Concave Cluster Shapes for Efficient Nearest Neighbor Search in High Dimensional Space

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

November 2001

Abstract

In this paper, we propose a new approach for cluster geometries used in high-dimensional nearest neighbor retrieval - the *R-Shell* cluster. In contrast to existing proposals, the R-Shell cluster features a concave geometry. We motivate our approach by the unsatisfactory approximation of nearest neighbor distances with convex clusters in high dimensions.

To find good approximations for the layout of high-dimensional points, we investigate distances and distributions in high-dimensional space. We introduce the *regular simplex* as a simple geometrical approximation for high-dimensional point layouts. Subsequently, we present our first proposal for concave cluster geometries which is based on established convex cluster geometries like enclosing boxed and hyperballs. Further efforts are put into the investigation of different appropriate clustering algorithms.

Finally, we compare convex and concave cluster geometries. We present the results of several experiments we ran.

Contents

List of Figures

Chapter 1

Motivation

Nearest neighbor search in high-dimensional space is an important problem for a wide spectrum of applications. For example, it is important for similarity retrieval in multimedia databases [Lu99] where multimedia objects are mapped to points in high dimensional space and a distance function measures the dis-similarity among them. There is a strong need for efficient data structures to find nearest neighbors for a given query point faster than checking every point sequentially (linear cost). If the number of dimensions is very low then there exist data structures which work in nearly logarithmic time (see [GG98] for an overview). But all general data structures known so far fail if the number of dimensions is too high ($d > 20$) (cf. for example [WSB98]). This problem is often called the *curse of dimensionality*.

The basic idea of our approach is to use a concave cluster shape for indexing instead of a convex shape which is traditionally used, e.g. in SR-tree [KS97], TV-tree [LJF94], X-tree [BKK96]. We will motivate our approach by showing limitations of convex clusters in high-dimensional space.

We assume a finite set $P = \{p_1 ... p_N\}$ of N data points in the d-dimensional unit cube ($p_i \in [0..1]^d$). Let D be a subset of P. The convex hull over $D \subset P$ is the set

$$CH_D = \{p \in [0..1]^d | p = \sum_1^N \lambda_i * p_i \text{ with } \lambda_i \in [0..1], p_i \in D \text{ and } \sum_1^N \lambda_i = 1\}.$$

Every cluster as subset of $[0..1]^d$ which is a superset of the convex hull is called a *convex* cluster. More formally, a cluster geometry is called convex if it can be specified by a convex function. Otherwise it is a *concave* cluster. When we speak about cluster points it is important to distinguish between data points from D which the cluster is based on and cluster points themselves. Obviously, D is always a subset of the points of a cluster. In the following we use the term *data points* if we refer to points of D. The cluster points will be referred to as *implicit points* or simply *points*.

Here we assume data points to be uniformly distributed and d dimensions to be independent from each other. Correlations in real data sets can be eliminated by a number of dimension reduction methods such as PCA [FK70, MKB79], MDS [KW78] and Fastmap [FL95]. We assume to have a data point set P after the application of one of the mentioned dimension reduction methods.

At first we will introduce some definitions and notations used in the rest of the paper. The L_2-norm of a vector v is defined as

$$||v||_2 = (v^T v)^{\frac{1}{2}} = \sqrt{\sum_{i=1}^{d} |v_i|^2} \text{ for } v \in [0..1]^d.$$

The L_2-distance between two points $v, w \in [0..1]^d$ is $||v - w||_2$.

The nearest neighbor of a given query point $q \in [0..1]^d$ is a data point from P having the smallest distance from q:

$$nn(q) = p \in P \text{ with } \forall p_i \in P : ||q - p||_2 \leq ||q - p_i||_2.$$

[Sch01] shows that the expecting value of the L_2-distance between two arbitrarily chosen points from P tends towards $\sqrt{d/6}$. There are many further publications, such as [AHK01, BBKK97, WSB98], which confirm the growth of L_2-distances when the number of dimensions increases. While the distances are growing the standard deviation remains stable. Both effects were investigated in [Sch01, AHK01] and are one important reason for the so called "curse of dimensionality": Up to now, in high-dimensional space ($d > 30$) there is no general index structure and algorithm to find the nearest neighbor in a uniformly distributed data set substantially faster than it can be found by testing every point sequentially.

As next, we will briefly discuss the relationship between the statistical results of L_2-distances and the curse of dimensionality.

The general approach to find the nearest neighbor for a given query point is to preprocess the points set P into an index. The nodes are arranged hierarchically in either an index tree or a cluster tree. The tree is used to perform the nearest neighbor search using a branch and bound algorithm [NG88] which is a classical technique from combinational optimization. Clusters are represented by tree nodes. Figure 1.1 sketches a version of a typical algorithm of the branch and bound algorithm.

The critical line of the algorithm is line 2. If the smallest possible distance between the query point and a node (cluster) is greater than the nearest neighbor distance found so far then the node can be pruned. The nearest neighbor distance found so far cannot be smaller than the final nearest neighbor distance. We use the ratio between nodes to be searched and the number of all nodes as relative search cost since it gives the fraction of cost in comparison with the linear scan:

$$cost = \frac{\text{number of nodes to be searched}}{\text{total number of nodes}} \geq \frac{|\{\text{node } n | LB(n) \leq ||q - nn(q)||_2\}|}{\text{total number of nodes}}.$$

Let $||q - nn(q, n)||_2$ be the distance between the query point q and its nearest data point within node n. The low efficiency of high-dimensional indexing is caused primarily by

```
algorithm BranchAndBound(QueryPoint: q; Node: c)
   Initialize UpperBound to the minimum of the distance of q from a
   random sample of points in P

   Examine the nodes in the tree one by one in a given order

   for each node n ∈ c do
     if n is a leaf node then
       for each point p ∈ n do
         Distance is distance between p and q
         if Distance < UpperBound then do
           UpperBound = Distance
           NearestNeighbor = p
         end
       end
     else do
1:     LB(n) = lower bound of the distance from q to any possible point in cluster n
2:     if LB(n) > UpperBound then
                   prune n and its descendents from further contention
3:     else recursively call BranchAndBound(q, n)
     end
   end
```

Figure 1.1: The branch and bound method for nearest neighbor search

the difference between $||q - nn(q,n)||_2$ and $LB(n)$. We denote this distance difference by $error(n) = ||q - nn(q,n)||_2 - LB(n)$. This approximation error is usually greater than zero as result of the cluster approximation. Cluster approximation is needed to compute very fast the cluster distances $LB(n)$ without checking the cluster points explicitly. Using the definition of the approximation error we obtain

$$cost \geq \frac{|\{\text{node } n|\ ||q - nn(q,n)||_2 - error(n) \leq ||q - nn(q)||_2\}|}{\text{total number of nodes}}$$

and

$$cost \geq \frac{|\{\text{node } n|\ ||q - nn(q,n)||_2 - ||q - nn(q)||_2 \leq error(n)\}|}{\text{total number of nodes}}.$$

We reformulate the formula by using a statistical notation and obtain:

$$cost \geq P(||q - nn(q,n)||_2 - ||q - nn(q)||_2 \leq error(n)).$$

As proved in [Sch01] the approximation error of a node described by a convex hull of its data points is at least linearly dependent on the expecting value of L_2-distances.

Since the L_2-distances grow with increasing number of dimensions, the right term of the inequality (the approximation error) grows, too. Let us now check the left term. It expresses the difference between two point distances. Since the standard deviation is stable independently from the number of dimensions the left term is limited by an upper bound. The probability of the left term to be smaller than the right term tends towards 100% when d increases. The development of both terms is therefore responsible for the curse of dimensionality as far as convex clusters[1] of uniformly distributed data points are involved. The problem is the growing approximation error in contrast to the stable distance differences when the number of dimensions increases.

As result from this discussion we do not try to find a new convex cluster geometry to improve the search costs. Our idea is to use a *concave* cluster geometry instead.

[1] Every convex cluster over a set of data points encloses the convex hull of that data points.

Chapter 2

The *R-Shell Cluster*: A Concave Cluster

In this chapter, we introduce a first concave cluster geometry: the *R-Shell Cluster*. The R-shell cluster consists of a minimal bounding box and two balls in $[0..1]^d$. The centers of the spheres coincide within the box. The outer ball is the smallest ball containing the cluster points and the inner ball the largest ball containing no cluster point. We denote the box by box, the outer ball by BALL, and the inner ball by ball. By definition we require every cluster point to lie in $\mathsf{box} \cap \mathsf{BALL} \setminus \mathsf{ball}$. Since the inner ball does not contain any point the resulting cluster is concave. Figure 2.1 shows a 2-dimensional R-shell cluster. After introducing the R-shell cluster three questions arise:

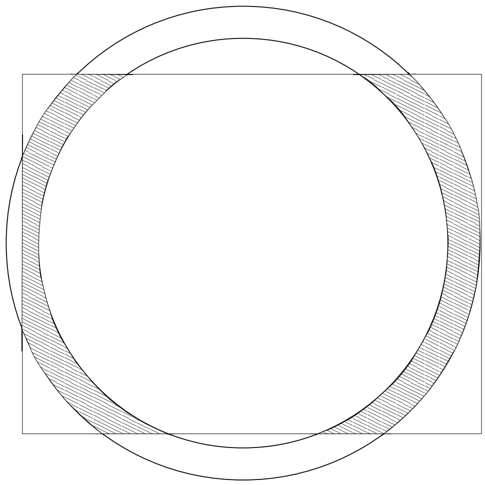


Figure 2.1: R-shell Cluster in $\mathbb{R}^2$

1. Which data points of P should be grouped into one R-Shell?

2. How to find the right R-Shell for a given group of data points?

3. How can LB of an R-shell be computed for a given query point?

Whereas the answers to the first two questions are discussed in the following chapters we give the answer to the last question in the rest of this section.

The box is specified by two d-dimensional points named Le and Ri. These points represent the "smallest" and the "largest" corner, respectively:

$$\mathsf{box} = [\mathsf{Le}_1 \dots \mathsf{Ri}_1] \times \dots \times [\mathsf{Le}_d \dots \mathsf{Ri}_d] = \mathrm{X}_{i=1}^{d}[\mathsf{Le}_i \dots \mathsf{Ri}_i] \text{ and } \mathsf{Le}_i, \mathsf{Ri}_i \in [0 \dots 1]^d.$$

Both balls have the same center p_c but different radii R and r, respectively:

$$\mathsf{BALL} = \left\{x \in [0 \dots 1]^d \mid ||x - p_c||_2 \leq R\right\}$$

and

$$\mathsf{ball} = \left\{x \in [0 \dots 1]^d \mid ||x - p_c||_2 \leq r\right\}.$$

LB is the distance between the query point and its nearest possible point in an R-shell:

$$LB = \min\{||x - q||_2 \mid x \in \mathsf{box} \cap \mathsf{BALL} \setminus \mathsf{ball}\}$$

In order to find LB algorithmically we first derive the projection of the query point q onto the box:

$$p = \text{project}(q) = \begin{pmatrix} p_1 \\ \cdot \\ \cdot \\ p_d \end{pmatrix} \text{ with } p_i = \begin{cases} \mathsf{Le}_i & \text{if } q_i \leq \mathsf{Le}_i \\ \mathsf{Ri}_i & \text{if } q_i \geq \mathsf{Ri}_i \\ q_i & \text{otherwise} \end{cases}$$

The projection always yields a box point. With respect to the balls we distinguish three cases:

1. *outside*: if $R < ||p - p_c||_2$,
2. *shell*: if $r \leq ||p - p_c||_2 \leq R$, and
3. *inside*: if $||p - p_c||_2 < r$.

Figure 2.2 shows the three regions of query points in correspondence to the three cases of the projections.

If p belongs to the shell region then it is itself the nearest possible R-shell point and LB is the distance between q and p.

If p is an inside point then we compute the next point on the surface of the inner ball and within the box by using the algorithm $nextPoint(p, r, p_c, \mathsf{Le}, \mathsf{Ri})$ (see Figure 2.4).

The idea to compute the next point on the surface of a ball is to draw a line between the center p_c and p and to compute the position where the line penetrates the sphere. The variable t expresses the ratio between the radius and the distance between p_c and p. Its value is used to find the right position $pnew$ on the line where the line meets the sphere (see Figure 2.3(a)). Unfortunately, two special cases require special attention:

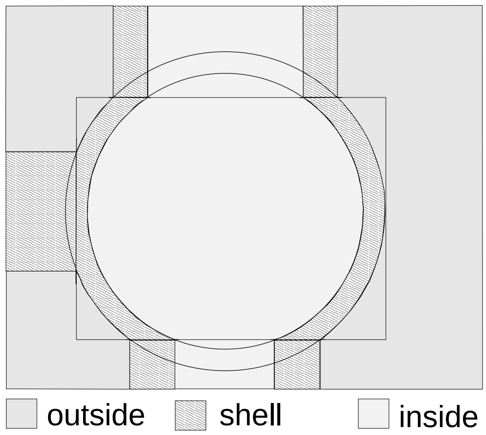


Figure 2.2: Three regions for query points

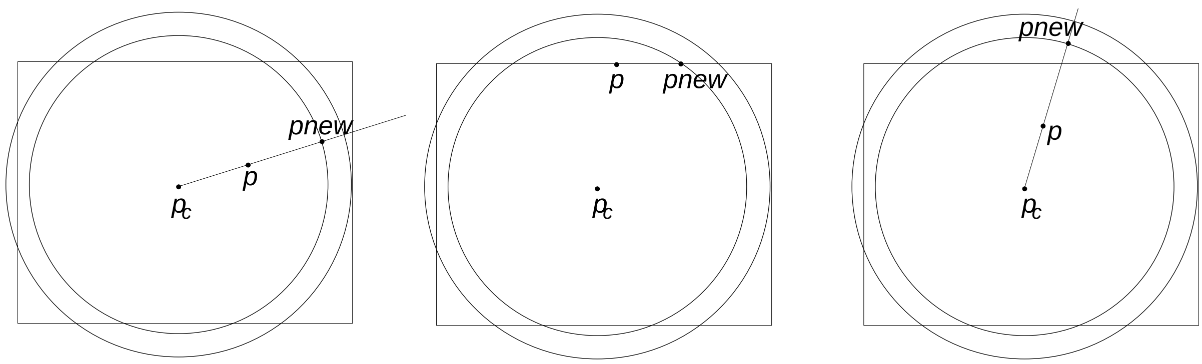


Figure 2.3: (a) normal case (b) p is on the border (c) *pnew* is outside of the box

- p is on the border of the box (see Figure 2.3(b)): Here, for some or for all dimensions i $pnew_i$ equals Le_i or Ri_i. That means those dimensions are not free for optimization. We compute their contribution to the distance and consider this part of distance as a constant term (LR) which is not directly used for finding *pnew*.

