Logarithmic-Time Point Location in General Two-Dimensional Subdivisions

Michal Kleinbort

Tel Aviv University, Dec 2015

Joint work with Michael Hemmer and Dan Halperin

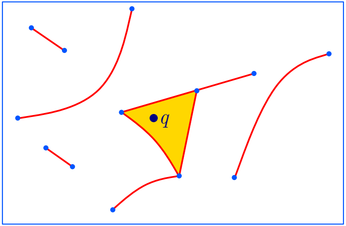
Planar Point Location - Definition

- Let S be a planar subdivision consisting of faces, edges, and vertices

The Planar Point Location Problem

Input: Query point q

Output: The feature of S containing q



n - the number of subdivision edges

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Two Variants of the Trapezoidal Map RIC Point Location

- **Basic algorithm** [Mulmuley '90, Seidel '91]
  - Expected $O(\log n)$ query time
  - Expected $O(n)$ size
  - Expected $O(n \log n)$ preprocessing time
- **Guaranteed variant** [de Berg et al. '00]
  - Guaranteed $O(\log n)$ query time
  - Guaranteed $O(n)$ size
  - Expected $O(n \log^2 n)$ preprocessing time (?)

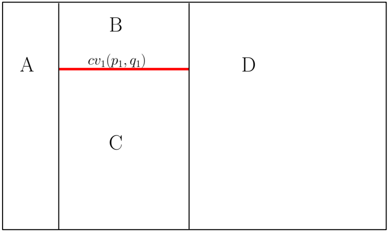
The Basic RIC Point Location Algorithm

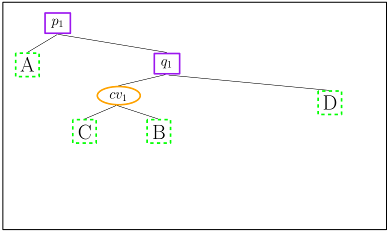
Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

