c has a common distance of r to all vertice points v_i.

(ii) The center point c lies within the convex hull defined by the vertice points.

With respect to (i), we obtain the following equations:

$$\begin{aligned} \|v_1 - c\|_2 &= \|v_2 - c\|_2 \\ \|v_1 - c\|_2 &= \|v_3 - c\|_2 \\ &\vdots \\ \|v_1 - c\|_2 &= \|v_n - c\|_2 \end{aligned}$$

As the center point is given by $c = (y_1, \ldots, y_d)$, we obtain:

$$\begin{aligned} (x - y_1)^2 + y_2^2 + \ldots + y_d^2 &= y_1^2 + (x - y_2)^2 + y_3^2 \ldots + y_d^2 \\ &\cdots \\ (x - y_1)^2 + y_2^2 + \ldots + y_d^2 &= y_1^2 + \ldots + y_{n-1}^2 + (x - y_n)^2 + y_{n+1}^2 + \ldots + y_d^2 \end{aligned}$$

This can be simplified to:

$$y_1 = y_2 = \ldots = y_n \ (= y)$$

[2] Unit vector: $\mathfrak{e}_i = (\underbrace{0, \ldots, 0}_{i-1}, 1, \underbrace{0, \ldots, 0}_{d-i})$

That is:

$$c = (y, \ldots, y, y_{n+1}, \ldots, y_d)$$

The second observation (ii) results in the following formulas:

$$\begin{aligned} c &= \sum_{i=1}^{n} \lambda_i \cdot v_i \\ 1 &= \sum_{i=1}^{n} \lambda_i \end{aligned}$$

We obtain the following formulas:

$$\begin{aligned} y &= \lambda_1 \cdot x \\ &\ldots \\ y &= \lambda_n \cdot x \\ y_{n+1} &= 0 \\ &\ldots \\ y_d &= 0 \end{aligned}$$

Thus, we can conclude that

$$\lambda = \lambda_1 = \ldots = \lambda_n = \frac{1}{n}$$

holds. Finally, we obtain the following coordinates of the center point c:

$$c = (\underbrace{x/n, \ldots, x/n}_{n}, \underbrace{0, \ldots, 0}_{d-n})$$

Based on this formula, now, we can directly derive the radius r as follows:

$$\begin{aligned} r &= \|v_i - c\|_2 \\ r^2 &= (x - x/n)^2 + (n-1)(x/n)^2 \\ &= x^2 - \frac{2x^2}{n} + n\frac{x^2}{n^2} \\ &= x^2 - \frac{x^2}{n} \\ &= \frac{l^2}{2} - \frac{l^2}{2n} \\ r &= l\sqrt{\frac{n-1}{2n}} \end{aligned}$$

Similarly, we can compute the height of an n-simplex which is the distance between an arbitrary vertice point v_i and the opposite boundary simplex (i.e. its center point c').

Without loss of generality, we pick v_n and c' which is the center point of the boundary simplex opposite to v_n:

$$c' = \Big(\underbrace{\frac{x}{n-1}, \dots, \frac{x}{n-1}}_{n-1}, \underbrace{0, \dots, 0}_{d-n+1}\Big)$$

Now, we can easily derive a formula to compute the height as follows:

$$\begin{aligned} h &= \|v_n - c'\|_2 \\ h^2 &= (n-1) \cdot \left(\frac{x}{n-1}\right)^2 + x^2 \\ &= \frac{x^2}{n-1} + x^2 \\ &= \frac{l^2}{2n-2} + \frac{l^2}{2} \\ &= l\sqrt{\frac{n}{2n-2}} \end{aligned}$$

Both, radius and height of a regular simplex depend on

(i) the edge length l and

(ii) the number n of vertices

of the simplex. Thus, we formulate two functions radius and height which ara parameterized by these values as follows:

$$\begin{aligned} \mathsf{radius}(l, n) &= l \cdot \sqrt{\frac{n-1}{2n}} \\ \mathsf{height}(l, n) &= l \cdot \sqrt{\frac{n}{2n-2}} \end{aligned}$$

2.2 Simplex-Tree

With respect to our hierarchical simplex approximation, we define that two sequencing (i.e. *superordinated* and *subordinated*) simplexes s_i and s_{i+1} are in terms of their radii and heights related as follows:

$$r_i = h_{i+1}$$

For an illustration, please refer to Figure 2.2 where a superordinated 3-simplex $s_0 = \{v_1, v_2, v_3\}$ and one of its subsimplexes $s_1 = \{v_1', v_2', v_3\}$ (shaded in grey) are depicted. The center point of s_0 is labeled as c_0. The radius r_0 of s_0, i.e. the distance of any of its

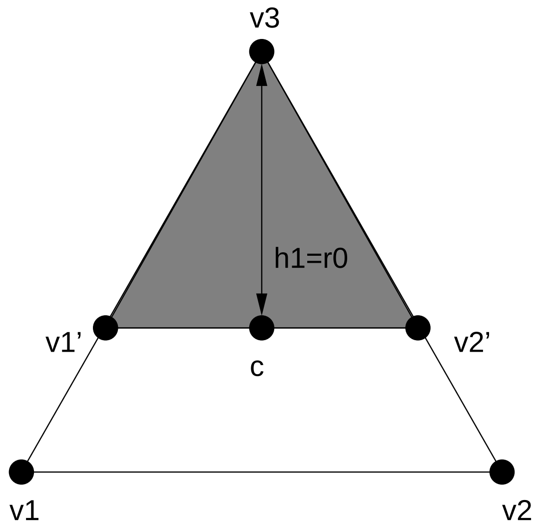

Figure 2.2: Radius and Height of a Simplex and one of its Subsimplexes

vertices (*here:* v_3) to c_0 obviously equals the height h_1 of s_1, i.e. the distance of any of its vertices (*here:* v_3) to the opposite boundary simplex (*here:* $\{v_1', v_2'\}$).
This procedure ensures a complete partition of the data space. However, we obtain potentially overlapping child simplexes. With respect to the corresponding edge lengths (l_i and l_{i+1}), this yields the following equations:

$$\begin{aligned} l_i\sqrt{\frac{n-1}{2n}} &= l_{i+1}\sqrt{\frac{n}{2n-2}} \\ &\dots \\ l_{i+1} &= l_i\frac{n-1}{n} \end{aligned}$$

That is, basing on some preset edge length of l_0 at the root-level, we can state the following formula to compute the edge length of a hierarchically determined simplex at a tree level of t:

$$\begin{aligned} l_t &= l_0 \cdot \prod_{i=1}^{t} \frac{n-1}{n} \\ &= l_0\left(\frac{n-1}{n}\right)^t \end{aligned}$$

2.2.1 Determination of the Root Simplex

In this section, we propose two alternatives to settle a root simplex. That is, we determine the vertice points of a root simplex (s_0) in a way, that any arbitrary data point of the data space lies in this root simplex and may be approximated by a subsimplex of s_0.