- *pnew* is outside of the box (see Figure 2.3(c)): *pnew* is projected on the box and the algorithm is started again ($nextPoint(project(pnew), rad, p_c, \mathsf{Le}, \mathsf{Ri})$). In this way, case (c) is transformed to case (b).

The projected query point p can lie within the outside region. Since the projected point itself is not always the best starting point to find LB we compute the next BALL-point s and project it on the box. After then, we apply our *nextPoint* algorithm with the radius of the outer ball: $nextPoint(project(s), R, p_c, \mathsf{Le}, \mathsf{Ri})$ (see Figure 2.5).

LB(n) for the inside and the outside case is computed as distance between the query point q and the point resulting from invocation of function *nextPoint*.

Function nextPoint(point: p; double: rad: point: p_c, Le, Ri)
store all indices i of which $p_i = \mathsf{Le}_i$ or $p_i = \mathsf{Re}_i$ into array lr
$LR := \sum_{j=1}^{lr.length} (p_{lr_j} - p_{c_{lr_j}})^2$
$dist := ||p - p_c||_2$
$t = \sqrt{\frac{rad^2 - LR}{dist^2 - LR}}$
point $pnew := p$
For each index $i \notin lr$
$pnew_i := p_{c_i} + (p_i - p_{c_i}) * t$
If $pnew \in box$(Le, Ri)
then return $pnew$
else return $nextPoint(project(pnew), rad, p_c,$ Le, Ri$)$

Figure 2.4: Algorithm $nextPoint(p, rad, p_c,$ Le, Ri$)$

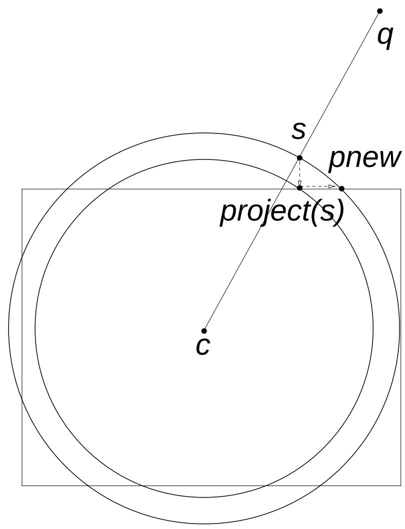


Figure 2.5: Query point in the outside region

Chapter 3

Cluster Approximation by Simplexes

In this section, we provide a formal framework for general concave cluster geometries. We discuss the alignment of data points in high-dimensional space and introduce the *regular simplex* as a suitable geometrical shape. Subsequently, we apply the regular simplex approach to the R-Shell cluster geometry.

Before we start our investigations regarding the layout of points in high-dimensional space, we introduce both – our *concave* and *convex* cluster geometry proposal used for the rest of this paper. The concave cluster geometry is already determined by the *R-Shell* cluster introduced in Chapter 2. To permit a reasonable comparability between concave and convex clusters in terms of retrieval performance, we argue that the convex cluster shape must be based on this (similar) concave cluster shape.

The convex shape we propose is not new but was already introduced in the *SR-tree* [KS97]. We re-use this geometry since it is similar to our R-Shell cluster and, thus, provides an established counterpart for performance comparisons.

The *concave* R-Shell cluster comprises the intersection of one outer (BALL) and one inner (ball) hyperball with a bounding box (box). Both hyperballs have a joint center point p_c. This shape is concave due to the fact, that the inner hyperball excludes a region which is enclosed by the outer hyperball. Furthermore, if the inner ball is large enough, the intersection of these three primitives (BALL, ball, and box) separates the cluster region into truly disjoint areas.

The *convex* R-Shell cluster comprises the intersection of one hyperball (BALL) with a bounding box (box). In contrast to the concave R-Shell cluster, there is no “inner” hyperball in this case. Thus, this cluster geometry is convex.

Subsequently, we abbreviatorily use the term “concave (convex) cluster” instead of “concave (convex) R-Shell cluster”.

3.1 Radii of Concave Hypersphere

Both, our convex and concave cluster geometries are based on high-dimensional balls enclosing a certain volume in the vector space. To obtain good cluster utilization, we need to size these hyperballs appropriately. That is, we have to determine the radii of these hyperballs for a given point set. As seen in the *Branch and Bound* algorithm (Figure 1.1), the cost of nearest neighbor search can be minimized, if the lower bound (LB) of the cluster distance to a query point is very similar to the distance of the actual nearest data point in that cluster to the query point. To achieve this, the shape of a cluster should closely follow the actual alignment of points in high-dimensional space.

In $\mathbb{R}^d$, there are at most $d+1$ points which are equally distant to each other. These points are geometrically aligned in shapes, called *regular simplexes*[1]. In $\mathbb{R}^1$ this shape is simply a line connecting two points, in $\mathbb{R}^2$ it is a triangle with equally long edges, and in $\mathbb{R}^3$ it is a regular tetrahedron. In higher dimensional space, we also obtain regular shapes, namely high-dimensional regular hyperpolyhedrons. These shapes can be inscribed into hyperballs in a way, that every vertice of the simplex touches the surface of the hyperball. These vertices are in fact the points to be clustered. This means, we are searching those data points which are distributed in a way that they are very near to a simplex. By doing this, we benefit from the fact that in high dimensions the data points are similarly distant to each other.

In Figure 3.1 three equally distant points p_0, p_1, and p_2 build a simplex in $\mathbb{R}^2$. The point distance (edge length) is L, and the radius of the hyperball is ρ. From

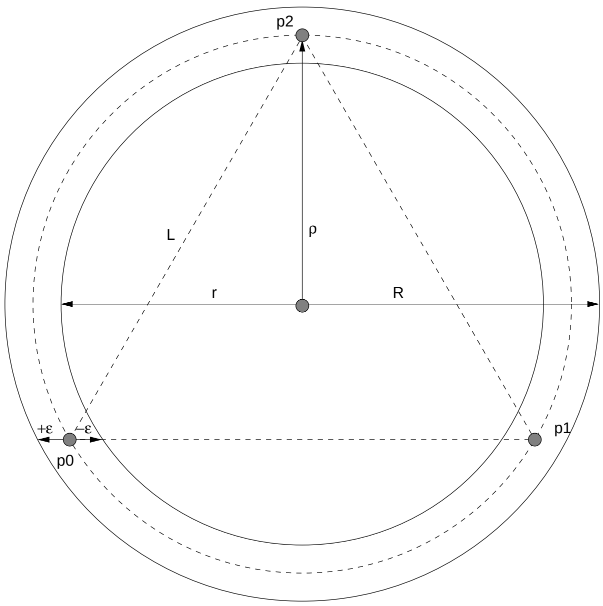


Figure 3.1: Simplex Inscribed into a Minimal Hyperball in $\mathbb{R}^2$

[1]Subsequently, we will abbreviatorily use the term *simplex*.

[Sch01] we know that the expecting value for point distances $E(X)_{\|p_i-p_j\|_2}$ grows with the dimensionality d[2]. As a starting point, it is a good idea to set the edge length L of our simplexes to that distance. However, high-dimensional points will not exactly take the distance of $E(X)$ from each other, but rather scatter around this value. This is specified by the standard deviation $\sigma_{\|p_i-p_j\|_2}$. It is a constant and does, therefore, not depend on the dimensionality[3]. *Chebyshevs Inequality* provides a rough estimation for the probability distribution of point distances L:

$$P(|L - E(X)| > \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2}$$

In Figure 3.1 this variation is labeled by the identifier ε. We cannot directly add or subtract ε from the radius ρ but rather have to manipulate L which is the actual point distance. In this way, we obtain one outer and one inner hyperball with radii R and r, respectively. These hyperballs are the basic geometry for our concave clustering approach. Please note that for convex clusters, the inner hyperball is not part of the cluster shape.

Computation of the radii R and r from dimensionality d, edge length L and some ε in high dimension is not that trivial as it is for low dimensionality. In the following, we present a straightforward approach to compute these radii based on a "normalized simplex". That is, the vertices of the simplex are in fact scaled unit vectors.

It is a simplifying approach to consider that simplex in $\mathbb{R}^{d+1}$ rather than in $\mathbb{R}^d$. In doing so, we can determine $d+1$ vertices of the simplex $p_0, \ldots, p_d$ very straightforwardly by:

$$\begin{aligned} p_0 &= (x, 0, 0, \ldots, 0) \\ p_1 &= (0, x, 0, \ldots, 0) \\ &\cdots \\ p_d &= (0, 0, \ldots, 0, x) \end{aligned}$$

Since all these vertices have the distance x from the origin, obviously we obtain

$$L = \sqrt{2}x$$

as the edge length of the simplex. The center point p_c of the simplex is then:

$$p_c = (\frac{x}{d+1}, \ldots, \frac{x}{d+1})$$

There is a simple proof for this. Assuming that p_c takes the coordinates $(y, \ldots, y)$, where y is unknown, we can state:

$$\|p_i - p_c\|_2^2 = (x-y)^2 + dy^2 \rightarrow \min$$

[2]Subsequently we will abbreviatorily use the term $E(X)$ instead of $E(X)_{\|p_i-p_j\|_2}$.

[3]For the rest of this paper, we will simply use the term σ here.

This means that the derivation of this term (to y) must be zero:

$$\begin{aligned}[(x-y)^2 + dy^2]' &= 0 \\ 2(d+1)y - 2x &= 0\end{aligned}$$

We can now conclude that $y = \frac{x}{d+1}$. Precisely, we also have to check if the second derivation (again, to y) is positive:

$$[2(d+1)y - 2x]' = 2d + 2$$

Obviously, the term $2d+2$ always yields a positive result. Thus, the distance $||p_i - p_c||_2$ is minimal.

Now, we can easily compute the radius ρ of the inscribed hyperball:

$$\begin{aligned}\rho &= ||p_i - p_c|| \\ \rho^2 &= \left(x - \frac{x}{d+1}\right)^2 + d\left(\frac{x}{d+1}\right)^2 \\ &= x^2 - 2\frac{x^2}{(d+1)} + \frac{x^2}{(d+1)^2} + d\frac{x^2}{(d+1)^2} \\ &= (d+1)\frac{x^2}{(d+1)^2} + x^2 - 2\frac{x^2}{(d+1)} \\ &= \frac{-x^2}{d+1} + x^2 \\ &= \frac{x^2 d}{d+1} \\ &= L^2\frac{d}{2d+2} \\ \rho &= L\sqrt{\frac{d}{2d+2}}\end{aligned}$$

In a second step, we need to compute the radii R and r out of the constant ε. That is, the edge length L is lengthened/shortened by ε to compute the radii R and r:

$$\begin{aligned}R &= (L+\varepsilon)\sqrt{\frac{d}{2d+2}} \\ r &= (L-\varepsilon)\sqrt{\frac{d}{2d+2}}\end{aligned}$$

In this way, we have an outer and an inner hyperball with their radii R and r.

3.2 Convex vs. Concave Cluster Shapes

Considering our concave and convex cluster proposal, we can make some statements about the nearest neighbor search performance. By means of an illustration, we compare queries against two clusters (concave and convex), which enclose the same set of data points. Thus, they have the same center point p_c, the same radius R of the outer hyperball, and the same enclosing box. We can now define the concave and convex clusters as:

$$\begin{aligned} \mathsf{concave} &= \mathsf{box} \cap (\mathsf{BALL} \setminus \mathsf{ball}) \\ \mathsf{convex} &= \mathsf{box} \cap \mathsf{BALL} \end{aligned}$$

Obviously, the convex cluster comprises the complete region of the concave cluster. That is, each possible point x of the concave cluster must be contained in the convex cluster, as well:

$$\forall x \in \mathsf{box} \cap (\mathsf{BALL} \setminus \mathsf{ball}) \rightarrow x \in \mathsf{box} \cap \mathsf{BALL}$$

Therefore, the concave and convex clusters are related as follows:

$$\mathsf{concave} \subseteq \mathsf{convex}$$

As stated in Chapter 2 the *Lower Bound Cluster Distance* is formalized as follows:

$$\mathrm{LB}(q) = \min\{\|x - q\|_2 | x \in \text{cluster}\}$$

The cluster distance for query points inside the cluster region is zero for all cluster shapes. However, the cluster distance for query points lying within regions which belong to the convex cluster but not inside the concave cluster will be greater than zero for the concave cluster. We can immediately conclude that

$$\mathrm{LB}_{\text{concave}}(q) \geq \mathrm{LB}_{\text{convex}}(q)$$

holds. Ideally, the region not covered by the concave cluster does not contain data points at all. Therefore, a non-zero cluster distance better approximates the nearest neighbor distance. Therefore, we expect concave clusters containing the same set of data points to perform better than convex clusters (c.f. Figure 3.2). For best search performance, the radius r of the inner hyperball should be as large as possible. As we pointed out in Section 3.1, the size (i.e. the radius) of the enclosing outer hyperball may be computed from the distance of the data points and a deviation from that distance.

In the following, we compare the volume of convex and concave clusters. Since query points are uniformly distributed, this volume directly corresponds to the probability of query points to lie inside the cluster. For an exact computation of the cluster volume, we need to consider the intersection of the outer (and inner, if applicable) hyperball with the enclosing box. However, this is still an open mathematical problem in high dimensions [SZ90, SS91, Sau94]. Therefore, we restrict our considerations on the volume of the hyperballs.

