# A Statistical Approach to Algorithmic Analysis of High-Dimensional Nearest-Neighbor Search

Alexander Gray

February 2004
CMU-CS-04-108

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Abstract**

The most commonly used algorithms for spatial data searches such as $k$-nearest-neighbor and spherical range queries are based on a class of data structures we call space-partitioning trees, which have remained the pragmatic method of choice due to their ability to often empirically provide sub-linear efficiency in reported dimensionalities in the tens and occasionally beyond, in constrast to methods designed for worst-case optimality. Despite long-standing practical interest in a more realistic runtime analysis of such methods, particularly in the high-dimensional case demanded by many modern applications, little further progress has been made since the seminal expected-time analysis of 1977. One fundamental reason for this is that algorithm analysis has not, to date, provided examples of analyses which link algorithmic runtime to probabilistic properties of the input distribution. This paper introduces some basic statistical machinery for making this link, and thereby presents initial steps toward providing a statistically principled framework for distribution-dependent runtime analysis of space-partitioning-based algorithms, with an emphasis on providing explanations for their observed behavior in high-dimensional spaces.

The author was supported by the NASA Graduate Research Fellowship.

**Keywords:** algorithmic analysis, data-dependent analysis, $k$-nearest neighbor, range queries, adaptive data structures, high dimensionality, distance distribution

# 1 Data-Dependent Algorithmic Analysis

Algorithm analysis, to date, has focused primarily on *worst-case* analyses. This type of analysis has unarguably provided much insight into the design of useful algorithms, as exemplified by a long list of successes. However, it is also clear that this paradigm is not a panacea for all purposes. While worst-case analysis is often useful *prescriptively*, *i.e.* providing insights for deciding between different designs and guiding ideas for new designs, it is widely understood that worst-case runtime analyses generally do not serve as practical *predictive* tools, as evidenced by the often large gap between the actual runtimes of algorithms and their (worst-case) analytical growth functions.

Having a predictive analysis in hand for a particular algorithm or class of algorithm has several potential benefits.

1. Most generally, as it must inevitably constitute a deeper model of the workings of the algorithm, deeper insight is obtained.

2. Understanding the dependence of algorithms on key aspects of their input provides an essential guide to the practitioner in selecting the method appropriate for the problem at hand, or in ensuring that the inputs are as favorable as possible for the algorithms at hand.

3. An effective predictive analysis would replace the time-consuming experimentation often necessary by users – by measuring a few critical properties of the data, the time cost would be estimated before the algorithm is ever run or even implemented.

4. Finally, an analysis with improved predictive relevance is also also likely to have improved prescriptive relevance. For example, space-partitioning methods, which exhibit relative success in arbitrary dimension, are completely eliminated from consideration when taking a more simplistic worst-case perspective.

Note that this is partially but not completely related to the common complaint that potentially large constants are ignored in *asymptotic* runtime analyses. In this paper we will perform both asymptotic analyses and analyses which are meant to be relatively tight upper bounds on actual runtime.

Our chief quantity of interest is expected runtime. We do not consider space cost, as the worst-case $O(N)$ data structure size is not particularly large or variable for the structures considered.

In this paper the expectation is over the *input*, in contrast to work on *randomized algorithms* [27, 24, 25], which are allowed to make decisions randomly at runtime, for example based on coin tosses. The analysis of randomized algorithms also yields expected-time analyses, but the expectation is over these random *decisions*. The algorithms we consider are deterministic at runtime.

A related thread of work in computational geometry is the notion of *output-sensitive* complexity analysis (*e.g.* [6]), where the runtime is given based on a property of the *output* (typically, its size) rather than a property of the input. However, since the output is presumably dependent on the input, this can be seen as an indirect form of data-dependent algorithmic analysis. It follows a similar spirit in that the result is a runtime parametrized by some measurable property which affects the problem's difficulty for the algorithm at hand. However, parametrizing runtime by a property of the input allows one to make statements about the algorithm's performance on $X$ without actually running it, whereas properties of the output are necessarily *ex post facto*.

## 1.1 Proximity Problems and Algorithms in Practice

In this paper we focus on two common types of proximity querying, $k$-nearest-neighbor and spherical range querying. We study them together because they are related by a simple common perspective which relates their analyses in this context. While the $k$-nearest-neighbor problem has been extensively studied, spherical range querying (also called circular range querying, in two dimensions, or fixed-radius nearest-neighbor querying) has received relatively little attention. The vast majority of work in computational geometry has been concerned with rectangular range queries.

The algorithms we will consider are based on a class of data structure we refer to as *space-partitioning trees*, to be described in § 2. While a huge number of variations have been proposed over the decades, the two most successful and popular types of such structures are *kd*-trees [4, 14] and *metric trees* [35, 9]. Their predominance in $k$-nearest-neighbor and range queries occuring daily in database searches [32], pattern recognition [15], image processing [30] and other applications is reason enough to study these structures.

However, there is also theoretical insight to be gained, since the reason for the widespread use of these structures stems from their empirically-observed ability to provide faster-than-exhaustive search efficiency in dimensionalities ranging up to about 20 to 40 according to different authors but occasionally even larger, *e.g.* [21]. As we'll show, this ability arises from their data-adaptive nature, which causes their runtime to be sensitive to the *effective* dimensionality $D'$ of the data, which is in general smaller than the explicit dimension $D$ of the measurement space. This is not possible in practice for methods which have been developed from a worst-case perspective where the runtime is exponential strictly in $D$. [23] actually showed that worst-case $O(D^5 \log N)$ query time is possible, but only at the untenable space cost of $O(N^{D+\varepsilon})$ for any $\varepsilon$. The resulting state of affairs is that space-partitioning methods represent the practical state of the art for exact proximity searching in arbitrary dimension [31]. This is echoed by Clarkson, who recently proposed a variant of metric trees as an all-around practical choice for arbitrary-dimension $k$-nearest-neighbor [11], though an analysis of the algorithm was not given.

## 1.2 Previous Analyses

In [14] the authors showed that *kd*-trees achieve expected $O(\log N)$ time, with a constant factor which is exponential in $D$, regardless of the data's distribution. The gap between this bound and the actual runtime has been observed by several authors (*e.g.* [34]).

