A Simple Randomized Algorithm for All Nearest Neighbors

Soroush Ebadian* Hamid Zarrabi-Zadeh*

Abstract

Given a set P of n points in the plane, the all nearest neighbors problem asks for finding the closest point in P for each point in the set. The following folklore algorithm is commonly used for the problem in practice: pick a line in a random direction, project all points onto the line, and then search for the nearest neighbor of each point in a small vicinity of that point on the line. It is widely believed that the expected number of points in the vicinity of each point needed to be checked by the algorithm is $O(\sqrt{n})$. We prove this common conjecture in affirmative by providing a careful analysis showing that the expected number of comparisons made by the algorithm is $O(n\sqrt{n})$, if (log of) the spread of the points is bounded by a constant. We also present a matching lower bound, showing that our analysis is essentially tight.

1 Introduction

The *all nearest neighbors* problem considers finding, for a set P of n points in the plane, the nearest neighbor of each point in P. This is a fundamental and well-studied problem in computational geometry, with various applications, e.g., in statistics, similarity search, and image processing.

Several $O(n \log n)$ time algorithms are available for the problem. In particular, it is well-known that the Delaunay triangulation of P contains all edges connecting nearest neighbors. (See Figure 1.) Therefore, one can solve the all nearest neighbors problem in the plane in $O(n \log n)$ time using any of the optimal algorithms available for the Delaunay triangulation [2, 4]. In higher fixed dimensions, one can solve the problem in $O(n \log n)$ time using the algorithms of Clarkson [1] and Vaidya [6]. Both algorithms make use of spatial data partitioning trees, such as compressed quad-trees [5] and R-trees [3].

In this paper, we study an extremely simple randomized algorithm for the all nearest neighbors problem that uses no geometric data structure, and can be implemented in a few lines of code. It basically projects all the points onto a random line and searches for the nearest neighbor of each point in a small vicinity of that point on the line.

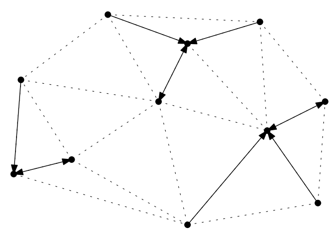

Figure 1: An example of the problem. Each point is connected to its nearest neighbor by an arrow. Dotted segments show the Delaunay triangulation edges.

The main contribution of this paper is a careful and tight analysis of the expected runtime of this randomized algorithm. More precisely, we show that the expected number of comparisons made by the algorithm is $O(n\sqrt{n} \cdot \sqrt{\log \Delta + 1})$, where Δ is the ratio of the furthest distance among all nearest neighbors, to the closest pair of the points. Note that Δ is upper bounded by the *spread* of the points, which is defined as the ratio of the maximum to the minimum distance of the points. In practice, this ratio is bounded by a constant. For example, when input coordinates are represented by rational numbers with 64-bit integers, $\Delta \leq 2^{128}$, and hence, $\sqrt{\log_2 \Delta}$ is at most 12.

The utter simplicity of the algorithm has made it a popular choice in cases where a fast implementation is preferable at the cost of slightly relaxing the optimal runtime. Due to its simplicity and removing the overhead of geometric data structures, the algorithm is even faster in practice compared to the other standard algorithms for the problem, such as Delaunay triangulations, when input data has only a few thousand points. Moreover, the algorithm finds the nearest neighbor of each point independently, after an initial step, which makes it highly flexible for parallel implementation.

2 Preliminaries

Let p and q be two point in the plane. We denote the Euclidean distance of p and q by $\|pq\|$. For a unit vector u in the plane, we denote by $\|pq\|_u$ the *projected distance* between p and q along direction u. In other words, $\|pq\|_u = (p-q) \cdot u = \|pq\| \cos\theta$, where θ is the angle between $\overrightarrow{pq}$ and u. Since $\cos\theta \leq 1$, we always have $\|pq\|_u \leq \|pq\|$.

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. Email: ebadian@ce.sharif.edu, zarrabi@sharif.edu.

