Logarithmic-Time Point Location in General Two-Dimensional Subdivisions

Michal Kleinbort

Tel Aviv University, May 2014

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)

![Diagram of the trapezoidal-map and search-structure](image)

[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)

\[ \text{Mulmuley '90, Seidel'91} \]
The Basic RIC Point Location Algorithm

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

![Diagram showing the trapezoidal-map and search-structure](image)

[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)

\[ \text{Mulmuley '90, Seidel'91} \]
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 [Mulumley ’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

- $S$ - 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

- $S$ - the size of the DAG
- $L$ - the length of the longest query path

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

[de Berg et al.]
Guaranteed $O(\log n)$ Query Time and $O(n)$ Size

- $S$ - the size of the DAG
- $L$ - the length of the longest query path

The main idea:
- Construct the DAG using the basic algorithm with some random insertion order
  - Verify $S$ on the fly ($S$ can be accessed in $O(1)$ time)
  - Abort and rebuild if $S \geq c_1 n$
- Verify that $L \leq c_2 \log n$, rebuild otherwise
- Only a constant number of rebuilds is expected
  - The probability that $L$ is bad is very small (Lemma 2)
  - The probability that $S$ is bad is very small (Lemma 3)

[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.]
The Probabilities for “Bad” \( \mathcal{L} \) or \( \mathcal{S} \) are Small

Lemma 1 (Prob. that a given search path is bad is small)

Given a set \( S \) of \( n \) non-crossing line segments, a query point \( q \), and a parameter \( \lambda > 0 \), the probability that the search path for \( q \) in the DAG has more than \( 3\lambda \log (n + 1) \) nodes is at most \( 1/(n + 1)^{\lambda \log 1.25 - 1} \).

Proof: using a tail estimate (Appendix)
The Probabilities for “Bad” $L$ or $S$ are Small

**Lemma 2 (Prob. that $L$ is bad is small)**

Given a set $S$ of $n$ non-crossing line segments, and a parameter $\lambda > 0$, the probability that the maximum length of a search path in the DAG is more than $3\lambda \log(n + 1)$ is at most $2/(n + 1)^{\lambda \log 1.25 - 3}$

Proof sketch:

- Extend vertical walls at each endpoint- defining at most $2(n + 1)^2$ regions
- Consider the search paths of representative points of these regions
- By Lemma 1 we get the required result
The Probabilities for “Bad” $\mathcal{L}$ or $S$ are Small

Lemma 3 (Prob. that $S$ is bad is small)

Given a set $S$ of $n$ non-crossing $x$-monotone curves, and a parameter $\rho \geq 1$, the probability that the size $S$ of the DAG is greater than $15\rho n$ is at most $1/\rho$

(Proof in the Appendix)
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 \( D \) 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 $D$ of the DAG instead (easy to maintain)

- $D$ represents the length of the longest DAG path
The Modified Algorithm Using $\mathcal{D}$

The modified algorithm:

- Observe $S$ and $\mathcal{D}$ during construction

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

for suitable constants $c_1, c_2 > 0$
The Modified Algorithm Using $\mathcal{D}$

The modified algorithm:

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

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 $D$ and $L$

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

\[ cv_1(p_1, q_1) \]
\[ cv_2(p_2, q_2) \]
\[ cv_3(p_3, q_3) \]
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 $D$ and $L$

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

![Diagram showing the difference between $D$ and $L$]

---

Michal Kleinbort (TAU)  
Point-Location in General 2D Subdivisions  
May, 2014  
13 / 33
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 $D/L$ 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

Theorem 1

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

By verifying $L$ 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 $S$ on the fly (can be accessed in $O(1)$ time)
  - Abort and rebuild if $S \geq c_1 n$
- Verify in $O(n \log n)$ time that $L \leq c_2 \log n$, rebuild otherwise
- Only a constant number of rebuilds is expected
An $O(n \log n)$ Verification Algorithm for $\mathcal{L}$

Ingredients:

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 $\mathcal{L}$

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 $q$. 

\[cv_{1}(p_{1}, q_{1})\]
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 $q$
Reducing $T^*$ to $R^*$

- $C$ - a set of interior disjoint $x$-monotone curves

Define a total order $<$ as follows [Guibas & Yao ’80]:

- $\prec$ - an acyclic relation on $C$

$$cv_i \prec cv_j \iff cv_i(x) < cv_j(x) \text{ for some } x \in x\text{-range}(cv_i) \cap x\text{-range}(cv_j)$$

- Extend $\prec^+$ (the transitive closure of $\prec$) to a total order $\lhd$:

$$cv_i \lhd cv_j \iff (cv_i \prec^+ cv_j) \lor (\neg (cv_j \prec^+ cv_i) \land (cv_i \text{ left } cv_j))$$
Reducing $T^*$ to $R^*$

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

A trapezoid $t \in T^*$ is reduced to a rectangle $r \in R^*$, 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
Showing that the Reduction Preserves the Depth

- Partition the plane into regions $\text{Regions}(\text{arr})$ by passing a vertical line through every endpoint of the arrangement $\text{arr}$.
- For any region $a_t \in \text{Regions}(\mathcal{A}(\mathcal{T}^*))$ the matching rectangular region is $a_r \in \text{Regions}(\mathcal{A}(\mathcal{R}^*))$.

It can be shown that:

1. $\text{Regions}(\mathcal{A}(\mathcal{R}^*))$ spans the plane
2. For every $t \in \mathcal{T}^*$ covering $a_t$ its reduced rectangle $r \in \mathcal{R}^*$ covers $a_r$
3. For every $r \in \mathcal{R}^*$ covering $a_r$ its original trapezoid $t \in \mathcal{T}^*$ covers $a_t$
Computing the Depth of $A(R^*)$ 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 4
The length $\mathcal{L}$ in a linear size DAG can be verified in $O(n \log n)$ time.
Lemma 4
The length $L$ 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 $L$

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 $D$ or the size $S$ exceed some thresholds.
- Can we still expect a constant number of rebuilds?
The End
Appendix
The Expected Query Time

- **Observation**: the depth of the DAG increases by at most 3 in every iteration

- Consider the path for a query \( q \)

**Lemma 5**: Given a DAG for \( i \) segments, the probability that the removal of a segment will destroy the trapezoid containing \( q \) is at most \( 4/i \)

(proof by figure)
The Expected Query Time

Bounding the expected length of the query path to $q$ using backwards analysis:

- $X_i$ - the number of nodes added to the path to $q$ in iteration $i$
- $P_i$ - the probability that there's a node on the path to $q$ that is created in iteration $i$

\[
\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] \leq \sum_{i=1}^{n} 3P_i \leq \sum_{i=1}^{n} \frac{12}{i} = 12 \sum_{i=1}^{n} \frac{1}{i} = O(\log n)
\]
The Expected Size

expected \# DAG nodes = \# leaves + expected \# inner nodes

- Bounding \# leaves:
  - \# leaves = \# trapezoids in the trapezoidal map
  - \# trapezoids in the trapezoidal map is at most \(3n + 1 = O(n)\)
    - The left side of every trapezoid is defined by a vertex (the left trapezoid is bounded by the boundary)
    - Each curve defines the left of 2 trapezoids with its left endpoint and 1 trapezoid with its right endpoint
The Expected Size

- Bounding the expected number of inner nodes created in the $i$th iteration:
  - Is at most the expected number of new trapezoids created in the $i$th iteration ($N_i$)
  - $T(S_i)$ - trapezoidal map for the first $i$ inserted curves

$$N_i = \frac{1}{i} \sum_{c \in T(S_i)} (\# \text{ trapezoids in } T(S_i) \text{ that disappear by removing } c)$$

$$\leq \frac{1}{i} \cdot 4(\# \text{ trapezoids in } T(S_i)) = \frac{O(i)}{i} = O(1)$$

- Summing up over all $i$ → we get: expected $O(n)$ inner nodes in the final DAG

Corollary: expected number of nodes is $O(n)$
The Expected Preprocessing Time

- Expected time to insert the $i$th curve: $O(\log i)$
  - Expected time to locate the left endpoint of the $i$th curve: $O(\log i)$
  - Expected # trapezoids created by the $i$th insertion: $O(1)$
- Each insertion takes at most expected $O(\log n)$ time
  $\Rightarrow O(n \log n)$ expected preprocessing time
A Tail Estimate

