Proximate Point Searching

Erik D. Demaine

MIT Laboratory for Computer Science,
200 Technology Square, Cambridge, MA 02139, USA.

John Iacono

Department of Computer and Information Science,
Polytechnic University,
5 MetroTech Center, Brooklyn, NY 11201, USA.

Stefan Langerman [*,1]

Département d'Informatique,
Université Libre de Bruxelles,
ULB CP 212, avenue F.D. Roosevelt 50, 1050 Bruxelles, Belgium.

Abstract

In the 2D point searching problem, the goal is to preprocess n points $P = \{p_1, \ldots, p_n\}$ in the plane so that, for an online sequence of query points $q_1, \ldots, q_m$, it can quickly determined which (if any) of the elements of P are equal to each query point q_i. This problem can be solved in $O(\log n)$ time by mapping the problem to one dimension. We present a data structure that is optimized for answering queries quickly when they are geometrically close to the previous successful query. Specifically, our data structure executes queries in time $O(\log d(q_{i-1}, q_i))$, where d is some distance function between two points, and uses $O(n \log n)$ space. Our structure works with a variety of distance functions. In contrast, it is proved that, for some of the most intuitive distance functions d, it is impossible to obtain an $O(\log d(q_{i-1}, q_i))$ runtime, or any bound that is $o(\log n)$.

Key words: Point Location, Dynamic Finger Property, Distance Functions
2000 MSC: 68P05, 68U05

* Corresponding author.
Email addresses: `edemaine@mit.edu` (Erik D. Demaine), `jiacono@poly.edu` (John Iacono), `stefan.langerman@ulb.ac.be` (Stefan Langerman).

[1] Chargé de recherches du FNRS, Research done while at McGill University, and supported by grants from MITACS, FCAR and CRM.

1 Introduction

Distribution-sensitive data structures have running times that can be expressed as a function of some distributional measure of the sequence of operations performed on the structure. Such structures can exploit sequences of operations that exhibit some desirable behavior. Because real-world access sequences are rarely uniformly random, if our structures are optimized to perform better on the types of distributions likely to be found in a given application, we can obtain running times that are faster than standard (distribution-insensitive) data structures.

Distribution-sensitive dictionaries. Distribution-sensitive structures are well-studied for the *dictionary* problem: maintain a collection of key-value pairs subject to queries for the value associated with a given key. For this problem, two major types of distributions have been studied: proximity and working sets.

The first type of distribution, *proximity* or *locality of reference*, is where items are more likely to be accessed (searched) if they are close, in terms of rank, to the previous access. The level-linked trees of Brown and Tarjan [1], splay trees of Sleator and Tarjan [2–4], and the unified structure of Iacono [5] all achieve $O(\log |r(q_i) - r(q_{i-1})|)$ query time, where q_i is the ith accessed element, and $r(q_i)$ is the rank of element q_i (the number of elements less than or equal to it in the dictionary). This is called the *dynamic finger property*.

The second type of distribution, *working sets*, is where items are more likely to be accessed if they have been accessed recently. Data structures with the *working-set property* can access an element in time logarithmic in the number of distinct accesses since the last time that element was accessed. Splay trees [2], the working-set structure [5], and the unified structure [5] are all dictionaries with the working-set property.

The *unified property* [5] integrates the dynamic finger and working-set properties into one, by saying that an access is fast if it is close (in rank) to an item that was accessed recently. Splay trees are conjectured to have this property, but so far only the (complicated and impractical) unified structure [5] is known to have it.

Other distribution-sensitive structures. While distribution sensitivity in dictionaries is well-studied, there are relatively few results for other types of data structures. One such result, by Iacono [6], is that the pairing heaps of Fredman et al. [7] have a working-set-like property for priority queues.

In computational geometry, there are several similar results pertaining to the problem of planar point location. The three papers [8–10] all present roughly the same result: Given the access probability of each region of a triangulation (assumed to be independent), it is possible to create a data structure whose expected search time is the entropy of that probability distribution. Such a result is analogous to the one-dimensional structures known as *optimal search trees* (Knuth [11]), which date back to 1971. In contrast, splay trees, and in fact any structure with the working-set property, have the same amortized asymptotic runtime as optimal search trees [6], without having to know the access probabilities. Another result, by Goodrich, Orletsky, and Ramaiyer [12], uses splay trees to obtain working-set properties for point location in a triangulation. However, the running times of this structure are relative not to the user-specified subdivision, but to one generated by the algorithm. Thus the state of the art in distribution-sensitive point location is equivalent to the results for one-dimensional point location obtained thirty years ago.