3 The Algorithm

In this section, we present the simple randomized algorithm for the all nearest neighbors problem, and prove its correctness. The algorithm in its entirety is given in Algorithm 1. It takes as input a set $P = \{p_1, \dots, p_n\}$ of n points in the plane, and returns for each point p_i its nearest neighbor q_i in P.

Algorithm 1 All Nearest Neighbors($p_1, \dots, p_n$)

```
1: pick a random unit vector $u$
2: for $i$ from 1 to $n$ do
3:     $d_i \leftarrow \infty$
4:     for $p_j$ in increasing order of $\|p_i p_j\|_u \le d_i$ do
5:         if $\|p_i p_j\| < d_i$ then
6:             $d_i \leftarrow \|p_i p_j\|$, $q_i \leftarrow p_j$
7: return $q_1, \dots, q_n$
```

The algorithm works as follows. After picking a random unit vector, the algorithm processes each point p_i by checking the points in $P \setminus \{p_i\}$ in their increasing projected distance to p_i, while keeping the minimum Euclidean distance found so far in d_i. The search for the nearest neighbor of p_i is terminated whenever we reach a point whose projected distance to p_i is more than d_i.

An example of the execution of Algorithm 1 for a point p is illustrated in Figure 2. In this example, points are numbered in their increasing projected distance to p. The algorithm stops whenever it reaches the point p_5, whose projected distance to p is more than the best distance found so far, i.e. $\|pp_2\|$.

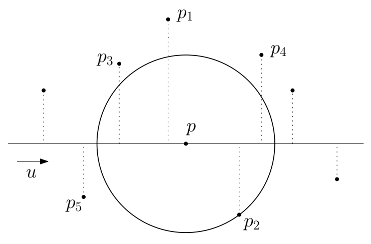

Figure 2: An example of the execution of Algorithm 1.

To quickly iterate over the points in their increasing projected distance from a point p, we can perform a simple preprocess step as follows. We select a line ℓ in direction u, project each point $p_i \in P$ to a point p'_i on ℓ, and sort the projected points along ℓ. Then, in the main loop for each point p_i, we keep two pointers on ℓ initially set to the points right before and after p'_i on ℓ, walking in opposite directions. At each step, we compare the distance of p'_i to the two projected points specified by the pointers, select whichever is smaller, and advance the corresponding pointer to the next one. This way, iterating over each point takes $O(1)$ time in the algorithm.

The correctness of the algorithm is proved in the following lemma.

Lemma 1 *For each point $p_i \in P$, the algorithm correctly finds the nearest neighbor of p_i.*

Proof. Fix a point p_i, and let q be the nearest point of p_i in P. Suppose by way of contradiction that the inner loop of the algorithm terminates on a point p_j, before reaching q. Thus, $\|p_i p_j\|_u < \|p_i q\|_u$, The inner loop terminates if $\|p_i p_j\|_u > d_i$, where d_i is the distance between p_i and a previously-visited point p_k. Therefore, we have $\|p_i q\| \ge \|p_i q\|_u > \|p_i p_j\|_u > d_i = \|p_i p_k\|$, which contradicts the fact that q is the closest point to p_i. □

4 The Analysis

Let $P = \{p_1, \dots, p_n\}$ be the set of input points. For $1 \le i \le n$, we denote by d_i the distance of p_i to its nearest neighbor in P. Let $P_i = \{p_j \in P - \{p_i\} : \|p_i p_j\|_u \le d_i\}$ be the set of points compared by the algorithm during the search for the nearest neighbor of p_i.

Let X be a random variable indicating the total number of comparisons made by Algorithm 1. We can decompose X into n^2 indicator variables

$$X_{i,j} = \begin{cases} 1 & \text{if } p_j \in P_i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $X_{i,i} = 0$ for all i, and $X = \sum_{1 \le i,j \le n} X_{i,j}$.

