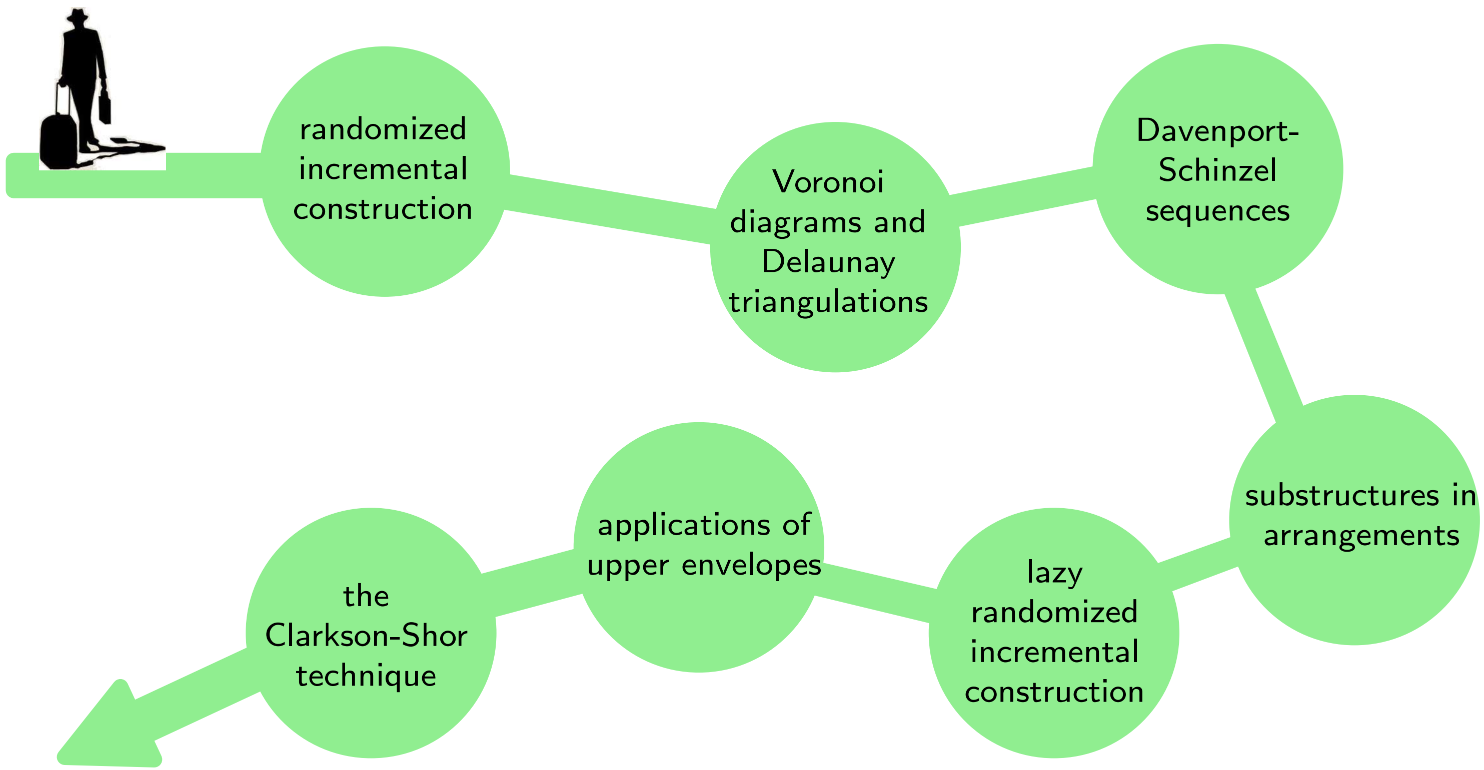
COMPUTATIONAL GEOMETRY

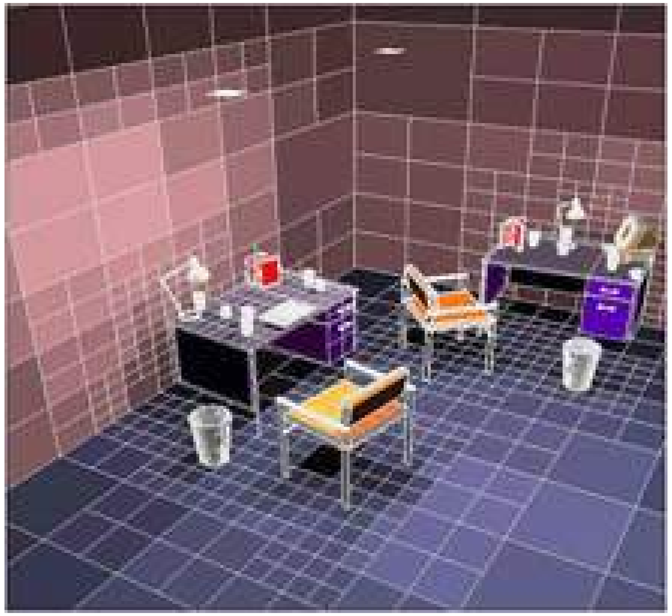
An Introduction Through Randomized Incremental Algorithms

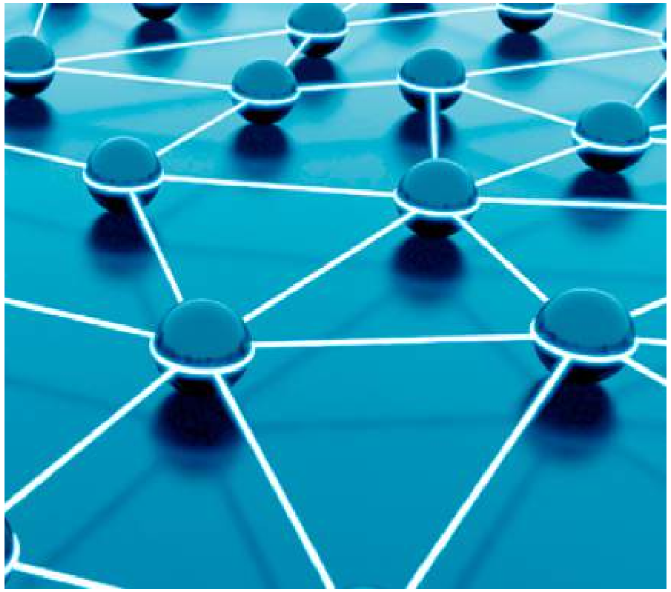


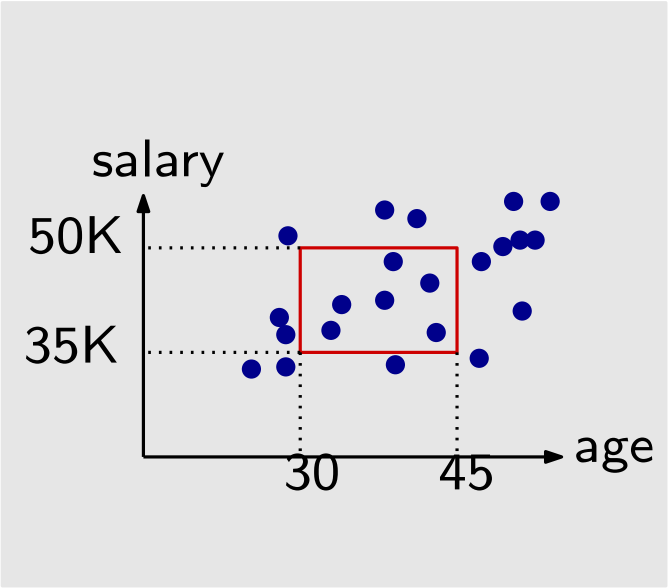
Mark de Berg (TU Eindhoven)

Geometry is everywhere . . .

- geographic information systems
- computer-aided design and manufacturing
- virtual reality
- robotics
- computational biology
- sensor networks
- databases
- and more . . .





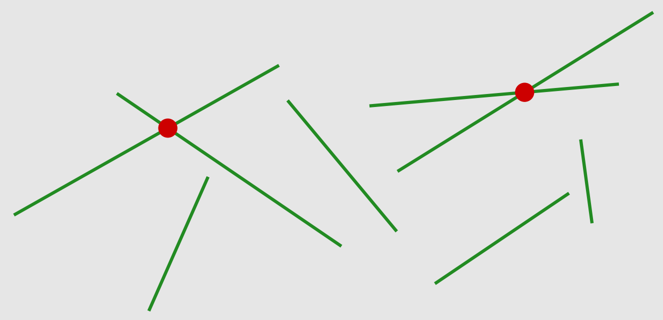


Computational Geometry

area within algorithms research dealing with spatial data

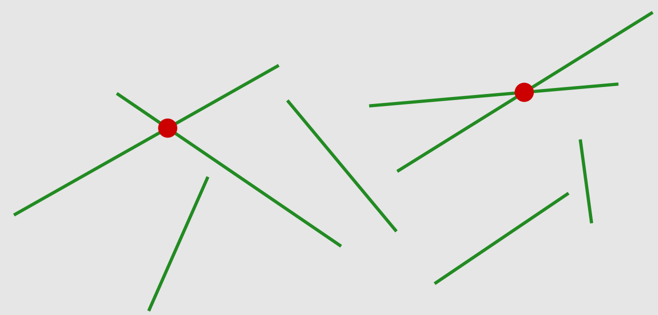
- aim for provably correct solutions (no heuristics)
- theoretical analysis of running time, memory usage: $O(\cdots)$

example problem: line-segment intersection



Compute all k intersections in a set S of n line segments.

example problem: line-segment intersection



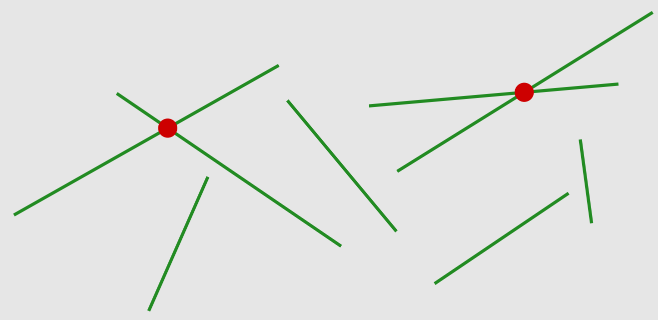
Compute all k intersections in a set S of n line segments.

1. **for** every pair of segments in S
2. **do** compute (possible) intersection

- running time $O(n^2)$
- can we do better if k is small?
  yes: $O(n \log n)$

Computational Geometry

example problem: line-segment intersection



Compute all k intersections in a set S of n line segments.

1. **for** every pair of segments in S
2. **do** compute (possible) intersection

- running time $O(n^2)$
- can we do better if k is small?
  yes: $O(n \log n)$

Computational geometry

- focus on scale-up behavior
- basic operations are assumed available (compute intersection of two lines, distance between two points, etc.)

Computational Geometry: Tools of the Trade

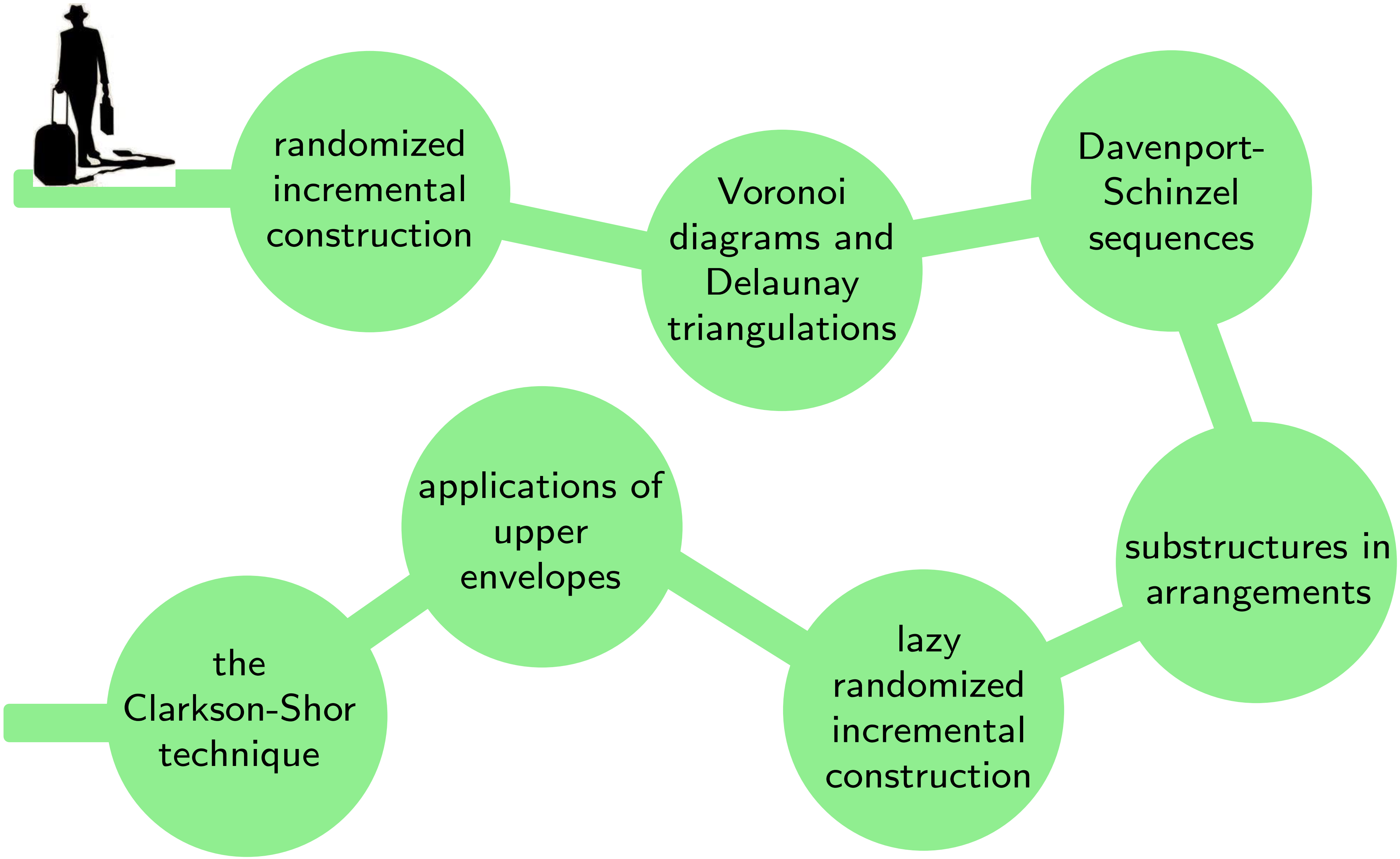
Algorithmic design techniques and tools

- plane sweep
- geometric divide-and-conquer
- randomized incremental construction
- parametric search
- (multi-level) geometric data structures

Geometric structures and concepts

- Voronoi diagrams and Delaunay triangulations
- arrangements
- cuttings, simplicial partitions, polynomial partitions
- coresets

Course Overview



Course Overview

randomized incremental construction

Voronoi diagrams and Delaunay triangulations

Davenport-Schinzel sequences

substructures in arrangements

lazy randomized incremental construction

applications of upper envelopes

the Clarkson-Shor technique

Warm-up Exercise

Warm-up Exercise

Analyze worst-case and the expected running time of the following algorithm

$\textsc{ParanoidMax}(A)$

$\triangleright$ computes maximum in an array $A[0..n-1]$

Randomly permutate the elements in the array A

$max \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

  if $A[i] > max$ **then**

    $max \leftarrow A[i]$

    to be on the safe side, check if $A[i]$ is indeed the largest element in $A[0..i]$

return max

Warm-up Exercise

Analyze worst-case and the expected running time of the following algorithm

$\textsc{ParanoidMax}(A)$

$\triangleright$ computes maximum in an array $A[0..n-1]$

Randomly permutate the elements in the array A
$max \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > max$ **then**
        $max \leftarrow A[i]$
        for $j \leftarrow 0$ **to** $i-1$ **do**
            if $A[j] > max$ **then** error
return max

Warm-up Exercise

Analyze worst-case and the expected running time of the following algorithm

ParanoidMax(A)

$\triangleright$ computes maximum in an array $A[0..n-1]$

Randomly permutate the elements in the array A

$max \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

  if $A[i] > max$ **then**

    $max \leftarrow A[i]$

    for $j \leftarrow 0$ **to** $i-1$ **do**

      if $A[j] > max$ **then** error

return max

- generates permutation uniformly at random
- assume this can be done in $O(n)$ time

Worst-case analysis

$$\begin{aligned}\text{running time} &= O(n) + \textstyle\sum_{i=1}^{n-1} (\text{worst-case time for } i\text{-th iteration}) \\ &= O(n) + \textstyle\sum_{i=1}^{n-1} O(i) \\ &= O(n^2)\end{aligned}$$

Analysis of expected running time

$$
\begin{aligned}
\mathrm{E}\left[\text{running time}\right] &= \mathrm{E}\left[O(n) + \textstyle\sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right] \\
&= O(n) + \textstyle\sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right]
\end{aligned}
$$

Analysis of expected running time

$$
\begin{aligned}
\mathrm{E}\left[\text{running time}\right] &= \mathrm{E}\left[O(n) + \textstyle\sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right] \\
&= O(n) + \textstyle\sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}\left[\text{time for } i\text{-th iteration}\right] &= \Pr\left[\text{max changes in } i\text{-th iteration}\right] \cdot O(i) \\
&\quad + \Pr\left[\text{max does not change}\right] \cdot O(1)
\end{aligned}
$$

Analysis of expected running time

$$\mathrm{E}\left[\text{running time}\right] = \mathrm{E}\left[O(n) + \textstyle\sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right]$$

$$= O(n) + \textstyle\sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right]$$

$$\mathrm{E}\left[\text{time for } i\text{-th iteration}\right] = \Pr\left[\text{max changes in } i\text{-th iteration}\right] \cdot O(i) + \Pr\left[\text{max does not change}\right] \cdot O(1)$$

backwards analysis

max changes when adding $A[i]$ to $\{A[0], \ldots, A[i-1]\}$ $\iff$ max changes when removing $A[i]$ from $\{A[0], \ldots, A[i]\}$

Analysis of expected running time

$$\mathrm{E}\left[\text{running time}\right] = \mathrm{E}\left[O(n) + \sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right]$$

$$= O(n) + \sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right]$$

$$\mathrm{E}\left[\text{time for } i\text{-th iteration}\right] = \overbrace{\boxed{\Pr\left[\text{max changes in } i\text{-th iteration}\right]}}^{\leqslant 1/i} \cdot O(i) + \Pr\left[\text{max does not change}\right] \cdot O(1)$$

backwards analysis

max changes when adding $A[i]$ to $\{A[0], \ldots, A[i-1]\}$ $\iff$ max changes when removing $A[i]$ from $\{A[0], \ldots, A[i]\}$

Analysis of expected running time

$$\begin{aligned}
\mathrm{E}\left[\text{running time}\right] &= \mathrm{E}\left[O(n) + \textstyle\sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right] \\
&= O(n) + \textstyle\sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right] \\
&= O(n)
\end{aligned}$$

$$\mathrm{E}\left[\text{time for } i\text{-th iteration}\right] = \overbrace{\mathrm{Pr}\left[\text{max changes in } i\text{-th iteration}\right]}^{\leqslant 1/i} \cdot O(i) + \mathrm{Pr}\left[\text{max does not change}\right] \cdot O(1)$$

backwards analysis

max changes when adding $A[i]$ to $\{A[0], \ldots, A[i-1]\}$ $\iff$ max changes when removing $A[i]$ from $\{A[0], \ldots, A[i]\}$

Analysis of expected running time

with respect to random choices of algorithm, no assumptions on input distribution

$$\begin{aligned}
\mathrm{E}\left[\text{running time}\right] &= \mathrm{E}\left[O(n) + \textstyle\sum_{i=1}^{n-1} \text{time for } i\text{-th iteration}\right] \\
&= O(n) + \textstyle\sum_{i=1}^{n-1} \mathrm{E}\left[\text{time for } i\text{-th iteration}\right] \\
&= O(n)
\end{aligned}$$

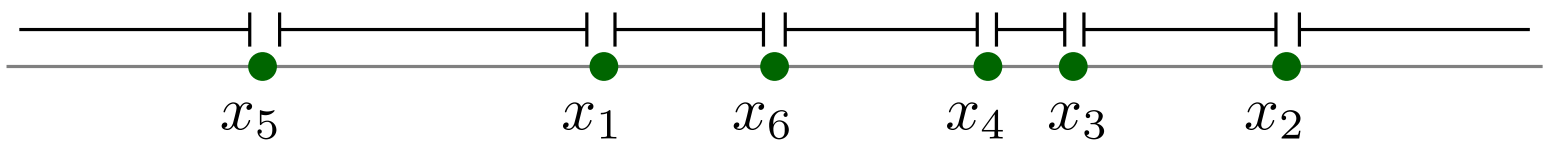
$$\mathrm{E}\left[\text{time for } i\text{-th iteration}\right] = \underbrace{\mathrm{Pr}\left[\text{max changes in } i\text{-th iteration}\right]}_{\leqslant 1/i} \cdot O(i) + \mathrm{Pr}\left[\text{max does not change}\right] \cdot O(1)$$

backwards analysis

max changes when adding $A[i]$ to $\{A[0], \ldots, A[i-1]\}$ $\iff$ max changes when removing $A[i]$ from $\{A[0], \ldots, A[i]\}$

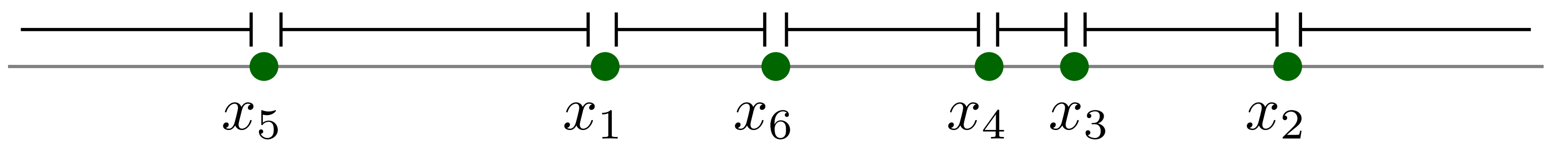
Sorting using (Randomized) Incremental Construction

A geometric view of sorting



Input: A set $S = \{x_1, \dots, x_n\}$ of n points in $\mathbb{R}^1$
Output: Sorted set $\mathcal{I}$ of intervals into which S partitions $\mathbb{R}^1$

A geometric view of sorting



Input: A set $S = \{x_1, \ldots, x_n\}$ of n points in $\mathbb{R}^1$
Output: Sorted set $\mathcal{I}$ of intervals into which S partitions $\mathbb{R}^1$

Incremental construction:

Add points one by one, and update $\mathcal{I}$ after each addition

Sorting using (Randomized) Incremental Construction

IC-SORT(S)

$\mathcal{I} \leftarrow \{[-\infty, +\infty]\}$
for $j \leftarrow 1$ **to** n **do**
    Find interval $I = [x, x']$ in $\mathcal{I}$ that contains x_j
    Remove I from $\mathcal{I}$ and insert $[x, x_j]$ and $[x_j, x']$ into $\mathcal{I}$
return $\mathcal{I}$

Sorting using (Randomized) Incremental Construction

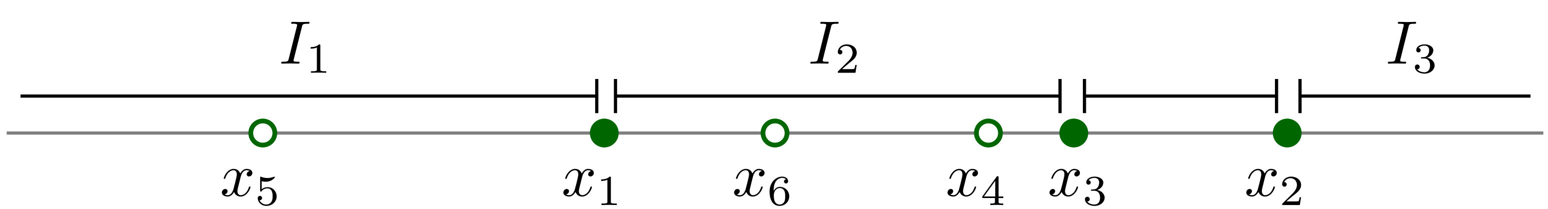
IC-SORT(S)

$\mathcal{I} \leftarrow \{[-\infty, +\infty]\}$
for $j \leftarrow 1$ **to** n **do**

Find interval $I = [x, x']$ in $\mathcal{I}$ that contains x_j
Remove I from $\mathcal{I}$ and insert $[x, x_j]$ and $[x_j, x']$ into $\mathcal{I}$

return $\mathcal{I}$

- for each point x_i maintain a pointer to the interval $I \in \mathcal{I}$ that contains x_i
- for each interval $I \in \mathcal{I}$ maintain a conflict list $K(I)$ that stores all points contained in I

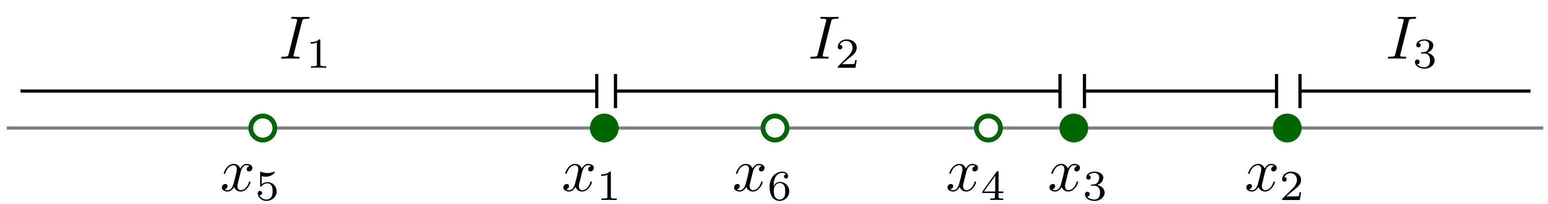


Sorting using (Randomized) Incremental Construction

IC-SORT(S)

1: $\mathcal{I} \leftarrow \{[-\infty, +\infty]\}$
2: **for** $j \leftarrow 1$ **to** n **do**
3:
4: **return** $\mathcal{I}$

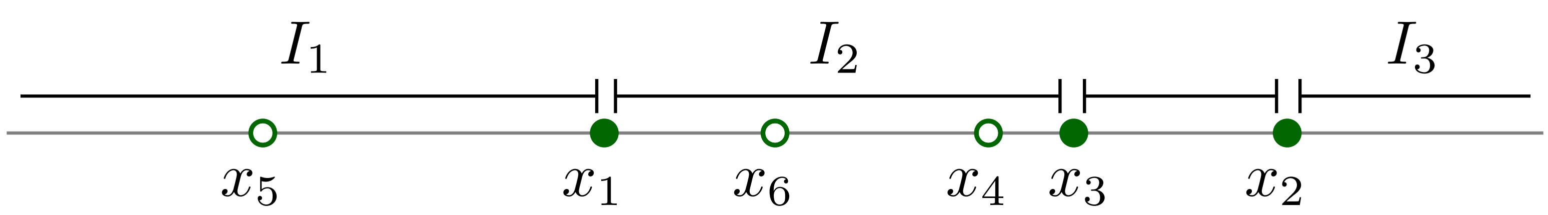
- for each point x_i maintain a pointer to the interval $I \in \mathcal{I}$ that contains x_i
- for each interval $I \in \mathcal{I}$ maintain a conflict list $K(I)$ that stores all points contained in I



IC-SORT(S)

1: $\mathcal{I} \leftarrow \{[-\infty, +\infty]\}$
2: **for** $j \leftarrow 1$ **to** n **do**
3: (i) Use pointer from x_i to find interval I containing x_i
   (ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
   (iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
   (iv) Update pointers of points in $K(I')$ and $K(I'')$
4: **return** $\mathcal{I}$

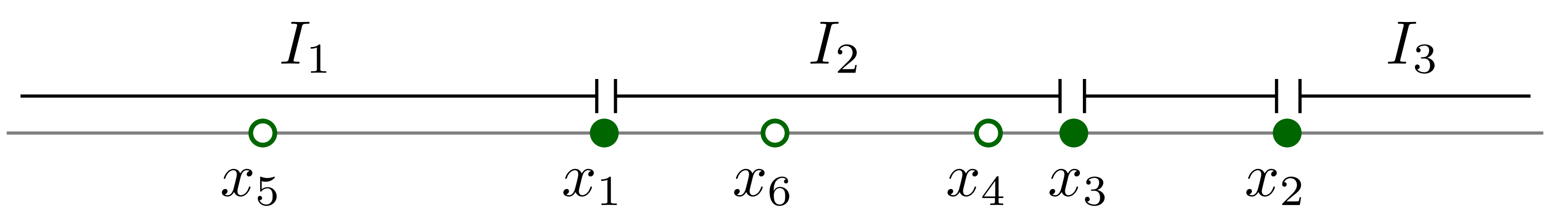
- for each point x_i maintain a pointer to the interval $I \in \mathcal{I}$ that contains x_i
- for each interval $I \in \mathcal{I}$ maintain a conflict list $K(I)$ that stores all points contained in I



IC-SORT(S)

$\mathcal{I} \leftarrow \{[-\infty, +\infty]\}$
for $j \leftarrow 1$ **to** n **do**
  (i) Use pointer from x_i to find interval I containing x_i
  (ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
  (iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
  (iv) Update pointers of points in $K(I')$ and $K(I'')$
return $\mathcal{I}$

- for each point x_i maintain a pointer to the interval $I \in \mathcal{I}$ that contains x_i
- for each interval $I \in \mathcal{I}$ maintain a conflict list $K(I)$ that stores all points contained in I



Sorting using (Randomized) Incremental Construction

IC-SORT(S)

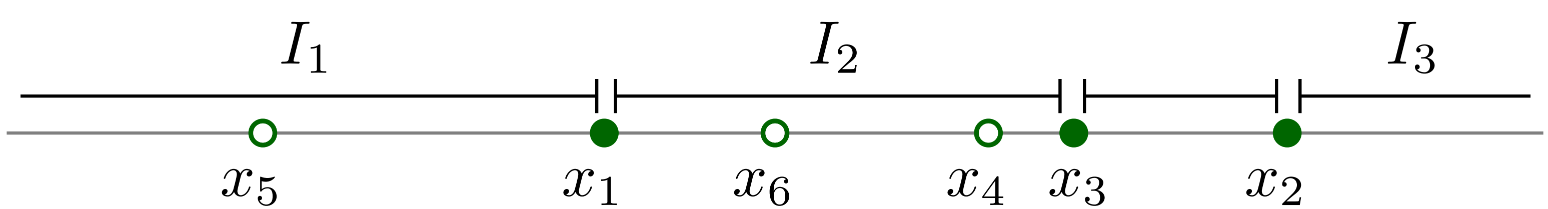
Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $K(I) \leftarrow S$

for $j \leftarrow 1$ **to** n **do**

- (i) Use pointer from x_i to find interval I containing x_i
- (ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
- (iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
- (iv) Update pointers of points in $K(I')$ and $K(I'')$

return $\mathcal{I}$

- for each point x_i maintain a pointer to the interval $I \in \mathcal{I}$ that contains x_i
- for each interval $I \in \mathcal{I}$ maintain a conflict list $K(I)$ that stores all points contained in I



Sorting using (Randomized) Incremental Construction

IC-SORT(S)

Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$
for $j \leftarrow 1$ **to** n **do**
    (i) Use pointer from x_i to find interval I containing x_i
    (ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
    (iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
    (iv) Update pointers of points in $K(I')$ and $K(I'')$
return $\mathcal{I}$

Sorting using (Randomized) Incremental Construction

IC-SORT(S)

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$
2: **for** $j \leftarrow 1$ **to** n **do**
3: (i) Use pointer from x_i to find interval I containing x_i
   (ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
   (iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
   (iv) Update pointers of points in $K(I')$ and $K(I'')$
4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

Sorting using (Randomized) Incremental Construction

IC-SORT(S)

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$
2: **for** $j \leftarrow 1$ **to** n **do**
3: (i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$
4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

- worst case: in each step j, we split a conflict list of size $n - j + 1$ into lists of size 0 and $n - j$

IC-SORT(S)

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$
2: **for** $j \leftarrow 1$ **to** n **do**
3: (i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$
4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

- worst case: in each step j, we split a conflict list of size $n - j + 1$ into lists of size 0 and $n - j$

  running time is $O(\sum_{j=1}^{n}(n-j+1)) = O(n^2)$

Sorting using (Randomized) Incremental Construction

IC-SORT(S)

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$
2: **for** $j \leftarrow 1$ **to** n **do**
3: (i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$
4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

Sorting using (Randomized) Incremental Construction

IC-SORT(S) Put points x_i in random order

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$

2: **for** $j \leftarrow 1$ **to** n **do**

3: (i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$

4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

Sorting using (Randomized) Incremental Construction

IC-Sort(S) ← Put points x_i in random order

Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$

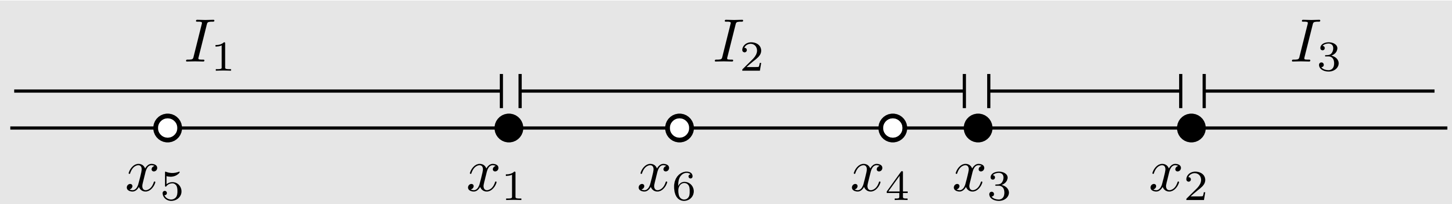
for $j \leftarrow 1$ **to** n **do**

(i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$

return $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

- expected:



apply backwards analysis

Sorting using (Randomized) Incremental Construction

IC-SORT(S) — Put points x_i in random order

1: Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$

2: **for** $j \leftarrow 1$ **to** n **do**

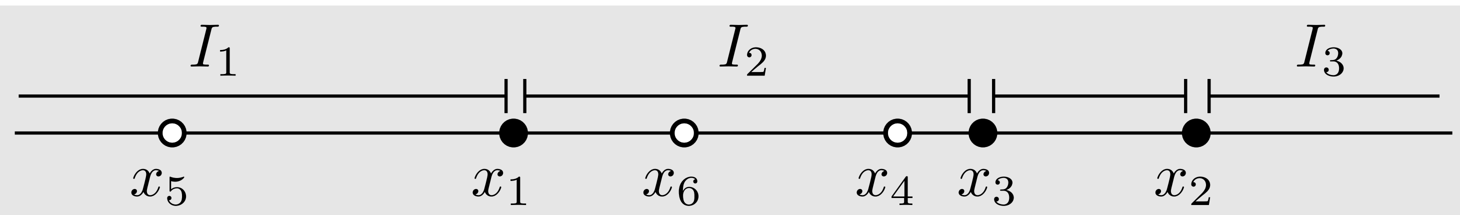
3: (i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$

4: **return** $\mathcal{I}$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

- expected:

at most
$1 + (n - j + 1) \cdot \frac{2}{j}$



Sorting using (Randomized) Incremental Construction

IC-Sort(S) ← Put points x_i in random order

Set $I \leftarrow [-\infty, \infty]$ and $\mathcal{I} \leftarrow \{I\}$, give each x_j a pointer to I, set $\mathcal{L}(I) \leftarrow S$

for $j \leftarrow 1$ **to** n **do**

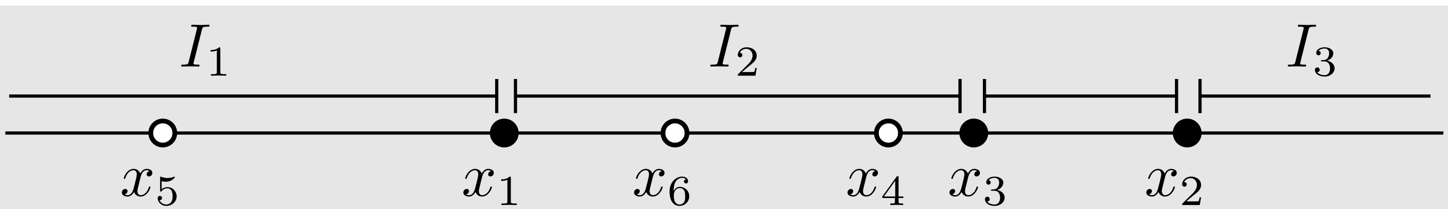
(i) Use pointer from x_i to find interval I containing x_i
(ii) Split I at x_i into intervals I' and I'', and replace I in $\mathcal{I}$ by I', I''
(iii) Construct $K(I')$ and $K(I'')$ from $K(I)$
(iv) Update pointers of points in $K(I')$ and $K(I'')$

return $\mathcal{I}$

$$\sum_{j=1}^{n} \left(1 + \frac{2(n-j+1)}{j}\right) = O\left(n + n\sum_{j=1}^{n} \frac{1}{j}\right) = O(n \log n)$$

Running time: $O(\sum_{j=1}^{n}$ (size of conflict list split in j-th iteration))

- expected:



at most
$1 + (n-j+1) \cdot \frac{2}{j}$

apply backwards analysis

Randomized Incremental Construction: The Framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
    $K(\Delta) \cap D(\Delta) = \emptyset$ for all Δ

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
    $K(\Delta) \cap D(\Delta) = \emptyset$ for all Δ

For $S' \subseteq S$, define $\mathcal{C}_{\mathrm{act}}(S') = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S'$ and $K(\Delta) \cap S' = \emptyset\}$
to be the set of configurations that are active with respect to S'

Goal: compute set $\mathcal{C}_{\mathrm{act}}(S)$ of active configurations with respect to S

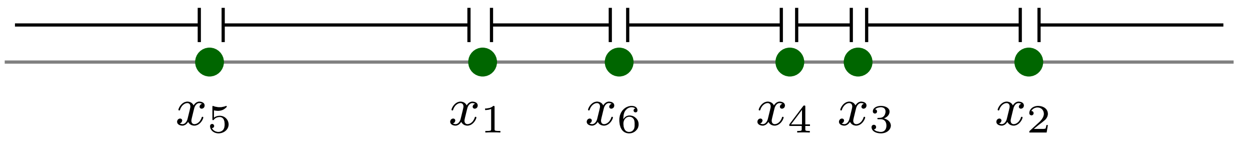
Randomized Incremental Construction: The Framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
    $K(\Delta) \cap D(\Delta) = \emptyset$ for all Δ

For $S' \subseteq S$, define $\mathcal{C}_{\text{act}}(S') = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S' \text{ and } K(\Delta) \cap S' = \emptyset\}$
to be the set of configurations that are active with respect to S'

Goal: compute set $\mathcal{C}_{\text{act}}(S)$ of active configurations with respect to S

Example: sorting



$\mathcal{C}(S) := \{ \ [x_i, x_j] : x_i, x_j \in S \cup \{-\infty, +\infty\} \text{ and } x_i < x_j \ \}$

Randomized Incremental Construction: The Algorithm

```
RIC-ALGORITHM(S)
Compute a random permutation x_1, ..., x_n of the objects in S
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove configurations from C_act that are in conflict with x_j
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

```
Compute a random permutation x_1, ..., x_n of the objects in S
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove configurations from C_act that are in conflict with x_j
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```

To find configurations that become inactive:

- for each x_j maintain a list of all configurations $\Delta \in \mathcal{C}_{\text{act}}$ with $x_j \in K(\Delta)$
- for each configuration $\Delta \in \mathcal{C}_{\text{act}}$ maintain its conflict list $K(\Delta)$

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\emptyset\}$
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\emptyset\}$
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$

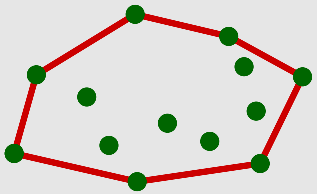
Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j) \setminus \mathcal{C}_{\mathrm{act}}(S_{j-1})| \,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\mathrm{act}}(S_j)]}{j}\right)$

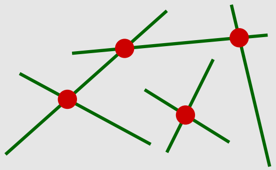
(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j)| \,\right]\right)$

Exercises

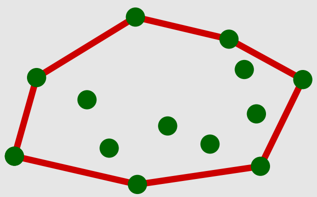
1. Give an algorithm that computes (all edges of) the convex hull of a set S of n points in the plane that runs in $O(n \log n)$ expected time.



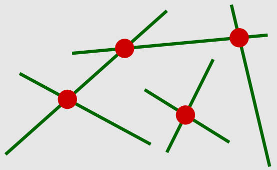
2. Give an algorithm that computes all k intersections in a set S of n segments in the plane that runs in $O(n \log n + k)$ expected time.



1. Give an algorithm that computes (all edges of) the convex hull of a set S of n points in the plane that runs in $O(n \log n)$ expected time.

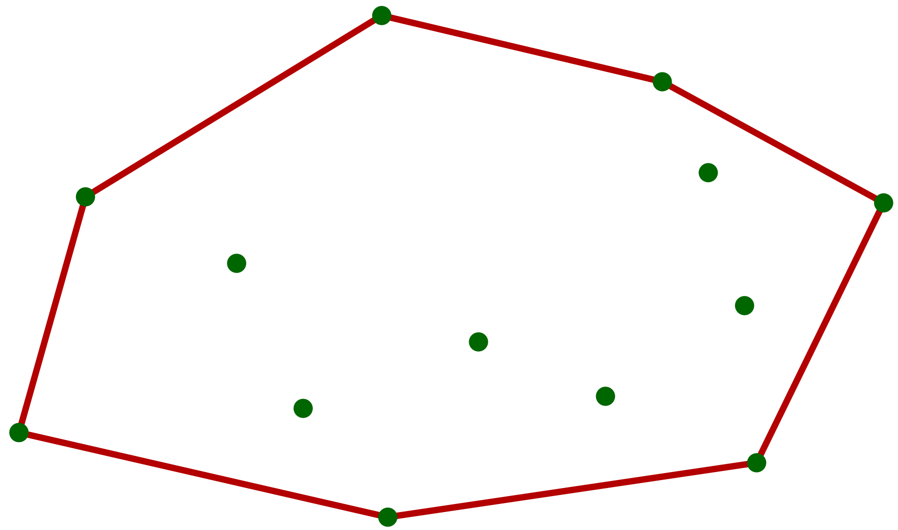


2. Give an algorithm that computes all k intersections in a set S of n segments in the plane that runs in $O(n \log n + k)$ expected time.