The volume of a hyperball is:

$$V_{\text{ball}} = r^d \frac{\pi^{\frac{d}{2}}}{\Gamma(1 + \frac{d}{2})}$$

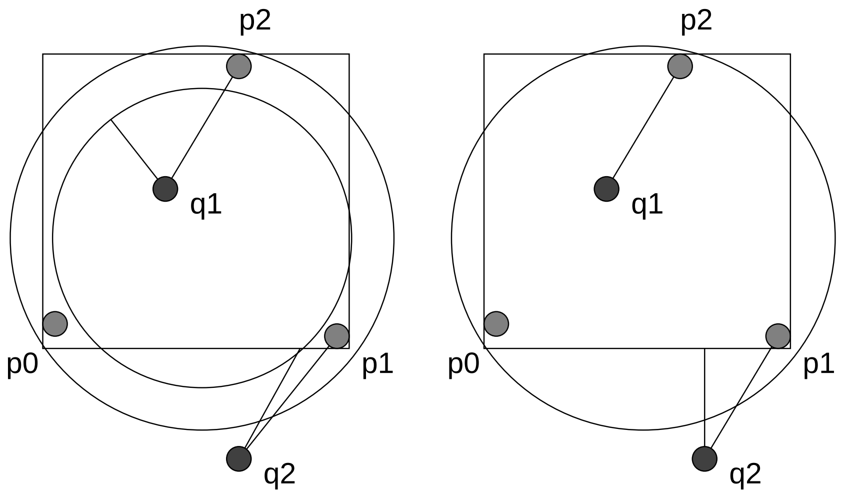


Figure 3.2: Concave and Convex R-Shell Cluster Shapes

Please note, that Γ denotes the *Gamma* function[4], r is the radius, and d denotes the dimensionality. The volume of the convex cluster is now defined as follows:

$$V_{\mathsf{BALL}} = R^d \frac{\pi^{\frac{d}{2}}}{\Gamma(1+\frac{d}{2})}$$

The concave cluster takes the following volume:

$$V_{\mathsf{BALL}\backslash\mathsf{ball}} = (R^d - r^d) \frac{\pi^{\frac{d}{2}}}{\Gamma(1+\frac{d}{2})}$$

This leaves R and r open to be determined. From [Sch01] we know, that the expecting value for point distances approaches the following value:

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

Furthermore, the standard deviation[5] is given by a constant:

$$\sigma_{\|p_i - p_j\|_2} \rightarrow_p c \approx 0.24$$

These are the parameters, we use for the computation of the cluster volumes. As a simple approach we take $E_{\|p_i - p_j\|_2}$ as the edge length L of the point simplex (see Section 3.1). For the deviation ε from that distance we use different values. In Table 3.1 we computed V_{BALL} and $V_{\mathsf{BALL}\backslash\mathsf{ball}}$ (depicted as $V_{\mathsf{BALL}}/V_{\mathsf{BALL}\backslash\mathsf{ball}}$ in the cells) for some dimensionalities d. Actually, we have to set ε to the smallest possible value ensuring the maximal cluster

[4] $\Gamma(x) = \lim_{n\rightarrow\infty} \frac{n!n^{x-1}}{x(x+1)\ldots(x+n-1)} = \int_0^\infty e^{-t}t^{x-1}dt$

[5] In the following, we will abbreviatorily use σ.

utilization. That is, $d+1$ data points fit in a cluster. Since we already need three points for the geometric representation of the cluster (center point p_c of hyperballs, lower and upper bound in each dimension for computation of the enclosing box), there got to be at least four data points in one cluster.

			ε		
d	$E_{\|p_i-p_j\|_2}$	ρ	10^{-1}	10^{-3}	10^{-5}
10	1.291	0.870	0.931/0.502	0.639/0.004	0.637/0.00005
20	1.826	1.260	4.498/2.994	2.635/0.029	2.620/0.0003
30	2.236	1.555	24.2/17.9	12.6/0.167	12.5/0.002
50	2.887	2.021	778/640	332/5.71	329/0.057
100	4.082	2.872	5345113/4883684	1601793/38759	1582490/388

Table 3.1: Estimated Cluster Volumes and Radii (all Values Rounded)

For high dimensionalities, the concave and convex volumes exceed the volume of the vector space ($= (1-0)^d = 1$). Of course, no cluster may be larger than the volume of the vector space. Therefore, each hyperball is restricted by the borders of the vector space and the supplementary minimal enclosing box. Due to the so far impossible computation of sphere-rectangle-intersections in high dimensions, we cannot determine the probability of point queries lying inside the cluster geometry. However, Table 3.1 provides a good indication onto the ratio between convex and concave cluster volumes.

Obviously, the dimensionality is the dominating parameter for cluster volumes. The deviation ε from the standard radius ρ which marks the "thickness" of the concave cluster sphere cannot entirely prevent the increase of the volume in high dimensions. Moreover, the smaller the value of ε becomes the better the data points have to approach to an ideal simplex alignment. This makes it even more complicated, to find complete simplexes from a given number of data points.

Chapter 4

Clustering Algorithms

In this chapter, we describe clustering algorithms for our R-Shell cluster approach. Generally, a clustering algorithm groups data points into certain clusters. Subsequently, we propose two straightforward algorithms for clustering:

- Parameterized Cluster Geometry
- Small Enclosing/Inscribed Hyperballs

Before we explain these algorithms in detail, we state some general requirements for these clustering algorithms.

4.1 General Cluster Requirements

Since the basic cluster shape – the R-Shell cluster – is predefined, the following requirements concern the assignment of appropriate data points to suitably sized clusters to improve the performance of the nearest neighbor retrieval process (the *Branch-and-Bound* algorithm).

The Branch-and-Bound algorithm needs to compute the minimal distance (*lower bound*, LB(q)) of a query point q to the cluster border by means of the cluster geometry. Clearly, it must not require a larger effort to retrieve the cluster geometry than the contained data points themselves. For our cluster proposal, the retrieval costs of the cluster geometry (*geometry retrieval costs*) are composed of as follows:

- Bounding Box (box): one interval in each dimension ($2d$ float values)
- Outer Hyperball (BALL): one center point p_c and one radius R ($d + 1$ float values)
- Inner Hyperball (concave clusters only, ball): shares the same center point, but one additional radius r (1 float value)

This results in $3d + 2$ ($3d + 1$) float values for the retrieval of each concave (convex) cluster geometry for dimensionality d. One data point requires exactly d float values for its storage. Accordingly, the costs which accumulate for the sequential retrieval of all data points that lie in the cluster (*point retrieval costs*) are made up as the product of the number of data points in the cluster and the dimensionality.

In this context, we define the term *efficient cluster*. That is, a cluster is called *efficient* whenever the sequential point retrieval costs lie above the geometry retrieval costs. For the R-Shell cluster geometry, efficient clusters must comprise at least four data points (for $d > 2$). Obviously, it makes no sense to retrieve clusters whose cluster retrieval costs exceed its point retrieval costs. However, in the following we propose clustering algorithms that cannot completely avoid the formation of non-efficient clusters. Therefore, non-efficient clusters require a special treatment in the retrieval process. That is, we do not retrieve the geometry data of these clusters but rather treat the contained data points as so called "*outliers*". This means, we retrieve them sequentially and, therefore, obtain linear costs for them. It often *does* make sense to maintain their assignment to a non-efficient cluster. In fact, this cluster may actually be complemented by further points at a later time. When new data points are inserted this cluster may still become efficient.

To obtain a good approximation of the real nearest neighbor distance for an arbitrary query point q out of the cluster geometry, we must have an appropriate distribution of points in the cluster. By the intersection of the enclosing box with the hyperballs, up to 2^d isolated regions in that cluster may emerge. That is, the vertices of the enclosing box are separated from each other. Although we cannot find 2^d points for large dimensions d, we try to occupy as many corners as possible. Choosing the simplex as a model for point patterns in high-dimensional space, we have only $d + 1$ points which we can assign to a cluster.

A further aspect that directly influences the approximation of the nearest neighbor distance is the radius R of the (outer) enclosing hyperball. If this radius is small, the data points contained in the emerging hyperball are closely located to each other. Our clustering algorithm is therefore targeted on assembling cluster shapes having a small or even minimal radius R for a given set of data points. In contrast to the *SR-tree* which uses a sort of weighted center point, we chose the *Minimal Enclosing Hyperball* approach to compute the outer hyperball. Although this approach turns out to be more costly than simpler algorithms, it provides the smallest radius by definition. We expect it to pay off in terms of an optimized retrieval performance.

Many publications (e.g. [BKSS90]) discuss the problem of cluster *overlap* as one main cause for deteriorated retrieval performance in high dimensions. [BKK96] provide useful definitions for overlap in R-trees. Even though we have to distinguish *volume overlap* and *point overlap* (in [BKK96] labeled with *overlap* and *weighted overlap*), we cannot confirm the statement of less overlap being necessarily a crucial cause for improved retrieval performance. Subsequently, we even present experiments where we can compare zero-volume concave clusters with non-zero-volume convex clusters, both assembling the same number of data points. In contrast to convex clusters, totally concave clusters with a volume of zero can obviously not produce volume overlap and, therefore point overlap becomes very unlikely. Without anticipating detailed results, our measurements indicate that overlap is no main cause for the resulting retrieval performance.

4.2 Parameterized Cluster Geometry

To choose a simple clustering algorithm, it is a very straightforward approach to find clusters by predefining certain cluster parameters. A set of data points that fits within a cluster of these predefined parameters can be assembled in that cluster.

The R-Shell cluster geometry is parameterized by one center point p_c, the radii R and r forming two hyperballs. Furthermore, we have an enclosing box aligned in parallel to the Cartesian coordinate axis. The box can, thus, be described by an interval $[\mathsf{Le}_i, \mathsf{Re}_i]$ for each dimension i. The main idea of our approach is to find out appropriate values to the parameters to obtain good cluster shapes.

Embracing all cluster parameters simultaneously is difficult. First of all, we can skip a consideration of the center point p_c. Since all data points are uniformly distributed, it does not make sense to restrict or preset p_c in a certain way.

In addition, we omit a parameterization of the enclosing box. The shape of the enclosing box does have an influence on resulting factors, like cluster volume and number of disjoint cluster areas. However, it is not possible to consider the enclosing box jointly with the hyperballs in a linear optimization problem. Moreover, we cannot compute the volume out of the intersection of the hyperballs and the enclosing box. Therefore, we also cannot make sensible assumptions about the number of points that fit within an R-Shell cluster having a certain enclosing box (for example specified by a dedicated edge length).

The only remaining parameters to be specified are the inner (r) and outer (R) radius which specify BALL and ball which are part of the cluster geometry.

The main idea of *parameterized cluster geometry* is to set a limit for a certain weight function w. In doing so, we obtain a *weight constraint* the cluster parameters must comply to. The clustering algorithm is bound to ensure that this weight constraint is fulfilled for all emerging clusters. The weight function may either return one of the radii (R or r) itself or any directly derived value. In this paper we chose the following weight functions:

- $w_1 : R$ and
- $w_2 : V_{\mathsf{BALL}}$ (*convex*) or $V_{\mathsf{BALL}} - V_{\mathsf{ball}}$ (*concave*)

denoting the outer radius (w_1) and the volume (w_2) of the outer hyperball (convex) or the volume of the hypershell (concave). Another conceivable weight function might be the term $R - r$ that specifies the "thickness" of the hypersphere.

The first weight function w_1 is meant to restrict the radius of the outer hyperball to a certain limit. The second weight function w_2 attempts to optimize the cluster volume. In this case, we compute the volume of the outer hyperball for convex clusters and the volume of the hypersphere (area between outer and inner hyperball) for concave clusters. Due to the fact, that computing V_{BALL} takes no other variable but R into account, w_1 and w_2 can be mapped into one another for convex R-Shell clusters.

Even if our weight functions may be easy to compute, it is difficult to *optimize* the cluster parameters to minimize the weight function, i.e. fulfill the weight constraint. That is, we have to determine the radii R and r for a given number of data points that the weight function returns a small (or even minimum) value. We propose two ways for this parameter computation:

- $R^2 - r^2 \to \min$
- Smallest Enclosing Hyperball

Actually, we would want to minimize the weight function directly. Unfortunately, this yields non-linear optimization problems for all considered weight functions.

4.2.1 Optimization by $R^2 - r^2 \to \min$

One way to tackle this problem, is a simple transformation to a different, yet similar weight function. In particular, w_2 contains the term $R^d - r^d$ which only differs in the exponent of the radii. Even if optimizations of w_1 and w_2 are non-linear, we found out that an objective function

$$R^2 - r^2 \to \min$$

may be transformed into a linear optimization problem:

Given a set of n points p_i in d dimensions with $p_i = (p_{i_1}, p_{i_2}, \ldots, p_{i_d})$ this raises the following side conditions:

$$\begin{aligned} R^2 &\geq (p_{c_1} - p_{i_1})^2 + \ldots + (p_{c_d} - p_{i_d})^2 \\ r^2 &\leq (p_{c_1} - p_{i_1})^2 + \ldots + (p_{c_d} - p_{i_d})^2 \end{aligned}$$

Please note, that we obtain $2n$ side conditions. The center point to the resulting hyperball is labeled by p_c. To avoid the consideration of non-linear optimization problems, we optimize

$$R^2 - r^2 \to \min$$

subject to the above side conditions. However, we must put the following substitution in place. Be

$$\begin{aligned} A &= p_{c_1}^2 + \ldots + p_{c_d}^2 - R^2 \\ B &= p_{c_1}^2 + \ldots + p_{c_d}^2 - r^2 \end{aligned}$$

two auxiliary variables. By using these variables, the optimization problem can be expressed as

$$B - A \rightarrow \min$$

with respect to the following side conditions:

$$\begin{aligned} 2(p_{c_1}p_{i_1} + \ldots + p_{c_d}p_{i_d}) &\geq A + (p_{i_1}^2 + \ldots + p_{i_d}^2) \\ 2(p_{c_1}p_{i_1} + \ldots + p_{c_d}p_{i_d}) &\leq B + (p_{i_1}^2 + \ldots + p_{i_d}^2) \end{aligned}$$

Furthermore we can state that every coordinate of the center point p_c must lie in the interval $[0, 1]$:

$$\begin{aligned} c_i &\leq 1 \\ c_i &\geq 0 \end{aligned}$$

This linear optimization problem can be solved. We expect it to return cluster parameters that also produce good results for our weight function.