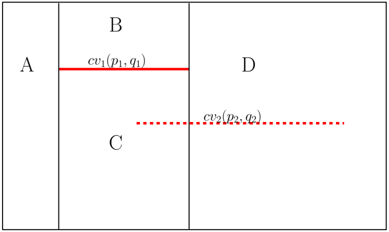


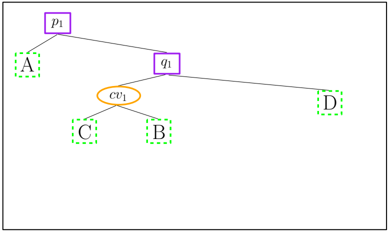


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

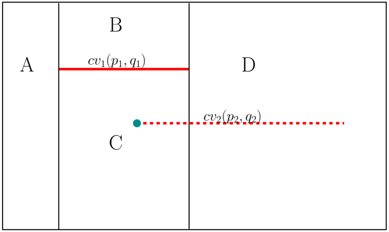


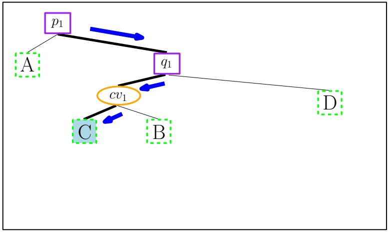


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

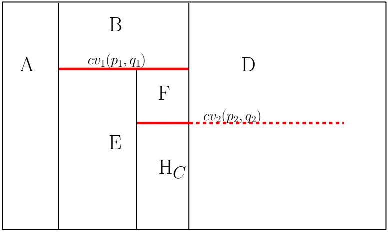


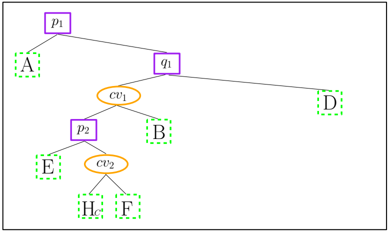


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

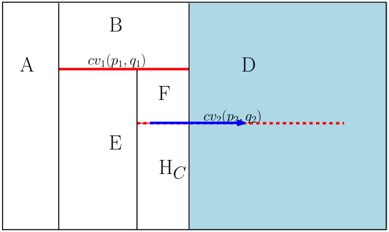


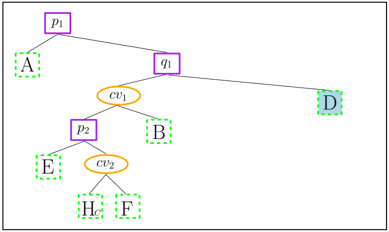


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

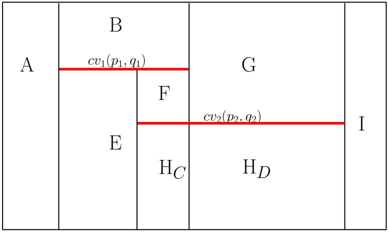


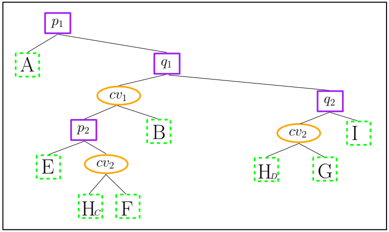


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)

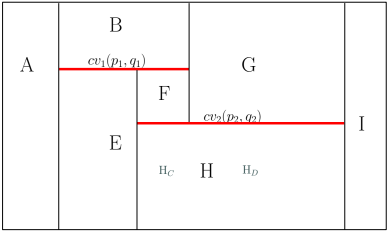


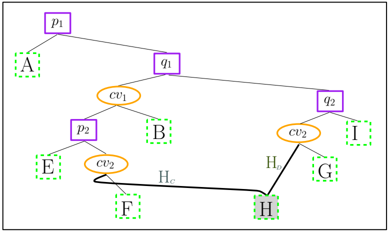


[Mulmuley '90, Seidel'91]

The Basic RIC Point Location Algorithm

Description: Builds the trapezoidal-map using a randomized incremental construction and maintains an auxiliary search-structure (DAG)





[Mulmuley '90, Seidel'91]

Basic Algorithm [Mulmuley '90, Seidel '91] - Complexity

- Expected $O(\log n)$ query time
- Expected $O(n)$ size
- Expected $O(n \log n)$ preprocessing time

Guaranteed $O(\log n)$ Query Time and $O(n)$ Size

- $\mathcal{S}$ - the size of the DAG
- $\mathcal{L}$ - the length of the longest query path

[de Berg et al.]

Guaranteed $O(\log n)$ Query Time and $O(n)$ Size

- $\mathcal{S}$ - the size of the DAG
- $\mathcal{L}$ - the length of the longest query path

The main idea:

- Construct the DAG using the basic algorithm with some random insertion order
  - Verify $\mathcal{S}$ on the fly ($\mathcal{S}$ can be accessed in $O(1)$ time)
  - Abort and rebuild if $\mathcal{S} \geq c_1 n$
- Verify that $\mathcal{L} \leq c_2 \log n$, rebuild otherwise

[de Berg et al.]

Guaranteed $O(\log n)$ Query Time and $O(n)$ Size

- $\mathcal{S}$ - the size of the DAG
- $\mathcal{L}$ - the length of the longest query path

The main idea:

- Construct the DAG using the basic algorithm with some random insertion order
  - Verify $\mathcal{S}$ on the fly ($\mathcal{S}$ can be accessed in $O(1)$ time)
  - Abort and rebuild if $\mathcal{S} \geq c_1 n$
- Verify that $\mathcal{L} \leq c_2 \log n$, rebuild otherwise
- Only a constant number of rebuilds is expected
  - The probability that $\mathcal{L}$ is bad is very small
  - The probability that $\mathcal{S}$ is bad is very small

[de Berg et al.]

Guaranteed $O(\log n)$ Query Time and $O(n)$ Size

- $f(n)$ - Time to verify that $\mathcal{L}$ is logarithmic on a DAG of n curves
- Overall expected time for construction: $O(n \log n + f(n))$
- It is unclear how to efficiently verify $\mathcal{L}$:
  - Claim that the expected verification time is $O(n \log^2 n)$
  - No concrete proof is given

[de Berg et al.]

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Can We Efficiently Maintain $\mathcal{L}$ On-the-fly?

- The best known solution requires $\Omega(n \log n)$ size

Can We Efficiently Maintain $\mathcal{L}$ On-the-fly?

