Lecture 2: “A Las Vegas Algorithm for finding the closest pair of points in the plane”

Sotiris Nikoletseas
Professor

CEID - ETY Course
2017 - 2018
Definition: A Las Vegas algorithm is a randomized algorithm that always returns the correct result.

However, its running time may change, since this time is actually a random variable.
The closest pair of points problem

Definition: Given a set of points $P$ in the plane, find the pair of points closest to each other. Formally, return the pair of points, realizing (the closest possible inter-point distance):

$$CP(P) = \min_{p,q \in P} \|pq\|$$

where $\|pq\|$ denotes the Euclidean distance of points $p, q$.

Note: The problem can naively be solved in $O(n^2)$ time, by computing all $\binom{n}{2}$ inter-point distances.

Here, we will present a Las Vegas algorithm of $O(n)$ expected time.
The grid $G_r$ - I

- For $r$ positive and a point $p = (x, y)$ in $\mathbb{R}^2$, let $G_r(p)$ the point $\left(\lfloor \frac{x}{r} \rfloor, \lfloor \frac{y}{r} \rfloor \right)$
  e.g. $p = (4.5, 7.6)$ and $r = 2 \Rightarrow G_2(p) = (2, 3)$

- We call $r$ the width of grid $G_r$.

- Actually, the grid $G_r$ partitions the plane into square regions, which we call grid cells. Formally, a grid cell is defined, for $i, j \in \mathbb{Z}$, by the intersection of the four half-planes: $x \geq ri, x < (r + 1)i, y \geq rj, y < (r + 1)j$
The partition of points in $P$ into subsets by the grid $G_r$ is denoted by $G_r(P)$. Formally, two points $p, q \in P$ belong into the same set of the $G_r(P)$ partition iff they belong into the same grid cell. Equivalently, they are mapped into the same grid point $G_r(p) = G_r(q)$.

- We call a block of continuous grid cells a grid cluster.
A data structure for the grid

- Note: every grid cell $C$ of $G_r$ has a unique ID. Indeed, let $p = (x, y)$ be any point in cell $C$ and consider $id_p = \left(\lfloor \frac{x}{r} \rfloor, \lfloor \frac{y}{r} \rfloor\right)$, which is actually the unique ID $id_C$ of cell $C$, since only points in the cell $C$ are mapped to $id_C$.

- This allows an efficient storage of the set $P$ of points inside a grid, as follows:
  1. given a point $p$, we compute $id_p$
  2. for each unique id (corresponding to a cell) we maintain a linked list of all the points in that cell
  3. we can thus fetch the data (the points) for a cell by hashing, in constant time.
(i.e. we store pointers to all those linked lists in a hash table, where each list is indexed by its unique id).
An intermediate decision problem

We will employ the following intermediate result.

**Lemma 1:** Given a set \( P \) of \( n \) points in the plane, and a distance \( r \), one can check in linear time whether \( CP(P) < r \) or \( CP(P) \geq r \)

**Proof:**

- We store the points of \( P \) in the grid \( G_r \) (i.e. for every non-empty grid cell we maintain a linked list of the points inside it)
- Thus, adding a new point \( p \) takes constant time (compute \( id(p) \), check if \( id(p) \) already exists in the hash table; if it exists just add \( p \) to it; otherwise, create a new linked list for the cell with this ID and store \( p \) in it)
- Totally (for all \( n \) points) this will take \( O(n) \) time.
Note: If any grid cell in $Gr(P)$ contains more than, say, 9 points of $P$, then $CP(P) < r$.

Indeed:

- Consider a cell $C$ with more than 9 points of $P$
- Partition $C$ into 3x3 equal squares
- Clearly, one of these 9 squares must contain two (or more) points of $P$ and let $C'$ this square
- The diameter of $C' = diam(C') = \frac{diam(C)}{3} = \frac{\sqrt{r^2 + r^2}}{3} < r$
- Thus, at least two points of $P$ in $C'$ are at distance smaller than $r$ from each other

Note: The 9 points argument is indicative (e.g. we could consider 16 points and partition the cell into 4x4 equal squares).
Proof of decision lemma 1 (continued)

- Thus, when we insert a point \( p \), we can fetch all \( P \) points already inserted, for the cell of \( p \), as well as its 8 adjacent cells.
- All those cells must contain at most 9 points of \( P \) each (otherwise we would have stopped knowing that \( CP(P) < r \)).
- Let \( S \) the set of all those points, so \( |S| \leq 9 \cdot 9 = \Theta(1) \).
- Thus, we can compute by brute force in \( O(1) \) time the closest point to \( p \) in \( S \). If its distance to \( p \) is \( < r \) then we stop (with \( CP(P) < r \)); otherwise we continue with the other (at most) 80 points.
- Overall this takes \( O(n) \) time.

(end of Lemma 1 proof)
An intuitive way of computing $CP(P)$

- Permute arbitrarily the points in $P$
- Let $P = \langle p_1, \ldots, p_n \rangle$ the resulting permutation
- Let $r_{i-1} = CP(\{p_1, \ldots, p_{i-1}\})$ i.e., the “partial knowledge” of $CP(P)$ after exposing the first $i - 1$ points of the permutation ($P_{i-1} = \langle p_1, \ldots, p_{i-1} \rangle$)
- We check whether $r_i < r_{i-1}$ by calling the algorithm of Lemma 1 on $P_i$ and $r_{i-1}$

**NOTE:** A grid $G_r$ can only answer (via Lemma 1) queries of the type $CP(P) < r$, while for finer queries $CP(P) < r' < r$ a finer granularity grid must be rebuilt!
Thus, when “exposing” one more point (i.e. going from $P_{i-1} = \langle p_1, \ldots, p_{i-1} \rangle$ to $P_i = \langle p_1, \ldots, p_{i-1}, p_i \rangle$) we distinguish two different cases:

- **THE BAD CASE:** If $r_i < r_{i-1}$ a new, finer granularity grid $G_{r-1}$ must be built, and insert points $p_1, \ldots, p_i$ to it. This takes obviously $O(i)$ time.

