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. The running time for a query q is $O(\lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a reasonably sized box containing p and q. Moreover, the data structure has $O(n)$ size and requires $O(n \lg n)$ preprocessing time, where n is the number of points in the data structure. The constant factors in the above bounds depend on the dimension d. For points in d dimensions, the L_t-norm approximation ratio is $O(d^{1+\frac{1}{t}})$. Our results use the RAM model with words of 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 hard when the dimension is 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 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 either in space or time or both have long been known (see references in [4]). Additionally, a few results have been achieved in 2D. For example, [12, 3] have shown how to exploit temporal locality in a random query sequence if the distribution is known, while [10, 13] have shown how to achieve dynamic-finger-like bounds in the 2D point search and point location problems.

*Department of Computer Science, Carnegie Mellon University, jonderry,dsheehy,sleator,maverick@cs.cmu.edu

Our ANN data structure is the first 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 $\Theta(\lg n)$, we show how to preprocesses a set of n points in d-dimensional space in $O(n \lg n)$ time to build an $O(n)$-sized data structure that serves a sequence of queries for which each query q costs $O(\lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a reasonably sized box containing p and q. The big-O notation suppresses constants that depend on d and the approximation ratio is $O(d^{1+\frac{1}{t}})$ in the L_t-norm. No insertions or deletions are allowed, though it is straightforward to make our data structure dynamic as long as we do not require spatially adaptive bounds for updates. Our method is an extension to the ideas presented in [15, 7].

Section 2 briefly discusses related work, including results in one dimension, which we will rely on as a black box. Section 4 discusses the notion of "distance" that we will use in this paper in the context of other notions of distance that have been used by spatially adaptive data structures in the past. Section 5 describes the data structure and search algorithm, proves their correctness and running time, and describes a few enhancements that can be made. Section 8 describes some future work.

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. For example, Brodal *et al.* gave an optimal design for finger search trees in the pointer machine model [6]. Additionally, Cole showed that splay trees, a simple heuristic for self-adjusting binary search trees, satisfy the Dynamic Finger Theorem [9]. With the added power of the RAM model, Andersson and Thorup [1] showed how to achieve an optimal query time of $O(\sqrt{\lg q / \lg \lg q})$ where q is defined to be the number of points between the previous and current queries, and Kaporis *et al.* showed how to achieve a query time of $O(\lg \lg q)$ for a large class of input distributions [14].

In this paper, we will be making use of a 1D finger search data structure as a black box. However, due to the way in which the black box finger search data structure will be used, we require that it is *not* self-adjusting, which rules out splay trees, for example. This require-

ment will become clear in Section 5. Also, we cannot use the result of Kaporis *et al.* since we do not want to restrict the class of instances on which our data structure works. Either of the other two 1D finger search data structures mentioned above will work for our purposes. In fact, if we do not require updates, we can simply use a sorted array as our black box and perform finger search in the obvious manner.

Finger Search in 2D. Though most of the work on spatially adaptive data structures is restricted to 1D, there has been some work on developing such distance-sensitive data structures in 2D. In particular, Demaine *et al.* [10] showed 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 a count of the number of points in a triangle-shaped region containing both p and q. Iacono and Langerman [13] subsequently presented a result that achieves 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 Section 3 we will expand on the two works [15, 7] that are most relevant to this paper.

3 Approximate Nearest Neighbors and Space Filling Curves

Space filling curves (SFCs) provide a natural mapping from a high-dimensional space to a 1D curve. The ordering of points on SFCs has been used extensively to give a meaningful ordering to points in geometric spaces. In particular, the problem of finding ANNs and related proximity problems can be solved by SFC methods [15, 7]. In this section, we describe a well-known algorithm for computing ANNs using SFCs from Liao *et al.* [15]. This algorithm is based on another similar algorithm that uses quadtrees. The quadtree version is due to Chan [8] and is a derandomized version of an algorithm of Bern [5]. We also describe the connection between quadtrees and SFCs.

Let us consider a particular SFC known as the Z-order curve. Points represented as d-tuples of w-bit words are easily mapped onto the Z-order 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 has bounded precision. The shuffle operation $\sigma : \mathbb{Z}^d \to \mathbb{Z}$ is defined as the binary number $\sigma(p) = p_{1:w} \cdots p_{d:w} \cdots p_{1:1} \cdots p_{d:1}$. For any pair of points p, q, we can order p and q on the curve by comparing $\sigma(p)$ and $\sigma(q)$. 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".

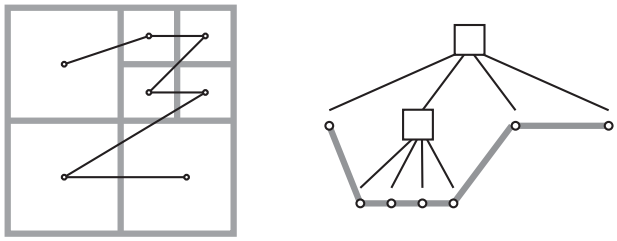

Figure 1: The in-order traversal of the quadtree leaves corresponds to the ordering of the points on the Z-order curve.

The algorithm for ANN is as follows. Observe that the Z-order depends on the placement of the origin. Note that for a particular Z-order, the nearest neighbor to a query is not necessarily the predecessor or the successor. However, we 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 $v_j = (j/s, \cdots, j/s)$ for $j = 0, 1, \cdots, (s-1)$ 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 [8].

4 Combinatorial Distance Measures for Geometric Points

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 desired guarantee is that a query for a point q following a query of 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 the difference in their ranks. This combinatorial distance measure has no ready analogue in geometric spaces of dimension 2 and greater.