Linear optimization problems tend to have an exponential worst case runtime behavior. The running time of this algorithm depends on the number ($2n$) of side conditions which is – in turn – depending on the number of data points. However, we can assess that the number of data points n will have an upper bound ($\approx d + 1$).

4.2.2 Smallest Enclosing Hyperball

The second method to achieve good results from the weight functions is to compute the smallest enclosing hyperball. That is, we compute the smallest enclosing hyperball BALL out of a given set of points. Additionally, we build the largest "foreclosing" hyperball ball with the same center point p_c. In this way, we attempt to optimize the weight functions. In case the used weight function returns a value below a certain threshold, we can assemble these set of data points into one cluster.

To compute the smallest enclosing hyperball, we use the randomized *move-to-front* method from [Wel91]. This algorithm is fast in choosing a subset of data points that lie on the surface of a hyperball. However, the actual problem of computing a d-dimensional hyperball out of $n + 1 \leq d + 1$ border points remains to be solved.

We propose an algorithm which computes the smallest enclosing hyperball out of a set of $n + 1$ border points $p_0, \ldots, p_n$. That is, we claim that every point p_i is on the surface of a hyperball, i.e. has the same distance (radius) to a center point p_c.

The task to compute the center point for a set of $n+1$ points in d dimensions, where $n = d$ holds is trivial. This case can generally be expressed by a set of d linear equations, where the coordinates of p_c is the solution.

However, we may also face the situation where less than $d + 1$ border points are available. Obviously, this case is not uniquely solvable since it does not provide one well-defined center point. Rather, we would want to obtain the center point for the smallest hyperball defined by $p_0, \ldots, p_n$.

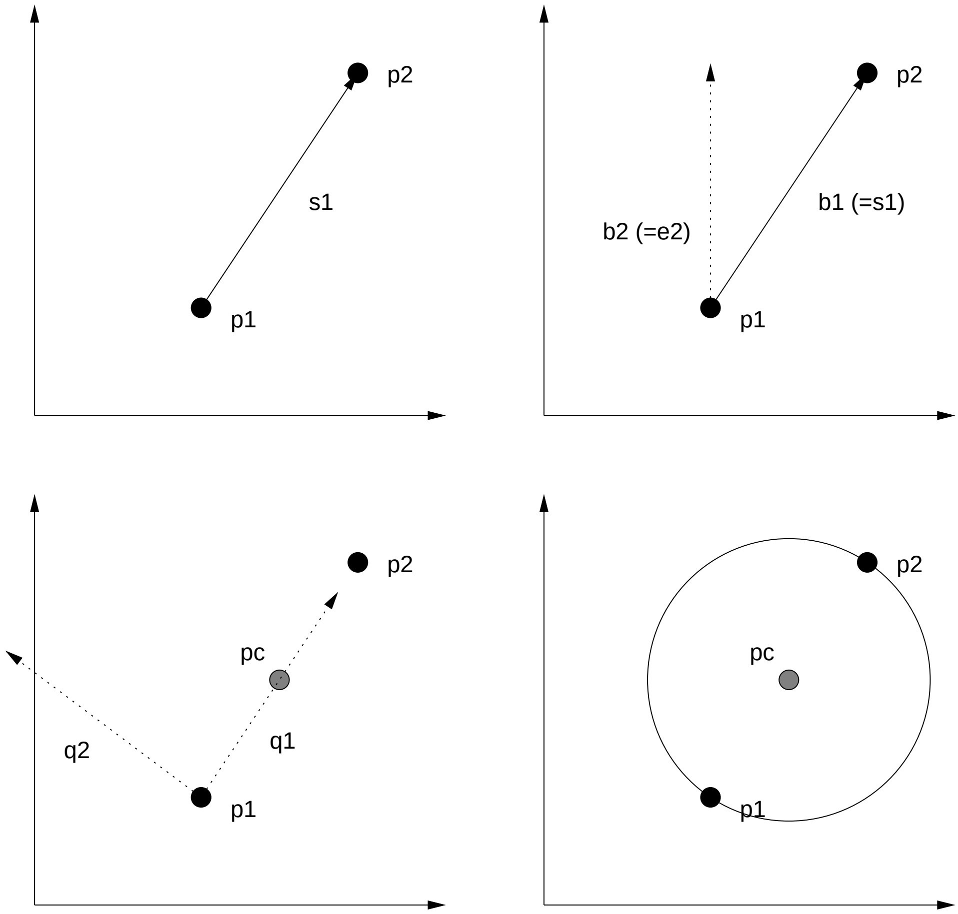


Figure 4.1: Smallest Hyperball out of $n + 1$ Border Points (with $n < d$)

Figure 4.1 depicts our algorithm by means of an example in $d = 2$ dimensions. In the example, we have two border points ($n = 2$). The smallest hyperball through these border points can be obtained from a transformed coordinate system. We have to transform these border points into a $n - 1$-dimensional coordinate system. The transformation has to maintain distances (and angles) between the border points. In the lower dimensional coordinate system, an explicit center point of these border points is determinable. Finally, a backward coordinate system transformation has to be applied to obtain a valid center point p_c in d dimensions. In our example, this yields the minimum hyperball with p_1 and p_2 as its border points.

We abstain from a detailed presentation of the algorithm at this place. However, a formal description of this algorithm is sketched in Appendix A.

4.2.3 Comparison of Optimization Methodologies and Weight Functions

To compare the introduced optimization methodologies ($R^2 - r^2 \rightarrow \min$ and *Smallest Enclosing Hyperball*) and weight functions (w_1 and w_2) we ran a first performance measurement. We chose to run these experiments for dimensionality $d = 15$ in an range of $N = 1000, \ldots, 10000$ data points. To find first suitable thresholds for the given weight functions, we applied the simplex model for point alignments along with the expecting value for point distances and computed the respective values for the outer radius R and the appropriate volume of the outer hyperball V_{BALL}. We also applied the value obtained for V_{BALL} for the concave case, i.e. setting the threshold of $V_{\mathsf{shell}} = V_{\mathsf{BALL}} - V_{\mathsf{ball}}$ to that value. In Table 4.1 these values are depicted.

d	$E(X)$	R	V_{ball}
10	1.291	0.870	0.636
15	1.581	1.083	1.253
20	1.826	1.260	2.620
25	2.041	1.415	5.660

Table 4.1: Expecting Value for Point Distances, Radius and Volume of Hyperspheres

Please note, that these values do in no way return optimal results. It is up to further experiments to find near-optimal thresholds for the weight functions. However, the values obtained from Table 4.1 are sufficient for our comparison purposes. For our experiments we have chosen to use the *Smallest Enclosing Hyperball* approach for convex clusters since it inherently optimizes both weight functions in this case. For the computation of the center point p_c of concave clusters, we applied the $R^2 - r^2 \rightarrow \min$ optimization. More measurements (not depicted here) indicate that this method performs better then *Smallest Enclosing Hyperball* here. That is due to the fact, that the objective function $R^2 - r^2$ is fairly similar to the computation of hypershell volumes V_{shell}. In Figure 4.2 we sketched the results obtained from the experiment.

Since the threshold value of V_{BALL} is directly computed out of the respective value for R (see Table 4.1) both weight functions are equivalent for the convex case. Therefore, we merged them in Figure 4.2. The results illustrate that concave and convex clusters with optimized radii R perform noticeably better than concave clusters with minimal hypershell volume. This supports our statement that an optimized (i.e. minimal) cluster volume does not outweigh the deteriorated influence of larger radii. The radius directly corresponds to the distance between data points which grows as the radius increases. Therefore, we already have to draw one vital conclusion: Concave R-shell clusters only seem to pay out, if their outer radius R is not larger than for convex clusters.

Unfortunately, both, the *Smallest Enclosing Hyperball* and the $R^2 - r^2 \rightarrow \min$ approach are very costly. In a simple incremental algorithm, for every inserted point p_{new}

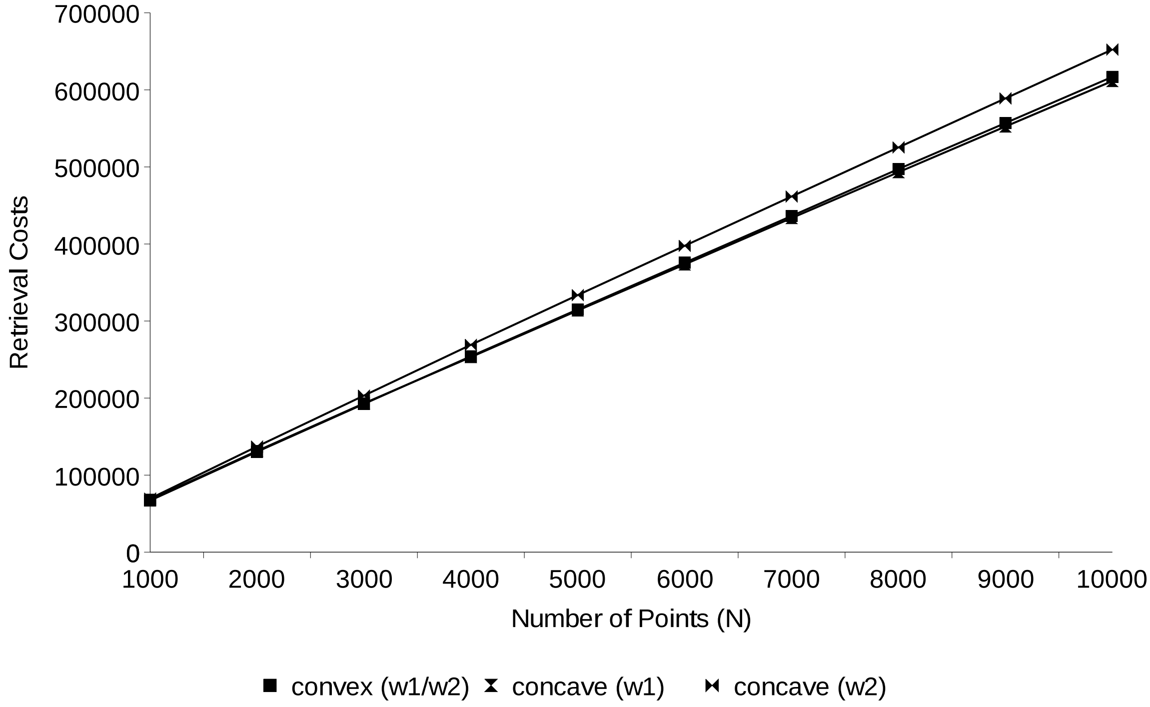


Figure 4.2: Retrieval Performance Experiment for Dimensionality 15

one has to compute the cluster parameters by means of one of these approaches for each existing cluster and compare the results from the selected weight function. In this way, the best suited cluster is determined. If its parameter (radius, ball/shell volume) lies beneath the predefined threshold value, p_{new} is inserted into that cluster. Otherwise, p_{new} is the "seed point" for a newly created cluster. Although we expect this procedure to yield good results it turns out to be too expensive in practice. Therefore, we propose a modified algorithm which takes advantage of some heuristics we will use within this algorithm. The algorithm *FindFastCluster* is sketched in Figure 4.3. The first heuristic is based on the fact, that a new data point may already lie in an area enclosed by one existing cluster. Due to the fact that our weight functions only involve the hyperballs but not the enclosing box, we can even lessen this requirement to the check if the new point lies within the shell (i.e. the region between inner and outer hyperball).

Secondly, we can check every cluster if the new point fits within the limits for its parameters without changing the center point of the hyperballs. That is, the radii R and r must be computed in consideration of p_{new}. This is a non-expensive operation which can be performed very fast.

If none of these checks yields a positive result, i.e. the new point does not fit within one of them, we have to examine them more thoroughly. This means, a new center point for the hyperballs has to be determined by means of one of our optimization methods. Since these operations run faster if fewer points are involved, we order the list of existing

```
function FindFastCluster(array clusters, point p)
    randomize array clusters
    for each cluster c_i ∈ clusters do
        if c_i.inShell(p) then return c_i
    od
    for each cluster c_i ∈ clusters do
        c_i.addTemporalPoint(p)
        if c_i.getWeight() ≤ w_max and c_i.getRadius() ≤ r_max then return c_i
        c_i.removeTemporalPoint(p)
    od
    sort array clusters ascendingly by size
    for each cluster c_i ∈ clusters do
        compute new center point
        if c_i.getWeight() ≤ w_max and c_i.getRadius() ≤ r_max then return c_i
    od
    create new cluster with p
end
```

Figure 4.3: Algorithm *FindFastCluster*

clusters ascendingly by their size. We expect the probability for a point to "fit" within a cluster to be greater for clusters with fewer data points inside.

Only if this test fails again, a new cluster is created to assimilate the point p_{new}.

4.3 Small Enclosing/Inscribed Hyperball

The *Parameterized Cluster Geometry* is a geometry-centric approach. That is, it does not make any assumptions about the average/minimal number of data points in a cluster. The parameterization values (*here:* radius R or ball/shell volume $V_{\mathsf{BALL}}/V_{\mathsf{shell}}$) must be provided and may only be obtained from statistical observations or predicted from models like the simplex model for data point alignments.

In contrast, we propose another approach which is targeted on providing convex and concave clusters which:

- have the same (*similar*) average outer radius R
- assemble the same number of data points.

In contrast to a parameterization, this methodology takes the value for the outer radius R of each cluster from enclosing hyperballs that assemble a certain number of near points. Therefore, it is a data point centric approach, which is based on the whole set P of existing data points with $|P| = N$.

The main objective of the proposed methodology is to allow a good comparability of our convex and concave cluster approaches. That is, we generate convex clusters which "behave" very well with respect to the retrieval performance. Based on the properties of these convex clusters (average radius R and number of clustered points) we build the concave clusters.

