Probabilistic analyses of closest pair algorithms

Mordecai Golin

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1471

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

PROBABILISTIC ANALYSES OF CLOSEST PAIR ALGORITHMS

Mordecai J. GOLIN

Juin 1991

RR-1471

Probabilistic Analyses of Closest Pair Algorithms (Extented Abstract)

Mordecai J. Golin*
INRIA-Rocquencourt

November 1990

Abstract: In this paper we present three simple fast algorithms for finding the closest pair among a set of n points. We perform probabilistic analyses on the three algorithms under the assumption that the points are chosen I.I.D. from the uniform distribution over the unit square $[0,1]^2$.

Analyses probabilistes d'algorithmes de paires les plus proches

Résumé : Dans cet article nous présentons trois algorithmes, simples et rapides pour trouver la paire la plus proche parmi n points. Nous faisons l'hypothèse que les n points sont I.I.D. dans le carré $[0,1]^2$. Notre analyse est probabiliste.

*This work was supported in part by NSF Grant DCR-8605962 and Office of Naval Research Grant N00014-87-K-0460. It was later supported by a Chateaubriand fellowship from the French Ministère des Affaires Étrangères and by the European Community, Esprit II Basic Research Action Program Number 3075 (ALCOM). Author's current address: Algorithms' Project, Inria Rocquencourt, 78153, Le Chesnay Cedex, France.

Probabilistic Analyses of Closest Pair Algorithms
(Extended Abstract)

Mordecai Golin[1]
Princeton University

November 1990

Abstract:

*In this paper we present probabilistic analyses of some simple, practical algorithms for finding the closest pair among a set of n points. The first algorithm is a simplification of the sweep-line algorithm given in [***HNS***]. We show that, given n points chosen I.I.D.* $\mathbf{U[0,1]^2}$ *, and then sorted by x-coordinate, the algorithm's i-th step requires an expected* $\Theta\left(\sqrt{\frac{n}{i}}\right)$ *amount of time, using* $\Theta(n)$ *time overall. This analysis can be modified to show that [***BP***]'s projection method also has the same* $\Theta\left(\sqrt{\frac{n}{i}}\right)$, $\Theta(n)$ *time bound. The final algorithm we examine is a modified version of the gridding algorithm, using the floor function, that appears in [***BWY***]. We show how some minor modifications to the gridding algorithm produces an algorithm which performs only an expected* $\Theta(\log^2 n)$ *distance calculations.*

§1 Introduction

The *closest pair* problem, finding the closest pair among a set of points, is one of the more basic problems in computational geometry. A large number of algorithms, approaching the problem from many different perspectives, exist for its solution. There are "practical" $O(n \log n)$ time algorithms that are relatively simple and easy to program: Shamos and Hoey's [**Sh**] divide-and-conquer algorithm and Hinrichs, Nievergelt, and Schorn's [**HNS**] sweep-line algorithm are representative. There are two randomized algorithms, one due to Rabin [**Ra**], the other to Weide [**We**], that both run in expected linear time on all inputs. There is a "theoretical" algorithm due to Fortune and Hopcroft [**FH**], really a modification of [**Ra**], that manages to get rid of the randomization by adding complexity and raising the running time. There are algorithms that solve the closest pair problem as a side effect of solving a more complicated problem such as constructing Voronoi diagrams [**SH**]. There is also an algorithm, due to Bentley, Weide, and Yao [**BWY**], that finds the closest pair by using a "gridding method", i.e. by using the floor function. When run on n points drawn I.I.D. $\mathbf{U[0,1]^2}$, (Independently, Identically Distributed from the uniform distribution over the unit square) this last algorithm requires $O(n)$ expected time.

Our primary purpose in this paper is to examine practical algorithms for solving the closest pair problem in the plane, algorithms that are robust and easy to implement yet run well on average, algorithms that would be implemented to solve real problems for large inputs. The first algorithm that we discuss, a modification of

[1] This work was supported in part by NSF Grant DCR-8605962 and Office of Naval Research Grant N00014-87-K-0460. It was also supported by a Chateaubriand fellowship from the French Ministère des Affaires Étrangères and by the European Community, Esprit II Basic Research Action Program Number 3075 (ALCOM). Author's current address: Algorithms' Project, Inria Rocquencourt, 78153, Le Chesnay Cedex, France.

the sweep line algorithm of Hinrichs, Nievergalt and Schorn [**HNS**], has these characteristics. It consists of two steps. The first step is a sort on the x-coordinates of the input points. The second step is a simple sweep through the points that can be implemented using only nine lines of Pascal code. The algorithm pays a price for its simplicity: the second stage has a worst case running time of $\Theta(n^2)$. It's actual expected running time is much better: we will show that, if the n input points are chosen I.I.D. $\mathbf{U}[0,1]^2$ then, for all but a negligible number of points, the i'th step of the sweep requires only expected $\Theta\left(\sqrt{\frac{n}{i}}\right)$ time. Summing over $i \leq n$ shows that the second stage of the sweep requires only $\sum_i \Theta\left(\sqrt{\frac{n}{i}}\right) = \Theta(n)$ expected time. Thus this is a very simple algorithm to implement physically that will run very fast on average. First sort the points by x-cordinate using whatever fine-tuned sorting algorithm you have on hand. Then run the sweep-stage over the sorted points. Our result shows that the dominant amount of work is performed in the sorting stage. This bound remains valid when distance is defined by any L_p metric.

The second algorithm we discuss, Bentley and Papadimitriou's [**BP**] projection method, is very similar to the first algorithm. This new algorithm is slightly more complicated than the one preceding it but the basic structure is the same: a first step sorting the points and a second step sweeping through them. It is possible to modify the analysis of the first algorithm to yield a $\Theta\left(\sqrt{\frac{n}{i}}\right)$ expected time bound for the i'th stage of the sweep step of this algorithm as well, again leading to a $\Theta(n)$ bound for the full sweep stage. This bound remains valid for any L_p metric. This algorithm's advantage over the preceding one is that in [**BP**] it was proved that it has a worst case running time of $O(n^{1.5})$ when closest is defined in the L_∞ sense as opposed to the $O(n^2)$ bound of the first algorithm. It's disadvantage is that it is slightly more complicated to program and has a higher overhead.

The third algorithm we discuss is a modification of the gridding algorithm presented by Bentley, Weide and Yao in [**BWY**]. In that paper they describe an algorithm that will find the closest pair (among other things) of n points chosen I.I.D. $\mathbf{U}[0,1]^2$ in expected $\Omega(n)$ time. Their algorithm required the use of an expected $\Theta(n)$ number of distance calculations. We will show how to modify their algorithm by adding a linear time sort so that only an expected $2\log^2 n + O(1)$ distance calculations will be performed.