Our results. In this paper, we present results for the *planar point searching problem*. We are given n points $P = \{p_1, \ldots, p_n\}$. We are allowed to preprocess the points, using roughly linear space, in order to support an online sequence of point search queries $q_1, \ldots, q_m$. For each query q_i, the data structure must return the index j of a point p_j such that $p_j = q_i$, or it must indicate that no such point exists. Our data structure works in the real-RAM model of computation, and lower bounds are valid in the algebraic computation tree model. In other models such as the word-RAM model, queries can trivially be performed in constant time using perfect hashing schemes (e.g. [14]).

We can easily solve a point-searching query in $O(\log n)$ time by reducing the problem to one dimension and using a standard balanced search tree. Our goal is to obtain a dynamic-finger type distribution-sensitive data structure, where a query is fast if it is geometrically close to the previous successful query.

Recall that, in one dimension, we can obtain a query time that is logarithmic in the difference in rank between the query point and the previous point. Unfortunately, the notion of difference in rank does not have a clear generalization to higher dimensions. In one dimension, the difference in rank between points x and y measures the number of points in the one-dimensional region between x and y. In two dimensions, we would like to have a similar point-counting distance function, where the distance between x and y is the number of points in some two-dimensional region containing both x and y. Such a *region-counting distance function* provides a bridge between the geometric distance between two points and the combinatorial complexity of the point set. In Section 2, we discuss properties of region-counting distance functions. In particular, we introduce the *fixed-triangle distance function*, and show that many desirable region-counting distance functions can be reduced to this one.

In Section 3, we present our data structure that executes queries in $O(\log d(q_{i-1}, q_i))$ time where d is a fixed-triangle distance function. Thus we obtain a dynamic-finger type result in 2D. Our data structure requires $O(n \log n)$ space and can be constructed in $O(n^2 \log n)$ time. The data structure is based on the idea of jump pointers and can be seen as a multidimensional generalization of deterministic skip lists [15,16]. In our structure, each data point stores $O(\log n)$ pointers to other points spaced at geometrically increasing ranks away. A search can be performed by greedily choosing the best pointer to follow. The details of selecting the specific destinations of the jump pointers is presented in Section 4.

2 Distance Functions

In this section we describe several definitions of a 2-dimensional distance function, with the goal of creating one that is a reasonable generalization of the one-dimensional rank-difference function. In an attempt to do so, we restrict ourselves to functions which involve counting the number of points inside some two-dimensional shape that contains x and y. A static point set $P = \{p_1, \ldots, p_n\}$ will be an implicit parameter in all of our distance functions.

We focus on one natural category of such functions, which we call *region-counting* distance functions. A region-counting distance function is defined by a triple $r = (a, b, S)$ where a and b are points and S is a region of the plane such that inclusion in S can be computed in $O(1)$ time. The distance function using the region-counting triple r, $d_r(x, y)$, is defined as follows: if $r(x, y)$ is the shape obtained by translating, rotating, and uniformly scaling S so that a maps to x and b maps to y, then $d_r(x, y)$ is the number of points in $P \cap r(x, y)$.

We place two restrictions on region-counting distance functions. The first requirement, which we call *monotonicity*, is that if x, y, and z appear in that order on a line, then $d(x, y) \leq d(x, z)$. This requirement stems from the intuitive idea that, as one point moves away from another, their distance should not decrease. The second requirement, which we call *sanity*, is that neighborhoods have polynomial size: $|\{y \in P \mid d(x, y) < k\}| \leq k^{O(1)}$. This requirement arises directly from our goal of a $O(\log d(x, y))$ query time, because any algebraic decision tree query algorithm cannot search among more than $k^{O(1)}$ different results in $O(\log k)$ time.

One requirement we do not impose is symmetry, that $d(x, y) = d(y, x)$ for all x, y. While symmetric distance functions are intuitively pleasing, we show below that for every symmetric distance function, there is a better asymmetric distance function (better meaning all possible distances are not larger). However, the converse is not true. Thus, surprisingly, asymmetric distance functions appear to be the natural choice. The distance functions also do not need to satisfy the triangle inequality, and so are not metrics.

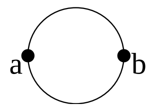

Fig. 1. The distance function defined by this seemingly reasonable shape is not sane.

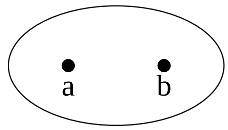

Fig. 2. The distance function defined by this shape is a target distance.

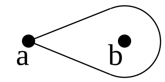

Fig. 3. By the monotonicity requirement, any target shape has an ice cream cone shaped subset.

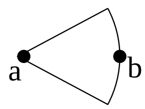

Fig. 4. The sector distance function is sane and monotone, but is not a target distance because there is no circle about b that is in the region. Any sane and monotone distance function has a sector as a subset of its region.