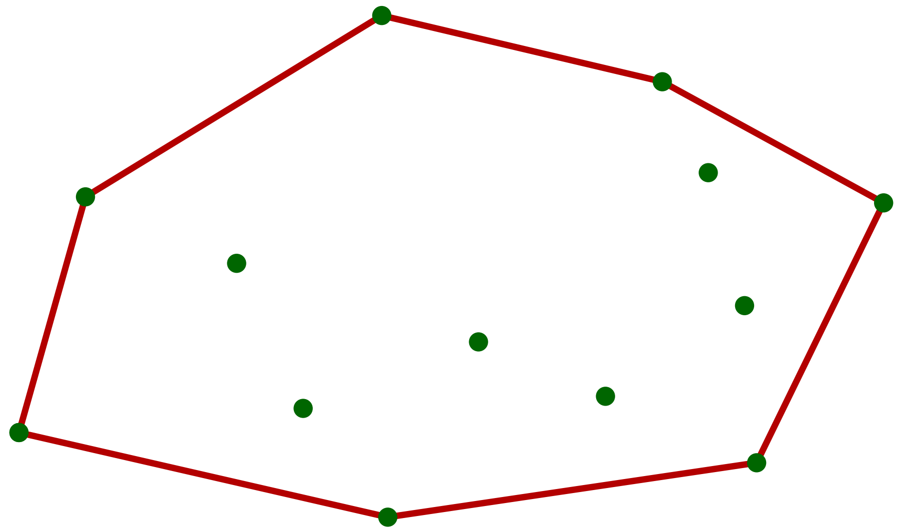
The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



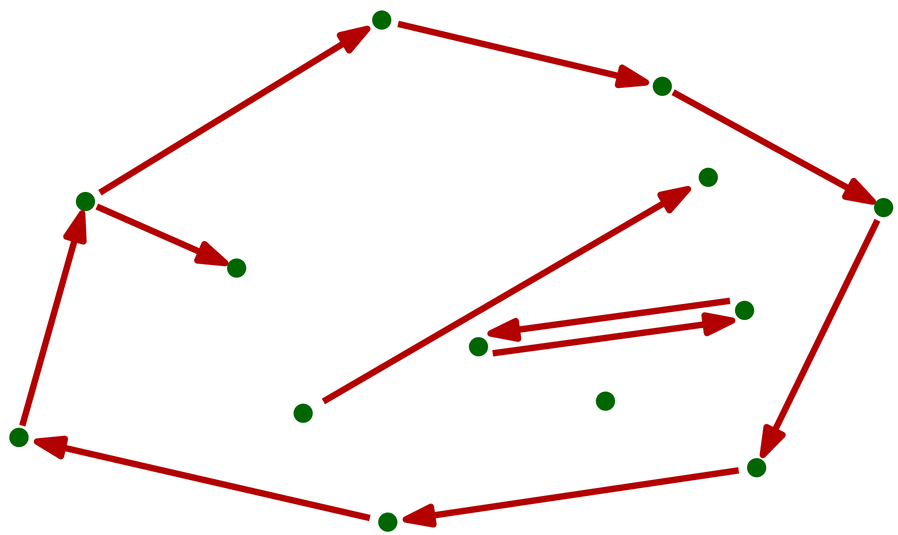
The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$



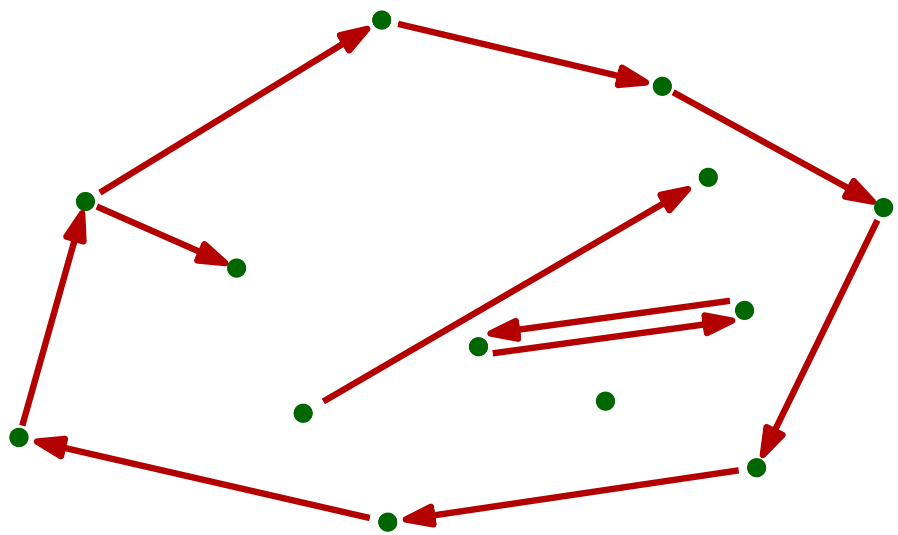
The framework

- S = set of n input objects   the points
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



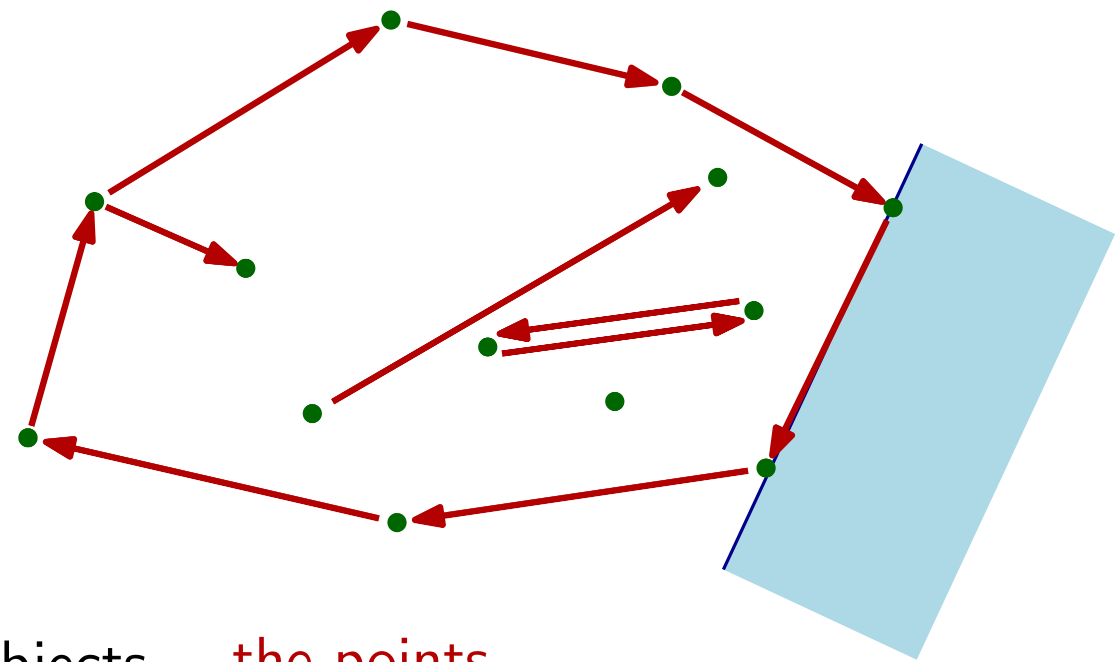
The framework

- S = set of n input objects   the points
- $\mathcal{C}(S)$ = set of configurations defined by S directed segments
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



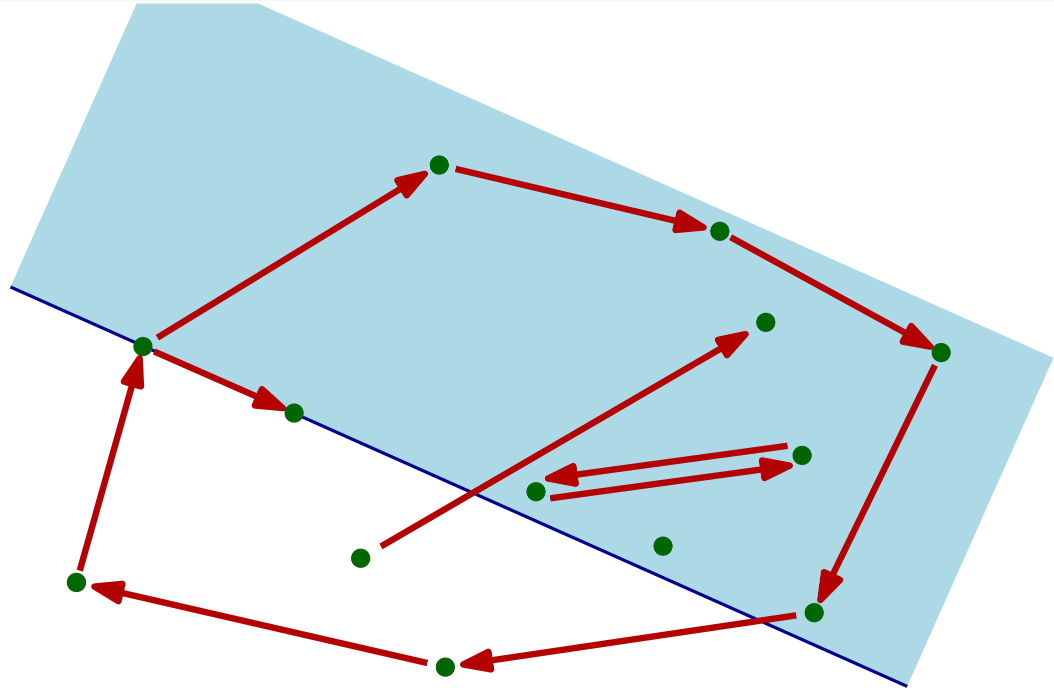
The framework

- S = set of n input objects   the points
- $\mathcal{C}(S)$ = set of configurations defined by S directed segments
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$ endpoints
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



The framework

- S = set of n input objects the points
- $\mathcal{C}(S)$ = set of configurations defined by S directed segments
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$ endpoints
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$ points left of (extended) segment
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



The framework

- S = set of n input objects   the points
- $\mathcal{C}(S)$ = set of configurations defined by S directed segments
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$ endpoints
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$ points left of (extended) segment
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Randomized Incremental Construction: The Algorithm

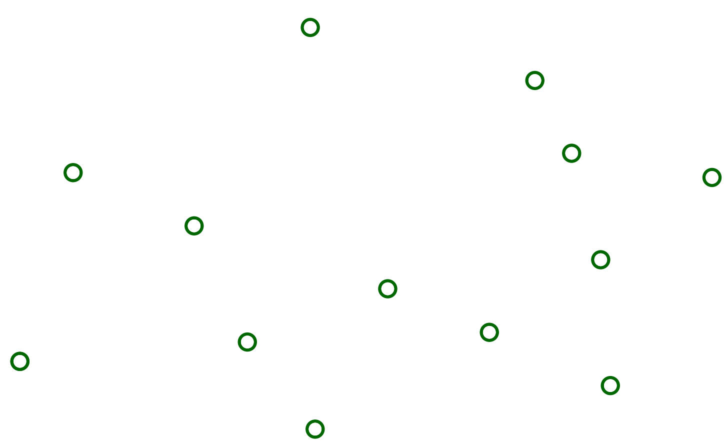
RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$

Randomized Incremental Construction: The Algorithm

RIC-Algorithm(S)

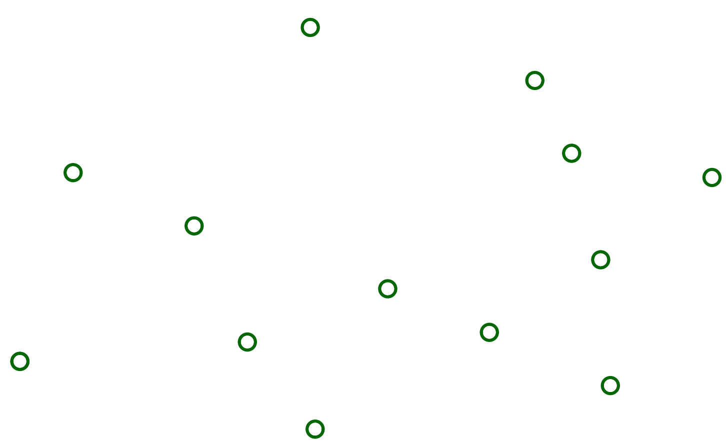
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$



Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

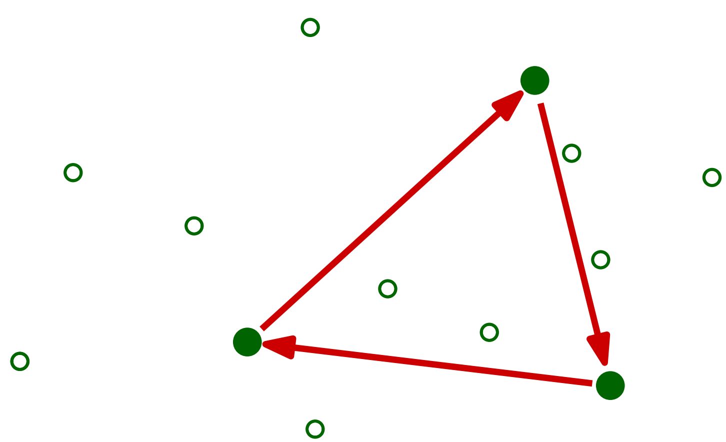
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$



Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

```
Compute a random permutation x_1, ..., x_n of the objects in S
C_act ← {active configurations with respect to ~~∅~~} first three points
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove configurations from C_act that are in conflict with x_j
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S

$\mathcal{C}_{\text{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points

Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$

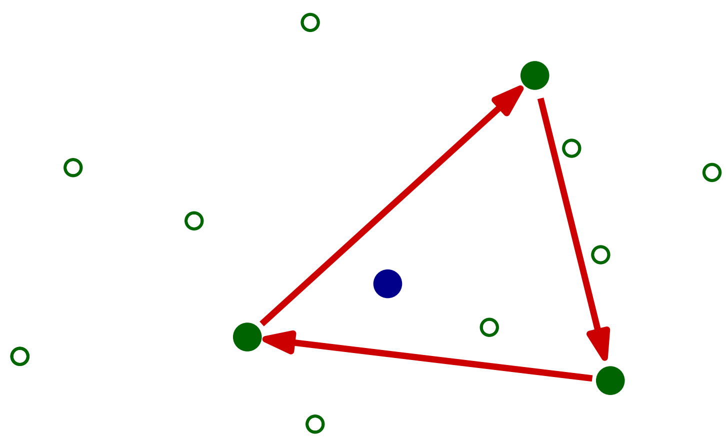
for $j \leftarrow 1$ **to** n **do**

  Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j

  Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$

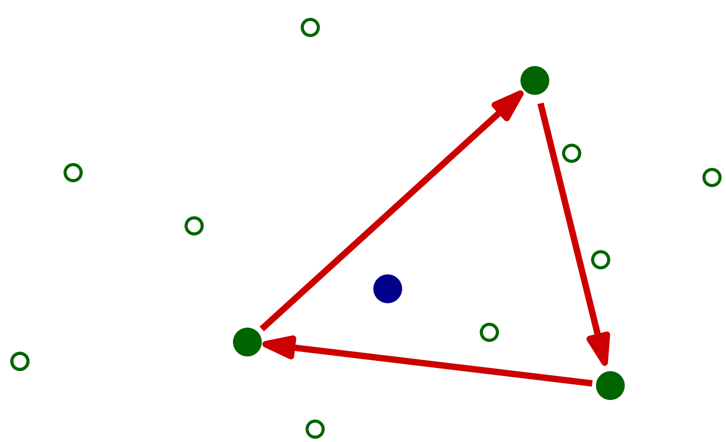
  Construct conflict lists of new active configurations

return $\mathcal{C}_{\text{act}}$



RIC-ALGORITHM(S)

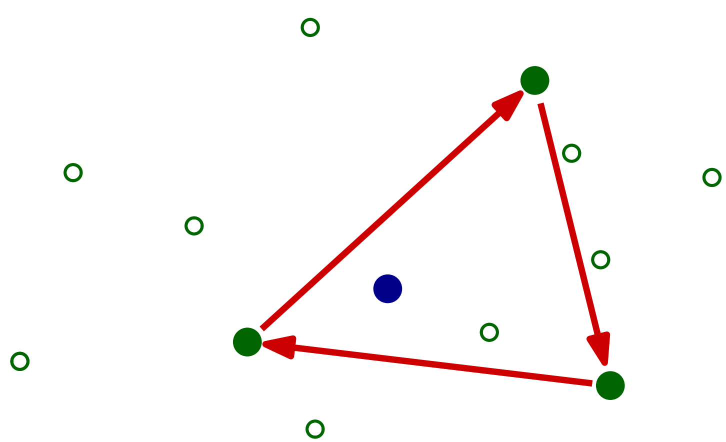
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\cancel{\emptyset}$} first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$



- not in conflict with active configs
- no new active configs

RIC-ALGORITHM(S)

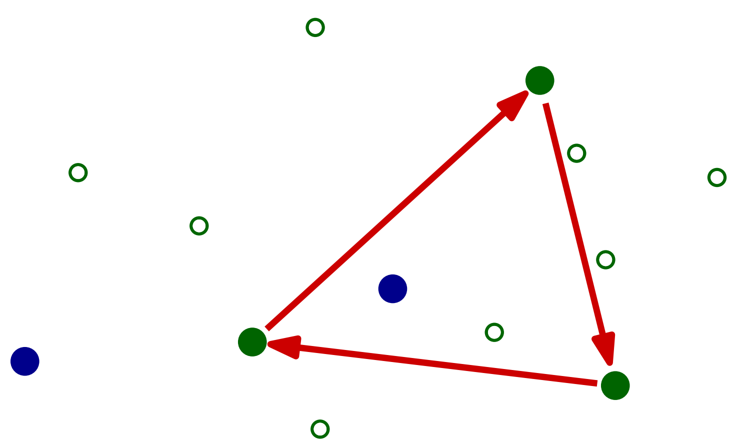
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\cancel{\emptyset}$} first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$



Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

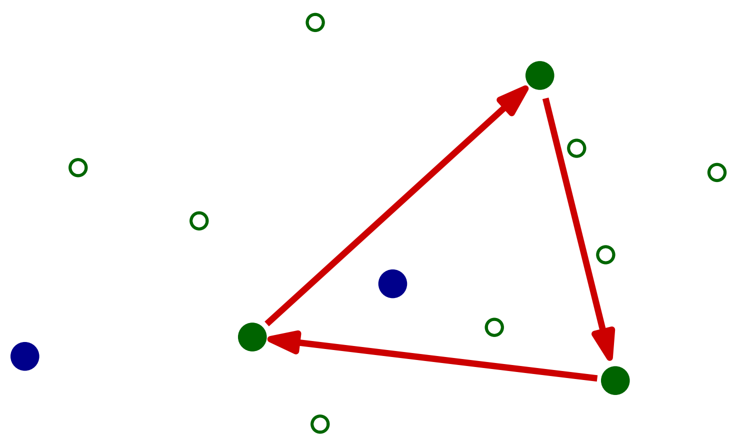
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$



Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$

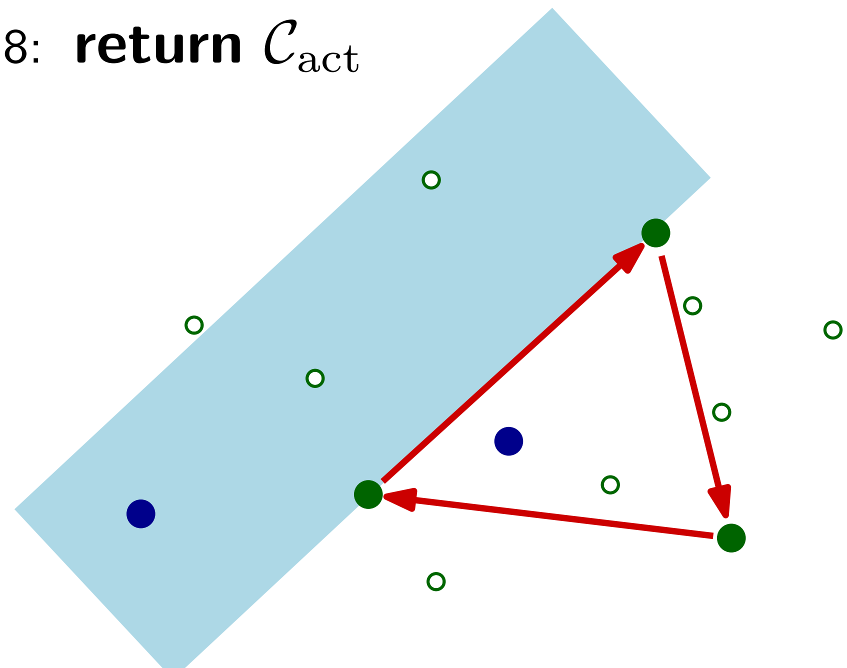


- in conflict with two configs

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

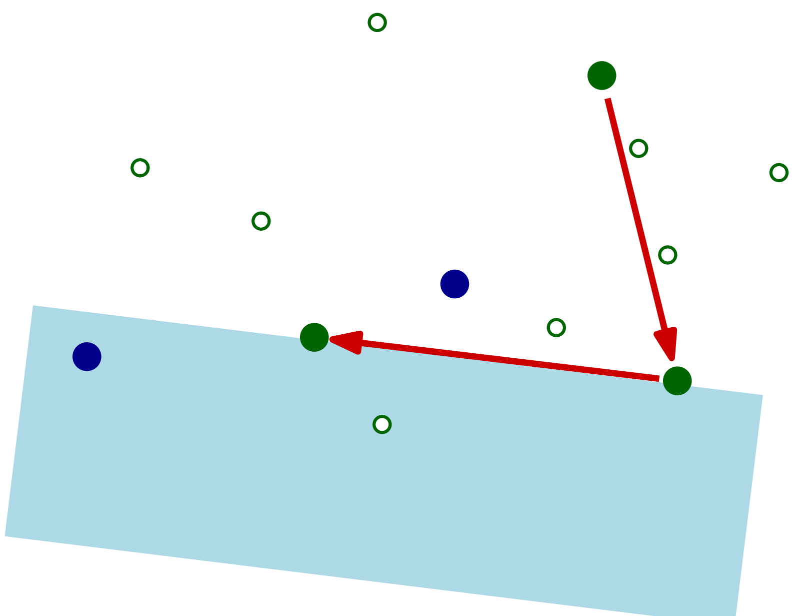
```
Compute a random permutation x_1, ..., x_n of the objects in S
C_act ← {active configurations with respect to ∅} first three points
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove configurations from C_act that are in conflict with x_j
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



- in conflict with two configs

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$

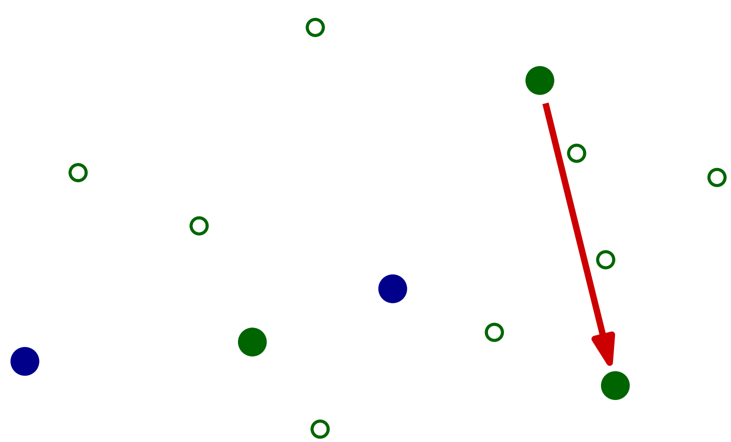


- in conflict with two configs

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow$ {active configurations with respect to $\cancel{\emptyset}$} first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$

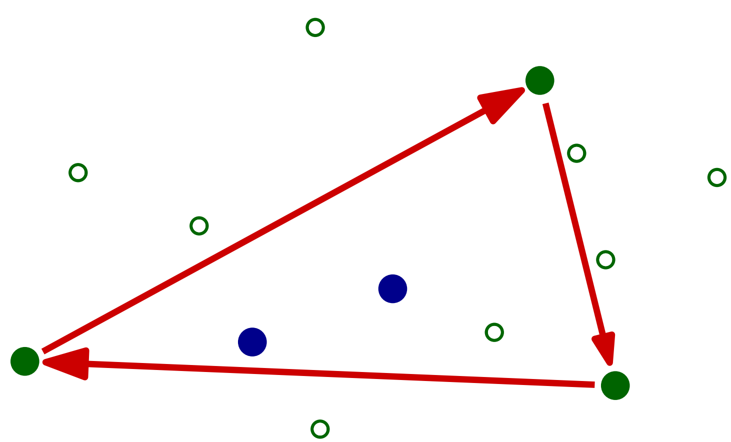


- in conflict with two configs
- two new configs appear

Randomized Incremental Construction: The Algorithm

RIC-ALGORITHM(S)

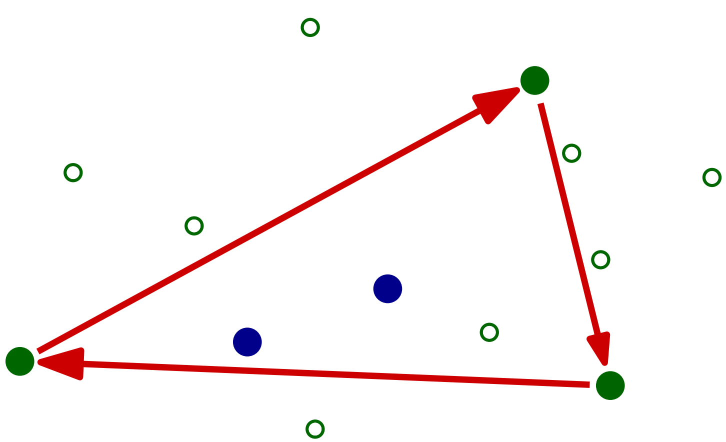
Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\cancel{\emptyset}\}$ first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\mathrm{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$



- in conflict with two configs
- two new configs appear

RIC-ALGORITHM(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\cancel{\emptyset}$} first three points
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove configurations from $\mathcal{C}_{\text{act}}$ that are in conflict with x_j
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\text{act}}$



- in conflict with two configs
- two new configs appear

  conflict lists are subset of union of old conflict lists

Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\, |\mathcal{C}_{\text{act}}(S_j) \setminus \mathcal{C}_{\text{act}}(S_{j-1)}| \,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\text{act}}(S_j)]}{j}\right)$

(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\text{act}}(S_j)| \,\right]\right)$

Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j) \setminus \mathcal{C}_{\mathrm{act}}(S_{j-1})| \,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\mathrm{act}}(S_j)]}{j}\right)$

(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j)| \,\right]\right)$

- $|\mathcal{C}_{\mathrm{act}}(S_j)| \leqslant j$
- total running time is linear in total size of all (dis)appearing conflict lists

Theorem. Let $S_j := \{x_1, \dots, x_j\}$. Then

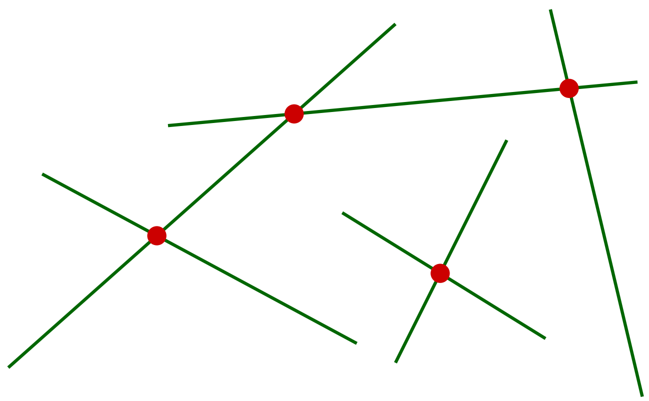
(i) $\mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j) \setminus \mathcal{C}_{\mathrm{act}}(S_{j-1)}|\, \right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\mathrm{act}}(S_j)]}{j} \right)$

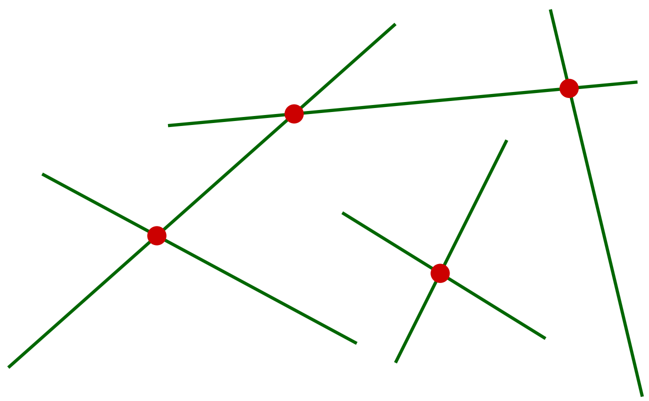
(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j)|\, \right]\right)$

- $|\mathcal{C}_{\mathrm{act}}(S_j)| \leqslant j$
- total running time is linear in total size of all (dis)appearing conflict lists

convex-hull algorithm runs in $O(n \log n)$ time

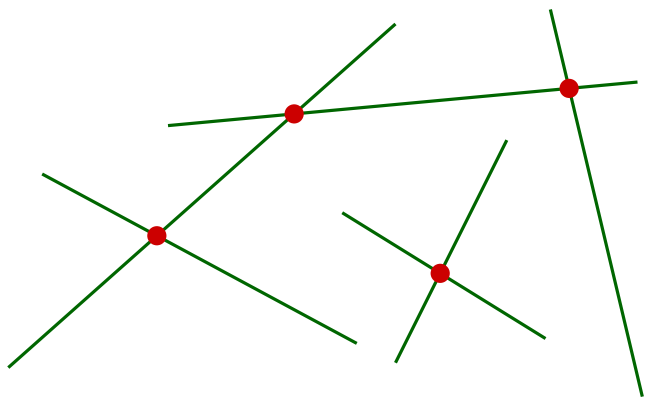
Line-Segment Intersection with RIC





The framework

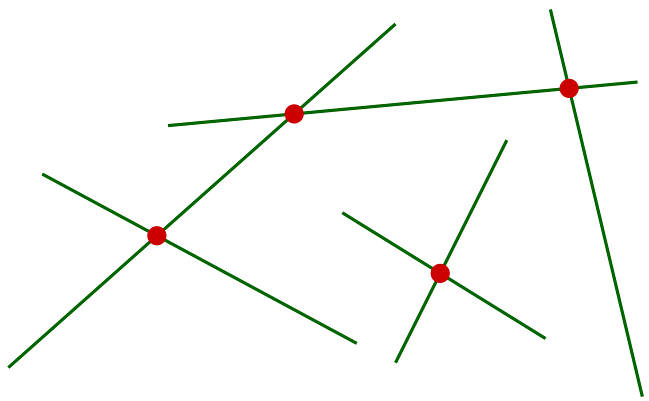
- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



Configurations?

The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

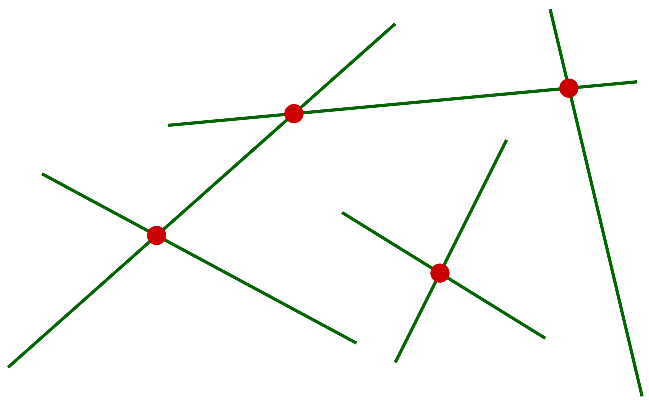


Configurations?

- intersection points does not work (find new configurations?)

The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

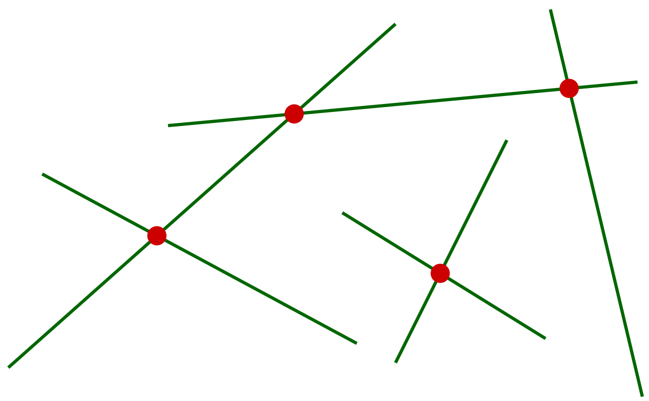


Configurations?

- intersection points does not work (find new configurations?)
- "subsegments" of segments does not work (initialization?)

The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$



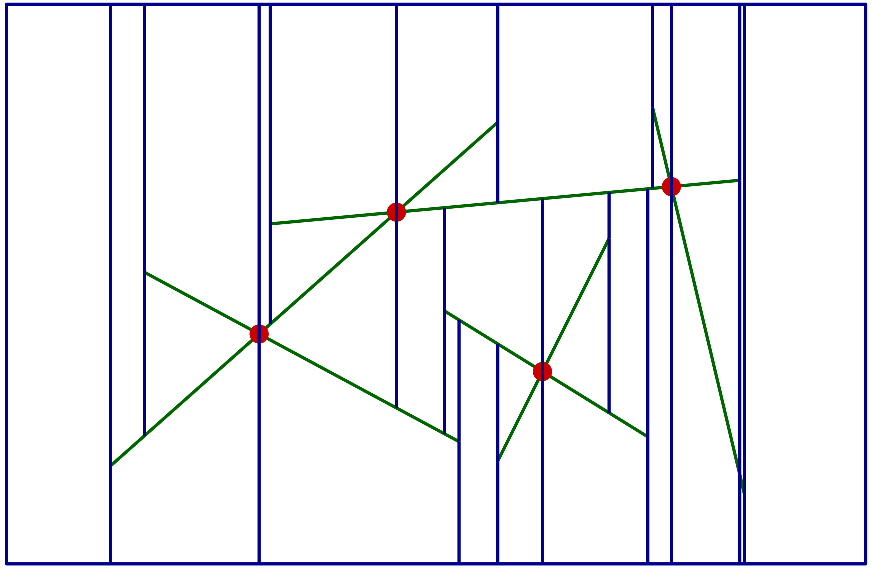
Configurations?

- intersection points does not work (find new configurations?)
- "subsegments" of segments does not work (initialization?)
- construct vertical decomposition

The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Line-Segment Intersection with RIC



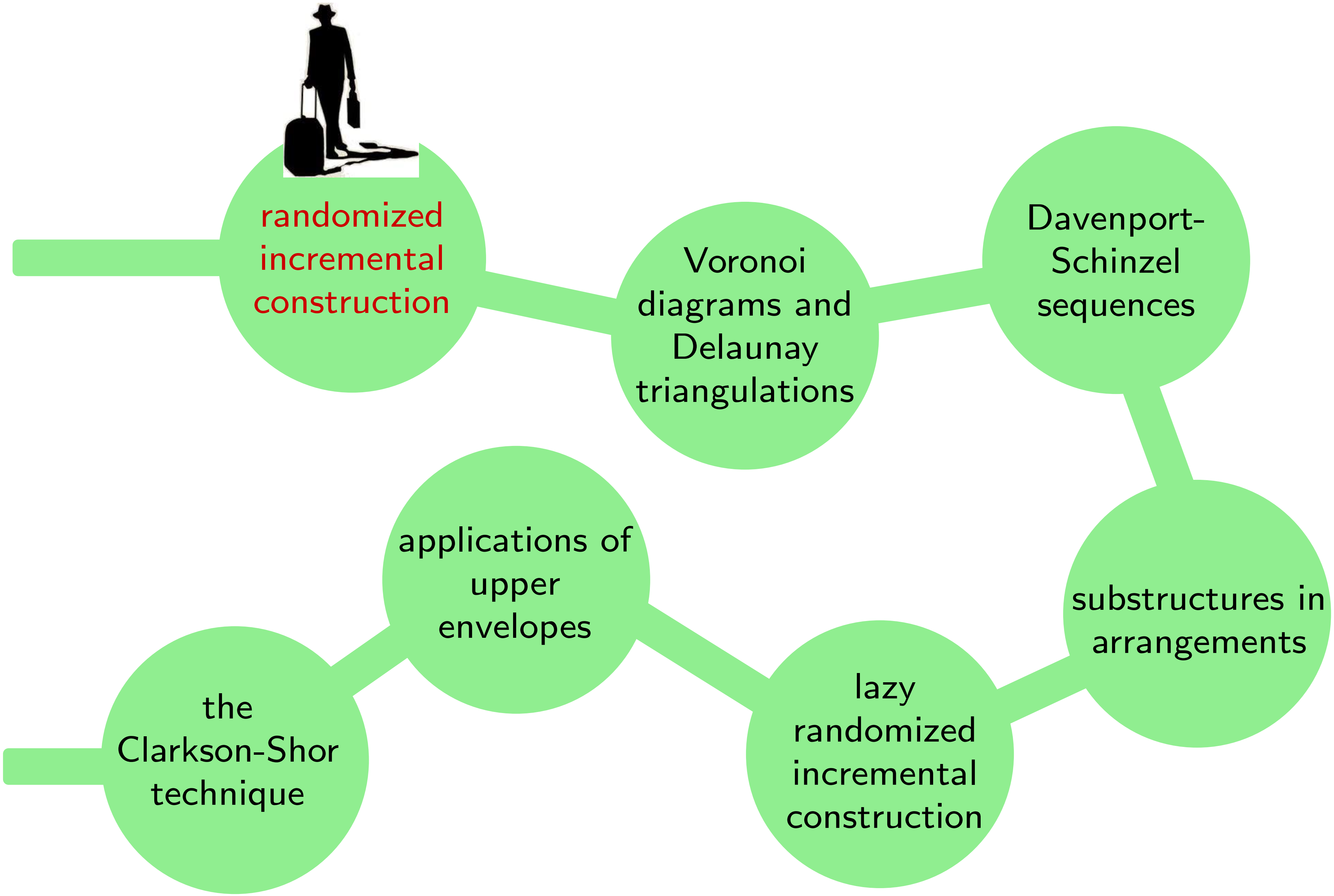
Configurations?

- intersection points does not work (find new configurations?)
- "subsegments" of segments does not work (initialization?)
- construct vertical decomposition

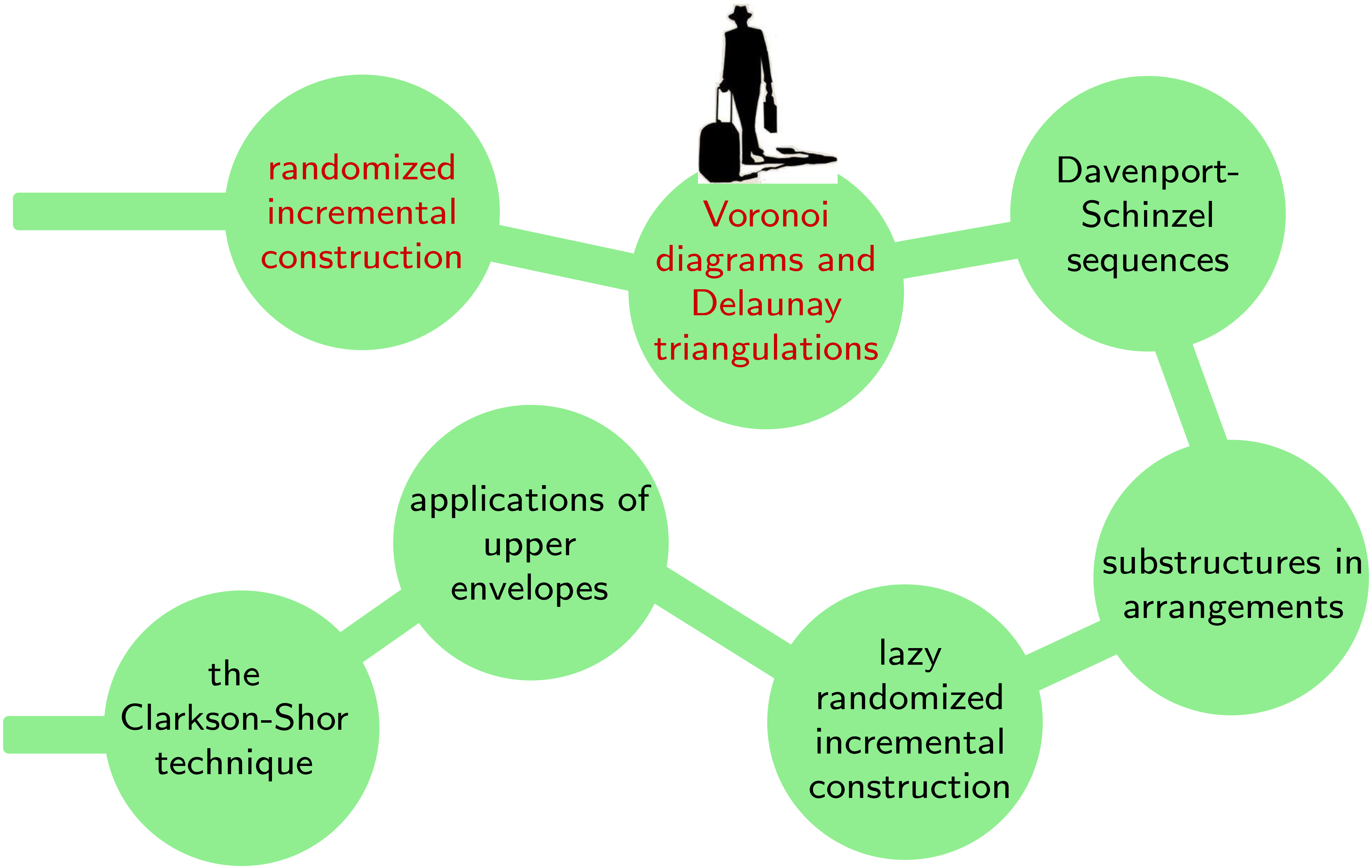
The framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size should be bounded by a fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Course Overview



Course Overview



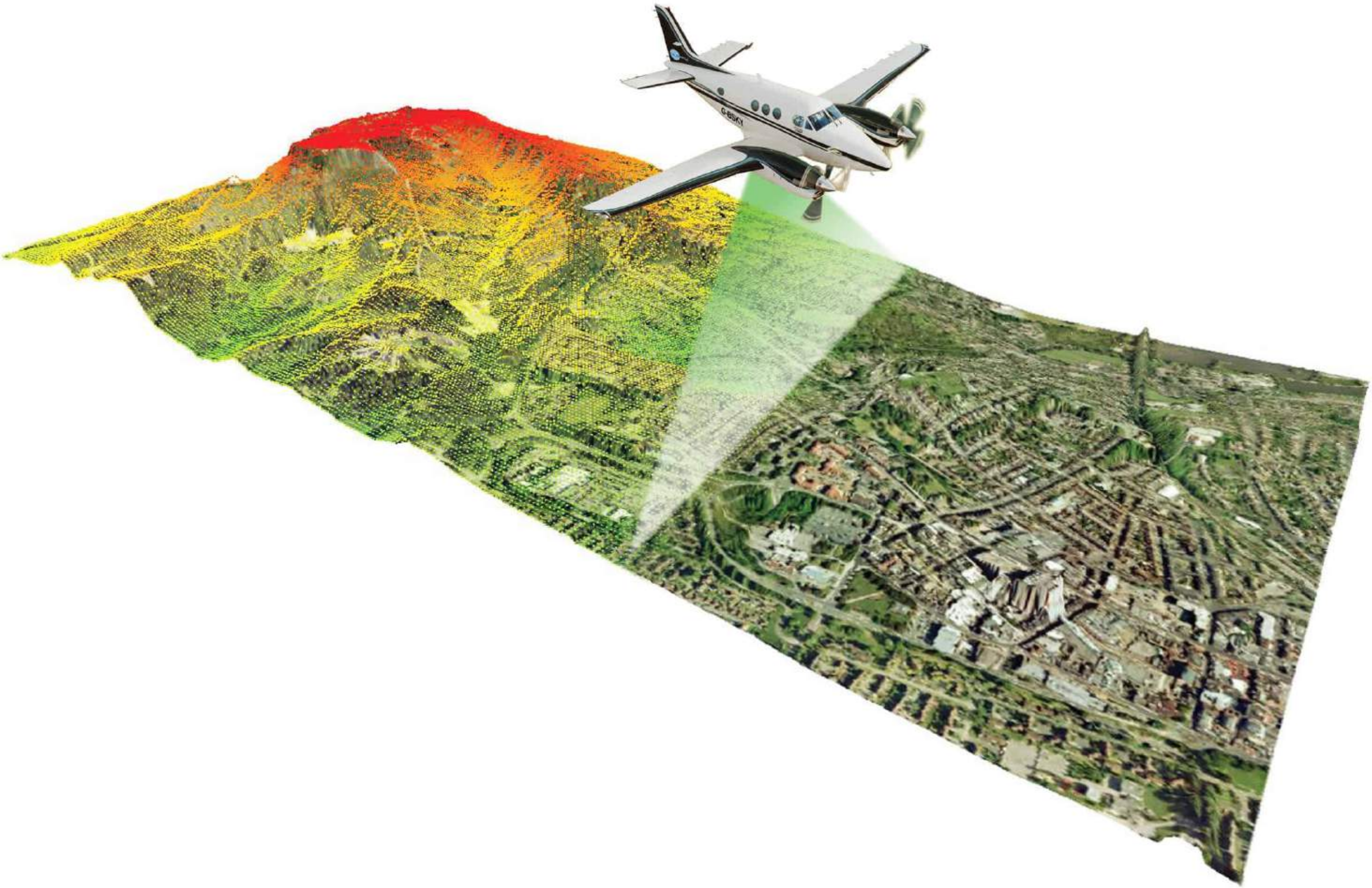
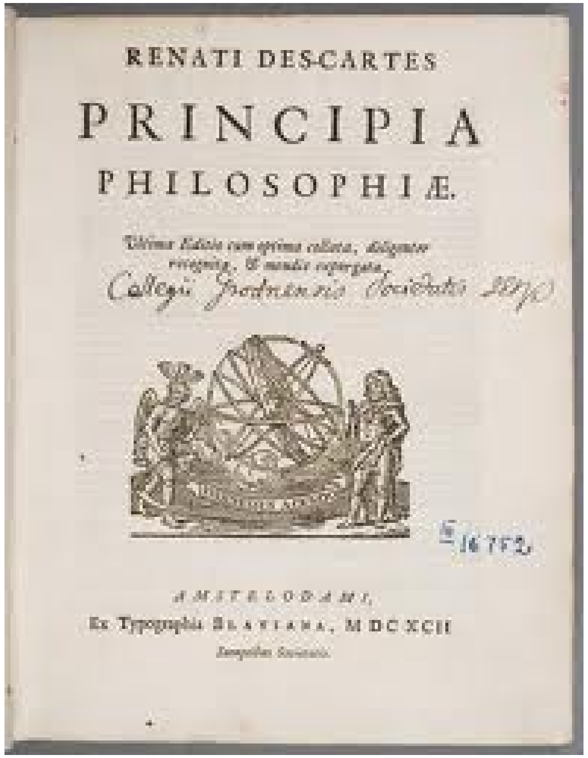


Image: www.aurorasolar.com

Principia Philosiphiae (Descartes, 1664)



RENATI DES-CARTES

PRINCIPIA

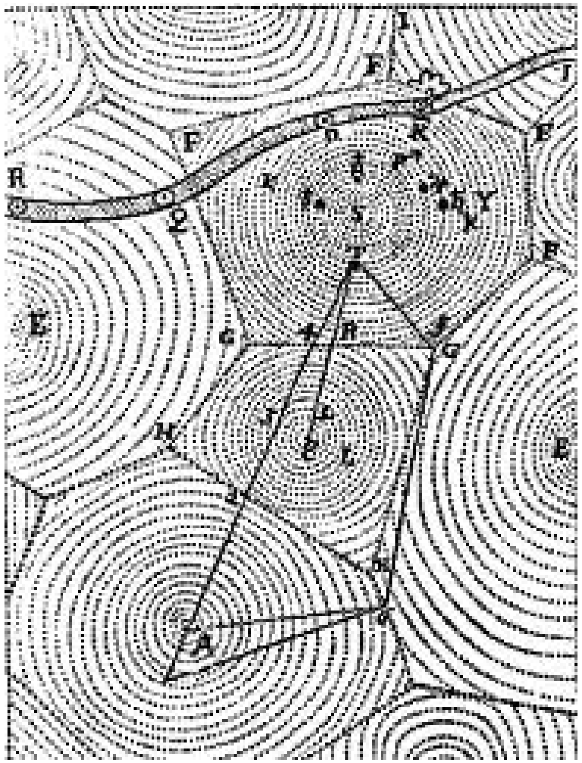
PHILOSOPHIÆ.

Ultima Editio cum optima collata, diligenter recognita, & mendis expurgata.

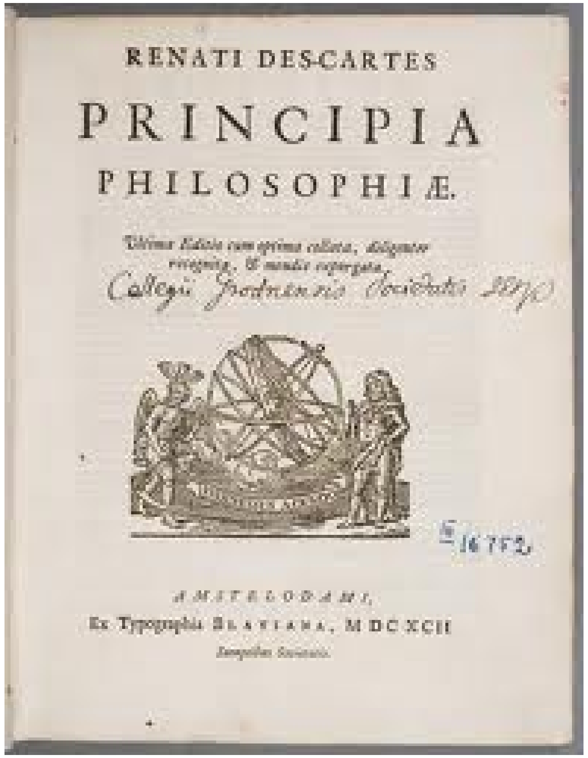
AMSTELODAMI,

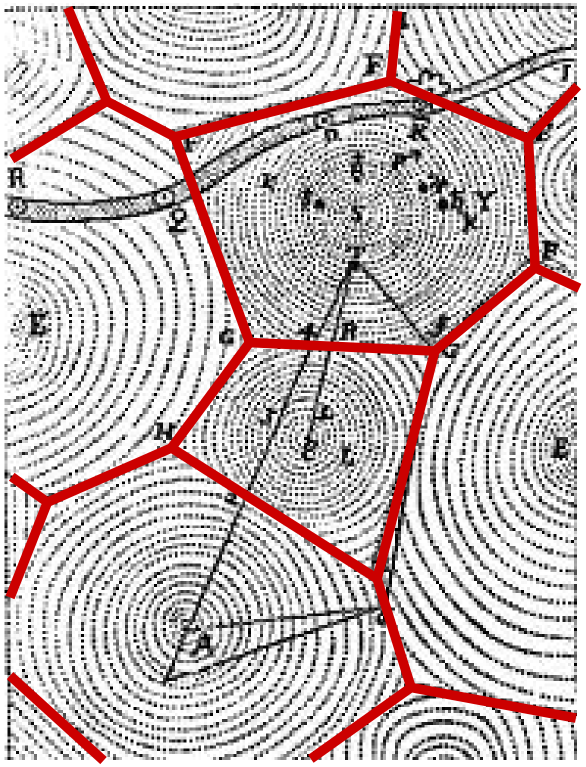
Ex Typographia BLAVIANA, MDCXCII

Sumptibus Societatis.



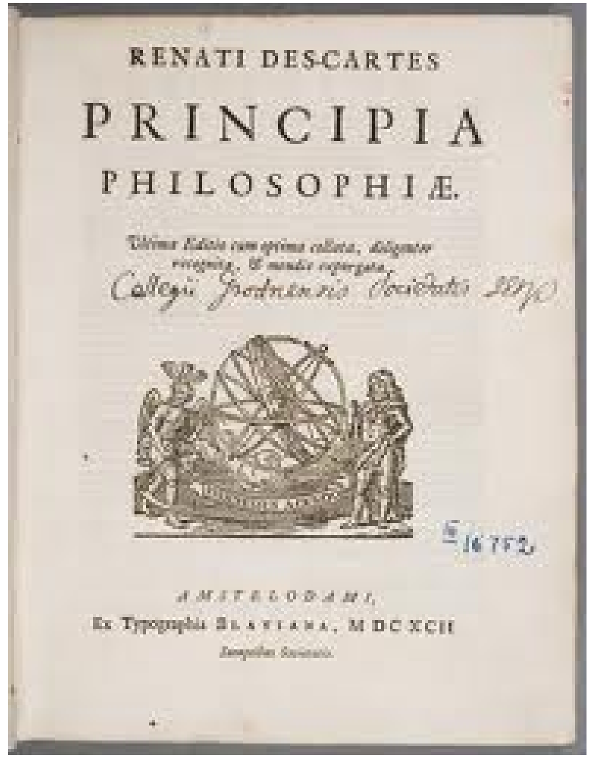
Principia Philosiphiae (Descartes, 1664)





Voronoi diagram

Principia Philosiphiae (Descartes, 1664)



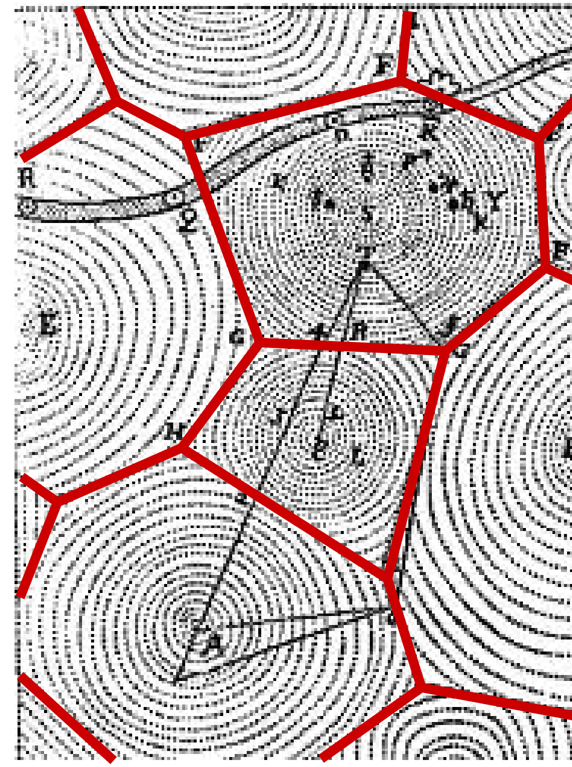
RENATI DESCARTES

PRINCIPIA

PHILOSOPHIÆ.