There is a major qualitative difference between the sweep algorithms and the gridding algorithm: the use of the floor function. The sweep algorithms don't use the floor function while the gridding one does. From a complexity theory standpoint this is not a trivial distinction. For example, it is known that any *randomized* algorithm for solving the closest pair problem must use expected $\Omega(n \log n)$ time under the decision theory model of computation ([**MT**] and a reduction to element uniqueness). Yet [**BWY**]'s gridding algorithm uses only $O(n)$ expected time. This is because its use of the floor function means that it can no longer be analyzed under the decision tree model of computation. We conjecture that given n points chosen I.I.D. $\mathbf{U}[0,1]^2$ any deterministic algorithm that finds their closest pair must use expected time $\Omega(n \log n)$ time under the decision tree model of computation.

The structure of this paper is as follows. In §2 we will examine the distribution of the distance between the closest pair and introduce notation. In §3 we will describe and analyze the sweep line algorithm. In §4 we will discuss how to extend this analysis to [**BP**]'s projection algorithm. In §5 we will describe and analyze the gridding algorithm.

Note: Most of the proofs in this extended abstract require straightforward but long calculations. In the interest of brevity we will sometimes only provide a sketch of the full proof.

§2 Distribution of the Nearest Neighbor Distance

In this section we will derive upper and lower bounds on the distribution function of the nearest neighbor distance for n points Independently, Identically Distributed (I.I.D.) from the uniform distribution from the rectangle $\mathtt{R} = [0,x] \times [0,1]$, i.e. the rectangle with width x and height 1. We will need these bounds for

our later algorithmic analyses. As a corollary we will show that the expected nearest neighbor distance for n points chosen I.I.D. uniformly from $[0,1]^2$ is $\Theta(1/n)$. This should be contrasted to the $\Theta(1/n)$ expected distance between an arbitrary point and its nearest neighbor.

Formally, let $q_1, \cdots, q_n$ be chosen I.I.D. from the uniform distribution over $[0,x] \times [0,1]$. The random variable that we would like to analyze is

$$Z(q_1, \cdots, q_n) = \min_{1 \le i < j \le n} d(q_i, q_j)$$

where $d((x,y),(x',y')) = \sqrt{(x-x')^2 + (y-y')^2}$. We will prove the following:

Lemma 1: Let $x \in [0,1]$ and $\alpha \le x$. Choose n points $q_1, \cdots, q_n$, I.I.D. from the uniform distribution over $\mathtt{R} = [0,x] \times [0,1]$. Then

$$e^{-\pi n^2\alpha^2/2x}\left[1 + O(n\alpha^2/x + n^3\alpha^4/x^2)\right] \le Pr(Z(q_1, \cdots, q_n) \ge \alpha) \le e^{-\pi n^2\alpha^2/8x}\left[1 + O(n\alpha^2/x + n^3\alpha^4/x^2)\right] \tag{2.1}$$

Proof: We will derive the lower bound first. The event $Z \ge \alpha$ can be rewritten as

$$Z(q_1, \cdots, q_n) \ge \alpha \quad \Leftrightarrow \quad \forall k : \; q_k \notin \bigcup_{j<k} B(q_j, \alpha) \tag{2.2}$$

where $B(q_j, \alpha)$ is the Euclidean ball of radius α around q_j. We can think of this characterization as defining a point placement process. The points are placed in $\mathtt{R}$ in order, each under the constraint that it is outside the α-balls surrounding its predecessors. Viewing the situation from this perspective lets us write

$$\begin{aligned} \Pr(Z(q_1, \cdots, q_n) \ge \alpha) &= \Pr(Z(q_1,q_2) \ge \alpha) \cdot \Pr(Z(q_1,q_2,q_3) \ge \alpha \mid Z(q_1,q_2) \ge \alpha) \cdots \\ &= \prod_{k \le n} \Pr(Z(q_1, \cdots, q_k) \ge \alpha \mid Z(q_1, \cdots, q_{k-1}) \ge \alpha) \\ &= \prod_{k \le n} \Pr\left(q_k \notin \bigcup_{j<k} B(q_j, \alpha) \mid Z(q_1, \cdots, q_{k-1}) \ge \alpha\right). \end{aligned} \tag{2.3}$$

Because the q_k are uniformly distributed in the rectangle $\mathtt{R}$

$$\Pr\left(q_k \notin \bigcup_{j<k} B(q_j, \alpha) \mid Z(q_1, \cdots, q_{k-1}) \ge \alpha\right) = 1 - \frac{\mathbf{E}\left(Area\left[\bigcup_{j<k} B(q_j, \alpha) \cap \mathtt{R}\right] \;\middle|\; Z(q_1, \cdots, q_{k-1}) \ge \alpha\right)}{x}. \tag{2.4}$$

No matter where the q_j are placed we can always bound $Area\,(B(q_k, \alpha) \cap \mathtt{R}) \le \pi\alpha^2$ so we can upper bound the expected area in (2.4) and lower bound the product in (2.3) by

$$\prod_{k<m}\left[1 - \frac{k\pi\alpha^2}{x}\right] \le \Pr\left(Z(q_1, \cdots, q_m) \ge \alpha\right). \tag{2.5}$$

Substituting the asymptotic identity

$$\left[1 - \frac{\pi k\alpha^2}{x}\right] = \exp(-\pi k\alpha^2/x + O(k^2\alpha^4/x^2)) \tag{2.6}$$

into this inequality yields the lower bound of the lemma.

Now we derive the upper bound in (2.2). For this we use a second characterization of $Z \ge \alpha$:

$$Z(q_1, \cdots, q_n) \ge \alpha \quad \Leftrightarrow \quad \forall j \ne k, \quad B(q_j, \alpha/2) \cap B(q_k, \alpha/2) = \emptyset. \tag{2.7}$$

Since we've assumed that $\alpha \leq x$ it follows that for every point, q_j, at least one full quadrant (upper right, lower right, etc.) of $B(q_j, \alpha/2)$ is totally contained in R. If $Z(q_1, \cdots, q_{k-1}) \geq \alpha$ then combining this last fact with (2.7) gives

$$Area\left[\bigcup_{j<k} B(q_j,\alpha)\cap \mathrm{R}\right] \geq Area\left[\bigcup_{j<k} B(q_j,\alpha/2)\cap \mathrm{R}\right] \geq (k-1)\pi\alpha^2/4. \tag{2.8}$$

This lower bounds the expected size of the area.

$$\mathrm{E}\left(Area\left[\bigcup_{j<k} B(q_j,\alpha)\cap \mathrm{R}\right] \;\middle|\; Z(q_1,\cdots,q_{k-1}) \geq \alpha\right) \geq (k-1)\pi\alpha^2/4.$$

Inserting this inequality back into (2.3) and evaluating using (2.6) yields the lower bound of the lemma.

Q.E.D.

As an immediate consequence Lemma 1 easily lets us find the expected value of Z when the input points are chosen I.I.D. from the uniform distribution on the unit square (from now on we will write that the points are I.I.D. $\mathbf{U}[0,1]^2$).