- The best known solution requires $\Omega(n \log n)$ size

Idea: Maintain the depth $\mathcal{D}$ of the DAG instead (easy to maintain)

Can We Efficiently Maintain $\mathcal{L}$ On-the-fly?

- The best known solution requires $\Omega(n \log n)$ size

Idea: Maintain the depth $\mathcal{D}$ of the DAG instead (easy to maintain)

- $\mathcal{D}$ represents the length of the longest DAG path

The Modified Algorithm Using $\mathcal{D}$

The modified algorithm:

- Observe $\mathcal{S}$ and $\mathcal{D}$ during construction
- Abort and rebuild structure if one of the following occurs:
  - $\mathcal{S} \geq c_1 n$
  - $\mathcal{D} \geq c_2 \log n$

  for suitable constants $c_1, \ c_2 > 0$

The Modified Algorithm Using $\mathcal{D}$

The modified algorithm:

- Observe $\mathcal{S}$ and $\mathcal{D}$ during construction
- Abort and rebuild structure if one of the following occurs:
  - $\mathcal{S} \geq c_1 n$
  - $\mathcal{D} \geq c_2 \log n$

  for suitable constants $c_1,\ c_2 > 0$

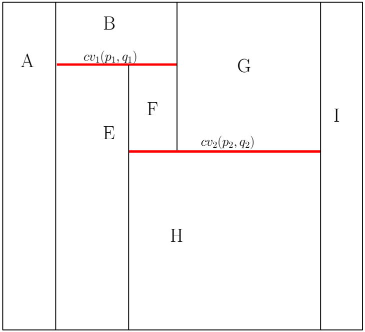
- $\mathcal{D}$ is not $\mathcal{L}$
  - Can we still expect a constant number of rebuilds?

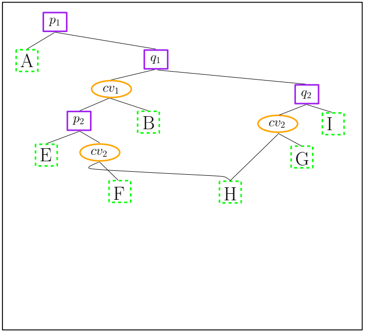
The Difference between $\mathcal{D}$ and $\mathcal{L}$

- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

The Difference between $\mathcal{D}$ and $\mathcal{L}$

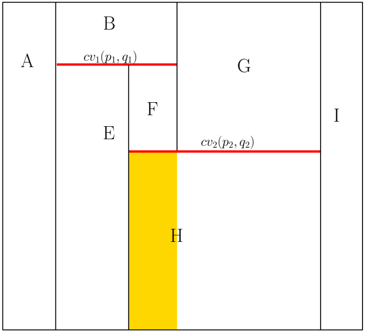
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

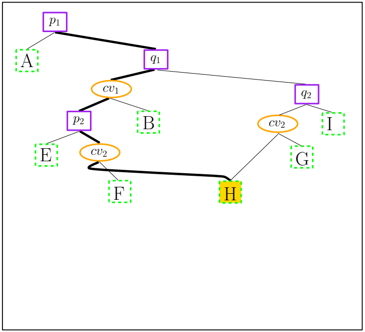




The Difference between $\mathcal{D}$ and $\mathcal{L}$

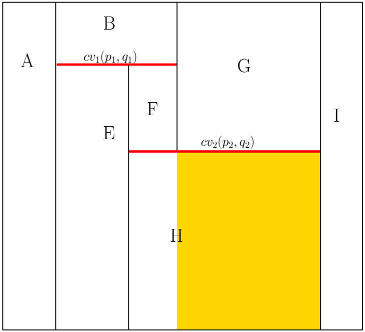
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