AMSTELODAMI,

Ex Typographia BLAVIANA, MDCXCII

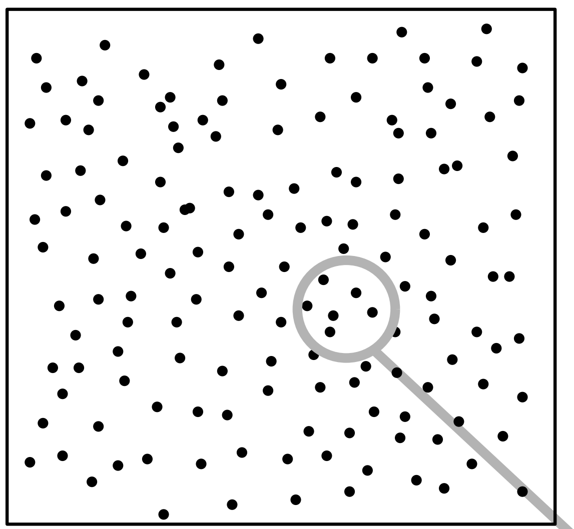


Voronoi diagram

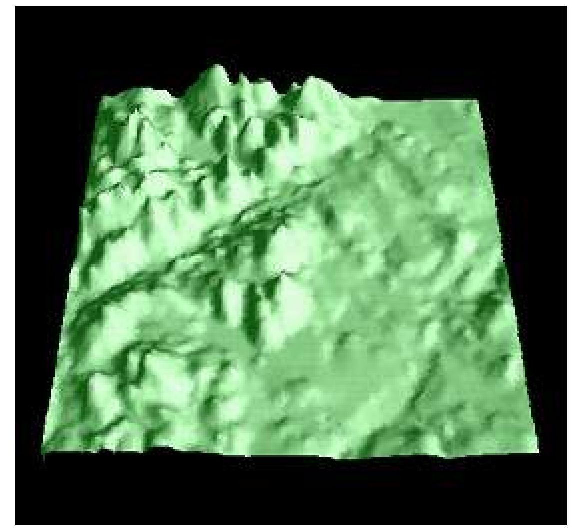
Georgy Voronoy
(1868-1908)

Terrain Reconstruction from Elevation Data

Back to terrain reconstruction . . .

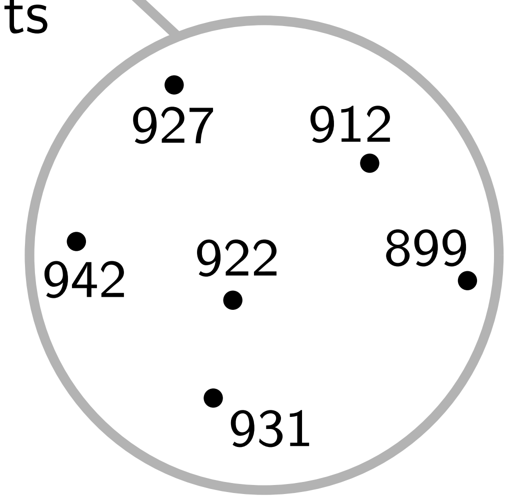


$\Longrightarrow$

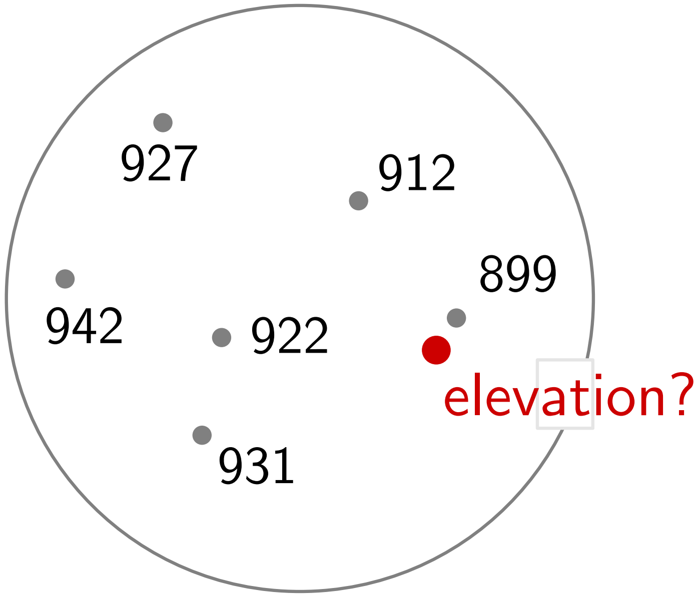


Input: terrain elevation at sample points

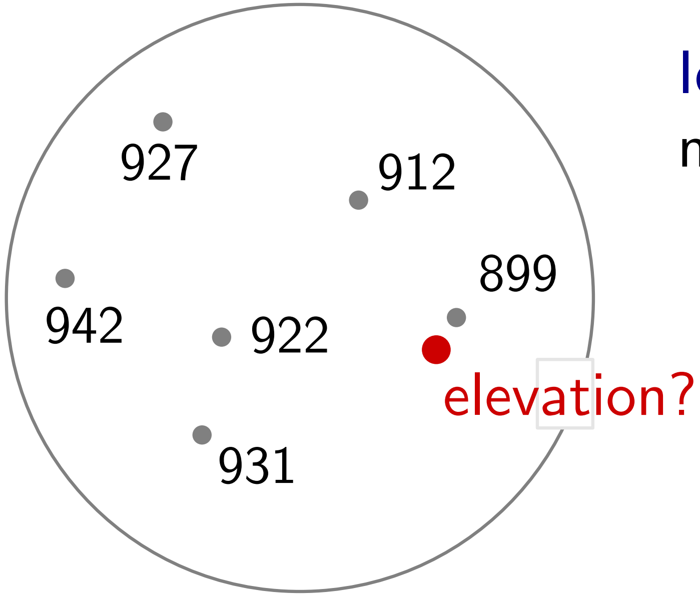
Goal: construct continuous terrain surface



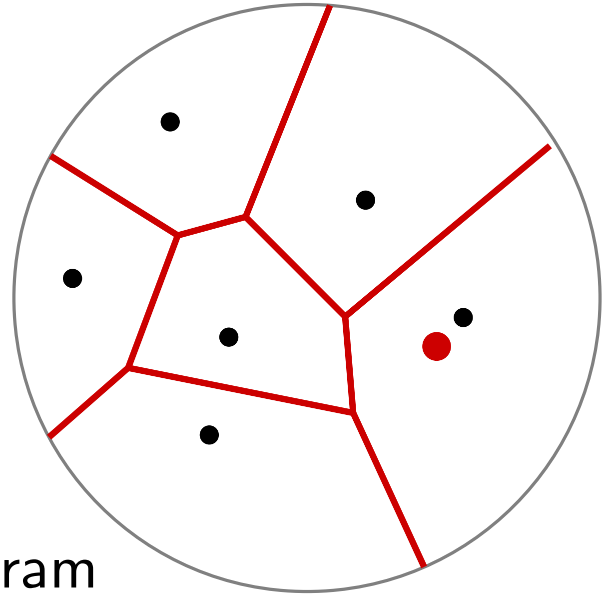
Terrain Reconstruction from Elevation Data



Terrain Reconstruction from Elevation Data

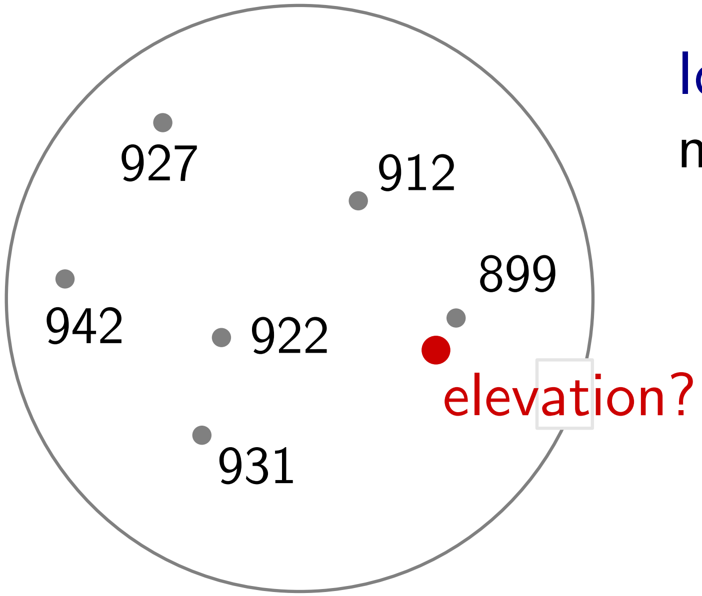


Idea: use elevation of nearest sample point

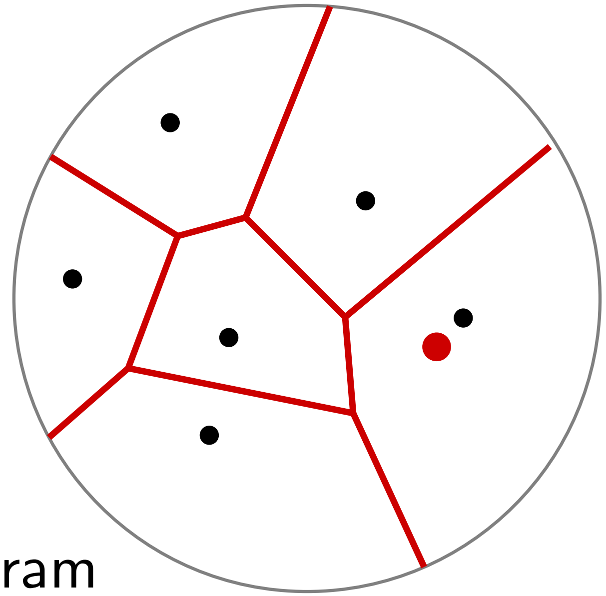


Voronoi diagram

Terrain Reconstruction from Elevation Data

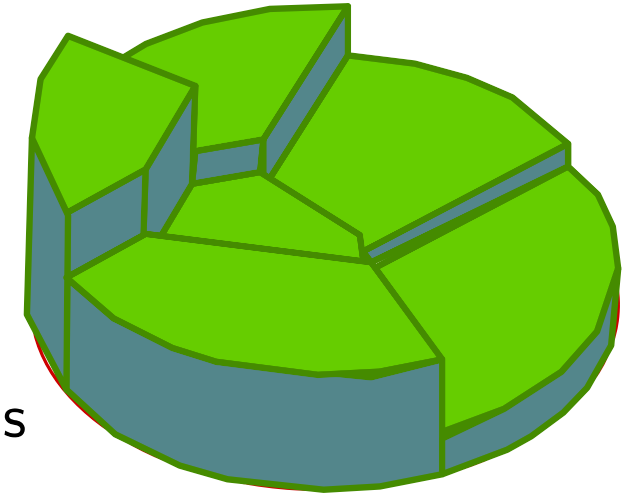


Idea: use elevation of nearest sample point



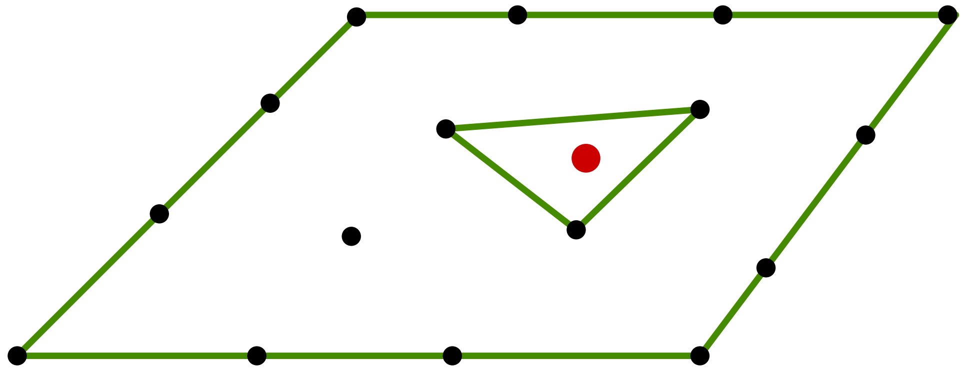
Voronoi diagram

Not good: surface not continuous



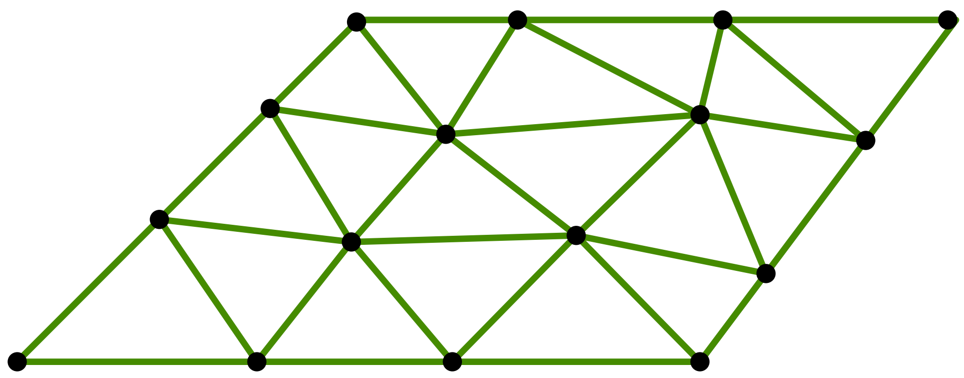
Terrain Reconstruction from Elevation Data

Better idea: determine elevation using interpolation



Terrain Reconstruction from Elevation Data

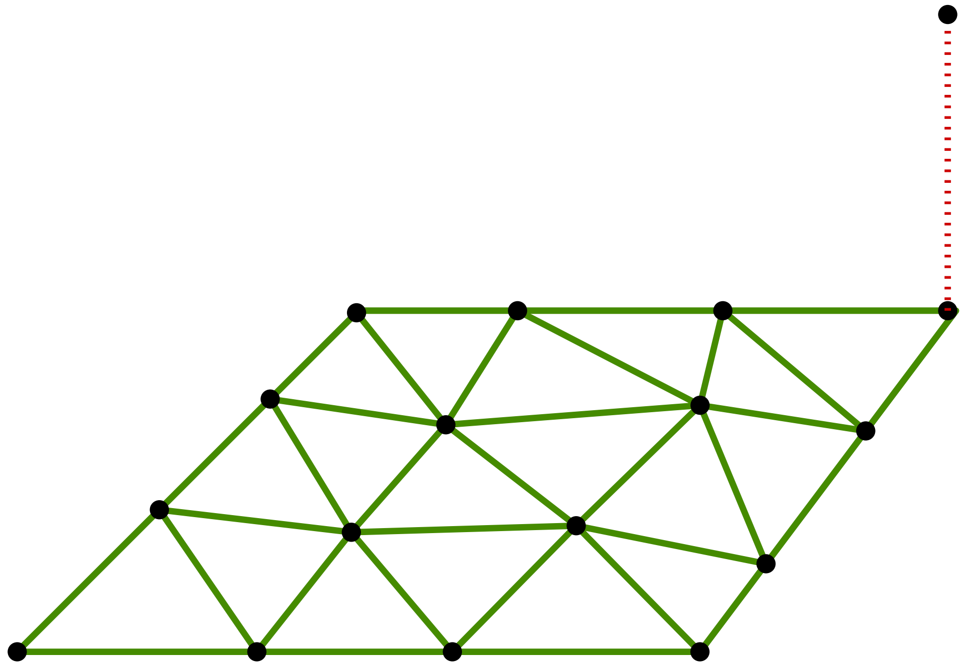
Better idea: determine elevation using interpolation



triangulation

Terrain Reconstruction from Elevation Data

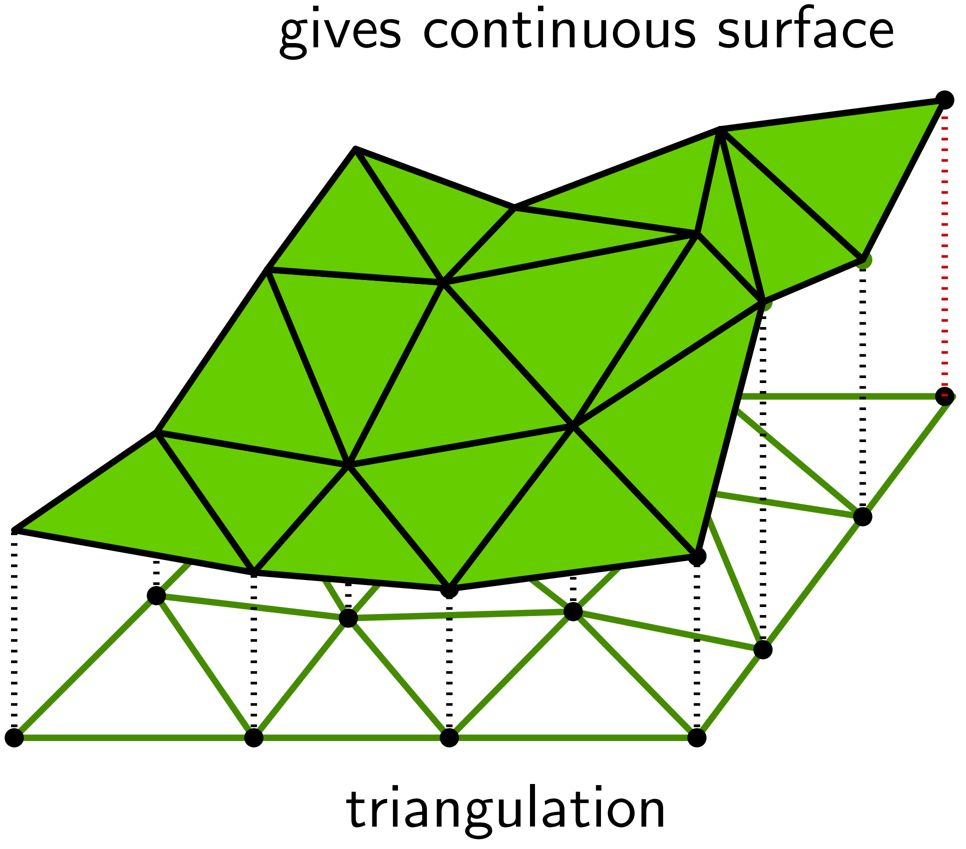
Better idea: determine elevation using interpolation



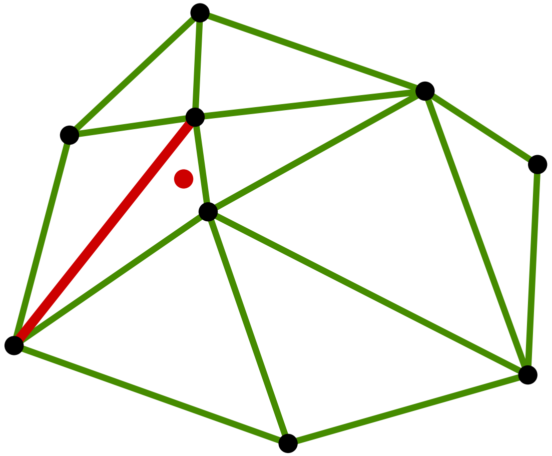
triangulation

Terrain Reconstruction from Elevation Data

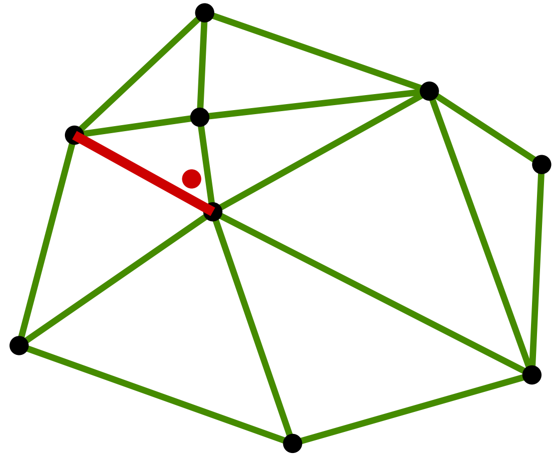
Better idea: determine elevation using interpolation



Which triangulation should we use?



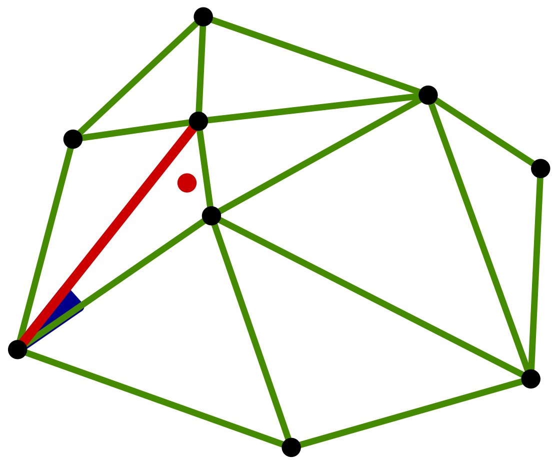
or



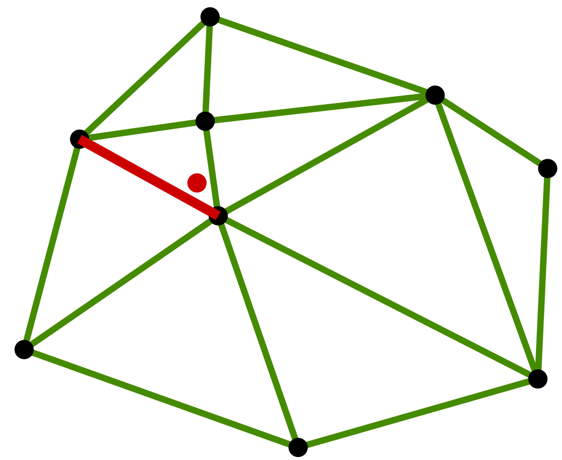
or . . .

Terrain Reconstruction from Elevation Data

Which triangulation should we use?



or



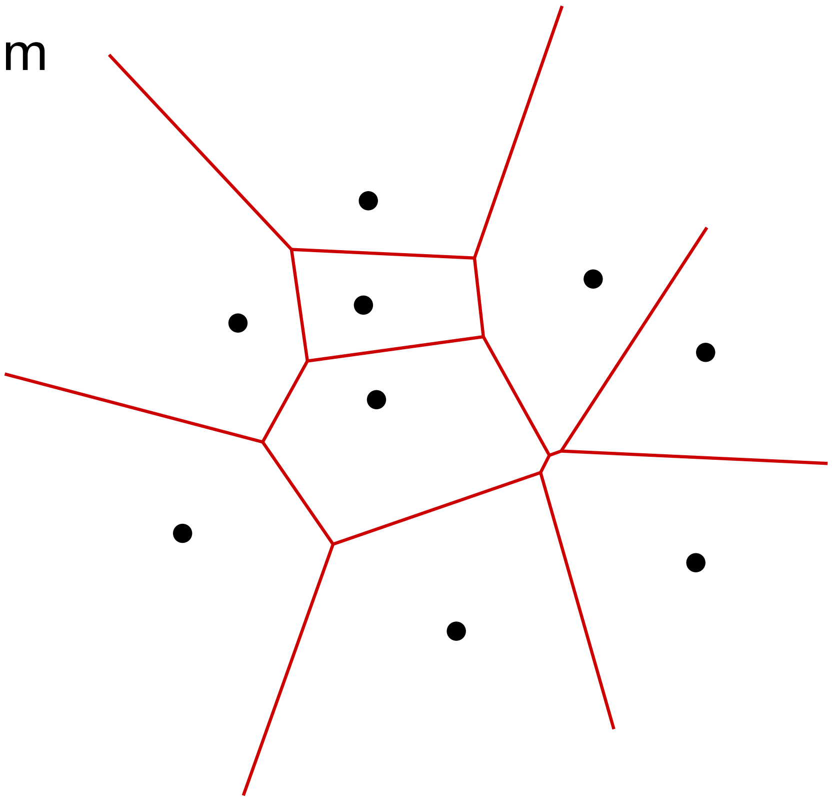
or . . .

long and thin triangles are bad $\Longrightarrow$ try to avoid small angles

Algorithmic problem: How can we quickly compute a triangulation that maximizes the minimum angle?

Terrain Reconstruction from Elevation Data

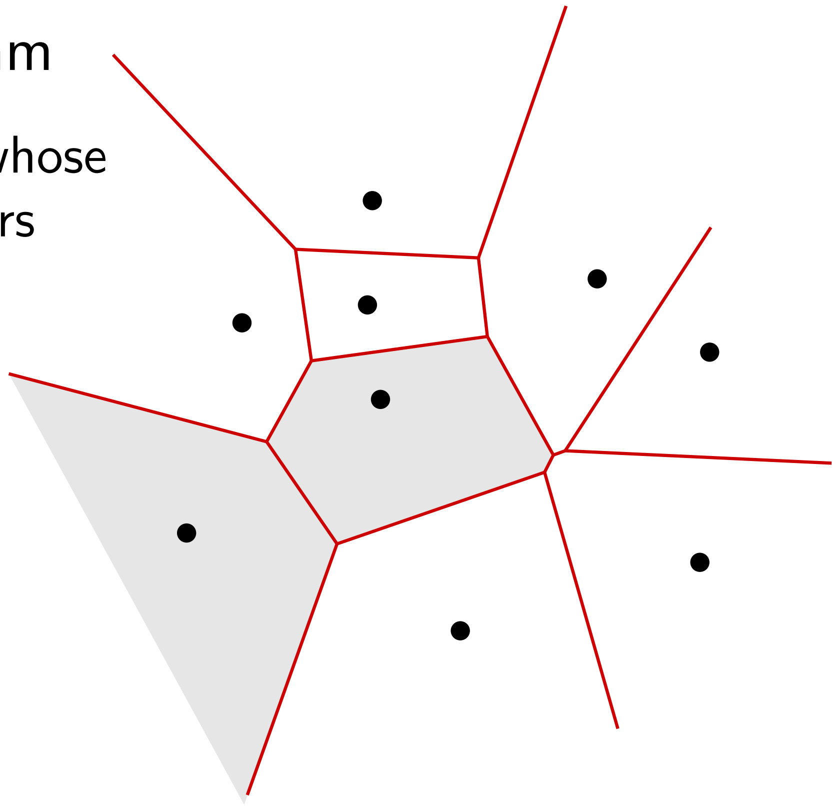
Voronoi diagram



Terrain Reconstruction from Elevation Data

Voronoi diagram

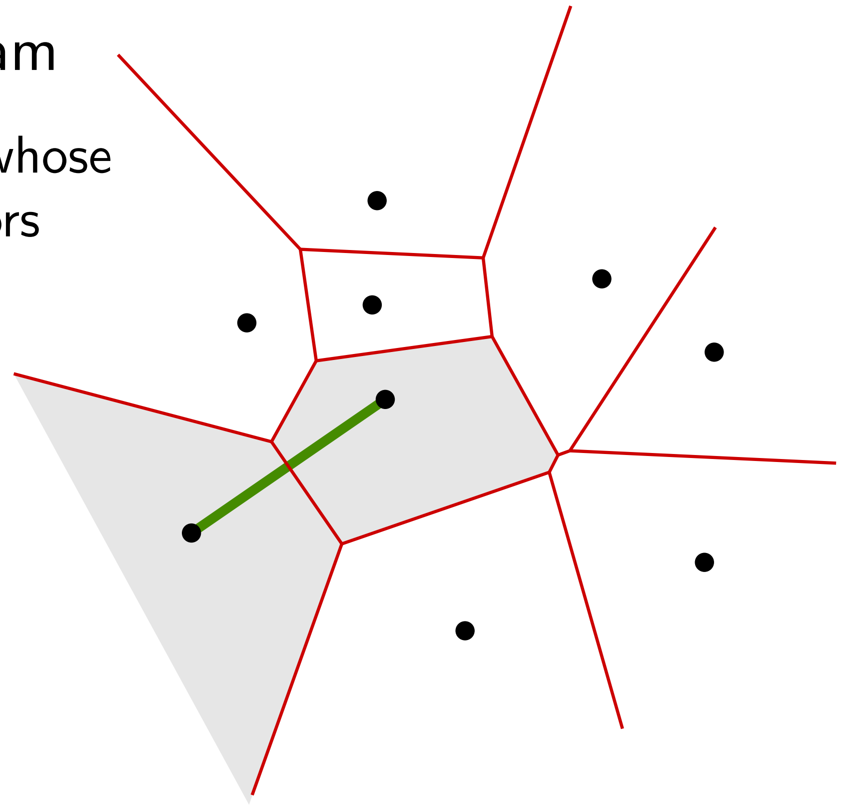
connect points whose
cells are neighbors



Terrain Reconstruction from Elevation Data

Voronoi diagram

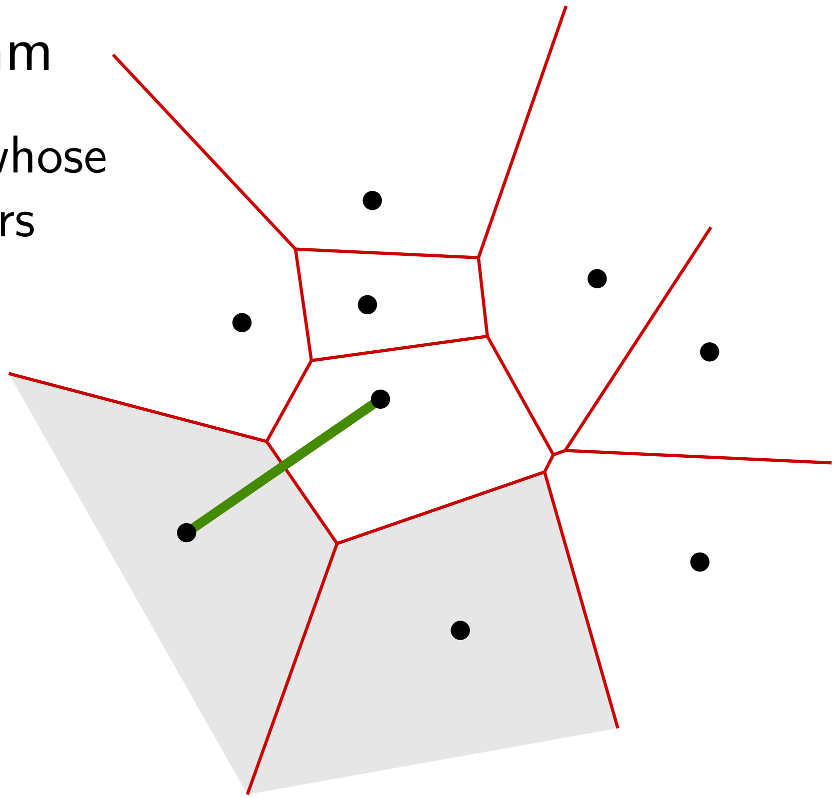
connect points whose cells are neighbors



Terrain Reconstruction from Elevation Data

Voronoi diagram

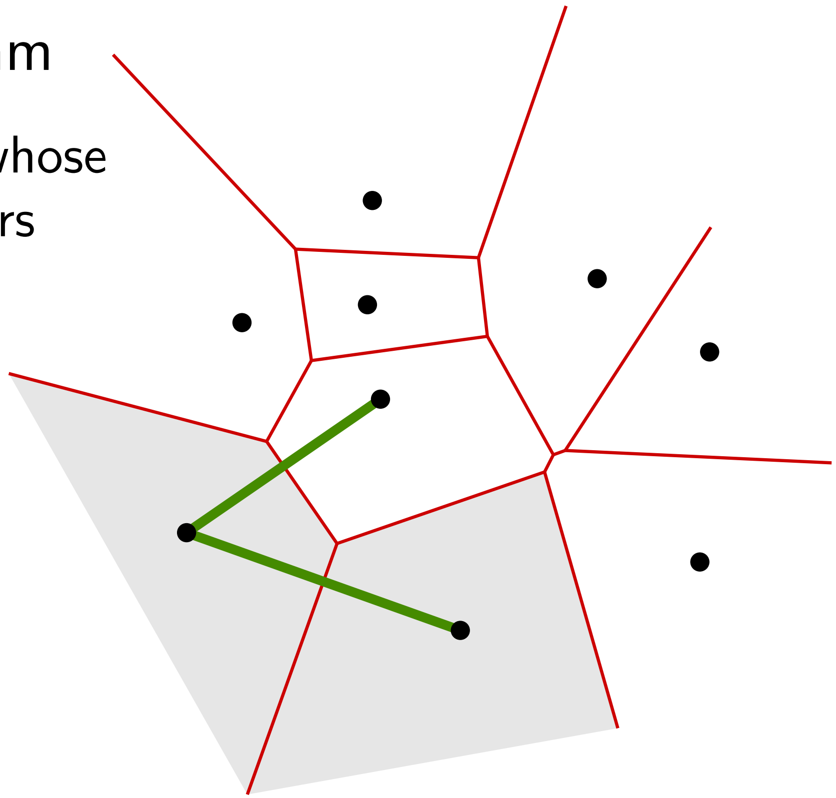
connect points whose
cells are neighbors



Terrain Reconstruction from Elevation Data

Voronoi diagram

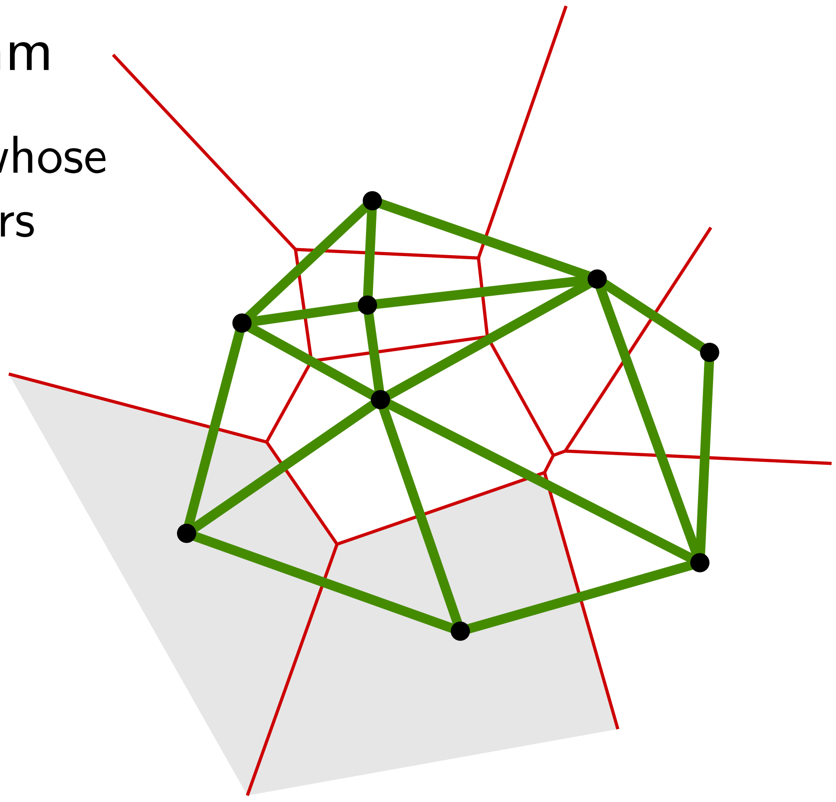
connect points whose
cells are neighbors



Terrain Reconstruction from Elevation Data

Voronoi diagram

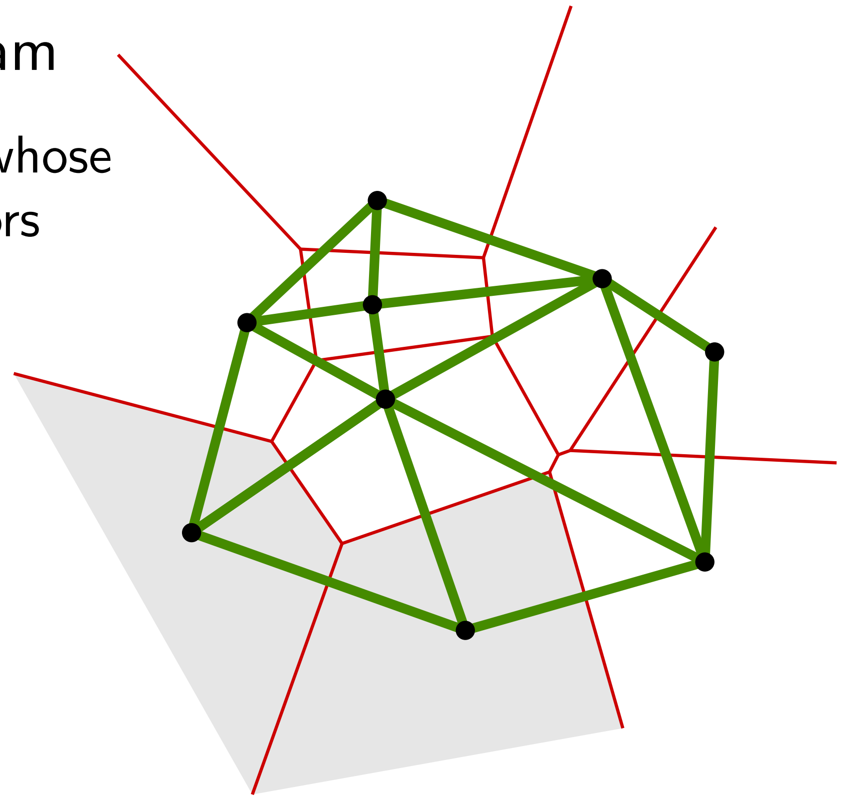
connect points whose
cells are neighbors



Terrain Reconstruction from Elevation Data

Voronoi diagram

connect points whose cells are neighbors



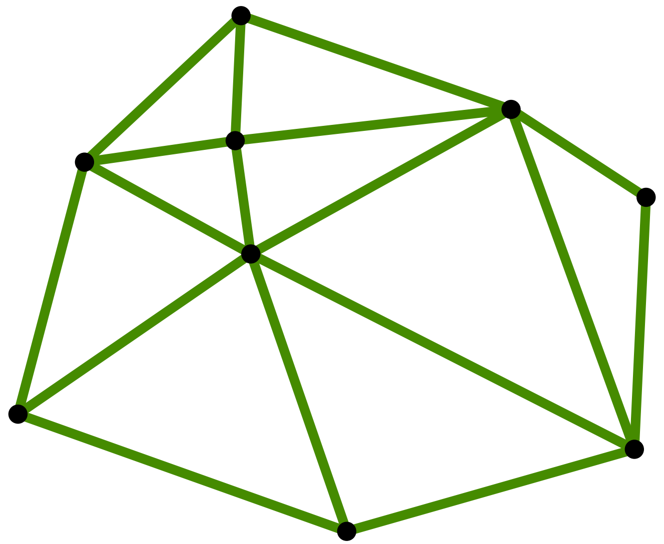
Delaunay triangulation:

triangulation that maximizes the minimum angle!

Boris Delaunay (1890 -1980)

Computing the Delaunay Triangulation

Computing the Delaunay Triangulation

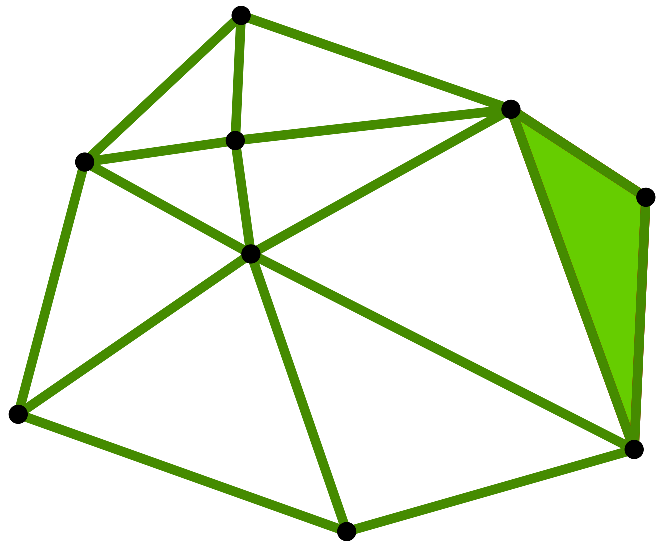


$\Delta(p, q, r)$ is in Delaunay triangulation

$$\Longleftrightarrow$$

Circle(p, q, r) contains no other point

Computing the Delaunay Triangulation

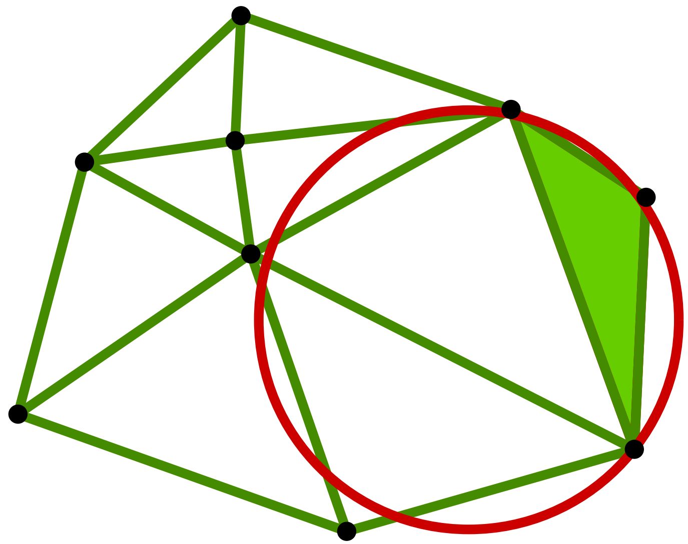


$\Delta(p, q, r)$ is in Delaunay triangulation

$$\Longleftrightarrow$$

Circle(p, q, r) contains no other point

Computing the Delaunay Triangulation

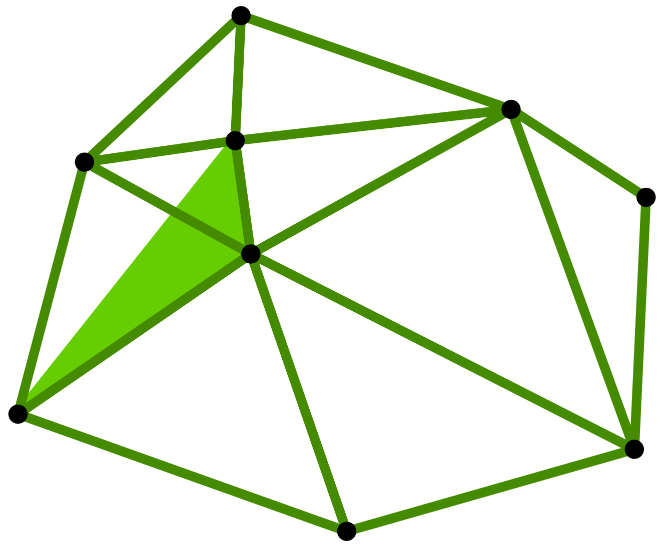


$\Delta(p, q, r)$ is in Delaunay triangulation

$\iff$

Circle(p, q, r) contains no other point

Computing the Delaunay Triangulation

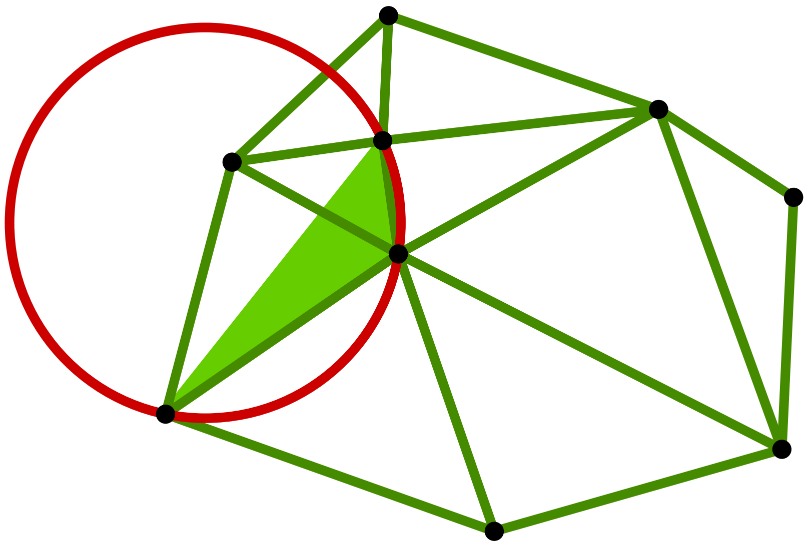


$\Delta(p, q, r)$ is in Delaunay triangulation

$$\Longleftrightarrow$$

Circle(p, q, r) contains no other point

Computing the Delaunay Triangulation

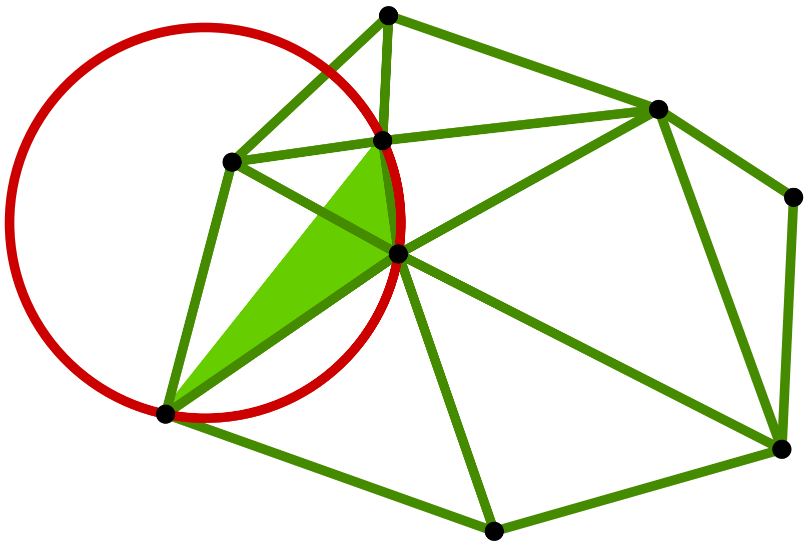


$\Delta(p, q, r)$ is in Delaunay triangulation

$\iff$

Circle(p, q, r) contains no other point

Computing the Delaunay Triangulation



$\Delta(p, q, r)$ is in Delaunay triangulation

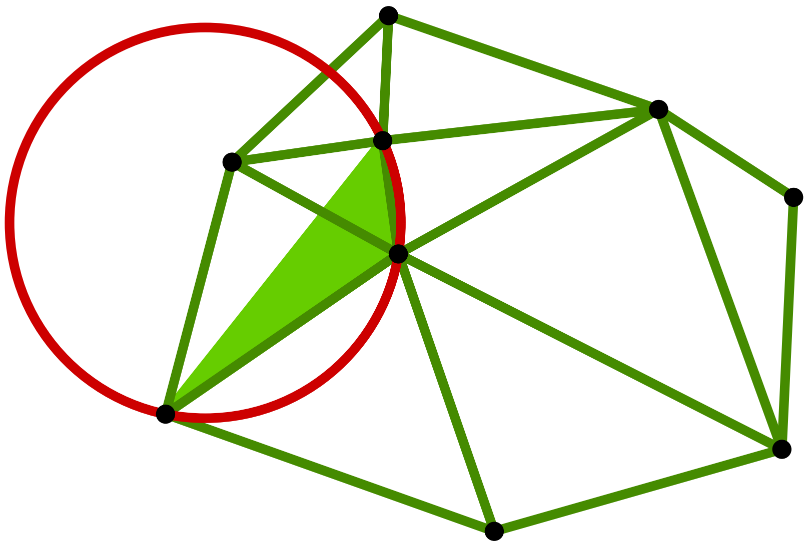
$\Longleftrightarrow$

Circle(p, q, r) contains no other point

DELAUNAY-ALGORITHM(S)

1: $\mathcal{T} \leftarrow \emptyset$
2: **for** every triple of points p, q, r from S **do**
3:     **if** all other points from S lie outside Circle(p, q, r) **then**
4:         Add $\Delta(p, q, r)$ to $\mathcal{T}$
5: **return** $\mathcal{T}$

Computing the Delaunay Triangulation



$\Delta(p, q, r)$ is in Delaunay triangulation

$\Longleftrightarrow$

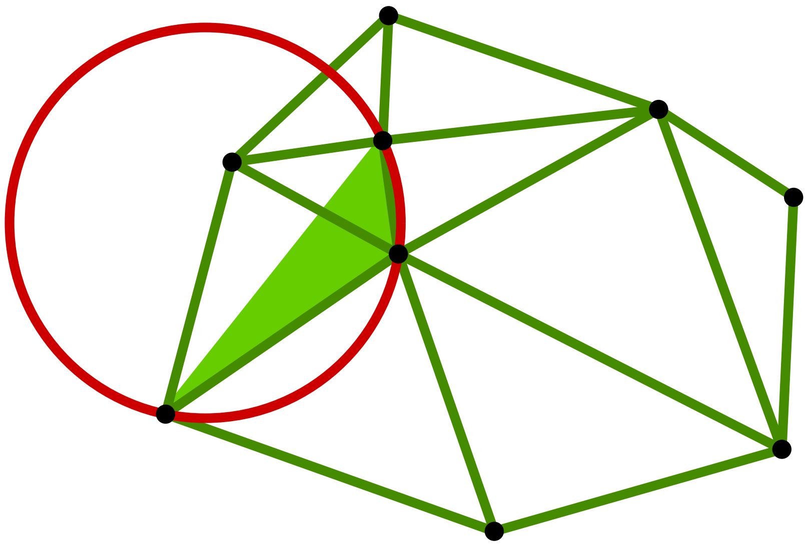
Circle(p, q, r) contains no other point

$\textsc{Delaunay-Algorithm}(S)$

1: $\mathcal{T} \leftarrow \emptyset$
2: **for** every triple of points p, q, r from S **do**
3:     **if** all other points from S lie outside Circle(p, q, r) **then**
4:         Add $\Delta(p, q, r)$ to $\mathcal{T}$
5: **return** $\mathcal{T}$

Running time:

Computing the Delaunay Triangulation



$\Delta(p,q,r)$ is in Delaunay triangulation

$\Longleftrightarrow$

Circle(p,q,r) contains no other point

$\textsc{Delaunay-Algorithm}(S)$

1: $\mathcal{T} \leftarrow \emptyset$
2: **for** every triple of points p,q,r from S **do**
3: **if** all other points from S lie outside Circle(p,q,r) **then**
4: Add $\Delta(p,q,r)$ to $\mathcal{T}$
5: **return** $\mathcal{T}$

Running time: $O(n^4)$

Computing the Delaunay Triangulation by RIC

Exercise

Apply the RIC framework to develop a randomized algorithm to compute the Delaunay triangulation, and analyze its running time.