In the database community in particular, where space-partitioning trees are central workhorses in running applications, there has been a large interest in obtaining more predictive runtime analyses, *e.g.* [10, 5]. Much of this work has in fact focused on attempting to relate notions of effective dimension to the cost of searching, *e.g.* [13]. However, these analyses are unsatisfactory for several reasons. First, the database context places emphasis on interactions with secondary storage – the tree structures are assumed to themselves be stored on disk rather than entirely in RAM, and thus even traversing a tree requires disk accesses in this context. The resulting analysis is different from the goal here, which is algorithmic runtime analysis rather than disk access cost prediction. This also brings in unnecessary factors which can serve to confound all analysis, such as the need to characterize the extent to which nodes that are close in space are also stored nearby on disk. Second, the tree construction methods studied are incremental, or online, and thus unnecessarily suboptimal. Good properties which can be easily achieved by batch construction methods can arbitrarily corrupted by bad orderings of tree insertions, again adding an unnecessary factor to the analysis. Finally and most importantly, many of the theoretical claims in this literature are not rigorous, or are made based on implicit and possibly incorrect assumptions, and thus are not general. Thus, while we do borrow the qualitative insight from this literature concerning the effective dimension, we cannot directly use any of these analyses for our purpose. In our view, then, [14] remains the most complete analysis to date. [2] performs an analysis with similar structure, though worst-case rather than expected-time, and for the case of approximate $k$-nearest-neighbor querying using BBD-trees, a kind of space-partitioning tree.

[2] pointed out another reason, besides dependence on the effective dimension rather than the explicit dimension, that space-partitioning methods are often faster in practice than their analyses suggest – that queries near the boundary of the data are cheaper since there are fewer points to consider. Indeed, our analysis also makes it clear that this is a source of unnecessary pessimism in the runtime, though we do not explicitly correct for this.

## 1.3 Our Goals

In order to obtain a data-dependent analysis for space-partitioning methods, we must somehow relate the structure of clouds of continuous data, *i.e.* $X \in \mathbb{R}^D$, to the discrete structure of a divide-and-conquer algorithm. The analytical tools we'll need for this continuous context are likely to be unfamiliar within the discrete algorithms community. It is clear that statistical notions (in fact the analysis of [14] is statistical in nature) are needed in addition to the traditional discrete computer science tools. Rather than acknowledging an absence of necessary theory, some authors simply dismiss space-partitioning methods as "heuristics".

Instead, we take the more constructive path of introducing the required tools from statistics and show that these algorithms are in fact analyzable.

This paper represents a proposed first step toward providing a rigorous data-dependent account of the runtime of space-partitioning tree-based algorithms. By 'data-dependent' we mean that our resulting analysis will yield a runtime which depends (at various levels) on certain key quantities which characterize the data $X = \{x_1, x_2, \ldots, x_N\}$: $f(x)$, the density of the data; $F(\delta)$, the distribution of the $N(N-1)/2$ pairwise distances $\delta$; and $D'$, a measure of the effective dimension of the dataset (to be described). These quantities may be known *a priori* or may be estimated. We have chosen $F(\delta)$ as the central focal point of our analyses because it is linked to several bodies of theory in different fields (we'll touch on spatial statistics, statistical physics, and fractal geometry), which we hope will open the door to more advanced analytical tools lying in these areas.

This statistical data dependence in the analysis is novel for this problem. More generally, we are unfortunately aware of no other such analyses for any algorithm. Among other things, our analysis 1) appears to be the first runtime analysis of spherical range querying with space-partitioning trees, and agrees with the prior analysis of [14] while providing further insights, 2) analytically demonstrates the dependence of the runtime of space-partitioning methods on the effective dimension, a notion which has been postulated though examined with less statistical rigor, and 3) yields additional clarity regarding the utility of $(1+\epsilon)$ approximation for $k$-nearest-neighbor querying.

We also hope the ideas in this paper will have implications for data-dependent (or adaptive) data structure design and data-dependent runtime analysis in general, which might be considered a largely unwritten branch of the algorithms and data structure literatures. For space-partitioning trees in particular, a good predictive analysis is a necessary step toward another possibility which would firmly remove them from the realm of 'heuristics', that of defining and deriving instances of *optimal* space-partitioning trees.

### 1.4 Outline of this Paper

In § 2 we describe the problems and algorithms with which we are concerned. Since the data for our sort of problem are points in space, in § 3 we take a brief tour of the fundamental theoretical quantities which are used to analyze properties of points in space, which will motivate building our analysis on the particular quantity $F(\delta)$. In § 4 we'll construct analyses of runtime in terms of $F(\delta)$. In § 5 we'll make the link from $F(\delta)$ to an even more basic quantity, $f(x)$. In § 6, using insights from $F(\delta)$, we examine and resolve misconceptions regarding the behavior of proximity searches in high dimensional spaces. We conclude by discussing a few of the many future avenues for further developing these ideas in § 7.

## 2 Proximity Problems and Algorithms

### 2.1 The Problems: $k$-Nearest-Neighbor and Spherical Range Queries

*$k$-nearest-neighbor* queries ask to return the point $x^*$ whose distance to the query point $q$ is the $k^{th}$ smallest among all points in $X$ other than $q$. All points with smaller distances are also returned. *Spherical range queries* ask to return all the points in $X$ other than $q$ whose distance to $q$ is less than some user-specified value $r$. These problems are related in the sense that they each have a different version of the notion of a *critical distance*, which we will use shortly.

### 2.2 The Algorithm: Divide-and-Conquer

Figure 1 shows the basic template divide-and-conquer algorithm for range searching using a space-partitioning tree, with range $r$. At any point in the recursion the query point $q$ is compared with the node, or region, $R$. A lower-bounding function $\phi(q, R)$, also written $\|\underline{x}_q - R\|^{lo}$, is invoked to obtain a distance $l_{qR}$ which is a lower bound on the distance of $q$ to any point in $R$. If this lower bound is greater than $r$, then node $R$ does not contribute to the answer and may be *pruned*, *i.e.* no further recursion is performed in the subtree represented by $R$.