Unit Cube Enclosing Approach

The first approach to determine the root simplex is to completely enclose the data space (unit hypercube $[0,1]^d$) by the root simplex which has a fixed orientation. That is, to determine the edge length of the root simplex (s_0), we have to determine its edge length l_0 in a way that s_0 completely encloses the high-dimensional unit hypercube $[0,1]^d$ (cf. Figure 2.3). To find the proper value of l_0 we must consider the following conditions.

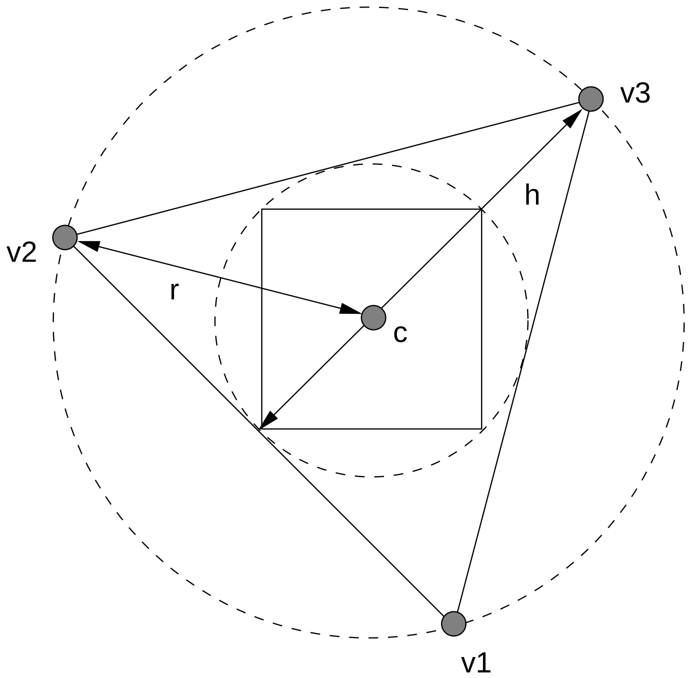

Figure 2.3: Inscription of Hypercube $[0,1]^d$ into a Regular Simplex

Since the vector space $[0,1]^d$ is hyperrectangular with an edge length of 1.0 we must settle a root simplex which completely encloses the vector space.

That is, $h_0 - r_0$ is the smallest possible distance from the center point c_0 of s_0 to the simplex boundary. Furthermore, $\frac{1}{2}\sqrt{d}$ is known to be the largest possible distance from the unit hypercube center $(0.5,\ldots,0.5)$ to the hypercube boundary (i.e. one of the simplex vertices). Thus, we can compute l_0 by equating both terms:

$$\begin{aligned}
\mathsf{height}(l_0,n) - \mathsf{radius}(l_0,n) &= \frac{\sqrt{d}}{2} \\
l_0\left(\sqrt{\frac{n}{2n-2}} - \sqrt{\frac{n-1}{2n}}\right) &= \frac{\sqrt{d}}{2}
\end{aligned}$$

Secondly, the root simplex vertices have to be determined. That is, based on a center point of

$$c_0 = (0.5,\ldots,0.5)$$

and the edge length l_0 the coordinates of $v_1, \ldots, v_{d+1}$ are fixed. In contrast to our previous considerations, both the root simplex and the unit cube have lie in the d-dimensional vector space $\mathbb{R}^d$. Thus, we create a canonical regular simplex $s'_0 = \{v'_1, \ldots, v'_{d+1}\}$ slightly different:

$$\begin{aligned} v'_1 &= (x, 0, \ldots, 0) \\ &\ldots \\ v'_d &= (0, \ldots, 0, x) \\ v'_{d+1} &= (z, \ldots, z) \end{aligned}$$

That is, the $(d+1)$-th vertice is created in a different manner. The value of z is given as follows ($1 \leq i \leq d$):

$$\begin{aligned} \|v'_i - v'_{d+1}\|_2 &= l \\ dz^2 - 2xz + x^2 - l^2 &= 0 \\ dz^2 - 2xz - x^2 &= 0 \\ z^2 - \frac{2x}{d}z - \frac{x^2}{d} &= 0 \end{aligned}$$

We obtain:

$$z = \frac{x}{d} \pm \sqrt{\frac{x^2}{d^2} + \frac{x^2}{d}}$$

We restrict our considerations to one case and obtain:

$$\begin{aligned} z &= \frac{x}{d}(1 + \sqrt{1+d}) \\ &= \frac{l_0}{d\sqrt{2}}(1 + \sqrt{1+d}) \end{aligned}$$

To compute the root simplex s_0 out of this canonical simplex s'_0, first, we have to compute the center point c'_0 of s'_0 as follows:

$$c'_0 = \frac{1}{d+1} \sum_{i=1}^{d+1} v'_i$$

Secondly, we add the difference of $c_0 - c'_0$ to each vertice v'_i to move the center of the root simplex s_0 to c_0:

$$s_0 = \{v_1, \ldots, v_{d+1}\} \mid v_i = v'_i + c_0 - c'_0$$

Simplex Transformation Approach

The second approach aims at reducing the large empty regions of the root simplex which result from the complete enclosing of the unit cube. Therefore, we propose a

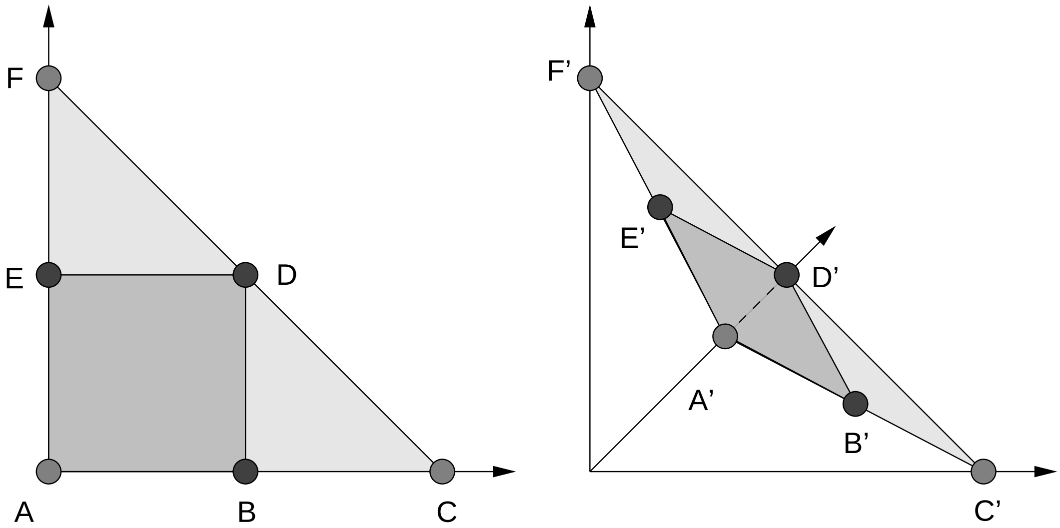

Figure 2.4: Simplex Transformation Approach

transformation which maps the unit cube in $\mathbb{R}^d$ to a root simplex in $\mathbb{R}^{d+1}$ with an edge length of $l_0 = \sqrt{2}$. In Figure 2.4 this transformation is illustrated. On the left hand side, the unit hypercube (in $\mathbb{R}^2$) is depicted with its four vertices:

$$\begin{aligned} A &= (0,0) \\ B &= (1,0) \\ D &= (1,1) \\ E &= (0,1) \end{aligned}$$