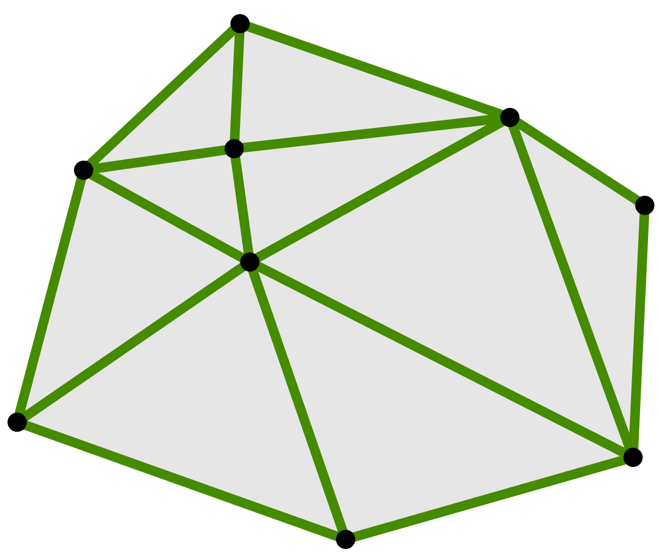
Fact: The number of triangles in the Delaunay triangulation of a set S of n points in the plane is at most $2n - 5$.

Computing the Delaunay Triangulation by RIC

Exercise

Apply the RIC framework to develop a randomized algorithm to compute the Delaunay triangulation, and analyze its running time.

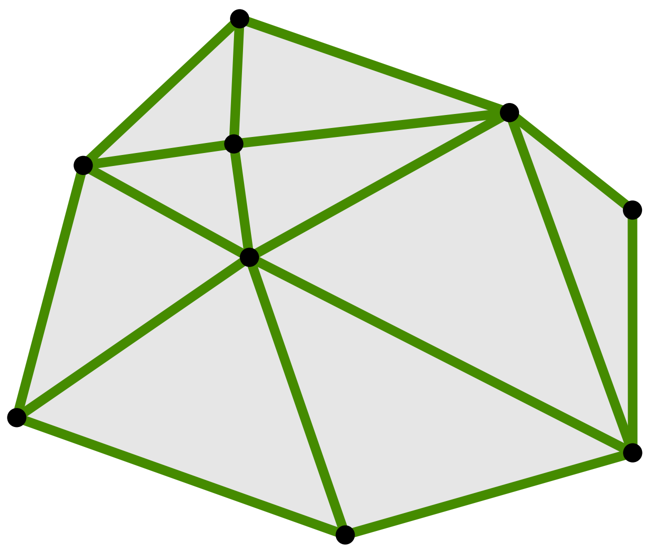
Fact: The number of triangles in the Delaunay triangulation of a set S of n points in the plane is at most $2n-5$.



The framework

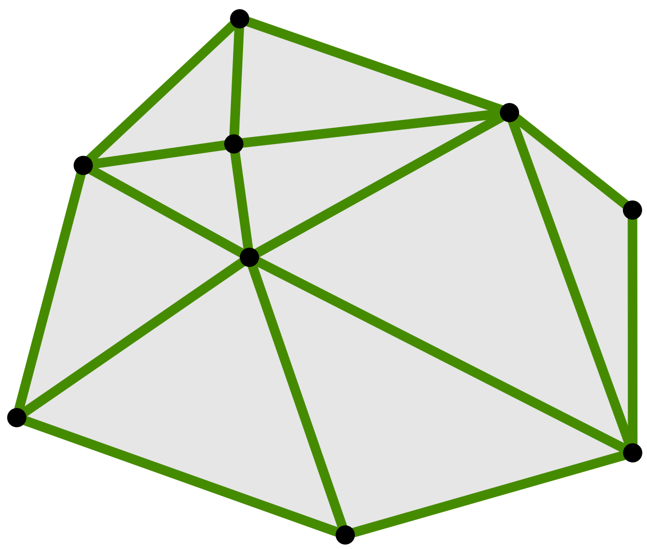
- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC



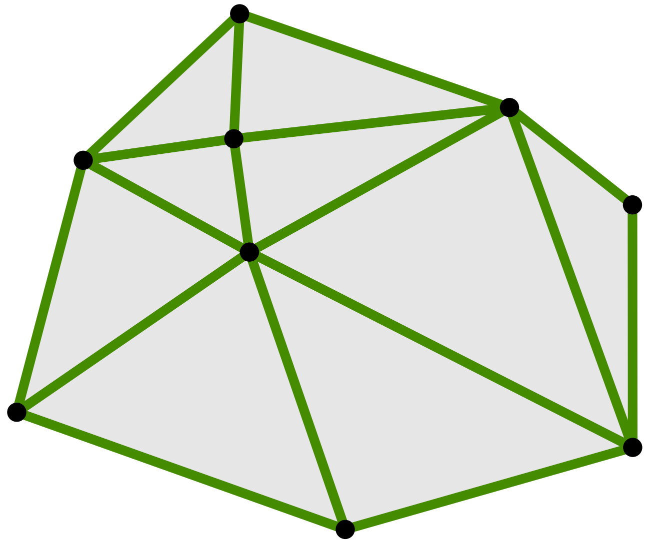
- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\mathrm{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC



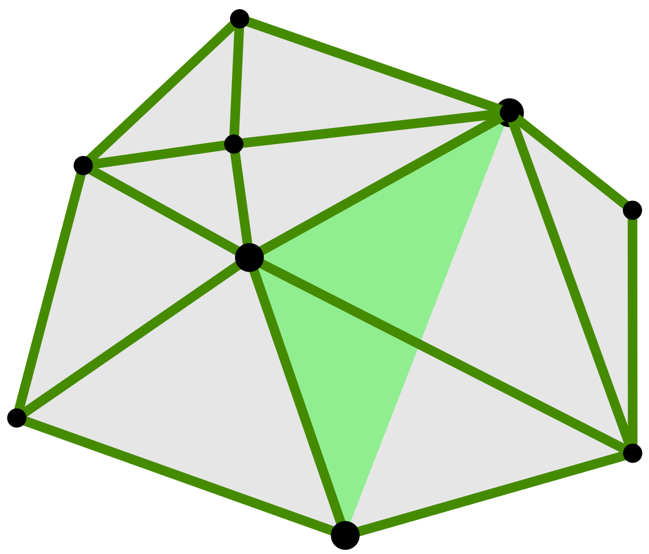
- S = set of n input points
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC



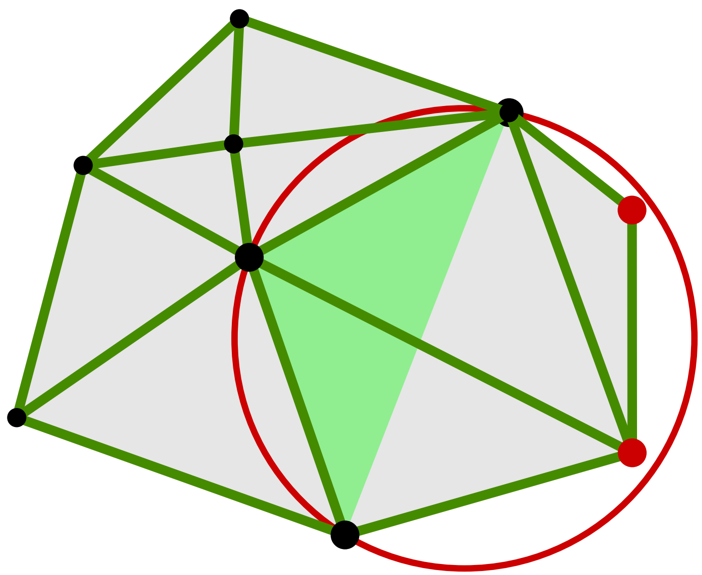
- S = set of n input points
- $\mathcal{C}(S)$ = all possible triangles defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC



- S = set of n input points
- $\mathcal{C}(S)$ = all possible triangles defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC



- S = set of n input points
- $\mathcal{C}(S)$ = all possible triangles defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
    all points contained in circumcircle of Δ
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Computing the Delaunay Triangulation by RIC

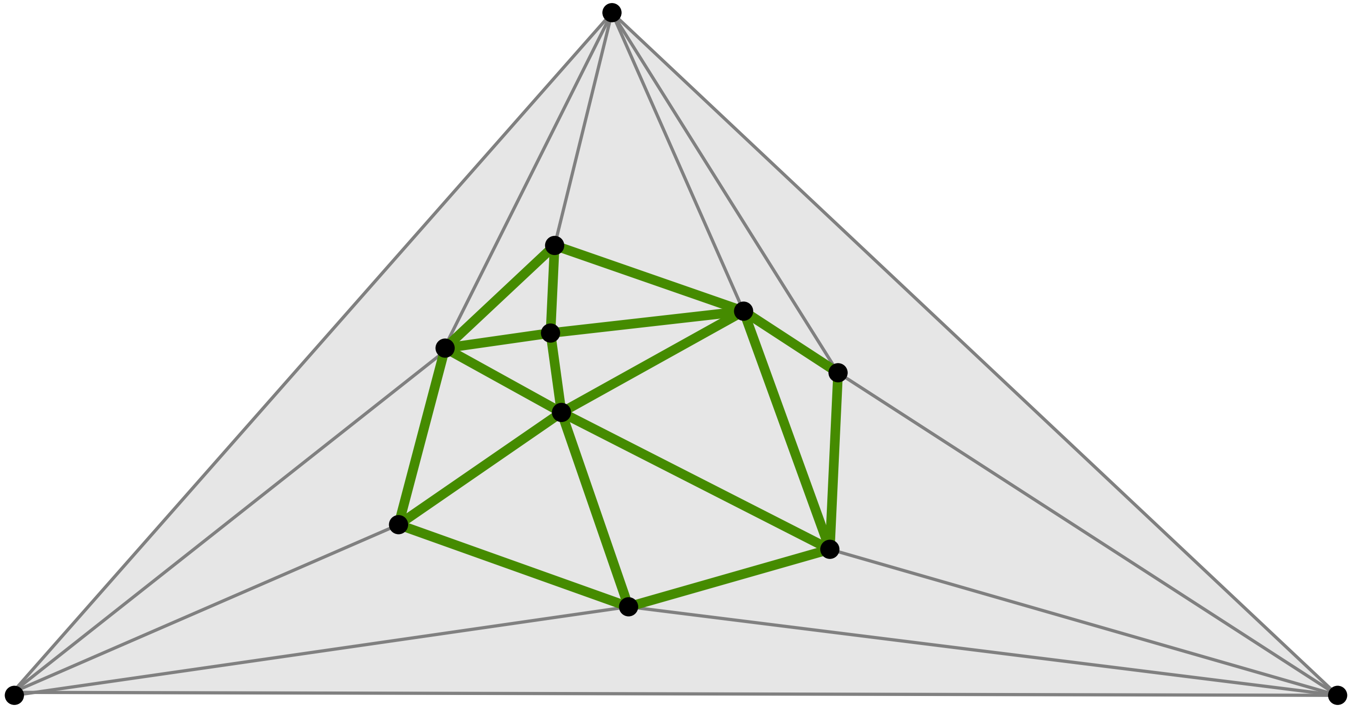
RIC-DELAUNAY(S)

```
Compute a random permutation x_1, ..., x_n of the objects in S.
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove all configurations from C_act that become inactive
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```

Computing the Delaunay Triangulation by RIC

RIC-Delaunay(S)

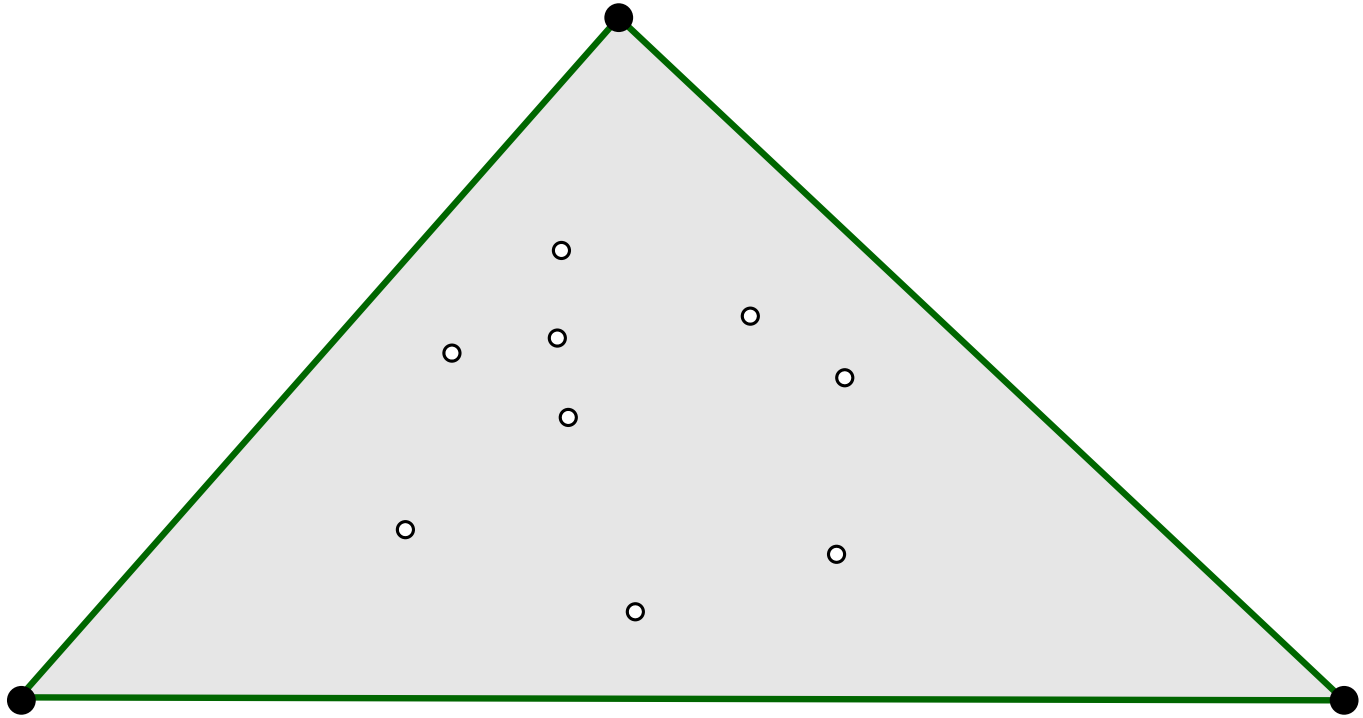
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\emptyset\}$
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\mathrm{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\mathrm{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

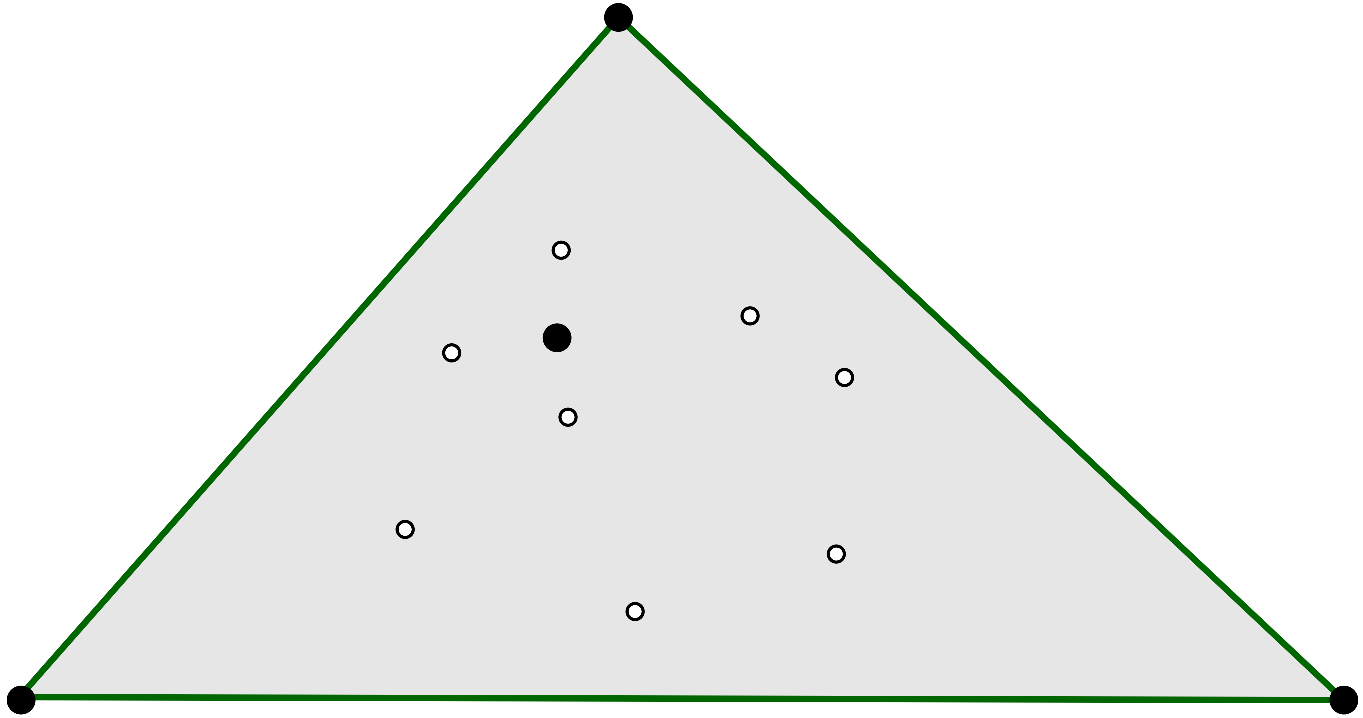
```
Compute a random permutation x_1, ..., x_n of the objects in S.
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove all configurations from C_act that become inactive
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

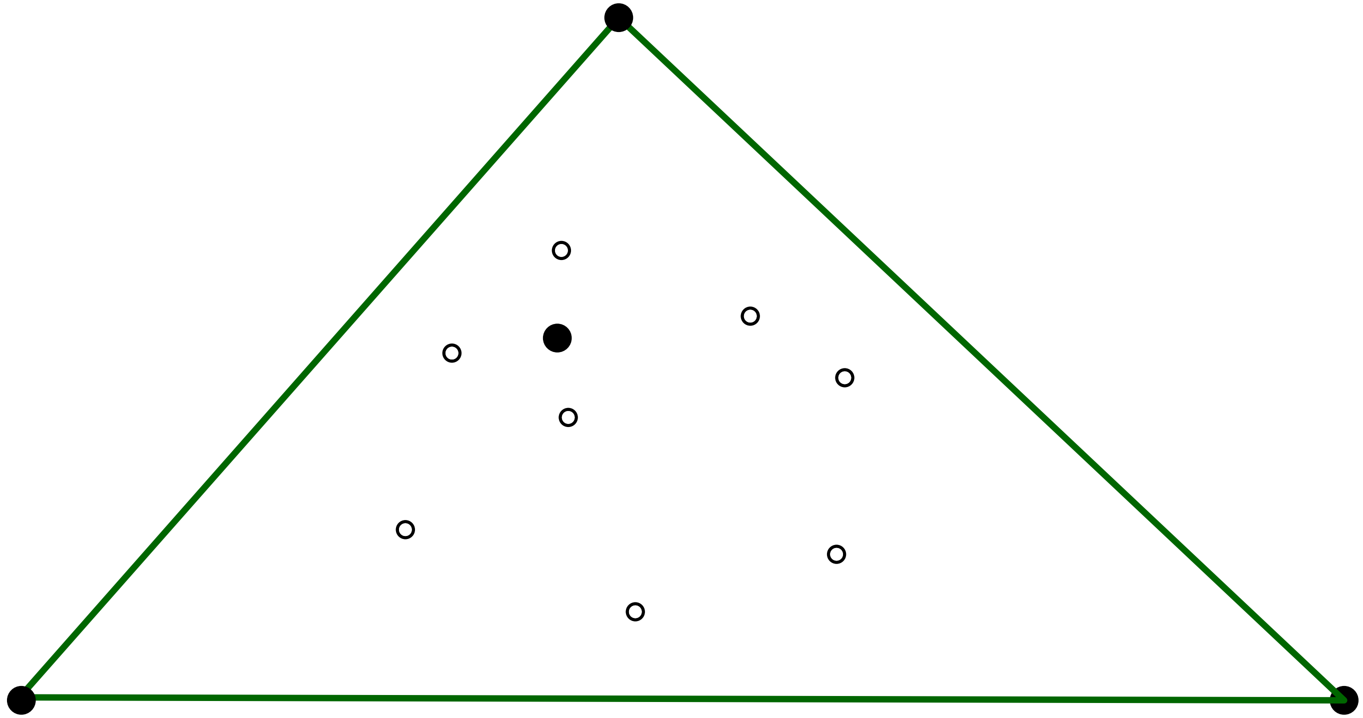
```
Compute a random permutation x_1, ..., x_n of the objects in S.
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove all configurations from C_act that become inactive
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

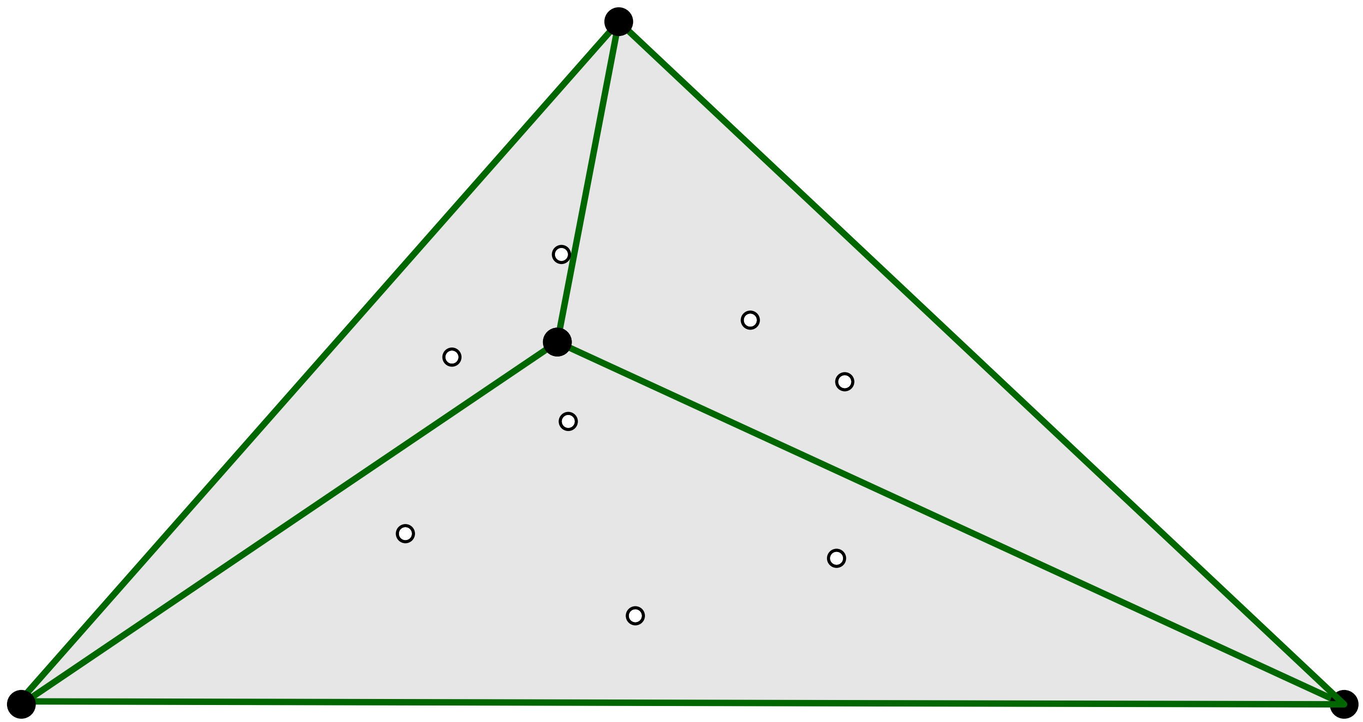
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\mathrm{act}} \leftarrow \{$active configurations with respect to $\emptyset\}$
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\mathrm{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\mathrm{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

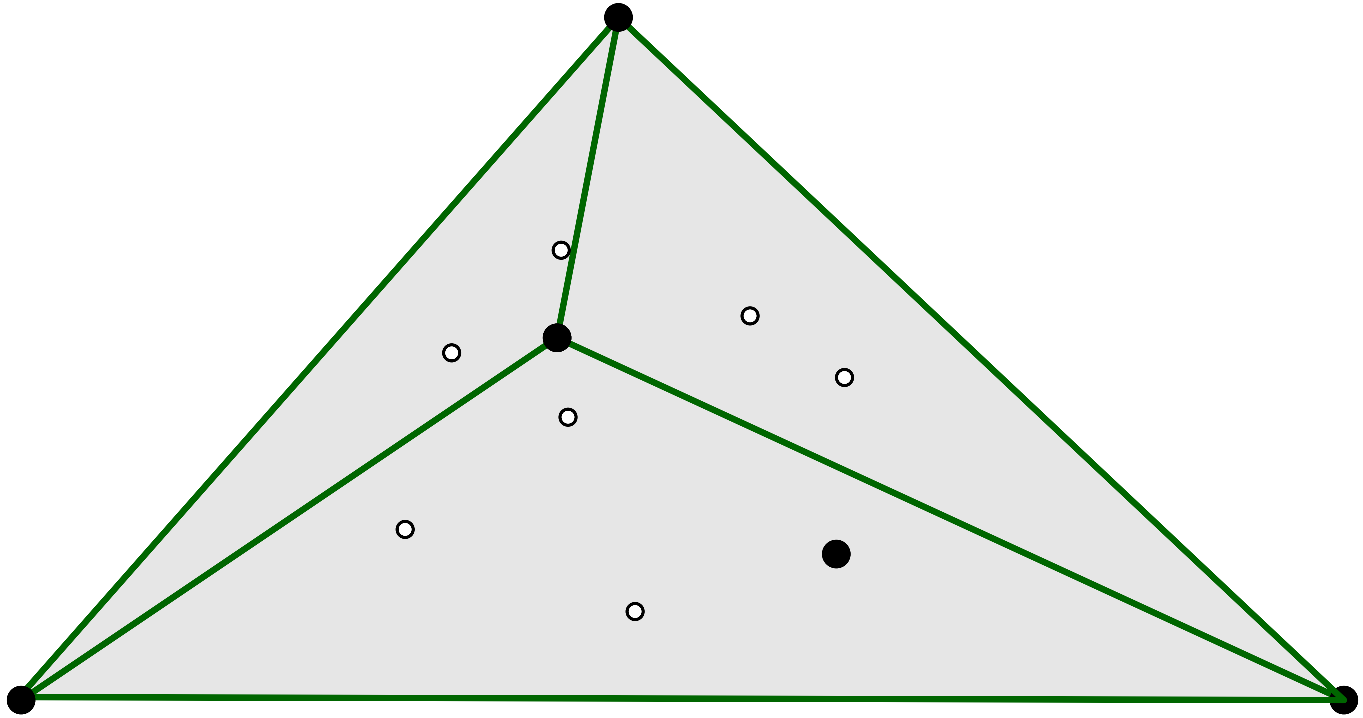
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\mathrm{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\mathrm{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\mathrm{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

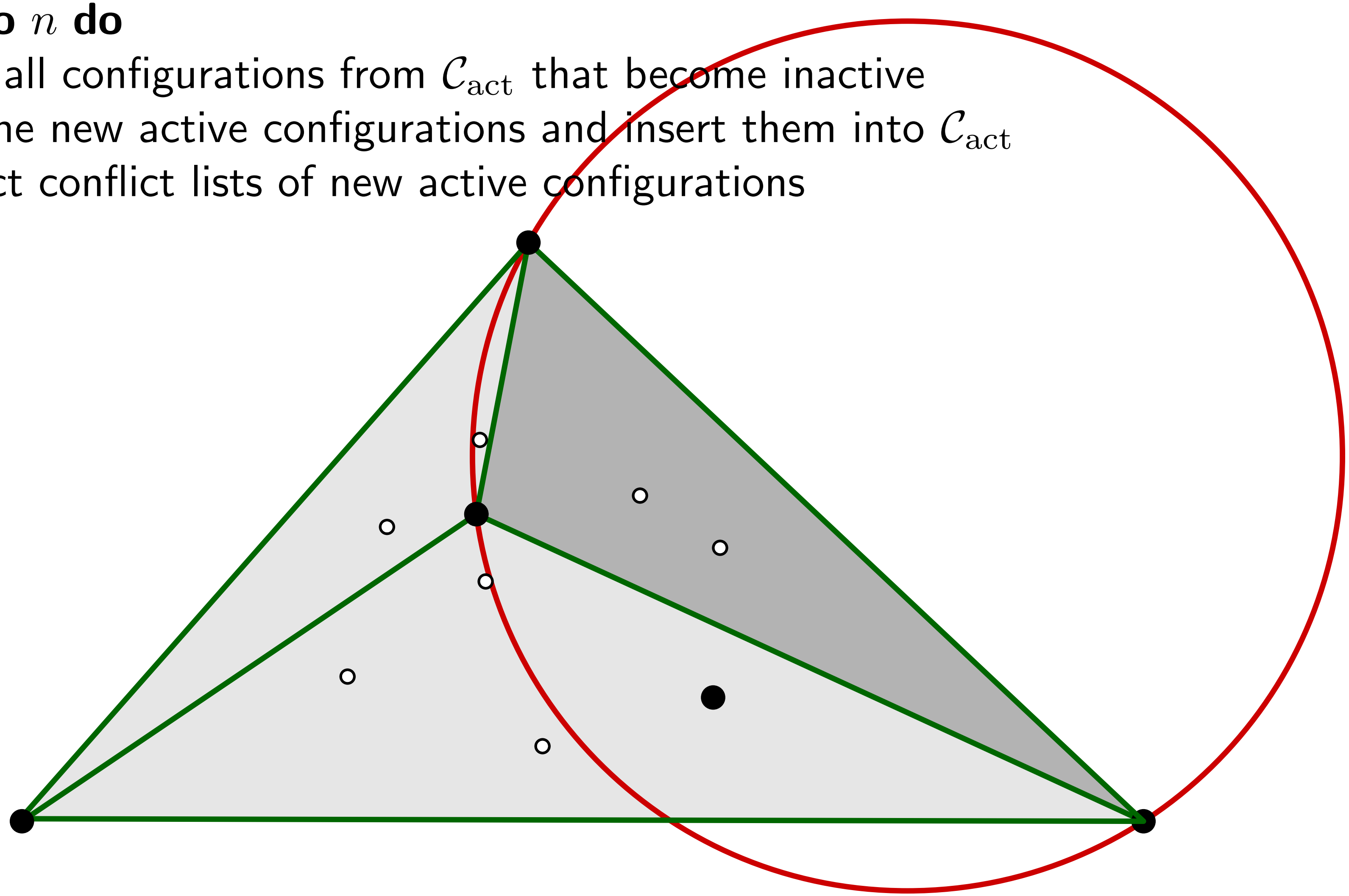
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\text{act}} \leftarrow \{$active configurations with respect to $\emptyset\}$
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\text{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

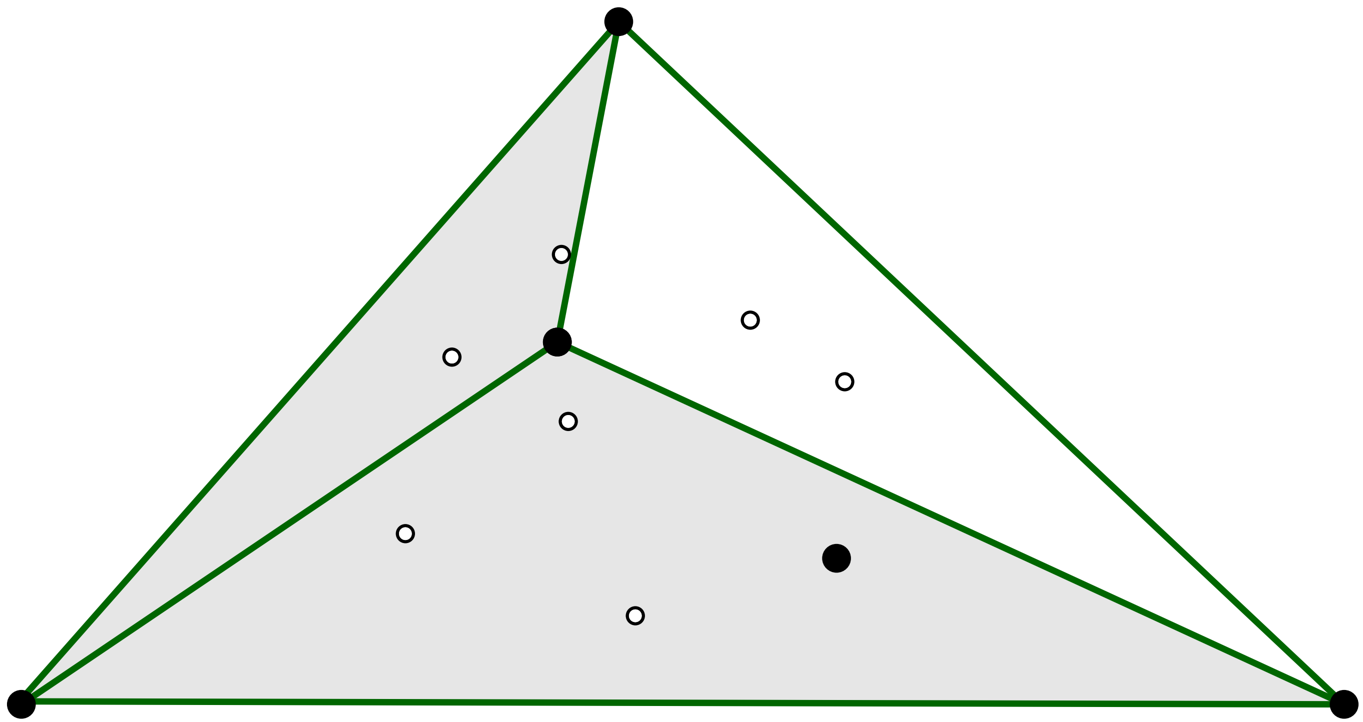
```
Compute a random permutation x_1, ..., x_n of the objects in S.
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove all configurations from C_act that become inactive
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

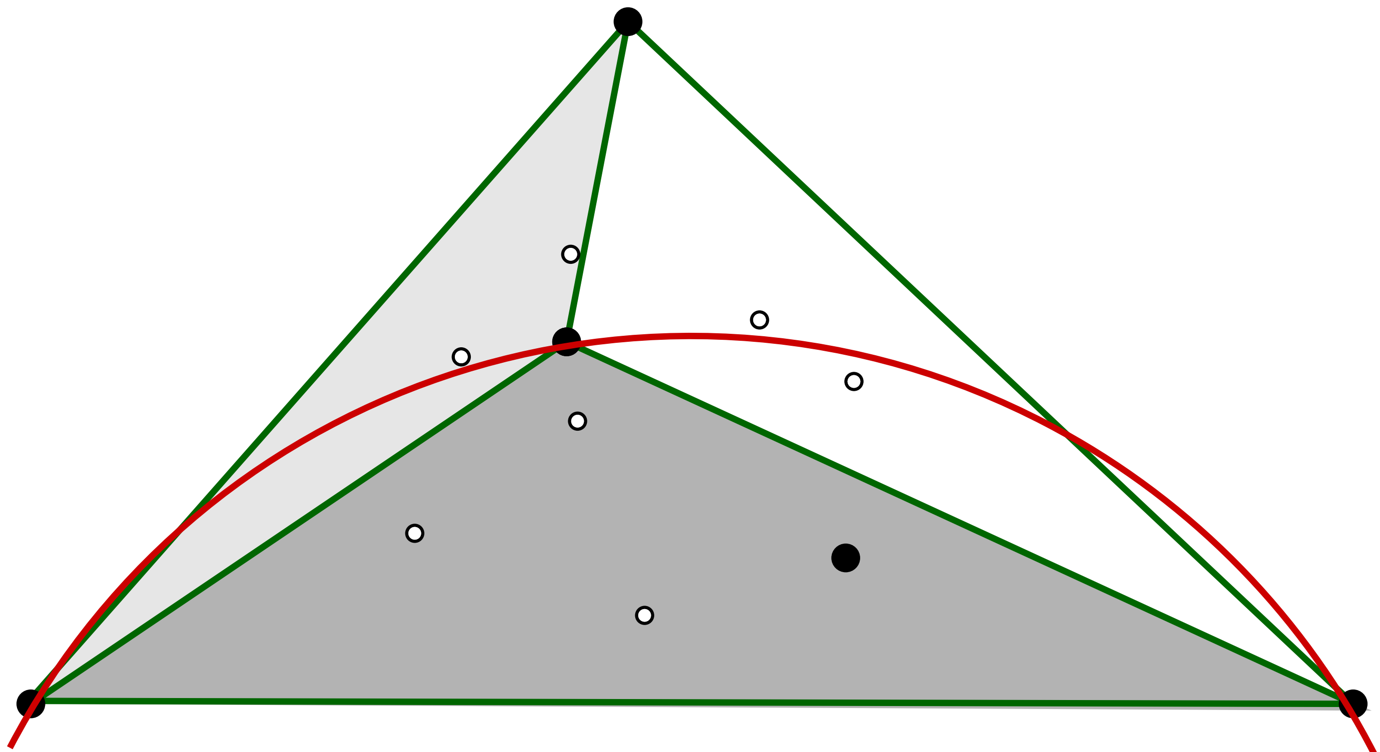
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\mathrm{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\mathrm{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\mathrm{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

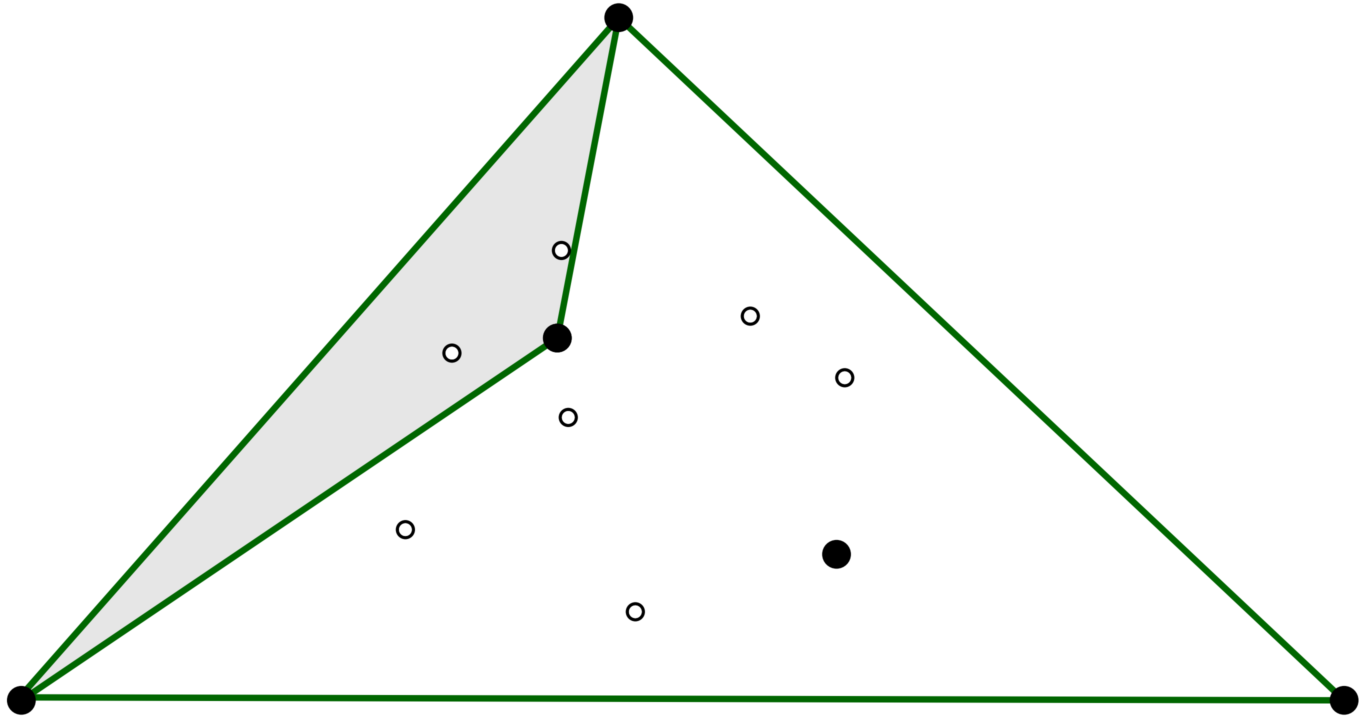
```
Compute a random permutation x_1, ..., x_n of the objects in S.
C_act ← {active configurations with respect to ∅}
Intitialize conflict lists of configurations Δ ∈ C_act
for j ← 1 to n do
    Remove all configurations from C_act that become inactive
    Determine new active configurations and insert them into C_act
    Construct conflict lists of new active configurations
return C_act
```



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

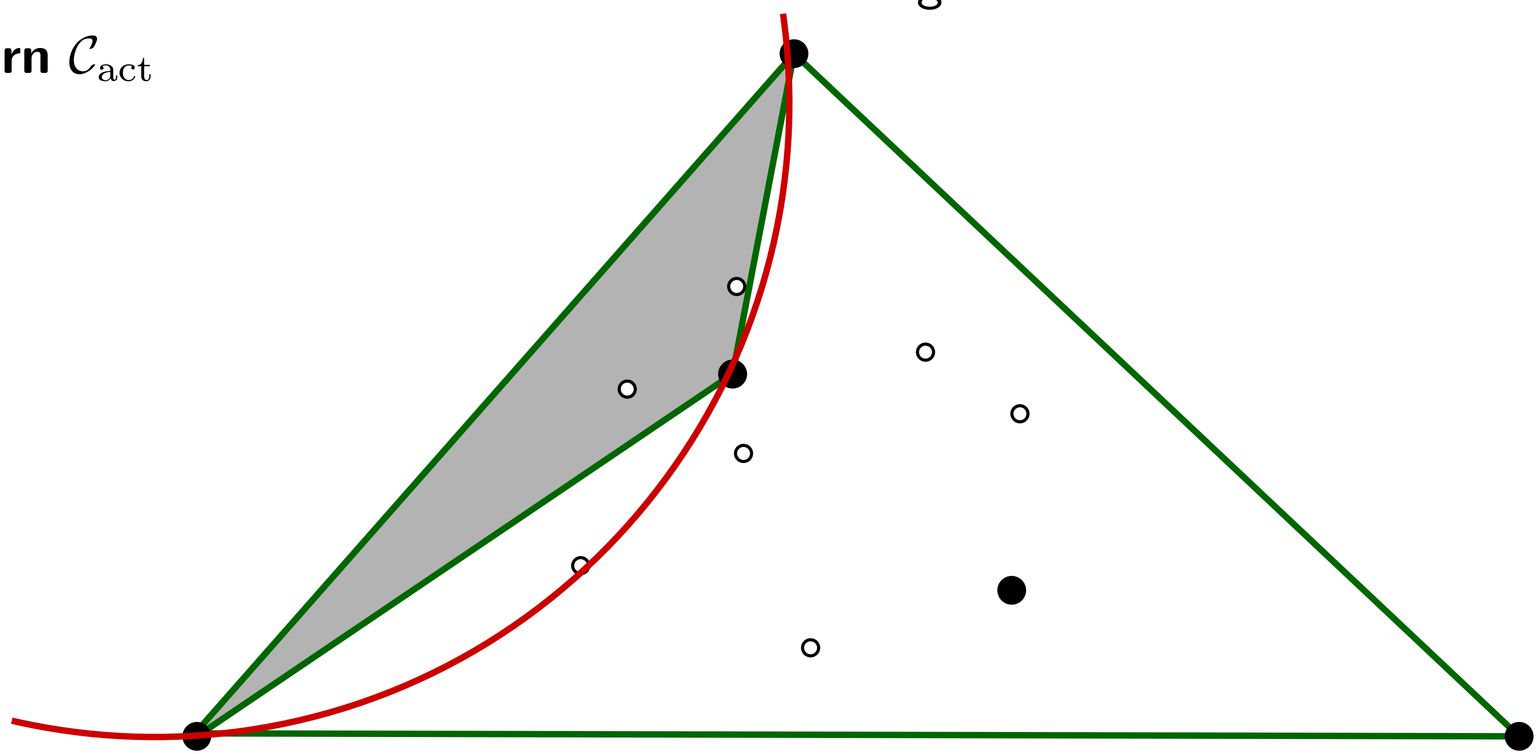
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\text{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

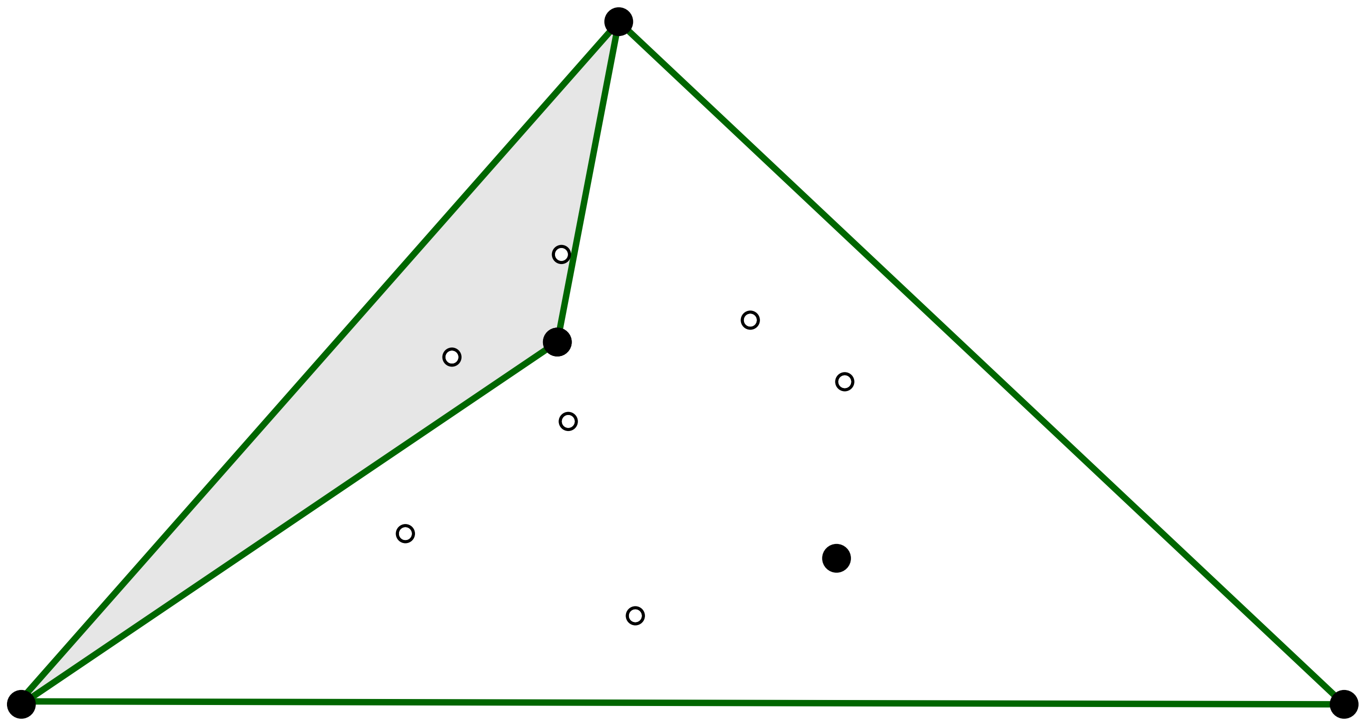
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\text{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

