Proximate Point Searching

Erik D. Demaine* John Iacono† Stefan Langerman‡

Abstract

In the 2D point searching problem, our goal is to preprocess n points $P = \{p_1, \ldots, p_n\}$ in the plane so that, for an online sequence of query points $q_1, \ldots q_m$, we can quickly determine 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 down 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 $O(\log d(q_i, q_{i-1}))$, where d is some distance metric between two points. Our structure works with a variety of distance metrics. In contrast, we prove that, for some of the most intuitive distance metrics d, it is impossible to obtain an $O(\log d(q_i, q_{i-1}))$ runtime, or any bound that is $o(\log n)$.

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. Thus 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 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 [4], splay trees of Tarjan and Sleator [13, 6, 5], and the unified structure of Iacono [10] 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 [13], the working-set structure [10], and the unified structure [10] are all dictionaries with the working set property.

The *unified property* [10] 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 [10] 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 [9], is that the pairing heaps of Fredman et al. [7] have a working-set-like property for priority queues.

In computational geometry, the closest results are about planar point location. The three papers [2, 3, 11] 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 [12]), 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 [9], without having to know the access probabilities. Another result, by Goodrich, Orletsky, and Ramaiyer [8], 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

*MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA. edemaine@mit.edu

†Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn NY 11201, USA. jiacono@poly.edu

‡School of Computer Science, McGill University, 3480 University Street Suite 318, Montreal, Quebec H3A 2A7, Canada. sl@cgm.cs.mcgill.ca. Research is supported by grants from MITACS, FCAR and CRM.

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.

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 metric, 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 metric* 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 metrics. In particular, we introduce one metric, the *fixed-triangle metric*, and show how many desirable region-counting metrics can be reduced to this metric.

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 discrete triangle metric. Thus we have obtained a dynamic-finger type result in 2D. Our data structure requires $O(n \log n)$ space and can be constructed in polynomial time. The data structure is based on the idea of jump pointers, where 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.

2 Distance Metrics

In this section we describe several methods for computing a 2-dimensional distance metric, with the goal of creating one that is a reasonable generalization of the one-dimensional rank-difference metric. In an attempt to do so, we restrict ourselves to metrics 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 metrics.

We focus on one natural category of such metrics, which we call *region-counting metrics.* A region-counting metric r is a triple (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 using the region-counting metric r, $d_r(x, y)$, is defined as follows: if S' 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 $P \cap S'$.

We place two restrictions on region-counting metrics. 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: $|\{y \mid d(x, y) < k\}| \leq k^{O(1)}$. 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 metrics are intuitively pleasing, it can be shown that for every symmetric metric, there is a better asymmetric metric (better meaning all possible distances are not larger). However, the converse is not true. Thus, surprisingly, asymmetric distance metrics appear to be the natural choice.

With the two requirements of monotonicity and sanity in hand, we can narrow down the allowable region-counting metrics $r = (a, b, S)$. In order to meet the monotonicity requirement, the shape S must be star-shaped with a in the kernel. In addition, because of the sanity requirement, we claim that the shape S must contain points other than b that are not strictly inside the circle centered at a with radius point b. This claim follows by contradiction, by placing an arbitrarily large number of points on a circle centered at x, with no points other than x inside the circle. Thus $d(x, y)$ is at most 2 for the many points y on the circle, contradicting sanity. A similar argument shows that S must include b.

Our data structure applies to a class of region-counting metrics which we call *target metrics.* A target metric $r = (a, b, S)$ 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 metric, which we call a *fixed-triangle metric*, that is not a region-counting metric in the strict sense defined above, but has the property that for any target metric d there is a fixed-triangle metric d' such that $d(x,y) \geq d'(x,y)$ for all x and y. Thus, because our structure supports any fixed-triangle metric, it can be modified to support any target metric.

Given any constant $k \geq 3$, the fixed-triangle metric $d_{t(k)}(x,y)$ 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 straight up. 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. We now define the fixed-triangle metric $d_{t(k)}(x,y)$ to be the number of points in S that are in the triangle $T_k(x,y)$.

