DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Locally Lifting the Curse of Dimensionality for Nearest Neighbor Search

Peter N. Yianilos

ABSTRACT. Our work gives a positive result for nearest neighbor search in high dimensions. It establishes that radius-limited search is, under particular circumstances, free of the *curse of dimensionality*. It further illuminates the nature of the curse, and may therefore someday contribute to improved general purpose algorithms for high dimensions and for general metric spaces.

We consider the problem of nearest neighbor search in the Euclidean hypercube $[-1,+1]^d$ with uniform distributions, and the additional natural assumption that the nearest neighbor is located within a constant fraction R of the maximum interpoint distance in this space, i.e. within distance $2R\sqrt{d}$ of the query.

We introduce the idea of *aggressive pruning* and give a family of practical algorithms, an idealized analysis, and describe experiments. Our main result is that search complexity measured in terms of d-dimensional inner product operations, is i) strongly sublinear with respect to the data set size n for moderate R, ii) asymptotically, and as a practical matter, independent of dimension.

Given a random data set, a random query within distance $2R\sqrt{d}$ of some database element, and a randomly constructed data structure, the search succeeds with a specified probability, which is a parameter of the search algorithm. On average a search performs $\approx n^{\gamma}$ distance computations where n is the number of points in the database, and $\gamma < 1$ is calculated in our analysis. Linear and near-linear space structures are described, and our algorithms and analysis are free of large hidden constants, i.e. the algorithms perform far less work than exhaustive search – both in theory and practice.

1. Introduction

Finding nearest neighbors in Euclidean spaces of very low dimension is theoretically fast, and practical, using the notion of a Voronoi diagram [**Aur91**]. In moderate dimension, or in general metric spaces with *intrinsically* moderate dimension, recursive projection-decomposition techniques such as kd-trees [**FBS75, FBF77, BF79, Ben80**] and vantage-point techniques [**Uhl91b, Uhl91a, RP92, Yia93**] for general metric spaces of intrinsically low dimension, are effective.

1991 *Mathematics Subject Classification.* 68W01, 68W05, 68W40, 68P10.
Key words and phrases. Nearest neighbor search, kd-tree, curse of dimensionality.
Completed during 1999 while visiting the Princeton University computer science department.
A preliminary version appeared as [**Yia00**].

As dimension d grows large, these tree techniques perform significantly better than brute-force search only if the number of dataset points n grows exponentially in d. Or, in the case of Voronoi diagrams, if space increases exponentially.

The motivation for this work is the observation that, in practice, one is usually interested in nearest neighbors only if they are somewhat close to the query. The main contribution of this paper is the algorithmic idea of *aggressive pruning* and its analysis for the uniformly distributed hypercube. In this setting, with a natural definition of *somewhat close*, we show that the expected time complexity of finding nearest neighbors is invariant with respect to dimension[1] — and the space cost[2] is independent of d, and linear in n.

In a Euclidean hypercube the maximum distance between two points grows with $\sqrt{d}$. By *somewhat close* we mean within a neighborhood whose radius is a constant fraction R of this distance. A parameter $0 < p < 1$ controls the probability that a search will locate the nearest neighbor within this search domain. For each choice of R and p we calculate an exponent $\gamma < 1$ such that the search will perform on average $\approx n^\gamma$ distance computations, and this dominates the work performed. Notice that γ is independent of d.

The practical significance of our work is that search time is strongly sublinear given moderately large values for R and acceptable success probabilities. For example, searching $1,000,000$ points uniformly distributed in $[-1,+1]^{1000}$ with our experimental software, given $R = 0.1$, requires on average $\approx 30,000$ distance computations, and succeeds with probability 0.9988. In this example $\gamma \approx 0.78$ from our analysis. Arbitrarily high success probabilities can be obtained at the expense of distance computations.

We remark that this work was motivated by the author's recent work of [**Yia99**], where the objective is to build a data structure that provides worst-case sublinear-time radius-limited nearest neighbor search (independent of query) for a given dataset. With uniform distributions in Euclidean space, the resulting structures support search neighborhood of only $O(1)$ size, in contrast with those of this paper that scale linearly with the maximum interpoint distance.

Early work in nearest neighbor search is surveyed in [**Das91**]. There is a large literature on the search problem, much of it elaborating on a single fact: that certain projections from a metric space to $\mathbb{R}$ have the property that projected distances are dominated by those in the original space.

The two most important such projections are i) inner product with a unit vector in Euclidean space, and ii) distance from a chosen *vantage point*.[3] These ideas were recognized early on in work including [**BK73, Fuk75, Fuk90, FS82, Sha77**].