4.3.1 Convex Clustering

Typical clustering algorithms for convex cluster shapes are based on the assembly of data points that are very close to each other. In case of our approach, we assemble a certain (preset) number k of data points in each cluster.

The main difficulty is the selection of appropriate subsets $S_i \subset P$ of data points ($|S_i| = k$). The subsets S_i are, in fact, the sets of data points assembled in the convex clusters. In our definition, the subset of data points which yields the smallest enclosing hyperball is best suited for one convex cluster. Obviously, checking all possible subsets S_i out of the set P cannot be computed efficiently (exponential complexity). Therefore, we randomly choose one starting point $p_0 \in P$ for each subset S_i. The algorithm depicted in Figure 4.4 sketches the methodology of choosing a suitable subset S_i out of a set P of data points. Initially, P comprises N data points but shrinks, as more and more clusters are created. The minimal enclosing ball around the data points of S_i is the basis for a convex cluster. The algorithm in Figure 4.4 is applied until the size of the data point

```
choose a random starting point p_0 ∈ P
S = {p_0}
P = P \ p_0
let p_c be the center point of the smallest enclosing hyperball out of S
while |S| < k do
    begin
    find the nearest neighbor point p_nn ∈ P to p_c
    S = S ∪ p_nn
    P = P \ p_nn
    let p_c be the new center point of the smallest enclosing hyperball out of S
    end
```

Figure 4.4: Convex Cluster Assembly by Small Enclosing Hyperball

set P shrinks to zero. That is, a number of subsets S_i is created successively, each one comprising k data points. Please note, that the "last" subset may possibly contain less than k points.

Furthermore, we compute an array of average outer radii that stores the radii of the average smallest enclosing balls assembling $1, \ldots k$ points:

$$\{\mathfrak{R}_{\text{convex}_2}, \ldots, \mathfrak{R}_{\text{convex}_k}\}$$

These values $\mathfrak{R}_{\text{convex}_j}$ are computed during the assembly of data points subsets as depicted in Figure 4.4. Thus, we obtain a sort of histogram based on average outer radii for convex clusters. Our experiments revealed that the values $\mathfrak{R}_{\mathrm{convex}_j}$ depend on the size N of the whole set of data points P. In fact, if there are more data points we can build our subsets S_i from, the average radii shrink. In Figure 4.5 we show the average radii $\mathfrak{R}_{\mathrm{convex}_2}, \ldots, \mathfrak{R}_{\mathrm{convex}_{16}}$ $(R2, \ldots, R16)$ of convex clusters for dimensionality $d = 15$ and $N = 1000, \ldots, 10000$.

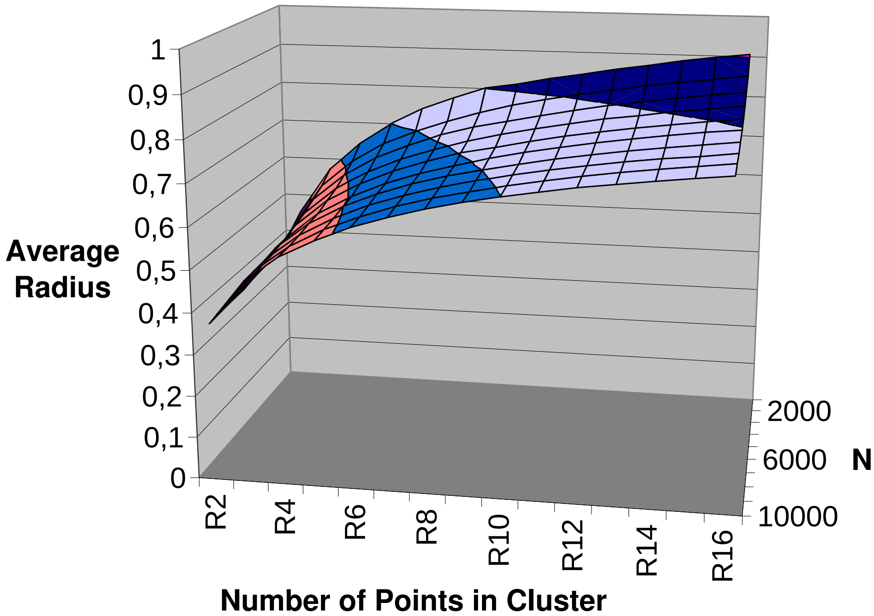


Figure 4.5: Average Outer Radii for Convex Clusters ($d = 15$)

4.3.2 Concave Clustering

As stated before, our intention is to generate concave clusters which comprise the same number of data points k. In addition, we aim at creating concave clusters with the same (or at least very similar) average outer radii. In this way, we obtain the average radius parameter out of the actual data point distribution. However, an appropriate number k of data points for each cluster remains to be specified.

Before we present our concave clustering algorithm, we briefly introduce a further restriction to our concave cluster shape: the *totally concave* R-Shell cluster. The totally concave R-Shell cluster follows the idea of concave clusters even more strictly. Instead of a hypershell which is formed by the space between the two hyperballs BALL and ball, the totally concave R-Shell cluster requires *all* of its data points to be aligned on the outer hyperball BALL (i.e. BALL = ball). That is, the distance from each data point p_i to the center point p_c exactly equals the outer radius R. This approach has two major advantages:

- Firstly, as the inner radius r takes the same value as R, we do not need the inner hyperball (ball) any longer. We do not have to store the inner radius r (which takes the same value as R). Moreover, this reduces the cluster cost for concave clusters to the same value as for convex clusters ($3d + 1$ float values).

- Secondly, our experiments with ordinary R-Shell clusters revealed, that the advantage of not totally concave clusters with $R > r$ seems to vanish in high-dimensional space. The main difference between convex and concave R-Shell clusters is that the inner hyperball ball penetrates the boundary face of the enclosing box in the concave case. Query points which are perpendicularly located "above" these penetration regions benefit from this fact in terms of a lower approximation error. However, the inner hyperball seems to penetrate the enclosing box in smaller regions for the high-dimensional case. We name this effect *vanishing concaveness* here.

  In Figure 4.6 this problem is illustrated. In all cases (convex, concave, totally concave), the same set of data points is captured in the cluster. Furthermore, we have the same enclosing box (box) and the same outer hyperball (BALL). In addition, we have an inner hyperball (ball) in the concave case. Please note, that the inner hyperball is not as large as it could be in the illustration. This is due to the fact, that vanishing concaveness usually occurs in higher dimensional space, but not for dimensionalities as low as two. Obviously, this inner hyperball does *not* penetrate the enclosing box.

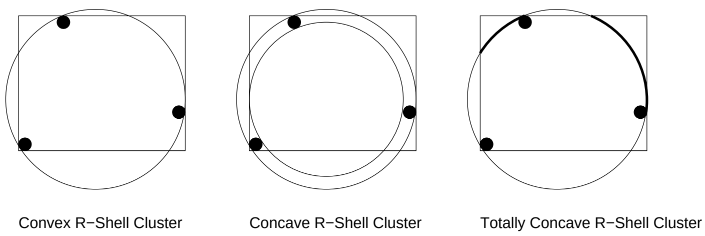


Figure 4.6: Convex, Concave and Totally Concave R-Shell Cluster

However, if we *inscribe* the outer hyperball BALL through the data points, we obtain a totally concave cluster whose volume is in fact zero. Therefore, we base the totally concave R-Shell cluster on minimal *inscribed* hyperballs whereas we use the minimal *enclosing* hyperball approach for convex R-Shell clusters.

To compute the (minimal) inscribed hyperball, we have to distinguish two cases. In the first case, the number k of border points $p_1, \ldots, p_k$ equals $d + 1$. In this case, the inscribed hyperball with its center point p_c is uniquely defined for d dimensions and can be computed by the following system of linear equations:

$$\begin{aligned} \|p_1 - p_c\|_2 &= \|p_2 - p_c\|_2 \\ \|p_1 - p_c\|_2 &= \|p_3 - p_c\|_2 \\ &\dots \\ \|p_1 - p_c\|_2 &= \|p_{d+1} - p_c\|_2 \end{aligned}$$

The only restriction is, that these points must be linearly independent:

$$\lambda_1 p_1 + \lambda_2 p_3 + \ldots + \lambda_{d+1} p_{d+1} = \mathbf{o} \Rightarrow \lambda_1 = \lambda_2 = \ldots = \lambda_{d+1} = 0$$

As a result we obtain a center point p_c along with the radius $R = \|p_1 - p_c\|_2$.

However, if $k < d + 1$ holds, the problem becomes more complicated. We actually have to determine the *minimal* inscribed hyperball out of less than $d + 1$ border points. This problem occurs in the computation of the minimal enclosing hyperball, as well. Therefore, we refer to Appendix A for a detailed description on how to tackle this problem.

The main problem in total concave clustering consists of the selection of appropriate data points that the average radius of the emerging inscribed hyperball is very close to the average convex radius (for clusters with the same number of data points). In fact, we assume that both radii are very similar in high dimensions. We argue that the simplex model proves this thesis. We abstain from a proof of the convergence of minimal enclosing and inscribed hyperballs in high dimensions at this place. Instead, we refer to Appendix B where we sketched a plain proof.

Besides this analytic investigation, we support our assumption by some experiments. Therefore, we built minimal enclosing and inscribed hyperballs (i) from random points, and (ii) from our convex (see Figure 4.4) and concave clustering (see Figure 4.7) approaches. We measured the average ratio of minimal enclosing radius and inscribed radius and sketched the results in Table 4.2. The results subsumed in Table 4.2 reveal

Dimensionality	Random Points (i)	Convex/Concave Clustering (ii)
5	0.6291	0.9202
10	0.4782	0.9512
15	0.4110	0.9621
20	0.3779	0.9738
25	0.3454	0.9786

Table 4.2: Ratio of Smallest Enclosing and Inscribed Hyperball Radii

two major phenomena. First of all, it supports our thesis that minimal enclosing and inscribed hyperballs converge in high dimensions (using appropriate clustering algorithms).

Even for dimensionalities as low as 25 these radii are very similar for particular convex and concave clustering algorithms.

Secondly, the utilization of randomly generated, yet uniformly distributed data points as "input variables" for minimal enclosing and inscribed ball computations reveals that the radii of these hyperballs noticeably differ in our measurements. Furthermore, this ratio deteriorates with rising dimensionality. This means that randomly generated points are not mandatorily aligned as regular simplexes which only seems to hold for selected data points. The positive consequence of this outcome is that distances between data points *do* differ even in high-dimensional space. This, in turn, motivates nearest neighbor search for high dimensions.

Next, we describe our concave clustering algorithm. This algorithm is based on the compliance of average concave radii to the average convex radii profile. We intend to apply this algorithm for dimensionalities, where there is still a difference between minimal enclosing an inscribed hyperballs radii for equally sized data point sets (see Table 4.2). Therefore, our priority is to create totally concave clusters which obey the convex radii profile. That, in turn, means that we cannot comprise the same number of data points in some cases. Therefore, we must ensure that even concave clusters with less than k data points match the convex cluster radii profile. The proposed algorithm is sketched in Figure 4.7. The concave clustering algorithm iteratively scans the data point set P for

```
S = ∅
while |S| < k do
    begin
    find the point p_min ∈ P returning the smallest inscribed hyperball out of S ∪ p_min
    r_min = radius of this minimal hyperball
    r_tmp = (r_min + ℜconcave_{|S|+1}) / (1 + N/(|S|+1))
    if r_tmp ≤ ℜconvex_{|S|+1} then
        begin
        S = S ∪ p_min
        P = P \ p_min
        end else break loop
    end
update ℜconcave_{|S|}
```

Figure 4.7: Totally Concave Cluster Assembly by Small Inscribed Hyperball

the point $p_{\min}$ that results in the smallest inscribed hyperball alongside with the existing points in the subset S. Before the data point $p_{\min}$ can finally be inserted into the subset S, the emerging radius $r_{\min}$ is utilized to compute a preliminary average concave radius r_{tmp} for the appropriate size of S. Subsequently, this radius r_{tmp} is checked against the convex radius profile $\mathfrak{R}_{\text{convex}}$ for this subset size. If this check succeeds, i.e. the preliminary average concave radius lies below the average convex radius, the procedure continues. That is, further data points may be added to the subset S. However, if the check fails, the subset of this particular size ($|S| < k$) forms the basis for a concave cluster.

Chapter 5

Experiments

In this section, we will present major results from various experiments we conducted. In particular, we examine the introduced clustering approaches in terms of retrieval performance. Following, we discuss the results from our experiments and compare both approaches.

In this paper, we use the term "retrieval costs" equivalent to the data volume (I/O costs) retrieved from the database. This means, we leave additional factors like CPU time out of consideration. We argue, that the I/O effort makes up the very major part of the time elapsing for a nearest neighbor retrieval.

Remember, the data volume is composed by the cluster retrieval costs and point retrieval costs which take different proportions depending on dimensionality d and number of data points N.

We measure the data volume independently from any concrete database implementation on a particularly equipped workstation. Especially, we intend to exclude external effects like database, hardware or network optimizations. Instead, we directly count the data volume transferred from the database. That is, we consider every real number (e.g. data point coordinate, cluster radii, enclosing box boundaries) as a single precision floating point number (named as FLOAT) which takes four byte (IEEE 754). Therefore, we obtain a certain data volume (measured in bytes) for every nearest neighbor retrieval.

To exclude accident variations, we computed the mean out of 1000 single measurements for all results presented in diagrams throughout this paper. In some diagrams the retrieval costs measured for our clustering approaches are supplemented by the retrieval costs accumulating for a sequential scan. That is, the database is sequentially scanned for the nearest neighbor point. Obviously, all N data points have to be retrieved for this case, thus resulting in:

$$\text{cost}_{\text{sequential scan}} = N \cdot d \cdot \text{sizeof}(\mathsf{FLOAT})$$

The predicate *sizeof* returns the size (in bytes) of the given data type. Please note, that sizeof(FLOAT) returns four byte and can be adapted for higher precision floating point numbers[1].