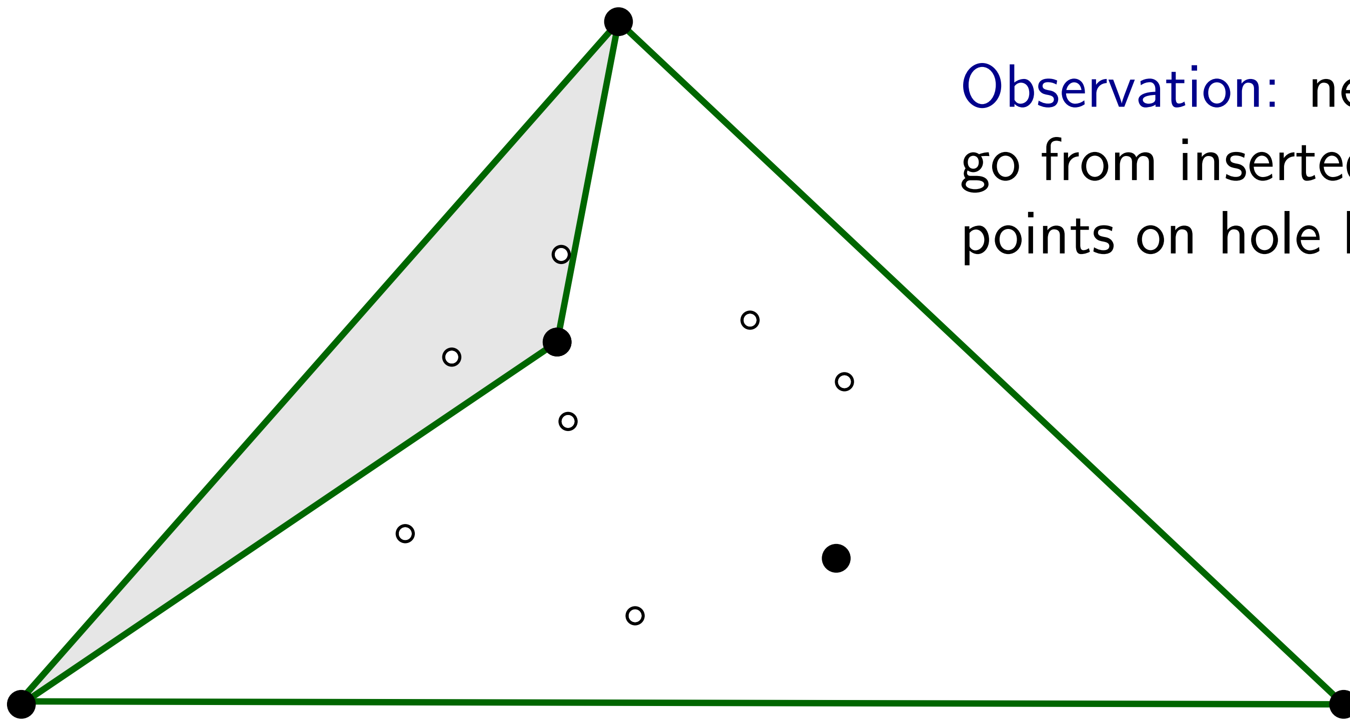
1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\text{act}}$



Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
$\mathcal{C}_{\mathrm{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\mathrm{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove all configurations from $\mathcal{C}_{\mathrm{act}}$ that become inactive
    Determine new active configurations and insert them into $\mathcal{C}_{\mathrm{act}}$
    Construct conflict lists of new active configurations
return $\mathcal{C}_{\mathrm{act}}$

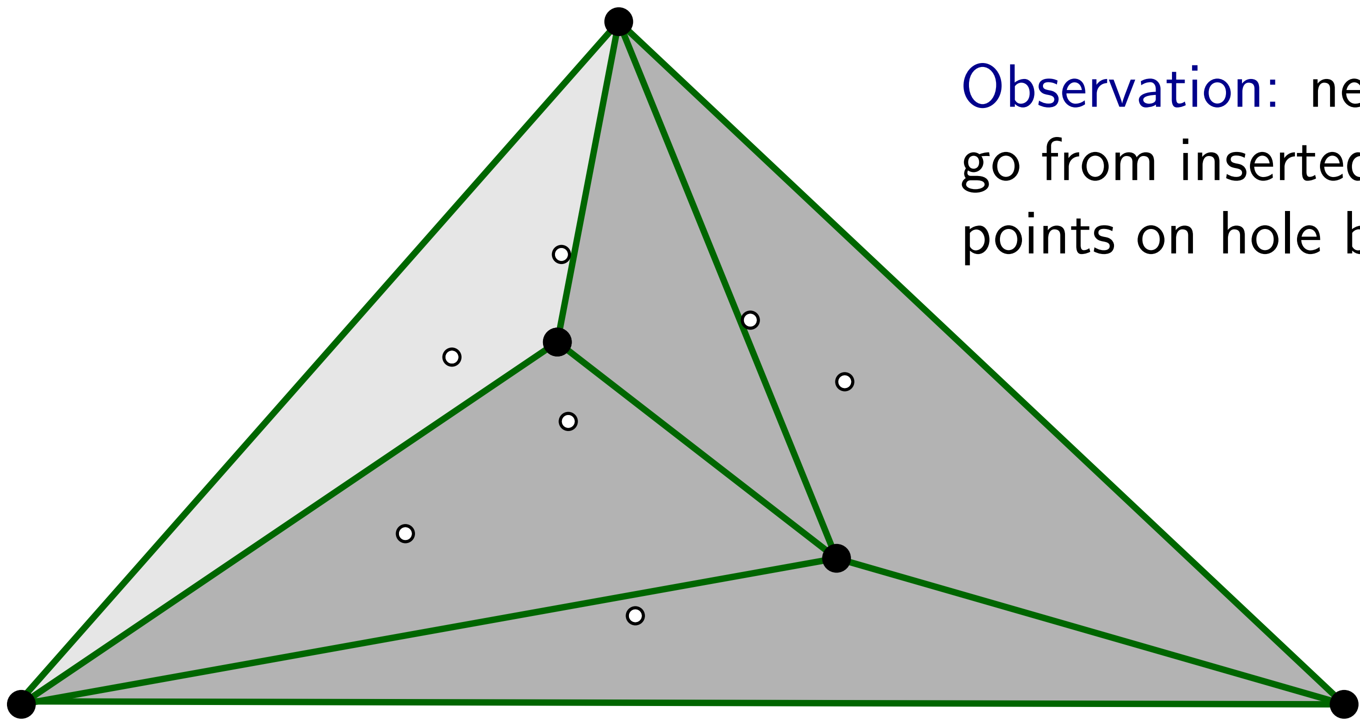


Observation: new edges go from inserted point to points on hole boundary

Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

1: Compute a random permutation $x_1, \ldots, x_n$ of the objects in S.
2: $\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
3: Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
4: **for** $j \leftarrow 1$ **to** n **do**
5: Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
6: Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
7: Construct conflict lists of new active configurations
8: **return** $\mathcal{C}_{\text{act}}$

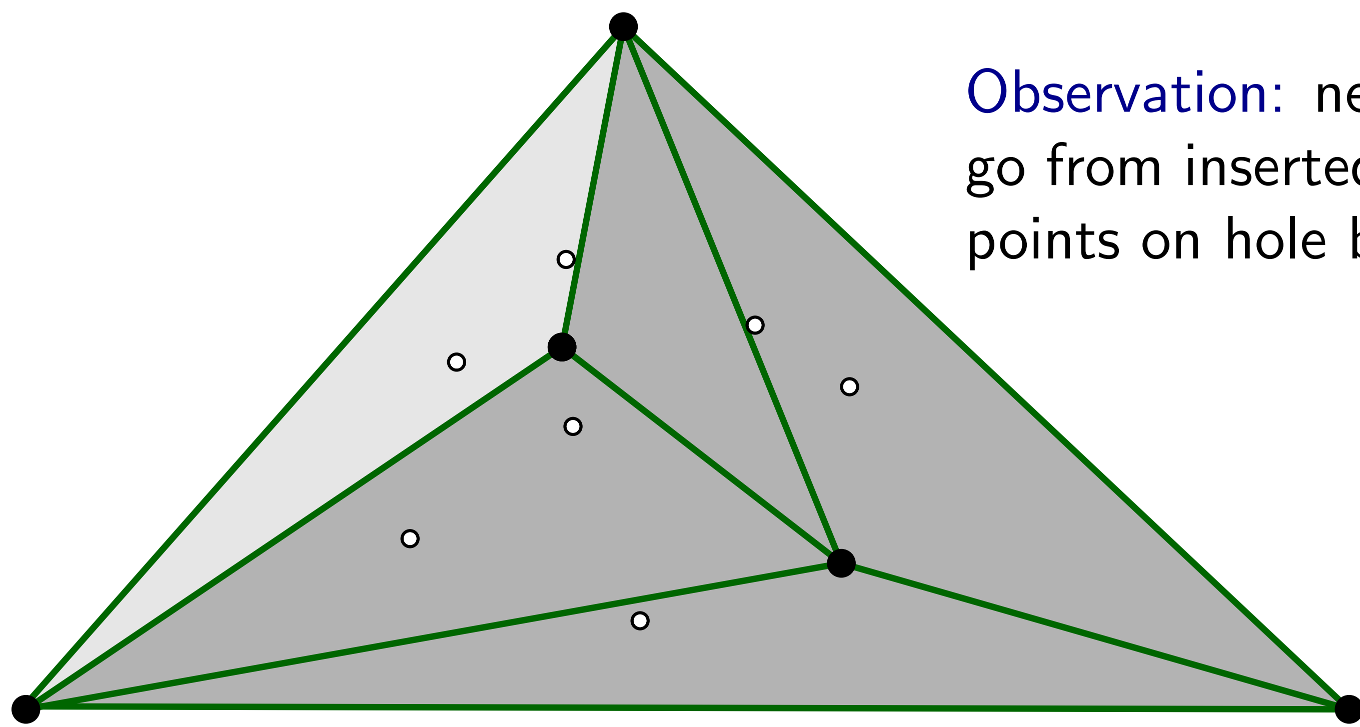


Observation: new edges go from inserted point to points on hole boundary

Computing the Delaunay Triangulation by RIC

RIC-DELAUNAY(S)

Compute a random permutation $x_1, \dots, x_n$ of the objects in S.
$\mathcal{C}_{\text{act}} \leftarrow$ {active configurations with respect to $\emptyset$}
Intitialize conflict lists of configurations $\Delta \in \mathcal{C}_{\text{act}}$
for $j \leftarrow 1$ **to** n **do**
    Remove all configurations from $\mathcal{C}_{\text{act}}$ that become inactive
    Determine new active configurations and insert them into $\mathcal{C}_{\text{act}}$
    Construct conflict lists of new active configurations   takes most time . . .
return $\mathcal{C}_{\text{act}}$



Observation: new edges go from inserted point to points on hole boundary

Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\,|\mathcal{C}_{\mathrm{act}}(S_j) \setminus \mathcal{C}_{\mathrm{act}}(S_{j-1})|\,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\mathrm{act}}(S_j)]}{j}\right)$

(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\,|\mathcal{C}_{\mathrm{act}}(S_j)|\,\right]\right)$

Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j) \setminus \mathcal{C}_{\mathrm{act}}(S_{j-1})| \,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\mathrm{act}}(S_j)]}{j}\right)$

(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\mathrm{act}}(S_j)| \,\right]\right)$

Delaunay triangulation in the plane:

size of $\mathcal{C}_{\mathrm{act}}(S_j) = \#$(triangles of Delaunay triangulation of j points) $= O(j)$

$\Longrightarrow$ total size of all conflict lists $= O(n \log n)$

Theorem. Let $S_j := \{x_1, \ldots, x_j\}$. Then

(i) $\mathrm{E}\left[\, |\mathcal{C}_{\text{act}}(S_j) \setminus \mathcal{C}_{\text{act}}(S_{j-1})| \,\right] = O\left(\frac{\mathrm{E}[\text{size of } \mathcal{C}_{\text{act}}(S_j)]}{j}\right)$

(ii) The total size of the conflict lists of the active configurations appearing over the course of the algorithm is $O\left(\sum_{j=1}^{n} \frac{n}{j^2} \cdot \mathrm{E}\left[\, |\mathcal{C}_{\text{act}}(S_j)| \,\right]\right)$

Delaunay triangulation in the plane:

size of $\mathcal{C}_{\text{act}}(S_j) = \#$(triangles of Delaunay triangulation of j points) $= O(j)$

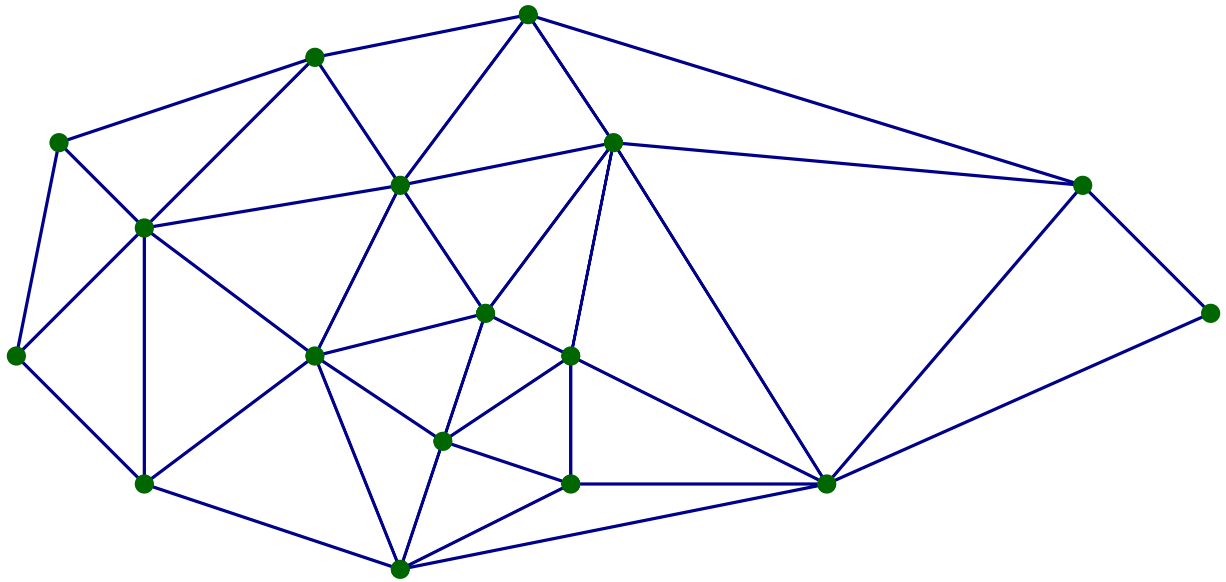
$\Longrightarrow$ total size of all conflict lists $= O(n \log n)$

Theorem. The Delaunay triangulation of a set of n points in the plane can be computed in $O(n \log n)$ expected time, using RIC.

Voronoi Diagrams and Delaunay Triangulations

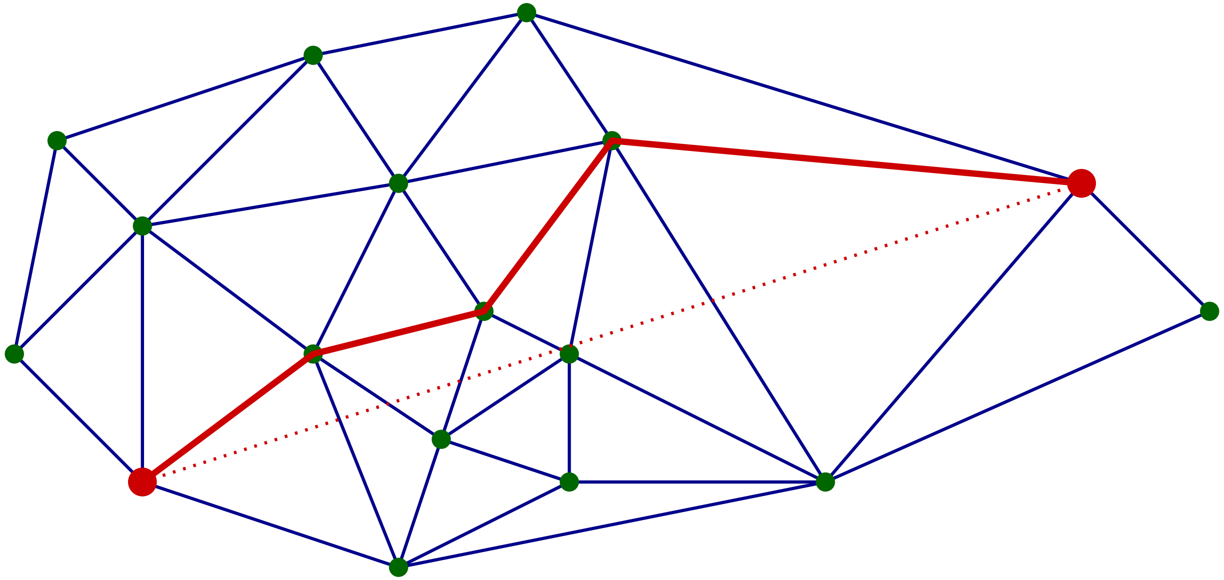
Fun Facts and Application

Voronoi Diagrams and Delaunay Triangulations: Fun Facts



dilation (= stretch factor = spanning ratio) of Delaunay triangulation is at most 1.998.

Voronoi Diagrams and Delaunay Triangulations: Fun Facts



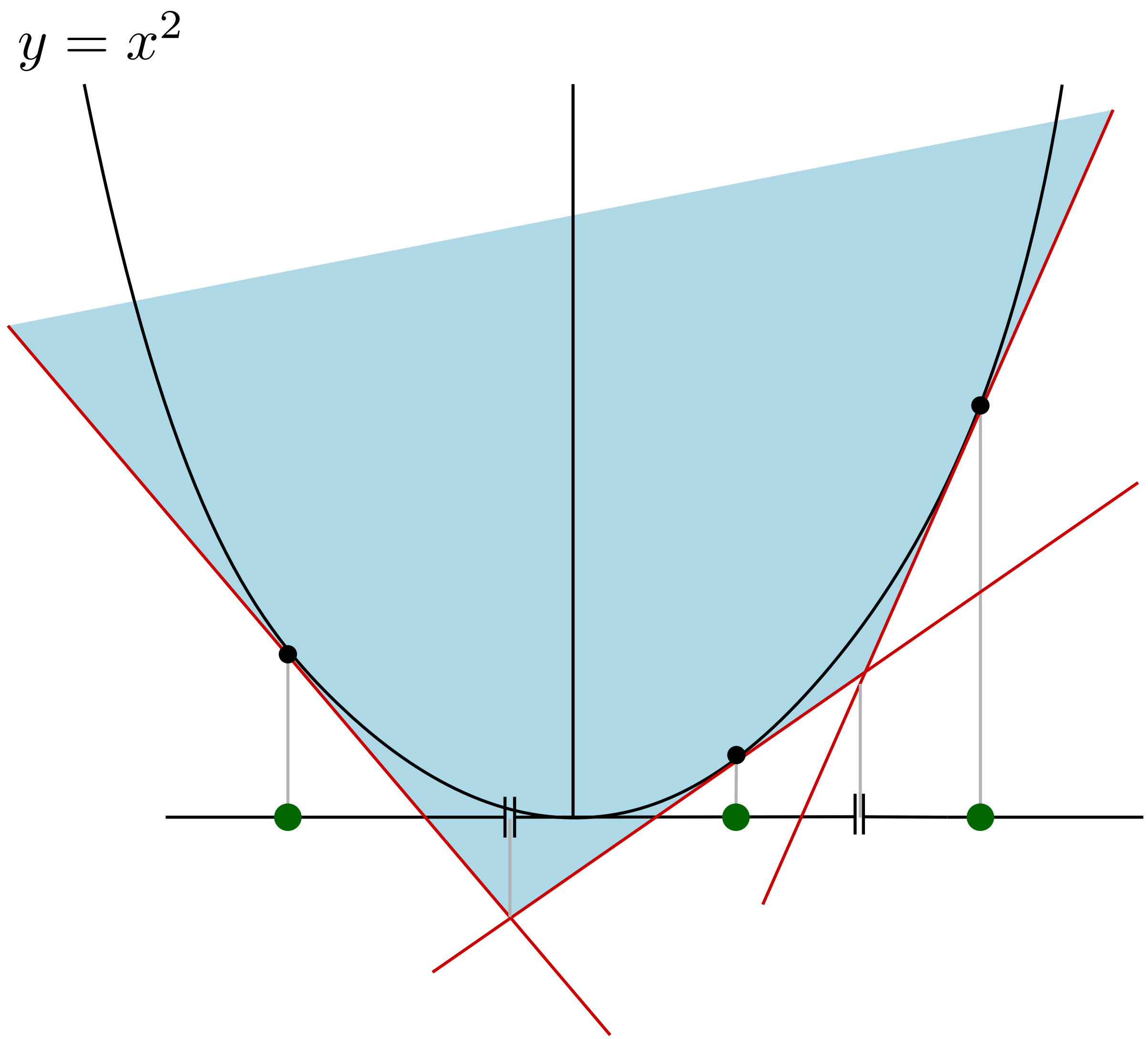
dilation (= stretch factor = spanning ratio) of Delaunay triangulation is at most 1.998.

Voronoi Diagrams and Delaunay Triangulations: Fun Facts

Voronoi diagram in $\mathbb{R}^d$ $\quad\equiv\quad$ half-space intersection in $\mathbb{R}^{d+1}$ $\quad\approx\quad$ convex hull in $\mathbb{R}^{d+1}$

Voronoi Diagrams and Delaunay Triangulations: Fun Facts

Voronoi diagram in $\mathbb{R}^d$ $\equiv$ half-space intersection in $\mathbb{R}^{d+1}$ $\approx$ convex hull in $\mathbb{R}^{d+1}$



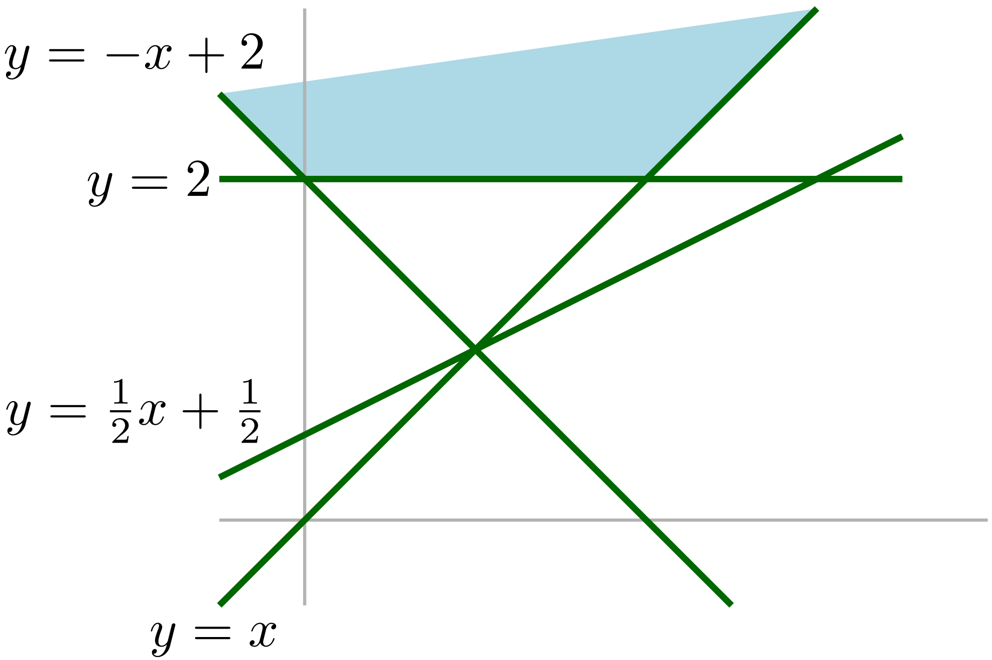
Voronoi Diagrams and Delaunay Triangulations: Fun Facts

Voronoi diagram in $\mathbb{R}^d$ $\equiv$ half-space intersection in $\mathbb{R}^{d+1}$ $\approx$ convex hull in $\mathbb{R}^{d+1}$

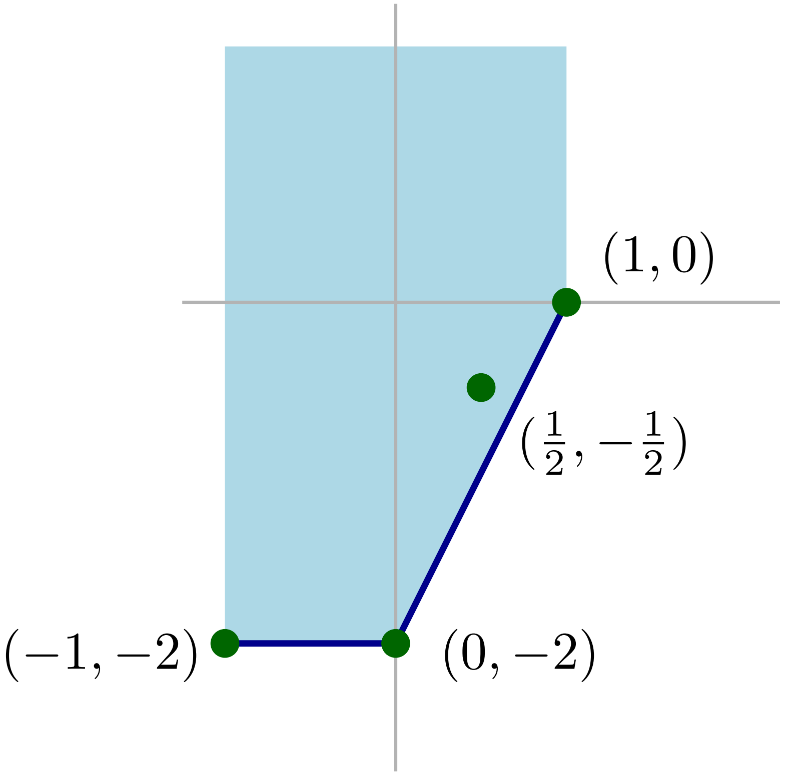
Voronoi Diagrams and Delaunay Triangulations: Fun Facts

Voronoi diagram in $\mathbb{R}^d$ $\equiv$ half-space intersection in $\mathbb{R}^{d+1}$ $\approx$ convex hull in $\mathbb{R}^{d+1}$

map line $y = ax + b$ to point $(a, -b)$

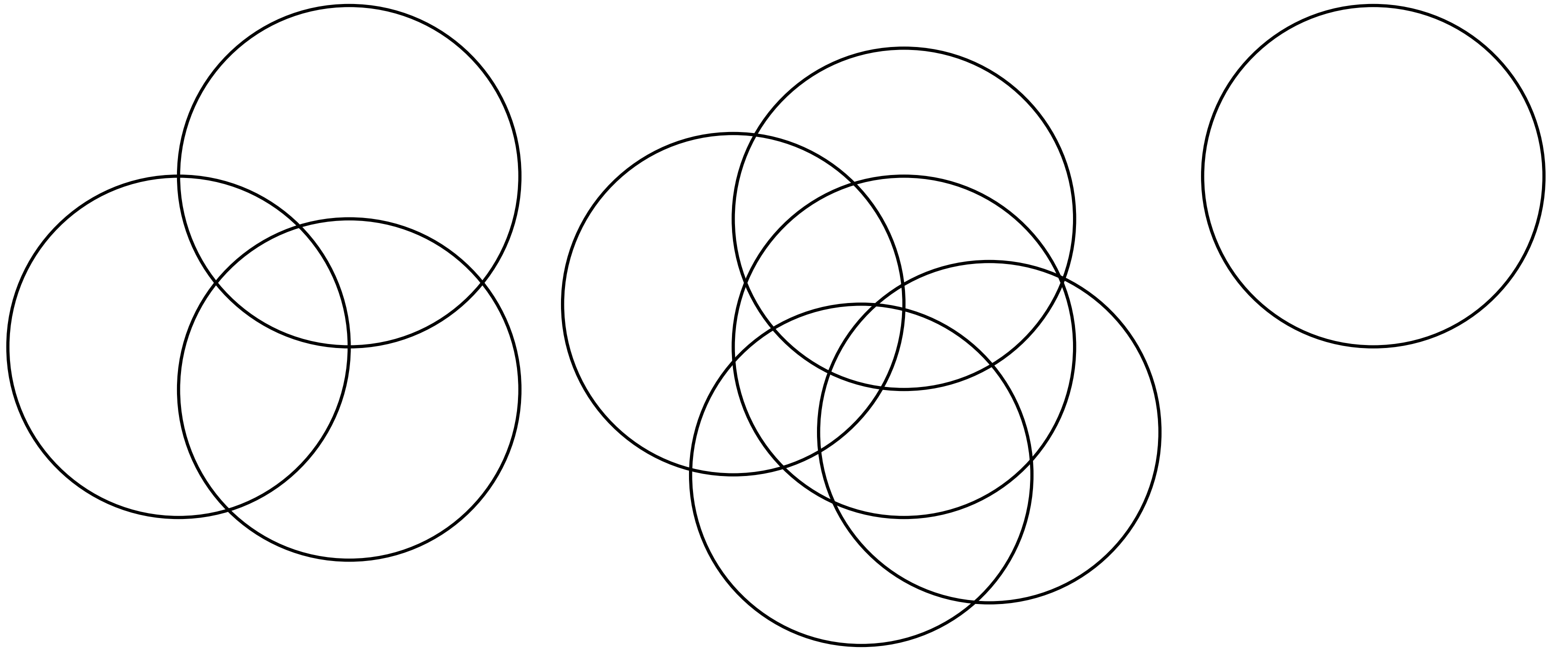


$\Longrightarrow$



upper envelope $\equiv$ lower hulll

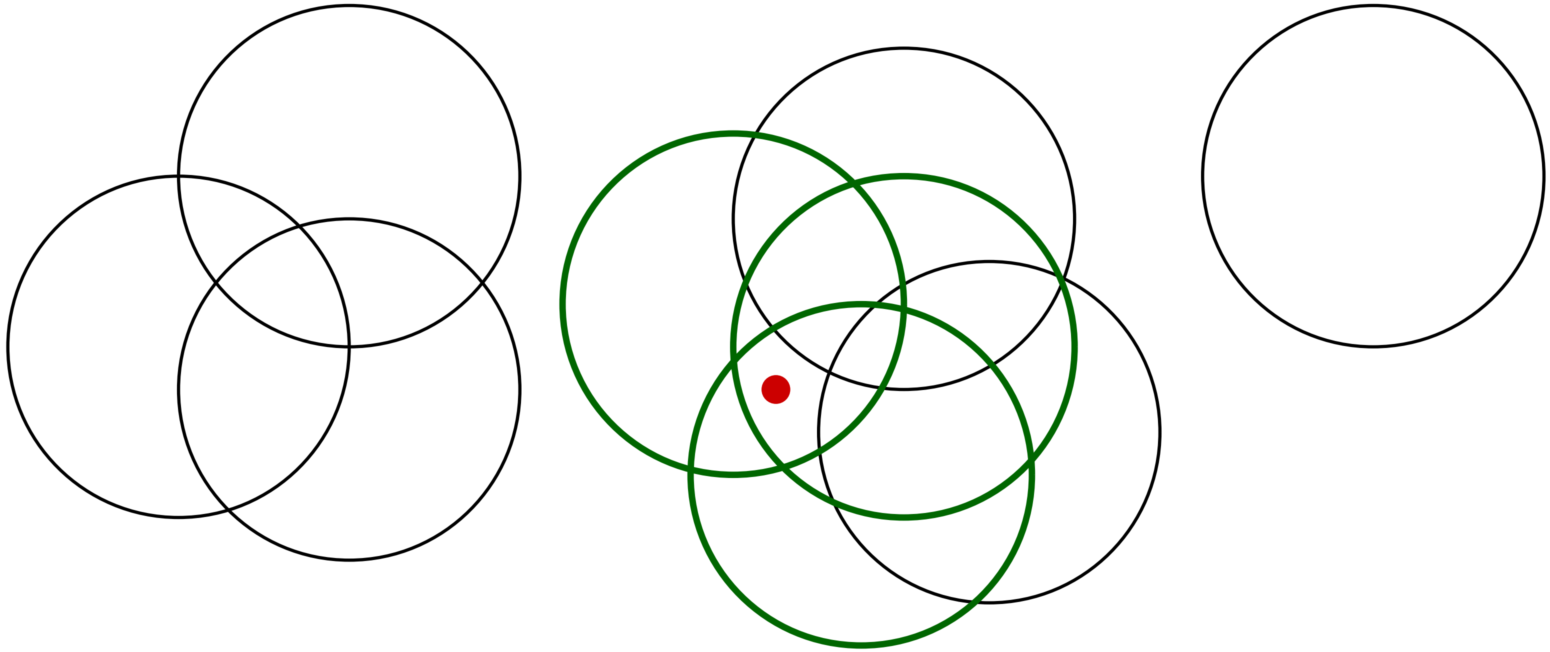
Delaunay Triangulations: Application to CF-Coloring



for $q \in \mathbb{R}^2$ define $D(q) :=$ { disks containing q }

Conflict-free coloring: coloring of disks such that, for any q with $S(q) \neq \emptyset$, the set $D(q)$ has a disk with a unique color

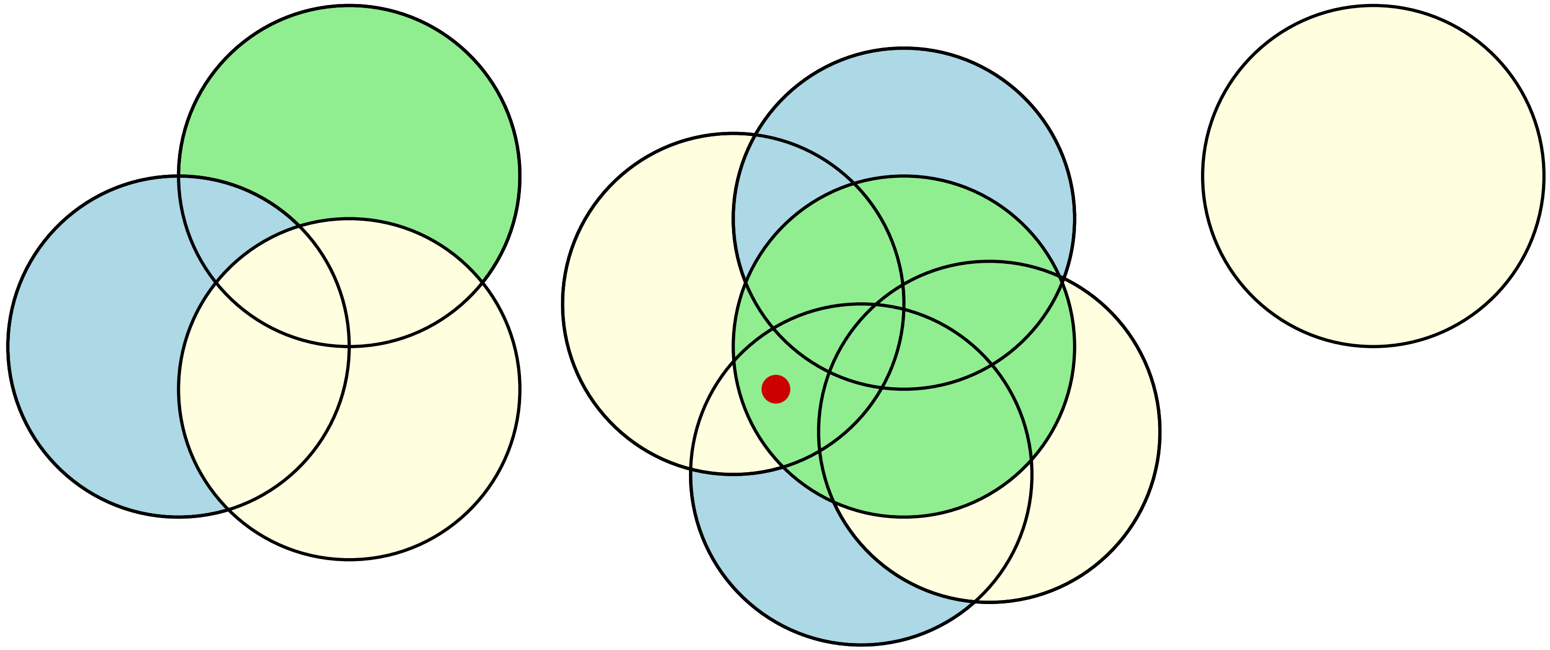
Delaunay Triangulations: Application to CF-Coloring



for $q \in \mathbb{R}^2$ define $D(q) :=$ { disks containing q }

Conflict-free coloring: coloring of disks such that, for any q with $S(q) \neq \emptyset$, the set $D(q)$ has a disk with a unique color

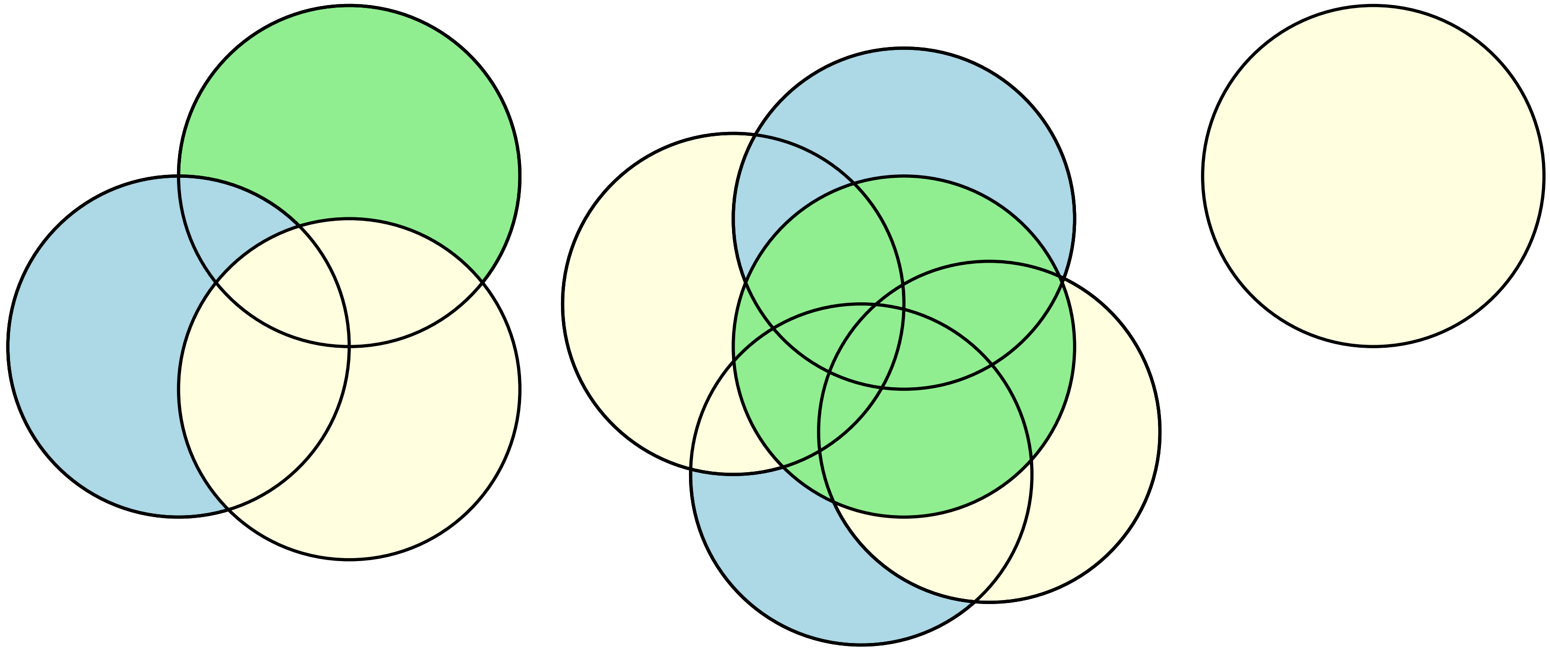
Delaunay Triangulations: Application to CF-Coloring



for $q \in \mathbb{R}^2$ define $D(q) :=$ { disks containing q }

Conflict-free coloring: coloring of disks such that, for any q with $S(q) \neq \emptyset$, the set $D(q)$ has a disk with a unique color

Delaunay Triangulations: Application to CF-Coloring



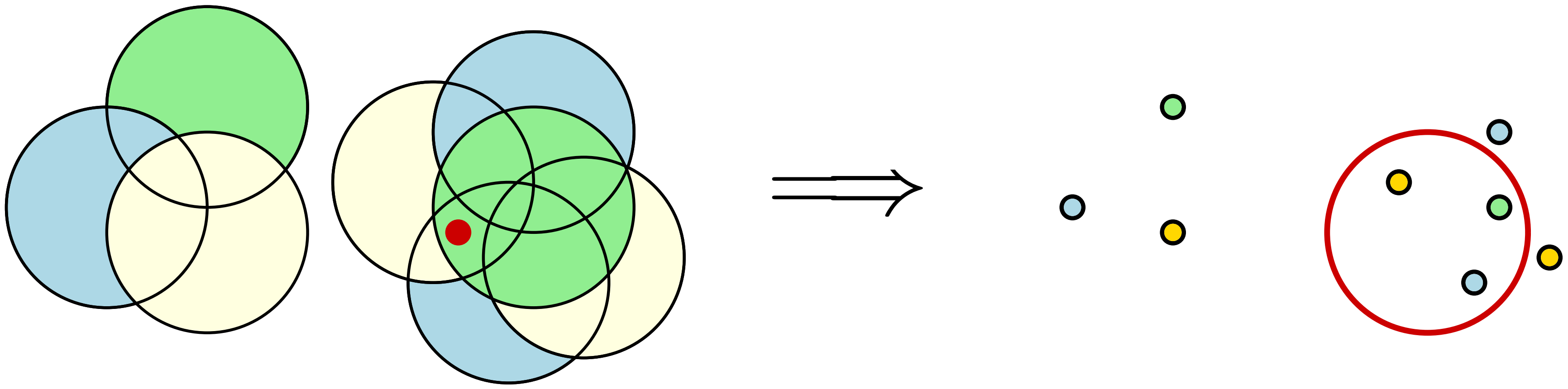
Theorem. For any set of n unit disks, there exists a conflict-free coloring with $O(\log n)$ colors, and this is best possible.