Taking the inner product with a canonical basis element in Euclidean space leads to the well-known *kd-tree* of Friedman and Bentley [**FBS75, FBF77, BF79, Ben80**]. They recursively divide a pointset in $\mathbb{R}^d$ by projecting each element onto a distinguished coordinate. Improvements, distribution adaptation, and incremental searches, are described in [**EW82**], [**KP86**], and [**Bro90**] respectively.

---

[1] Measured as d-dimensional inner product operations

[2] In addition to the space required to store the dataset itself

[3] The first of these may be viewed as the second *in the limit* as a vantage point moves toward infinity along the direction of the unit vector.

The search tree we build has essentially the same structure as a kd-tree built given randomly pretransformed data[4], and constrained to use the same cut coordinate for all nodes at a given tree level. Our criterion for pruning branches during search, and its analysis, are the primary contributions of this paper and distinguish our methods from kd-tree search.

More recently, the field of computational geometry has yielded many interesting results such as those of [**Vai89, Cla88, Cla87, FMT92**] and earlier [**DL76**].

Very recently a number of papers have appeared striving for more efficient algorithms with worst-case time bounds for the *approximate* nearest neighbor problem [**AMN+94, Kle97, KOR98, IM98**]. These exploit properties of random projections beyond the simple *projection distance dominance* fact mentioned above, and additional ideas to establish worst case bounds. Our work may be viewed as exploiting the fact that random projections of uniformly random data in a hypercube, and of neighborhood balls of radius proportional to $\sqrt{d}$, both have constant variance with respect to d. See also [**Cla97**] for very recent work on search in general metric spaces.

Several of the papers mentioned above include interesting theoretical constructions that trade polynomial space, and in some cases expensive preprocessing, for fast performance in the worst case. We remark that to be useful in practice, a nearest neighbor algorithm must require very nearly linear space — a stringent requirement. As datasets grow, even low-degree polynomial space becomes rapidly unacceptable.

For completeness, early work dealing with two special cases should be mentioned. Retrieval of similar binary keys is considered by Rivest in [**Riv74**] and the L_∞ setting is the focus of [**Yun76**]. Also see [**BM80**] for worst case data structures for the *range search* problem. Finally, recent works [**BOR99**] and [**CCGL99**] establish nontrivial lower bounds for nearest neighbor search.

In the following section we give construction and search algorithms specialized for our uniform setting. Section 3 gives a *concrete* analysis of these algorithms that include calculations of the applicable search time complexity exponent, and failure probability. Experiments are presented in section 4, which confirm in practice the dimensional invariance established by analysis. Finally, some directions for further work are mentioned in section 5.

2. Algorithms

2.1. Construction: A search tree is built with the set of data points as its leaves. It has essentially the same structure as a kd-tree built on data that has first been transformed to a random coordinate system. Construction time is easily seen to be $O(n \log n)$ and space is linear.

Construction proceeds recursively. Each interior node has as input a list of points. The root's list is the entire set. Associated with each interior node is a randomly selected unit vector u_i, where i denotes level (distance from the root). The number of such vectors is then equal to the depth of tree minus one (because there is no vector associated with a leaf). This set is constructed so as to be

[4]That is, where the dataset is first transformed to the coordinate system induced by a random basis. When the kd-tree *cuts* space using a single coordinate in this transformed setting, is then the same as cutting based the inner product of a data element with a particular basis vector.

orthonormal. First, random vectors are drawn, then they are orthonormalized in the usual way (Gram-Schmidt).

Construction of a node consists of reading its input list, computing the inner product of each element with the node's associated u_i, and adding the element to a *left* or *right* output list depending on its sign. In general the dividing point is chosen more intelligently, e.g. by computing the median of the inner product values. But in our uniform $[-1,+1]^d$ setting we just use zero for simplicity. Left and right children are then generated recursively. If a node's input list consists of a single element, then that node becomes a leaf and stores the element within it.

2.2. Search: The search is parameterized by a query q, a value $R \in (0,1)$ giving the proportional size of the search domain, and a probability $0 < p < 1$ that is related to the success rate we expect.

The inner product of q and each u_i is first computed. Next the positive threshold distance $\ell = \Phi^{-1}_{4R^2}(p)$ is computed, where Φ denotes the cumulative distribution function for a normal density with the variance indicated by subscript.

Search proceeds recursively from the root. For a node at level i the value $< q, u_i >$ is examined. If it is less than ℓ then the left child is recursively explored. If it exceeds $-\ell$ then the right child is explored. Notice that when $< q, u_i > \in (-\ell, +\ell)$ both children are explored. When a leaf is encountered, the distance is computed between the element it contains and q.

This decision rule is centered at zero because of our particular uniform $[-1,+1]^d$ setting, but is easily translated to an arbitrary cut point, e.g. the median of the projected values.