We refer to the distance beyond which we can prune as the critical distance. In range searching the critical distance is simply $r$. Note that in the $k$-nearest-neighbor case, the critical distance would ideally

```
RS(q,R)
  l_qR = φ(q, R).
  if l_qR > r, return.
  if leaf(R), RSBase(q,R).
  else,
    RS(q,closer-of(q,{R.left,R.right})).
    RS(q,farther-of(q,{R.left,R.right})).

RSBase(q,R)
  foreach x_i ∈ R,
    δ_qi = ||q − x_i||.
    if δ_qi ≤ r,
      add-to-set(r−Set_q, x_i).
```

Figure 1: RANGE SEARCH ALGORITHM. In the pseudocode `a = b` denotes assignment. Note that this algorithm can be written in a slightly different way such that distance computations are never repeated, but this presentation is more conceptually transparent for our purposes. closer-of($A$,$B$) returns the closer of its two node arguments $A$ and $B$; farther-of() is similar. add-to-set($A$,$B$) implements $A = A \cup B$ and could also have been written $A \cup = B$.

be $\delta^* = \|q - x^*\|$, but its value is not actually known until the algorithm terminates, and instead an upper bound of it is used for pruning, the distance of the best $k^{th}$ nearest neighbor found so far, which we'll call $\hat{\delta}$. The $k$-nearest-neighbor version of the algorithm is very similar, except the critical distance changes to that of the best neighbor candidate found so far, and slightly more code is needed to maintain the validity of the candidate set of neighbors. Thus we only show the range-search case, to avoid these distracting elements.

Several variations are possible, including the use of a priority queue to guide the search rather than best-first search. (*e.g.* [3]). When a Fibonacci queue is used, the runtime anaysis will be essentially the same. In practice, the runtimes are observed to be almost identical. Another variation, in the $k$-nearest-neighbor case, is to simply compare the lower bound distances to $(1 + \epsilon)$ times the critical distance [2], to obtain an approximation in which the distance to any point returned is guaranteed to lie within $(1 + \epsilon)$ times the true $k$-nearest-neighbor distance.

## 2.3 The Data Structures: Space-Partitioning Trees

By *space-partitioning trees*, we mean a large class of tree data structures which includes *kd*-trees and metric trees. See [31] for a large number of examples. Though they differ in the way space is partitioned, *i.e.* the way data are split at each level into two chunks, and the shape of the resulting regions, their central commonality of interest with respect to this paper is the fact that they supply a (generally cheap) lower-bounding function $\phi(q, R)$ of some sort, and typically maintain balance of the tree while also minimizing variance (*i.e.* the maximum extent of a node in any direction). We'll first define space-partitioning trees generally, then describe real examples.

**Definition 1 (Space-partitioning tree)** *A space-partitioning tree is defined by a 4-tuple* $\mathcal{T} = \{T, g(\cdot), \phi(\cdot, \cdot), X\}$*:*

- **Structure.** *$T$ is the discrete tree structure. It is the set of nodes and the connectivity between them. Let $v$ index the nodes.*
- **Partitioning function.** *Let $x \in \mathbb{R}^D$, $x \sim f$, the density of $x$. Each node is associated with a set $R \subset \mathbb{R}^D$. $g_v : x \mapsto R$ is a partitioning function mapping a point $x$ to either $R_1$ or $R_2$, the two children of a node. (We often think of $R$ as a region in $\mathbb{R}^D$ though it need not be contiguous or connected.) $R_1 \cup R_2 = R$ and $R_1 \cap R_2 = \emptyset$. We'll usually refer to a node $v$ by its associated region $R$.*
- **Bounding function.** *$\phi_v(q, R)$ is a function yielding a lower bound on the distance from a point $q$ to any point in $R$. $\phi_v(q, R) \leq \inf_i^{N_R} d(q, x_i) = \delta_R^*$, where $N_R$ is the number of data in the set $R$.*
- **Dataset.** *$X$ is a collection of points $x_1, x_2, \ldots, x_N \in \mathbb{R}^D$. If $X \neq \emptyset$ then $\mathcal{T}$ is called a data-dependent space-partitioning tree and may be written $\mathcal{T}_X$.*

*The parameters associated with the functions $g_v(\cdot)$ and $\phi_v(\cdot,\cdot)$ will be collectively called $\Theta$.*

If $g_v(\cdot)$ and $\phi_v(\cdot,\cdot)$ are the same for every node $v$ then the $v$ subscript is left off. This is the typical case. Most often the partitioning is done one node at a time in a recursive top-down fashion for computational reasons. Optionally there are also upper bounds in addition to lower bounds. Though we did not show this in the pseudocode for range searching, it is possible to also use upper bounds to computational advantage (see [18]).

All successful methods use simple ways of constructing the tree in a data-dependent manner. Thus we always consider this case from now on. Several common structures correspond to different data-dependent choices for the functions $\phi_v(\cdot,\cdot)$ and $g_v(\cdot)$.

**Example 1 (Metric tree)** *A typical metric tree, or ball tree includes in its parameters $\Theta$ a center $c_v$ for each node, and makes the following choices for $\phi(q,R)$ and $g(x)$:*

$$\phi(q,R) = ||q-c|| - \max_{i\in R} ||x_i - c|| \quad \textit{where} \quad c = \frac{1}{N_R}\sum_{i\in R} x_i, \tag{1}$$

*i.e. the triangle inequality is used with respect to the node center to provide a distance lower bound. $\{R_1, R_2\}$ is defined by a discriminant function $g(x)$ which returns the index of either the left or right branch according to the distance to the corresponding node center:*

$$g(x) = \arg\min\{||x - c_{left}||, ||x - c_{right}||\} \tag{2}$$

*or equivalently, $g(x) = \mathrm{sgn}(w^T x)$, when expressed in terms of a separating hyperplane $w$. Procedures for constructing metric trees typically seek to minimize the radii of the balls as well as minimize the overlap between balls at the same level.*

*kd*-trees correspond to hyperrectangular regions and univariate axis-aligned splits. The method of [14] chooses the dimension with largest spread (assuming each dimension has been normalized to have the same range). A common improvement to their scheme uses the minimum bounding hyperrectangle to define $\phi(q,R)$. We won't review *kd*-trees in more detail as they are perhaps the most familiar structures in this class.

# 3 Analytical Tools

Several fields are concerned with the distribution and other properties of points in space. *Spatial statistics* [28, 12] is broadly concerned with constructing statistics capturing the existence of pattern, in various senses, the most fundamental of which is the amount of deviation from complete spatial randomness. *Statistical physics* [22] has similar requirements, arising from the analysis of ensembles of objects, with or without mutual interactions, such as molecules. *Nonlinear dynamics* [26] is concerned with ensembles of points collected over time, which represent the trajectory of a dynamical system.

