Achieving Spatial Adaptivity while Finding Approximate Nearest Neighbors

Jonathan Derryberry Don Sheehy Daniel D. Sleator Maverick Woo*

Abstract

We present the first spatially adaptive data structure that answers approximate nearest neighbor (ANN) queries to points that reside in a geometric space of any constant dimension d. The L_t-norm approximation ratio is $O(d^{1+1/t})$, and the running time for a query q is $O(d^2 \lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a suitably-sized box containing p and q. Our data structure has $O(dn)$ size and requires $O(d^2 n \lg n)$ preprocessing time, where n is the number of points in the data structure. The size of the bounding box for δ depends on d, and our results rely on the Random Access Machine (RAM) model with word size $\Theta(\lg n)$.

1 Introduction

The problem of finding the nearest neighbor to a query point is a fundamental data structure problem with numerous applications in areas such as computational geometry and machine learning. Unfortunately, finding the *exact* nearest neighbor seems difficult in dimensions 3 or higher as there is no known data structure that achieves nearly-linear preprocessing time and nearly-logarithmic query time. Hence, researchers turned to the approximate version of the problem, achieving significant performance gains by permitting the data structure to return merely a *near* neighbor, a point whose distance to the query is at most a constant times the distance from the nearest neighbor.

A large number of papers have sought to improve the performance of ANN data structures (see references in [2]), but none has shown how nonrandom patterns in query sequences might be exploited to improve running time. Examples of exploiting such nonrandomness abound in the 1D version of the exact nearest neighbor problem, for which data structures whose query performance depends upon the locality of queries in space and/or time have long been known (see references in [4]). Results in 2D, however, have only started to appear in recent years. For example, [12, 3] have shown how to exploit temporal locality in a random query sequence if the distribution is known, whereas [10, 13] have shown how to achieve dynamic-finger-like bounds in the 2D point search and point location problems.

Contribution. We extend ideas from [14, 9] to present the first ANN data structure to achieve, in any constant dimension, a provable speedup according to the degree of spatial locality in the query sequence. More specifically, in the RAM model with word size $w = \Theta(\lg n)$, we are given a set P of n points in d-dimensional space, each represented as a tuple of d words. We show how to preprocess P in $O(d^2 n \lg n)$ time to build an $O(dn)$-sized data structure that serves a sequence of queries for which each query q costs $O(d^2 \lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a suitably-sized box containing p and q. The L_t-norm approximation ratio is $O(d^{1+1/t})$, and the bounding box size for δ depends on d. While our description of the data structure does not include insertions or deletions, they are straightforward to support as long as spatially adaptive time bounds are not required.

Outline. Section 2 briefly discusses related work, including results in 1D on which we rely as a black box and Section 4 discusses the notion of "distance" we use in this paper in the context of other notions of distance that have been used by previous spatially adaptive data structures. We describe and analyze the data structure and the search algorithm in Sections 5 and 6 and we conclude in Section 7.

2 Related Work

Finger Search in 1D. There exists a variety of 1D nearest neighbor data structures that exploit spatial locality in a query sequence. Here we merely highlight two optimal results and we refer the reader to these two papers for references to previous works. Let q be the number of points between the previous and current queries. For Pointer Machines, Brodal *et al.* [6] have designed finger search trees with $O(\lg q)$ query time and $O(1)$ update time. With the added power of the RAM model, Andersson and Thorup [1] have shown how to achieve a query time of $O(\sqrt{\lg q / \lg \lg q})$. The running times above are all worst-case.

In this paper, we will be making use of a 1D finger search data structure as a black box. Any finger search data structure can be used as long as it does *not* restructure during a search. (This requirement will be explained in Section 5.) When updates are not required,

*Department of Computer Science, Carnegie Mellon University, jonderry,dsheehy,sleator,maverick@cs.cmu.edu. The last author was sponsored by the National Science Foundation under contract no. EIA-9706572, CCR-0122581 and CNS-0435382 and Appalachian Regional Commission under contract no. CO-14574. This work was partially supported by the National Science Foundation under grant number CCF-0635257.

we can simply use a sorted array as our black box and perform finger search in the obvious manner. Only when updates are required do we need to employ more sophisticated data structures such as level-linked 2-3 trees by Brown and Tarjan [7] or several other finger search data structures that were designed later.