Corollary 2: Let $q_1, \cdots, q_n$ be I.I.D. $\mathbf{U}[0,1]^2$ and $Z(q_1, \cdots, q_n)$ be the nearest neighbor distance. Then

$$\mathrm{E}(Z(q_1, \cdots, q_n)) = \Theta(1/n).$$

Proof: Apply Lemma 1 with $x = 1$ and integrate $\mathrm{E}(Z) = \int_0^{\sqrt{2}} \Pr(Z \geq \alpha)\, d\alpha$.

Q.E.D.

§3 The Sweep Line algorithm

§3.1 Description of the algorithm

The algorithm we consider here is the sweep line algorithm for computing the closest distance among points in the plane. Before presenting our analysis we give a quick description of the algorithm. For a more complete description as well as a correctness proof and implementation details see **[HNS]**.

Suppose we are given n points $p_i = (x_i, y_i), \quad i = 1 \ldots n$, in the plane. Sort them in increasing x order (arbitrarily breaking ties). We use the standard notation for order statistics. The i'th sorted point is denoted by $p_{(i)} = (x_{(i)}, y_{(i)})$ where

$$x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n-1)} \leq x_{(n)}.$$

The x-coordinate of p_i is the i-th smallest x-value. The y-coordinate of $p_{(i)}$ is the y coordinate that was associated with $x_{(i)}$. For example, if the original points were $p_1 = (5,3)$, $p_2 = (3,7)$, and $p_3 = (18,6)$, then the relabeled points are $p_{(1)} = (3,7)$, $p_{(2)} = (5,3)$, and $p_{(3)} = (18,6)$.

Let δ_i be the distance between between the closest pair among the first i points, i.e.

$$\delta_i = \min_{1\leq k<l\leq i} d\left(p_{(k)}, p_{(l)}\right) = Z(p_{(1)}, \cdots, p_{(k)}).$$

The closest pair distance problem is then seen to be the problem of determining δ_n. It follows from the definition of δ_i that

$$\delta_{i+1} = \min\left(\delta_i, \min_{1\leq k\leq i} d\left(p_{(i+1)}, p_{(k)}\right)\right). \tag{3.1}$$

Updating δ_i to δ_{i+1} only requires calculating the minimum among $d\left(p_{(i+1)}, p_{(k)}\right)$. Furthermore, if there is some $k \leq i$ such that $d\left(p_{(i+1)}, p_{(k)}\right) < \delta_i$ then for that k, $x_{(i+1)} - x_{(k)} < \delta_i$. Thus we only have to check the distance between $p_{(i+1)}$ and those points whose x-coordinates are in the interval $(x_{(i+1)} - \delta_i, x_{(i+1)}]$, a region which we will call the δ_i-interval (see Figure 1(a)).

There is a geometric fact (proven in [**HNS**]) that is essential to understanding how to insure the algorithm $O(n \log n)$ worst case behavior. Suppose we have sorted the points in the δ_i-interval by their y-coordinates. Furthermore suppose that

$$\exists k \leq i \quad \text{such that} \quad d\left(p_{(i+1)}, p_{(k)}\right) < \delta_i.$$

Then $p_{(k)}$ has to be one of the four points immediately above or one of the four points immediately below the point $p_{(i+1)}$ in the δ_i-interval. Thus, if the points in the δ_i-interval are stored in a balanced tree sorted by y-coordinate, we can calculate δ_{i+1} in $O(\log n)$ time by retrieving these eight points. All that remains is to describe how to update the δ_i-interval, i.e. update the y-sorted balanced tree so that it contains the points in the δ_{i+1}-interval. This is not difficult. First we insert $p_{(i+1)}$ into the balanced tree in $O(\log n)$ time. Then, using the fact that we already have the points sorted in increasing x-order (and have stored somewhere the index of the leftmost point in the δ_i-interval) we scan rightward deleting points from the tree until we enter the δ_{i+1}-interval (see Figure 1(b)). Since each point is deleted at most once during the execution of the algorithm the total time for deletion is $O(n \log n)$. Furthermore, since all other operations are $O(\log n)$ time per step we find that the entire second stage of the algorithm can be implemented in $O(n \log n)$ time.

The main purpose of this section is to show that, probabilistically, the performance of the second stage of this algorithm - scanning through the sorted points while updating δ_i - will be much better than the worst case analysis suggests. In fact, we will show that it is possible to dispense totally with complicated balanced tree data structures and *still* get very good behavior: if we compare $p_{(i+1)}$ to *all* of the points in the δ_i-interval the algorithm will still run very fast on the average. We give a short but full listing of code for this version of the algorithm in Figure 2. It is not hard to see that this simplified version of the algorithm will require $\Omega(n^2)$ time in the worst case when run on n, x-sorted points, e.g. if the points are all on the same vertical line. We will show that, on average, it will be much faster than that.

Mathematically, given n points $p_1, \ldots p_n$, chosen I.I.D. $\mathbf{U}[0,1]^2$, we set

$$\mathbf{N}_i = \text{ number of points in the } \delta_i\text{-interval .}$$

The amount of work needed to update δ_i to δ_{i+1} is $\mathbf{N}_i$ so the total amount of work performed by the algorithm is $\sum \mathbf{N}_i$. To prove that the algorithm runs in expected linear time we must show that

$$\mathbf{E}(\sum_{i=1}^{n} \mathbf{N}_i) = \sum_{i=1}^{n} \mathbf{E}(\mathbf{N}_i) = O(n). \tag{3.2}$$

This, along with a matching lower bound, will be an immediate consequence of the following theorem.

Theorem 3: Choose n points I.I.D. $\mathbf{U}[0,1]^2$. Let $\mathbf{N}_i$ be the number of points in the δ_i-interval . Then

$$\mathbf{E}(\mathbf{N}_i) = \Theta\left(\sqrt{\frac{n}{i}}\right), \qquad i > n^{1/3} \lg n. \tag{3.3}$$

More specifically

$$\sqrt{\frac{2}{3\pi}}\sqrt{\frac{n}{i}} + O\left(\sqrt{\frac{n}{i^3}}\right) \leq \mathbf{E}(\mathbf{N}_i) \leq \sqrt{\frac{16}{\pi}}\sqrt{\frac{n}{i}} + 4 + O\left(\sqrt{\frac{n}{i^3}}\right).$$

Combining this theorem with the fact that $\sum_{i \leq n} \sqrt{\frac{n}{i}} \sim 2n$ and $\sum_{i \leq n} \sqrt{\frac{n}{i^3}} = O(\sqrt{n})$ proves what we want:

n	$\sum_i \mathbf{N}_i$	$\frac{1}{n}\sum_i \mathbf{N}_i$
1000	1287.9 ± 437.6	1.287900 ± 0.4376
5000	6665.0 ± 2179.6	1.332996 ± 0.4359
10000	14254.9 ± 4981.7	1.425485 ± 0.4982
20000	27536.2 ± 9550.5	1.376812 ± 0.4775
30000	38866.0 ± 12462.7	1.295532 ± 0.4154
40000	56985.9 ± 17986.8	1.424649 ± 0.4497
50000	67907.2 ± 19342.2	1.358143 ± 0.3868

Table 1. For each value of n we generated 100 sets of n random points I.I.D. $\mathbf{U}[0,1]^2$. Each set was sorted and $\sum \mathbf{N}_i$ was calculated. The value in the second column is the average value of $\sum \mathbf{N}_i$ over all 100 point sets. The value in the third column is the value in the second normalized by division by n.