## 3.1 Continuous Probability

To review basic probability [8] for a continuous random variable $\tau \in \mathbb{R}$, recall that its *cumulative distribution function* (CDF) $F : \mathbb{R} \to [0,1]$ is defined as

$$F(t) = \Pr(\tau < t) \tag{3}$$

and its *probability density function* (PDF) is a function $f(t) \geq 0$ such that $\int_{-\infty}^{\infty} f(t)dt = 1$ and

$$\Pr(t_1 < \tau < t_2) = \int_{t_1}^{t_2} f(t)dt. \tag{4}$$

They are related by

$$F(t) = \int_{-\infty}^{t} f(t)dt \tag{5}$$

$$f(t) = F'(t) \tag{6}$$

at all points where $F(\cdot)$ is differentiable. We will refer to the $f$ and $F$ of several different random variables, allowing the argument to distinguish the random variable to which the function refers.

### 3.2 Fundamental Statistical Quantities

In spatial statistics, the perspective surrounds that of a *spatial point process*, or a (possibly unknown) stochastic mechanism which generates a finite set of points. We will assume the points fall within a bounded region $\Re$ in $D$ dimensions. Let $\Re$ be the unit cube $[0,1]^D$ unless otherwise specified. The starting point in spatial statistics (and thus in our analyses) is the uniform distribution. When the number of data $N$ is fixed we can think of points which are uniformly distributed within $\Re$ as being drawn from a *stationary, isotropic Poisson process* which has one parameter $\lambda$, the *intensity* (which can be thought of as the point density) $\lambda(x) = \lim_{V(dx)\to 0}\left\{\frac{\mathbb{E}[N(dx)]}{V(dx)}\right\}$, where $dx$ is an infinitesimal region containing $x$ and $N(A)$ is the number of points contained in region $A$, which has volume $V(A)$. An estimator for $\lambda$ given a finite dataset $X$ is $\hat{\lambda} = N/V(A)$. Over the entire region $\Re$, then, $\hat{\lambda} = N$.

In this paper we assume the standard Euclidean distance, though all of our results are easily extended to arbitrary Minkowsky $L_p$ norms $\delta(x_1, x_2) = ||x_1 - x_2||_p$.

So-called second-order properties of spatial point processes form the main theoretical quantities which are analyzed. The two main second-order properties of interest are the *second-order intensity function*

$$\lambda_2(x_1, x_2) = \lim_{V(dx_1),V(dx_2)\to 0}\left\{\frac{\mathbb{E}[N(dx_1)N(dx_2)]}{V(dx_1)V(dx_2)}\right\}, \tag{7}$$

which can also be written as $\lambda_2(\delta_{12})$, where $\delta_{12}$ is the distance between $x_1$ and $x_2$ and the function

$$K(\delta) = \frac{1}{\lambda}\mathbb{E}[N_0(\delta)], \tag{8}$$

where $N_0(\delta)$ is the number of points within distance $\delta$ of an arbitrary point, *i.e.* one drawn uniformly at random from within the region of interest. An estimator for $K(\delta)$ given finite dataset $X$ is $\hat{K}(\delta) = \frac{1}{N}\sum_i \sum_{j\neq i} I(\delta_{ij} < \delta)$ where $I(\cdot)$ is the indicator function and $\delta_{ij}$ is the distance between points $x_i \in X$ and $x_j \in X$.

In statistical mechanics, ensembles of points are characterized by the so-called *radial distribution function* or *pair correlation function*, which is defined in terms of the probability

$$\lambda^2 g(\delta) dV_1 dV_2 \tag{9}$$

of finding a point in each of two infinitesimal balls having volumes $V_1$ and $V_2$ and separated by distance $\delta$. We can see that $g(\delta) = \frac{1}{\lambda^2}\lambda_2(\delta)$.

In nonlinear dynamics, we find the following quantity called simply the *correlation function*:

$$C(\delta) = \lim_{N\to\infty}\left\{\frac{1}{N^2}\sum_i \sum_j I(\delta_{ij} < \delta)\right\} \tag{10}$$

which will come up again in § 6.

### 3.3 Inter-Relationships

We'll now synthesize all of these theoretical frameworks by showing that they are all related to each other through two simple quantities, $f(\delta)$, the density of distances, and $F(\delta)$, the cumulative distribution of distances.

First, we see that by definition, $g(\delta) = f(\delta)$, and $\lambda_2(\delta)$ is a version which is not normalized to be a density. An estimator of $f(\delta)$ is

$$\hat{f}(\delta) = \frac{1}{h}\frac{1}{2N(N-1)}\sum_i\sum_{j\neq i} I(|\delta_{ij} - \delta| < h), \tag{11}$$

where $h$ is a parameter of the estimator, which is called a *kernel density estimator*. In general the selection of $h$ for optimal estimation is nontrivial [33].

Noting that the *empirical distribution function*, an estimator of $F(\delta)$, has the form

$$\hat{F}(\delta) = \frac{1}{2N(N-1)}\sum_i\sum_{j\neq i} I(\delta_{ij} < \delta), \tag{12}$$

we see that $K(\delta) = 2(N-1)F(\delta)$ and $C(r) = 2F(\delta)$.

Thus it is clear that the various analytical quantities used in these various fields are simply variations of what we might intuitively expect to be fundamental quantities of interest, $f(\delta)$ and $F(\delta)$. Further, since $f(\delta)$ and $F(\delta)$ are each the derivative or integral of the other, we need only focus our attention on one of the two. The fact that the density is more difficult to estimate directly than the cumulative distribution, due to the parameter $h$, helps to motivate use of $F(\delta)$ as the primary 'property' of the data $X$ upon which our algorithmic analysis will be based.

Our goal is now to construct a runtime analysis which is parametrized by $F(\delta)$, so that measurement of $F(\delta)$ for an input dataset $X$ will give us information about the runtime behavior of the algorithm on $X$.

# 4 Data-Dependent Complexity

We want to be able to say what the expected runtime is for a given a dataset $X$ having distance distribution $F(\delta)$ and a tree $\mathcal{T}_X$. We'll measure computational cost in terms of distance calculations, where we include the distance calculations used in pruning checks. This is quite realistic, as virtually all of the time cost is in the distance calculations, *i.e.* the growth in the number of distance calculations is a very good proxy for growth in runtime in practice.

The main idea is as follows. If we had perfect pruning, the runtime would be simply by the usual recurrence for binary search:

$$\begin{aligned} T(N) &= T(N/2) + O(1), \\ T(1) &= O(1). \end{aligned} \tag{13}$$

However since pruning may or may not happen, we can characterize the runtime in terms of some probability $\alpha$ that the 'wrong' side of the tree is also traversed:

$$\begin{aligned} \mathbb{E}[T(2^l)] &= \mathbb{E}[T(2^i/2)] + \alpha_i\mathbb{E}[T(2^i/2)] + O(1), \\ \mathbb{E}[[T(1)] &= O(1). \end{aligned} \tag{14}$$

where $i = \log_2 N$ indexes the level of the tree.