Finger Search in 2D. Though most of the work on spatially adaptive data structures is restricted to 1D, there has been some recent work on developing such distance-sensitive data structures in 2D. In particular, Demaine *et al.* [10] have shown how to preprocess a set of points P to permit a sequence of *membership* queries to points in P with a time bound of $O(\lg \delta_{PPS}(p,q))$, where $\delta_{PPS}(p,q)$ represents the distance between the previous query p and the current query q as measured by counting the number of points in a triangle-shaped region that contains both p and q. Subsequently, Iacono and Langerman [13] have shown how to achieve a similar distance-sensitive bound for point location in 2D.

Previous Work on ANN. The literature on the ANN problem is rich and we refer the reader to [11, Chapter 11] for numerous references. However, in the next section we will expand on the two previous works [14, 9] that are most relevant to this paper.

3 ANN and Space Filling Curves

Space filling curves (SFCs) provide a natural mapping from a high-dimensional space to an 1D curve and the ordering of points on SFCs has been used extensively as a meaningful order of points. In particular, the problem of finding ANNs and related proximity problems can be solved by SFC methods [14, 9]. Here we will describe a well-known algorithm for computing ANNs using SFCs from Liao *et al.* [14]. This algorithm is based on a similar algorithm that uses quadtrees. The quadtree version is due to Chan [8] and can be seen as a derandomization of an algorithm by Bern [5].

Let us consider a particular SFC known as the Z-order curve. Points are easily mapped onto this curve by a simple bit shuffling operation. Let $p_{i:j}$ represent the jth bit of the ith coordinate of point $p \in \mathbb{Z}^d$, assuming that each coordinate can be represented in a w-bit word. The shuffle operation $\sigma : \mathbb{Z}^d \to \mathbb{Z}$ is defined as the binary number $\sigma(p) = p_{1:w} \cdots p_{d:w} \cdots p_{1:1} \cdots p_{d:1}$, which we call the "shuffled value of p". For any pair of points p, q, we can order p and q on the curve by comparing their shuffled values. For a set of points $P = \{p_1, \ldots, p_n\}$, their Z-order is exactly their order in an in-order traversal of a quadtree constructed from the points in P. Figure 1 depicts this relationship and gives some intuition for the name "Z-order".

The algorithm for ANN is as follows. Observe that the Z-order depends on the placement of the origin and that for a particular Z-order, the nearest neighbor to a

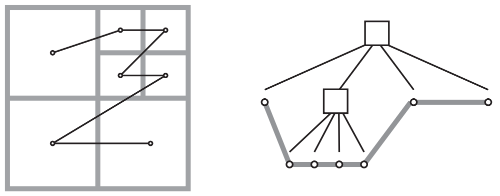
Figure 1: The in-order traversal of quadtree leaves corresponds to the ordering of the points on Z-order curve.

query is not necessarily the predecessor or the successor. Fortunately, one can show that there is a shift of the origin such that either the predecessor or the successor in the resulting Z-order is an ANN. In particular, consider a set of s shifts $v^{(j)} = (j/s, \ldots, j/s)$ for $j = 0, 1, \ldots, (s-1)$ and let s be $(d+1)$. Construct a set of search structures, one for each of the $(d+1)$ shifts. We compare p to q under the shift $v^{(j)}$ by comparing $\sigma(p+v^{(j)})$ to $\sigma(q+v^{(j)})$ and insert each input point into each of the structures. For a query q, do all $(d+1)$ searches for q and return the closest of the results. This algorithm gives an $O(d^{\frac{3}{2}})$ approximation in L_2 as shown in [8].