We show: the probability that the maximum query time is bad is very small

**Lemma 1:** Given a set $S$ of $n$ non-crossing line segments, a query point $q$, and a parameter $\lambda > 0$, the probability that the search path for $q$ in the DAG has more than $3\lambda \log(n + 1)$ nodes is at most $1/(n + 1)^{\lambda \log 1.25 - 1}$

**Proof sketch:**

- Define a DAG with one source and one sink, the paths correspond to the permutations of $S$
  - A node for every subset of $S$, grouped in layers according to cardinality
  - A node in layer $i$ has $i$ incoming edges from nodes in layer $i - 1$ and $n - i$ outgoing edges to nodes in layer $i + 1$
  - An edge is marked if its insertion at that point changes the trapezoid containing $q$
  - Backwards analysis argument: at most 4 segments change the trapezoid containing $q$ when they are removed from the subset
  - Therefore, any node has at most 4 marked incoming arcs (we always mark exactly 4)
A Tail Estimate

Proof sketch - continued:

- Finding the expected \# marked edges on a path in the DAG
- \( X_i \) - (random variable) = 1 if the \( i \)-th arc on the path in the DAG is marked
- \# nodes in the path is at most 3\( Y \), where \( Y := \sum_{i=1}^{n} X_i \)
- Using Markov's inequality:
  \[
  \Pr[Y \geq \lambda \log(n + 1)] = \Pr[e^{tY} \geq e^{t\lambda \log(n+1)}] \leq e^{-t\lambda \log(n+1)} \mathbb{E}[e^{tY}]
  \]
- Since the random variables are independent:
  \[
  \mathbb{E}[e^{tY}] = \mathbb{E}\left[\sum_{i=1}^{n} tX_i\right] = \mathbb{E}\left[\prod_{i=1}^{n} e^{tX_i}\right] = \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}]
  \]
  \[
  \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}] \leq \frac{2}{1^2} \cdots \frac{n+1}{n} = n + 1, \text{ for } t = \log 1.25
  \]
  \[
  \Pr[Y \geq \lambda \log(n + 1)] \leq 1/(n + 1)^{\lambda t - 1}
  \]
A Tail Estimate

**Lemma 2:** Given a set $S$ of $n$ non-crossing line segments, and a parameter $\lambda > 0$, the probability that the maximum length of a search path in the DAG is more than $3\lambda \log(n + 1)$ is at most $\frac{2}{(n + 1)^{\lambda \log 1.25}}$.

**Proof sketch:**
- Extend vertical walls at each endpoint- defining at most $2(n + 1)^2$ regions.
- Consider the search paths of representative points of these regions.
- By Lemma 1 we get the required result.
Size

We show: the probability that the size is bad is very small

**Lemma 3:** Given a set $S$ of $n$ non-crossing $x$-monotone curves, and a parameter $\rho \geq 1$, the probability that the size $S$ of the DAG is greater than $15\rho n$ is at most $1/\rho$

Proof sketch:
- $C$ - random variable representing the number of DAG nodes
- $C = \#$ leaves + sum of inner nodes created in iterations $1, \ldots, n$
- $\#$ leaves $= |T(S)| \leq 3n + 1$
- $\#$ inner nodes created in iteration $i = \#$ new trapezoids created in iteration $i$ minus $1 = k_i - 1$
Size

\[ \mathbb{E}[C] = \mathbb{E}[|T(S)| + \sum_{i=1}^{n} (k_i - 1)] = \mathbb{E}[|T(S)|] + \mathbb{E}[\sum_{i=1}^{n} (k_i - 1)] \]
\[ \leq (3n + 1) + \mathbb{E}[\sum_{i=1}^{n} k_i] - n = 2n + 1 + \mathbb{E}[\sum_{i=1}^{n} k_i] = 2n + 1 + \sum_{i=1}^{n} \mathbb{E}[k_i] \]

\[ \mathbb{E}[k_i] \leq \frac{4 \cdot |T(S_i)|}{i} \leq \frac{4(3i+1)}{i} = 12 + \frac{4}{i} \]

Therefore, \( \mathbb{E}[C] \leq 14n + 1 + 4H_n \)
and \( \mathbb{E}[C] < 15n \), for \( n \geq 12 \)

Using Markov’s inequality: \( \Pr[C \geq 15\rho n] \leq \frac{\mathbb{E}[C]}{15\rho n} = \frac{15n}{15\rho n} = \frac{1}{\rho} \)