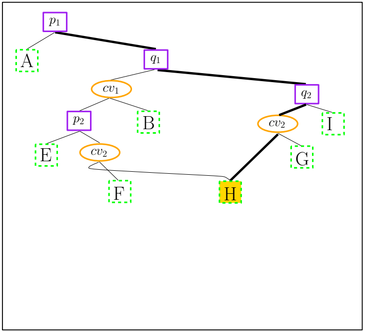




The Difference between $\mathcal{D}$ and $\mathcal{L}$

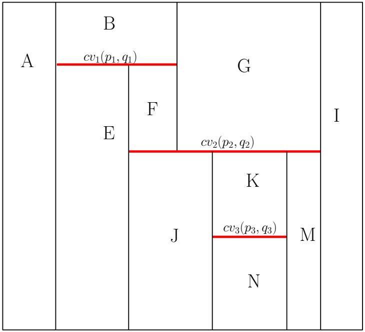
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

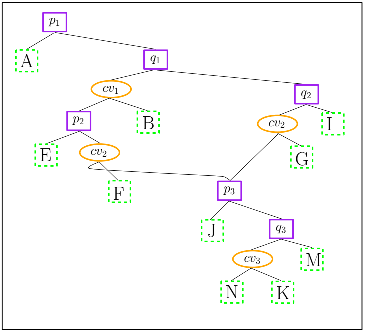




The Difference between $\mathcal{D}$ and $\mathcal{L}$

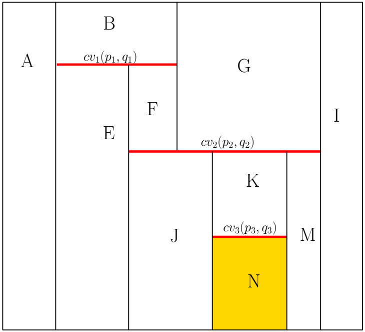
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

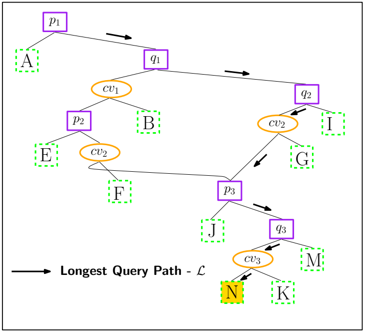




The Difference between $\mathcal{D}$ and $\mathcal{L}$

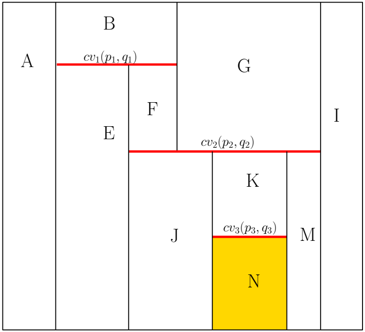
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

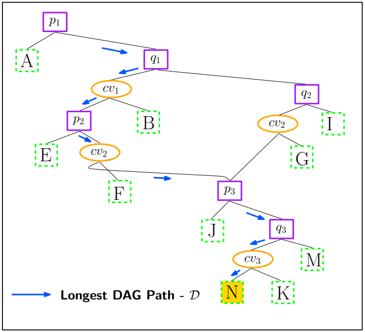




The Difference between $\mathcal{D}$ and $\mathcal{L}$

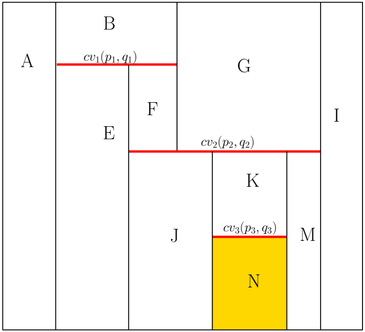
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$

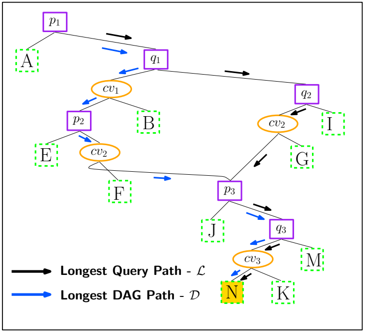




The Difference between $\mathcal{D}$ and $\mathcal{L}$

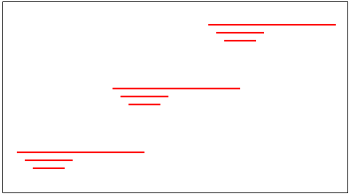
- Reminder: $\mathcal{D}$ represents the length of the longest DAG path
- Some DAG paths are not search paths
  - $\mathcal{D}$ is an upper bound on $\mathcal{L}$
  - $\mathcal{D}$ may be significantly larger than $\mathcal{L}$