Let $l_{qR} = \phi(q, R)$, the lower bound on the distance from the query $q$ to any point in region (node) $R$ given by the tree. In the standard divide-and-conquer algorithm, if $\delta_{crit} \leq l_{qR}$, we can prune node $R$, otherwise we must enter it. We define $\alpha_i$ as the chance that we must enter node $R$, given $\delta_{crit}$, expressed as an expectation over all nodes $R$ at level $i$ and all queries $q$ drawn from $f_x$:

**Definition 2 (Recursion probability)**

$$\alpha_i \equiv \mathbb{E}[p(\delta_{crit} > l_{qR})]_{q\sim f_x, R\in\{T\}_i} \tag{15}$$

*where $\{T\}_i$ denotes the set of nodes in tree $T$ at level $i$.*

Now we can relate the runtime to $F(\delta)$.

**Lemma 1 (Recursion probability wrt $F(\delta)$)**

$$\begin{aligned}
\alpha_i &= F(r) \quad \textit{(range-query)} && (16)\\
\alpha_i &= \mathbb{E}[F(\hat{\delta})] \quad \textit{(k-nearest-neighbor)} && (17)
\end{aligned}$$

**Proof:** We can see that $l_{qR} \sim f(\delta)$ because as $N$ grows and node sizes become infinitesimal, distances to regions are indistinguishable from distances to points. Then

$$\begin{aligned}
\mathbb{E}[\Pr(\delta_{crit} > l_{qR})] &= \mathbb{E}[\Pr(l_{qR} < \mathbb{E}[\delta_{crit}])] \\
&= \mathbb{E}\left[\int_{-\infty}^{\mathbb{E}[\delta_{crit}]} f(\delta) d\delta\right] \\
&= \mathbb{E}[\{F(\mathbb{E}[\delta_{crit}]) - F(-\infty)\}] \\
&= \mathbb{E}[F(\mathbb{E}[\delta_{crit}])].
\end{aligned}$$

In the case of range searching, $\mathbb{E}[\delta_{crit}] = \mathbb{E}[r] = r$. In the case of $k$-nearest-neighbor, $\mathbb{E}[\delta_{crit}] = \hat{\delta}$. ■

Though we can derive an expression for $\mathbb{E}[\hat{\delta}]$, in practice it is extremely close to the true value, and thus a reasonable approximation is $\mathbb{E}[\hat{\delta}] \approx \delta^*$. We'll see how to obtain a value for $\alpha_i$ in more detail in § 5.

Now let's examine the behavior of $\alpha_i$ as a function of the level $i$ in the tree.

**Lemma 2 (Bound on $\alpha_i$)**

$$\alpha_i \le S_D(\delta_{crit})/(2^i\sqrt{2})^{1/D} \qquad (18)$$

*where $S_D(\delta_{crit})$ is the surface area of the ball of radius $\delta_{crit}$ in $D$ dimensions, $S_D(\delta_{crit}) = {\delta_{crit}}^D\{\pi^{D/2}/(D/2)!\}$.*

**Proof:** The result of typical space-partitioning tree construction procedures such as those for *kd*-trees and metric trees, which recursively reduce the maximum spatial extent of node regions, is that the expected maximum node diameter for a node at level $i$ in the tree is the $D^{th}$ root of the expected volume, or $1/(2^i\sqrt{2})^{1/D}$. $\alpha_i$ is the chance that a node at level $i$ intersects the ball of radius $\delta_{crit}$. We can bound this by

$$\begin{aligned}
\mathbb{E}[\Pr(\delta_{crit} \le l_{qR})] &\le \frac{\mathbb{E}[\text{maximum intersection volume}]}{\text{total volume}} \\
&= \frac{S_D(\delta_{crit})\mathbb{E}[\text{max. node diameter} \in \{T\}_i]}{V(\Re)} \\
&= \frac{S_D(\delta_{crit})(1/(2^i\sqrt{2})^{1/D})}{1^D} \\
&= S_D(\delta_{crit})/(2^i\sqrt{2})^{1/D}.
\end{aligned}$$

■

We see that $\alpha_i$ gets smaller as we descend lower in the tree. In light of this, to obtain a simple expression for the runtime, we can now define a single quantity $\alpha$, which upper-bounds $\alpha_i$ for all levels lower than some fixed intermediate level $k$ in the tree: $\alpha \equiv \alpha_k > \alpha_i, \forall i > k$.

**Lemma 3 (Expected runtime I)** *The expected runtime for the standard divide-and-conquer algorithm is*

$$\mathbb{E}[T(N)] = O\left(\sum_{i=0}^{\log_2 N} (1+\alpha)^i\right) \qquad (19)$$

**Proof:** Using the single upper bound $\alpha$ we can work with the simpler recurrence

$$\begin{aligned}
T'(N) &= T'(N/2) + \alpha T'(N/2) + O(1), \\
T'(1) &= O(1). && (20)
\end{aligned}$$

To solve it, the Master Theorem can't be applied, but we can telescope it to obtain

$$\begin{aligned}
T'(N) &= T'(N/2) + \alpha T'(N/2) + 1 \\
&= (1+\alpha)T'(N/2) + 1 \\
&= (1+\alpha)[(1+\alpha)T'(N/4) + 1] + 1 \\
&= (1+\alpha)^2 T'(N/4) + (1+\alpha) + 1 \\
&= (1+\alpha)^2[(1+\alpha)T'(N/8) + 1] + (1+\alpha) + 1 \\
&= (1+\alpha)^3 T'(N/8) + (1+\alpha)^2 + (1+\alpha) + 1 \\
T'(2^l) &= (1+\alpha)^l T'(1) + \sum_{i=0}^{l-1}(1+\alpha)^i + 1, \;\; l = \log_2 N \\
&= \sum_{i=0}^{\log_2 N}(1+\alpha)^i + 1.
\end{aligned}$$

Note that we must also account for cost in the first $k$ levels of the tree, where the $\alpha$ bound does not apply. The additional cost is bounded by $\sum_{i=0}^{k-1} 2^i$, a constant which we can ignore. Thus $\mathbb{E}[T(N)] = O(T'(N))$, yielding the result. ■

We can see that if $\alpha = 1$, the runtime is $O(N)$, and if $\alpha = 0$, the runtime is $O(\log N)$. So $\alpha_i$ can be seen as a pivotal dial which determines the efficiency of the algorithm.