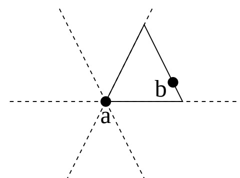

Fig. 5. This figure illustrates how the fixed-triangle distance $d_{T_6}(a, b)$ is calculated. The dotted lines are the 6-star from a and the solid lines indicate the triangle $T_6(a, b)$.

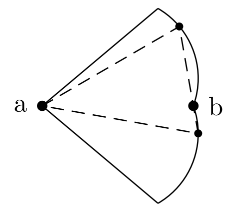

Fig. 6. The unfixed-triangle distance $d_{UT_k}(a, b)$ is defined by this shape $UT_k(a, b)$ which contains $T_k(a, b)$ for any orientation of the plane.

With the two requirements of monotonicity and sanity in hand, we can narrow down the allowable region-counting triples $r = (a, b, S)$. In order to meet the monotonicity requirement, the shape S must be star-shaped with a in the kernel. For simplicity, we further require that S be a finite union of closed bounded convex shapes. In particular, this rules out sets S with a fractal or unbounded boundary. Distance functions with this property will be called *simple*. We say the distance functionis *axis-symmetric* if the line through a and b is a symmetry axis for S.

One particular distance function, the α*-sector* distance, is of particular interest. Its region S is defined as a closed sector (pie wedge) of angle α, apex at a, and with b at the center of the arc (see Figure 4). We show that this is the minimal simple axis-symmetric region-counting distance.

Lemma 1 *An axis-symmetric simple monotone region-counting distance function defined by a triple (a, b, S) is sane if and only if there exists an $\alpha > 0$ such that S contains an α-sector.*

Proof: Let C be the circle centered at a and with radius point b. Suppose S does not contain any α-sector. Then because S is star-shaped, there must exist an arc of positive length γ (measured in radians) and containing b, whose intersection with S is either empty or contains only the point b.

We construct a set of n points $P = \{p_0, \ldots, p_{n-1}\}$ where p_i has polar coordinates $(i\gamma/2n, 1)$, plus one point o at the origin. Note that since all n points in P are contained in an arc of length $\gamma/2$, the regions S moved to o and p_i does not contain any points except maybe for o and p_i, and so all points in P are at distance at most 2 from o and this distance function is not sane.

Conversely, if S does contain an α-sector, then suppose we have k points at a distance $\leq d$ from some point o. Draw an α-sector for each of them. Each sector contains at most d points. Perform the following steps repeatedly:

(1) Find the sector with the largest radius.
(2) Remove all points it contains and the corresponding sectors.

We prove that these steps are repeated at most $8\pi/\alpha$ times. The boundary of a sector is composed of a left side, a right side, and an arc. The largest sector either

A. Does not intersect any larger deleted sector.
B. Intersects larger deleted sectors only on one side.
C. Intersects larger deleted sectors on both sides.

In Case A, we remove all points within an angle α around o. This can only be done $2\pi/\alpha$ times. In Case B, the point p that defines the sector being deleted has a larger sector s that was removed earlier, say to its right. Since p didn't get deleted when we removed s, p is not in s and so the whole left half of the sector of p doesn't overlap with any previously deleted sectors. So we remove all points within an angle $\alpha/2$ around o, which can only be done $4\pi/\alpha$ times. In Case C, the sector we remove is surrounded by two large sectors (of angle α) on both sides, thus in step 2, we remove all points in a small gap between cones of angle α. This can only be done $2\pi/\alpha$ times. Thus, $k \leq 8d\pi/\alpha$. $\square$

Our data structure applies to a class of region-counting distance functions which we call *target distances*. The triple $r = (a, b, S)$ defining a target distance must satisfy three properties: a and b are in S, S is star-shaped with a in the kernel, and S contains a disk of some radius $z > 0$ centered at b. The last requirement is slightly stronger than what is necessary for sanity.

Our data structure works directly with a special distance function, which we call a *fixed-triangle distance*, that is not a region-counting distance function in the strict sense defined above, but has the property that for any target distance function d there is a fixed-triangle distance function d' such that $d(x,y) \geq d'(x,y)$ for all x and y. Thus, because our structure supports any fixed-triangle distance, it can be modified to support any target distance function.

Given any constant $k \geq 3$, the fixed-triangle distance function $d_{T_k}(x,y)$ is computed as follows. Define the *k-star* of x to be the shape formed by k equally angularly spaced rays radiating from x, with one ray pointing horizontally to the right. Let $T_k(x,y)$ be the smallest isosceles triangle containing y, having one vertex at x, and whose two identical sides are line segments from the k-star of x. (In the ambiguous case that y lies on the k-star, we arbitrarily choose $T_k(x,y)$ to be the clockwise-most such triangle.) We now define the fixed-triangle distance $d_{T_k}(x,y)$ to be the number of points in S that are in the triangle $T_k(x,y)$.