5.1 Results

For the parameterized cluster geometry, previous experiments revealed that the weight function w_1 (outer radius R) alongside the *Smallest Enclosing Hyperball* approach for convex R-Shell clusters and the $R^2 - r^2 \to \min$ optimization for concave R-Shell clusters yields the best results. Therefore, we consider these two parameters as fixed here. However, appropriate limits for w_1 still remain to be settled.

As the expecting value for point distances grows with dimensionality, we know from our simplex model (Section 3.1) that the average radius is expected to grow, as well. Furthermore, the radii histogram captured in Figure 4.5 (Section 4.3.1) revealed that the radius is also influenced by the number of data points N. Therefore, we decided to figure out the optimal performance of the parameterized cluster geometry approach by experimentally examining several threshold values (limits) for the weight function w_1.

Since w_1 reflects the outer radius R we may again use the expecting value and our simplex model as a starting point for the intervals we chose. We run our experiments for three dimensionalities: $d = 10, 15, 20$. In Table 5.1 we depicted expecting value $E(X)$ and radius ρ emerging from these dimensions. Based on these values, we decided to

Dimensionality d	Expecting Value $E(X)$	Radius ρ
10	1.2910	0.8704
15	1.5811	1.0825
20	1.8257	1.2599

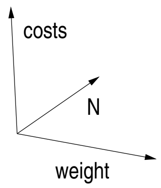
Table 5.1: Expecting Value and Radius for Dimensionalities $d = 10, 15, 20$

run the clustering experiments with the following set of threshold values for the weight function w_1:

$$\begin{aligned}
10 &: \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\} \\
15 &: \{0.6, 0.7, 0.8, 0.9, 1.0, 1.1\} \\
20 &: \{0.8, 0.9, 1.0, 1.1, 1.2, 1.3\}
\end{aligned}$$

In Figure 5.2 the emerging retrieval costs are depicted in dependence on the number of data points and the threshold value for the weight function w_1. We may gather the optimal threshold value for the weight function w_1 for any number of data points and dimensionality (in the respective scaling) these diagrams. Moreover, we can compare the optimal retrieval performance, yielded for convex and concave clusters.

[1] For example: sizeof(DOUBLE) = 8 Bytes



The results obtained from the *Parameterized Cluster Geometry* approach are twofold. For lower dimensions ($d = 10, 15$) convex clusters outperform concave clusters in terms of retrieval costs. However, for higher dimensionalities ($d = 20$), this effect reverses.

We have chosen the spectra of weight threshold parameters for $d = 10, 15, 20$ to obtain clustering results from (i) almost as many clusters as data points (N) to (ii) very few (≈ 1) cluster(s). In the first case, obviously very few of the considered clusters are efficient. That is, most of their included data points are treated as "outliers" which – in turn – makes up retrieval costs similar to a sequential scan.

If there are very few or even just one cluster(s) in the second case, the *Branch-and-Bound* algorithm unlikely prunes any of these clusters. This due to the fact that their approximation error is very large. In consequence, we have to retrieve all data points *and* the geometry data of the (few) existing clusters. The optimal case, i.e. minimal overall retrieval costs results from a clustering with a weight threshold parameter somewhere between these extremes.

In Figure 5.1 we assembled the best (i.e. lowest) retrieval costs for each measures dimensionality ($\Box$ convex, $\bowtie$ concave). The worse retrieval performance of concave clusters for lower dimensionalities is due to the fact that we use a different optimization method ($R^2 - r^2 \to \min$) here. This results in a larger number of concave clusters which is an important factor for the overall retrieval costs. In particular, cluster retrieval costs make up a large proportion w.r.t. the overall retrieval costs in lower dimensions. In higher dimensionalities the point retrieval costs make up the larger proportion. Moreover, concave clusters are more "expensive" than convex clusters (one additional FLOAT value).

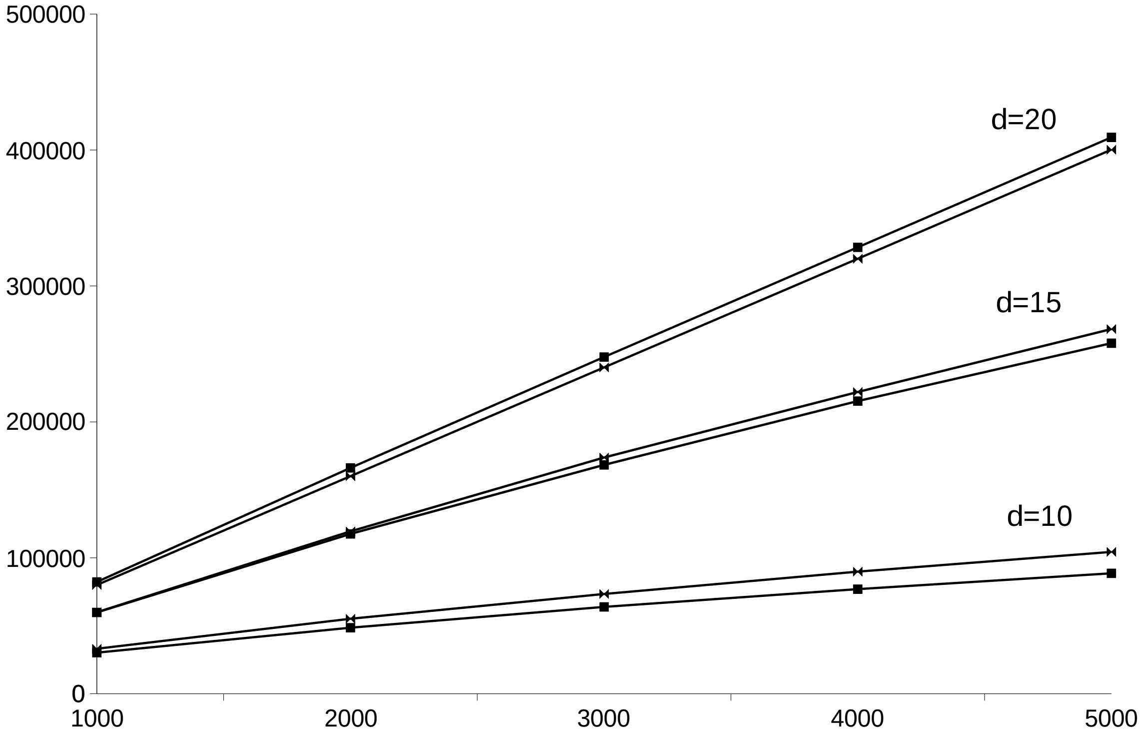


Figure 5.1: Comparison of Best Convex and Concave Retrieval Costs

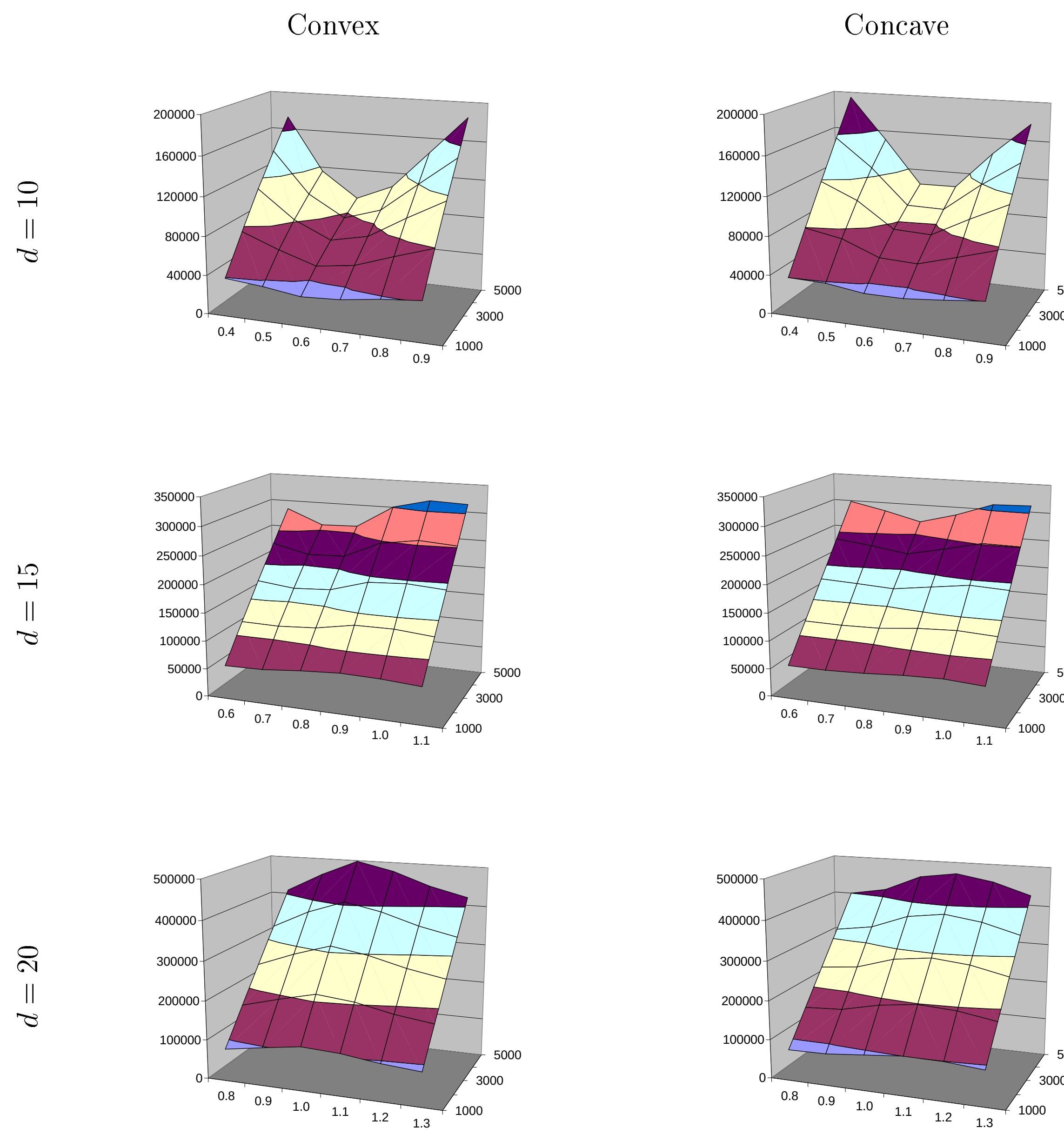


Figure 5.2: Weight Profile for *Parameterized Cluster Geometry*

For the small inscribed hyperball approach, all experiments were run with convex and *totally* concave R-Shell clusters. We utilized the clustering algorithms from Figure 4.4 and 4.7. Again, we run our experiments in three dimensions (d=10, 15, 20) and for data point sets of different sizes ($N = 1000, 2000, \ldots, 10000$).

Besides the retrieval effort for totally concave clusters ($\bowtie$), we also measured convex clusters ($\diamond$) and sequential scan ($\square$) costs. In Figure 5.3 these costs are depicted in dependence on the number of data points N for dimensionality $d = 10$. Please note, that

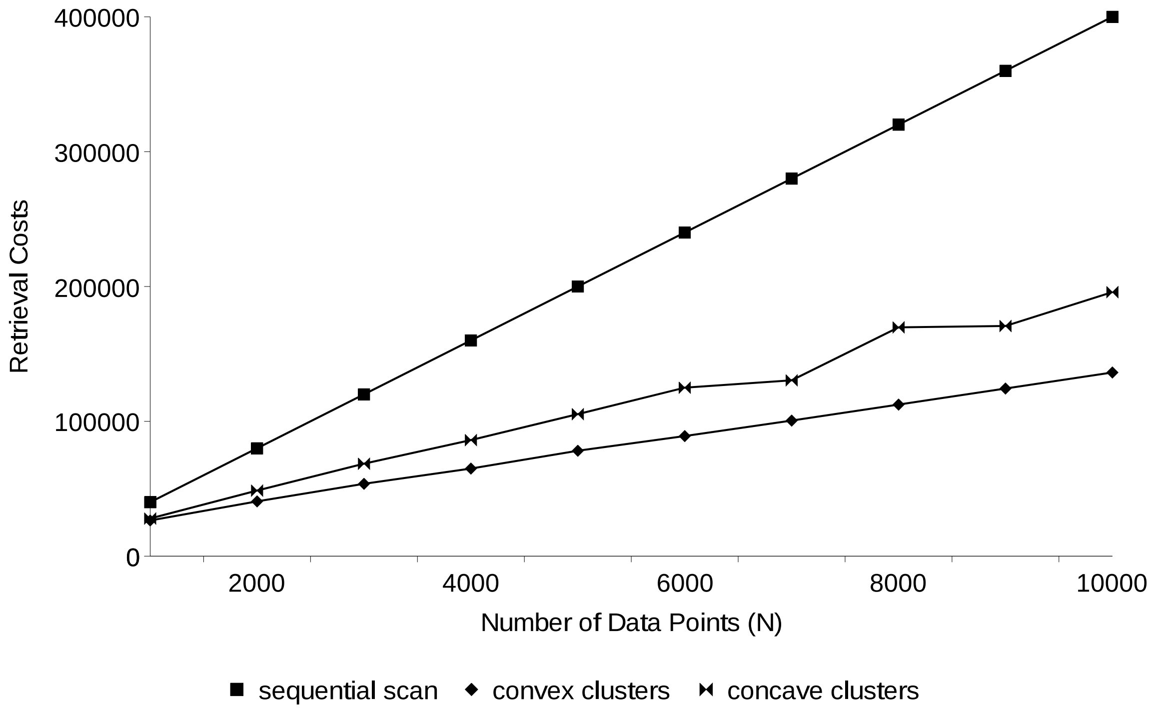


Figure 5.3: Small Inscribed Hyperball: Convex and Totally Concave Clustering ($d = 10$)

the pictured costs are the overall retrieval costs, composed of point retrieval costs and cluster retrieval costs. It turns out that for a dimensionality as low as 10 convex R-Shell clusters perform noticeably better than both, totally concave clusters and sequential scan. The totally concave R-Shell cluster still performs better than sequential scan. However, there are no advantages of concave cluster geometries in this particular dimension. Both retrieval costs (convex and totally concave clustering) grow sublinearly.