Theorem 1 *Given any target metric d_r, $r = (a,b,S)$, there is a fixed-triangle metric $d_{t(k)}$ such that $d(x,y) \geq d_{t(k)}(x,y)$ for any x and y.*

Proof: For any points x, y, consider the circle centered at x and with radius point y. The target metric guarantees the existence of a disk centered at y that is contained in S. The circle and the disk intersect at an arc. Let θ be the angular length of this arc, measured in radians. This angle is invariant under uniform scaling as well as translation and rotation, and hence constant over all x and y. Let k be $\lceil \pi/\theta \rceil$, and consider the fixed-triangle metric $d_{t(k)}$. Because S must be star-shaped, its region must include not only the disk around y, but also the ice-cream-cone shaped region formed by touching the two tangents from x to the disk (see Figure 5). In particular, this ice-cream-cone shape contains the isosceles triangle $T_k(x,y)$. □

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 metric 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 upward-vertical 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 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). (The scaling 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 for the $d_{t(6)}$ metric, and thus k ($= O(1)$) versions of this structure can be used to simulate the $d_{t(k)}$ metric. The $d_{t(6)}$ metric is appealing because the isosceles triangles become equilateral.

The data structure for the $d_{t(6)}$ metric 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. The direction θ is in the range $0 \ldots 5$ and the depth r is in the range $0 \ldots \lceil \log n \rceil$.

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 smallest 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 least 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 22 carefully selected points. A description of how these pointers are selected to meet the above criterion is omitted because of space constraints.

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_k 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_k and discrete angles θ'.
   (b) Set j to k.
   (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 $\lceil \log d_{t(k)}(p_i, x) \rceil$. 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 it that the return value is correct. The invariant throughout the while loop of Step 5 is that $d(p_j, x) \leq 2^r$. The crucial property for the data structure described above guarantees that there is a jump pointer that will halve the distance (measured by the fixed-triangle metric) to the query point. This results in p_j converging to the

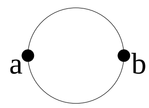

Figure 1: This seemingly reasonable metric is not sane.

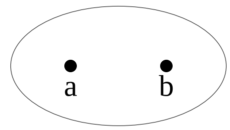

Figure 2: This metric is a target metric.

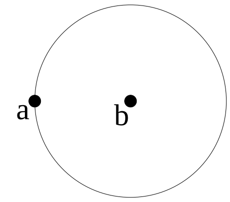

Figure 3: This metric is a target metric.

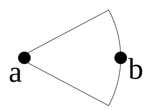

Figure 4: This pie-wedge metric is sane and monotone, but is not a target metric because there is no circle about b that is in the metric. It can be shown that any sane and monotone metric has a pie-wedge metric as a subset.

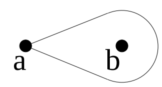

Figure 5: This ice cream cone shape is a target metric, and it can be shown that any target metric has an ice cream cone shaped subset.

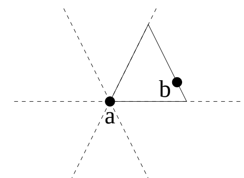

Figure 6: This figure illustrates how the triangle metric $d_{t(6)}(x, y)$ is calculated. The dotted lines are the 6-star from x and the solid lines indicate the triangle $T_6(a, b)$.

item in P with the same coordinates as x.

4 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 measures are not sane, that is, would not allow fast data structures in the algebraic decision tree model of computation. For example, the edge 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 measures supported by our structure, one that would seem very natural is the *pie-wedge metric* (see Figure 4), and it would be interesting to see whether our data structure could be adapted to such a distance metric. 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.

References

[1] S. Arya, S. Cheng, D. Mount, and H. Ramesh. Efficient expected-case algorithms for planar point location. In *Scandanavian Workshop on Algorithm Theory. (LNCS 1851)*, pp. 353–366, 2000.

[2] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cuttings and space-efficient planar point location. In *Symposium on Discrete Algorithms*, pp. 256–261, 2001.

[3] S. Arya, T. Malamatos, and D. M. Mount. A simple emtropy-based algorithm for planar point location. In *Symposium on Discrete Algorithms*, pp. 262–268, 2001.

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

[5] R. Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. Technical Report Computer Science TR1995-701, New York Univerity, 1995.

[6] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting log n-block sequences. Technical Report TR1995-700, New York Univerity, 1995.

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

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

[9] J. Iacono. New upper bounds for pairing heaps. In *Scandanavian Workshop on Algorithm Theory (LNCS 1851)*, pp. 32–45, 2000.

[10] J. Iacono. Alternatives to splay trees with $O(\log n)$ worst-case access times. In *Symposium on Discrete Algorithms*, pp. 516–522, 2001.

[11] J.Iacono. Optimal planar point location. In *Symposium on Discrete Algorithms*, pp. 240–241, 2001.

[12] D. E. Knuth. Optimum binary search trees. *Acta Inf.*, 1:14–25, 1971.

[13] D. D. Sleator and R. E. Tarjan. Self-adjusting binary trees. *JACM*, 32:652–686, 1985.