# All Nearest Neighbours via Quadtrees

Kasturi Varadarajan

May 2, 2013

We are given a set $P$ of $n$ points in the plane. We assume that the *spread* of the point set $P$, the ratio of the largest interpoint distance to the smallest interpoint distance, is bounded by a polynomial in $n$, say $n^5$. We describe a quadtree based algorithm that finds, for each $q \in P$, its closest point in $P \setminus \{q\}$. We will show that the running time of the algorithm is $O(n \log n)$. For an $O(n \log n)$ algorithm without this assumption on the spread, see the original paper of Vaidya [1].

## 1 Quadtree Construction

We first construct the quadtree corresponding to $P$ – this will be a 4-ary tree, where each internal node can have up to four children. Each node of the quadtree will be associated with a square. The set of squares at nodes that are at depth $j$ from the root will be denoted by $F_j$. It will be convenient to describe the tree in a top-down fashion, starting from the root. The root corresponds to any smallest axis-parallel square containing the point set $P$; $F_0$ is a singleton set containing just this square. For this square $\Box$, let $\mathrm{pts}(\Box) = P$.

Now if $|P| \geq 2$, we will partition this root square into four equal squares, which will be the elements of $F_1$. In general, suppose $j \geq 1$, and we have already obtained $F_0, \ldots, F_{j-1}$; that is, the quadtree has been described upto depth $j-1$. We construct $F_j$ as follows: Each $\Box \in F_{j-1}$ with $|\mathrm{pts}(\Box)| \leq 1$ will be a leaf of the quadtree. For each $\Box \in F_{j-1}$ with $|\mathrm{pts}(\Box)| > 1$, we partition $\Box$ into four equal squares $\Box_1$, $\Box_2$, $\Box_3$ and $\Box_4$, which will become the children of $\Box$. We parttion $\mathrm{pts}(\Box)$ into four sets $\mathrm{pts}(\Box_1)$, $\mathrm{pts}(\Box_2)$, $\mathrm{pts}(\Box_3)$, and $\mathrm{pts}(\Box_4)$ so that $\mathrm{pts}(\Box_i) \subseteq \mathrm{pts}(\Box) \cap \Box_i$. The squares $\Box_1$, $\Box_2$, $\Box_3$, and $\Box_4$ are added to $F_j$. See Figure 1.

At each node $\Box$ of the quadtree, we store the information determining the square itself, plus a representative point $\mathrm{rep}(\Box) \in \mathrm{pts}(\Box)$ if $\mathrm{pts}(\Box)$ is non-empty. Thus, for a leaf square $\Box$ with $|\mathrm{pts}(\Box)| = 1$, the representative $\mathrm{rep}(\Box)$ will be the only point in $\mathrm{pts}(\Box)$. A leaf square $\Box$ with $|\mathrm{pts}(\Box)| = 0$ has no representative point. Notice that any internal node $\Box$ has $|\mathrm{pts}(\Box)| \geq 2$, and thus has a representative point.

Our assumption about the spread being polynomially bounded implies that the depth of the quadtree is $O(\log n)$. This implies that the total number of nodes in the quadtree is $O(n \log n)$. This is because the number of squares in any $F_j$ is $O(n)$ (Why?). Notice that a node $\Box$ in the quadtree does not explicitly store $\mathrm{pts}(\Box)$; it only stores $\mathrm{rep}(\Box)$. Finally,

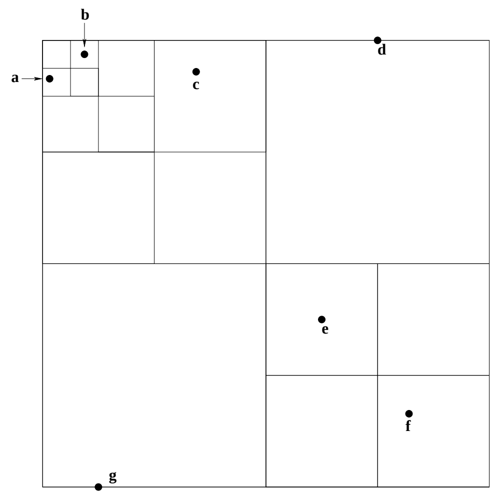


Figure 1: Quadtree for the set of seven points

it is not too hard to see that constructing the quadtree in the straightforward way takes $O(n \log n)$ time.