In Figure 5.4 the results from our retrieval performance measurements in dimensionality $d = 15$ are shown. The remaining parameters were adopted from the previous experiment. However, the results obtained from this experiments differ. That is, retrieval performance of totally concave clustering is almost the same as of convex clustering. In fact, it is even slightly better. Again, the retrieval costs grow sublinearly. In particular, totally concave clusters perform better than sequential scan for $N > 2000$ and convex clustering is better for $N > 3000$ data points.

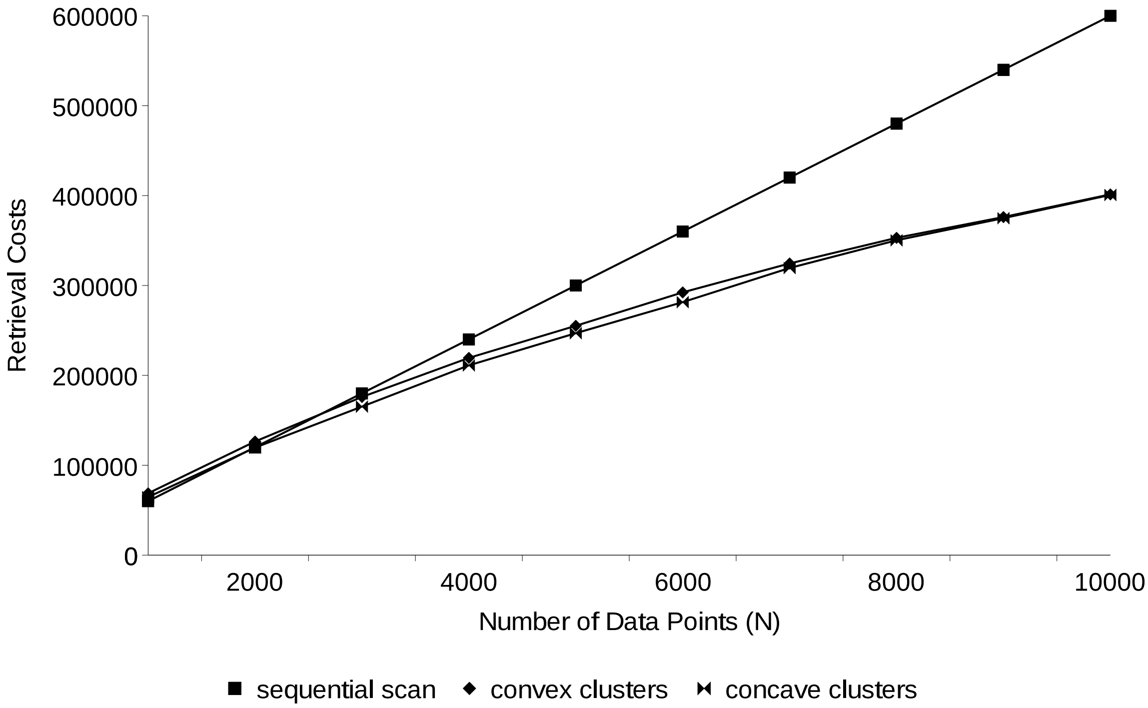


Figure 5.4: Small Inscribed Hyperball: Convex and Totally Concave Clustering ($d = 15$)

In Figure 5.5 this experiment is repeated for our third investigated dimensionality $d = 20$. For the first time, totally concave clusters perform noticeably better than convex clusters. For a growing number of data points (N) this improvement even increases. Totally concave clustering exposes a sublinear growth of retrieval costs with respect to N. Although both approaches perform worse than sequential scan for less than $N < 10000$ data points, the concave cluster geometry shows an remarkable impact. For more data points we expect both, totally concave *and* convex cluster geometries to perform better than sequential scan.

5.2 Discussion

The results obtained from both – the Minimal Enclosing/Inscribed Hyperball *and* the Parameterized Cluster Geometry approach – are twofold. The varying ratios of convex and totally concave retrieval costs in different dimensions unveil the impact of an actual alignment of data points as regular simplexes. As Table 4.2 illustrates, minimal enclosing and inscribed hyperballs become more similar sized with rising dimensions. As (total) concave retrieval costs lie noticeably beneath convex retrieval costs for $d = 20$, our results confirm this claim.

For lower dimensions, the minimal enclosing hyperball is still smaller than the mini-

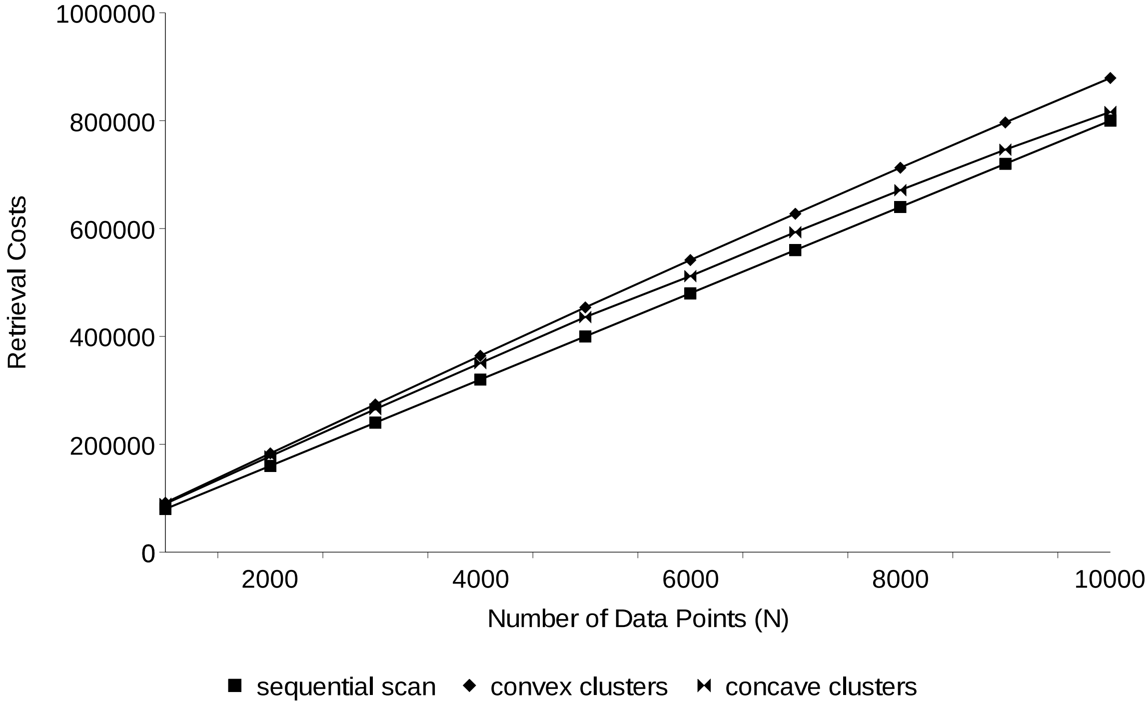
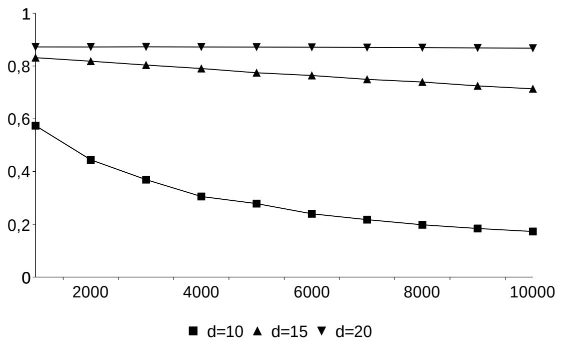


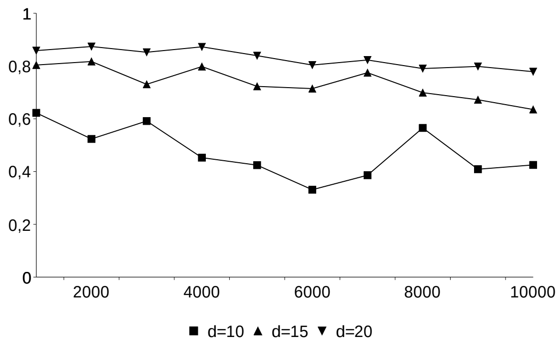
Figure 5.5: Small Inscribed Hyperball: Convex and Totally Concave Clustering ($d = 20$)

mal inscribed hyperball. As we suggested earlier, the emerging different radii outweigh the advantage of concaveness in lower dimensions. This also stresses our thesis, that cluster volume is no critical parameter in terms of retrieval performance. However, the radius directly reflects distances of points inside the cluster which is a lot more crucial for the retrieval costs.

Another reason for superior retrieval performance of convex clusters in lower dimensions is the larger proportion of cluster retrieval costs in comparison to point retrieval costs w.r.t. the *overall* retrieval costs. Our totally concave clustering algorithm (see Figure 4.7) ensures the same average radius for inscribed and enclosing hyperballs. Unfortunately, a larger number of concave clusters emerge for the same number of data points. In Figure 5.6 the proportion of point retrieval costs with respect to the overall retrieval costs is depicted for convex and totally concave clusters in different dimensionalities ($d = 10, 15, 20$). For a dimensionality as low as ten, the convex clusters unveil a lower proportion of point retrieval costs. Moreover, this proportion even reduces for a larger number of data points N. For totally concave clusters, there is only a slight reduction of this proportion which heavily varies as the number of points increases. At first sight, this result is surprising. Although the number of concave clusters exceeds the number of convex clusters for a given number of data points N the point retrieval performance is worse for the concave case. A more detailed perception of this phenomenon explains these results. As distances are comparatively small in low dimensions, their



Convex Clusters



Totally Concave Clusters

Figure 5.6: Proportion of Point Retrieval Costs in Different Dimensionalities

relative variation is large. Therefore, it does make sense to build convex clusters as they group near data points which have similar distances to a query point. On the other hand, concave clusters contain less data points for the same radius. Therefore, more clusters have to be inspected which deteriorates the point retrieval performance.

For higher dimensionalities, the effect of "concaveness" becomes visible. As data points are aligned in regular simplexes, a similar number of convex and concave clusters emerges. The lower proportion of point retrieval costs for concave clusters results in a superior overall retrieval performance. Nevertheless, as the diagram (Figure 5.6) reveals, both – the convex and the (totally) concave – cluster geometry approaches still suffer from the "curse of dimensionality" which is expressed by the large proportion of point retrieval costs in *both* cases.

The *Parameterized Cluster Geometry* and *Minimal Enclosing/Inscribed Hyperball* approaches differ from each other therein that the Parameterized Cluster Geometry approach allows a dynamic insertion of new data points. In contrast, the Minimal Enclosing/Inscribed Hyperball works on a predefined set of data points. Obviously, there are applications for both approaches.

In terms of retrieval performance, both approaches yield similar results. This is due to the fact that the concave optimization methodology ($R^2 - r^2 \to \min$) aims at producing concave clusters with a small (i.e. minimal) volume. On the other hand the weight function w_1 ensures an equal outer radius R of convex and concave clusters. Thereby, the concave parameterized cluster geometry follows the same guidelines as the minimal inscribed hyperball approach (*totally* concave R-Shell clusters).

Chapter 6

Summary, Conclusions, and Outlook

In this final chapter we briefly summarize the major results of our research presented in this paper. We draw some conclusions which arise from our results. Finally, we present an outlook on further research and future work.

6.1 Summary

In *Chapter 1* we presented a statistical analysis (expecting value and standard deviation) of Euclidean distances between uniformly distributed data points. Moreover, we developed cost formulas and error estimations which help us to estimate the effort for nearest neighbor retrieval based on the Branch-and-Bound algorithm.

Motivated by the results from Chapter 1, in the subsequent *Chapter 2*, we presented a concave cluster geometry proposal – the R-Shell cluster. We formally described its geometry and presented an algorithm to compute the lower bound distance from a query point to the next (virtual) cluster point.

In *Chapter 3*, we aimed at providing a formal framework for concave cluster geometries. In particular, we presented the *regular simplex* as an alignment for (uniformly distributed) high-dimensional data points. Furthermore we introduced the convex R-Shell cluster as the counterpart of the concave R-Shell cluster. In addition, we developed the simplex model for point alignment based on normed regular simplexes. Based on this model, we derived a simple formula to compute the radius of a hyperball inscribed into a regular simplex.

In the second part of this chapter, we compared convex and concave cluster geometries and exhibited the advantage of concave cluster geometry in terms of retrieval performance.

Chapter 4 deals with clustering algorithms. That is, we propose two approaches for assignment of particular data points to certain clusters. First, we stated important general cluster requirements which heavily influence the nearest neighbor retrieval performance.

Subsequently, we introduced the "Parameterized Cluster Geometry" clustering approach which is based on the conformance to predeterminated weight functions. In this context, we elaborately discussed two exemplary weight functions and two appropriate optimization methodologies: the Smallest Enclosing Ball approach and the $R^2 - r^2 \rightarrow$ min-optimization. A comparison of the introduced weight functions and optimization methodologies concludes the consideration of this clustering approach.

Following, we presented the second clustering approach: the "Small Inscribed Hyperball". This approach is targeted on the assembly of convex and concave R-Shell clusters which have identical geometrical parameters and comprise the same number of data points. We presented a convex clustering algorithm which is based on the location of smallest enclosing hyperballs around data point sets of a particular size.

For the concave clustering algorithm, we previously introduced the totally concave R-Shell cluster based on inscribed hyperballs. We motivated it with the effect of vanishing concaveness. We argued, that identical geometrical parameters follow from our simplex model for point alignments. To evidence this assumption, we both presented a a formal proof and experimentally verified this effect. Subsequently, we introduced a totally concave clustering algorithm.