For illustration purposes, two more points are depicted:

$$\begin{aligned} C &= (2,0) \\ F &= (0,2) \end{aligned}$$

Please note, that A, C, F denote the polyhedron which is transformed to the root simplex in the simplex transformation step. Thus, on the right hand side of Figure 2.4 the root simplex (along with the transformed data space) is depicted in $\mathbb{R}^3$. Please note, that the transformed points are labeled with a prime tag.

The transformation must obey some requirements. That is, (i) any transformed point p' (with $p \in [0,1]^d$) must lie in the convex hull of the root simplex s_0. Furthermore, (ii) the "empty" regions of the root simplex should be as small as possible. Most importantly, (iii) the transformation must maintain angles (i.e. distance relations) between the data points.

Therefore, we propose the transformation simtran to map any data point $p \in [0,1]^d$ to its counterpart $p' \in \mathbb{R}^{d+1}$:

$$\mathsf{simtran} : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$$

Precisely, for $p \in \mathbb{R}^d$, that is[3]:

$$\mathsf{simtran}(p) = \left(\frac{p[1]}{d}, \ldots, \frac{p[d]}{d}, 1 - \frac{\sum_{i=1}^{d} p[i]}{d}\right) = p'$$

The root simplex is computed as follows:

$$s_0 = \{\mathsf{simtran}(0, \ldots, 0), \mathsf{simtran}(d, 0, \ldots, 0), \ldots, \mathsf{simtran}(0, \ldots, 0, d)\}$$

In case of our example, the root simplex in $\mathbb{R}^3$ is $\{A', C', F'\}$, the vertices of the unit hypercube are mapped to the points A', B', E', D' with the following coordinates:

$$\begin{aligned} A' &= (0, 0, 1) \\ B' &= (0.5, 0, 0.5) \\ C' &= (1, 0, 0) \\ D' &= (0.5, 0.5, 0) \\ E' &= (0, 0.5, 0.5) \\ F' &= (0, 1, 0) \end{aligned}$$

Both, the unit cube enclosing and the simplex transformation approach, have their pros and cons. The unit cube enclosing approach maintains the dimensionality of the data space. The simplex transformation approach, however, yields smaller empty regions of the root simplex.

2.2.2 Computation of Vertice Points

In this section, we provide formulas to compute a subsimplex s' out of a given superordinated simplex s and a certain (reference) vertice $v_{\text{code}} \in s$. That is, a subsimplex is always assigned to a certain vertice of its superordinated simplex. Be the superordinated simplex given as $s = \{v_1, \ldots, v_{d+1}\}$ and the subsimplex given as $s' = \{v'_1, \ldots, v'_{d+1}\}$. Furthermore, s' is assigned to the vertice $v_{\text{code}} \in s$. Thus, the vertices of s' are computed as follows[4]:

$$v'_i = \begin{cases} v_i & \text{if } v_i = v_{\text{code}} \\ v_{\text{code}} + l' \cdot (v_i - v_{\text{code}})^N & \text{otherwise} \end{cases}$$

Please note, that l' denotes the edge length of the subsimplex. Figure 2.5 illustrates this situation. The superordinated simplex is given as $s_1 = \{v_1^1, v_2^1, v_3^1\}$ and has an edge length of l_1.

The subsimplex $s_2 = \{v_1^2, v_2^2, v_3^2\}$ with an edge length of $l_2 = l_1 \cdot \frac{d}{d+1}$ shall be assigned to the vertice $v_3^1 \in s_1$. The vertices of s_2 compute as follows:

$$\begin{aligned} v_1^2 &= v_3^1 + l_2 \cdot (v_1^1 - v_3^1)^N \\ v_2^2 &= v_3^1 + l_2 \cdot (v_2^1 - v_3^1)^N \\ v_3^2 &= v_3^1 \end{aligned}$$

[3]Please note, that $p[i]$ denotes the i-th coordinate of p.

[4]Normalization operation: $(\bullet)^N = \frac{\bullet}{\|\bullet\|_2}$

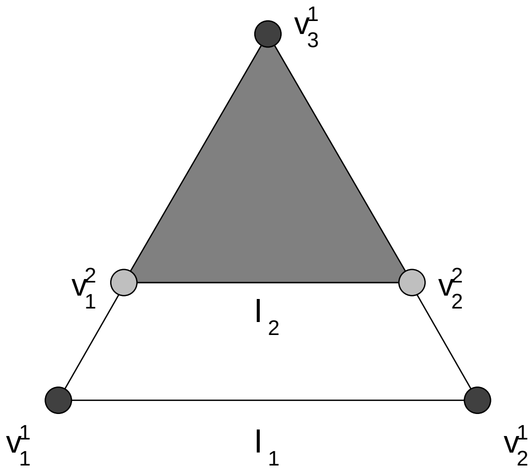

Figure 2.5: Computation of Vertice Points

This functionality is implemented by the subSimplex algorithm (cf. Figure 2.6) which is parameterized by an superordinated simplex s and one reference vertice v_{code}.

function subSimplex(s: Simplex, v_{code}: Point): Simplex
  Simplex $s' = \{\}$;
  $l =$ edge length of s;
  $l' = l \cdot \frac{d}{d+1}$;
  for each $v_i \in s$ **do**
    if $v_i = v_{\text{code}}$ **then** $s' = s' \cup \{v_i\}$;
    else $s' = s' \cup \{v_{\text{code}} + l' \cdot (v_i - v_{\text{code}})\}$;
  od
  return s';
end

Figure 2.6: Algorithm subSimplex(s, v_{code})

Chapter 3

Retrieval

In this chapter, we elaborately discuss all aspects of the nearest neighbor retrieval process. That is, we cite and adapt the retrieval algorithm from the VA-file approach. Moreover, we thoroughly discuss the computations of the LowerBound and UpperBound parameters of the approximation shapes along with estimations of the runtime complexities of the involved sub problems.