Delaunay Triangulations: Application to CF-Coloring

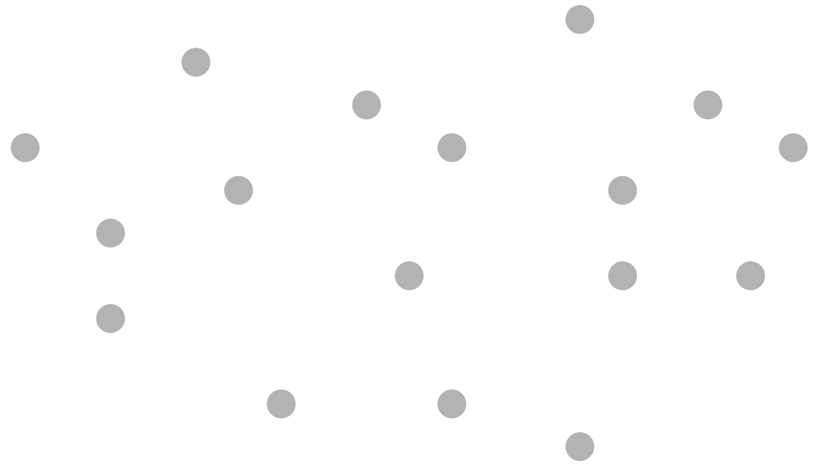
Invert problem: color disk centers with respect to unit disks as ranges

Delaunay Triangulations: Application to CF-Coloring

Invert problem: color disk centers with respect to unit disks as ranges



Delaunay Triangulations: Application to CF-Coloring

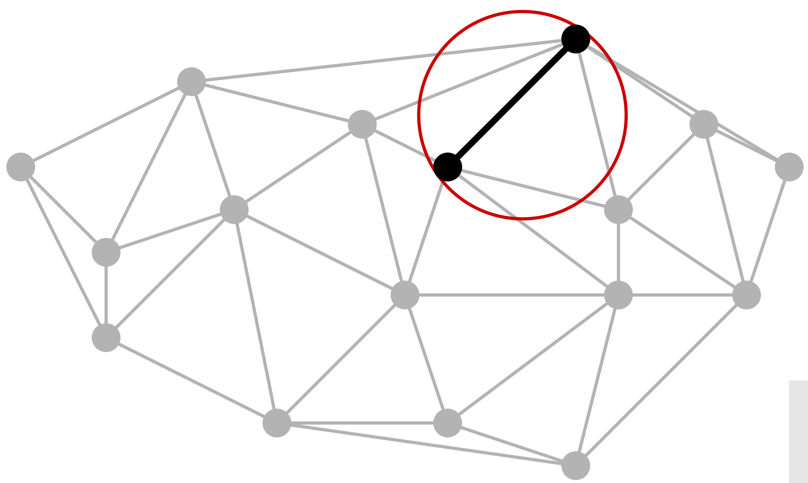


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

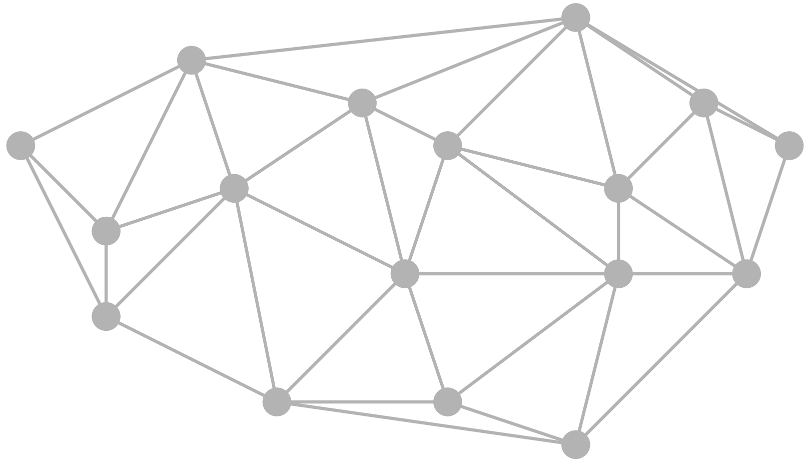


(p, q) is edge in DT iff there is a circle containing only p, q

The coloring algorithm

Initally $P = \{$all points$\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

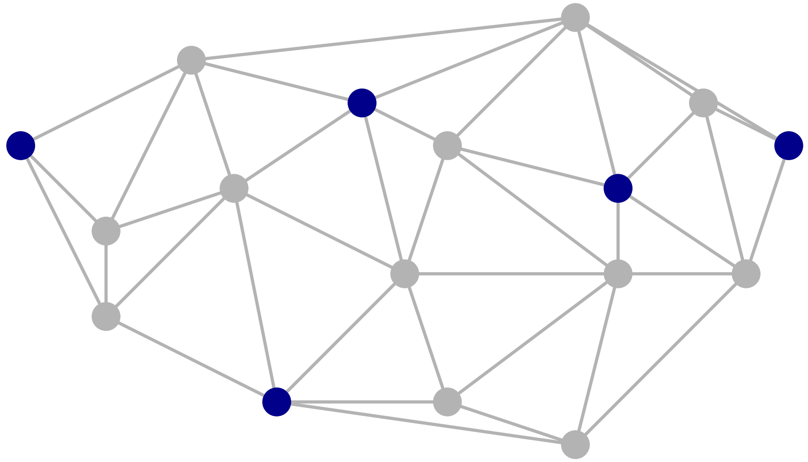


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

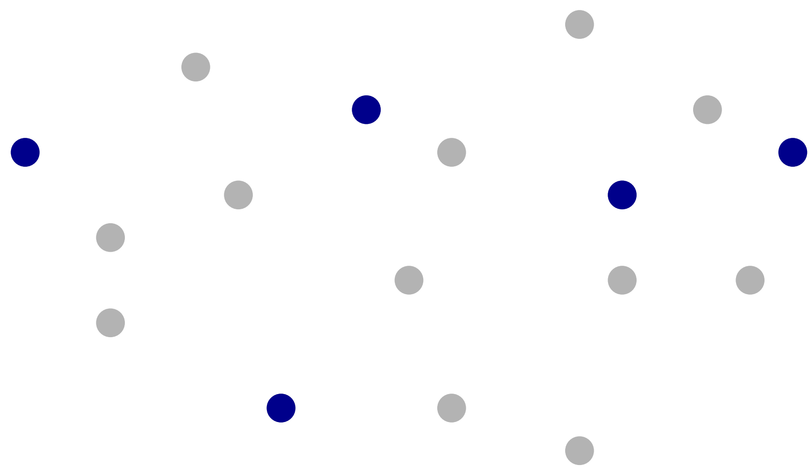


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

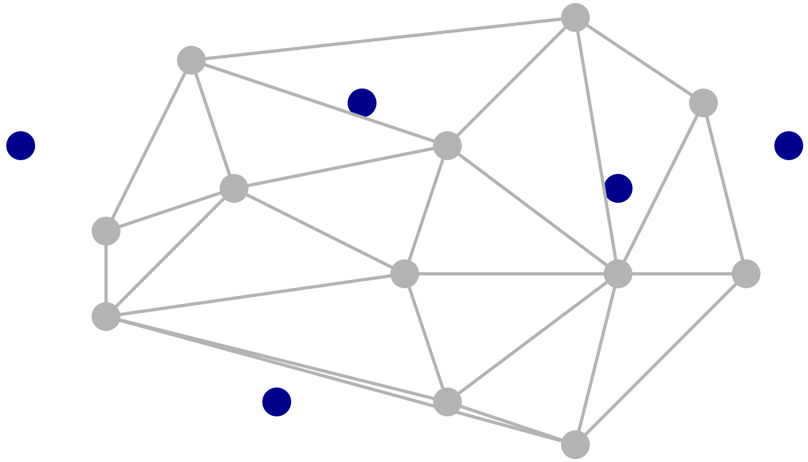


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

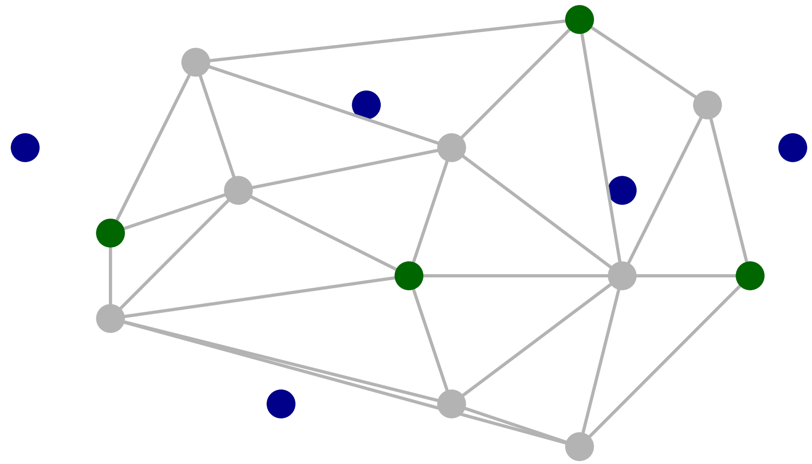


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

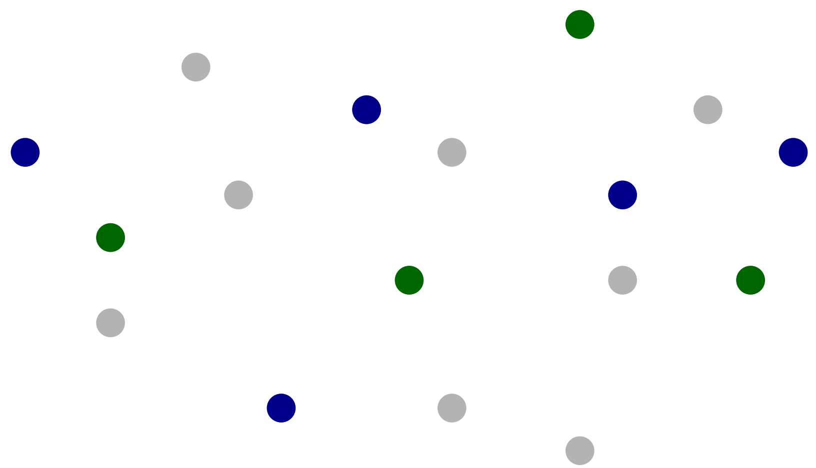


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

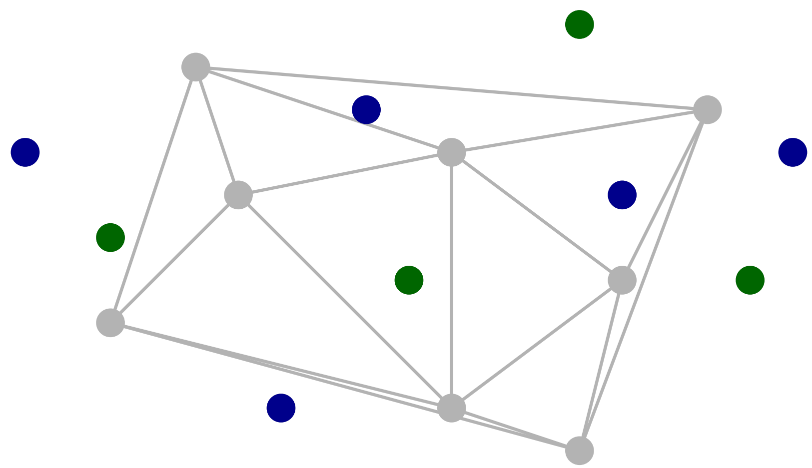


The coloring algorithm

Initally $P = \{$all points$\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

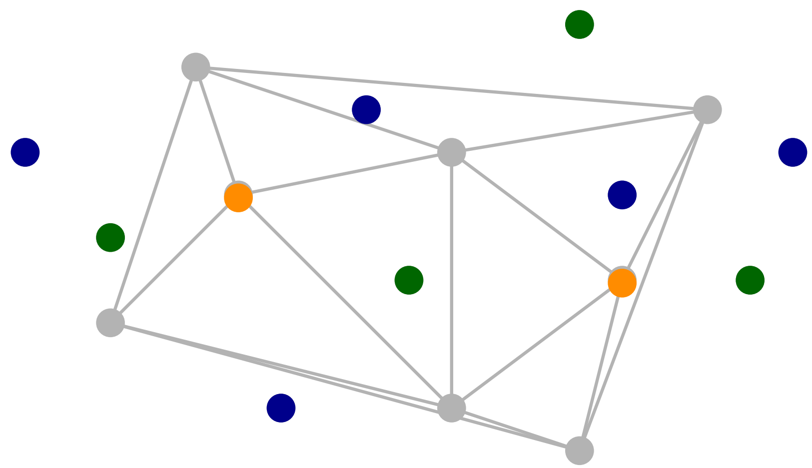


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring

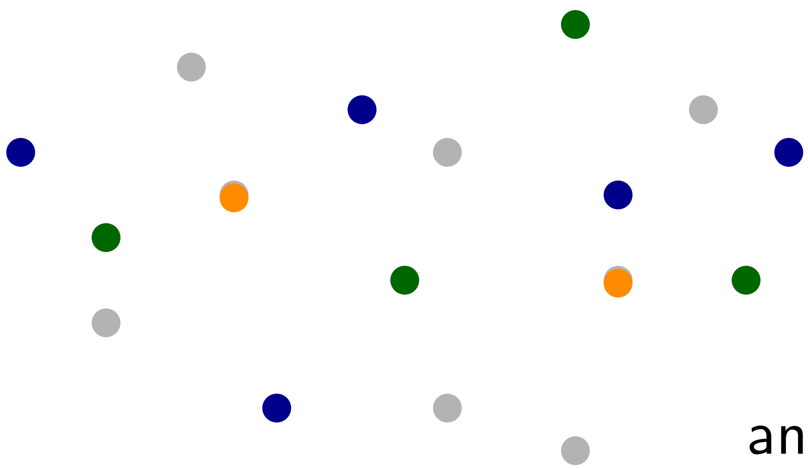


The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring



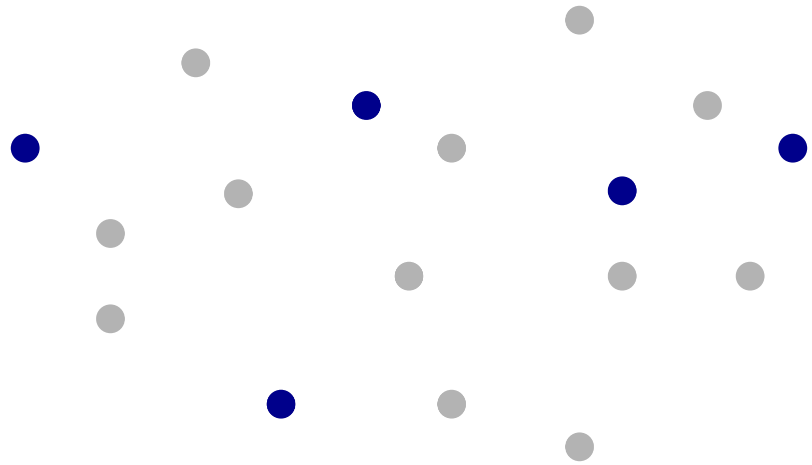
and so on . . .

The coloring algorithm

Initally $P = \{\text{all points}\}$ and $i = 1$

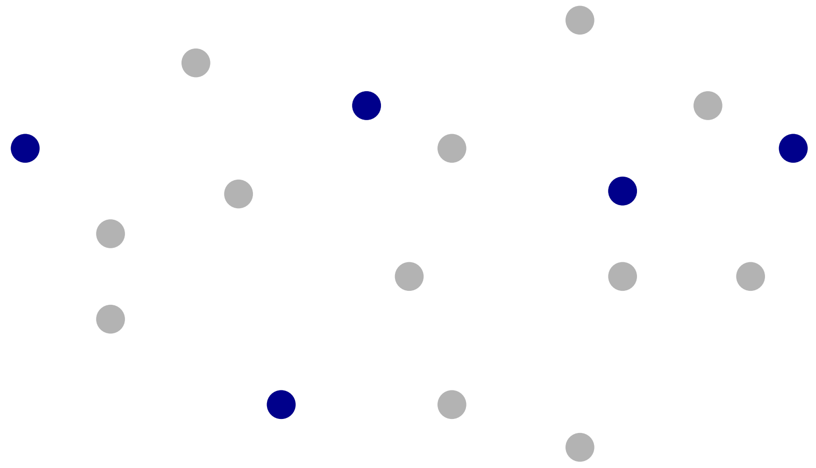
1. $I :=$ max independent set in Delaunay triangulation
2. Give all points in I color i
3. Set $i := i + 1$ and recurse on $P \setminus I$

Delaunay Triangulations: Application to CF-Coloring



Claim. Number of colors $= O(\log n)$.

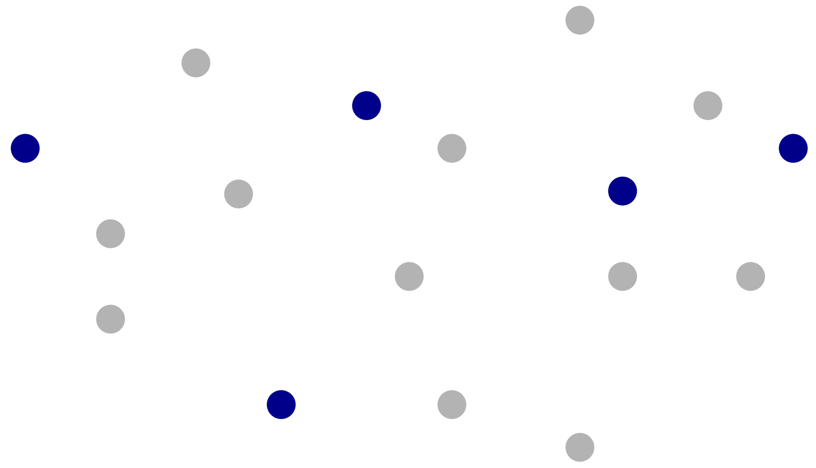
Delaunay Triangulations: Application to CF-Coloring



Claim. Number of colors $= O(\log n)$.

- Four Color Theorem $\implies$ size max indep set $\geqslant \frac{1}{4}n$

Delaunay Triangulations: Application to CF-Coloring

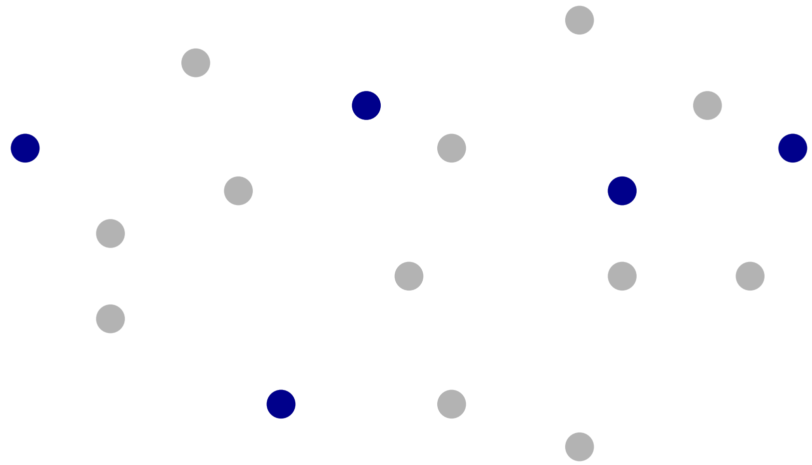


Claim. Number of colors $= O(\log n)$.

- Four Color Theorem $\implies$ size max indep set $\geqslant \frac{1}{4}n$
- $C(n) :=$ number of colors

$$C(n) \leqslant 1 + C\left(\frac{3}{4}n\right) \implies C(n) = O(\log n)$$

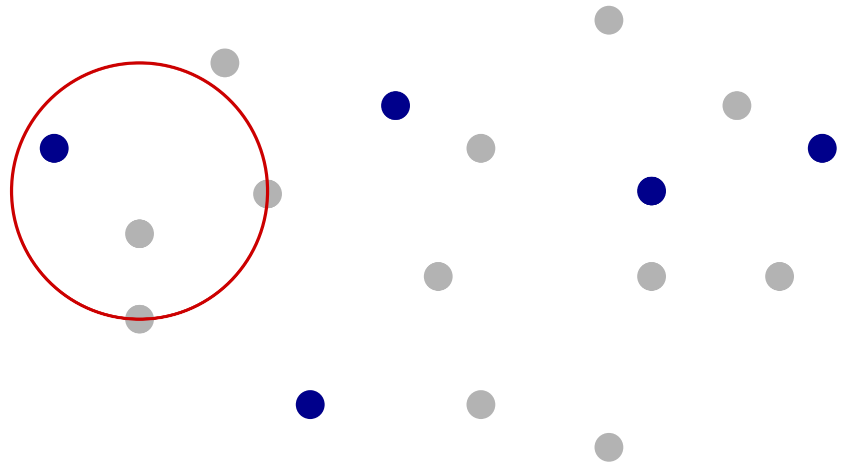
Delaunay Triangulations: Application to CF-Coloring



Claim. Coloring is conflict-free.

any non-empty disk must have point with unique color

Delaunay Triangulations: Application to CF-Coloring

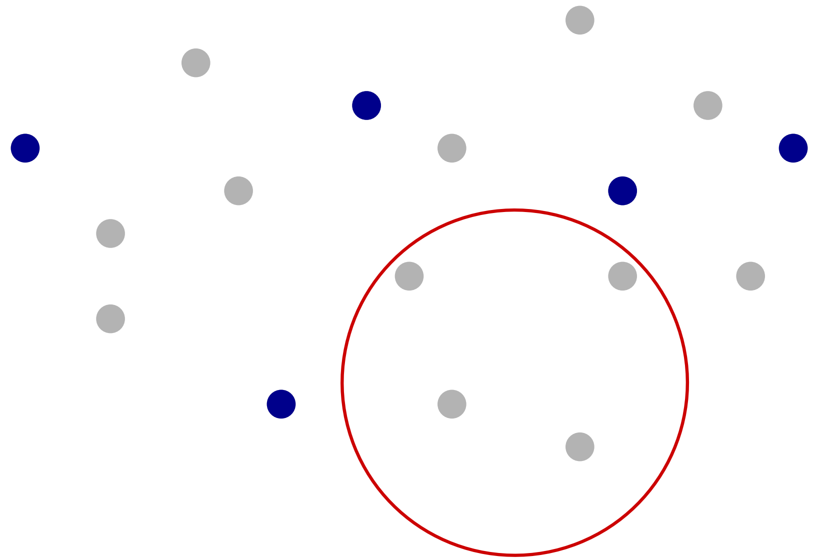


Claim. Coloring is conflict-free.

any non-empty disk must have point with unique color

- disk has single point with color 1

Delaunay Triangulations: Application to CF-Coloring

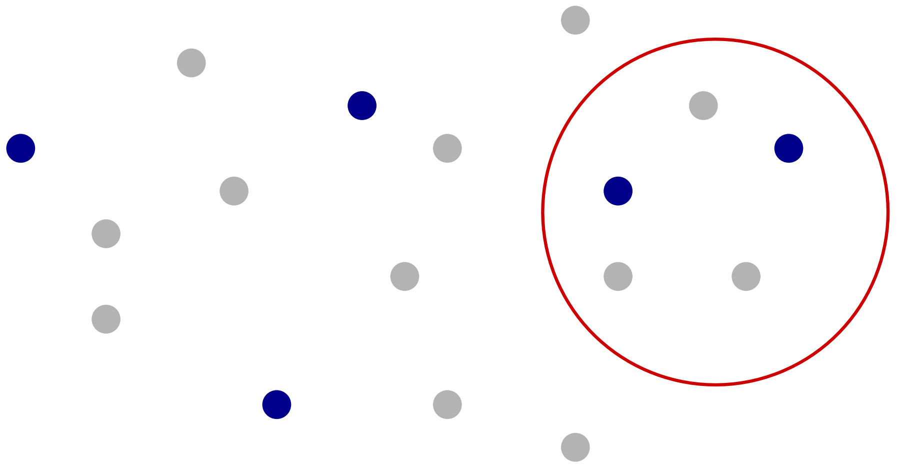


Claim. Coloring is conflict-free.

any non-empty disk must have point with unique color

- disk has single point with color 1
- disk has no point with color 1: induction

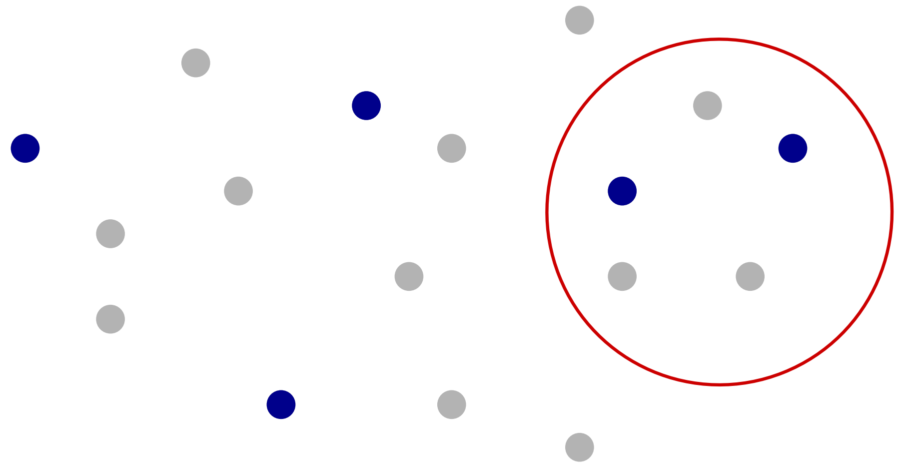
Delaunay Triangulations: Application to CF-Coloring



Claim. Coloring is conflict-free.

any non-empty disk must have point with unique color

- disk has single point with color 1
- disk has no point with color 1
- disk has $\geqslant 2$ points with color 1
  disk must contain other points $\Longrightarrow$ induction

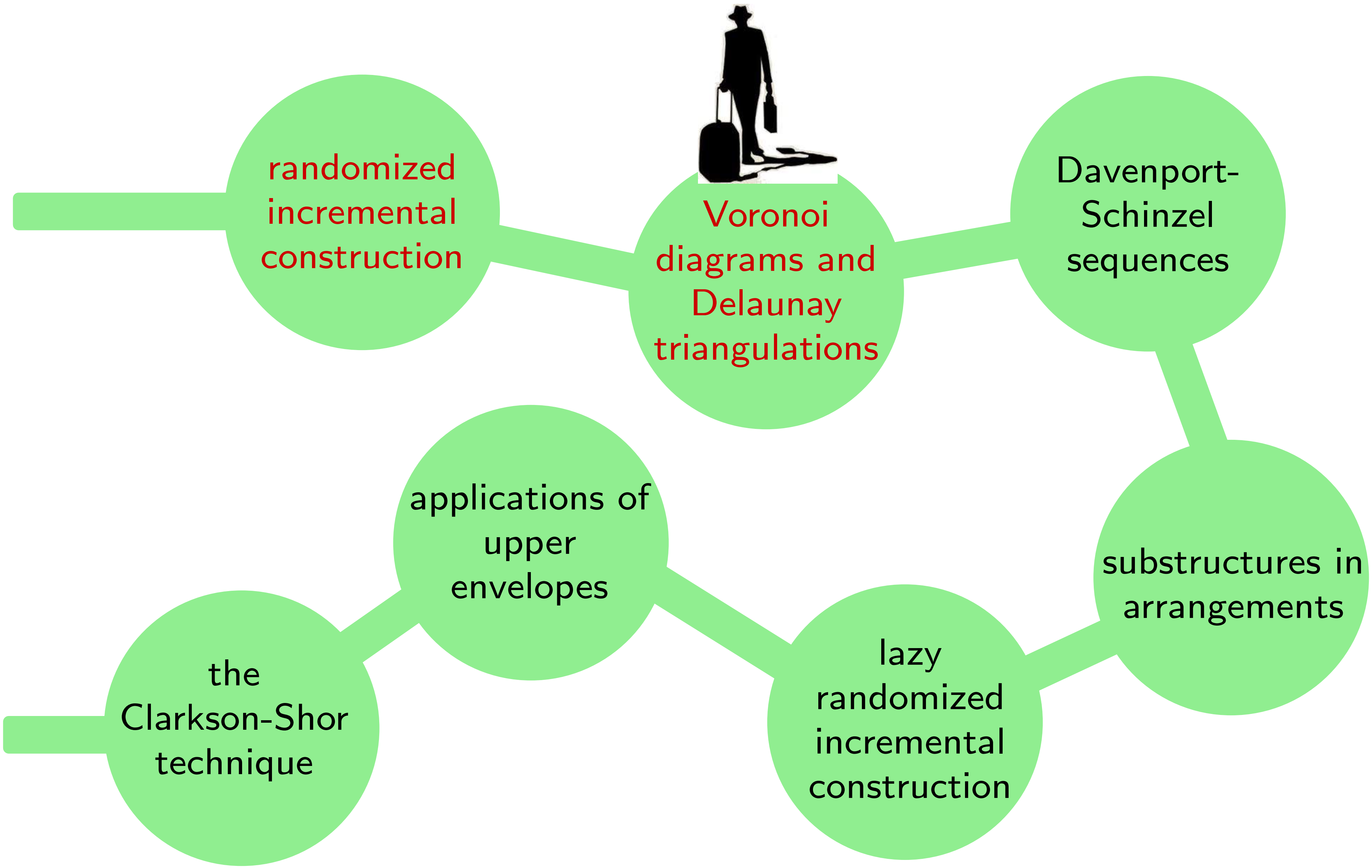


Claim. Coloring is conflict-free.

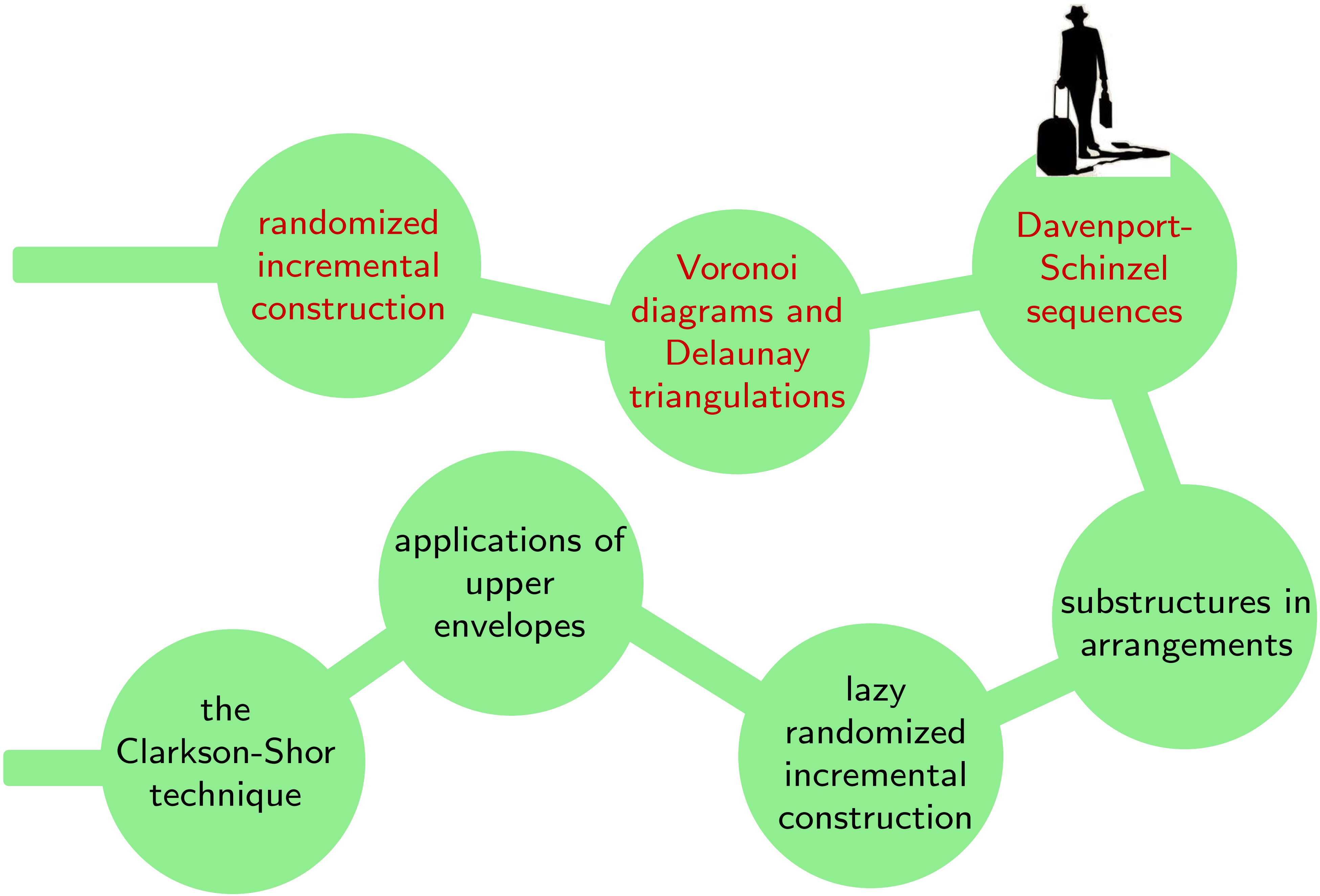
any non-empty disk must have point with unique color

- disk has single point with color 1
- disk has no point with color 1
- disk has $\geqslant 2$ points with color 1
  disk must contain other points $\Longrightarrow$ induction

Course Overview



Course Overview



Davenport-Schinzel sequences

A COMBINATORIAL PROBLEM CONNECTED WITH DIFFERENTIAL EQUATIONS.

By H. Davenport and A. Schinzel.

1. Let

(1) $$F(D)f(x) = 0$$

be a (homogeneous) linear differential equation with constant coefficients, of order d. Suppose that $F(D)$ has real coefficients, and that the roots of $F(\lambda) = 0$ are all real though not necessarily distinct. As is well known, any solution of (1) is of the form

(2) $$f(x) = P_1(x)e^{\lambda_1 x} + \cdots + P_k(x)e^{\lambda_k x},$$

where $\lambda_1, \cdots, \lambda_k$ are the distinct roots of $F(\lambda) = 0$ and $P_1(x), \cdots, P_k(x)$ are polynomials of degrees at most $m_1 - 1, \cdots, m_k - 1$, where $m_1, \cdots, m_k$ are the multiplicities of the roots, so that $m_1 + \cdots + m_k = d$.

Let

(3) $$f_1(x), \cdots, f_n(x)$$

be n distinct (but not necessarily independent) solutions of (1). For each real number x, apart from a finite number of exceptions, there will be just one of the functions (3) which is greater than all the others. We can therefore dissect the real line into N intervals

$$(-\infty, x_1), (x_1, x_2), \cdots, (x_{N-1}, \infty)$$

such that inside any one of the intervals (x_{j-1}, x_j) a particular one of the functions (3) is the greatest, and such that this function is not the same for two consecutive intervals. It is almost obvious that N is finite, and a formal proof will be given below.

The problem of finding how large N can be, for given d and given n, was proposed to one of us (in a slightly different form) by K. Malanowski. This problem can be made to depend on a purely combinatorial problem, by the following considerations. With each $j = 1, 2, \cdots, N$ there is associated the integer $i = i(j)$ for which $f_i(x)$ is the greatest of the functions (3) in the interval (x_{j-1}, x_j). (We write $x_0 = -\infty$ and $x_N = \infty$ for convenience.) This defines a sequence of N terms

(4) $$i(1), i(2), \cdots, i(N),$$

Received August 26, 1964.
684

American Journal of Mathematics 87:684–694 (1965)

Harold Davenport (1907–1965)

Andrzej Schinzel (1937–2021)

A combinatorial problem

Consider a sequence over the alphabet $\{1, \ldots, n\}$ such that

- $\ldots\ i\ i\ \ldots$ does not appear
- $\ldots\underbrace{i\ldots j\ldots i\ldots j}_{s+2\text{ times}}\ldots$ does not appear

How long can such a sequence be?

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ — no two consecutive symbols are the same
- $\ldots\underbrace{i\ldots j\ldots i\ldots j}_{s+2\text{ times}}\ldots$ — no alternating subsequence of length $s+2$

Example $(n = 9,\ s = 2)$

- 6, 4, 5, 6, 1, 2, 2, 7, 3
- 2, 5, 1, 2, 7, 8, 7, 1, 3, 4
- 3, 6, 4, 2, 5, 1, 5, 9, 8, 9, 7

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\dots\ i\ i\ \dots$ no two consecutive symbols are the same
- $\dots\underbrace{i\dots j\dots i\dots j}_{s+2 \text{ times}}\dots$ no alternating subsequence of length $s+2$

Example $(n=9,\ s=2)$

- 6, 4, 5, 6, 1, 2, 2, 7, 3 ✕
- 2, 5, 1, 2, 7, 8, 7, 1, 3, 4
- 3, 6, 4, 2, 5, 1, 5, 9, 8, 9, 7

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ — no two consecutive symbols are the same
- $\ldots\underbrace{i\ldots j\ldots i\ldots j}_{s+2 \text{ times}}\ldots$ — no alternating subsequence of length $s+2$

Example $(n = 9,\ s = 2)$

- 6, 4, 5, 6, 1, 2, 2, 7, 3 ✗
- 2, 5, 1, 2, 7, 8, 7, 1, 3, 4 ✗
- 3, 6, 4, 2, 5, 1, 5, 9, 8, 9, 7

Davenport-Schinzel sequences

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ no two consecutive symbols are the same
- $\ldots \underbrace{i \ldots j \ldots i \ldots j}_{s+2 \text{ times}} \ldots$ no alternating subsequence of length $s+2$

Example $(n = 9,\ s = 2)$

- 6, 4, 5, 6, 1, 2, 2, 7, 3 ✕
- 2, 5, 1, 2, 7, 8, 7, 1, 3, 4 ✕
- 3, 6, 4, 2, 5, 1, 5, 9, 8, 9, 7 V

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\dots\ i\ i\ \dots$ no two consecutive symbols are the same
- $\dots\underbrace{i\dots j\dots i\dots j}_{s+2\text{ times}}\dots$ no alternating subsequence of length $s+2$

Example $(n=9,\ s=2)$

- 6, 4, 5, 6, 1, 2, 2, 7, 3 ✕
- 2, 5, 1, 2, 7, 8, 7, 1, 3, 4 ✕
- 3, 6, 4, 2, 5, 1, 5, 9, 8, 9, 7 V

Exercise: Determine the maximal possible length of a DS-sequence of order s as a function of n, for $s=1$, $s=2$, $s=3$, ...

Davenport-Schinzel sequences

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ no two consecutive symbols are the same
- $\ldots\underbrace{i\ldots j\ldots i\ldots j}_{s+2 \text{ times}}\ldots$ no alternating subsequence of length $s+2$

Davenport-Schinzel sequences

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ no two consecutive symbols are the same
- $\ldots\underbrace{i\ldots j\ldots i\ldots j}_{s+2\text{ times}}\ldots$ no alternating subsequence of length $s+2$

$\mathrm{DS}_s(n) :=$ maximum length of DS-sequence of order s on n symbols

- $s = 1$:

- $s = 2$:

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ — no two consecutive symbols are the same
- $\ldots \underbrace{i \ldots j \ldots i \ldots j}_{s+2 \text{ times}} \ldots$ — no alternating subsequence of length $s+2$

$\mathrm{DS}_s(n) :=$ maximum length of DS-sequence of order s on n symbols

- $s = 1$: $\left.\begin{array}{l}\text{possible sequence: } 1, 2, 3, \ldots, n \\ \text{no symbol can appear twice}\end{array}\right\} \Longrightarrow \quad \mathrm{DS}_1(n) = n$

- $s = 2$:

Davenport-Schinzel sequences

Davenport-Schinzel sequence of order s (over alphabet of size n) is sequence that does not contain the following:

- $\ldots\ i\ i\ \ldots$ — no two consecutive symbols are the same
- $\ldots \underbrace{i \ldots j \ldots i \ldots j}_{s+2 \text{ times}} \ldots$ — no alternating subsequence of length $s+2$

$\mathrm{DS}_s(n) :=$ maximum length of DS-sequence of order s on n symbols

- $s = 1$: $\left.\begin{array}{l}\text{possible sequence: } 1, 2, 3, \ldots, n \\ \text{no symbol can appear twice}\end{array}\right\} \implies \mathrm{DS}_1(n) = n$

- $s = 2$: possible sequence $1, 2, \ldots, n-1, n, n-1, \ldots, 2, 1$

  $\implies \mathrm{DS}_2(n) \geqslant 2n - 1$

  Proof by induction, remove symbol whose first occurrence is last, plus at most one adjacent symbol:

  $\mathrm{DS}_2(n) \leqslant \mathrm{DS}(n-1) + 2 \implies \mathrm{DS}_2(n) \leqslant 2n - 1$

Theorem. $\mathrm{DS}_s(n)$ is near-linear for any constant s. In particular,

- $\mathrm{DS}_1(n) = n$
- $\mathrm{DS}_2(n) = 2n - 1$
- $\mathrm{DS}_3(n) = \Theta(n\alpha(n))$
- $\mathrm{DS}_s(n) = o(n \log^* n)$ for any fixed constant s

where $\alpha(n)$ is the inverse Ackermann function

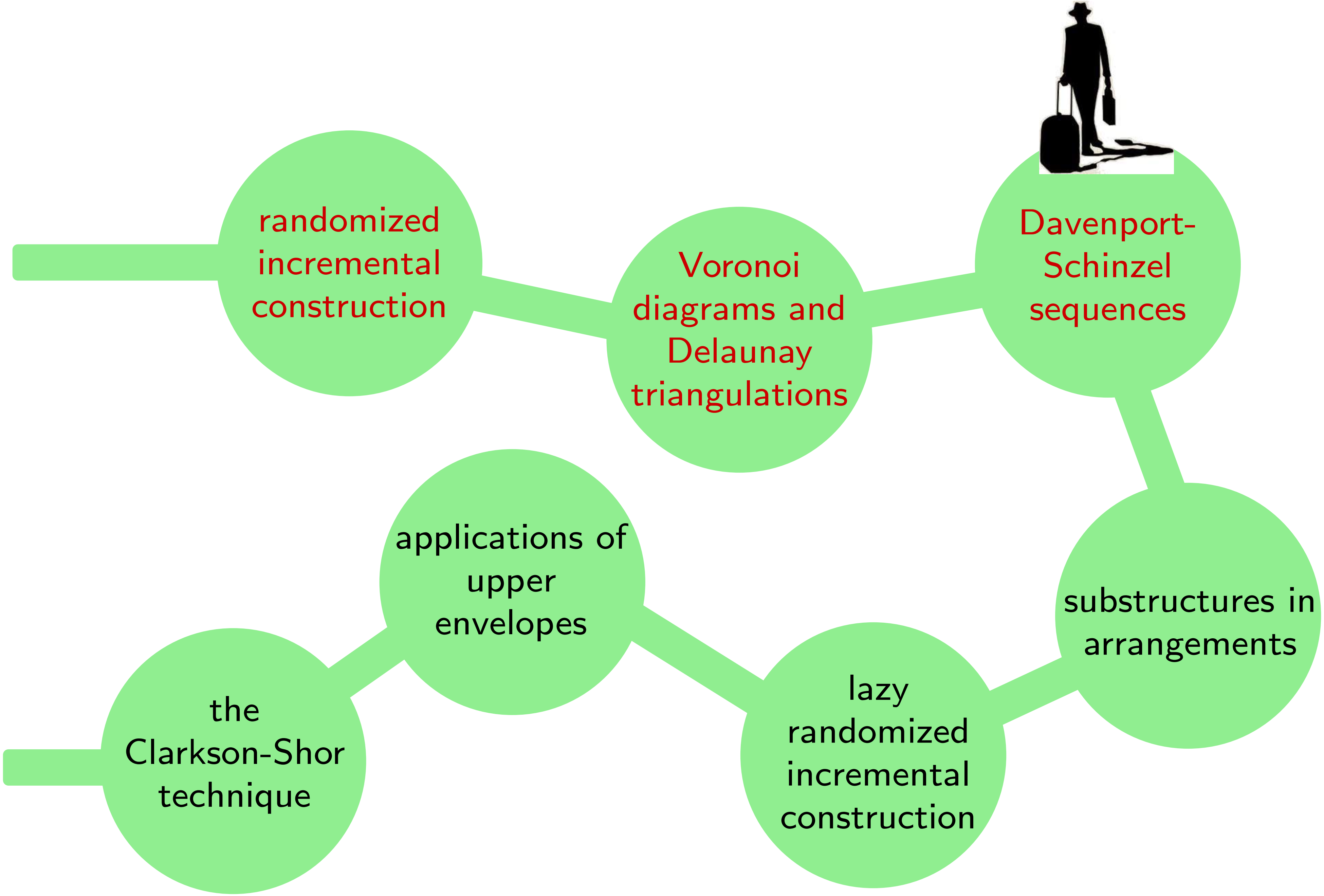
$\alpha(n)$ grows slower than super-super-super-super-super-slowly ...

$\alpha(n)$ is inverse of Ackermann function $A(n)$, where $A(n) = A_n(n)$ with:

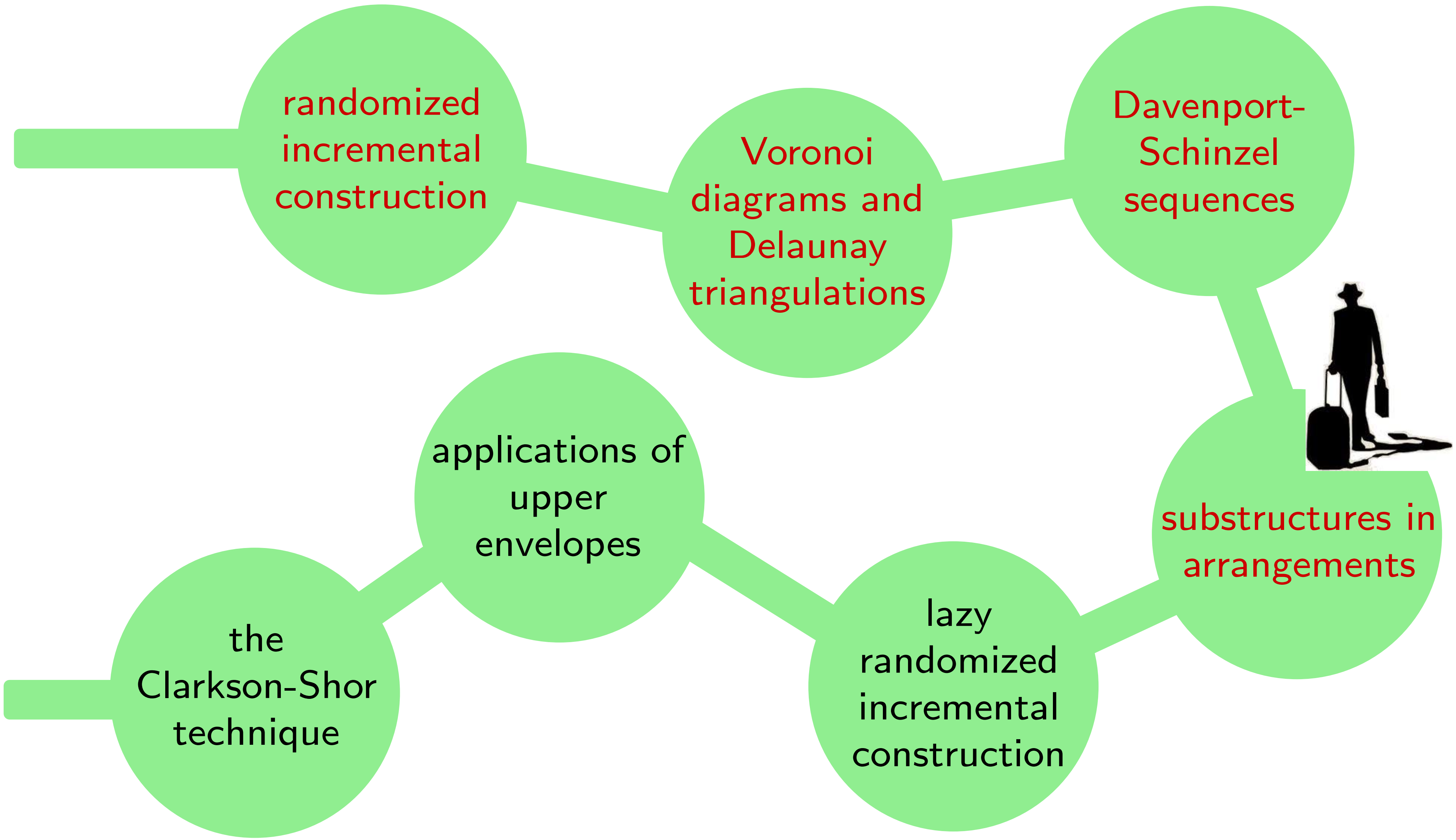
$$\begin{array}{rcll} A_1(n) & = & 2n & \text{for } n \geqslant 1 \\ A_k(1) & = & 2 & \text{for } k \geqslant 1 \\ A_k(n) & = & A_{k-1}(A_k(n-1)) & \text{for } k \geqslant 2 \text{ and } n \geqslant 2 \end{array}$$