Lemma 2 *For all $1 \le i, j \le n$, $i \ne j$,*

$$\Pr\{X_{i,j} = 1\} \le \frac{d_i}{\|p_i p_j\|}.$$

Proof. Fix two points p_i and p_j in P. For all $r \ge d_i$, let C_r be a circle of radius r centered at p_i. Consider a strip S of width $2d_i$ enclosing C_{d_i} orthogonal to direction u (see Figure 3). Note that for all points $p \in P$, we have $p \in P_i$ if and only if p lies in S.

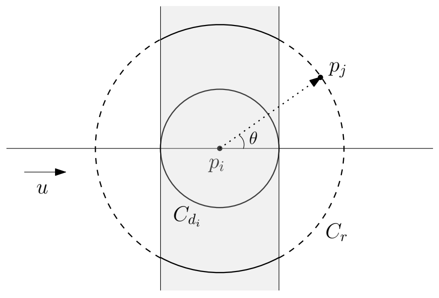

Figure 3: An illustration of Lemma 2.

Let $A(r)$ denote the length of $C_r \cap S$. The curvature of the arcs in $C_r \cap S$ decreases by increasing r, and hence, $A(r)$ is a decreasing function on $[d_i, \infty)$. Therefore, for all $r \geq d_i$, $A(r) \leq A(d_i) = 2\pi d_i$.

Since direction u is chosen uniformly at random, the angle θ between $\overrightarrow{p_i p_j}$ and u is uniformly chosen from the range $[0, 2\pi)$. In other words, p_j lies uniformly at random on a circle C_r with $r = \|p_i p_j\|$. Therefore, the event p_j lies in S corresponds to the fraction $A(r)/2\pi r$ of the points on C_r. Hence,

$$\Pr\{X_{i,j} = 1\} = \frac{A(r)}{2\pi r} \leq \frac{2\pi d_i}{2\pi r} = \frac{d_i}{r},$$

which completes the proof. □

Let $\mathbb{E}[X_{\cdot,j}] = \sum_{i=1}^{n} \mathbb{E}[X_{i,j}]$. An upper bound B on $\mathbb{E}[X_{\cdot,j}]$ yields an upper bound nB on $\mathbb{E}[X]$, because $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_{\cdot,j}]$. The rest of this section focuses on finding such an upper bound on $\mathbb{E}[X_{\cdot,j}]$.

Lemma 3 *For each $1 \leq j \leq n$, there is a permutation σ of $\{1, \ldots, n\}$ such that*

$$\mathbb{E}[X_{\cdot,j}] \leq 3 \sum_{i=1}^{n} \frac{d_{\sigma_i}}{\sqrt{\sum_{k=1}^{i} {d_{\sigma_k}}^2}}.$$

Proof. Let $p_{\sigma_1}, \ldots, p_{\sigma_n}$ be the points of P ordered in their increasing distance from p_j. Note that $p_{\sigma_1} = p_j$. For $1 \leq i \leq n$, let C_i be a circle of radius $d_{\sigma_i}/2$ centered at p_{σ_i}. For any pair of points p_{σ_i} and p_{σ_j}, $\|p_{\sigma_i} p_{\sigma_j}\| \geq \max\{d_{\sigma_i}, d_{\sigma_j}\} \geq (d_{\sigma_i} + d_{\sigma_j})/2$. Therefore, all circles C_i's are non-overlapping.

Fix an index $2 \leq i \leq n$. Let $\ell = \|p_{\sigma_i} p_j\|$, and $B_i = \{C_1, \ldots, C_i\}$. Every circle in B_i has radius at most $\ell/2$, and its center lies within distance ℓ to p_j. (See Figure 4.) Therefore, all circles in B_i fit in a disk C of radius $\frac{3}{2}\ell$ centered at p_j. As the circles are non-overlapping, the area of C must be at least as large as the total area of the circles in B_i. Therefore, $(\frac{3\ell}{2})^2\pi \geq \sum_{k=1}^{i} (\frac{d_{\sigma_k}}{2})^2\pi$, and thus, $\ell = \|p_{\sigma_i} p_j\| \geq \frac{1}{3}\sqrt{\sum_{k=1}^{i} {d_{\sigma_k}}^2}$. Now,