Corollary 4: The total amount of expected work, $\sum \mathbf{E}(\mathbf{N}_i)$, performed by the second stage of the sweep-line algorithm is $\Theta(n)$. More specifically

$$0.6514\ldots n + O(n^{2/3}\log^2 n) \;\le\; \mathbf{E}(\mathbf{N}_i) \;\le\; 6.2567\ldots n + O(n^{2/3}\log^2 n).$$

The constant on the right hand side is small so the algorithm should run relatively fast. How well does this work in practice? In Table 1 we provide statistics on $\mathbf{E}(\sum \mathbf{N}_i)$ collected by running the algorithm on random point sets. For each value of n we ran 100 random trials and averaged the results. The statistics seem to show that the running time of the algorithm is much closer to the lower bound than to the upper.

In §3.2 we will introduce some probabilistic notation and facts and in §3.3 we will prove the Theorem 4.

§3.2 Probabilistic Assumptions and Notation

We will assume that we are given n points $p_i = (x_i, y_i)$, $i = 1, \ldots, n$, chosen I.I.D. $\mathbf{U}[0,1]^2$. The p_i-s are then sorted by x-coordinate[2] and renamed $p_{(i)} = (x_{(i)}, y_{(i)})$ where

$$x_{(1)} < x_{(2)} < \cdots < x_{(n-1)} < x_{(n)}\,. \tag{3.4}$$

We define the random variables $\mathbf{X}_{(i)} = x_{(i)}$ and let $\delta_i = \min_{1 \le k < l \le i} d\left(p_{(k)}, p_{(l)}\right)$ as before. Given a real value X we say that the *α-interval* is the strip of width α to the left of the line $x = X$. The number of points in an α-interval will be denoted by

$$\begin{aligned} L(X,\alpha) &= \text{ number of } p_i \text{ in the } \alpha \text{ strip to the left of } X \\ &= |\{p_j \mid X - \alpha \le x_j \le X\}|. \end{aligned}$$

In this notation the number of points in the δ_i-interval is

$$\mathbf{N}_i = L(\mathbf{X}_{(i+1)}, \delta_i).$$

Directly from the definitions we see that $\mathbf{N}_i \le i$. Our goal is to analyze $\mathbf{E}(\mathbf{N}_i) = \mathbf{E}(L(\mathbf{X}_{(i+1)}, \delta_i))$. If α is a constant then it is not hard to see that $L(\mathbf{X}_{(i+1)}, \alpha) \mid \mathbf{X}_{(i+1)} = x)$ is a binomial[3] $\mathcal{B}(i, \alpha/x)$ random variable

[2] A technical quibble: this definition is not valid if there are two points with the same x-coordinate. We don't have to worry about this happening. If the points are chosen I.I.D. $\mathbf{U}[0,1]^2$ then two points sharing the same x-coordinate is a zero probability event which we can safely ignore.

[3] A random variable Y is binomial $\mathcal{B}(m,p)$ if $\Pr(Y = k) = \binom{m}{k} p^k (1-p)^{m-k}$ for $0 \le k \le m$.

and therefore easy to analyze. The random variable $L(\mathbf{X}_{(i+1)}, \delta_i) \mid \mathbf{X}_{(i+1)} = x,\ \delta_i = \alpha)$ is not binomial though because conditioning on $\delta_i = \alpha$ totally changes the distribution of the points. It restricts all pairs of points from being less than a distance α from each other and also requires that there is some pair that is *exactly* a distance α apart. A large portion of our analysis will be devoted to dealing with this conditionality.

First we examine the $\mathbf{X}_{(i)}$. We take advantage of the fact that the $\mathbf{X}_{(i)}$ have been extensively studied [De] because they are the *order statistics* of random variables. More specifically, since the $p_i = (x_i, y_i)$ are I.I.D. uniformly in the unit square the x_i have to be I.I.D. uniformly distributed in $[0, 1]$. The sorted x-coordinates, $\mathbf{X}_{(i)}$, are the order statistics of n uniformly distributed random variables on $[0, 1]$. The probability density $f_{(i)}(x)$, of $\mathbf{X}_{(i)}$ is well known (see [De], p. 17, for a derivation) and is given by the Beta distribution:

$$f_{(i)}(x) = \frac{n!}{(i-1)!(n-i)!}\, x^{i-1}(1-x)^{n-i} \quad x \in [0,1]. \tag{3.5}$$

Integration gives us the expectation and the variance:

$$\mathrm{E}(\mathbf{X}_{(i)}) = \frac{i}{n+1} \qquad \mathrm{Var}(\mathbf{X}_{(i)}) = \frac{i(n+1-i)}{(n+2)(n+1)^2}. \tag{3.6}$$

We can now use Chebyshov's inequality to restrict the range of $\mathbf{X}_{(i+1)}$:

Corollary 6:

$$\Pr\left(\mathbf{X}_{(i+1)} \notin \left[\tfrac{1}{2}\tfrac{(i+1)}{(n+1)}, \tfrac{3}{2}\tfrac{(i+1)}{(n+1)}\right]\right) \le \frac{4}{i+1}. \tag{3.7}$$

Proof. Chebyshov's inequality gives

$$\begin{aligned}\Pr\left(\mathbf{X}_{(i+1)} \notin \left[\tfrac{1}{2}\tfrac{(i+1)}{(n+1)}, \tfrac{3}{2}\tfrac{(i+1)}{(n+1)}\right]\right) &= \Pr\left(|\mathbf{X}_{(i+1)} - \mathrm{E}(\mathbf{X}_{(i+1)})| > \frac{(i+1)}{2(n+1)}\right) \\ &\le \mathrm{Var}(\mathbf{X}_{(i+1)})\left(\frac{2(n+1)}{(i+1)}\right)^2 \le \frac{4}{(i+1)}.\end{aligned}$$

§3.3 Proof of Theorem 4

Proof: We will prove that

$$\sqrt{\frac{2}{3\pi}}\sqrt{\frac{n}{i}} + O\left(\sqrt{\frac{n}{i^3}}\right) \le \mathrm{E}(\mathbf{N}_i) \le \sqrt{\frac{16}{\pi}}\sqrt{\frac{n}{i}} + 4 + O\left(\sqrt{\frac{n}{i^3}}\right), \qquad i > n^{1/3}\lg n.$$