We also define the *unfixed-triangle* region-counting distance. Given points a and b, let Δ_1 and Δ_2 be the two isosceles triangles with vertices a and b and angle $2\pi/k$ at a, where Δ_1 is to the left of the ray ab and Δ_2 to its right. Let D_1 and D_2 be the disks circumscribed to Δ_1 and Δ_2 and let C be the cone of angle $4\pi/k$ with apex at a and symmetry axis ab and containing b. The unfixed-triangle distance is defined as $UT_k = (a, b, (D_1 \cup D_2) \cap C)$.

Lemma 2 *For any $k \geq 4$, and any two points x and y, $d_{T_k}(x,y) \leq d_{UT_k}(x,y)$.*

Proof: Consider the fixed triangle $T_k(x,y)$ with vertices x, p_1 and p_2 and assume p_1 is to the left of the ray xy and p_2 to its right. Because the angle at p_1 is $\pi/2 - \pi/k$, p_1 is on the boundary of D_1 and so the triangle xyp_1 is contained in D_1. Likewise, the triangle xyp_2 is contained in D_2. Furthermore, $T_k(x,y)$ is contained in C which completes the proof. $\square$

Interestingly, any target region with target radius at least $d_2(a,b)\tan(2\pi/k)$ (where d_2 is the Euclidean distance) contains the unfixed-triangle region UT_k. So we have:

Corollary 1 *Given any target distance function d_r, $r = (a, b, S)$, there is a fixed-triangle distance function d_{T_k} such that $d_r(x,y) \geq d_{T_k}(x,y)$ for any x and y.*

3 The Structure

The data structure is parameterized by the data points $p_1, \ldots, p_n$ and the parameter k in the fixed-triangle distance function that is to be used. We define the *discrete direction* $\Theta(x,y)$ of a ray from x to y to be $\lfloor \angle(x,y)k/2\pi \rfloor$, where $\angle(x,y)$ is the *absolute angle* of the ray from x to y, that is, the angle of the ray from the horizontal $(1, 0)$ direction (in radians).

We will actually define k separate data structures $S_0, \ldots, S_{k-1}$, and the structure to be used for a particular query will be $S_{\Theta(q_{i-1},q_i)}$. Thus, each structure S_i handles travel in a direction with absolute angle between $2\pi i/k$ and $2\pi(i+1)/k$. Structure S_i views the point set as scaled non-uniformly so that the actual angle between absolute angles $2\pi i/k$ and $2\pi(i+1)/k$ becomes $\pi/3$ (60 degrees) (i.e., the scaling, by a factor $2\sin(\pi/k)$, takes place along the line that bisects the angle between discrete angle i and discrete angle $i+1$). Thus, if $\Theta(x, y) = i$, then $d_{T_6}(x, y)$ in the S_i structure equals $d_{T_k}(x, y)$ in the original point set. With this observation, we can produce one structure using the d_{T_6} distance function for the scaled point set for each of the S_i, $i = 0, \ldots, k-1$. This will simulate the d_{T_k} distance function of the original point set. The d_{T_6} distance function is appealing because the isosceles triangles become equilateral.

The data structure for the d_{T_6} distance function consists of the n points, each of which is augmented with a list of $O(\log n)$ pointers to other points. These pointers, which need not be distinct, are organized according to two parameters: depth and direction. We denote by $p_i(r, \theta)$ the set of $O(1)$ pointers at depths r and direction θ associated with point p_i. The direction θ is in the range $0 \ldots 5$ and the depth r is in the range $0 \ldots \lceil \log_{3/2} n \rceil$. Both r and θ are discrete values.

The data structure also stores the description of a triangle, $\tau(p_i, r, \theta)$, with each set of pointers $p_i(r, \theta)$. In general, the triangle at direction θ and distance r from x, denoted $\tau(x, r, \theta)$, is the largest isosceles triangle with a vertex at x, whose equal sides lie along rays emanating from x at absolute angles $2\pi\theta/k$ and $2\pi(\theta + 1)/k$, and that contains at most $(3/2)^r$ points in S.

A crucial property of the data structure is that, for any point $x \in \tau(p_i, r, \theta)$, there is a data point p_j pointed to by $p_i(r, \theta)$ and a θ' such that $x \in \tau(p_j, r - 1, \theta')$. Intuitively, the point p_j serves as a stepping stone on the way from p_i to x by which the depth r decreases. We also require that $\tau(p_i, 0, \theta) = \{p_i\}$. The pointers in $p_i(r, \theta)$ point to 7 carefully selected points described in the next section.