3.1 Retrieval Algorithm

In this section, we briefly sketch the Nearest Neighbor retrieval algorithm as introduced in the VA-file approach. We reuse this retrieval procedure which consists of two major stages and apply some slight adaptations to our Simplex approach.

In the first (*filtering*) stage the index of compressed data point representations is completely scanned and processed into a list of remaining candidates. For the VA-file, the index is simply an array of approximation bit codes of equal length ($d \cdot b$ bits). As the Simplex approach seeks to take advantage out of shared bit code prefixes, the index is a secondary storage representation of a prefix tree like structure.

The second stage of the algorithm retrieves the actual (i.e. non compressed) data point representations of the bit code candidates from secondary storage to determine the Nearest Neighbor point among them. In Figure 3.1 the retrieval algorithm is depicted in a pseudo code like notation.

In the scope of this considerations, we introduce two terms which make up the overall retrieval costs: compression and selectivity. While compression refers to the size of the index which has to be completely scanned, selectivity reflects the amount of data which has to be retrieved from secondary storage during the second stage of the algorithm. Precisely, we define the overall costs as:

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

```
function retrieveNN(q: Point): Point
      candidates = {}; nn = p_1; min = ||q - nn||_2;
  1   for i = 1 to N do
  s     if LB(q, B_i) <= min then
  t      if UB(q, B_i) < min then min = UB(q, B_i);
         candidates = candidates ∪ {B_i};
  s     endif
  t   od
  a   sort candidates ascendingly by LB;
  g   for each B_i ∈ candidates do
  e     if LB(q, B_i) <= min then
         if ||q - p_i||_2 < min then
          nn = p_i; min = ||q - p_i||_2;
         endif
        endif
      od
return nn
```

Figure 3.1: Algorithm retrieveNN

3.2 LowerBound and UpperBound

In this section, we provide an algorithm to compute the *LowerBound* (LB) and *UpperBound* (UB) distances from a query point q to an approximation simplex s. Please note, that we assume s to be defined as a set of n vertice points:

$$s = \{v_1, \ldots, v_n\}$$

The computation of $\mathsf{UB}(q, s)$ is trivial. Obviously, the farthest virtual point of the simplex must be located in one of the simplex vertices. Therefore,

$$\mathsf{UB}(q, s) = \max_{i=1}^{n} \|q - v_i\|_2$$

holds. In contrast, a computation of $\mathsf{LB}(q, s)$ requires more complicated considerations. We provide a recursive algorithm to compute the closest virtual point in the simplex in Figure 3.2.

The runtime complexity of this algorithm depends on the size of B, i.e. the number of boundary $(n-1)$-simplexes. We know, that an n-simplex comprises n boundary $(n-1)$-simplexes. Since $n = d + 1$ holds, we can upper bound the runtime complexity of closestPoint as:

$$\Theta(d + 1)$$

That is, we have a linear runtime complexity. To actually compute LB, we have to invoke:

$$\mathsf{LB}(q, s) = \|q - \mathsf{closestPoint(q,s)}\|_2$$

```
function closestPoint(q: Point, s: Simplex): Point
  set B = boundarySimplexes(s);
  for each b_i ∈ B do
   if not insideBoundary(q,s,b_i) then
    Point p = projectPoint(b_i,q);
    return closestPoint(p,s_i);
   endif
  od
  return q;
end
```

Figure 3.2: Algorithm $\mathsf{LB}(q, s)$

Figure 3.3 illustrates this situation in $\mathbb{R}^3$. Obviously,

$$\mathsf{closestPoint}(q, \{v_1, v_2, v_3, v_4\}) = v_4$$

holds in this example. Thus, $\mathsf{LB}(q, \{v_1, v_2, v_3, v_4\})$ simply computes as $\|q - v_4\|_2$.
In the next subsections, we provide efficient algorithms for the sub problems of boundarySimplexes, insideBoundary, and projectPoint.

3.2.1 Function **boundarySimplexes**

The algorithm in Figure 3.2 comprises several secondary problems. First of all, the function boundarySimplexes(s) computes the set of n boundary $(n-1)$-simplexes of a given n-simplex $s = \{v_1, \ldots v_n\}$:

$$\mathsf{boundarySimplexes}(s) = \bigcup_{i=1}^{n} \left\{ \bigcup_{\substack{j=1 \\ j\neq i}}^{n} \{v_j\} \right\}$$

Due to the simplicity of this problem, we abstain from providing an algorithm. The runtime complexity is clearly given by:

$$\Theta((d+1)^2)$$

In our example (cf. Figure 3.3) we obtain:

$$\mathsf{boundarySimplexes}(v_1, v_2, v_3, v_4) = \{\{v_1, v_2, v_3\}, \{v_2, v_3, v_4\}, \{v_3, v_4, v_1\}, \{v_4, v_1, v_2\}\}$$

3.2.2 Function **insideBoundary**

The function insideBoundary(p, s, b) determines, if a given point p lies inside a simplex s with respect to a given boundary simplex b. In other words, it checks if p lies on the same half space (defined by b) as the center point of s.

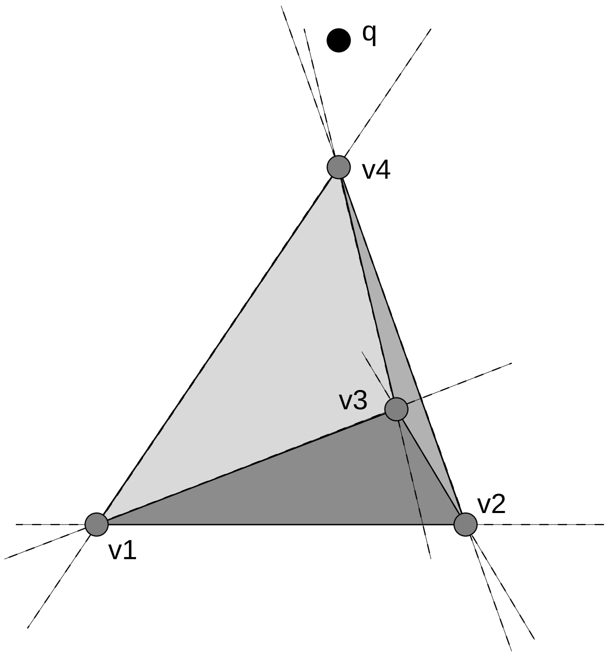

Figure 3.3: Computation of LB in a 4-Simplex

Figure 3.4 clarifies this problem. The 3-simplex s is given by $s = \{v_1, v_2, v_3\}$. It comprises three boundary 2-simplexes: $b_1 = \{v_1, v_2\}$, $b_2 = \{v_2, v_3\}$, and $b_3 = \{v_1, v_3\}$. The hyperplanes defined by b_1, b_2, and b_3 are depicted as dashed lines. The center point of s is labeled with c. Moreover, we have three exemplary query points q_1, q_2, and q_3 located at different positions. In Table 3.1 we depict the results of insideBoundary with respect to our example simplex and query points. We propose a very simplistic, yet

insideBoundary(q_i, s, b_j)	b_1	b_2	b_3
q_1	–	✓	✓
q_2	✓	–	–
q_3	✓	✓	✓

Table 3.1: Example Query Points (cf. Figure 3.4)

efficient approach to determine the result of insideBoundary(q, s, b_j) by taking advantage out of the following formula[1]:

$$\cos\theta = \frac{\langle x, y\rangle}{|x| \cdot |y|}$$

This theorem computes the angle θ between two (non-zero) vectors x and y. Figure 3.5 illustrates the utilization of this formula in the scope of the insideBoundary function. Be

[1] $\langle \bullet, \bullet \rangle$ denotes the scalar (inner) product.