$A(1) = 2$, $A(2) = 4$, $A(3) = 16$, $A(4) =$ tower of 65536 2's

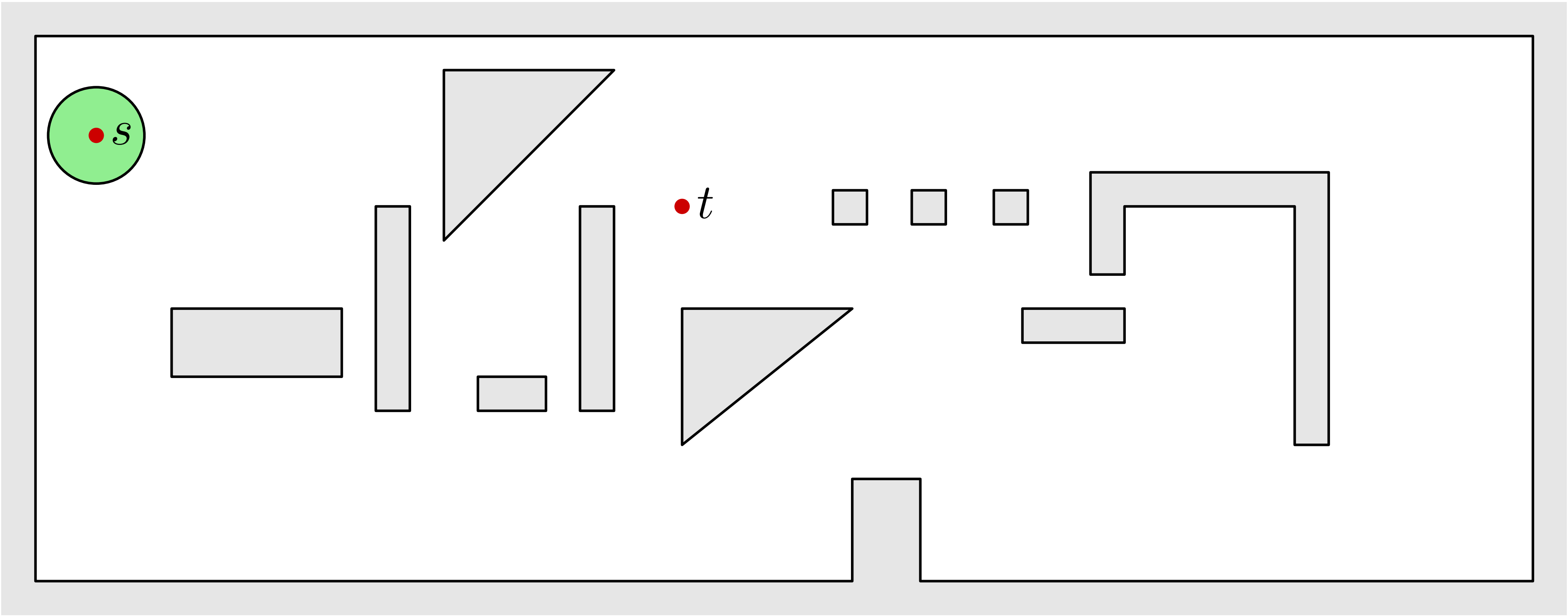
Course Overview

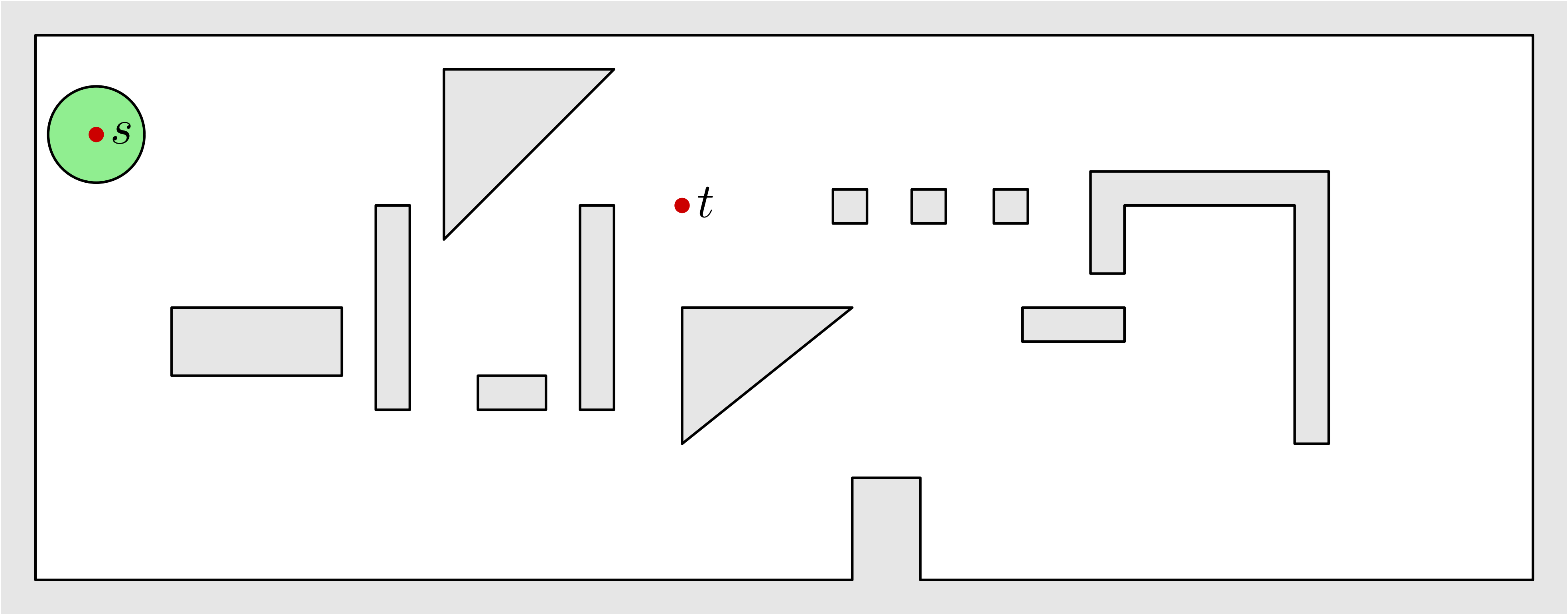


Course Overview

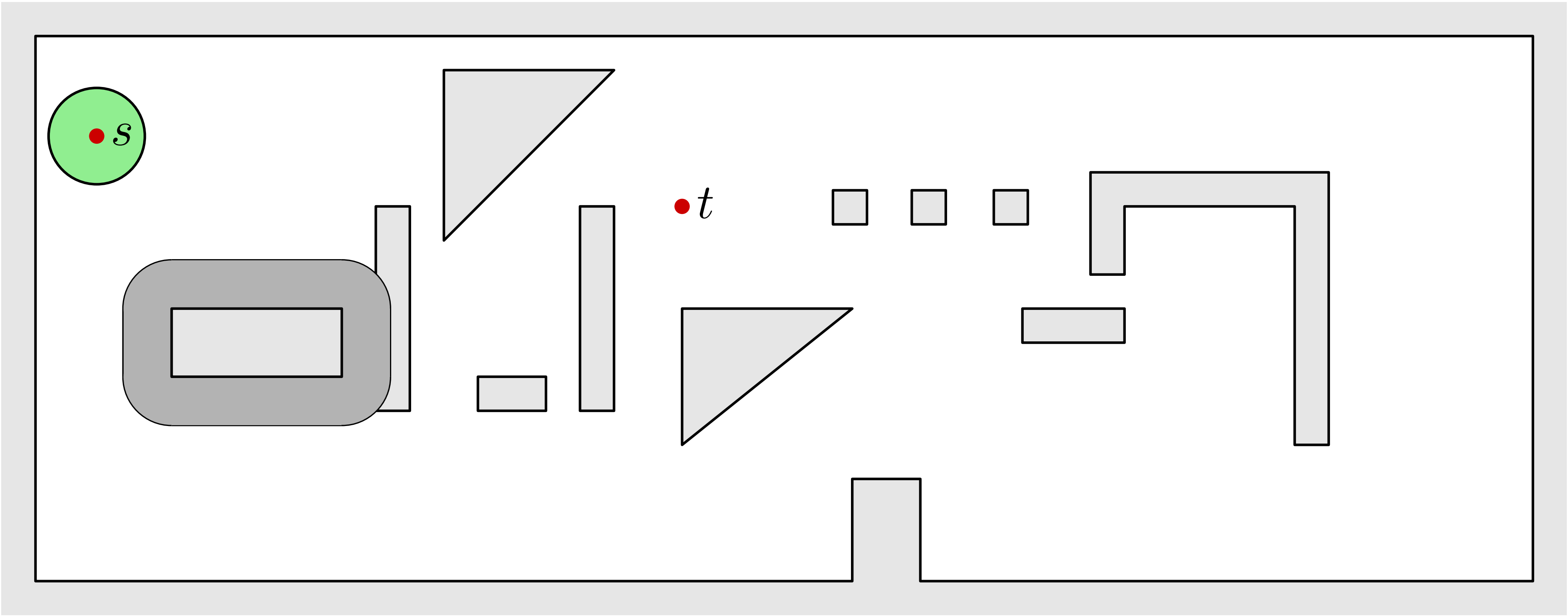


Robot Motion Planning

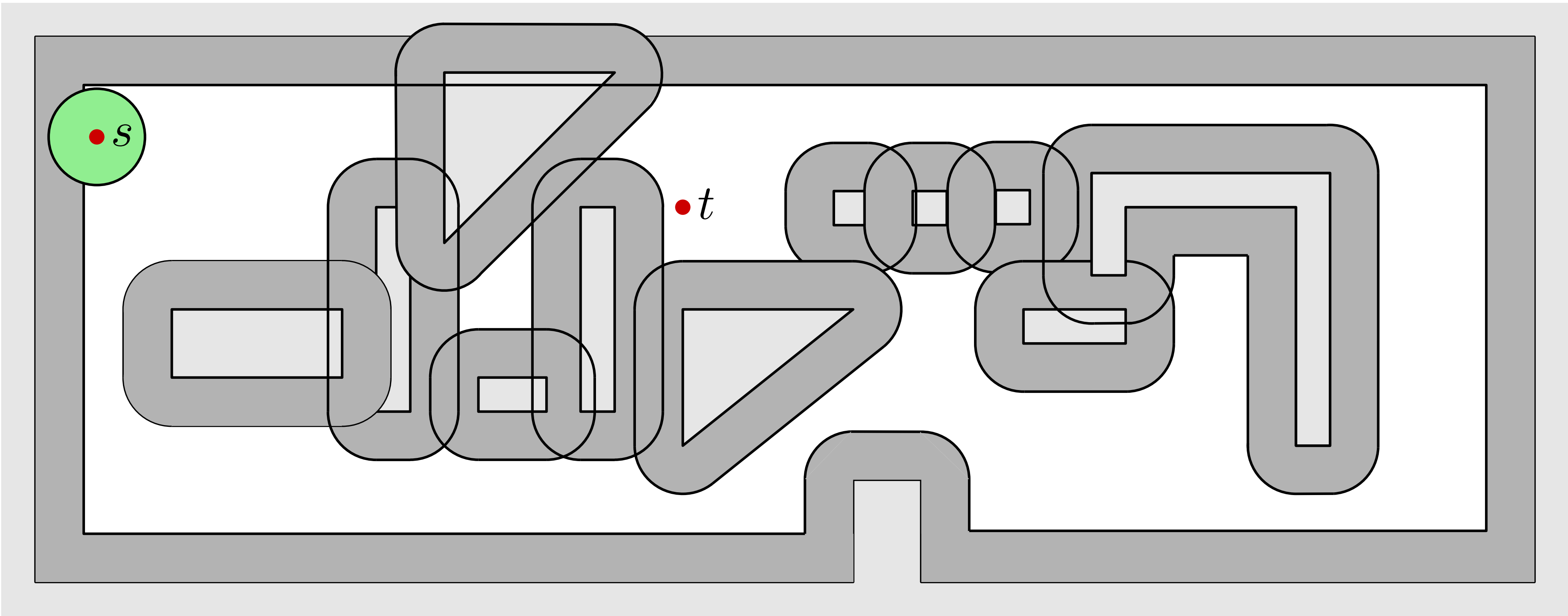




1. Transform problem to motion-planning problem for a point-shaped robot

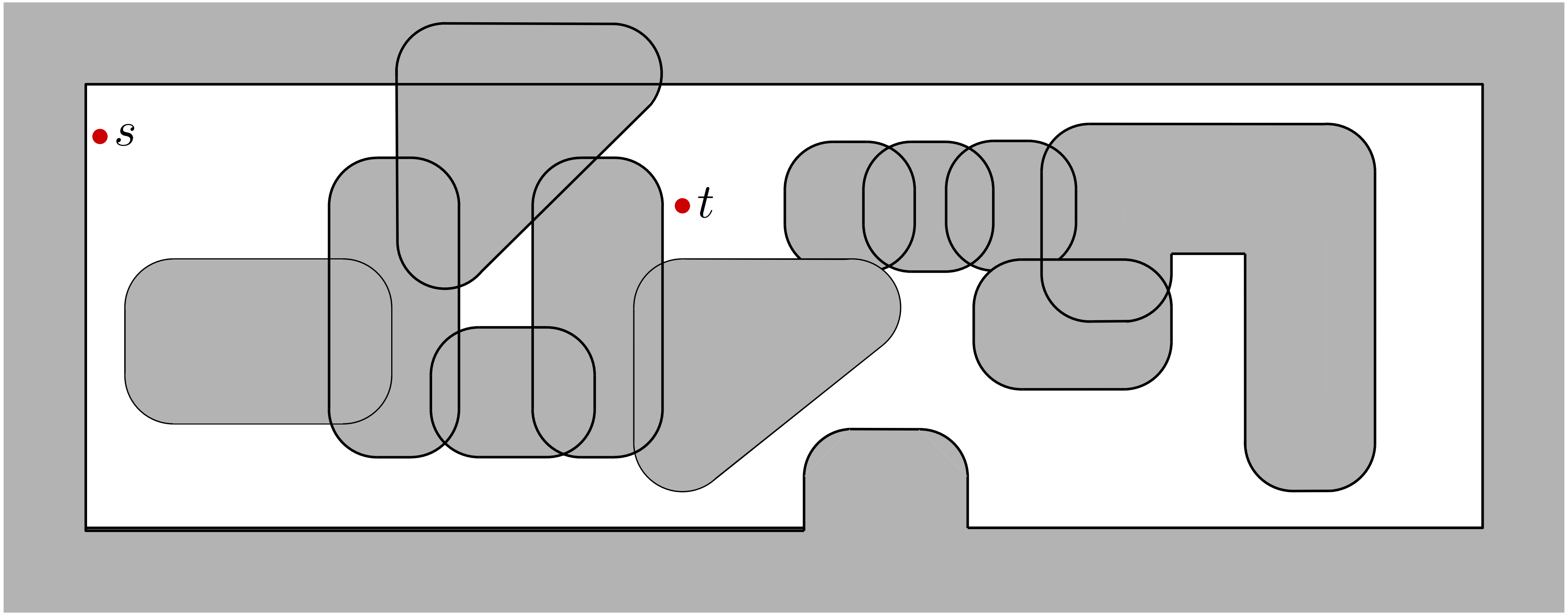


1. Transform problem to motion-planning problem for a point-shaped robot

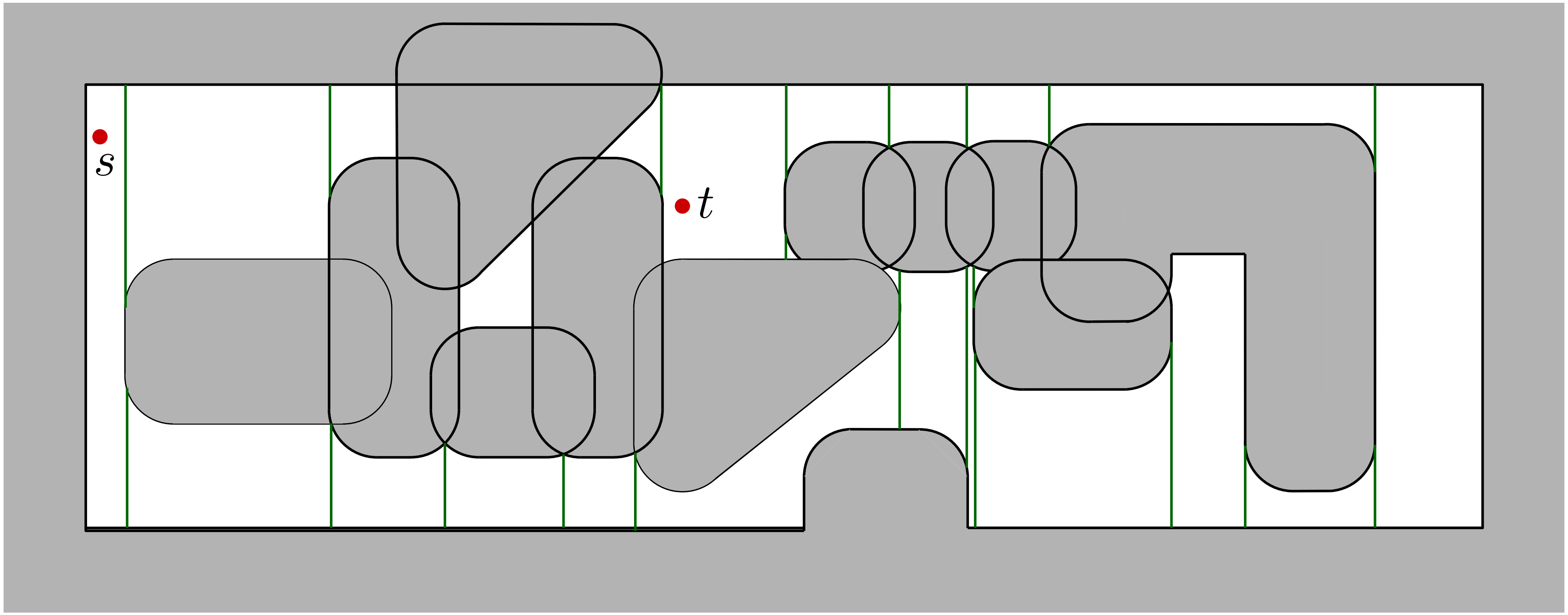


1. Transform problem to motion-planning problem for a point-shaped robot

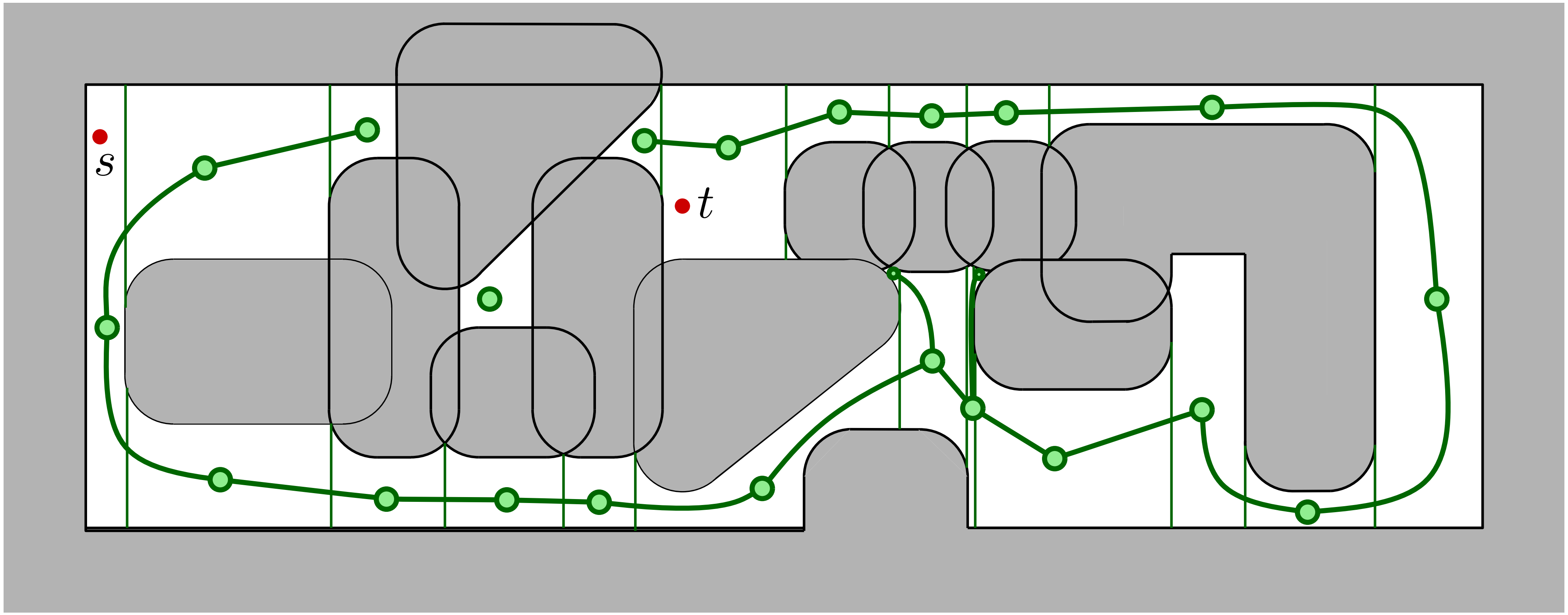
Robot Motion Planning



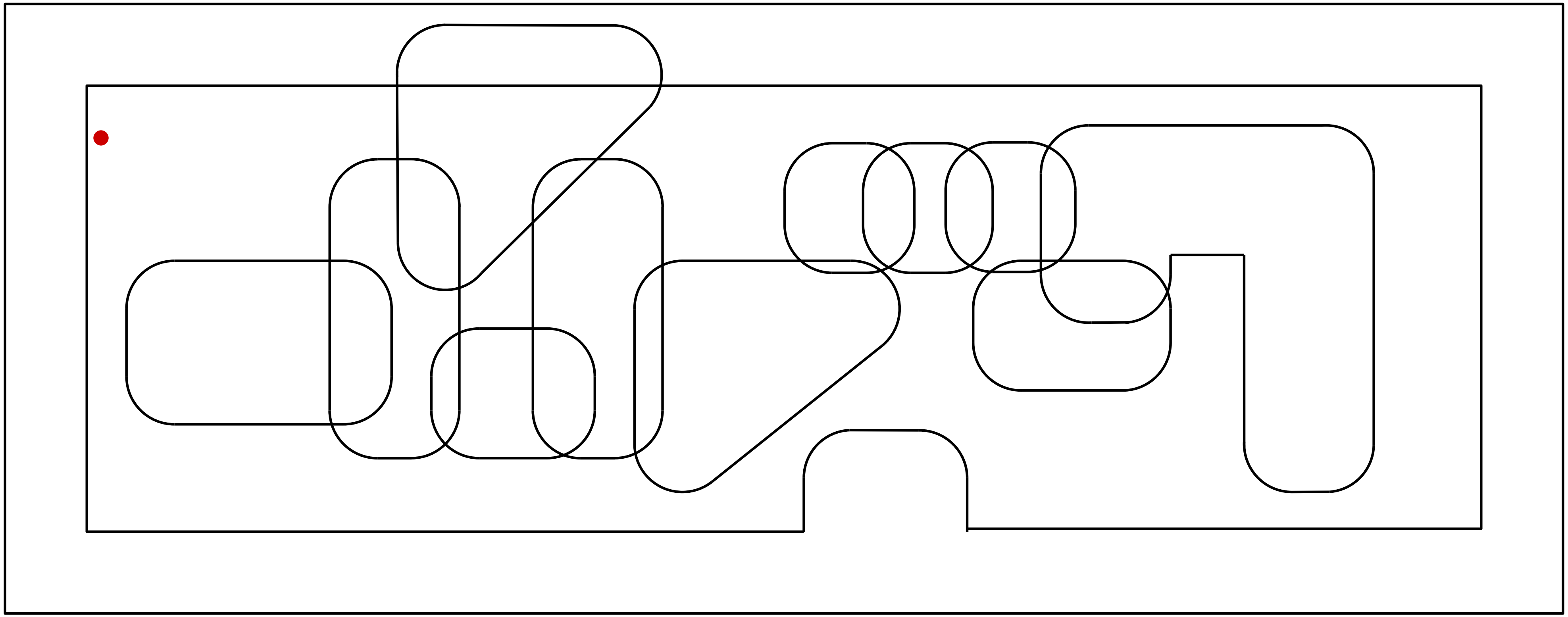
1. Transform problem to motion-planning problem for a point-shaped robot by expanding each obstacle. (Expanded obstacles can intersect!)



1. Transform problem to motion-planning problem for a point-shaped robot by expanding each obstacle. (Expanded obstacles can intersect!)
2. Decompose free space into "quadrilaterals"



1. Transform problem to motion-planning problem for a point-shaped robot by expanding each obstacle. (Expanded obstacles can intersect!)
2. Decompose free space into "quadrilaterals"
3. Construct motion graph $\mathcal{G}$ and compute path from s to t in $\mathcal{G}$



reachable region of the robot

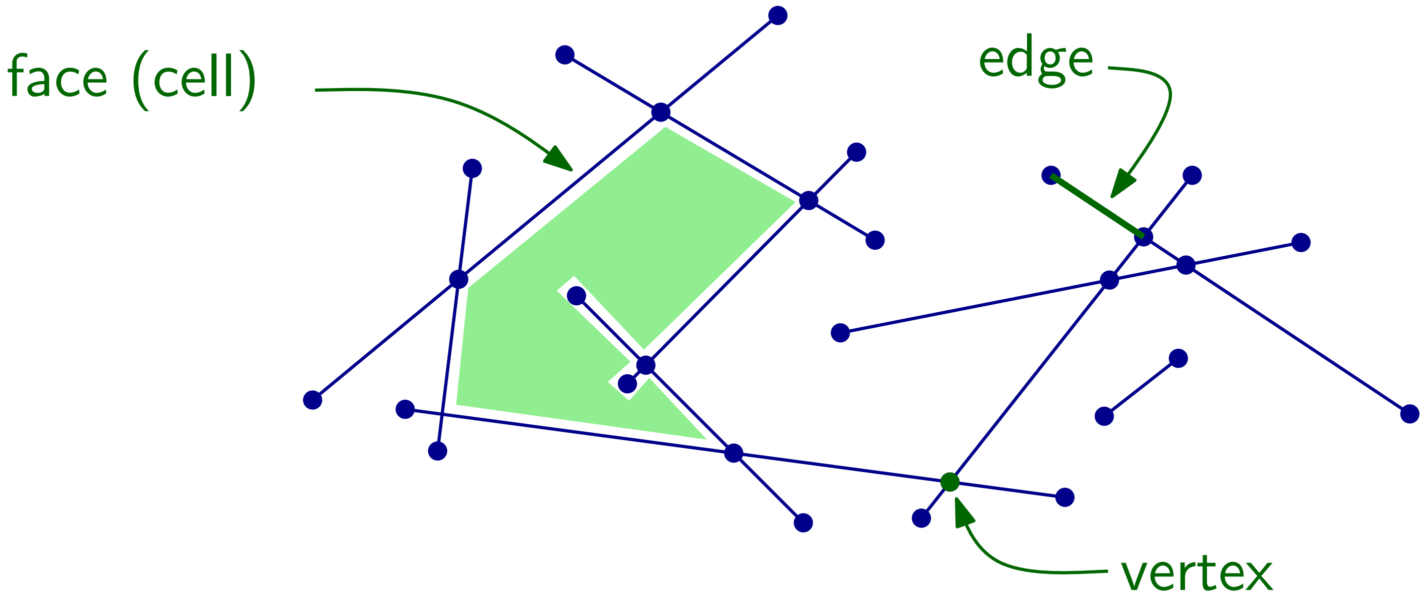
$=$

single cell in arrangement induced by a set S of n curves in $\mathbb{R}^2$

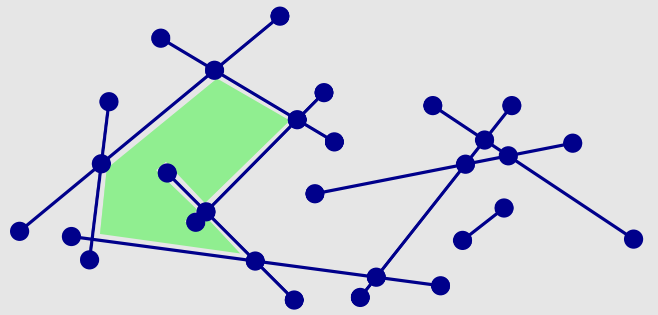
for other types of robots: in $\mathbb{R}^d$, where $d = \#$(degrees of freedom)

S: set of n lines / segments / curves / etc in $\mathbb{R}^2$

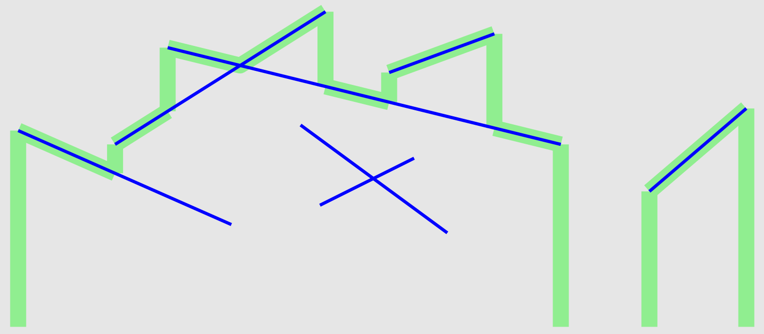
$\mathcal{A}(S)$ = arrangement induced by S
= partitioning of $\mathbb{R}^2$ into faces, edges, and vertices induced by S



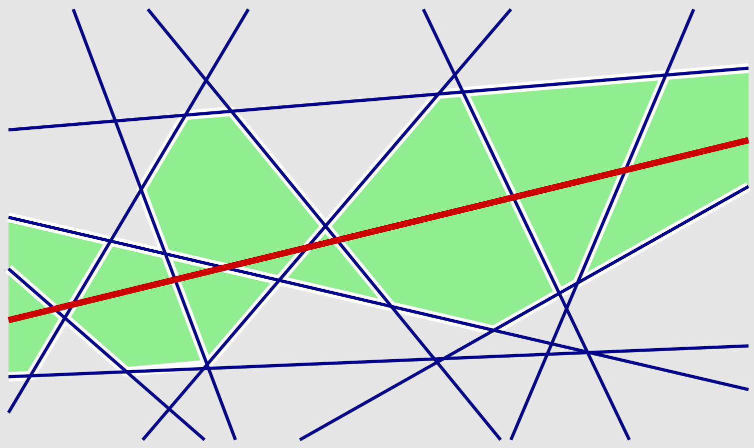
combinatorial complexity of $\mathcal{A}(S)$ = total number of vertices, edges, faces



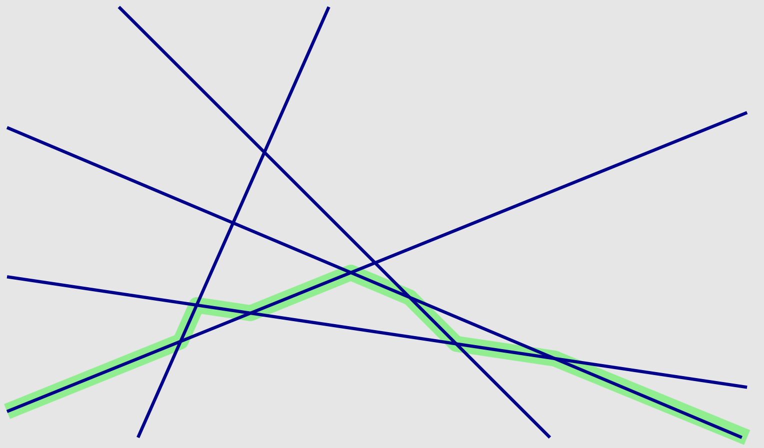
single cell



upper envelope



zone

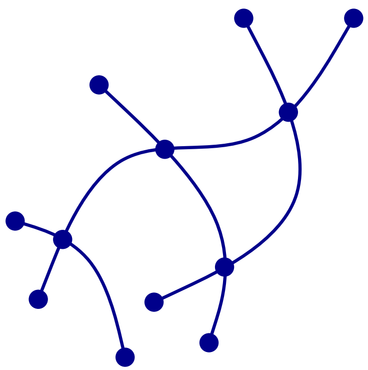


k-level

Theorem. Let S be a set of n simple curves such that any two curves intersect at most s times, where S is a fixed constant. Then the complexity of the full arrangement $\mathcal{A}(S)$ is $O(n^2)$.

Theorem. Let S be a set of n simple curves such that any two curves intersect at most s times, where S is a fixed constant. Then the complexity of the full arrangement $\mathcal{A}(S)$ is $O(n^2)$.

Proof.

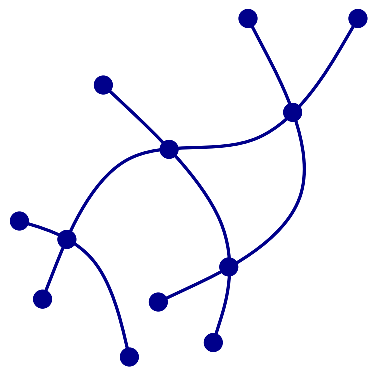


Assume curves are finite.

- number of vertices
- number of edges
- number of faces

Theorem. Let S be a set of n simple curves such that any two curves intersect at most s times, where S is a fixed constant. Then the complexity of the full arrangement $\mathcal{A}(S)$ is $O(n^2)$.

Proof.

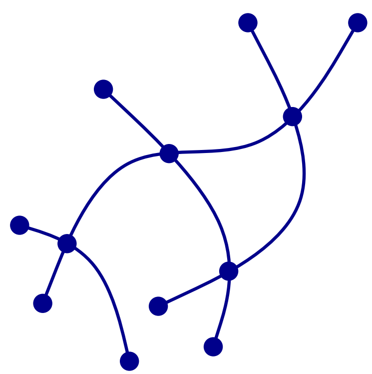


Assume curves are finite.

- number of vertices $|V| \leqslant 2n + s \cdot \binom{n}{2} = O(n^2)$
- number of edges
- number of faces

Theorem. Let S be a set of n simple curves such that any two curves intersect at most s times, where S is a fixed constant. Then the complexity of the full arrangement $\mathcal{A}(S)$ is $O(n^2)$.

Proof.

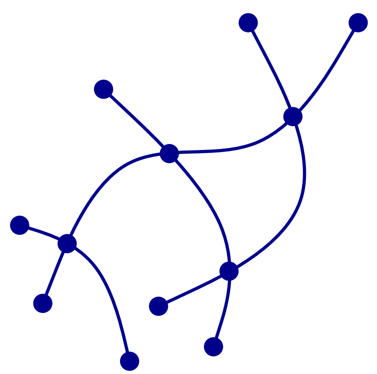


Assume curves are finite.

- number of vertices $|V| \leqslant 2n + s \cdot \binom{n}{2} = O(n^2)$
- number of edges $|E| \leqslant n \cdot (s(n-1)+1) = O(n^2)$
- number of faces

Theorem. Let S be a set of n simple curves such that any two curves intersect at most s times, where S is a fixed constant. Then the complexity of the full arrangement $\mathcal{A}(S)$ is $O(n^2)$.

Proof.



Assume curves are finite.

- number of vertices $|V| \leqslant 2n + s \cdot \binom{n}{2} = O(n^2)$
- number of edges $|E| \leqslant n \cdot (s(n-1)+1) = O(n^2)$
- number of faces

Euler's formula:

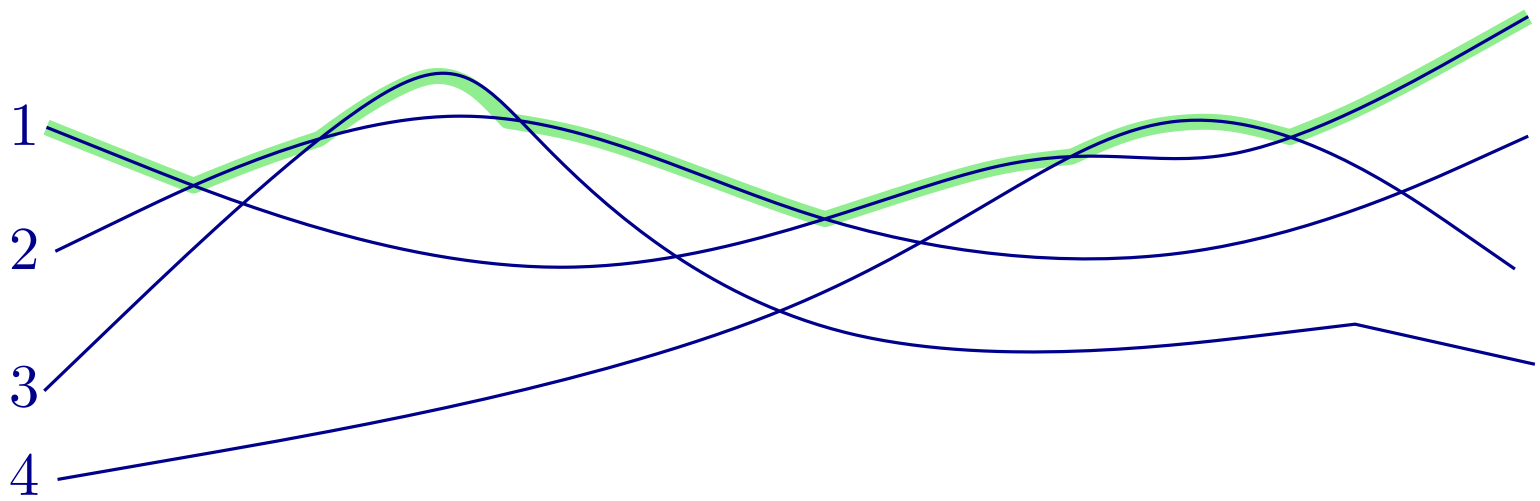
$|V| - |E| + |F| = 2$

■

Theorem. Let S be a set of n infinite x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_s(n))$.

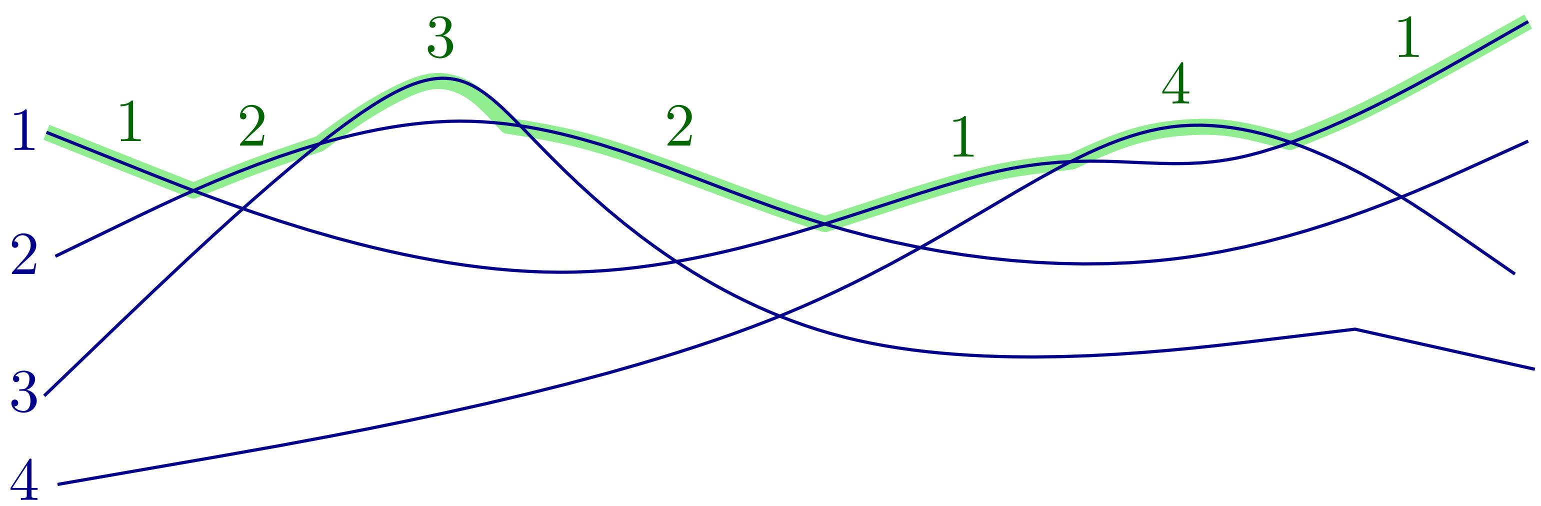
Theorem. Let S be a set of n **infinite** x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_s(n))$.

Proof.



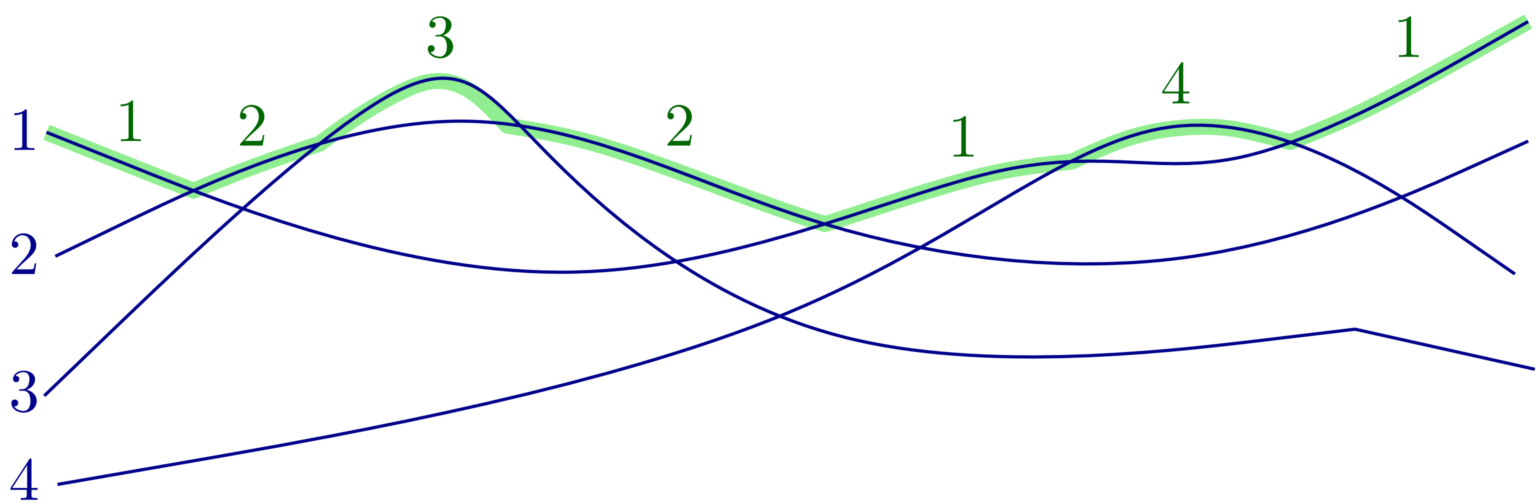
Theorem. Let S be a set of n **infinite** x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_s(n))$.

Proof.

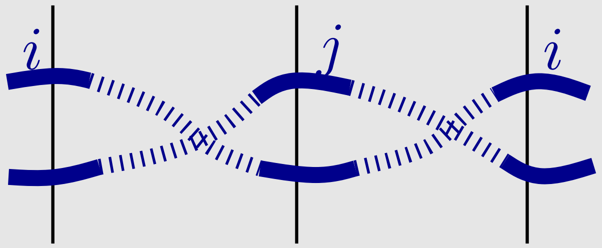


Theorem. Let S be a set of n **infinite** x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_s(n))$.

Proof.

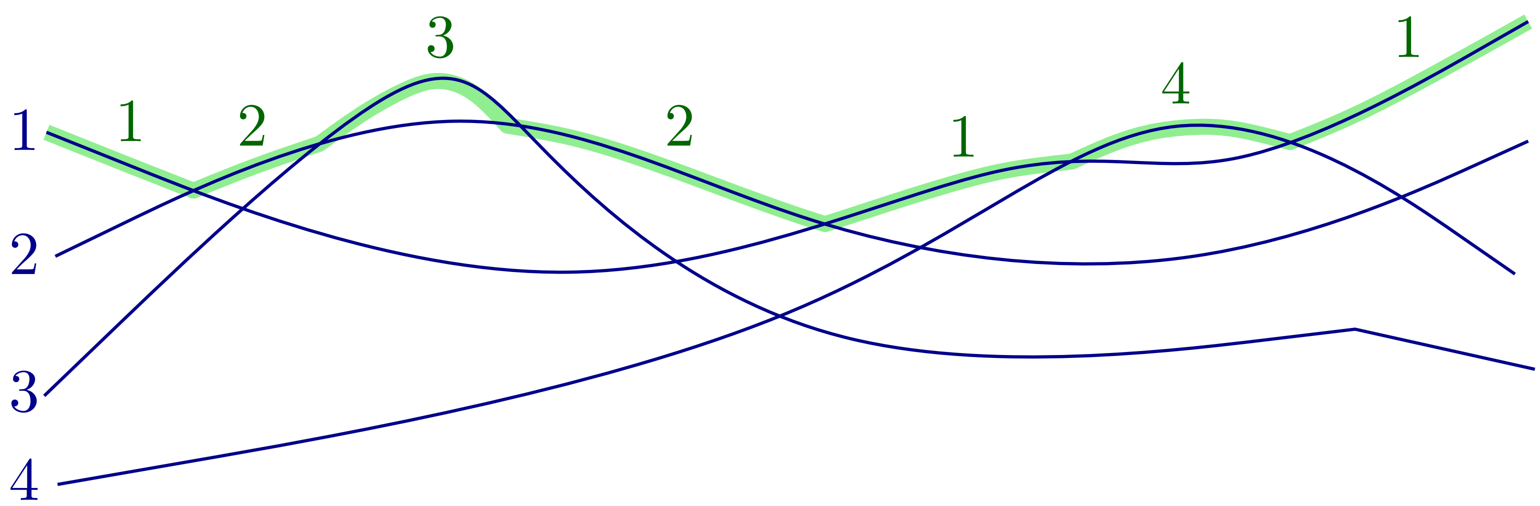


alternating sequence of length t implies $t-1$ intersections



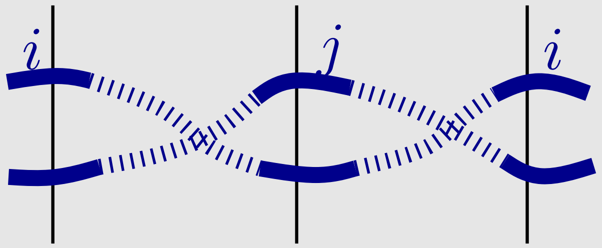
Theorem. Let S be a set of n **infinite** x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_s(n))$.

Proof.



we cannot have alternating sequence of length $s+2$
$\Longrightarrow$ DS(n, s)-sequence

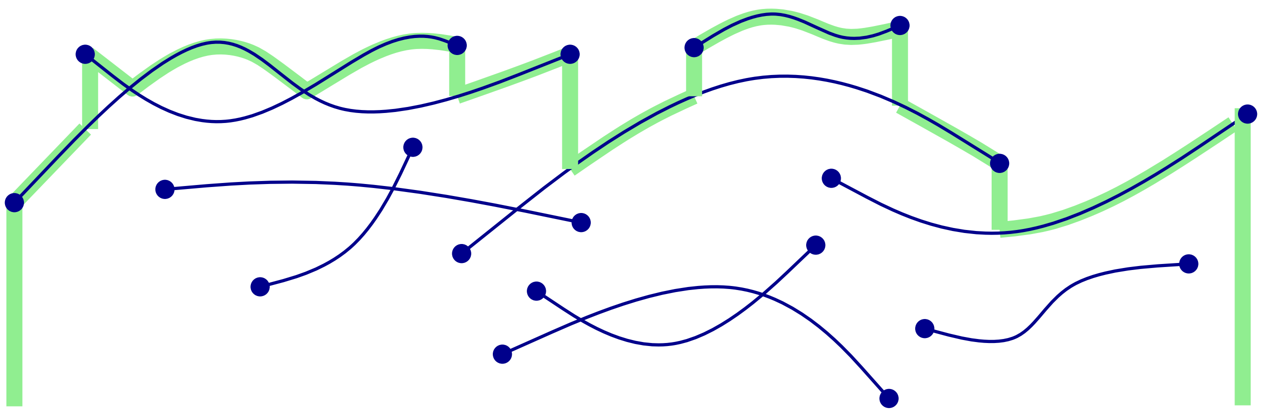
alternating sequence of length t implies $t-1$ intersections



Theorem. Let S be a set of n x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_{s+2}(n))$.

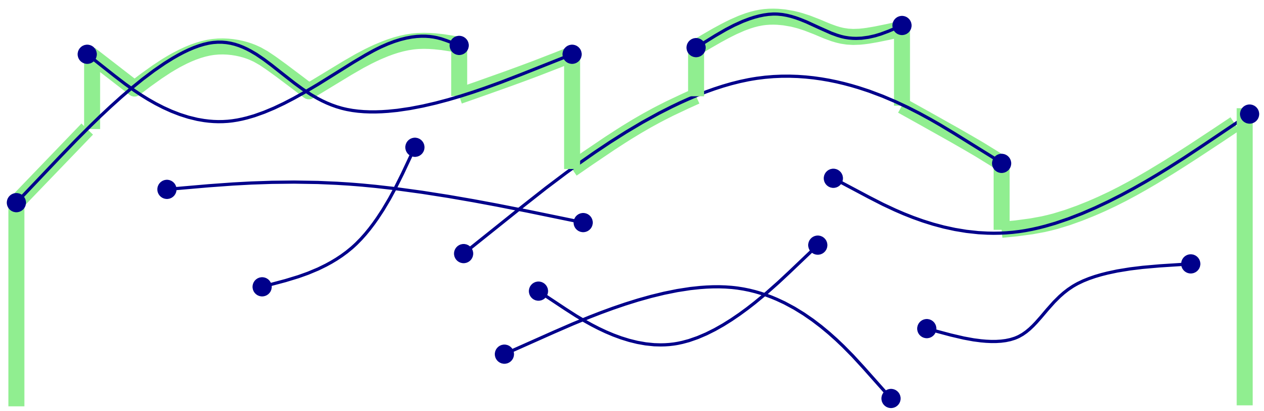
Theorem. Let S be a set of n x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_{s+2}(n))$.

Proof.

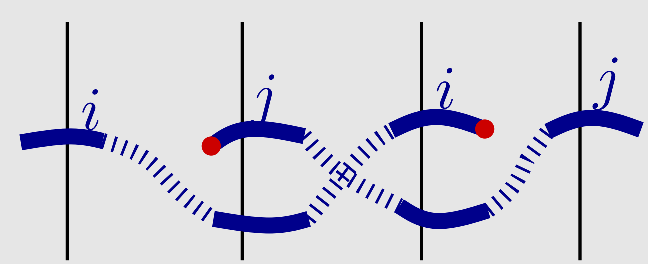


Theorem. Let S be a set of n x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_{s+2}(n))$.

Proof.

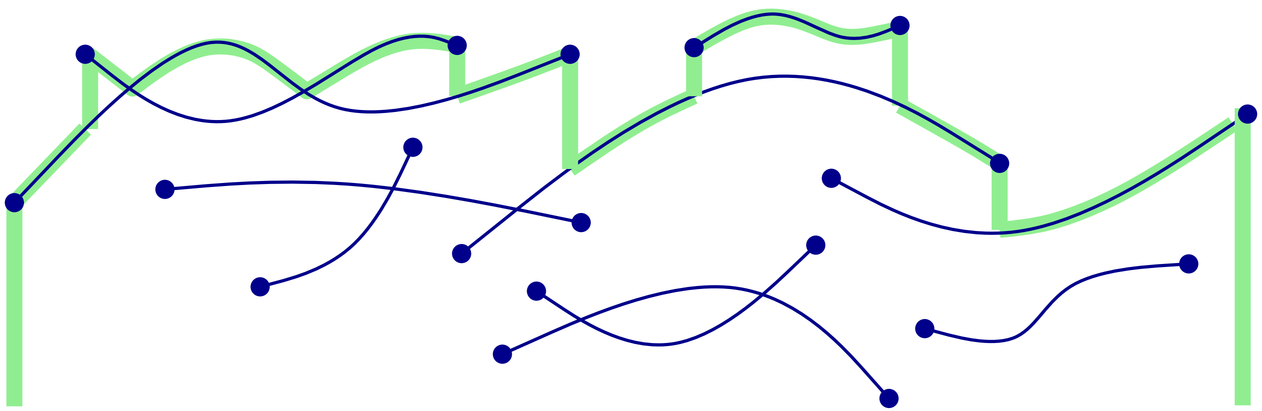


alternating sequence of length t implies $t-3$ intersections



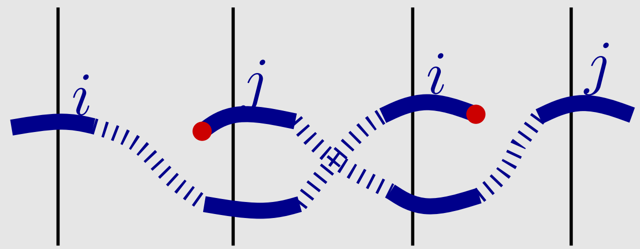
Theorem. Let S be a set of n x-monotone curves such that any two curves intersect at most s times. Then the maximum complexity of the upper envelope of S is $O(\mathrm{DS}_{s+2}(n))$.

Proof.