Given a pointer to the previous successful query point in the structure, p_i, and the coordinates of the current query point, x, the search proceeds as follows:

(1) Initialize θ, the direction of search, to $\Theta(p_i, x)$.
(2) Initialize r, the depth of search, to 0.
(3) While x is not in $\tau(p_i, r, \theta)$, increment r.
(4) Initialize j, the index of the currently visited point, to i.
(5) While $r > 0$:

(a) Search for a point $p_{j'}$ in $p_j(r, \theta)$ and a discrete angle θ' (between 0 and 5) such that x is in $\tau(p_{j'}, r-1, \theta')$, by trying all such points $p_{j'}$ and discrete angles θ'.
(b) Set j to j'.
(c) Set θ to θ'.
(d) Decrement r.

(6) If $x = p_j$, return j; otherwise, the query is unsuccessful.

The time bound and correctness of the algorithm can be argued as follows. The while loop of Step 3 terminates with r set to precisely $\lceil \log_{3/2} d_{T_k}(p_i, x) \rceil$. Thus, because the search of Step 5(a) examines $O(1)$ possibilities and hence takes $O(1)$ time, the total running time is $O(\log d_{T_k}(p_i, x))$. Now all we must show is that the return value is correct. The invariant throughout the while loop of Step 5 is that $d(p_j, x) \leq (3/2)^r$. The crucial property for the data structure described above guarantees that there is a jump pointer that reduces the distance (measured by the fixed-triangle distance function) to the query point by a factor $(2/3)$. This results in p_j converging to the point in P with the same coordinates as x, if such a point exists.

4 Point Selection

In this section we describe how the 7 points in $p_i(r, \theta)$ are chosen. Recall that $\tau(p_i, r, \theta)$ is an equilateral triangle emanating from p_i at direction θ with at most $(3/2)^r$ points. We define $\rho(\tau) = \tau \cap P$ to be the set of points in P inside a triangle τ. The goal of this section is to find at most 7 points from $\rho(\tau(p_i, r, \theta))$ to be in $p_i(r, \theta)$ such that $\cup_{p \in p_i(r,\theta), \phi \in \{0 \ldots 5\}} \tau(p, r-1, \phi) \supseteq \rho(\tau(p_i, r, \theta))$.

We define the three triangles $\tau_1(p_i, r, \theta) \ldots \tau_3(p_i, r, \theta)$ as follows: The triangle $\tau_1(p_i, r, \theta)$ is $\tau(p_i, r-1, \theta)$. The triangles $\tau_2(p_i, r, \theta)$ and $\tau_3(p_i, r, \theta)$ are the two equilateral triangles containing $(3/2)^{r-1}$ points (or less if there are not enough points), contained in $\tau(p_i, r, \theta)$, and rooted at the two vertices of $\tau(p_i, r, \theta)$ other than p_i. Each of these three triangles contains two thirds of the points in $\tau(p_i, r, \theta)$ and so they must also contain the centerpoint of the points in $\tau(p_i, r, \theta)$. (The *centerpoint* of a set is a point such that any halfplane containing it contains at least one third of the set. Such a point always exists; see e.g. [17, pp. 63–66].) Thus the three triangles have a common intersection and they cover $\tau(p_i, r, \theta)$.

The triangles $\tau_1(p_i, r, \theta) \ldots \tau_3(p_i, r, \theta)$ have the following two properties:

Property 1 *The union of $\tau_1(p_i, r, \theta) \ldots \tau_3(p_i, r, \theta)$ is $\tau(p_i, r, \theta)$.*

Property 2 *Each of $\tau_1(p_i, r, \theta) \ldots \tau_3(p_i, r, \theta)$ contains at most $(3/2)^{r-1}$ points.*

Given a triangle τ we define $e(\tau)$ to be a set of at most three points containing one of the closest points to each edge of τ.

We can now define the points in $p_i(r, \theta)$ to be the point p_i and the up to six points in $e(\tau_2(p_i, r, \theta))$ and $e(\tau_3(p_i, r, \theta))$.

Lemma 3 $\cup_{\phi \in \{0 \ldots 5\}} \rho(\tau(p_i, r-1, \phi)) \supseteq \rho(\tau_1(p_i, r, \theta))$

By setting $\phi = \theta$ and noting that $\tau(p_i, r-1, \theta) = \tau_1(p_i, r, \theta)$ completes the proof.

Lemma 4 *For any equilateral triangle τ' in which one edge is horizontal, containing at most $(3/2)^{r-1}$ points, $\cup_{p \in e(\tau'), \phi \in \{0 \ldots 5\}} \tau(p, r-1, \phi) \supseteq \rho(\tau')$.*

Proof: We first define a new triangle τ by shrinking the triangle τ' by moving its edges inward while preserving their angles until all three edges have at least one point of $e(\tau)$ on them. We may now work with τ instead of τ' since by construction there are no points in the region $\tau' - \tau$.