Now we derive a more interpretable form for the complexity in terms of the critical distance.

**Theorem 1 (Expected Runtime II)** *The expected runtime for the standard divide-and-conquer algorithm is*

$$\mathbb{E}[T(N)] = O(N^{\varepsilon}) \quad \textit{(range-query)} \tag{21}$$

$$\mathbb{E}[T(N)] = O(\log_2 N) \quad \textit{(k-nearest-neighbor)} \tag{22}$$

*where* $\varepsilon \propto r^D, 0 \leq \varepsilon \leq 1$.

**Proof:** Now to get $T'(N)$ into a more convenient form, we note that it is a geometric series to obtain, when $\alpha > 0$,

$$\begin{aligned}
T'(N) &= \sum_{i=0}^{\log_2 N}(1+\alpha)^i \\
&= \frac{(1+\alpha)^{\log_2 N+1} - 1}{(1+\alpha) - 1} \\
&= \frac{(1+\alpha)(1+\alpha)^{\log_2 N} - 1}{\alpha} \\
&= \left(\frac{1+\alpha}{\alpha}\right)(1+\alpha)^{\log_2 N} - \frac{1}{\alpha} \\
&= O((1+\alpha)^{\log_2 N}) \\
&= O((e^{\alpha})^{\log_2 N}) \quad \text{since } \alpha \leq 1 \\
&= O((e^{\log_2 N})^{\alpha}) \\
&= O(N^{\alpha}).
\end{aligned}$$

We know that $\alpha \propto r^D$ because $\alpha \propto S_D(\delta_{crit})$.

As $N \to \infty$, $\delta_{crit} \to 0$ and thus $\alpha \to 0$, so that

$$T'(N) = \sum_{i=0}^{\log_2 N}(1+\alpha)^i \to \sum_{i=0}^{\log_2 N} 1 = \log_2 N.$$

■

Thus we can see that $\mathbb{E}[T(N)]$ rises exponentially in $r$ from $O(\log_2 N)$ to $O(N)$. We were unable to find any other runtime analysis for spherical range querying using space-partitioning trees. Note that the $k$-nearest-neighbor result is in agreement with the result of [14], where it was proved for the specific case of *kd*-trees, though it was obtained in a very different way, and the exact effect of the dimensionality on the runtime is somewhat different.

# 5 The Distance Distribution

Where do we get the cumulative distribution of the distances? For a given density of *points* $f(x)$, it is straightforward enough, for simple cases, to find the density of *distances* $f(\delta)$, and thus $F(\delta)$. However, taking into account the boundary of the data makes the calculation much more complex. Some simple cases can be found in the spatial statistics literature, such as [12]. For example the cumulative distribution for points uniformly distributed within a square of unit side is

$$F(\delta) = \begin{cases} \pi\delta^2 - 8\delta^3/3 + \delta^4/2 & \text{for } 0 \leq \delta \leq 1 \\ 1/3 - 2\delta^2 - \delta^4/2 & \text{for } 1 < \delta < \sqrt{2} \\ +4(\delta^2-1)^{1/2}(2\delta^2+1)/3 & \\ +2\delta^2 \sin^{-1}(2\delta^{-2}-1) & \end{cases} \tag{23}$$

For points uniformly distributed within a circle of unit radius (thus $0 \leq \delta \leq 2$) it is

$$\begin{aligned} F(\delta) &= 1 + \pi^{-1}\{2(\delta^2-1)cos^{-1}(\delta/2) \\ &\quad \delta(1+\delta^2/2)\sqrt{(1-\delta^2/4)}\}. \end{aligned} \tag{24}$$

The need to work with the distribution of $k^{th}$ *nearest* distances, $F_{(k)}(\delta)$, adds more complication. An approximation for uniformly-distributed (or Poisson) data used heavily in spatial statistics which ignores edge effects is

$$F_{(1)}(\delta) = 1 - \exp(-\pi\lambda\delta) \tag{25}$$

To explore the effect of high dimensionality on the performance of space-partitioning trees, we can utilize a result from the mathematical statistics literature due to Hammersley in 1950:

**Theorem 2 ($f(\delta)$ for uniform in $D$, I [20])** *Let $\underline{x}_1$ and $\underline{x}_2$ be any two points in a vector space chosen at random from within the $D$-dimensional sphere with radius $R$, $S_D(R)$, the distributions of $\underline{x}_1$ and $\underline{x}_2$ being independent and uniform over the interior of $S_D(R)$. Denote a normalization of their distance $\delta = \|\underline{x}_1 - \underline{x}_2\|$ by $\lambda = \frac{\delta}{2R}$. Then the probability density function of $\lambda$ is*

$$\begin{aligned} f_D(\lambda) &= \frac{2D\lambda^{D-1}}{B(\frac{1}{2}D+\frac{1}{2},\frac{1}{2}D+\frac{1}{2})}\int_\lambda^1 (1-z^2)^{(D-1)/2}dz \\ &\quad + C\lambda^{2D-1} \end{aligned} \tag{26}$$

*and the cumulative distribution of $\lambda$ is*

$$\begin{aligned} F_D(\lambda) &= (2\lambda)^D I_{1-\lambda^2}(\frac{1}{2}D+\frac{1}{2},\frac{1}{2}) \\ &\quad + I_{\lambda^2}(\frac{1}{2}D+\frac{1}{2},\frac{1}{2}D+\frac{1}{2}) + \frac{C\lambda^{2D}}{2D} \end{aligned} \tag{27}$$

*where $C$ is the constant of integration, $B(\cdot)$ is the beta function, and $I_x(p,q)$ is the incomplete beta function ratio $I_x(p,q) = \int_0^x z^{p-1}(1-z)^{q-1}dz/B(p,q)$.*

Fortunately, a remedy for the fact that these functions are somewhat painful to analyze is also provided by Hammersley:

**Lemma 4 ($f(\delta)$ for uniform in $D$, II [20])**

$$\lim_{D\to\infty} f_D(\lambda) = N_{\lambda^2}(\frac{1}{2}, 1/4D) \tag{28}$$

*where $N_x(\mu,\sigma^2)$ is the normal density of $x$.*

Now let's obtain a form where we can see more qualitatively what happens. For clarity let's first restate the result of Theorem 4 by changing variables: the asymptotic distribution of $\delta$ is $f_D(\delta) = N_\delta(R\sqrt{2}, \frac{R^2}{2D})$, or in the unit sphere $N_\delta(\sqrt{2}, \frac{1}{2D})$. It is remarkable that as $D$ increases, the variance of distances goes down as $O(\frac{1}{D})$ while the mean distance $E[\delta]$ stays about the same.