Chan's Comparison Procedure. We remark that in both the algorithm above as well as our algorithm in Section 5, we merely need the ability to compare the shuffled values. This can be done using a clever comparison procedure by Chan [9] that, given two points, compares their shuffled values using $O(d)$ exclusive-or word operations. This technique allows us to avoid computing and storing shuffled values at a cost of $O(d)$ slowdown and also mitigates the concern that a shuffled value may not fit inside a word.

4 Combinatorial Distance Measures for Point Sets

The goal of a geometric data structure supporting the dynamic finger property is to be distribution-sensitive so that sequences of geometrically close queries can be answered quickly. The ideal guarantee is that a query for a point q following a query for a point p takes time $O(\lg \mathsf{dist}(p,q))$ for some distance measure dist. For 1D problems, the distance between two points is simply the number of points between them. Unfortunately, such a combinatorial distance measure has no ready analogue in geometric spaces of dimension 2 or higher.

As the purpose of the finger p is to limit the search space to points that are geometrically close to p and q, a natural way to define a distance measure is to count the number of points in a suitable restriction of the search space. This intuition guided previous works in geometric finger search to use the notion of a region counting distance, in which $\mathsf{dist}(p,q)$ is defined as the number of input points in some carefully defined region containing both p and q [10, 13]. Formally, a region counting distance is defined by a triple (x, y, R) where

x and y are points and R is a region whose membership can be decided in $O(1)$ time. Given this triple, $\mathsf{dist}(p, q)$ is the number of points in the image of R under the affine transformation that takes x to p and y to q. The two previous works [10, 13] only applied to 2D where this transformation is unique.

Here we propose a new combinatorial distance measure similar to a region counting distance. Let c be a constant to be chosen later and let U be some axis-aligned box containing p and q with side length $c|p-q|_\infty$. The distance is defined as $\delta(p, q) = |P \cap U^*|$, where U^* is the choice among all such U that maximizes the distance measure. It should be clear that the points counted all have the desired property that their distances from p and q are bounded by a constant times the distance between p and q and therefore we have a distance measure that captures a notion of geometric locality in any dimension.

5 The Data Structure and Search Algorithm

Compared to the data structure in Section 3, our data structure uses $(2d+1)$ shifts versus $(d+1)$, and therefore consists of $(2d + 1)$ 1D structures. Each of the 1D structures stores pointers to the input points in its nodes using the Z-ordering defined by the corresponding shift. Note that we do not store the keys, which are the shuffled values of the shifted points. Instead, we use Chan's comparison procedure on the points directly. For each input point x_i, we also maintain a circularly linked list comprising the $(2d+1)$ nodes that represent x_i in the 1D structures. The preprocessing time is $O(d^2 n \lg n)$ since there are $(2d + 1)$ 1D structures to build, each requiring $O(n \lg n)$ comparisons that use $O(d)$ word operations each. The size of the data structure is $O(dn)$.

Given a query q, a search for an ANN is straightforward. Let p be the result of the previous query. We perform $(2d + 1)$ finger searches from p in parallel, one for each shift. Let $x_1, \ldots, x_{d+1}$ be the results found by the first $(d + 1)$ searches that complete. We return the x_i that is closest to q as an ANN and abandon the other d searches. Finally, we update the 1D structures to prepare for the next query by re-establishing their finger pointers to point at x_i using the circularly linked list associated with x_i. (The reason why we do not allow restructuring during a search in the 1D structures is to support the abandon and the re-establish steps.)

6 Algorithmic Guarantees

6.1 Centering Points in Quadtree Boxes

Chan [8] proved that $(d+1)$ shifts of the quadtree suffice to guarantee that for any point p and scale r, there is a shift that puts p roughly in the center of the quadtree square corresponding to that shift at scale r. This is the key lemma to prove that some quadtree will return an $O(d^{\frac{3}{2}})$-ANN.