Chapter 5 contains an experimental verification of our concave clustering approach in terms of nearest neighbor retrieval performance. Both clustering algorithms were run for different dimensionalities and sizes of the overall point set. We measured the IO-effort of the retrieval process which is superior to unprecise values like CPU or disc time on particular computers. Besides the results depicted in various diagrams, we thoroughly discussed the outcome of the experiments.

6.2 Conclusions and Outlook

The major outcome of this paper is the suitability of concave cluster geometries for efficient nearest neighbor retrieval. With respect to this particular concave cluster geometry (the R-Shell cluster) we have to name some restrictions. First of all, the advantage of concave clusters turns out (i) in high dimensions and (ii) if the inner hyperball penetrates the enclosing box, i.e. we have no "vanishing concaveness" effect.

Secondly, the radius of the outer hyperball is the dominant factor for both convex and concave R-Shell clusters. Therefore, the clustering algorithm must be targeted on the creation of equally sized outer radii in both cases. For lower dimensions, this, in turn means that we obtain a higher number of clusters for the concave case. As a consequence, totally concave R-Shell clusters perform worse for lower dimensions and better for higher dimensions than convex clusters. Also, we optimized our clustering approaches with regard to retrieval performance. Nevertheless, in future work, more effort has to be put into the investigation of more efficient clustering algorithms.

A major difference between the introduced clustering methodologies is, that "Parameterized Cluster Geometry" starts with an empty set of points, allowing a later insertion of data points. In contrast, the "Small Enclosing/Inscribed Hyperball" approach, presupposes an existing set of data points. In this way, "Parameterized Cluster Geometry" behaves more dynamically. In particular, the totally concave clustering should be applied to the "Parameterized Cluster Geometry" approach.

In theory, each conceivable convex cluster geometry could be broaden to a resulting concave cluster geometry. However, a major outcome of the simplex model for data point alignments is, that near data points are mandatorily aligned as regular simplexes. This fact has two consequences.

First of all, to retain the same number of data points in the cluster, it only makes sense to exclude these areas from the convex cluster shape which may not contain a vertice (i.e. data point) of the simplex.

Secondly, the probability for a data point to be located inside the simplex is very low. Due to our assumption that even query points are uniformly distributed, we must accept the fact that virtually no query point will lie inside the simplex. Instead, almost all query points will be located outside the convex geometry of the simplex. Therefore, mutating a convex cluster geometry to become concave is only reasonable if the approximation error the the emerging concave cluster is remarkably lower than before. In our R-Shell cluster proposal this can be induced by a large inner hyperball which penetrates the enclosing box. Ideally, a concave cluster consists of several truly separated regions in the vector space. These regions should precisely approximate the location of an actual data point. In fact, this is partially founded in the vanishing concaveness effect that we observed for our non-totally concave R-Shell clusters. Future research, should therefore deal with concave clusters which precisely locate small separated regions in the vector space.

The (totally) concave R-Shell cluster should, thus, be regarded as a starting point for the investigation of concave clustering. The idea of concave clusters is not restricted to alteration of established convex cluster geometries to become concave. Rather, the results from R-Shell clusters unveil that an ideal cluster shape should strictly follow the simplex model for point alignments. Therefore, new proposals for concave cluster shapes should be based on this consideration.

Another problem we did not yet consider is the utilization of cluster hierarchies as it is used for all tree based clustering approaches. However, this turns out to be a complicated problem. Concave clusters are meant to precisely locate the comprised data points. In a cluster hierarchy, "non-leaf" clusters should contain other clusters which have a width or diameter and, therefore, span a certain area in the vector space which is not true for a single data point.

In this paper, we restricted all considerations onto uniformly distributed data points. Although we consider this as the more complicated case, "real data" should also be investigated in future work. Nevertheless, only uniformly distributed data points permit an "incorrupt" comparison of clustering methodologies.

Acknowledgment

The authors would like to thank Professor Michiel Smid (formerly University of Magdeburg, Computer Science Department) for his useful remarks on non-linear optimization problems.

Furthermore, we are much obliged to Professor Bernulf Weißbach (University of Magdeburg, Department of Mathematics) for his fruitful discussion on high dimensional sphere/rectangle intersections.

Finally, we also thank the responsive members of de.sci.mathematik. This newsgroup turned out to be a very qualified forum for manifold mathematical problems.

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*, lncs, Vol. 1973, pages 420–434. Springer-Verlag, Berlin, 2001.

[BBKK97] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space. In *Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'97, Tuscon, Arizona, USA*, pages 78–86. ACM Press, New York, 1997.

[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.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 322–331. Atlantic City, NJ, May 1990.

[FK70] K. Fukunaga and W. L. G. Koontz. Representation of random processes using the finite Karhunen-Loève expansion. *Information and Control*, 16(1):85–101, 170.

[FL95] Christos Faloutsos and King-Ip Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 163–174, ACM Press, 1995.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *Computing Surveys*, 30(2):170–231, June 1998.

[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.

[KW78] Joseph B. Kruskal and Myron Wish. *Multidimensional Scaling.* Sage Publications, Beverly Hills, Calif., USA, 1978.

[LJF94] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree – an index structure for high-dimensional data. *The VLDB Journal*, 3(4):517–549, October 1994.

[Lu99] Guojun Lu. *Multimedia Database Management Systems.* Artech House, Boston/London, 1999.

[MKB79] K.V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis.* Academic, 1979.

[NG88] H. Niemann and R. Goppert. An efficient Brand-and-Bound Nearest Neighbour Classifier. *Pattern Recognition Letters*, 7:67–72, 1988.

[Sau94] Peter Sauer. On a Problem by J. M. Wills. *Mathematische Nachrichten*, 162:239–245, 1994.

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

[SS91] G. Schechtman and M. Schmuckenschläger. Another remark on the volume of the intersection of two L_p^n balls. *Lecture Notes in Mathematics, Geometric aspects of functional analysis (1989–90)*, 1469:174–178, 1991.

[SZ90] G. Schechtman and J. Zinn. On the volume of the intersection of two L_p^n balls. *Proceedings of the American Mathematical Society*, 110(1):217–224, 1990.

[Wel91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science, LNCS*, Vol. 555, pages 359–370, 1991.

[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.

Appendix A

Enclosing Hyperball with $n+1$ Border Points

Given a set of $n+1$ border points $P = \{p_0, \ldots, p_n\}$ in d dimensions with $n < d$, first, a set of n spanning vectors is determined:

$$\begin{aligned} s_1 &= p_1 - p_0 \\ &\ldots \\ s_n &= p_n - p_0 \end{aligned}$$

These vectors span an n-dimensional basis. Clearly, we must make sure that these vectors, are linearly independent. That is,

$$\lambda_1 s_1 + \ldots + \lambda_n s_n = \mathfrak{o} \Rightarrow \lambda_1 = \ldots = \lambda_n = 0$$

holds[1]. We must augment this n-dimensional basis to a d-dimensional basis b_i. This is made by the n spanning vectors s_i and $(d-n)$ unit vectors $\mathfrak{e}_i$ (based on the exchange principle by Steinitz). For example:

$$\begin{aligned} b_1 &= s_1 \\ &\ldots \\ b_n &= s_n \\ b_{n+1} &= \mathfrak{e}_{n+1} \\ &\ldots \\ b_d &= \mathfrak{e}_d \end{aligned}$$

Again, we must ensure, that the resulting basis is linearly independent. With the orthonormalization procedure by Erhard Schmidt, we can very easily construct an

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

orthonormal coordinate system with d coordinate axis q_i:

$$
\begin{aligned}
q_1 &= \frac{b_1}{||b_1||_2} \\
q_{k+1} &= \frac{b_{k+1} - \sum_{i=1}^{k} \langle b_{k+1}, q_i \rangle q_i}{||b_{k+1} - \sum_{i=1}^{k} \langle b_{k+1}, q_i \rangle q_i||_2}
\end{aligned}
$$

Next, all boundary points p_i need to be transformed into the new coordinate system. This transformation (rotation) can be done by a multiplication with a $(d \times d)$-matrix M with $M_{(x,y)} = q_{x_y}$. This operation results in a set of $n+1$ transformed boundary points r_i:

$$r_i = p_i \cdot M$$

Due to the fact that all these points r_i lie on one n-dimensional hyperplane, we can reduce their dimensionality to n:

$$l_i = (r_{i_1}, \ldots, r_{i_n})$$

Clearly, all points r_i must have $d-n$ coordinates in common. Following this dimensionality reduction, we can fairly easy compute the center point out of these $n+1$ points l_i in dimensionality n:

$$
\begin{aligned}
||p''_c - l_0||_2 &= ||p''_c - l_1||_2 \\
&\ldots \\
||p''_c - l_0||_2 &= ||p''_c - l_n||_2
\end{aligned}
$$

We obtain the following system of linear equations:

$$
\begin{aligned}
2p''_{c_1}(l_{2_1} - l_{1_1}) + \ldots + 2p''_{c_{n-1}}(l_{2_{n-1}} - l_{1_{n-1}}) &= (l^2_{2_1} - l^2_{1_1}) + \ldots + (l^2_{2_{n-1}} - l^2_{1_{n-1}}) \\
&\ldots \\
2p''_{c_1}(l_{n_1} - l_{1_1}) + \ldots + 2p''_{c_{n-1}}(l_{n_{n-1}} - l_{1_{n-1}}) &= (l^2_{n_1} - l^2_{1_1}) + \ldots + (l^2_{n_{n-1}} - l^2_{1_{n-1}})
\end{aligned}
$$

In the next step the resulting preliminary center point p''_c has to be extended to dimensionality d. The attributes in dimension $n \ldots d$ have to be "filled" with values from some (arbitrary) r_i.

$$p'_c = (p''_{c_1}, \ldots, p''_{c_{n-1}}, r_n, \ldots, r_d)$$

Finally, the center point p'_c is back-transformed to the original (Cartesian) coordinate system[2]:

$$M^T p_c = p'_c$$

The solution to this system of linear equations is the final center point p_c.

[2]Due to the orthonormalization procedure $M^T M = E$ holds.

Appendix B

Enclosing and Inscribed Hyperballs

To proof the equivalence of the smallest *enclosing* and the *inscribed* hyperball in high-dimensional space in terms of radius and number of border points, we reuse our simplex model for point alignments from Section 3.1. We assume, that all given data points are aligned as regular simplexes. However, this assumption only holds for high dimensions.

In this indirect proof we assume some vertices of a simplex to be "inner" (i.e. non-border) points of the minimal enclosing hyperball. We show that this assumption leads to a contradiction.

For a given number k_1 of data points p_i in d dimensions the inscribed hyperball can be constructed as follows. Without loss of generality, we built up our k_1-simplex in $\mathbb{R}^{d+1}$:

$$\begin{array}{rcl} p_0 & = & (x, 0, 0, \ldots, 0) \\ & \ldots & \\ p_i & = & (\underbrace{0, \ldots, 0}_{i}, x, \underbrace{0, \ldots, 0}_{d-i}) \\ & \ldots & \\ p_{k_1-1} & = & (\underbrace{0, \ldots, 0}_{k_1-1}, x, \underbrace{0, \ldots, 0}_{d+1-k_1}) \end{array}$$

Please note, that for $k_1 < d+1$ the emerging simplex is incomplete. For a given distance L between two arbitrary data points p_i, the coordinate x is computed by $x = \frac{L}{\sqrt{2}}$. Therefore, we obtain the following center point p_{c_1} for the inscribed hyperball:

$$p_{c_1} = (\underbrace{x/k_1, \ldots, x/k_1}_{k_1}, \underbrace{0, \ldots, 0}_{d+1-k_1})$$

The first k_1 coordinates of p_{c_1} take the value of x/k_1, whereas the remaining $d+1-k_1$ coordinates take a value of zero (0.0). The resulting radius r_1 (i.e. the distance from p_{c_1} to any p_i) is (see Page 10):

$$r_1 = L\sqrt{\frac{k_1 - 1}{2(k_1 - 1) + 2}} = L\sqrt{\frac{k_1 - 1}{2k_1}}$$

Generally, an enclosing hyperball encloses a certain subset of boundary points and a remaining subset of inner points. If all its data points are boundary points, the minimal enclosing hyperball *equals* the inscribed hyperball.

In our case, we assume a number of $k_2 < k_1$ data points to be boundary points. The remaining $k_1 - k_2$ data points have, thus, to be situated somewhere inside the enclosing hyperball. Again we consider the emerging regular k_2 simplex in $\mathbb{R}^{d+1}$. We obtain the following center point:

$$p_{c_2} = (\underbrace{x/k_2, \ldots, x/k_2}_{k_2}, \underbrace{0, \ldots, 0}_{d+1-k_2})$$

Here, the first k_2 coordinates of p_{c_2} are computed by x/k_2, the remaining $d+1-k_2$ take a value of zero. The resulting radius is:

$$r_2 = L\sqrt{\frac{k_2 - 1}{2(k_2 - 1) + 2}} = L\sqrt{\frac{k_2 - 1}{2k_2}}$$

Obviously, for $k_1 > k_2$ the relation $r_1 > r_2$ holds. Remember, for p_{c_2} being the center point of a minimal enclosing hyperball with radius r_2 all k_1 of its data points p_i to have a distance of:

$$||p_i - p_{c_2}||_2 \leq r_2$$

For the first k_2 (boundary) data points p_i with $i = 0, \ldots, k_2 - 1$ we obtain:

$$||p_i - p_{c_2}||_2 = \sqrt{(k_2 - 1)(\frac{x}{k_2})^2 + (\frac{x}{k_2} - x)^2}$$

Clearly, this equals r_2. However, for the remaining data points p_j with $j \geq k_2$ the distance to the center point p_{c_2} is computed as follows:

$$||p_j - p_{c_2}||_2 = \sqrt{k_2(\frac{x}{k_2})^2 + x^2}$$

We immediately see that

$$||p_j - p_{c_2}||_2 > ||p_i - p_{c_2}||_2$$

holds. This, in turn means than all data points p_j do not lie inside the enclosing hyperball with its radius r_2. Due to the fact, that the minimal enclosing hyperball comprises its boundary points only, and, thus, does not have any *inner* points, we can conclude that the minimal inscribed hyperball equals the minimal enclosing hyperball.