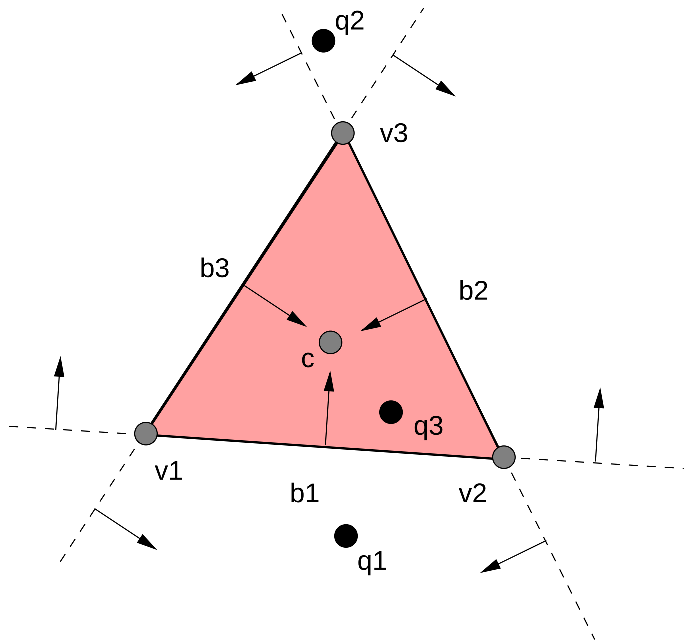

Figure 3.4: Position with respect to Halfspaces

c the center point of the 3-simplex $s = \{v_1, v_2, v_3\}$, c' the center point of the boundary 2-simplex $b = \{v_1, v_2\}$ and q the query point. We can state that $\mathsf{insideBoundary}(q, s, b)$ returns true, if the angle between the vectors $q - c'$ and $c - c'$ lies in a range of $-\frac{\pi}{2}, \ldots, \frac{\pi}{2}$. That is, for $c = \mathsf{centerPoint}(s)$ and $c' = \mathsf{centerPoint}(b)$ (function provided below),

$$\mathsf{insideBoundary}(q, s, b) = \text{true} \quad \Leftrightarrow \quad -\frac{\pi}{2} \leq \arccos\left(\frac{\langle c - c', q - c' \rangle}{\|c - c'\|_2 \cdot \|q - c'\|_2}\right) \leq \frac{\pi}{2}$$

holds.

Since $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2} \Rightarrow \cos\theta \geq 0$ holds, we can simplify this consideration as follows:

$$\begin{aligned} \mathsf{insideBoundary}(q, s, b) = \text{true} \quad &\Leftrightarrow \quad \frac{\langle c - c', q - c' \rangle}{\|c - c'\|_2 \cdot \|q - c'\|_2} \geq 0 \\ &\Leftrightarrow \quad \langle c - c', q - c' \rangle \geq 0 \end{aligned}$$

Provided that scalar product and vector subtraction are considered as atomic operations, the function $\mathsf{insideBoundary}$ has a constant runtime complexity of:

$$\Theta(1)$$

3.2.3 Function centerPoint

The function $\mathsf{insideBoundary}$ makes use of center point computations of regular simplexes. Therefore, we need to determine the center point out of the vertices of a k-simplex s

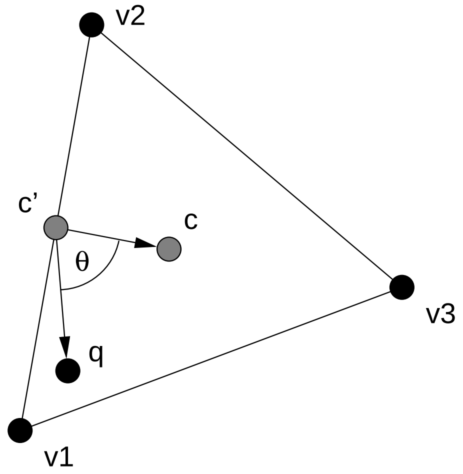

Figure 3.5: Function insideBoundary by Measurement of Angles

by means of the function centerPoint. In a regular simplex s, the center point can be efficiently computed out of the simplex vertices v_i as follows:

$$\mathsf{centerPoint}(s) = \frac{1}{k}\sum_{i=1}^{k} v_i$$

A proof of this formula can be found in [Ebe01]. The runtime complexity of this algorithm depends on the number of vertices k which are part of this simplex. Thus, we obtain the following runtime complexity:

$$\Theta(k)$$

3.2.4 Function projectPoint

The function $\mathsf{projectPoint}(p, b)$ computes the orthogonal projection of the point p onto the boundary simplex $b = \{v_1, \ldots, v_k\}$. To do so, we take advantage of the orthogonal projection theorem. That is, the boundary simplex defines a $(k-1)$-dimensional subspace $(\mathbb{R}^{k-1})$ of $\mathbb{R}^d$. If $o_1, \ldots, o_{k-1}$ is a orthonormal basis of this subspace, we can map a point $p \in \mathbb{R}^d$ to this subspace as follows:

$$\mathsf{mapPoint}(p) = \sum_{i=1}^{k-1} \langle o_i, p\rangle \cdot o_i$$

To compute a orthonormal basis of b, we can proceed as follows. Firstly, we define a set of $k-1$ spanning vectors $s_1, \ldots, s_{k-1}$ with:

$$s_i = v_{i+1} - v_1$$

Secondly, we apply the orthonormalization procedure by ERHARD SCHMIDT to obtain an orthonormal basis:

$$
\begin{aligned}
o_1 &= \frac{s_1}{\|s_1\|_2} \\
o_{m+1} &= \frac{s_{m+1} - \sum_{i=1}^{m} \langle s_{k+1}, o_i \rangle \cdot o_i}{\|s_{m+1} - \sum_{i=1}^{m} \langle s_{k+1}, o_i \rangle \cdot o_i\|_2}
\end{aligned}
$$