For finger search to work, we need it to be true that for two different points, p, p' and two different scales r, r', there is a shift that puts p near the center of a square at scale r and p' near the center of a square at scale r'. Two guarantees rather than one are needed so that both the finger search will run quickly *and* the result will be a good approximation. The usual $(d + 1)$ shifts would suffice if we were willing to accept only one of these guarantees, but we will show that $(2d + 1)$ shifts suffice to get both.

To maintain consistency with the work we are extending, we assume the input points are scaled to finite precision real numbers in $[0, 1)^d$.

Say that p is *α-central* at scale r if for all $i = 1, \ldots, d$, we have $(p_i + \alpha) \bmod r \geq 2r\alpha$. The following is a slight extension of [8, Lemma 3.3] and its proof follows the same pattern as the original.

Lemma 1 *Let $s > d$ be an odd integer representing the number of shifts $v^{(j)} = (\frac{j}{s}, \ldots, \frac{j}{s})$, $j = 0, \ldots, (s-1)$. For a point $p \in [0, 1)^d$, and scale $r = 2^{-\ell}$, there are at most d shifts $v^{(j)}$ such that $p + v^{(j)}$ is not $\frac{1}{2s}$-central at scale r.*

Proof. We will prove that at most one shift is bad for each dimension. Formally, we prove that for each $i \in \{1, \ldots, d\}$, there is at most one shift $v^{(j)}$ such that

$$\left(p_i + \frac{j}{s} + \frac{1}{2s}\right) \bmod r < \frac{r}{s}, \qquad (1)$$

or equivalently, by multiplying through by s/r,

$$\left(2^\ell s p_i + 2^\ell j + 2^{\ell-1}\right) \bmod s < 1. \qquad (2)$$

Suppose on the contrary that we have distinct $j, j' \in \{0, \ldots, s-1\}$ for which (2) holds. Letting $z = 2^\ell s p_i + 2^{\ell-1}$, we have $(z+2^\ell j) \bmod s < 1$ and also $(z+2^\ell j') \bmod s < 1$. So, for integers q, q' and remainders $0 \leq x, x' < 1$, the above inequalities imply $z + 2^\ell j = qs + x$, and $z+2^\ell j' = q's+x'$. It follows that $2^\ell(j-j')-(q-q')s = x-x'$. Since the LHS here is an integer and $0 \leq x, x' < 1$, it must be that in fact $x = x'$ and thus $2^\ell j \equiv 2^\ell j'$ (mod s). We can divide both sides of this congruence by 2^ℓ because 2^ℓ and s are relatively prime (s is odd). The result is $j = j'$, a contradiction. □

6.2 Query Time and Approximation Ratio

To analyze query time we must first choose the constant for our distance measure. Say $\delta(p, q) = |\{x \in P : |x - p|_\infty \leq (8d + 4)|p - q|_\infty\}|$.

Using Lemma 1 for a scale r, we know that of the $(2d + 1)$ shifts, p is $\frac{1}{4d+2}$-central in at least $(d + 1)$ shifts. In particular, we are interested in the smallest scale $r \geq (4d + 2)|p - q|_\infty$. At this scale, p has distance at least $|p - q|_\infty$ from the boundary of any box B_i for which it is $\frac{1}{4d+2}$-central. So q is also in each of these $(d + 1)$ boxes B_i. The SFC touches each point in a box B before leaving, so each finger search will take time

$O(d \lg |P \cap B_i|)$. As all points in $P \cap B_i$ are counted in $\delta(p, q)$, we see that $(d+1)$ different shifts are guaranteed to finish in $O(d^2 \lg \delta(p, q))$ time. Choosing the best of these $(d+1)$ answers can be done in $O(d^2)$ time. Thus, the total running time is $O(d^2 \lg \delta(p, q))$.