$$\mathbb{E}[X_{\cdot,j}] = \sum_{i=1}^{n} \mathbb{E}[X_{i,j}] \leq \sum_{i \in [n]-\{j\}} \frac{d_i}{\|p_i p_j\|} \quad \text{(by Lemma 2)}$$
$$= \sum_{i=2}^{n} \frac{d_{\sigma_i}}{\|p_{\sigma_i} p_j\|} \leq \sum_{i=2}^{n} \frac{3 \cdot d_{\sigma_i}}{\sqrt{\sum_{k=1}^{i} {d_{\sigma_k}}^2}},$$

which implies the lemma's statement. □

Based on the upper bound proved in Lemma 3, we define the following function:

$$f(a_1, \ldots, a_n) = \sum_{i=1}^{n} \frac{a_i}{\sqrt{\sum_{j=1}^{i} {a_j}^2}},$$

where $a_1, \ldots, a_n$ is a sequence of real numbers. We prove some useful properties of f in the next lemmas.

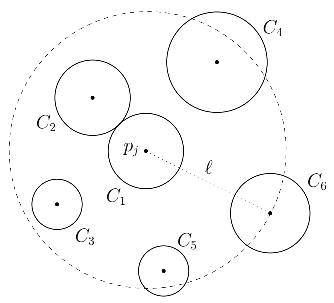

Figure 4: A set of non-overlapping circles $\{C_1, \ldots, C_6\}$.

Lemma 4 *Let $A = \{a_1, \ldots, a_n\}$ be a set of positive real numbers, and σ be a permutation of A. Then $f(\sigma)$ is maximized if σ is a non-decreasing sequence.*

Proof. Suppose by contradiction that f is maximized by a permutation σ which is not non-decreasing. Then, there exists an index i such that $x = \sigma_i > \sigma_{i+1} = y$. Let π be the ordering achieved by swapping σ_i and σ_{i+1}. As σ is an ordering that maximizes f, we have $f(\sigma) \geq f(\pi)$. Since the two permutations only differ in the i-th and $(i+1)$-th term, by the definition of f, and by setting $s = x^2 + y^2 + \sum_{j=1}^{i-1} {\sigma_j}^2$, we have

$$\frac{x}{\sqrt{s - y^2}} + \frac{y}{\sqrt{s}} \;\geq\; \frac{x}{\sqrt{s}} + \frac{y}{\sqrt{s - x^2}}$$

which yields

$$x \cdot \left[\frac{1}{\sqrt{s - x^2}} - \frac{1}{\sqrt{s}}\right]^{-1} \geq y \cdot \left[\frac{1}{\sqrt{s - y^2}} - \frac{1}{\sqrt{s}}\right]^{-1}.$$

Since function $z \cdot \left[\frac{1}{\sqrt{s-z^2}} - \frac{1}{\sqrt{s}}\right]^{-1}$ is decreasing in the range $(0, s)$, the last inequality implies that $x \leq y$. But this contradicts the fact that $x > y$. □

Lemma 5 *For any real number $a \geq 0$, and any integer $n \geq 1$,*

$$\sum_{i=1}^{n} \frac{1}{\sqrt{a + i}} \leq 2\sqrt{n}.$$

Proof. Since $\frac{1}{\sqrt{x}}$ is a decreasing function on $(0, +\infty)$, for any real number $b > 1$, we have

$$\int_{x=b-1}^{b} \frac{1}{\sqrt{x}} \;>\; \int_{x=b-1}^{b} \frac{1}{\sqrt{b}} \;=\; \frac{1}{\sqrt{b}}.$$

Therefore,

$$\sum_{i=1}^{n} \frac{1}{\sqrt{a + i}} \leq \int_{x=a}^{a+n} \frac{1}{\sqrt{x}} dx = 2(\sqrt{a + n} - \sqrt{a}) \leq 2\sqrt{n},$$

where the last inequality follows from the fact that $\sqrt{x + y} \leq \sqrt{x} + \sqrt{y}$, for all $x, y \geq 0$. □

Lemma 6 *Given real numbers $a_1, \ldots, a_n$ with $1 \leq a_i \leq c$, for some constant $c \geq 1$,*

$$f(a_1, \ldots, a_n) \leq 2b\sqrt{n}\sqrt{\log_b c + 1}.$$

for all $b > 1$.