The purpose of the finger p is to limit the search space to points that are geometrically close to p and q. Thus, a natural way to define a distance measure between p and q is to simply count the number of points in a reasonable restriction of the search space. This intuition guided previous work 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 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 computed 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. Previous work only applied only to 2D where this transformation is unique.

We propose a new combinatorial distance measure similar to a region counting distance. Let U be some axis-aligned box containing p and q with side length $c|p-q|_\infty$. The distance is defined as $\delta(p, q) = \max_U |S \cap U|$. Clearly, the points counted all have the desired property that their distances from p and q are bounded by the distance between p and q multiplied by a constant that depends on d. Thus, we have a distance measure that captures a notion of geometric locality.

5 The Data Structure and Search Algorithm

Our data structure consists of $(2d+1)$ 1D search structures, where d is the dimension. Each one corresponds to the Z-ordering defined by one of the $(2d+1)$ shifts.

Given a query q, a search for an ANN is straightforward. Let p be the result of the previous query. Similar to Chan's algorithm, we perform $(2d+1)$ finger searches from p in parallel, one for each shift, where d is the dimension. Let $x_1, \ldots, x_{d+1}$ be the results found by the first $(d+1)$ searches that complete. Abandon the other d searches and return the x_i closest to the query q.

After returning an approximate nearest neighbor x_i, we need to update the 1D search structures to prepare for the next query. This involves setting the finger pointer in each structure to point at x_i. To do this quickly, we augment the data structure with a circularly linked list containing all the nodes representing x_i in each of the 1D search structures.

6 Algorithmic Guarantees

6.1 Centering points in quadtree boxes

Chan 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 of scale r. This is the key lemma to prove that some quadtree will return an $O(d^{\frac{3}{2}})$-approximate nearest neighbor.

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. 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 $i = 1, \cdots, d$, we have $p_i + \alpha \mod r \geq 2r\alpha$.

The following is a slight generalization of the Lemma by Chan in [8]. The 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

$$p_i + \frac{j}{s} + \frac{1}{2s} \mod r < \frac{r}{s}, \tag{1}$$

or equivalently, by multiplying through by s/r,

$$2^\ell s p_i + j2^\ell + 1/2 \mod s < 1. \tag{2}$$

Suppose on the contrary that we have distinct $j, j' \in \{0, \ldots, s-1\}$ for which Equation 2 holds. Letting $z = 2^\ell s p_i + 1/2$, we have $z + j2^\ell \mod s < 1$, and $z + j'2^\ell \mod s < 1$.

So, for integers q, q' and remainders $0 \leq x, x' < 1$, the above inequalities imply $z + j2^\ell = qs + x$, and $z + j'2^\ell = q's + x'$. 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 $j2^\ell \equiv j'2^\ell \pmod s$. 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

To analyze query times 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 $(2d+1)$ shifts, we will have at least $(d+1)$ for which p is $\frac{1}{4d+2}$-central. 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 $O(\lg |P \cap B_i|)$. Observing that 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 $O(\lg \delta(p, q))$ time. Choosing the best of these $(d+1)$ answers takes constant time. Thus, the total running time is $O(\lg \delta(p, q))$.

Second, we need to show that the returned point is indeed a good ANN. 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 was 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 a $O(d)$-approximate nearest neighbor in the L_∞ norm. It follows that x is a $O(d^{1+\frac{1}{t}})$-approximate nearest neighbor in the L_t norm.

7 Enhancements to the algorithm

Using the quadtree to speed up search. We have observed that the Z-order of the input is the same as 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 linear in the quadtree distance by walking up and down within the quadtree.

Observe that even using *compressed quadtree*, in which paths of degree two nodes of the tree are collapsed, the quadtree distance may still be linear. Furthermore, even with a good shift, this distance could be $O(\delta(p,q))$, which is exponentially worse than the SFC method. However, it is not hard to construct examples in which the quadtree distance is $o(\lg \delta(p,q))$. As an example, consider a dense set of points lying between p and q while p and q lie in a relatively sparse region. This example shows that there is no strict ordering of geometric and combinatorial distance measures.

Speeding up Finger Search by Exploiting RAM. Andersson and Thorup [1] have shown a RAM data structure that supports finger search in $O(\sqrt{\lg q / \lg\lg q})$ time, where q is the rank difference. If we use their data structure for each of the 1D structures, we get a matching running time. Note, however, that this requires us to explicitly compute the shuffle operation and store the shuffled address for each point, whereas one may choose to use the bit procedures by Chan [7] to perform comparisons without actually storing these addresses.

8 Conclusions and Future Work

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.

There are a variety of avenues for future research related to the work in this paper. The most obvious potential enhancement to this work would be to improve the approximation ratio to $(1+\epsilon)$.

Another enhancement to this work 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. This can be achieved 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. Another 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. Tight(er) worst-case bounds on dynamic searching and priority queues. In *Proceedings of the 32th ACM Symposium on Theory of Computing*, pages 335–342, 2000.

[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):1–17, 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, 26 Feb. 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] T. Chan. Closest-point problems simplified on the ram. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA-02)*, 2002.

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

[9] R. Cole. On the dynamic finger conjecture for splay trees, part II: The proof. *SIAM Journal of Computing*, 30(1):44–85, 2000.

[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] A. C. Kaporis, C. Makris, S. Sioutas, A. K. Tsakalidis, K. Tsichlas, and C. D. Zaroliagis. Improved bounds for finger search on a ram. In *11th Annual European Symposium on Algorithms*, pages 325–336, 2003.

[15] 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.