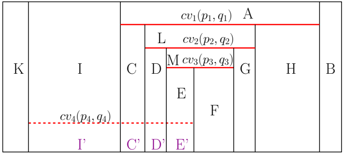


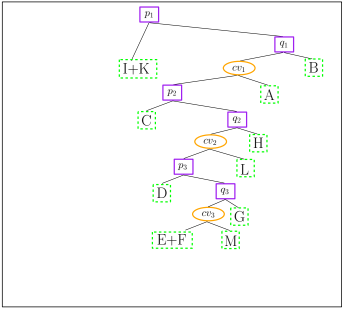
Towards a Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio

- Top-to-bottom insertion order
- $\sqrt{n}$ blocks
- $\sqrt{n}$ segments in each block
- $\mathcal{D}$ is $\Omega(n)$
- $\mathcal{L}$ is $O(\sqrt{n})$



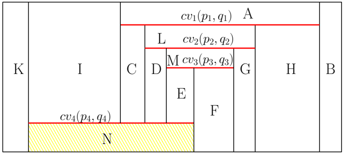
Towards a Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio

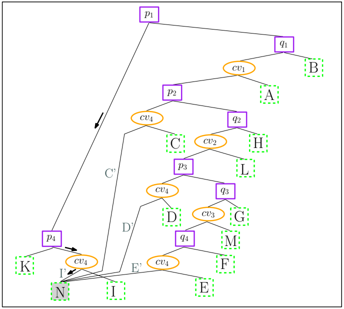




- $\mathcal{D}$ is $\Omega(n)$
- $\mathcal{L}$ is $O(\sqrt{n})$

Towards a Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio





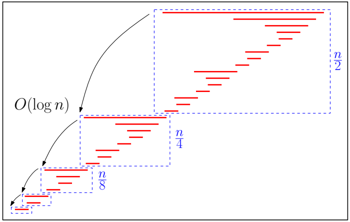
- $\mathcal{D}$ is $\Omega(n)$
- $\mathcal{L}$ is $O(\sqrt{n})$

Towards a Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio

- This construction ensures that each newly inserted segment intersects the trapezoid with the largest depth
- A query can skip an entire block using only one comparison
- Within the relevant block there are at most $O(\sqrt{n})$ comparisons

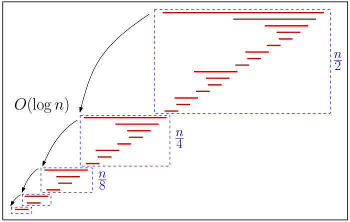
Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio

- Top-to-bottom insertion order
- $\mathcal{D}$ is $\Omega(n)$
- $\mathcal{L}$ is $O(\log n)$
  - Achieved due to the recursive structure



Worst-Case $\mathcal{D}/\mathcal{L}$ Ratio

- Top-to-bottom insertion order
- $\mathcal{D}$ is $\Omega(n)$
- $\mathcal{L}$ is $O(\log n)$
  - Achieved due to the recursive structure



Theorem 1

The worst-case ratio between $\mathcal{D}$ and $\mathcal{L}$ is $\Omega(n/\log n)$ and this bound is tight.

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

Verifying $\mathcal{L}$ After Construction in $O(n \log n)$ time

By verifying $\mathcal{L}$ in $O(n \log n)$ time we get the following expected $O(n \log n)$ time construction algorithm:

- Construct the DAG with some random insertion order
  - Verify $\mathcal{S}$ on the fly (can be accessed in $O(1)$ time)
  - Abort and rebuild if $\mathcal{S} \geq c_1 n$
- Verify in $O(n \log n)$ time that $\mathcal{L} \leq c_2 \log n$, rebuild otherwise
- Only a constant number of rebuilds is expected

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Ingredients:

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Ingredients:

- Observation: The length of a path in the DAG for a query point q is at most 3 times the number of all trapezoids that covered q throughout the algorithm [Har-Peled]

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Ingredients:

- Observation: The length of a path in the DAG for a query point q is at most 3 times the number of all trapezoids that covered q throughout the algorithm [Har-Peled]
- A reduction from the collection of all trapezoids to a collection of axis-aligned rectangles
  - Uses a total order according to which curves can be translated one by one to $y = -\infty$ without hitting other curves that have not been moved yet [Guibas & Yao '80]
  - Can be computed in $O(n \log n)$ time [Ottmann & Widmayer '83]

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Ingredients:

- Observation: The length of a path in the DAG for a query point q is at most 3 times the number of all trapezoids that covered q throughout the algorithm [Har-Peled]
- A reduction from the collection of all trapezoids to a collection of axis-aligned rectangles
  - Uses a total order according to which curves can be translated one by one to $y = -\infty$ without hitting other curves that have not been moved yet [Guibas & Yao '80]
  - Can be computed in $O(n \log n)$ time [Ottmann & Widmayer '83]
- An $O(n \log n)$ time algorithm for computing the cover-depth of a collection of n axis-aligned rectangles [Alt & Scharf '10]

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

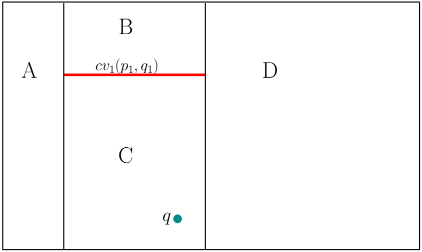
The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q

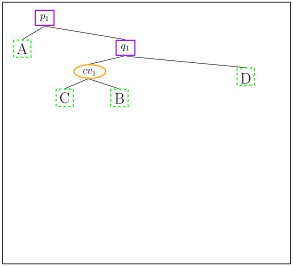
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



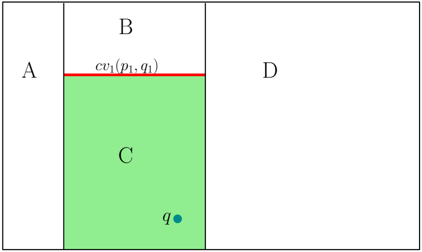


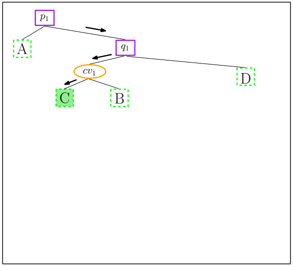
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



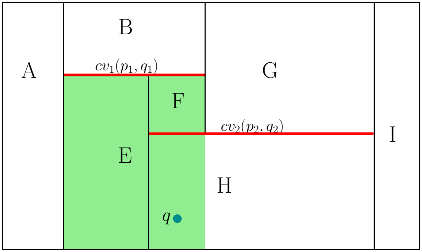


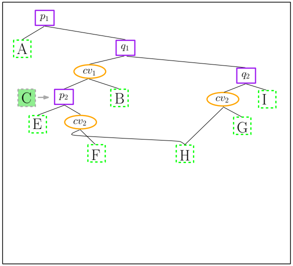
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



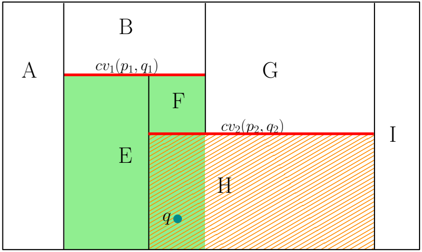


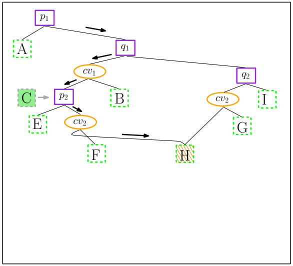
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



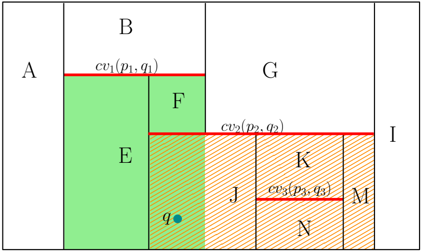


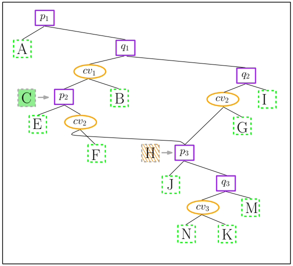
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