After each distance computation $d(q,x)$ is performed the proportion $d(q,x)/2\sqrt{d}$ is computed. If smaller than R, then R is reduced and ℓ recomputed.

This concludes the description of our search algorithm and we now briefly discuss related issues and extensions.

An important idea in kd-tree search involves the computation of the minimum distance from the query to the subspace corresponding to a node to be explored. If this distance grows beyond the radius of interest, the node is pruned. We do not, however, include it in our analysis or experimental implementation because in our high dimensional setting, in the absence of exponentially many data elements, this idea has vanishingly little effect. Intuitively, this is because the search tree is not nearly deep enough for the minimum distance to grow larger than the search radius.

The analogue of this kd-tree idea in our setting is an alternative version of our algorithm above that slightly reduces ℓ while descending through interior nodes to reflect the fact that the distribution of data elements within a ball about the query is no longer uniform. But, again, this is a second order effect.

Finally we remark that the ℓ-cutoff approach taken above might be replaced with an entirely probabilistic pruning scheme that passes probabilities from our analysis down to each child during search. The probabilities upward to the root are then multiplied and search continues downward until the result falls below a specified threshold.

3. Analysis

We assume both data points and queries are uniformly distributed within the hypercube $[-1,+1]^d$. The Euclidean distance metric applies and the maximum

distance between two points in this space is $2\sqrt{d}$. We consider the problem of finding the nearest neighbor of a query q within some distance δ that is a constant proportion of the maximum interpoint distance, i.e. $\delta = 2R\sqrt{d}$ with $0 < R < 1$.

Any unit vector u gives rise to a projection π_u mapping $x \in \mathbb{R}^d$ into $\mathbb{R}$ defined by $< x, u >$. It is immediate that distances in the range of this projection are dominated by distances in the domain. It is this fact that kd-trees exploit to prune branches during branch-and-bound search.

If τ represents the distance to the nearest neighbor encountered so far during a search, then this fact implies that every member of the ball of radius τ centered at the query q maps under any π_u into the interval $[\pi_u(q) - \tau, \pi_u(q) + \tau]$. So kd-tree search may confidently disregard any data points that project outside of this interval. Unlike the kd-tree, which finds nearest neighbors with certainty, we will consider randomized constructions that succeed with some specified probability.

Since δ scales up linearly with $\sqrt{d}$, the interval grows $[\pi_u(q) - \delta, \pi_u(q) + \delta]$ too, and soon the kd-tree can *confidently* prune almost nothing, and performs a nearly exhaustive search.

But in our uniformly random setting we will show that the δ ball about q projects to a distribution about $\pi(q)$ having constant variance. This is why it is possible to continue to *probabilistically* prune just as effectively even as $d \to \infty$. Since dimension does not affect our ability to prune, we have lifted the curse of dimensionality in our setting.

We now proceed with the description and idealized analysis of our algorithm with the following proposition, which establishes two elementary but key dimensional invariants

PROPOSITION 3.1. *i) Let u be a random unit vector, and let X denote a random dataset in our setting, then the one dimensional set of values $\pi_u(X)$ has mean zero and variance $1/3$ – independent of dimension – where both u and X are random variables. ii) Consider any $q \in \mathbb{R}^d$ and let r denote a random vector located on the surface of a ball centered at q of radius $2R\sqrt{d}$. Then for any unit vector u, the distribution of values $\pi_u(r)$ has mean $\pi_u(q)$ and variance $4R^2$ – independent of dimension.*

PROOF. The variance of each component of u is $1/d$ since $< u, u >= 1$ by definition and the components are i.i.d. Consider a random element $x \in [-1, +1]^d$. Here a simple integration establishes that the variance of each component is $1/3$. So the variance of each term of $< u, x >$ is $1/3d$, and that of the entire inner product is then $1/3$ as required. Now each component of u and x has mean zero so that $< u, x >$ has mean zero — and part i) is established.

$\pi_u(r)$ is centered at $\pi_u(q)$ by linearity of inner product. So we may assume without loss of generality that $q = 0$. Now the inner product of two random unit vectors is easily seen to have variance $1/d$. Scaling one of them by a factor of $2R\sqrt{d}$ increases the variance by a factor of $4R^2d$, so that it becomes $4R^2$ as required by part ii). □

Now fix some unit vector u and consider the query's projection $\pi_u(q)$. Then by Proposition 3.1, and ignoring hypercube *corner effects*, the projection of the data points within the domain of search are distributed about $\pi_u(q)$ with variance no greater than $4R^2$.

Because distances between projected points are dominated by their original distance it is clear that $|\pi_u(x) - \pi_u(q)| < 2R\sqrt{d}$ for any x in the search domain.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.