- **THE GOOD CASE:** If $r_i = r_{i-1}$, i.e. the distance of the closest pair does not change by adding $p_i$. In this case, we do not need to rebuild the grid and inserting the new point $p_i$ takes constant time.
Intuitive remark on time complexity

- No change in closest pair distance after a point insertion $\Rightarrow$ constant time needed
- A change after inserting point $i \Rightarrow O(i)$ time needed (to rebuild the data structure)
- If the closest pair distance never changes $\Rightarrow O(1)$ cost $n$ times $\Rightarrow O(n)$ time needed
- If it changes all the time $\Rightarrow O\left(\sum_{i=3}^{n} i\right) = O(n^2)$ time
- If it changes $K$ times $\Rightarrow$ in the worst case $O(Kn)$ time needed
Lemma 2: Let $P$ a set of $n$ points in the plane. One can compute the closest pair of them in expected linear time.

Proof:

- Randomly permute the points of $P$ into $P_n = \langle p_1, \ldots, p_n \rangle$
- Let $r_2 = \|p_1p_2\|$ and start inserting points to the data structure based on Lemma 1
- If at the $i$th iteration $r_i = r_{i-1} \Rightarrow$ addition of $p_i$ takes constant time
- If $r_i < r_{i-1}$ then rebuild the grid, and reinsert the $i$ points in $O(i)$ time
- Let $X_i$ a random indicator variable:

\[
X_i = \begin{cases} 
1, & r_i < r_{i-1} \\
0, & r_i = r_{i-1}
\end{cases}
\]
Proof of Lemma 2

- Let $T$ the running time of the method. Clearly

$$X = 1 + \sum_{i=2}^{n} (1 + X_i \cdot i)$$

- By linearity of expectation it is:

$$E(X) = E \left[ 1 + \sum_{i=2}^{n} (1 + X_i \cdot i) \right] = 1 + \sum_{i=2}^{n} E (1 + X_i \cdot i) =$$

$$1 + \sum_{i=2}^{n} 1 + \sum_{i=2}^{n} E (X_i \cdot i)$$

But

$$E (X_i \cdot i) = 0 \cdot Pr\{X_i = 0\} + i \cdot Pr\{X_i = 1\} = i \cdot Pr\{X_i = 1\}$$

Thus

$$E(X) = 1 + n - 1 + \sum_{i=2}^{n} i \cdot Pr\{X_i = 1\} = n + \sum_{i=2}^{n} i \cdot Pr\{X_i = 1\}$$
Bounding the probability of a change ($Pr\{X_i = 1\}$)

We will bound $Pr\{X_i = 1\} = Pr\{r_i < r_{i-1}\}$

- Fix the points of $P_i = \{p_1, p_2, \ldots, p_i\}$
- Randomly permute these points

**Definition:** A point $q \in P_i$ is called **critical** (at phase $i$) if $CP(P_i\{q\}) > CP(P_i)$ i.e. if its “consideration” leads to a “change” (e.g. smaller inter point closest distance)

**Note:**

1. Whether a node, at a given phase, is critical or not, only depends on geometry (not on the order that the algorithm examines the points). The order only affects the probability of a change or not.
2. The notion of criticality refers to a given phase of algorithm evolution.
3. There are 3 cases: 0, 1 or 2 critical points.
Bounding the probability of a change ($Pr\{X_i = 1\}$)

Case 1: no critical points

If there are no critical points $\Rightarrow r_i = r_{i-1} \Rightarrow$ no change

$\Rightarrow Pr\{X_i = 1\} = 0$

(When more than one, vertex disjoint closest distance edges exist.)
Bounding the probability of a change \( (Pr\{X_i = 1\}) \)

Case 2: 1 critical point

If there is \textit{one} critical point \( \Rightarrow Pr\{X_i = 1\} = \frac{1}{i} \)

(this is the probability that \( p_i \) is last in permutation)

(When a single set of more than one, adjacent closest pair edges exists.)
Bounding the probability of a change ($Pr\{X_i = 1\}$)

Case 3: 2 critical points

If there are *two* critical points, let them $p, q$ and notice that this is the unique points pair realizing $CP(P_i)$. But then $r_i < r_{i-1}$ iff either $p$ or $q$ are the last point ($p_i$) in the permutation, an event with probability $\frac{2}{i}$ (When a single closest distance edge exists.)
Bounding the change of probability
Case >2 critical points

Note that there cannot be more than two critical points.

Proof:
Indeed, let $p$ and $q$ be critical (and realize $CP(P)$). Let now a third critical point $r$. Then it must be $CP(P_i \setminus r) > CP(P_i)$.

But, $CP(P_i \setminus r) = ||pq||$ (since if we exclude $r$ then the closest distance is the one of the $p, q$ critical points). But $||pq|| = CP(P_i) \Rightarrow CP(P_i) > CP(P_i)$, a contradiction. □
Concluding the expected time analysis

Thus, \[
E[T] = n + \sum_{i=2}^{n} i \Pr\{X_i = 1\} \leq n + \sum_{i=2}^{n} \frac{2}{i} = \]

\[
= n + \sum_{i=2}^{n} 2 = n + 2n - 2 < 3n
\]

Overall, the expected running time is $O(n)$ i.e., linear.