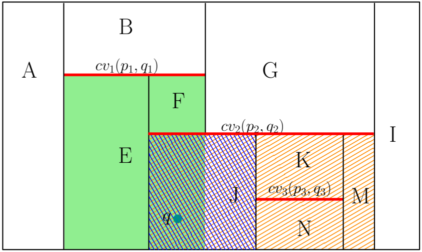


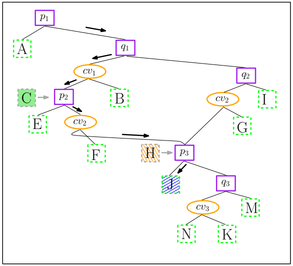
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

The key ingredient:

Observation (Har-Peled)

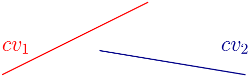
The length of a path in the DAG for a query point q is at most three times the number of trapezoids created throughout the algorithm that cover q



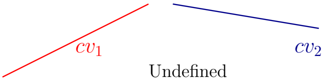


Reduction from Trapezoids to Rectangles

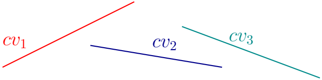
- C - a set of interior disjoint x-monotone curves
- Define a total order $<$ as follows [Guibas & Yao '80]:
  - $\prec$ - an acyclic relation on C
    $cv_i \prec cv_j \Leftrightarrow cv_i(x) < cv_j(x)$ for some $x \in x\text{-range}(cv_i) \bigcap x\text{-range}(cv_j)$



$cv_2 \prec cv_1$



  - Extend $\prec^+$ (the transitive closure of $\prec$) to a total order $<$:
    $cv_i < cv_j \Leftrightarrow (cv_i \prec^+ cv_j) or (\neg(cv_j \prec^+ cv_i) and (cv_i$ left $cv_j))$



$cv_2 < cv_1 < cv_3$

Reduction from Trapezoids to Rectangles

- $Rank : C \rightarrow \{1, ..., n\}$ - returns the order of $cv \in C$ when sorting C according to $<$

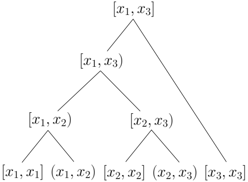
A trapezoid t is reduced to a rectangle r, s.t.:

- t and r have the same x-range
- top and bottom edges of r lie on $y = Rank(top(t))$ and $y = Rank(bottom(t))$, respectively

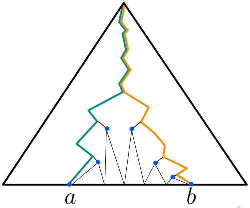
We show that this reduction preserves the cover depth

Computing the Cover-Depth of a Collection of n Axis-Aligned Rectangles in $O(n \log n)$ Time

- Algorithm by Alt & Scharf (2010)
- Basic data structure:
  a balanced binary tree for the intervals



- Keep *coverage* and *max-coverage* in every node
- Sweep from $y = +\infty$ to $y = -\infty$
- Sweep-line event: rectangle starts or ends
- Update a rectangle event with x-interval (a, b) in $\sim 2 \log n$ time



An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Lemma

The length $\mathcal{L}$ in a linear size DAG can be verified in $O(n \log n)$ time.

An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Lemma

The length $\mathcal{L}$ in a linear size DAG can be verified in $O(n \log n)$ time.

Theorem 2

A point location data structure for a planar subdivision with n edges, which has $O(n)$ size and $O(\log n)$ query time in the worst case, can be built in expected $O(n \log n)$ time.

A Simpler Verification Algorithm for $\mathcal{L}$

We also suggest a randomized verification algorithm which:

- Runs in expected $O(n \log n)$ time
- Is much simpler
- Uses the existing structures (the DAG)

Outline

- Trapezoidal-map RIC point-location variants
- Depth vs. maximum query path length
- An efficient construction algorithm for static settings
- Open Problems

A Major Open Problem

- Suppose that the structure is rebuilt whenever either the depth $\mathcal{D}$ or the size $\mathcal{S}$ exceed some thresholds.
- Can we still expect a constant number of rebuilds?

The End