Chapter 1

Introduction

We consider the *nearest-neighbor problem*, which is defined as follows: given a fixed set P of n data points in some metric space X, build a data structure such that for each given *query point* q a data point from P closest to q can be found efficiently.

The underlying metric space is usually the d-dimensional real space $\mathbb{R}^d$ together with one of the L_p-metrics, $1 \leq p \leq \infty$. The efficiency of a solution is measured essentially by the *query time* and the *storage requirement* of the data structure. The *preprocessing time* to build the data structure is of less importance, since this computation is done only once. The nearest-neighbor problem has been among the first problems investigated in the field of computational geometry. Originally, it was formulated by Minsky and Papert [54]. In two dimensions the problem is well understood and known as the post-office problem. An optimal solution in two dimensions is provided by data structures based on Voronoi diagrams, which are closely related to the nearest-neighbor problem and have been broadly surveyed [12]. An early point location structure for nearest neighbor in two dimensions has been proposed by Dobkin and Lipton [21], with $O(\log n)$ query time, $O(n)$ storage requirement and $O(n \log n)$ preprocessing time.

The Curse of Dimensionality

The nearest-neighbor problem is a natural geometric problem which has numerous applications in several areas of computer science like statistics and data analysis [20], information retrieval [58, 26], data compression [32], pattern recognition [23], and multimedia databases [61].

One of the earlier applications is the nearest-neighbor based pattern recognition [23]. Here the points are feature vectors describing certain objects. The objects are classified in pattern classes. Given an unclassified query pattern, its feature vector is compared with a set of *prototypes* in the data structure, which are typical feature vectors for the different pattern classes. The prototype vector closest to the query vector is determined. This represents the pattern class to which the pattern is classified to belong to. A typical example is character recognition. In this case, the dimension of the feature vectors is around 20.

In multimedia applications such as IBM's QBIC [30] and MIT's Photobook [55], the number of features could be several hundreds. The multimedia retrieval system QBIC (Query by Image Content) was developed at the IBM Almaden Research Center and operates on large image and video databases. During the process of creating the database, images and videos are processed to extract feature vectors describing their content. The preprocessed feature vectors are stored in the

database and contain information about color, textures, shapes, and camera and object motion. A user can query the database by specifying an image or scene graphically, for example by a sample image, a drawing or color patterns and texture patterns. Features are generated from the graphical query, and a nearest-neighbor algorithm finds the images and videos from the database with feature vectors closest to the query vector.

In other applications like in information retrieval for text documents the dimension of the vector space can reach several thousands. This is based on the fact that the standard approach used in information retrieval is to map text documents into vectors by counting for each word the number of their occurrences in the document [60, 59].

For these applications, which involve data sets of high dimension, at least in the order of a few hundred, it is essential that the data structures used do not have running times and storage requirements that are exponential in d.

The known data structures based on Voronoi diagrams [21, 19] are impractical for high dimensions. The randomized data structure of Clarkson [19] is based on triangulated Voronoi diagrams and needs $O\left(n^{\lceil d/2 \rceil (1+\epsilon)}\right)$ space and preprocessing (for each fixed $\epsilon > 0$), and $O(2^d \log n)$ query time. The technique by Meiser [53] improves the query time to $O(d^3 \log n)$ while the storage requirement is $O(n^{d+\varepsilon})$. The huge amount of storage required by methods based on Voronoi diagrams is explained by the fact that the complexity of the d-dimensional Voronoi diagram is $\Theta\left(n^{\lceil d/2 \rceil}\right)$.

Agarwal and Matoušek [1], and Matoušek and Schwarzkopf [51] present a data structure based on ray shooting with a time-space tradeoff of $O\left((n/m^{1/\lceil d/2 \rceil})\text{polylog } n\right)$ query time for a $\Theta(m)$-space data structure. Their data structures require $\Omega(n^{c\lceil d/2 \rceil})$ space (for some fixed $0 < c < 1$) in order to achieve a sublinear query time.

Kapoor and Smid [45] use range trees to solve the L_∞-nearest-neighbor problem, where the distance between d-dimensional points is measured in the L_∞ metric. Their data structure has $O\left((\log n)^{d-1} \log\log n\right)$ query time and needs $O\left(n(\log n)^{d-1}\right)$ space.

Because of their exponential dependence on the dimension, all these techniques are in high dimensions not competitive with the brute-force method, which just determines the distance of q to each point in P and selects the minimum. This method obviously requires no preprocessing, only the set P itself needs to be stored, and the query time is $O(nd)$ for all L_p-distances, $1 \leq p \leq \infty$.

This difficulty to design efficient data structures where the dimension of the metric space is large has been referred to as the "curse of dimensionality". To circumvent this exponential growth in the dimension, researchers looked for $(1+\epsilon)$-*approximate* solutions for the nearest-neighbor problem. These are data structures that efficiently find points that are possibly not the nearest neighbor to the query point, but whose distance differs by at most a factor $(1+\epsilon)$ from the minimal one, where $\epsilon > 0$. One of the early approximate solutions is given by Arya and Mount [10] which has still an exponential dependence on the dimension where the base depends on the quality of the approximation. The traditional methods based on quadtrees, KD-trees or variants like BBD trees [11, 9] have a query time which is exponential in d. A break-through was achieved by working with randomized techniques. The data structures presented by Kushilevitz et al. [49], Indyk and Motwani [44], and Indyk [42, 43] find $(1+\epsilon)$-approximate nearest neighbors for a fixed constant $\epsilon > 0$, with query time that is polynomial in d and $\log n$, and storage and preprocessing time that are polynomial in d and n, with an exponent that depends on ϵ.