If there are only one or two points in $e(\tau)$, then the lemma trivially holds since one of those points must be a vertex of τ'. Thus we consider the case where $|e(\tau)| = 3$. We label these three points a, b and c in counter-clockwise order and visualize them as being on the bottom, right, and left side of equilateral triangle. The proof breaks into several cases. Figure 7 illustrates the notation we use to enumerate these cases. Each of the three points is categorized as l, m or r, depending on which third of the side of the triangle it lies in. Note that if a point lies on the boundary between two categories, it can be classified either way and the proof will remain valid.

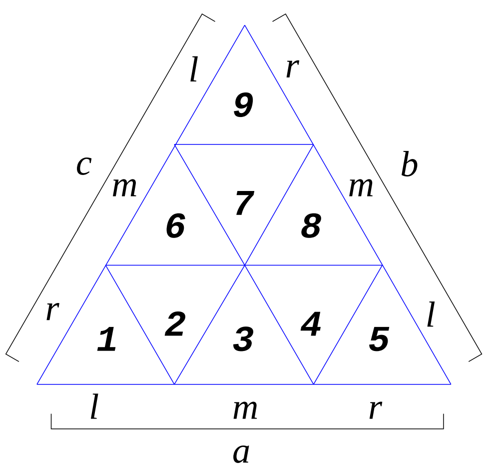

Fig. 7. Notation.

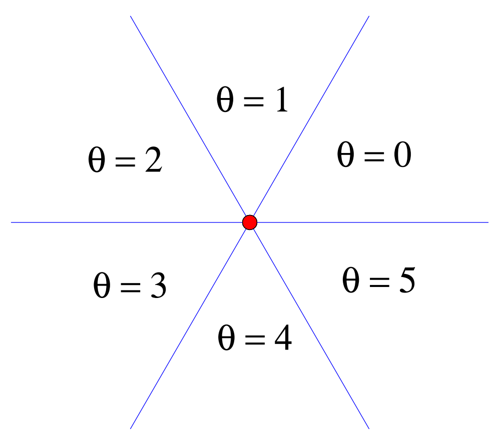

Fig. 8. Discrete angles in d_{T_6}.

We provide a proof for four representative configurations of the points, all other configurations can be solved though symmetry or relabeling a, b and c. The exact mapping of all of the possible configurations to cases in the proof

a	b	c	case	a	b	c	case	a	b	c	case
l	l	l	1	m	l	l	4	r	l	l	2
l	l	m	4	m	l	m	4	r	l	m	2
l	l	r	2	m	l	r	4	r	l	r	2
l	m	l	4	m	m	l	4	r	m	l	4
l	m	m	4	m	m	m	3	r	m	m	4
l	m	r	2	m	m	r	4	r	m	r	4
l	r	l	2	m	r	l	2	r	r	l	2
l	r	m	4	m	r	m	4	r	r	m	4
l	r	r	2	m	r	r	4	r	r	r	1

Table 1
All possible configurations and their mapping to cases in the proof of Lemma 4.

is given in Table 1.

Case 1. In this case all three points are in the l regions, or all three points are in the r regions. We consider the case where a, b and c are in l regions. Figure 9 shows that a is in region l. The three shaded areas emanating from a represent the smallest size of the triangles in $\tau(a, r-1, \theta)$ for three values of θ. We know that these three triangles extend at least all the way the the edge of τ since τ only has $(3/2)^{r-1}$ points. No assumptions can be made about extending the small triangles beyond the border of τ since we can not make any assumptions about the distribution of points outside of τ. The proof holds if and only if τ may be covered with the small triangles that emanate from a, b, and c and stop when they encounter the edge of τ. From the diagram, it can be concluded that no matter what the position of a is in the l region, $\cup_{\phi \in \{0 \ldots 5\}} \tau(a, r-1, \phi)$ covers regions 3, 4, 5, and 8. We will state this as a covers 3, 4, 5, and 8. The same reasoning, when applied to b and c in the l region yields the fact that b covers 6, 7, 8, and 9 and c covers 1, 2, 3, and 6. Thus a, b and c cover all numbered regions.

Case 2. In this case, two of the three points a, b and c are in the l and r regions incident to one of the three vertices of the large triangle. For example, b could be in region r and c could be in region l. We also assume that c is closer to their common vertex than b; other cases are handled symmetrically. In Figure 11, we have shaded areas that must be covered by b. In Figure 12, we have shaded the area that must be covered by c. The union of these shaded areas covers τ.

Case 3. In this case, all three points are in the m region. In Figure 10, the point a is in the left half of the m region. In this case a covers 1, 2, 3, 4, and 5. The same will happen due to symmetry if a in the right half of the m region.