Proof. Let $\hat{a}_i = b^{\lfloor \log_b a_i \rfloor}$. Since $b > 1$ and $a_i \geq 1$, we have $\hat{a}_i \leq a_i < b \cdot \hat{a}_i$. Therefore,

$$\begin{aligned} f(a_1, \ldots, a_n) &= \sum_{i=1}^{n} \frac{a_i}{\sqrt{\sum_{j=1}^{i} {a_j}^2}} \\ &\leq \sum_{i=1}^{n} \frac{b \cdot \hat{a}_i}{\sqrt{\sum_{j=1}^{i} \hat{a}_j^2}} = b \cdot f(\hat{a}_1, \ldots, \hat{a}_n). \end{aligned}$$

By Lemma 4, $f(\hat{a}_1, \ldots, \hat{a}_n)$ is maximized when $\hat{a}$'s are sorted non-decreasingly. Let $s_i = |\{j : \lfloor \log_b a_j \rfloor = i\}|$, for all $i \in \{1, 2, \ldots, \lfloor \log_b c \rfloor\}$. Then

$$\begin{aligned} f(\hat{a}_1, \ldots, \hat{a}_n) &\leq \sum_{i=0}^{\lfloor \log_b c \rfloor} \sum_{j=1}^{s_i} \frac{b^i}{\sqrt{j \cdot b^{2i} + \sum_{k=0}^{i-1} s_k \cdot b^{2k}}}, \\ &= \sum_{i=0}^{\lfloor \log_b c \rfloor} \sum_{j=1}^{s_i} \frac{1}{\sqrt{j + \sum_{k=1}^{i-1} s_k \cdot b^{2(k-i)}}}, \end{aligned}$$

which by Lemma 5 is at most $2\sum_{i=0}^{\lfloor \log_b c \rfloor} \sqrt{s_i}$. As $\sum s_i = n$, the last sum is maximized at equality. Therefore,

$$\sum_{i=0}^{\lfloor \log_b c \rfloor} \sqrt{s_i} \leq \sum_{i=0}^{\lfloor \log_b c \rfloor} \sqrt{\frac{n}{\lfloor \log_b c \rfloor + 1}} \leq \sqrt{n} \cdot \sqrt{\log_b c + 1}.$$

Thus, $f(a_1, \ldots, a_n) \leq 2b\sqrt{n}\sqrt{\log_b c + 1}$. □

Now, we have all the ingredients needed to prove the main theorem of this section.

Theorem 7 *The expected runtime of Algorithm 1 on any set of n points is $O(n\sqrt{n} \cdot \sqrt{\log \Delta + 1})$, where $\Delta = \max_i \{d_i\} / \min_i \{d_i\}$.*

Proof. By Lemma 3, $\mathbb{E}[X_{\cdot,j}]$ is upper bounded by $f(\sigma_1, \ldots, \sigma_n)$ for some permutation σ of $\{d_1, \ldots, d_n\}$. Scaling all variables by a constant does not change $f(\sigma_1, \ldots, \sigma_n)$. Therefore, we can assume w.l.o.g. that $1 \leq \sigma_i \leq \Delta$ for all i. By setting $b = 2$ and $c = \Delta$ in Lemma 6, we get

$$\mathbb{E}[X_{\cdot,j}] \leq 12\sqrt{n}\sqrt{\log_2 \Delta + 1}.$$

Therefore, $\mathbb{E}[X] = \sum_{j=1}^{n} \mathbb{E}[X_{\cdot,j}]$ is upper bounded by $12n\sqrt{n}\sqrt{\log_2 \Delta + 1}$, which completes the proof. □

4.1 Lower Bound

In this section, we show that the analysis presented in Section 4 is essentially tight by providing a lower bound example on which Algorithm 1 has a matching expected runtime. Our example is simply formed by the points of a $\sqrt{n} \times \sqrt{n}$ square lattice. The nearest neighbor to each point in this lattice has distance exactly one, and hence, $\Delta = 1$ in this case. The following theorem proves a lower bound of $\Omega(n\sqrt{n})$ on the expected runtime of the algorithm on this example, which matches the upper bound of $O(n\sqrt{n})$ proved in the previous section.