Our results

In this thesis we consider the high-dimensional *exact* nearest-neighbor problem. We analyze the *average-case* situation when the data points are chosen independently at random under uniform distribution. We do not assume any distribution of the query point.

The Voronoi diagram based data structures mentioned above have an expected storage requirement that is exponential in d, if the points are independently chosen at random from the interior of a d-dimensional ball. Dwyer [24] shows that the expected combinatorial complexity of the Voronoi diagram is $\Theta(c_d n)$, where c_d depends only on the dimension, but is of order $d^{\Theta(d)}$.

We concentrate on the L_∞-distance. In many multimedia retrieval applications like the mentioned QBIC system, the used distances are rather similar to the L_∞-distance than to other Minkowski distances. This is based on the fact that most of the vector features are independent and the similarity of two vectors is measured by the maximum of the scaled distances between corresponding features.

Our main contribution is to present algorithms which considerably improve in the average case the brute-force method. Furthermore, they have the advantage to be simple and easy to implement, and to have a low storage requirement.

Model of computation The model of computation which we adopt in this thesis is the *real RAM* as proposed in [57]. This is similar to the classical *random-access machine* (RAM) model described in [3]. The main difference is that each storage location can hold a single real number, and that comparisons, arithmetic operations, as well as some analytic functions, like d-th roots and logarithms, are available at unit cost.

Preliminaries We consider $P = \{p^1, \ldots, p^n\}$ to be a fixed set of n data points $p^i = (p^i_1, \ldots, p^i_d) \in \mathbb{R}^d$. Since we can translate and scale any given set P without changing the problem we will assume without loss of generality that $P \subset [0,1]^d$.

We present the expected runtime analysis of our algorithms, under the assumption that the n data points are drawn independently at random from $[0,1]^d$ under uniform distribution. A generalization to other "well-behaved" distributions is shown in Section 4.3. We do not assume any distribution of the query point.

Overview The contributions of this thesis are:

- In Chapter 2 we consider two query algorithms, which, like the brute-force method, need no preprocessing and require storage only for the point set P. Their average running time, assuming that the set P is drawn randomly under uniform distribution, is $\Theta\left(\frac{nd}{\ln n} + n\right)$. One of the algorithms is the CUBE METHOD, which has been introduced and analyzed by Alt and Hoffmann [5, 40].

- In Chapter 3 we present two strategies which speed up the CUBE METHOD by using preprocessing.

  In Section 3.1 we make use of a simple preprocessed data structure of linear size, where the coordinates of the data points are stored sorted in each of the dimensions $\{1, \ldots, d\}$. The

query algorithm has an expected running time of $O\left(n \ln\left(\frac{d}{\ln n}+1\right)+n+d\ln d\right)$, assuming that the data set is drawn randomly under uniform distribution.

The data structure developed in Section 3.2 is based on a preprocessed partition of the data set into sequences, which are monotone with respect to some of the dimensions. The query algorithm works well for dimensions $d \leq \left(\frac{\ln n}{\ln\ln n}\right)^2$, when it has an expected running time of $O\left(\sqrt{d}\cdot n^{1-\frac{1}{\sqrt{d}}}\cdot \ln n\right)$, assuming uniform distribution of the data set.

- Chapter 4 presents several generalizations of the CUBE METHOD, in particular to the important problem of finding the k nearest neighbors to a query point. Next, we generalize the analysis of the considered algorithms to other "well-behaved" probability distributions. Furthermore, we develop in Section 4.4 extensions of the algorithms which work efficiently in the external-memory model of computation introduced in [2].

  Section 4.5 shows an experimental comparison of implementations of the presented variants of the CUBE METHOD. We emphasize the factors by which the algorithms are faster than the brute-force method.

- In Chapter 5 we develop a method which provides tradeoffs between the space complexity of the data structure and the time complexity of the query algorithm.

The method presented in Section 3.1 improves the brute-force method by a factor of $O(\frac{\ln(d/ln\ n)}{d})$ for high dimensions $d > \ln n$. The constant in the O-term is close to one, and for dimensions d in the range of some hundred the speedup factor comes close to 50, which is a considerable improvement in practice. Furthermore, the query algorithm is simple and easy to implement, and uses a data structure of linear size.

Some of our results have been published in [7], [38] and [39].

Acknowledgements

I would like to thank my advisor, Helmut Alt, for sharing his insights and experience with me. I benefited very much from his excellent guidance and support. It was an honor and an important experience to be a member of the graduate program *Computational Discrete Mathematics*, which provided me an ideal research environment. I would like to thank my colleagues from the work group *Theoretical Computer Science* at the Free University Berlin for the creative and energizing atmosphere, and for their support during the phase of writing my thesis. I especially thank Peter Braß, Géraldine Morin and Martin Kutz for reading earlier versions of this thesis and for their useful comments related to its presentation. I thank Piotr Indyk for his fruitful suggestions on the topic. My thanks to Tilo Heinrich for giving me strength, backup and constant encouragement.