## 2 Nearest Neighbor Queries

Now let us see how we can use the quadtree to find, for each $q \in P$, a nearest point in $P \setminus \{q\}$. This is accomplished by the query algorithm Query($q$). Let $L$ denote the number of levels in the quadtree. The algorithm explores $E_0, E_1, \dots, E_L$ where $E_j \subseteq F_j$. We start with $E_0 = F_0$. To compute $E_j$, we first compute $\text{best}_{j-1}$, the closest point to $q$ among the representatives of squares in $E_0 \cup E_1 \cup \cdots \cup E_{j-1}$. Notice that for $j \geq 2$ this is simply the closest among $\text{best}_{j-2}$ and the representatives of squares in $E_{j-1}$, so this can be done in $|E_{j-1}|$ time. We then look at every child of a square in $E_{j-1}$ and include it in $E_j$ provided the distance of $q$ from the square is at most $d(q, \text{best}_{j-1})$. Here the distance of $q$ from square $\Box$ is the minimum distance of $q$ to any of the uncountably infinitely many points in $\Box$.

The running time of the query algorithm is $O(|E_0| + |E_1| + \cdots + |E_L|)$. As an aside, the query algorithm can terminate once some $E_j$ is empty, since subsequent $E_i$'s will be empty as well.

**Algorithm 1** Query($q$)

```
E_0 ← F_0
for all j ← 1 to L do
    best_{j-1} ← closest point to q in {rep(□)| □ ∈ E_0 ∪ E_1 ··· E_{j-1}} − {q}
    E_j ← {□ | □ is child of some square in E_{j-1} and d(q,□) ≤ d(q, best_{j-1})}
Return best_{L+1}.
```

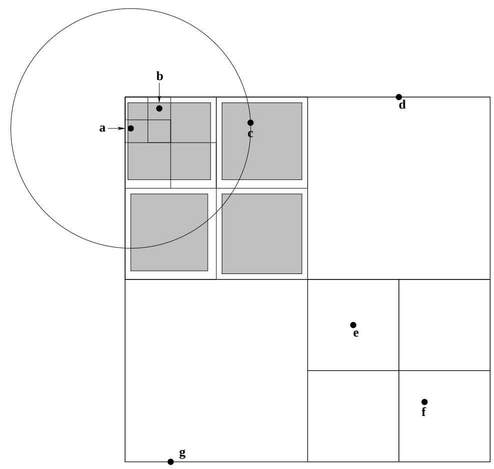


Figure 2: Assume that the representative point in any square is the last point by alphabetic order. Then in this example for Query($a$), $\text{best}_1 = c$. This is used to compute the squares in $E_2$, which are the shaded ones that intersect the circle.

The algorithm returns the closest point among the representatives of the squares in any of the $E_j$'s. That the call to Query($q$) actually returns a closest point in $P \setminus \{q\}$ is implied by the following lemma. The proof of the lemma was done in class but is omitted here. Note that the lemma implies that Query($q$) does explore the leaf cell containg a closest point $q'$ to $q$, and the representative of such a cell is of course $q'$ itself.

**Lemma 2.1** *Fix* $1 \leq j \leq L$. *Then* $\Box \in F_j$ *belongs to* $E_j$ *if and only if* $d(q, \Box) \leq d(q, best_{j-1})$.

# 3 Running Time for All Queries

Let us bound the running time for performing Query($q$) over all $q \in P$. For this, it will be convenient to denote by $E_j(q)$ the set $E_j$ encountered within Query($q$). We partition $E_j(q)$ into two sets: $\text{Near}_j(q) = \{\Box \in E_j(q) \mid d(q, \Box) \leq 10 * \text{diam}(\Box)\}$, and $\text{Far}_j(q) = E_j(q) \setminus \text{Near}_j(q)$. Note that $\text{Near}_j(q)$ is the set of squares in $E_j$ whose distance from $q$ is within 10 times the diameter of a level $j$ square. By a simple packing argument, $|\text{Near}_j(q)| \leq c$, where $c$ is some constant. On the other hand, the size of $\text{Far}_j(q)$ need not be bounded by a constant, particularly when $\text{Near}_j(q)$ is non-empty. Fix $0 \leq j \leq L$. We have

$$
\begin{aligned}
\sum_{q\in P} |E_j(q)| &= \sum_{q\in P} |\mathrm{Near}_j(q)| + \sum_{q\in P} |\mathrm{Far}_j(q)| \\
&\leq \sum_{q} c + \sum_{\square\in F_j} |\{q \in P \mid \square \in \mathrm{Far}_j(q)\}| \\
&\leq \sum_{q} c + \sum_{\square\in F_j} c' \\
&= O(n) + O(|F_j|).
\end{aligned}
$$

Here, we use the fact that for any $\square \in F_j$, the number of points $q$ for which it is far (belongs to $\mathrm{Far}_j(q)$) is bounded by a constant $c'$. This fact is not obvious at all, but I still hope it is true, and you should try to prove it.

The overall running time of the query algorithm is then

$$\sum_j \sum_q |E_j(q)| = \sum_j O(n + |F_j|) = O(n \log n).$$

# References


[1] Pravin M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4:101–115, 1989.