We start by splitting $\mathrm{E}(\mathbf{N}_i)$ into the sum of two integrals:

$$\mathrm{E}(\mathbf{N}_i) = \int_{x \in D_i} \mathrm{E}\left(\mathbf{N}_i \mid \mathbf{X}_{(i+1)} = x\right) f_{(i+1)}(x)\, dx + \int_{x \notin D_i} \mathrm{E}\left(\mathbf{N}_i \mid \mathbf{X}_{(i+1)} = x\right) f_{(i+1)}(x)\, dx. \tag{3.8}$$

From Corollary 6 and the fact that $\mathbf{N}_i \le i$ we immediately see that

$$\int_{x \notin D_i} \mathrm{E}\left(\mathbf{N}_i \mid \mathbf{X}_{(i+1)} = x\right) f_{(i+1)}(x)\, dx \le i \cdot \Pr\left(\mathbf{X}_{(i+1)} \notin D_i\right) < 4.$$

We can therefore rewrite (3.8) as

$$\mathrm{E}(\mathbf{N}_i) = \int_{x \in D_i} \mathrm{E}\left(\mathbf{N}_i \mid \mathbf{X}_{(i+1)} = x\right) f_{(i+1)}(x)\, dx \ + O(1). \tag{3.9}$$

We will now show that, for $i > n^{1/3} \lg n$ and any $x \in D_i$,

$$\mathbf{E}\left(\mathbf{N}_i \mid \mathbf{X}_{(i+1)} = x\right) = \Theta\left(\sqrt{\frac{n}{i}}\right).$$

The proof of the theorem will follow by the insertion of this last statement into (3.9).

Henceforth we will assume that $\mathbf{X}_{(i+1)} = x$ for some constant $x \in D_i$. As before, R is the rectangular region to the left of the line $x = \mathbf{X}_{(i+1)}$: R $= [0, x] \times [0, 1]$. Conditioning on $\mathbf{X}_{(i+1)} = x$ we know that there are exactly i points in R. We can randomly relabel the points so that $q_1 = (x_1, y_1), \cdots, q_i = (x_i, y_i)$ are the ones in R. The crucial fact upon which our analysis will be based is that *after the points are relabeled the* $q_1, \cdots, q_i$ *are I.I.D. uniformly in* R.

We define new random variables

$$M_j = \begin{cases} 1 & \text{if } x - \delta_i < x_j \leq x \\ 0 & \text{otherwise.} \end{cases}$$

These are the indicator functions for the events q_j *is in the* δ_i*-interval,* so $\mathbf{N}_i = \sum_{j \leq i} M_j$. Because the q_j-s are I.I.D., linearity of the expectation operator yields[4]

$$\mathbf{E}(\mathbf{N}_i) = i\mathbf{E}(M_1). \tag{3.10}$$

Since q_1 is (uniformly) distributed in R we can find $\mathbf{E}(M_1)$ by integrating over the conditional expectation given that q_1 is fixed, i.e.

$$\mathbf{E}(M_1) = \frac{1}{x} \int_{y_1=0}^{1} \int_{x_1=0}^{x} \mathbf{E}\left(M_1 \mid q_1 = (x_1, y_1)\right) dx_1\, dy_1. \tag{3.11}$$

At first glance it might seem that all we've done is exchange one complicated integral, (3.8), for another, (3.11). In the next few lines we give a few simple transformations that enable (3.11) to be rewritten in a more pliable form.

$$\begin{aligned} \mathbf{E}(M_1) &= \frac{1}{x} \int_{y_1=0}^{1} \int_{x_1=0}^{x} \Pr\left(M_1 = 1 \mid q_1 = (x_1, y_1)\right) dx_1\, dy_1 \\ &= \frac{1}{x} \int_{y_1=0}^{1} \int_{x_1=0}^{x} \Pr\left(x - \delta_i < x_1 \leq x \mid q_1 = (x_1, y_1)\right) dx_1\, dy_1 \\ &= \frac{1}{x} \int_{y_1=0}^{1} \int_{x_1=0}^{x} \Pr\left(\delta_i \geq x_1 \mid q_1 = (x - x_1, y_1)\right) dx_1\, dy_1. \end{aligned} \tag{3.12}$$

The first equality follows from M_1 being an indicator function, the second from the definition of M_1, and the third from the transformation $x_1 \rightarrow (x - x_1)$.

Our plan is to bound $\mathbf{E}(M_1)$ by calculating upper and lower bounds on $\Pr(\delta_i \geq x_1 \mid q_1 = (x - x_1, y_1))$ and integrating over them in the last line of (3.12). Since $\delta_i = Z(q_1, \cdots, q_i)$ we really want bounds on

$$\Pr(Z(q_1, \cdots, q_i) \geq x_1 \mid q_1 = (x - x_1, y_1)).$$

If this random variable wasn't conditioned on the event $q_1 = (x - x_1, y_1)$ Lemma 1 would give us good bounds. But, as pointed out in the remark after Lemma 1, the proof of the lemma never used the fact that

[4] As we said before, all random variables and expectations in this part are conditioned on $\mathbf{X}_{(i+1)} = x$. For the sake of clarity we don't write this explicitly. For example, (3.10) should be read as $\mathbf{E}(\mathbf{N}_i | \mathbf{X}_{(i+1)} = x) = i\mathbf{E}(M_1 | \mathbf{X}_{(i+1)} = x)$.

q_1 was chosen from the uniform distribution and it would still be true even if q_1 was deterministically chosen. Therefore we can apply lemma 1 to find that

$$e^{-\pi n^2\alpha^2/2x}\left[1+O(n\alpha^2/x+n^3\alpha^4/x^2)\right] \le \Pr(\delta_i \ge x_1 \,|\, q_1=(x-x_1,y_1)) \le e^{-\pi n^2\alpha^2/8x}\left[1+O(n\alpha^2/x+n^3\alpha^4/x^2)\right] \tag{3.13}$$

whenever $x_1 \le x$ (this is the $\alpha \le x$ condition). Recall that we are assuming that x is fixed, $x \in D_i$. We can therefore write $x = ci/n$ where $1/2 \le c \le 3/2$. Recall too the theorem's assumption that $i > n^{1/3}\log n$. Combining these two assumptions we see that when $x_1 \le \sqrt{\lg n/in}$ then $x_1 \le x$ and we can use (3.13). We will now show how to use this fact to upper bound $\mathrm{E}(\mathrm{N}_i) = i\mathrm{E}(M_1)$. First we substitute $x = ci/n$ and integrate

$$\begin{aligned}\int_0^1 \Pr(\delta_i \ge x_1 \,|\, q_1=(x-x_1,y_1))dx_1 &= \int_0^{\sqrt{\lg n/in}} \Pr()dx_1 + \int_{\sqrt{\lg n/in}}^1 \Pr()dx_1 \\ &\le \int_0^{\sqrt{\lg n/in}} e^{-\pi i n x_1^2/2c}\left[1+O(nx_1^2+in^2x_1^4)\right]dx_1 + n^{-\Omega(\lg n)} \\ &= \sqrt{\frac{8c}{\pi in}}\left[1+O\left(\frac{1}{i}\right)\right].\end{aligned}$$

