Imprecise Point Location

Maarten Löffler  Joe Simons  Darren Strash
Estimated location

Imprecision
Why Imprecision?

Data source is imprecise
Why Imprecision?

Data is an approximate region
Why Imprecision?

Data is “stale”
Data points may move
Infrequent or unreliable information
Over time, less confident on exact location
If we get more / better info,
then more confident on location
Problem Statement

Data:

Maintain $n$ imprecise points in $\mathbb{R}^d$

Represented by bounding regions
Problem Statement

Query: (Point Enclosure)

Given query point: •

Which (if any) region contains it?
Problem Statement

Query:  (Imprecise Equality)

is there a point that is equal to the query point?
Problem Statement

Query:  (Imprecise Equality)

is there a point that is equal to the query point?

Possibly:
Problem Statement

Query: (Imprecise Equality)

is there a point that is equal to the query point?

No:
Problem Statement

Query: (Imprecise Equality)

is there a point that is equal to the query point?

Yes:

Not valid answer

Data points are imprecise
Problem Statement

Insert:

Add new region of any size:
Problem Statement
Delete:
Remove a region:
Problem Statement

Update:

Grow or Shrink a region:
Problem Statement

Update:

Move a region
Problem Statement

Update:

Move a region
Assumptions

Regions grow or shrink by a constant factor

Good

Bad
Assumptions

Region movement limited by size

Good

Bad
Assumptions

Region movement limited by size

But not other regions
Assumptions

Regions don’t overlap

Good

Bad

Ugly
Goal: (for constant dimension)

Query: $\Theta(\log n)$

Insert / Delete: $\Theta(\log n)$

Update: $o(\log n)$

$\Theta(\log \log n)$?

$\Theta(1)$?
Results - 1d

Query: $O(\log n)$

Insert / Delete: $O(\log n)$

Update: $O(1)$
1d Data

Regions are intervals
1d Data Structure

Space

Decomposition Tree

Intervals

Decomposition Tree

Data

$O(1)$ Updates

$O(\log n)$ Queries
1d Data Structure

Space

Decomposition Tree

1d version of compressed quadtree

Stores center points of intervals
$1d$ Data Structure

$1d$ version of compressed quadtree

Stores center points of intervals
1d Data Structure

Each leaf cell is proportional to interval
1d Data Structure

BST over leaves of quadtree

$O(1)$ finger updates
A Balanced Search Tree with $O(1)$ Worst-case Update Time

[Levcopoulos and Overmars 1988]

“Bags” of leaves
Bag size: $O(\log^2 n)$

$O(\log n)$ lists $\times O(\log n)$ leaves each
1d Query

Find bag:

Binary Search: $O(\log n)$
1d Query

Find list:

Iterative Search: $O(\log n)$
1d Query

Find leaf cell:

Iterative Search: $O(\log n)$
1d Query

Find leaf cell in quadtree
(pointers between trees)
1d Query

Find nearby intervals

$O(1)$ time
1d Query

Check which interval contains query point

$O(1)$ time
1d Query

Total time:

$O(\log n)$
1d Update

\[ O(1) \text{ time} \]
1d Update

Shrink Interval

$O(1)$ time
1d Update

\textbf{Shrink Interval}

\(O(1)\) time
1d Update

Grow Interval

$O(1)$ time
1d Update

Grow Interval

$O(1)$ time
1d Update

Move Interval

$O(1)$ time
1d Update

Move Interval

$O(1)$ time
1d Update

Move Interval

$O(1)$ time
1d Insert / Delete

Query to find leaf in both trees

Local updates in both trees

$O(\log n)$ time
2d Data Structure

Similar Strategy
2d Data Structure

Space

Decomposition Tree

Rectangles

Data

Decomposition Tree

Compressed Quadtree

Similar Strategy

Search tree on Quadtree cells
2d Data Structure

Compressed Quadtree

image from The Skip Quadtree by Eppstein, Goodrich, and Sun
2d Data Structure

Project quadtree to a path

Space Filling Curves

Euler Tour

Decomposition Tree

Data

Search tree on Quadtree cells
2d Data Structure

Update:
- Make local updates

Insert / Delete:
- Find quadtree leaf
- Make local updates

Query:
- Find quadtree leaf
- ?

Similar Strategy
1d Query

Find nearby intervals

$O(1)$ time
1d Query

Max # of intervals overlapping leaf cell: 3
1d Query

Max # of intervals overlapping leaf cell: 3

Adding interval splits cell
Max # of rectangles overlapping leaf cell:
2d Query

Max # of rectangles overlapping leaf cell: $O(1)$

Add corner points to small rectangles
Max # of rectangles overlapping leaf cell: $O(1)$
1d Query

Distance to center of interval: $O(1)$
2d Query

Distance to center of region: $O(n)$
2d Query

Distance to overlapping region: $O(n)$

How to find very large rectangles?

work in progress...
2d Ideas

At some level of the quadtree

Distance to center point is $O(1)$
$2d$ Ideas

Size of quadtree cell $> \frac{1}{4} \cdot$ size of region
2d Ideas

From leaf cell:
First check cell itself

Does cell contain region points?
Yes

Does region contain query point?
No
2d Ideas

From leaf cell:

Follow pointers up, or over
2d Ideas

From leaf cell:
Follow pointers up, or over
2d Ideas

From leaf cell:
Follow pointers up, or over

Right
No point in neighbor, move up
2d Ideas

From leaf cell:
Follow pointers up, or over

Right

No point in neighbor \textit{at this level} move up
2d Ideas

From leaf cell:
Follow pointers up, or over

Right
We’ve moved past...
2d Ideas

From leaf cell:
Follow pointers up, or over
2d Ideas

From leaf cell:
Follow pointers up, or over

Top
Point in neighbor, move over
2d Ideas

From leaf cell:
Follow pointers up, or over

Top
Check if region contains query point

No
2d Ideas

From leaf cell:
Follow pointers up, or over

Top-Right
2d Ideas

From leaf cell:
Follow pointers up, or over

Top-Right
No point in neighbor, move up
2d Ideas

From leaf cell:
Follow pointers up, or over

Top-Right
Point in neighbor, move over
2d Ideas

From leaf cell:
Follow pointers up, or over

Top-Right
Check if region contains query point

Yes
2d Ideas

From leaf cell:

Directions to consider:
2d Ideas

Still $O(n)$ distance

But can we reduce search in quadtree to marked successor queries?