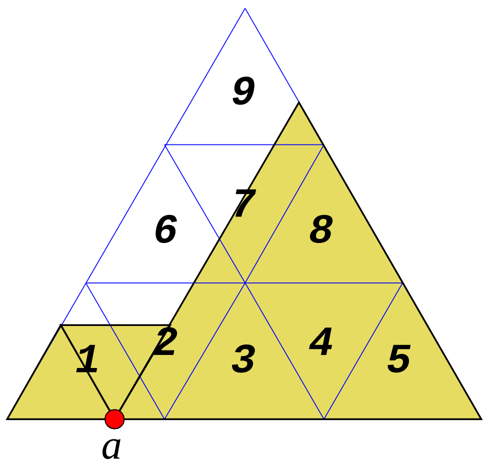

Fig. 9. Point a is in the l region.

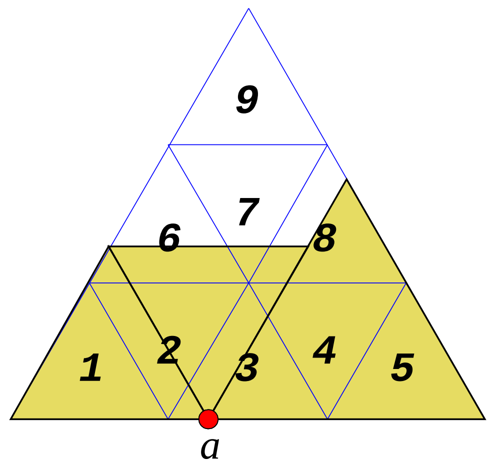

Fig. 10. Point a is in the m region.

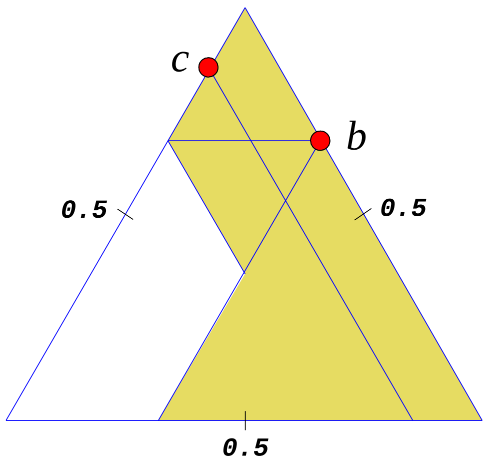

Fig. 11. Case 2, region covered by b.

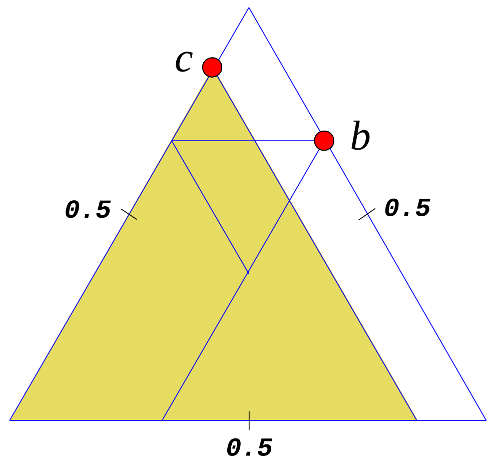

Fig. 12. Case 2, region covered by c.

Using the same logic when b and c are also in the m region, b covers 4, 5, 7, 8, and 9 and c covers 1, 2, 6, 7, and 9. Thus all regions are covered and their union is τ.

Case 4. In this case, one of the points is in the m region and one of the other points is in an l or r region closest to the first point. For example, a could be in the m region and b in the l region. Then according to Figure 10, a covers 1, 2, 3, 4, and 5, while according to Figure 9, suitably rotated, b covers 6, 7, 8, and 9. Thus all regions are covered and their union is τ.

This concludes the proof of the lemma. □

We now may prove the main result of this section, that the 7 selected points have the properties needed to continue the search:

Theorem 2 *When* $p_i(r,\theta) = \{p_i\} \cup e(\tau_2(p_i,r,\theta)) \cup e(\tau_3(p_i,r,\theta))$, $\cup_{p \in p_i(r,\theta), \phi \in \{0 \ldots 5\}} \rho(\tau(p, r-1, \phi)) \supseteq \rho(\tau(p_i, r, \theta))$ *and* $|p_i(r,\theta)| \leq 7$.