Inserting this back into (3.12) gives

$$\mathrm{E}(M_1) \le \frac{n}{ic}\int_0^1 \sqrt{\frac{8c}{\pi in}}\left[1+O\left(\frac{1}{i}\right)\right]dy_1 \le \sqrt{\frac{8}{\pi c}}\sqrt{\frac{n}{i^3}}+O\left(\frac{\sqrt{n}}{i^5}\right)$$

and therefore

$$\mathrm{E}(N_i) \le i\mathrm{E}(M_1) \le \sqrt{\frac{8}{\pi c}}\sqrt{\frac{n}{i}}+O\left(\sqrt{\frac{n}{i^3}}\right).$$

All probabilistic statements this section were conditioned on $\mathbf{X}_{(i+1)} = x \in D_i$. For these x, we know that $1/2 \le c \le 3/2$. Therefore we have actually proven that, for $x \in D_i$,

$$\mathrm{E}(\mathbf{N}_i \,|\, \mathbf{X}_{(i+1)} = x) \le \sqrt{\frac{16}{\pi}}\sqrt{\frac{n}{i}}+O\left(\sqrt{\frac{n}{i^3}}\right)$$

Reinserting this into (3.9) immediately proves the upper bound of the theorem. The lower bound is proved similarly. The only difference is that we use the left hand side of (3.13) and not the right hand one.

Q.E.D.

Note: Until now we have assumed that the distance function $d(p,q)$, is the Euclidean L_2 metric. Working through the proof of Theorem 3 one finds that it remains valid when $d()$ is replaced by the L_p ($p \ge 1$) metric

$$d_p(p_i,p_j) = (|x_i-x_j|^p + |y_i-y_j|^p)^{1/p}$$

or the L_∞ metric

$$d_p(p_i,p_j) = \max(|x_i-x_j|, |y_i-y_j|).$$

The only thing that changes in the proof is that π is replaced by ω_p, the area of the unit ball under the L_p metric:

$$\omega_p = \mathrm{Area}\{q \,|\, d_p((0,0),q) \le 1\}.$$

§4 The Projection Algorithm

In [**BP**] Bentley and Papadimitriou describe an algorithm for solving the "post-office" problem. Given n points, $p_1, \cdots, p_n$, the "post-office" problem is to preprocess them so that, given a new point, q, we can always find the point p_i which is closest to q, i.e.

$$\forall j, \qquad d(q, p_i) \leq d(q, p_j).$$

The preprocessing step proposed by [**BP**] sorts the p_j's by increasing x-coordinate. To find the closest point to a new point, q, they compare q to the points which are nearest to q by projection on the x axis. This is a two step procedure. They first use a binary search to find p_i such that $p_i.x < q.x \leq p_{i+1}.x$. They then, using their own phrase, "search out," q's nearest neighbor by always comparing q to the point whose x-coordinate is closest to q and to which q has not yet been compared. While doing this they keep track of d, the closest distance between q and all of the points it has been compared to so far. They terminate the search when q has been compared to all points p_j whose x-coordinates are within the range $[q.x - d, q.x + d]$. We call this algorithm NNX : NNX(q) returns the distance between q and its nearest neighbor.

The problem with this algorithm is that it can take $\Omega(n)$ time, e.g when all of the p_i lie on the same vertical line. In an effort to avoid this problem [**BP**] notes that there is nothing sacred about projection on the x-axis; we can just as easily write the algorithm so that it projects on the y-axis by "searching-out" the points whose y-coordinates are closest to q's y-coordinate. This algorithm, which we call NNY, also requires $\Omega(n)$ worst case time. We can also interleave projection on the x-axis with projection on the y-axis. That is, we can run the two versions, NNX and NNY, in lockstep: at each stage comparing q to the point with closest x-coordinate, and then to the point with closest y-coordinate This last version of the algorithm, which we call NN, has an interesting combinatorial worst case analysis. [**BP**] prove that if closest is defined by the L_∞ metric then if we calculate [5] NN(p_1), NN(p_2),...NN(p_{n-1}), and NN(p_n) the total time taken by the algorithm for these n calls will be $O(n^{3/2})$ as compared to an $\vee(n^2)$ worst case bound if we had run just the x-projection version NNX(p_1), NNX(p_2),...NNX(p_{n-1}), and NNX(p_n).

We now modify NN so that it finds the distance between the closest pair. Obviously one way of doing this would be to calculate all of the values NN(p_1), ..., NN(p_n) and report the minimum value. A more sophisticated technique uses information already calculated. First sort the points so that they are in x sorted order as before relabeling them $p_{(1)}, p_{(2)}, \cdots, p_{(n)}$:

$$p_{(1)}.x \leq p_{(2)}.x \leq \cdots \leq p_{(n)}.x.$$

We examine the $p_{(i)}$ in this x-sorted order. Suppose we have already calculated ν_i, the minimum distance between the first i points and all of the others:

$$\nu_i = \min(\text{NN}(p_{(1)}), \text{NN}(p_{(2)}), \ldots, \text{NN}((p_i))) = \min_{\substack{1 \leq k \leq i \\ 1 \leq l \leq n \\ k \neq j}} d(p_{(k)}, p_{(j)}),$$

We will only be interested in calculating NN($p_{(i+1)}$) if NN($p_{(i+1)}$) $< \nu_i$. Thus, if we can convince ourselves that NN($p_{(i+1)}$) $\geq \nu_i$, we can truncate the search performed by NN($p_{(i+1)}$). The truncation is implemented by stopping the execution of NN($p_{(i+1)}$) if there are no points in $\{(x, y) \,|\, p_{(i+1)}.x - \nu_i < x < p_{(i+1)}.x + \nu_i\}$ *or* if there are no points in $\{(x, y) \,|\, p_{(i+1)}.y - \nu_i < y < p_{(i+1)}.y + \nu_i\}$. We name this interleaved closest pair algorithm CP. [**BP**]'s result implies that CP takes $O(n^{3/2})$ time in the worst case when *closest* is defined by the L_∞ metric. They also point out that, while, in general, they can't extend this $O(n^{3/2})$ bound to the L_2 metric, in practice, the use of finite precision arithmetic does let them extend it.

[5] By calculating NN(p_j) we mean finding the point $p_i \neq p_j$, such that p_i is closest to p_j. This requires us to slightly modify our code so that it p_j is not reported as its own nearest neighbor.