To compute the orthogonal projection by means of the mapPoint function, we also have to take into account that the origin[2] of $o_1, \ldots, o_{k-1}$ ($= \mathfrak{o}$) does not correlate with the origin of the spanning vectors $s_1, \ldots, s_{k-1}$ ($= v_1$). Therefore, we have to subtract (add) v_1 before (after) the invocation of mapPoint (cf. Figure 3.6). A formal substantiation

function projectPoint(p: Point, b: Simplex): Point
  set S = spanning vectors of b;
  set O = orthonormal basis of S;
  Point $p' = p - v_1$;
  Point p'' = mapPoint(p');
  return $p'' + v_1$;
end

Figure 3.6: Algorithm projectPoint(p, b)

of this orthogonal projection can be found in the German textbooks [Koe97, Beu01]. Provided that the computation of the spanning vectors and the orthonormal basis of b are considered as atomic operations, we have a constant runtime complexity of projectPoint:

$$\Theta(1)$$

[2] Null vector: $\mathfrak{o} = (0, \ldots, 0)$

Chapter 4

Maintenance

In this chapter, we introduce a dynamic data point insertion algorithm. That is, this algorithm may dynamically insert approximation bit codes to the index at runtime. Our considerations comprise

(i) the determination of an approximation simplex and its bit code,

(ii) the mapping to a secondary memory representation, and

(iii) further compression proposals (e.g. using a HUFFMAN encoding).

We propose two alternative insertion algorithms. In consequence, both aim at generating a small amount of approximation data, i.e. imply a high compression ratio. The first approach bases on a fixed simplex hierarchy depth of t whereas the second approach bases on a preset maximal distance r of a data point to an approximation simplex center point which results in variable values of t. We discuss advantages and drawbacks along with optimization opportunities for both approaches.

4.1 Fixed Hierarchy Depth

In this section, we introduce efficient insertion and maintenance algorithms for a fixed hierarchy depth of t. That is, we preset the number of subordinated (child-) simplexes relative to the root simplex to approximate an arbitrarily located data point p.

4.1.1 Insertion Algorithm

Remember, that the idea of the Simplex approach bases on a hierarchy of subsimplexes. In fact, a simplex approximation is a regular simplex which is located in a preset depth t of the approximation hierarchy. That is, starting at the root simplex s_0 the approximation simplex results from a t time invocation of the subSimplex function (cf. Section 2.2.2).

Precisely, since a subsimplex s_{i+1} of a simplex s_i corresponds to a vertice $v_j \in s_i$, the approximation simplex may be regarded as a sequence of t reference vertices. Each of these reference vertices identifies the subsimplex at the corresponding hierarchy level. Since there are exactly $d+1$ vertices (possible subsimplexes) at each tree level, we identify each vertice at one tree level with a unique integer key value. That is, the vertice v_j is identified by the key value of j.

Please note, that this key value does not identify the precise location of the corresponding vertice but rather its alignment/direction. Therefore, the same $d+1$ key values (integer values: $1, \ldots d+1$) are used at each hierarchy level. Figure 4.1 illustrates this situation where three subsequent simplexes (s_0, s_1, and s_2) are depicted. Their vertices are tagged with the corresponding key values. Obviously,

$$\begin{aligned} s_1 &= \mathsf{subSimplex}(s_0, v_3) \\ s_2 &= \mathsf{subSimplex}(s_1, v_1) \end{aligned}$$

holds. In Figure 4.2 we sketch an algorithm which returns a sequence of vertice (i.e.

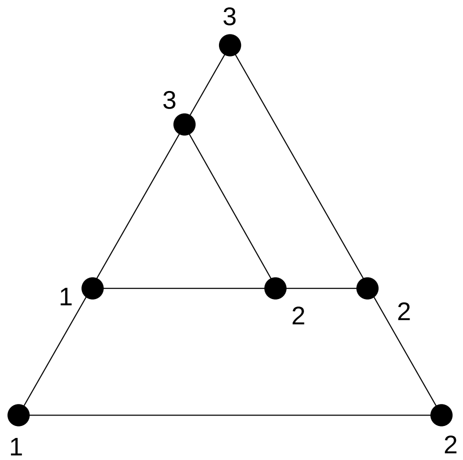

Figure 4.1: Key Values assigned to Simplex Vertices

subsimplex) identifiers for a given point p. The length of this sequence is preset by the tree depth t, $\mathsf{sequence}[i]$ refers to the i-th position in the sequence. The approximation simplex of p can be computed out of this sequence. This algorithm invokes the function insideSimplex which checks if a point lies inside a given simplex. The algorithm starts at the root simplex s_0. At the next hierarchy level, all possible subsimplexes are successively checked, if the data point p lies within their convex shape. The loop breaks, as soon as a "matching" subsimplex is found. Following, the subsimplex for the next hierarchy level is determined until a sequence of t simplexes is determined. We may upper bound the runtime complexity of getSimplexSequence as follows:

$$\Theta(dt + t)$$

```
function getSimplexSequence(p: Point): array of integer
  Simplex s = s_0;
  array sequence = {};
  for j = 1 to t do
    for each v_i ∈ s do
      Simplex s' = subSimplex(s,v_i);
      if insideSimplex(p,s') then
        s = s';
        sequence[j] = i;
        break loop
      endif
    od
  od
  return sequence;
end
```

Figure 4.2: Algorithm getSimplexSequence(p) (Fixed Hierarchy Depth)

We may state, that the following holds:

$$\mathsf{insideSimplex}(p, s) \;\Leftrightarrow\; \forall b_i \in \mathsf{boundarySimplexes(s)} : \;\mathsf{insideBoundary}(p, s, b_i)$$

That is, we can efficiently determine if a given point p lies within the convex shape of a simplex s by subsequently checking p with respect to all $(d+1)$ boundary simplexes. Thus, we obtain a runtime complexity of insideSimplex of:

$$\Theta(d+1)$$

4.1.2 Huffman Encoding

Next, the sequence of subsimplex identifiers must be translated into a sequence of bit code representations for secondary storage. Since getSimplexSequence processes potential subsimplexes in a fixed order and breaks as soon as a matching subsimplex is found, the used subsimplex identifiers will not be uniformly distributed. This is due to the fact, that the subsimplexes of a given simplex may overlap. Any data point which lies in an overlap area may be assigned to any of the overlapping subsimplexes.