Proof: The size constraint is easily met since the function e yields a set of at most 3 points. From Property 2:

$$\rho(\tau_1(p_i, r, \theta)) \cup \rho(\tau_2(p_i, r, \theta)) \cup \rho(\tau_3(p_i, r, \theta)) = \rho(\tau(p_i, r, \theta))$$

From Lemma 3:

$$\cup_{p \in \{p_i\}, \phi \in \{0 \ldots 5\}} \rho(\tau(p, r-1, \phi)) \supseteq \rho(\tau_1(p_i, r, \theta))$$

From Lemma 4:

$$\cup_{p \in e(\tau_2(p_i, r, \theta)), \theta \in \{0 \ldots 5\}} \tau(p, r-1, \theta) \supseteq \rho(\tau_2(p_i, r, \theta))$$

$$\cup_{p \in e(\tau_3(p_i, r, \theta)), \theta \in \{0 \ldots 5\}} \tau(p, r-1, \theta) \supseteq \rho(\tau_3(p_i, r, \theta))$$

Combining all of the above assertions proves the theorem. □

5 Conclusion

This paper presents a first step towards the development of geometric data structures with properties similar to the dynamic finger property of dictionary data structures. One major obstacle encountered is that most intuitive distance functions are not sane, that is, would not allow fast data structures in the algebraic decision tree model of computation. For example, the edge crossing distance in the Delaunay triangulation of the point set would seem desirable, but sets of points can be constructed where some point is at a distance k from 2^k other points, and hence any query algorithm will require $\Omega(k) = \Omega(\log n)$ time on average.

Many questions remain open. Besides the distance functions supported by our structure, one that would seem very natural is the *sector distance* (see Figure 4), and it would be interesting to see whether our data structure can be adapted to such a distance function. Generalizations to higher dimensions should also be investigated. We hope that our techniques can be extended to the more general problems of finding the closest neighbor of a query point, or performing point location in a set of regions. A first solution to the proximate point location problem using distance functions similar to the ones described in this article has been proposed recently [18]. Our data structure naturally bears some ressemblance with other multidimensional generalizations of skip lists such as the randomized neighborhood graphs [19]. It would be interesting to explore these connections to see if the qualities of the two structures could be combined.

Acknowledgements

The authors thank the anonymous referees for many useful comments and suggestions.

References

[1] M. R. Brown, R. E. Tarjan, Design and analysis of a data structure for representing sorted lists, SIAM J. Comput. 9 (1980) 594–614.

[2] D. D. Sleator, R. E. Tarjan, Self-adjusting binary trees, JACM 32 (1985) 652–686.

[3] R. Cole, B. Mishra, J. Schmidt, A. Siegel, On the dynamic finger conjecture for splay trees. Part I: Splay sorting $\log n$-block sequences, SIAM Journal on Computing 30 (1) (2000) 1–43.

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

[5] J. Iacono, Alternatives to splay trees with $O(\log n)$ worst-case access times, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, pp. 516–522.

[6] J. Iacono, New upper bounds for pairing heaps, in: Proc. 7th Scandinavian Workshop on Algorithm Theory (LNCS 1851), 2000, pp. 32–45.

[7] M. L. Fredman, R. Sedgewick, D. D. Sleator, R. E. Tarjan, The pairing heap: A new form of self-adjusting heap, Algorithmica 1 (1986) 111–129.

[8] S. Arya, T. Malamatos, D. M. Mount, Entropy-preserving cuttings and space-efficient planar point location, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, pp. 256–261.

[9] S. Arya, T. Malamatos, D. M. Mount, A simple entropy-based algorithm for planar point location, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, pp. 262–268.

[10] J. Iacono, Optimal planar point location, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, pp. 240–241.

[11] D. E. Knuth, Optimum binary search trees, Acta Inf. 1 (1971) 14–25.

[12] M. T. Goodrich, M. Orletsky, K. Ramaiyer, Methods for achieving fast query times in point location data structures, in: Proc. 8th ACM-SIAM Sympos. Discrete Algorithms, 1997, pp. 757–766.

[13] S. Arya, S. Cheng, D. Mount, H. Ramesh, Efficient expected-case algorithms for planar point location, in: Proc. 7th Scandinavian Workshop on Algorithm Theory (LNCS 1851), 2000, pp. 353–366.

[14] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.

[15] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, in: F. Dehne, J.-R. Sack, N. Santoro (Eds.), Proceedings of the Workshop on Algorithms and Data Structures, Vol. 382 of Lecture Notes in Computer Science, Ottawa, Ontario, Canada, 1989, pp. 437–449.

[16] J. I. Munro, T. Papadakis, R. Sedgewick, Deterministic skip lists, in: Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms, Orlando, Florida, 1992, pp. 367–375.

[17] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Vol. 10 of EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Heidelberg, West Germany, 1987.

[18] J. Iacono, S. Langerman, Proximate point location, in: Proceedings of the 2003 ACM Symposium on Computational Geometry (SoCG 2003), 2003, pp. 220–226.

[19] S. Arya, D. M. Mount, Approximate nearest neighbor queries in fixed dimensions, in: Proc. 4th ACM-SIAM Sympos. Discrete Algorithms, 1993, pp. 271–280.