Orthogonal Range Queries

Piotr Indyk
Range Searching in 2D

• Given a set of \( n \) points, build a data structure that for any query rectangle \( R \), reports all points in \( R \)
Kd-trees [Bentley]

- Not the most efficient solution in theory
- Everyone uses it in practice
- Algorithm:
  - Choose x or y coordinate (alternate)
  - Choose the median of the coordinate; this defines a horizontal or vertical line
  - Recurse on both sides
- We get a binary tree:
  - Size: $O(N)$
  - Depth: $O(\log N)$
  - Construction time: $O(N \log N)$
Kd-tree: Example

Each tree node $v$ corresponds to a region $\text{Reg}(v)$. 
Kd-tree: Range Queries

1. Recursive procedure, starting from $v=\text{root}$

2. Search $(v,R)$:
   a) If $v$ is a leaf, then report the point stored in $v$ if it lies in $R$
   b) Otherwise, if $\text{Reg}(v)$ is contained in $R$, report all points in the subtree of $v$
   c) Otherwise:
      • If $\text{Reg}(\text{left}(v))$ intersects $R$, then Search($\text{left}(v),R$)
      • If $\text{Reg}(\text{right}(v))$ intersects $R$, then Search($\text{right}(v),R$)
Query demo
Query Time Analysis

- We will show that Search takes at most $O(n^{1/2} + P)$ time, where $P$ is the number of reported points
  - The total time needed to report all points in all sub-trees (i.e., taken by step b) is $O(P)$
  - We just need to bound the number of nodes $v$ such that $\text{Reg}(v)$ intersects $R$ but is not contained in $R$. In other words, the boundary of $R$ intersects the boundary of $\text{Reg}(v)$
  - Will make a gross overestimation: will bound the number of $\text{Reg}(v)$ which are crossed by any of the 4 horizontal/vertical lines
Query Time Continued

• What is the max number $Q(n)$ of regions in an $n$-point kd-tree intersecting (say, vertical) line?
  – If we split on $x$, $Q(n)=1+Q(n/2)$
  – If we split on $y$, $Q(n)=2*Q(n/2)+2$
  – Since we alternate, we can write $Q(n)=3+2Q(n/4)$

• This solves to $O(n^{1/2})$
Analysis demo
A Faster Solution

• Query time: $O(\log^2 n + P)$
• Space: $O(n \log n)$
Idea I: Ranks

• Sort x and y coordinates of input points

• For a rectangle $R=\left[x_1, x_2\right] \times \left[y_1, y_2\right]$, we have point $(u, v) \in R$ iff
  
  $\begin{align*}
  &\text{succ}_x(x_1) \leq \text{rank}_x(u) \leq \text{pred}_x(x_2) \\
  &\text{succ}_y(y_1) \leq \text{rank}_y(v) \leq \text{pred}_y(y_2)
  \end{align*}$

• Thus we can replace
  
  $\begin{align*}
  &\text{Point coordinates by their rank} \\
  &\text{Query boundaries by succ/pred; this adds } O(\log n) \text{ to the query time}
  \end{align*}$
Dyadic intervals

• Assume \( n \) is a power of 2. Dyadic intervals are:
  – \([1,1]\), \([2,2]\) ... \([n,n]\)
  – \([1,2]\), \([3,4]\) ... \([n-1,n]\)
  – \([1,4]\), \([5,8]\) ... \([n-3,n]\)
  – ....
  – \([1...n]\)

• Any interval \( \{a...b\} \) can be decomposed into \( O(\log n) \) dyadic intervals:
  – Imagine a full binary tree over \( \{1...n\} \)
  – Each node corresponds to a dyadic interval
  – Any interval \( \{a...b\} \) can be “covered” using \( O(\log n) \) sub-trees
Range Trees

• For each level $l=1 \ldots \log n$, partition x-ranks using level-$l$ dyadic intervals
• This induces vertical strips
• Within each strip, construct a BST on y-coordinates
Range Trees
Analysis

• Each point occurs in $\log n$ different levels
• Space: $O(n \log n)$
• How do we implement the query?
Query procedure

• Consider query $R = X \times Y$
• Partition $X$ into dyadic intervals
• For each interval, query the corresponding strip BST using $Y$
Query procedure
Query procedure
Analysis ctd.

• Query time:
  – $O(\log n + \text{output})$ time per strip
  – $O(\log n)$ strips
  – Total: $O(\log^2 n + P)$

• Faster than kd-tree, but space $O(n \log n)$

• Recursive application of the idea gives
  – $O(\log^d n)$ query time
  – $O(n \log^{d-1} n)$ space

for the $d$-dimensional problem
Approximate Nearest Neighbor (ANN)

- **Given:** a set of points $P$ in the plane
- **Goal:** given a query point $q$, and $\epsilon > 0$, find a point $p'$ whose distance to $q$ is at most $(1+\epsilon)$ times the distance from $q$ to its nearest neighbor
Our “solution”

• We will “solve” the problem using kd-trees…
• …under the assumption that all leaf cells of the
  kd-tree for $P$ have bounded aspect ratio
• Assumption somewhat strict, but satisfied in
  practice for most of the leaf cells
• We will show
  – $O(\log n/\varepsilon^2)$ query time
  – $O(n)$ space (inherited from kd-tree)
ANN Query Procedure

- Locate the leaf cell containing \( q \)
- Enumerate all leaf cells \( C \) in the increasing order of distance from \( q \) (denote it by \( r \))
  - Update \( p' \) so that it is the closest point seen so far
  - Note: \( r \) increases, \( \text{dist}(q,p') \) decreases
- Stop if \( \text{dist}(q,p') < (1 + \epsilon) \times r \)
Analysis

• Running time:
  – All cells $C$ seen so far (except maybe for the last one) have diameter $> \epsilon * r$
  – …Because if not, then $p(C)$ would have been a $(1 + \epsilon)$-approximate nearest neighbor, and we would have stopped
  – The number of cells with diameter $\epsilon * r$, bounded aspect ratio, and touching a ball of radius $r$ is at most $O(1/\epsilon^2)$