Using a few facts we can now compute *alpha* for the uniform distribution in high dimension. To simplify slightly we'll see what things look like for $k=1$:

**Lemma 5 ($\alpha$ wrt $f(\delta)$, high-$D$ uniform)** *For the 1-nearest-neighbor problem*

$$\alpha \approx \int_{-\infty}^{\sqrt{2}} N f_D(\delta)^N \left(\frac{a_1}{(1+b\delta)} + \frac{a_2}{(1+b\delta)^2} + \frac{a_3}{(1+b\delta)^3}\right)^{N-1} d\delta \tag{29}$$

**Proof:** For $k$-nearest-neighbor, we are interested in the density of the $k^{th}$ nearest distance. In general the density of the $k^{th}$ order statistic $\delta_{(k)}$ is

$$f_{(k)}(\delta) = N f(\delta) \begin{pmatrix} N-1 \\ k-1 \end{pmatrix} F(\delta)^{k-1}\{1-F(\delta)\}^{N-k} \tag{30}$$

Lemma 1 gives us that $\alpha = \mathbb{E}[F(\hat{\delta})]$, yielding

$$\begin{aligned}
\alpha &\approx \mathbb{E}[F(\delta^*)] \\
&\approx \mathbb{E}[\Pr(\delta^* > l)] \\
&\approx \mathbb{E}[1 - \Pr(\delta^* < l)] \\
&\approx 1 - \Pr(\delta^* < \mathbb{E}[l])] \\
&\approx \int_{-\infty}^{\mathbb{E}[l]} f_D(\delta^*) d\delta^* \\
&\approx \int_{-\infty}^{\sqrt{2}} N f_D(\delta)\left\{1 - F_D(\delta)\right\}^{N-1} d\delta^*.
\end{aligned}$$

To compute this we must evaluate $F_D(\delta) = \int_{-\infty}^{\delta} \frac{2D}{\sqrt{2\pi}} e^{-2D^2(z-\sqrt{2})^2} dz$. Unfortunately this integral has no known analytical solution. However, an approximation to the cumulative normal which is known to be accurate to within $10^{-5}$ for all arguments is $F_D(\delta) = 1 - [f_D(\delta)(\frac{a_1}{(1+b\delta)} + \frac{a_2}{(1+b\delta)^2} + \frac{a_3}{(1+b\delta)^3})]$, for certain constants $a_1$, $a_2$, $a_3$, and $b$ [1]. This gives

$$\begin{aligned}
\alpha &\approx \int_{-\infty}^{\sqrt{2}} N f_D(\delta)\left\{1 - [1 - f_D(\delta) g(\delta)]\right\}^{N-1} d\delta^* \\
&\approx \int_{-\infty}^{\sqrt{2}} N f_D(\delta)^N g(\delta)^{N-1} d\delta.
\end{aligned}$$

where $g(\delta) = \frac{a_1}{(1+b\delta)} + \frac{a_2}{(1+b\delta)^2} + \frac{a_3}{(1+b\delta)^3}$. ∎

Note that this expression involves both $N$ and $D$ simultaneously, both finite quantities in practice. This captures the fact that there is a practical interplay between $N$ and $D$ – it is insufficient to specify one without the other. For example large $D$ can still be $O(\log N)$ if $N$ is large enough. Likewise even for moderate $D$ runtime growth will be poor in the small $N$ regime.

This has hopefully illustrated that given a dataset $X$ and tree $\mathcal{T}_X$, we can compute the expected runtime. But how can we interpret this to make qualitative conclusions? We'll consider some conclusions about the behavior in high dimensionalities next.

# 6 Effect of Dimensionality

We'll now show how we can use the simple relationships we've developed to clear up two misconceptions which continue to proliferate in the literature concerning high-dimensional proximity searches with space-partitioning trees.

## 6.1 Effective Dimensionality

We've considered so far the canonical case of uniformly distributed data. Realistic datasets are quite different from this case for a simple reason: the dimensions are virtually always correlated to some degree, typically quite significantly. This is sensible because the measurements we tend to record tend to be dependent on the same underlying hidden variables. Any underlying dependence will be reflected in the data as correlation.

The result is that data can in general be assumed to lie on a manifold. This realization become prominent recently in the context of data analysis [29]. The mathematical definition of a manifold is the subject of topology, and can be found in [19]. The important thing about a manifold of dimension $D'$ for our purposes is that locally it acts like $\mathbb{R}^{D'}$. This allows us to simply replace $D$ with $D' \leq D$, but we must have a viable way of measuring $D'$. We can define $D'$ as the dimensionality which the data act as though they reside in, *i.e.* by:

$$F(\delta) = \delta^{D'}. \tag{31}$$

Then we have

$$D'(\delta) = \frac{\ln F(\delta)}{\ln \delta}. \tag{32}$$

Now to define a single dimension, let

$$D' \equiv \lim_{\delta \to 0} \frac{\ln F(\delta)}{\ln \delta}. \tag{33}$$

Note that this is identical for practical purposes to a standard definition for the notion of intrinsic dimension, called the *correlation dimension* [26], which uses the quantity $C(\delta)$ mentioned in § 3 where we have used $F(\delta)$.

To see why correlation of variables lead to an effective reduction in dimensionalty, consider two variables $X$ and $Y$ which have a joint bivariate normal distribution with correlation $\rho$. Given the value $X = x$, $Y$ is normally distributed with mean $\rho x$ and variance $1 - \rho^2$. From this we can see that as the correlation between $X$ and $Y$ goes to 1 or -1 (perfect correlation), the variance of $Y$ goes to zero. Pictorially this can be visualized as the loss of one dimension.

The upshot of this is that the explicit dimension $D$ of the space in which the points are embedded is a worst case – because the performance of space-partitioning trees is tied to the distribution of the data, it is sensitive to the data's underlying dimensionality $D' \leq D$.

## 6.2 Dimensionality and Rank

Hammersley's result has filtered rather roughly into the vernacular (without reference to or apparently knowledge of the original result), leading to the following conclusion by some authors: because all distances 'become indistinguishable' in high dimensionalities, $k$-nearest-neighbor queries become meaningless.