Theorem 8 *The expected runtime of Algorithm 1 on a $\sqrt{n} \times \sqrt{n}$ square lattice is $\Omega(n\sqrt{n})$.*

Proof. Let $P = \{p_1, \ldots, p_n\}$ be the set of points on the lattice, and let u be the random unit vector chosen by the algorithm. The nearest neighbor to each point in P has distance one. Therefore, $\mathbb{E}(|\{(p_i, p_j) : \|p_i p_j\|_u \leq 1\}|)$ is a lower bound on the expected runtime of the algorithm.

We first claim that $\Pr\{\|p_i p_j\|_u \leq 1\} \geq \frac{1}{\pi \cdot \|p_i p_j\|}$, for all $1 \leq i, j \leq n$. Fix two points p_i and p_j. Let ℓ be a line in direction u passing through p_i, and let p'_j be the projection of p_j on ℓ. Therefore, p_i, p_j, and p'_j form a right triangle. (See Figure 5.) Now, $\|p_i p_j\|_u \leq 1$ holds if and only if

$$\angle p_i p_j p'_j = \arcsin(\frac{\|p_i p'_j\|}{\|p_i p_j\|}) \leq \arcsin(\frac{1}{\|p_i p_j\|}).$$

As $\angle p_i p_j p'_j$ is chosen randomly, and $\arcsin(x) > x$ for all $0 < x \leq 1$, we have

$$\Pr\{\|p_i p_j\|_u \leq 1\} \geq \frac{1}{\pi} \arcsin(\frac{1}{\|p_i p_j\|}) > \frac{1}{\pi \cdot \|p_i p_j\|}.$$

Every two points in the lattice have distance at most $2\sqrt{n}$. Therefore, $\Pr\{\|p_i p_j\|_u \leq 1\} > \frac{1}{2\pi\sqrt{n}}$. Thus,

$$\mathbb{E}(|\{(p_i, p_j) : \|p_i p_j\|_u \leq 1\}|) > \frac{n(n-1)}{2\pi\sqrt{n}},$$

and hence, the expected runtime of the algorithm is $\Omega(n\sqrt{n})$. □

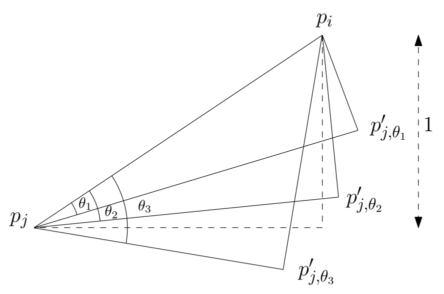

Figure 5: Projection of p_j in different directions.

5 Conclusions

In this paper, we analyzed an extremely simple randomized algorithm for the all nearest neighbors problem. We proved that the algorithm has $O(n\sqrt{n})$ expected runtime if the spread of the points is bounded by a constant.

Our analysis can be extended in a natural way to the case of general L_p metric, yielding the same expected running time. For higher d-dimensional spaces, we conjecture that the expected runtime of the algorithm is $O(n^{2-\frac{1}{d}}\,\mathrm{poly}(\log \Delta))$. We can also extend the algorithm to report k nearest neighbors of each point. While our analysis implies an upper bound of $O(k\sqrt{n})$ on the expected number of comparisons made by the algorithm for each point, it is intriguing to obtain a tighter analysis for this version of the problem.

References

[1] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, Nov 1983.

[2] S. Fortune. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean geometry*, pages 225–265, 1995.

[3] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf.*, pages 47–57, 1984.

[4] D.-T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *Int. J. of Comput. & Inform. Sci.*, 9(3):219–242, 1980.

[5] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[6] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4(2):101–115, 1989.