In a given simplex s the inscribed sub-simplexes s_i $(i = 1, \ldots, n)$ overlap. In fact, a point p which is located in the overlap area of two or more sub-simplexes may be located in any of these simplexes. However, the insertion procedure ensures that p is inserted into the *first* possible sub-simplex. Figure 4.3 illustrates this situation. The 3-simplex

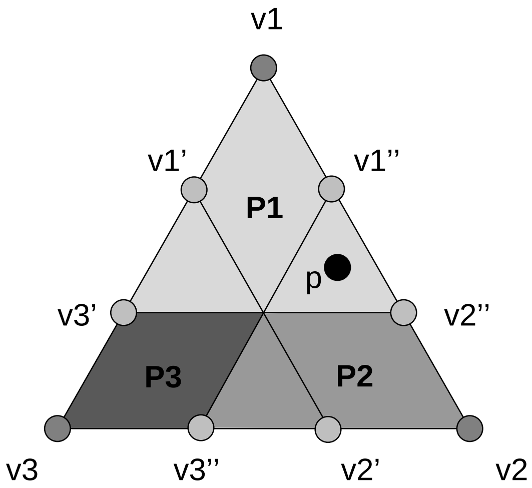

Figure 4.3: Probabilities of Sub-Simplexes

$s = \{v_1, v_2, v_3\}$ comprises three subsimplexes:

$$\begin{aligned} s_1 &= \{v_1, v_2'', v_3'\} \\ s_2 &= \{v_1'', v_2, v_3''\} \\ s_3 &= \{v_1', v_2', v_3\} \end{aligned}$$

When a new point p (cf. Figure 4.3) is inserted, all subsimplexes of s are processed in the specified order. Obviously, p fits in both s_1 and s_2. Despite of this fact, the algorithm will always select s_1 and proceed to the next hierarchy level. Therefore, there is a decreasing probability for the subsimplexes to be selected for further processing. In the example (in $\mathbb{R}^2$), these probabilities are reflected by the shaded areas (P_1, P_2, and P_3).

To achieve a better compression (i.e. lower the compression term), we apply a Huffman-style encoding to the subsimplex identifiers. That is, frequently used subsimplex identifiers will be tagged with shorter bit codes than rarely used identifiers. Unfortunately, it is somewhat complicated to analytically compute the probabilities of the different subsimplexes. For example, in $\mathbb{R}^1$ there is no overlap at all, i.e. both subsimplexes have the same probability which is not the case for higher dimensionalities. Furthermore, the subsimplexes may intersect with the data space (cf. Figure 2.3 on Page 12), so we only have to consider those volume which lies inside the unit hypercube. Finally, the effective volume may be related to the probability for uniformly distributed data, only.

Therefore, we compute these probabilities by means of a sample dataset which must be representative for data points to be inserted. As a result, we obtain a mapping from subsimplex identifiers to Huffman encoded bit codes. That is, none of these bit codes is a prefix of another bit code. For example, for a uniform data distribution in $\mathbb{R}^3$ and $t = 5$, we obtain the mapping depicted in Table 4.1.

Subsimplex Identifier	Bit Code	Probability
1	0	0.47
2	11	0.4
3	100	0.065
4	101	0.065

Table 4.1: Example Huffman Codes for $d = 3$, $t = 5$

4.2 Variable Hierarchy Depth

In contrast to a fixed hierarchy depth of t, we may also use different hierarchy depths depending on how well a data point is approximated by a (sub-)simplex at a certain (yet variable) hierarchy depth. Precisely, we introduce approximation hyperballs around the center points of the simplexes as a second approximation shape (cf. Figure 4.4). These hyperballs are defined by the center point c of the respective subsimplex s and a preset radius r. Clearly, an overlap of both approximation shapes may be utilized for a superior

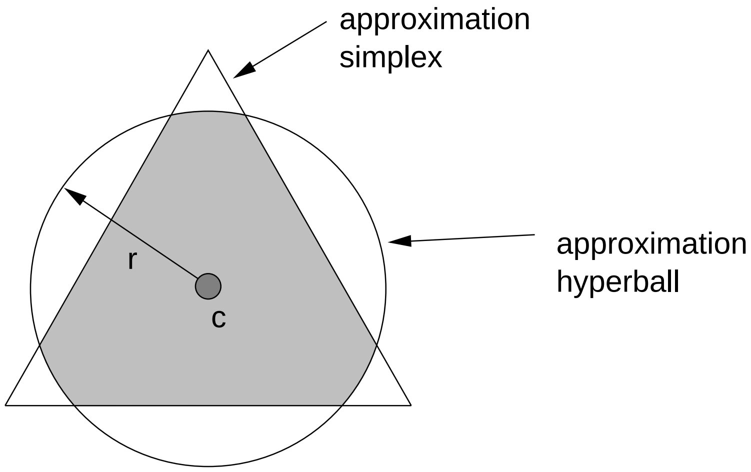

Figure 4.4: Alternate Approximation Shape

data point approximation. In Figure 4.4 this approximation shape is depicted as shaded area. That is, we slightly modify the computation[1] of LB and UB (cf. Section 3.2) as follows:

$$\begin{aligned}
\mathsf{LB}(q,s,r) &= \begin{cases} \max(\|q - \mathsf{closestPoint}(q,s)\|_2, 0) & \text{if } \|q-c\|_2 \leq r \\ \max(\|q - \mathsf{closestPoint}(q,s)\|_2, \|q-c\|_2 - r) & \text{otherwise} \end{cases} \\
\mathsf{UB}(q,s,r) &= \min(\max_{i=1}^{n} \|q - v_i\|_2, \|q-c\|_2 + r)
\end{aligned}$$

[1]Please note, that the modified formulas additionally take the parameter r into account.

Please note, that c is the center point of the simplex $s = \{v_1, \ldots, v_n\}$:

$$c = \mathsf{centerPoint}(s)$$

Besides a modified computation of the LB and UB parameters, the insertion algorithm differs from Figure 4.2 as well. That is, we aim at obtaining the shortest possible sequence of (subsequent) subsimplexes to approximate a data point p. In Figure 4.5 we present a modified algorithm which returns a sequence of subsimplex identifiers. Again, a data

```
function getSimplexSequence(p: Point, r: float): array of integer
  Simplex s = s_0;
  Point c = centerPoint(s);
  array sequence = {};
  while ||p − c||_2 > r do
   integer min = 0;
   float r_min = √d;
   for each v_i ∈ s do
    Simplex s' = subSimplex(s,v_i);
    if insideSimplex(p,s') then
     Point c' = centerPoint(s');
     float r' = ||c' − p||_2;
     if r' < r_min then
      r_min = r';
      min = i;
     endif
    endif
   od
   sequence = sequence ⊗ min;
   s = subSimplex(s,v_min);
   c = centerPoint(s);
  od
  return sequence;
end
```