This can be quashed in two simple ways. First, the fact that practitioners currently utilize Euclidean distances in fairly high dimensionalities, for example in image retrieval applications, contradicts this idea – these systems could not exist if they exhibited nonsensical behavior. Second, it must be remembered that the task is *k-nearest*-neighbor, *i.e.* its essence is the relative *rank* of distance values, not their absolute magnitude. As long as the variance of distances is nonzero, there is an ordering on them – their absolute values are irrelevant.

A related misconception related to this is the notion that $(1+\epsilon)$ approximation is a solution to the curse of dimensionality, without further qualification. $(1+\epsilon)$ has been suggested prominently by [16], for example. It can also be performed by space-partitioning trees [2], as described in § 2. The qualification needed to make $(1+\epsilon)$ approximation meaningful for the $k$-nearest-neighbor problem is $\epsilon$ must somehow be chosen to preserve rank. Consider the percentage of the distances which are additionally allowed to be returned as the answer:

$$s \equiv Pr(\delta^* < \delta < \delta^*(1+\epsilon)) = F(\delta^*(1+\epsilon)) - F(\delta^*). \tag{34}$$

We can see that as $D \to \infty$, $s$ will grow until eventually all the points will be returned by the $k$-nearest-neighbor query. It is clear that greater speedup can be obtained by allowing most of the data to be in the answer returned, but it is not clear that the meaningfulness of the result is maintained. A more sensible notion of approximation would bound the shift in rank rather than the numerical value of the distance, *e.g.* instead of $\delta' = \delta^*(1+\epsilon)$, perhaps something like $\delta' = F^{-1}(F(\delta^*(1+\epsilon)))$.

# 7 Conclusion

By introducing some basic statistical ideas and machinery, we have developed a data-dependent analysis of typical space-partitioning methods for common proximity queries, which is more detailed than previous analyses. Among other things this has allowed us to reason more clearly about the behavior of such searches in high dimensionalities. These ideas formalize and put into perspective several previous observations and proposals throughout various literatures, *e.g.* [34, 13, 16].

## 7.1 Future Directions

Our analysis focused on the gross behavior of typical space-partitioning trees, considered as an entire class. In further refinement of this line of analysis we hope to arrive at the point where the *optimal* parameters $\Theta$ of a space-partitioning tree can be computed for a given dataset $X$, where optimality would be defined in terms of expected runtime over the distribution of $X$. Further refinement of this sort of analysis might finally allow analytical predictions of runtime, and might make measurements of the intrinsic dimension of databases a commonplace diagnostic tool, for example. Because this analysis is linked to the basic recurrence driving the discrete structure of the algorithm's runtime, we hope to extend it to more complex algorithms which use space-partitioning in different ways, corresponding to different recurrences. For example, the algorithms of [7] and [17], which use node-node comparisons rather than the point-node comparisons which were our concern in this paper. Finally, we hope that this simple analysis will spur similar use of statistical tools to develop data-dependent analyses for other algorithms and data structures.

### Acknowledgements

The author would like to thank Andrew Moore for incisive critique and discussions, which in particular led to Lemma 2, as well as his continued support of the goals of this work.

# References

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions.* Dover Publications, 1972.

[2] S. Arya and D. Mount. Approximate Nearest Neighbor Queries in Fixed Dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[4] J. L. Bentley. Multidimensional Binary Search Trees used for Associative Searching. *Communications of the ACM*, 18:509–517, 1975.

[5] C. Bohm. A Cost Model for Query Processing in High Dimensional Data Spaces. *ACM Transactions on Database Systems*, 25(2):129–178, 2000.

[6] D. Bremner, E. Demaine, J. Erickson, J. Iacono, S. Langerman, P. Morin, and G. Toussaint. Output-Sensitive Algorithms for Computing Nearest-Neighbor Decision Boundaries. In *Proceedings of the 2003 Workshop on Algorithms and Data Structures*, 2003.

[7] T. Brinkhoff, H. P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. ACM, 1993.

[8] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury Press, 1990.

[9] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, September 1997.

[10] P. Ciaccia, M. Patella, and P. Zezula. A Cost Model for Similarity Queries in Metric Spaces. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems : PODS 1998*. ACM Press, 1998.

[11] K. Clarkson. Nearest Neighbor Searching in Metric Spaces: Experimental Results for sb(S). , 2002.

[12] P. J. Diggle. *Statistical Analysis of Spatial Point Patterns, 2nd edition*. Arnold Publishers, London, 2003.

[13] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems : PODS 1994*. ACM Press, 1994.

[14] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[15] K. Fukunaga. *Introduction to Statistical Pattern Recognition, 2nd edition*. Academic Press, New York, 1990.

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc 25th VLDB Conference*, 1999.

[17] A. Gray and A. W. Moore. N-Body Problems in Statistical Learning. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (December 2000)*. MIT Press, 2001.

[18] A. G. Gray. *Bringing Tractability to Generalized N-Body Problems in Statisical and Scientific Computation*. PhD. Thesis, Carnegie Mellon University, Computer Science Department, 2003.

[19] V. Guillemin and V. Pollack. *Differential Topology*. Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[20] J. M. Hammersley. The Distribution of Distances in a Hypersphere. *Annals of Mathematical Statistics*, 21:447–452, 1950.

[21] N. Katayama and S. Satoh. The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. ACM, 1997.

[22] K. R. Mecke and D. Stoyan, editors. *Statistical Physics and Spatial Statistics: The Art of Analyzing and Modeling Spatial Structures and Pattern Formation*. Springer, 2000.

[23] S. Meiser. Point Location in Arrangements of Hyperplanes. *Information and Computation*, 106:286–303, 1993.

[24] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 1995.

[25] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Pearson Education POD, 1998.

[26] A. H. Nayfeh and B. Balachandran. *Applied Nonlinear Dynamics: Analytical, Computational, and Experimental Methods*. John Wiley & Sons, New York, 1995.

[27] M. O. Rabin. Probabilistic Algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, 1976.

[28] B. D. Ripley. *Spatial Statistics*. John Wiley & Sons, 1981.

[29] S. Roweis and L. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500), December 2000.

[30] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[31] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[32] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice-Hall, 2003.

[33] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986.

[34] R. F. Sproull. Refinements to Nearest-neighbor Searching. *Algorithmica*, 6:579–589, 1991.

[35] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.