To analyze this algorithm we need to define notation analogous to $L(X, \alpha)$ which, instead of counting the number of points in the α-strip *left* of X counts the number to the *right* of X. We set

$$\begin{aligned} R(X, \alpha) = &\text{ number of } p_i \text{ in the } \alpha \text{ strip to the right of } X \\ &= |\{p_j \mid X \leq x_j \leq X + \alpha\}|. \end{aligned}$$

During the truncated execution of NN($p_{(i)}$ algorithm CP will examine at most

$$2\left[L(\mathbf{X}_{(i+1)}, \nu_i) + R(\mathbf{X}_{(i+1)}, \nu_i)\right] = 2\,|\{j : j \neq i,\ |p_{(j)}.x - p_{(i)}.x| \leq \nu_i\}|$$

points. This is because at most $L(\mathbf{X}_{(i+1)}, \nu_i) + R(\mathbf{X}_{(i+1)}, \nu_i)$ points are examined during the x-projection part of NN($p_{(i)}$) because their x-coordinates are within ν_i of $p_{(i)}$-s x-coordinate. Furthermore at most one point will be examined in the y-projection part of $NN(p_{(i)})$ for each point examined in the x-projection part.

Using techniques very similar to those used to prove Theorem 3 we can prove

Theorem 7: Choose n points I.I.D. $\mathbf{U}[0,1]^2$. Then

$$\mathbf{E}(L(\mathbf{X}_{(i+1)}, \nu_i) + R(\mathbf{X}_{(i+1)}, \nu_i)) = O\left(\sqrt{\frac{n}{i}}\right), \qquad i > n^{1/3} \lg n.$$

Furthermore this result remains valid when the Euclidean distance function is replaced by the L_p $(p \geq 1)$ metric

$$d_p(p_i, p_j) = (|x_i - x_j|^p + |y_i - y_j|^p)^{1/p}$$

or the L_∞ metric

$$d_p(p_i, p_j) = \max(|x_i - x_j|, |y_i - y_j|).$$

Algorithm CP starts with a deterministic $O(n \lg n)$ preprocessing stage that sorts the points by x coordinate and also by y-coordinate.

It then performs an expected $\sum_{i \leq n} \mathbf{E}\left[L(\mathbf{X}_{(i+1)}, \nu_i) + R(\mathbf{X}_{(i+1)}, \nu_i)\right]$ amount of work in the truncated calls to NN(). The theorem tells us that all of the calls to NN() require at most $O(n)$ expected time. Thus this algorithm has the same time bounds as the sweep-line algorithm we analyzed in the previous section. This new algorithm has the advantage that we know – for the L_∞ distance function – that it's worst case running time is $O(n^{(3/2)})$ while the worst case running time for the sweep line algorithm was $O(n^2)$. It has the disadvantage of requiring more complicated code than the sweep-line algorithm and executing two sorts as opposed to the sweep-line's one sort.

In this section we only provided a sketch of the algorithm. For a full description and a complete analysis see [Go].

§5 A "Gridding" algorithm

In this section we describe another algorithm that efficiently finds the closest pair among a set of n points $p_1, \cdots, p_n$, that are chosen I.I.D. $\mathbf{U}[0,1]^2$. Unlike the algorithms in the previous two sections this new algorithm is not robust. It only works well when the input points are actually drawn from the unit square. Counterbalancing this flaw is that fact that it is a truly linear expected time algorithm. Furthermore it does very few costly distance calculations. More specifically the algorithm has a deterministic overhead of $\Theta(n)$ time (floor functions and radix sorting). The remainder of the work it performs, including all distance computations will be shown to take only expected $\Theta(\log^2 n)$ time.

This algorithm is a natural extension of the gridding algorithm found in [**BWY**]. There they present an expected $O(n)$ time algorithm for finding the Vornoi diagram (and therefore the closest pair) of n points chosen I.I.D. $\mathbf{U}[\mathbf{0},\mathbf{1}]^{\mathbf{2}}$. Their basic idea was to partition the unit square into n smaller squares, or cells, each with dimension $n^{-1/2} \times n^{-1/2}$. They then showed that to construct the Vornoi region associated with each point it is only necessary to examine examine the distances between the point and an expected constant number of points in the same and "neighboring" cells. This lets them construct the entire Vornoi diagram in expected $O(n)$ time using expected $O(n)$ distance computations.

In this paper we are examining closest pair algorithms so we don't need to calculate the entire Vornoi structure. We will show that it is possible to modify their algorithm in such a way that only an expected $O(\log^2 n)$ number of distance calculations are done. Given n points $p_1, \cdots, p_n$, I.I.D. $\mathbf{U}[\mathbf{0},\mathbf{1}]^{\mathbf{2}}$ we will set $M = \lfloor n/\log n \rfloor$. The algorithm essentially divides the unit square into $\Theta(M^2)$ smaller squares and only calculates the distance between points if they are in the same smaller square.

Now the specifics: First note that Lemma 1 tells us that

$$\Pr(Z(p_1, \cdots, p_n) \geq 1/2M) = n^{-\Omega(\log n)}. \tag{5.1}$$

We will demonstrate an efficient procedure that will find all pairs of points (p_i, p_j) such that

$$\max(|p_i.x - p_j.x|, |p_i.y - p_j.y|) < 1/2M. \tag{5.2}$$

Our algorithm will execute this procedure. If it finds that there are no pairs that satisfy (5.2) then it finds the closest pair by examining all $\binom{n}{2}$ pairs of points. (5.1) tells us that this will happen with such a small probability that this second part of the algorithm only takes $n^{-\Omega(\log n)}$ expected time.

A pair of points p_i, p_j will satisfy (5.2) only if one of the following conditions is satisfied:

(a) $\lfloor M * p_i.x \rfloor = \lfloor M * p_j.x \rfloor$ and $\lfloor M * p_i.y \rfloor = \lfloor M * p_j.y \rfloor$:
(b) $\lfloor M * p_i.x + 1/2 \rfloor = \lfloor M * p_j.x + 1/2 \rfloor$ and $\lfloor M * p_i.y \rfloor = \lfloor M * p_j.y \rfloor$:
(c) $\lfloor M * p_i.x \rfloor = \lfloor M * p_j.x \rfloor$ and $\lfloor M * p_i.y + 1/2 \rfloor = \lfloor M * p_j.y + 1/2 \rfloor$:
(d) $\lfloor M * p_i.x + 1/2 \rfloor = \lfloor M * p_j.x + 1/2 \rfloor$ and $\lfloor M * p_i.y + 1/2 \rfloor = \lfloor M * p_j.y + 1/2 \rfloor$.