Figure 4.5: Algorithm getSimplexSequence(p) (Variable Hierarchy Depth)

point may likely fit into many subsimplexes simultaneously. That is, we may choose one of these subsimplexes for further processing. In Section 4.1 we selected the subsimplex with the smallest subsimplex identifier by definition. In this new algorithm we choose the subsimplex with the shortest distance from the data point p to the subsimplex center point c. By doing so, our algorithm returns a remarkably shorter simplex sequence until the breakup criterion[2] is reached. In Figure 4.6 this situation is depicted. That is, a

[2] $\|p - c\|_2 > r$

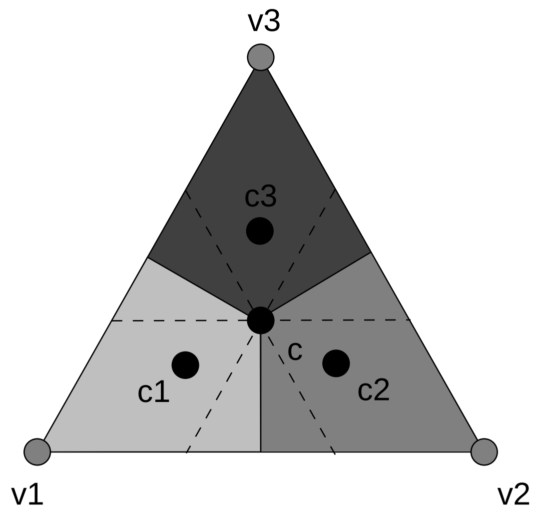

Figure 4.6: Data Point Assignments as Voronoi Diagram

simplex $s = \{v_1, v_2, v_3\}$ with its center point c is split up into three subsimplexes with their respective center points c_1, c_2, and c_3. Please note, that the subsimplex edges are depicted as dashed lines. Whenever a new point p which lies within the convex hull of s shall be inserted into the approximation hierarchy (i.e. an appropriate subsimplex is selected), the algorithm (cf. Figure 4.5) processes all subsimplexes. In particular, the distance of p to the respective center points is computed. Subsequently, the algorithm "picks" the subsimplex with the smallest distance for further processing.

As a result, we obtain a splitup of the simplex into Voronoi cells around the subsimplex center points (for a uniform data distribution). These Voronoi cells are included in their corresponding subsimplexes. Therefore, we may maintain the LB/UB computations. However, we know that no data point which lies outside a Voronoi cell is approximated by this particular subsimplex. Since the computation of Voronoi cell boundaries turns out to be numerically very costly, we stick to the consideration of this Voronoi cells as an intersection of an approximation simplex and one approximation hyperball.

Chapter 5

Summary

This paper is ment to be a "Technical Report" in the best sense. That is, it is mainly a reference work where we formally presented numerous ideas for an improved approximation technique for high-dimensional nearest neighbor retrieval.

In Chapter 1 we briefly surveyed related work on this field of research. Moreover, we recaptitulated our *Active Vertice* approach and motivated our novel proposal by the improvement potential of our previous work. Following, we informally introduced the basic ideas of the *Simplex* approach.

Chapter 2 deals with the vital technical foundations of the Simplex approach. That is, we derived formulas for the radius and height of regular simplexes, related the edge length of hierarchically ordered (sub-)simplexes and discussed two different proposals to settle a root simplex for this Simplex-tree.

In Chapter 3 we thoroughly discussed the most important aspects of the nearest neighbor retrieval algorithm for the Simplex approach. That is, we presented efficient formulas and algorithms to compute the LowerBound/UpperBound parameters for regular simplexes. We also stated the runtime complexity of these algorithms.

Finally, Chapter 4 introduced maintenance aspects of the Simplex approach. Mainly, that is, we presented two different data point insertion algorithms. Both proposals (fixed hierarchy depth vs. variable hierarchy depth) have different advantages and optimization strategies which we discussed in detail.

Bibliography

[Bal02] S. Balko. Compression Techniques Based on Concave Cluster Geometries for Efficient High-Dimensional Nearest Neighbor Retrieval. In *8th Conference on Extending Database Technology (EDBT), Ph.D. Workshop 2002, Prague, Czech Republic*, March 2002.

[BBJ+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 *ICDE*, pages 577–588, 2000.

[Beu01] Albrecht Beutelspacher. *Lineare Algebra - Eine Einführung in die Wissenschaft der Vektoren, Abbildungen und Matrizen*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig, Wiesbaden, 2001. ISBN: 3-528-46508-5.

[BS01a] S. Balko and I. Schmitt. Active Vertice Clusters – A Sophisticated Concave Cluster Shape Approach for Efficient High Dimensional Nearest Neighbor Retrieval. Preprint No. 24, Fakultät für Informatik, Universität Magdeburg, 2001.

[BS01b] S. Balko and I. Schmitt. Concave Cluster Shapes for Efficient Nearest Neighbor Search in High Dimensional Space. Preprint No. 23, Fakultät für Informatik, Universität Magdeburg, 2001.

[BS02] S. Balko and I. Schmitt. Efficient Nearest Neighbor Retrieval by Using a Local Approximation Technique - the Active Vertice Approach. Preprint No. 2, Fakultät für Informatik, Universität Magdeburg, 2002.

[BSS02] S. Balko, I. Schmitt, and G. Saake. Towards Enhanced Compression Techniques for Efficient High-Dimensional Similarity Search in Multimedia Databases. In *2nd Int. Workshop on Multimedia Data Document Engineering (MDDE'02), March 24, Prague*, 2002.

[Ebe01] David Eberly. Centers of a simplex. Technical report, Magic Software, 6006 Meadow Run Court, Chapel Hill, NC 27516, 2001. URL: http://www.magic-software.com/Documentation/circscrb.pdf.

[Fal96] Christos Faloutsos. *Searching Multimedia Databasees by Content.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1996.

[FTAA00] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. Vector approximation based indexing for non-uniform high-dimensional data sets. In *Proc. of the 2000 ACM CIKM Int. Conf. on Information and Knowledge Management, LcLean, VA, USA, November 6–11*, pages 202–209, 2000.

[Koe97] Max Koecher. *Lineare Algebra und analytische Geometrie.* Springer, Berlin, Heidelberg, New York, 1997. ISBN: 3-540-62903-3.

[KS97] Norio Katayama and Shin'ichi Satoh. The SR-tree: an Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, 1997.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[Sch01] I. Schmitt. Nearest Neighbor Search in High Dimensional Space by Using Convex Hulls. Preprint No. 6, Fakultät für Informatik, Universität Magdeburg, 2001.

[SYUK00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko 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.

[WB97] R. Weber and S. Blott. An Approximation-Based Data Structure for Similarity Search. Technical Report No. 24, ESPRIT project HERMES (no. 9141), 1997.

[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 24st Int. Conf. on Very Large Data Bases, VLDB'98, Ney York City, August 24–27, 1998*, pages 194–205. Morgan Kaufmann Publishers, San Francisco, CA, 1998.