Second, we need to show that the returned point is indeed a good ANN and this also follows from Lemma 1. Let q^* be the nearest neighbor of q. The lemma implies that at the smallest scale $r' \geq (4d+2)|q - q^*|_\infty$, there can be at most d shifts for which q is not $\frac{1}{4d+2}$-central. Therefore one of the $(d+1)$ shifts that finished searching found q in a box for which it is central at scale r'. The search returned a point x in that box, and thus $|q-x|_\infty < (8d+4)|q-q^*|_\infty$. So, x is an $O(d)$-ANN in the L_∞ norm and therefore an $O(d^{1+1/t})$-ANN in the L_t norm.

7 Concluding Remarks

In this paper, we showed how to achieve spatial adaptivity for the ANN problem in any constant dimension by extending prior work based on SFCs. Here we describe an enhancement and discuss some future research.

Using the Quadtree to Speed Up Search. Recall that the Z-ordering of the input points corresponds to the in-order traversal of the leaves in a quadtree. For any two points p, q in a quadtree, there is a unique path along the link structure. Let us call the length of this path the *quadtree distance*. The quadtree distance approximates the log of the Euclidean distance after normalization by the empty space around p and q. One can imagine building a quadtree that supports finger search in time proportional to the quadtree distance by walking up and down within the quadtree.

However, observe that even when using a *compressed quadtree*, in which paths of degree two nodes of the tree are collapsed, the quadtree distance may still be linear in the number of input points. Furthermore, even with a good shift, this distance could still be significantly worse than the distance computed by the measure in Section 4. On the other hand, it is also not hard to construct examples in which the quadtree distance is $o(\lg \delta(p, q))$. As an example, consider the effect of adding a dense set of points between p and q while p and q each remains in its own, relatively sparse region. This example not only shows that there is no strict ordering of geometric and combinatorial distance measures, but it also suggests that one can exploit using *both* the structure of the quadtree and our data structure to speed up searches.

Future Work. A good enhancement to make in the future would be to improve the approximation ratio to $(1+\epsilon)$, though it is not clear how to do this without the exponential blowup incurred by analogous enhancements to other SFC-based data structures for ANN. A more modest enhancement would be to shrink the distance measure so that the distance between two successive queries to p and q would be the number of points inside a smaller box that more tightly bounds p and q. We can achieve this to a degree by using a constant number of shifts *independently* in each dimension at the expense of an exponential factor in d to space usage, but perhaps there other methods. We may also try to more thoroughly exploit the power of RAM. For example, if we use the finger search structure by Andersson and Thorup [1] as the 1D structure, then we get an improved running time. However, we do not know how to avoid computing the shuffled values explicitly when using this. Finally, a general direction for future work would be to extend other ANN techniques—or even algorithms for serving exact nearest neighbor queries in high dimensions—to allow spatial adaptivity, temporal adaptivity, or a combination of the two.

References

[1] A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):Article 13, 2007.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[3] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Transactions on Algorithms*, 3(2):Article 17, 2007.

[4] M. Badoiu, R. Cole, E. D. Demaine, and J. Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.

[5] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(2):95–99, 1993.

[6] G. S. Brodal, G. Lagogiannis, C. Makris, A. K. Tsakalidis, and K. Tsichlas. Optimal finger search trees in the pointer machine. *Journal of Computer and System Sciences*, 67(2):381–418, 2003.

[7] M. R. Brown and R. E. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM Journal of Computing*, 9(3):594–614, 1980.

[8] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.

[9] T. M. Chan. Closest-point problems simplified on the RAM. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 472–473, 2002.

[10] E. D. Demaine, J. Iacono, and S. Langerman. Proximate point searching. *Computational Geometry*, 28(1):29–40, 2008.

[11] S. Har-Peled. *Geometric Approximation Algorithms.* (working draft), 2008.

[12] J. Iacono. Optimal planar point location. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 340–341, 2001.

[13] J. Iacono and S. Langerman. Proximate planar point location. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 220–226, 2003.

[14] S. Liao, M. A. Lopez, and S. T. Leutenegger. High dimensional similarity search with space filling curves. In *17th International Conference on Data Engineering*, pages 615–622, 2001.