We will show how to find all pairs of points that satisfy condition (a). Associate with each point p_i the ordered pair of integers $g(p) = (\lfloor M * p_i.x \rfloor, \lfloor M * p_i.y \rfloor)$. Now sort the points $p_1, \cdots, p_n$ lexicographically[6] using $g(p_i)$ as the key for point p_i. It is known [**AHO**] that n pairs of integers can be lexicographically sorted in $O(n)$ time if the components of the pairs are all less than n. This is certainly the case here because $\lfloor M * p_i.x \rfloor < n$ and $\lfloor M * p_i.y \rfloor < n$. Now notice that $g(p_i) = g(p_j)$ if and only if the pair p_i, p_j fulfills condition (a). Therefore finding all pairs of points that fulfill condition (a) is equivalent to finding all pairs of repeated keys in a sorted list. Let

$$Y_a = |\{(i,j) \mid i < j, \lfloor M * p_i.y \rfloor = \lfloor M * p_j.y \rfloor \text{ and } \lfloor M * p_i.x \rfloor = \lfloor M * p_j.x \rfloor\}|.$$

There will be Y_a such repeated pairs and we can find them in time $O(n) + Y_a$ by walking through the list. A pair of points will satisfy condition (a) with probability M^{-2} and there are $\binom{n}{2}$ such pairs so

$$\mathrm{E}(Y_a) = \binom{n}{2} M^{-2} = \frac{\log^2 n}{2} + O(1).$$

Therefore we can find all pairs of points that fulfill condition (a) in expected time $O(n)$ (without performing any distance calculations) and the total expected number of such pairs will be $\log^2 n/2 + O(1)$.

[6] The pair (X, Y) is lexicographically less than the pair (X', Y') if $X < X'$, or $X = X'$ and $Y < Y'$.

Similarly we can find all pairs of points fulfilling each of conditions (b), (c) and (d) in expected $O(n)$ time without performing any distance calculations and the expected number of points that ful¿fills each of these conditions is $\mathrm{E}(Y) = \log^2 n/2 + O(1)$.

Thus we have managed to find a superset of all pairs of points (some pairs will be found two or more times depending on how many conditions they fufill) that satisfy (5.2) and the expected number of pairs reported will be $2\log^2 n + O(1)$.

The total amount of time that it took to find these pairs was $O(n)$. While finding them no distance calculations were performed because the distance calculations are performed after the points are reported. Thus the entire algorithm takes expected $O(n)$ time and performs only an expected $O(\log^2 n)$ number of distance calculations.

Acknowledgements: The author would like to thank Jon Bentley for the many conversations and suggestions that he provided.

REFERENCES:

[AHU] A. Aho, J. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass. (1974)

[BP] J. L. Bentley and C. H. Papadimitriou, "A Worst-Case Analysis of Nearest-Neighbor Searching by Projection," *7th Int. Conf. on Automata, Languages and Programming,* (August 1980) 470-482.

[BWY] J.L. Bentley, B.W. Weide, and A.C. Yao, "Optimal Expected-Time Algorithms for Closest Point Problems," *ACM Trans. on Mathematical Software,* 6(4) (Dec. 1980) 563-580.

[De] Luc Devroye, *Non-Uniform Random Variate Generation,* Springer-Verlag, New York. (1986).

[FH] S. Fortune and J.E. Hopcroft, "A Note on Rabin's Nearest Neighbor Algorithm," *Information Processing Letters,* 8(1) (1979) 20-23.

[Go] M. J. Golin *Probabilistic Analysis of Geometric Algorithms (Thesis),* Princeton University Technical Report CS-TR-266-90. June 1990.

[HNS] Klaus Hinrichs, Jurg Nievergelt, and Peter Schorn, "Plane-Sweep Solves the Closest Pair Problem Elegantly," *Information Processing Letters,* **26** (January 11, 1988) 255-261.

[MT] U. Manber and M. Tompa "The Complexity of Problems on Probabilistic, Nondeterministic, and Alternating Decision Trees," *JACM,* **32**(3) (July 1985) 720-732.

[Ra] M.O. Rabin, "Probabilistic Algorithms," *Algorithms and Complexity: New Directions and Recent Results (J.F. Traub ed.),* (1976) 21-39.

[Sh] Michael Ian Shamos, "Computational Geometry," *Thesis (Yale),* (1978).

[SH] Michael Ian Shamos and Dan Hoey, "Closest Point Problems," *16'th Annual Symposium on Foundations of Computer Science (FOCS),* (1975) 151-161.

[We] Bruce W. Weide, "Statistical Methods in Algorithm Design and Analysis," *Thesis (Carnige-Mellon University). CMU-CS-78-142,* (August 1978).

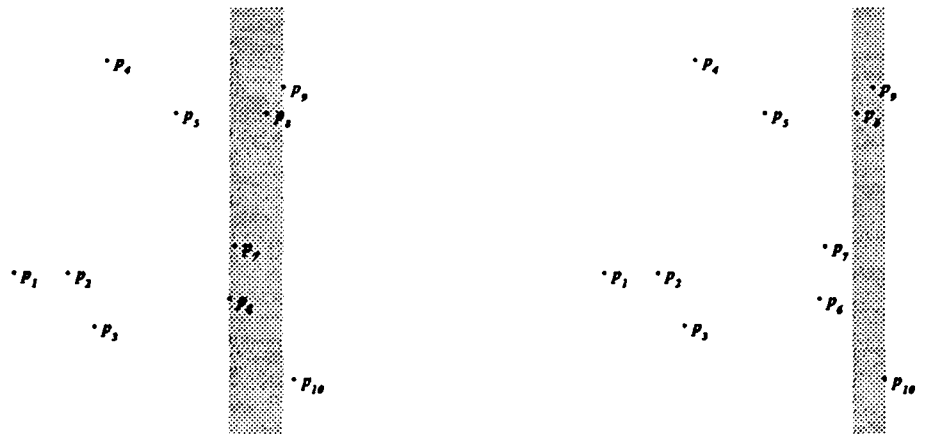

Figure 1. The minimum pair among all points preceeding p_9 is (p_1, p_2) so $\delta_8 = ||p_1 - p_2||$. In figure 1(a) the shaded region has width δ_8 and is what we called the δ_8-interval. While checking all the points in the δ_8-interval we find that $||p_9 - p_8|| < \delta_8$ and thus $\delta_9 = ||p_9 - p_8||$. We now find the δ_9 interval by scanning rightward from p_6 until we hit the first point (p_8) whose x-coordinate is within δ_9 of $p_{10}.x$. This gives us figure 1(b).

Figure 2. A Pascal listing of code that implements the linear scan for the closest pair. It is assumed that the points are sorted by x-coordinate in the array $p[]$ and that there is some distance function *dist*. *Left* and i point to the leftmost and rightmost points in the δ_i-interval . The **while** loop updates the points in the δ_i-interval . The inner **for** loop updates the current value of δ_i by comparing $p[i+1]$ to all of the points in the δ_i-interval .

```
left := 1;   delta := dist(p[1], p[2]);
for i := 1 to (n-1) do
    begin
        while (p[i+1].x - p[left].x) < delta
            left := left + 1;
        for j:= left to i do
            if dist(p[j], p[i+1]) < delta
                delta := dist(p[j], p[i+1]);
    end;
```

ISSN 0249-6399