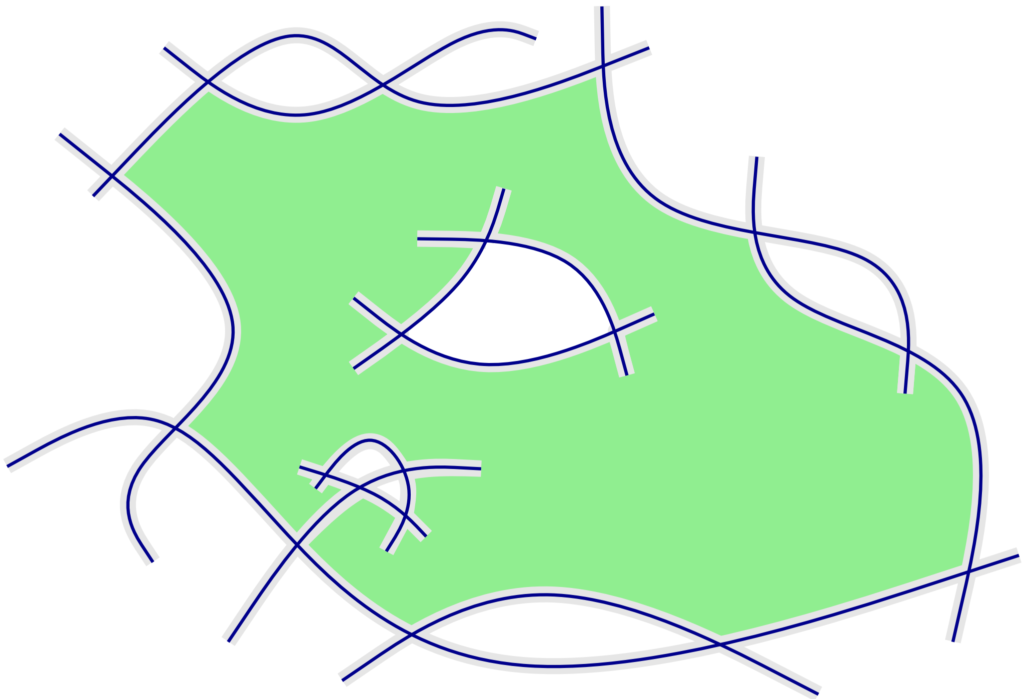
we cannot have alternating sequence of length $s+4$
$\Longrightarrow$ DS(n, $s+2$)-sequence

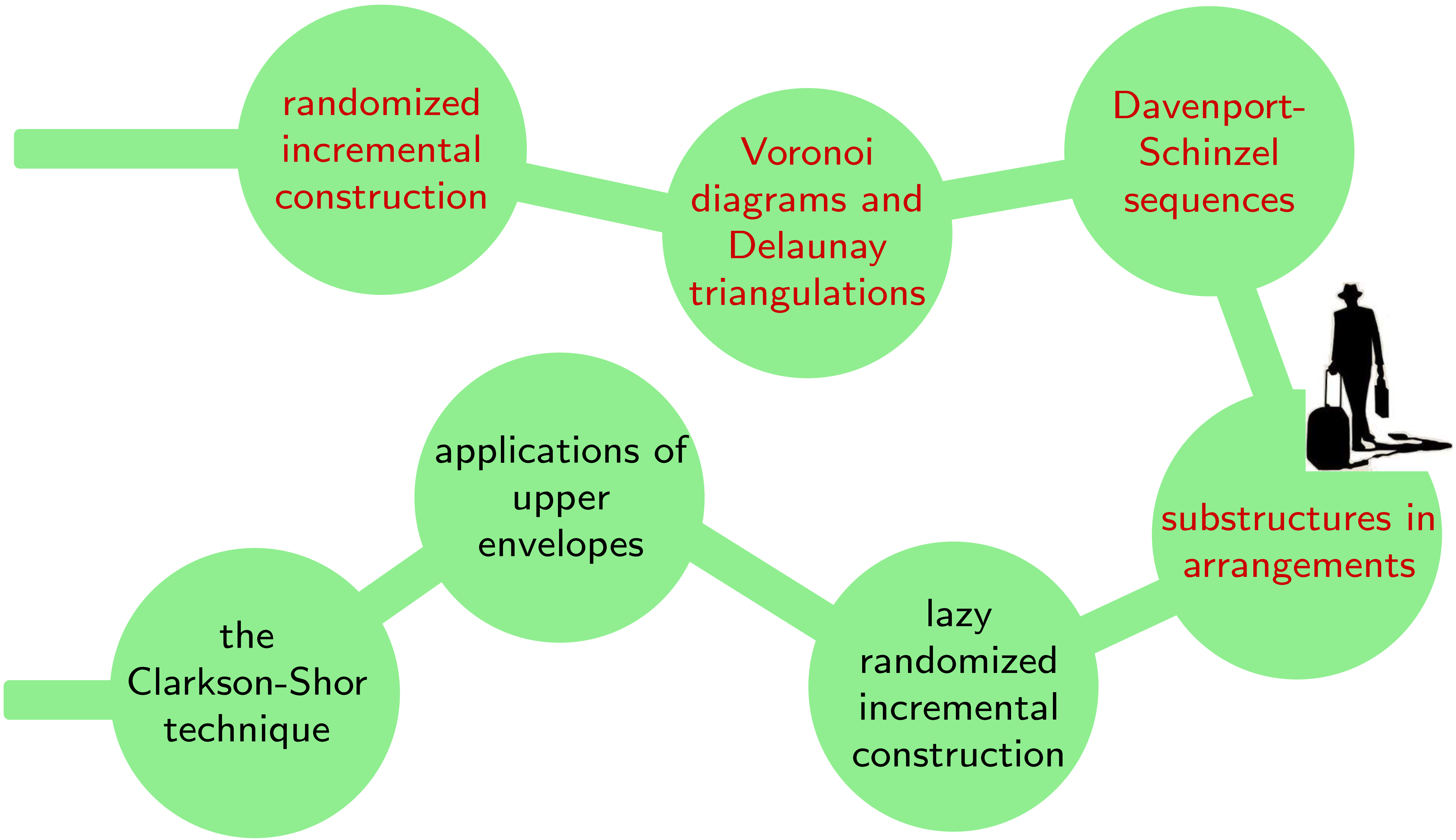
alternating sequence of length t implies $t-3$ intersections



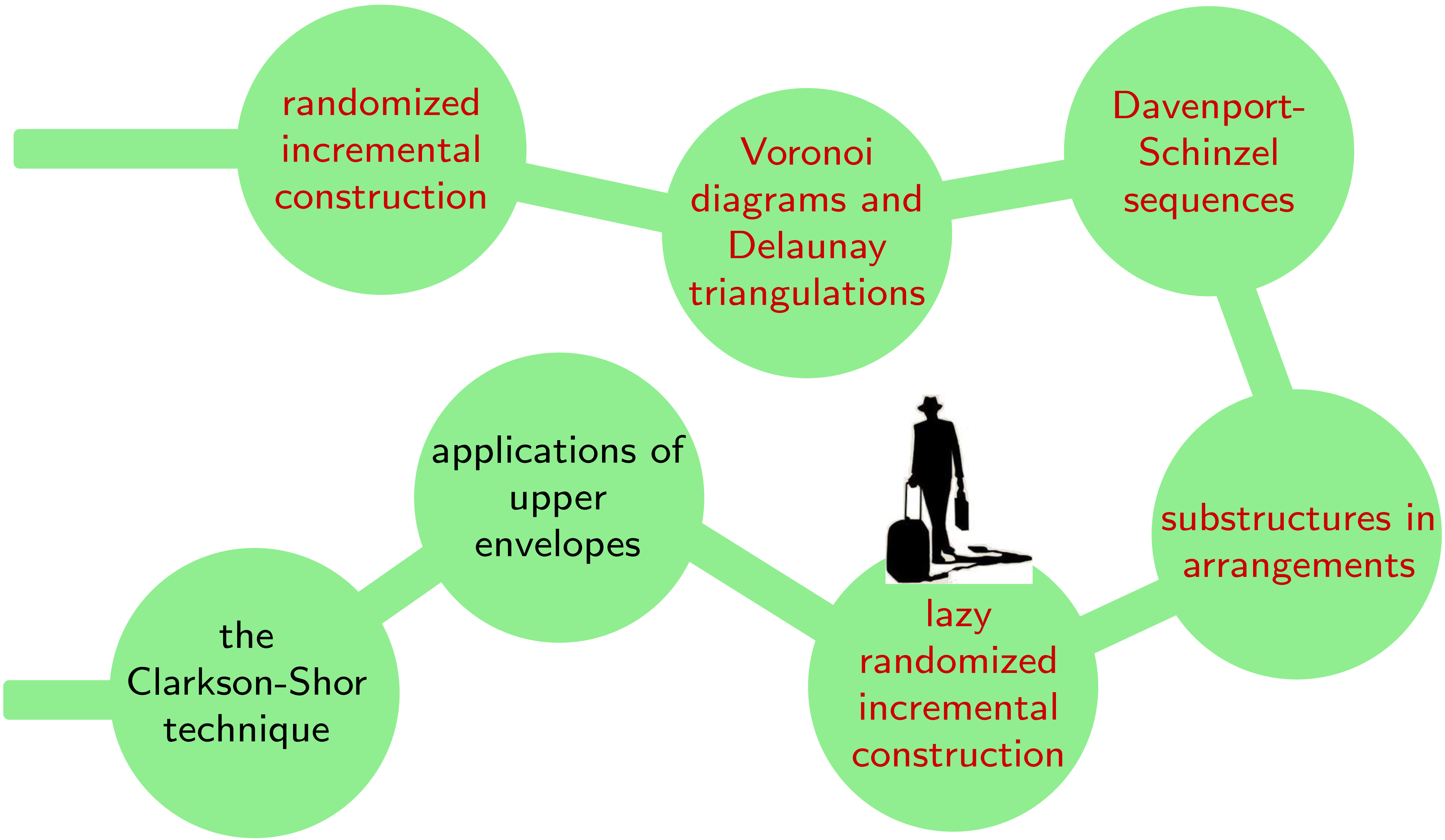
■

Theorem. Let S be a set of n curves in the plane such that any two curves intersect at most s times. Then the maximum complexity of a single cell of $\mathcal{A}(S)$ is $O(\mathrm{DS}_{s+2}(n))$.




randomized incremental construction
Voronoi diagrams and Delaunay triangulations
Davenport-Schinzel sequences
substructures in arrangements
lazy randomized incremental construction
applications of upper envelopes
the Clarkson-Shor technique

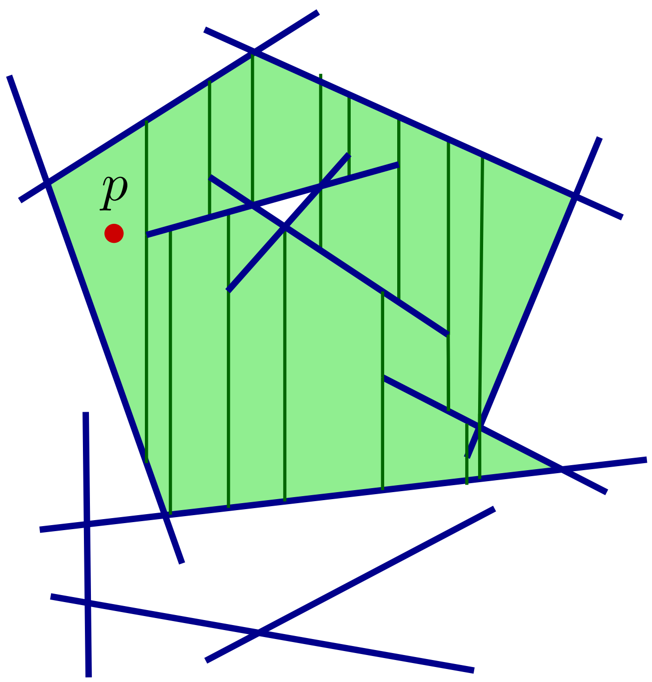
Course Overview



Computing a single cell with RIC?

Input: Set S of n segments in the plane, and a point p

Goal: Compute the face of $\mathcal{A}(S)$ containing p



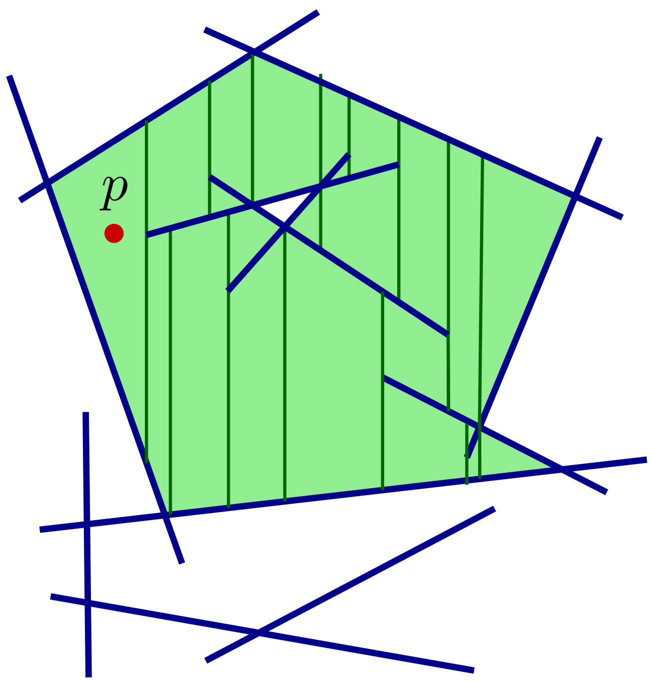
The RIC framework

- S = set of n input objects
- $\mathcal{C}(S)$ = set of configurations defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Computing a single cell with RIC?

Input: Set S of n segments in the plane, and a point p

Goal: Compute the face of $\mathcal{A}(S)$ containing p



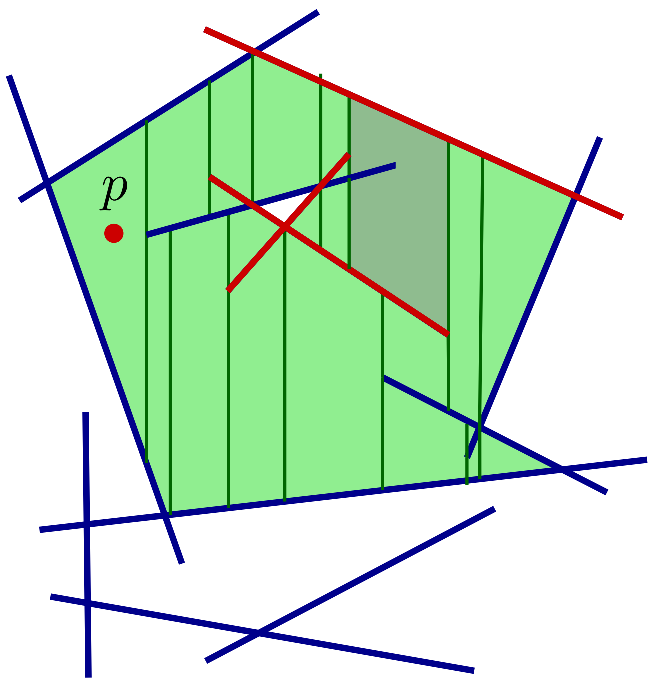
The RIC framework

- S = set of n input segments
- $\mathcal{C}(S)$ = set of trapezoids defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Computing a single cell with RIC?

Input: Set S of n segments in the plane, and a point p

Goal: Compute the face of $\mathcal{A}(S)$ containing p

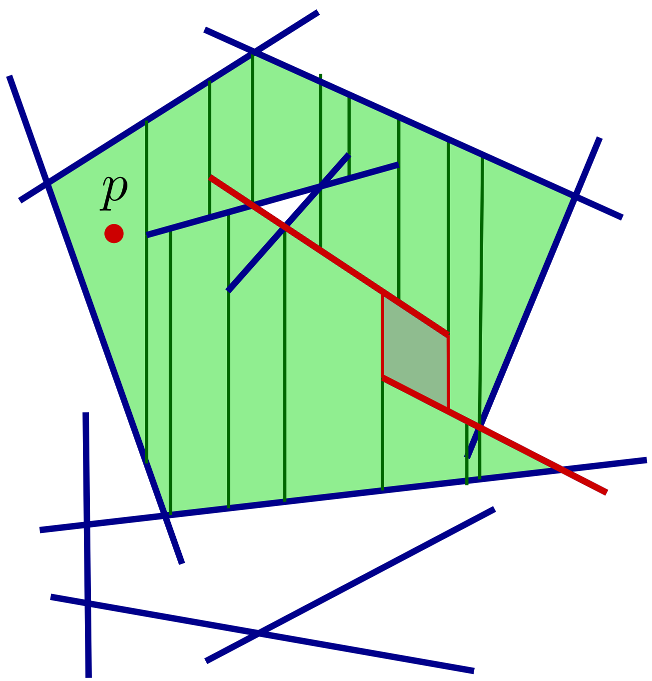


The RIC framework

- S = set of n input segments
- $\mathcal{C}(S)$ = set of trapezoids defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) = \{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Input: Set S of n segments in the plane, and a point p

Goal: Compute the face of $\mathcal{A}(S)$ containing p

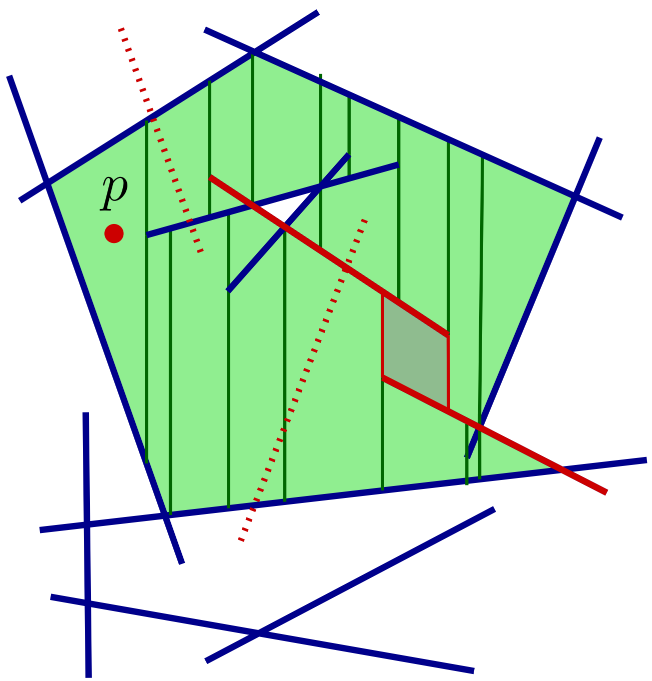


The RIC framework

- S = set of n input segments
- $\mathcal{C}(S)$ = set of trapezoids defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$ size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$ $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

Input: Set S of n segments in the plane, and a point p

Goal: Compute the face of $\mathcal{A}(S)$ containing p



The RIC framework

- S = set of n input segments
- $\mathcal{C}(S)$ = set of trapezoids defined by S
  - $D(\Delta) \subset S$ = defining set of $\Delta \in \mathcal{C}(S)$
    size bounded by fixed constant
  - $K(\Delta) \subset S$ = conflict list of $\Delta \in \mathcal{C}(S)$ **?**
- Goal: Compute $\mathcal{C}_{\text{act}}(S) =$
  $\{\Delta \in \mathcal{C}(S) : D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}$

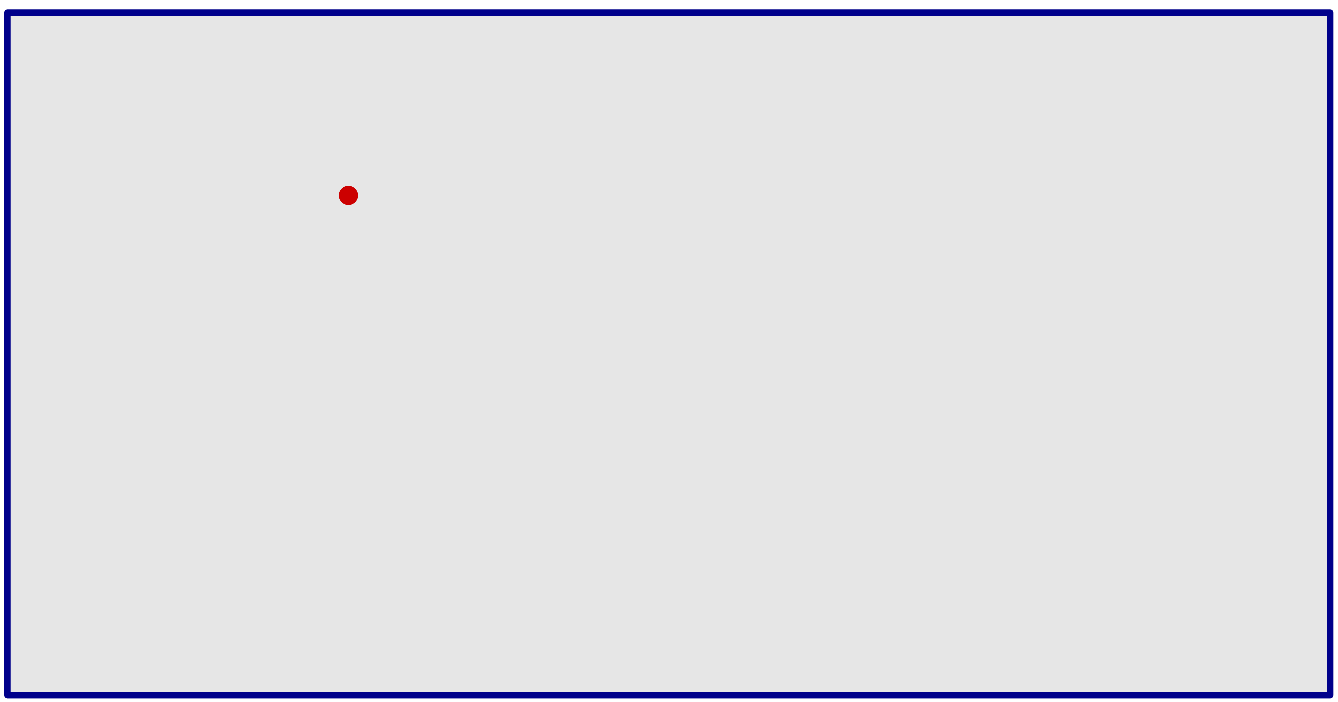
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.

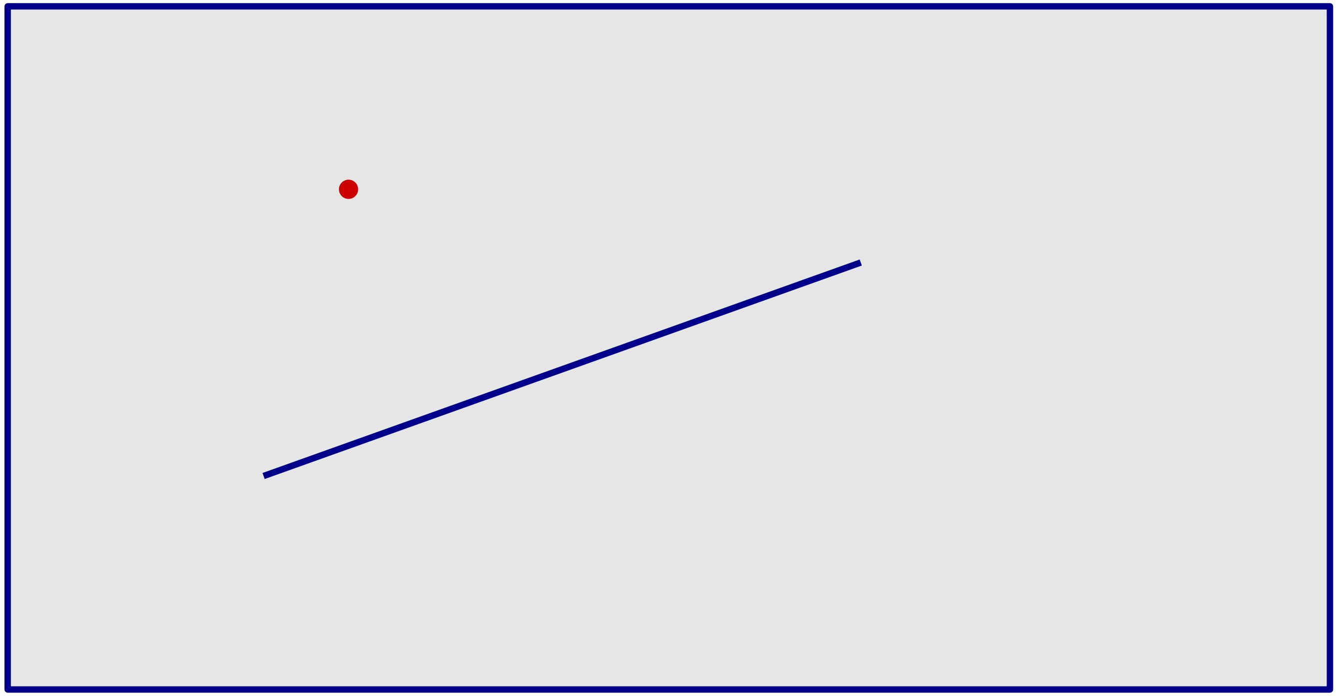
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

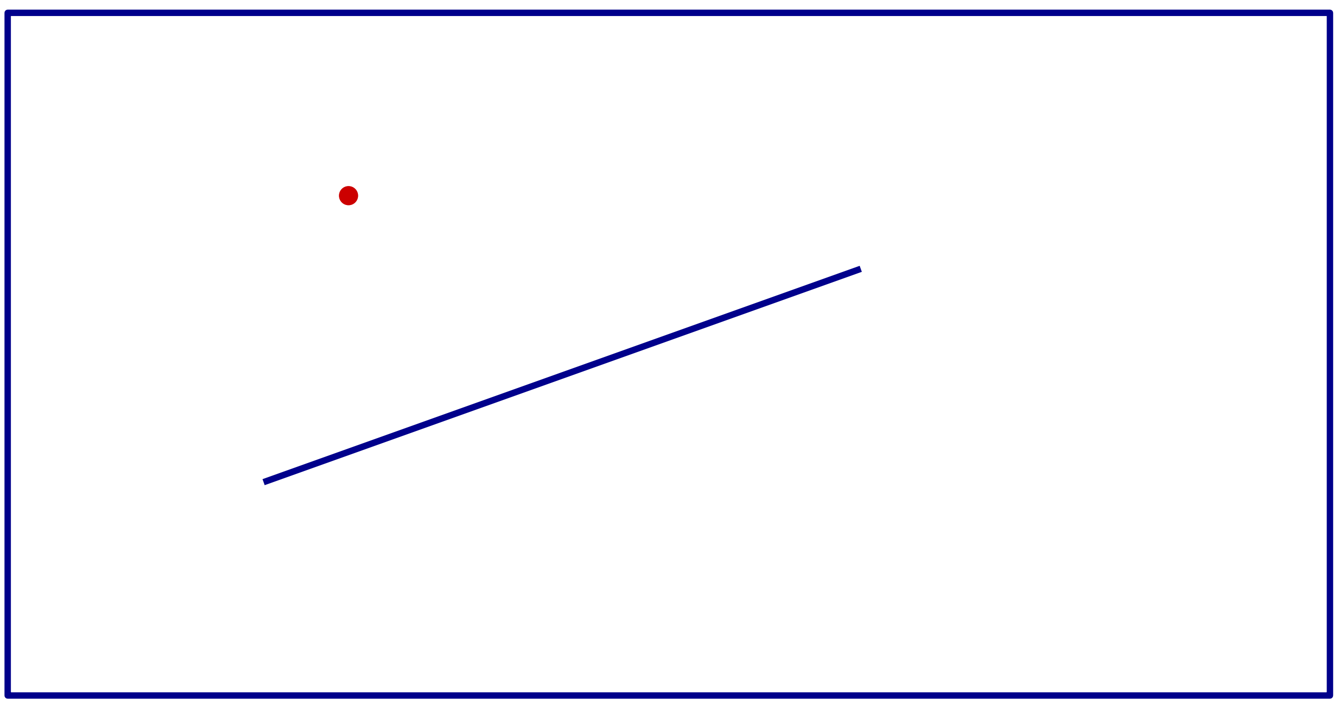
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



Lazy Randomized Incremental Construction

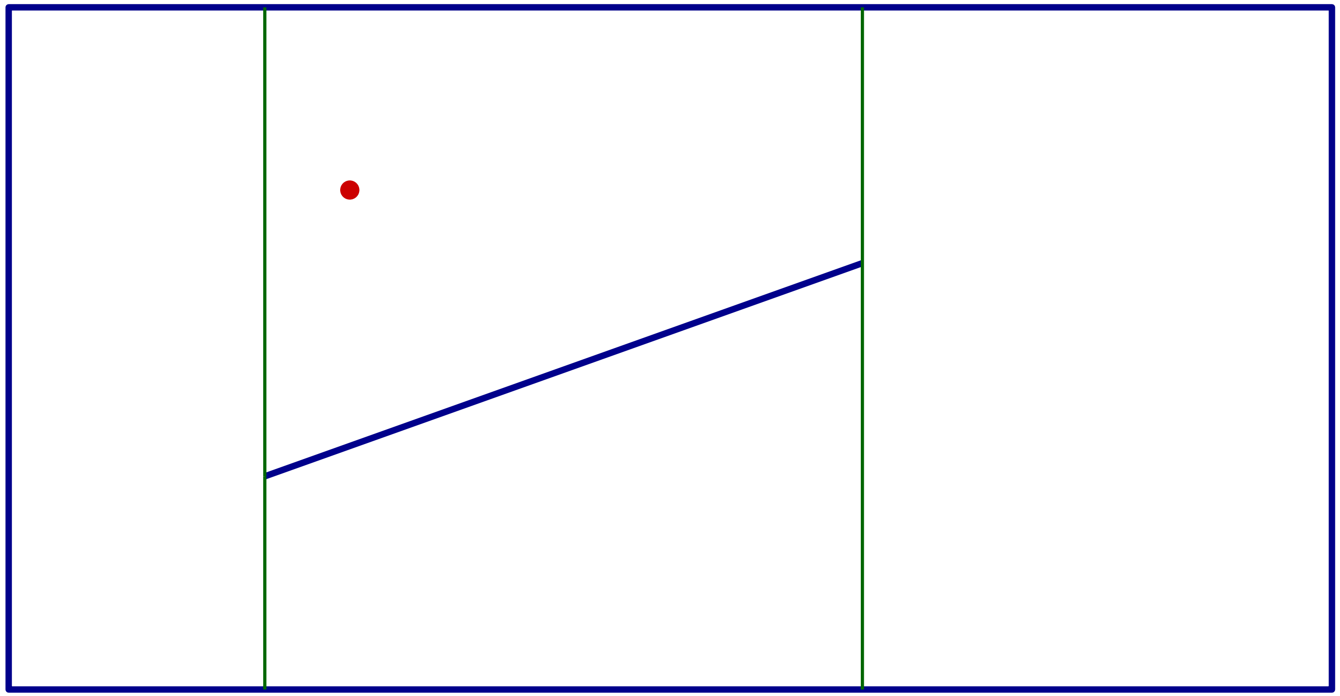
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

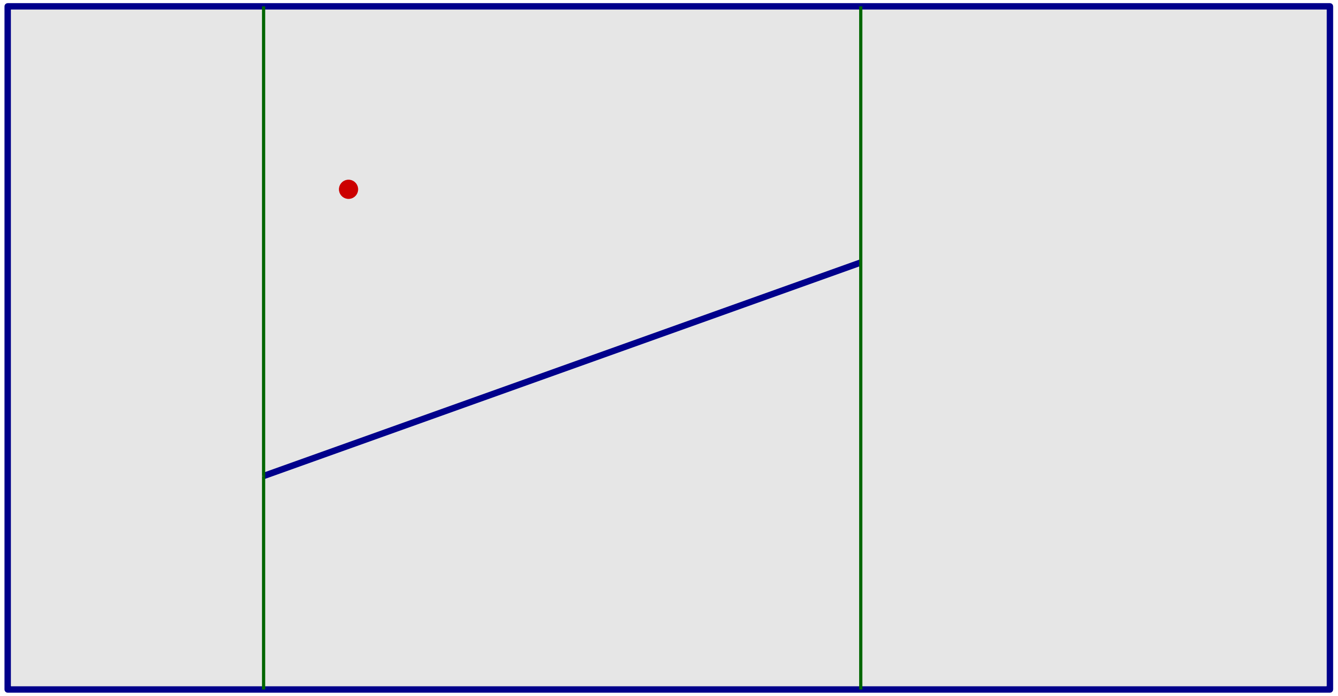
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



Lazy Randomized Incremental Construction

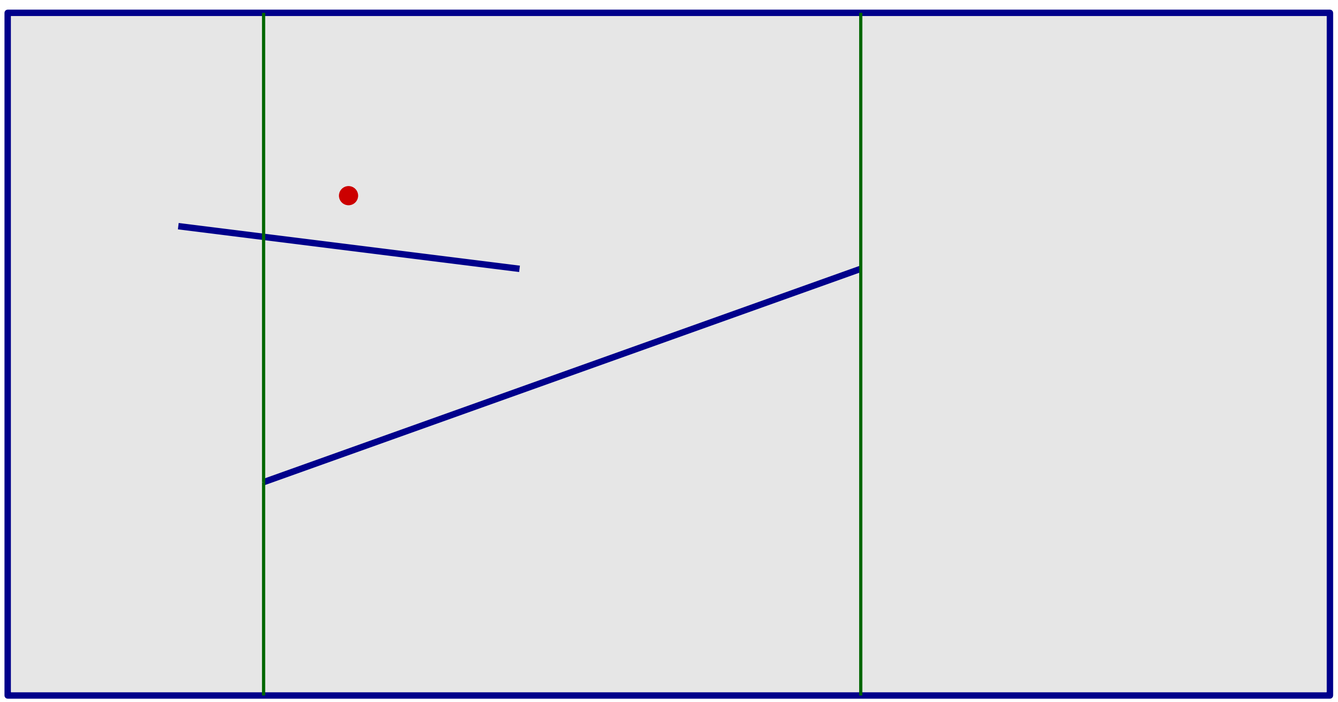
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



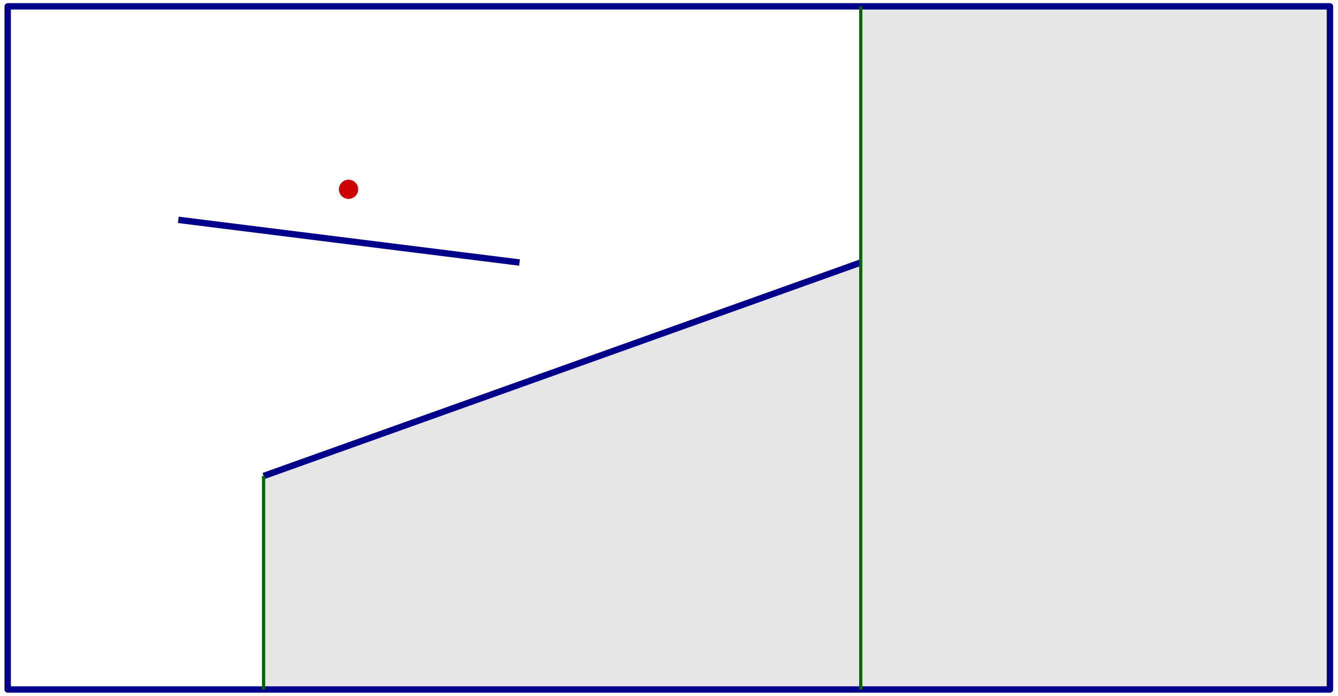
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



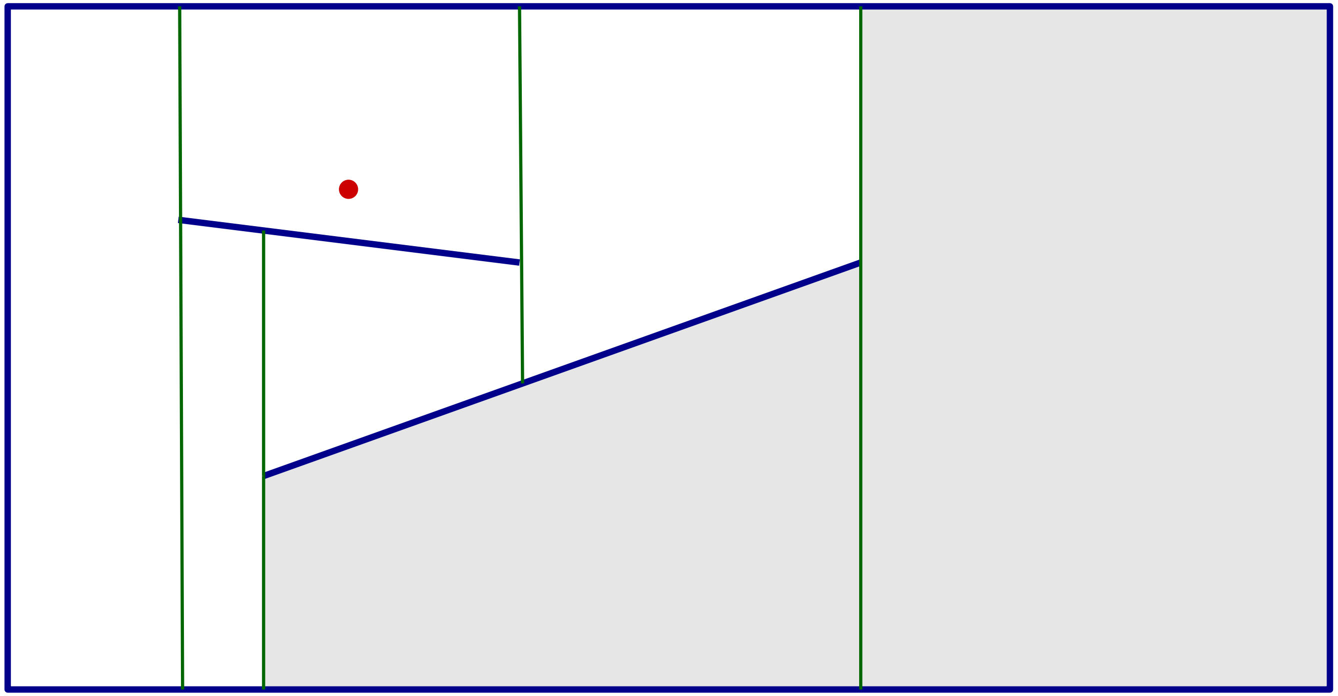
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



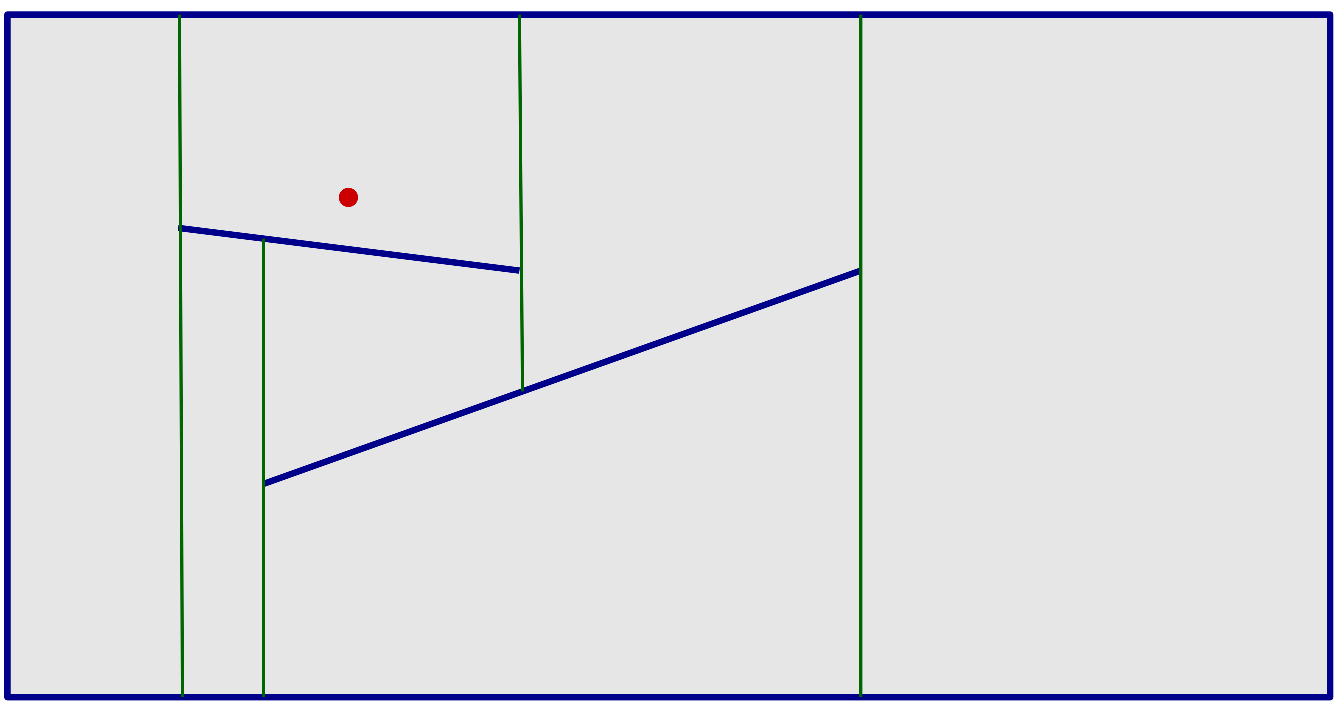
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, . . . perform a clean-up step.



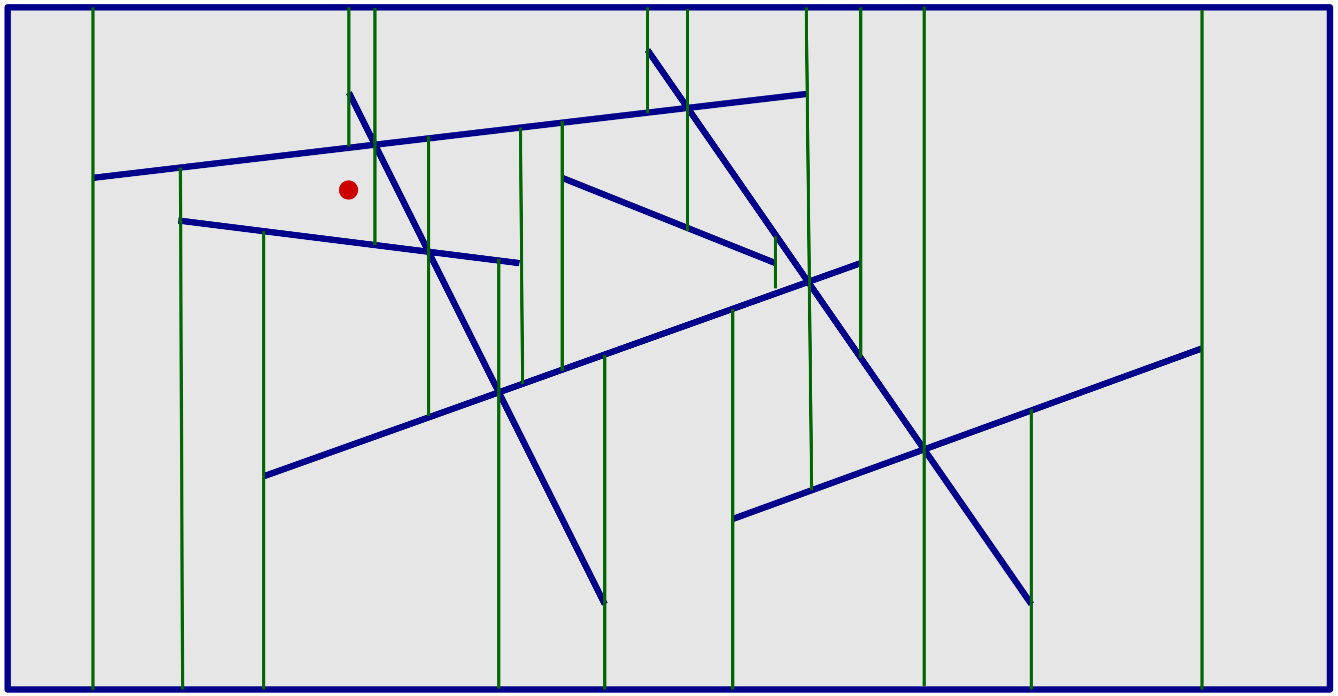
Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

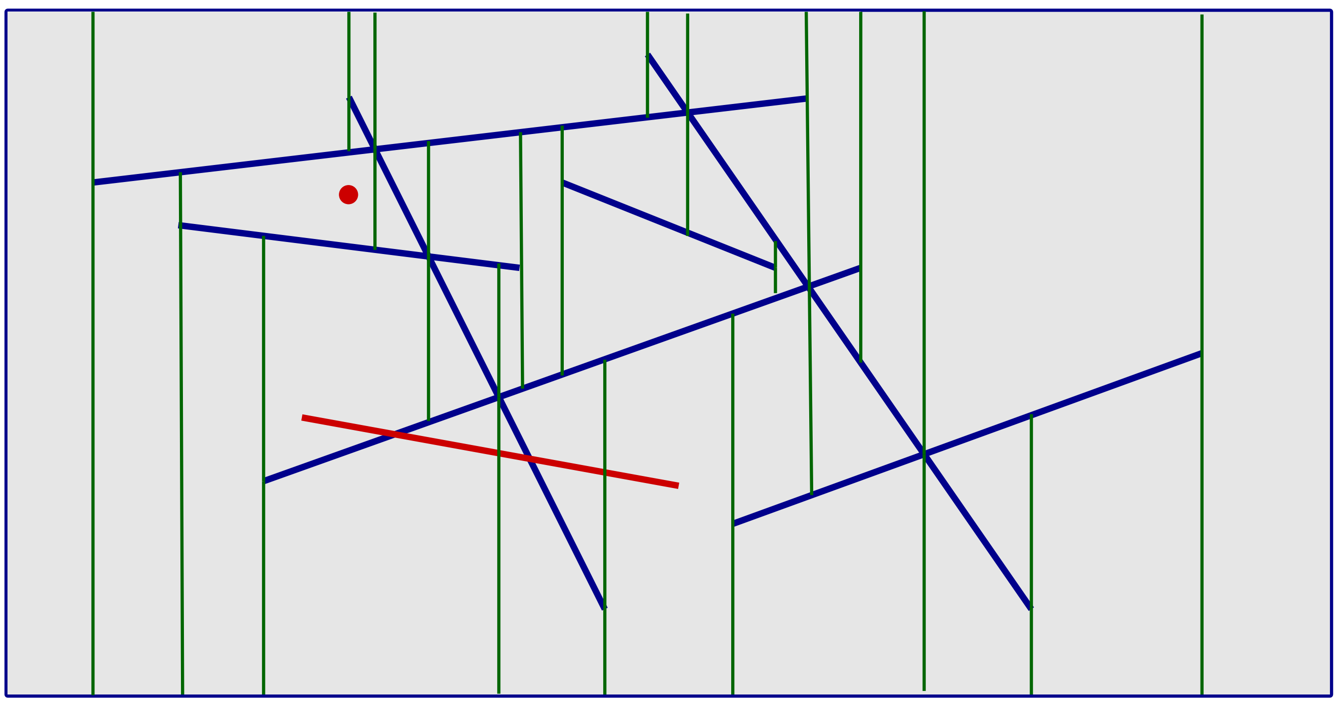
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



after 7 iterations

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.

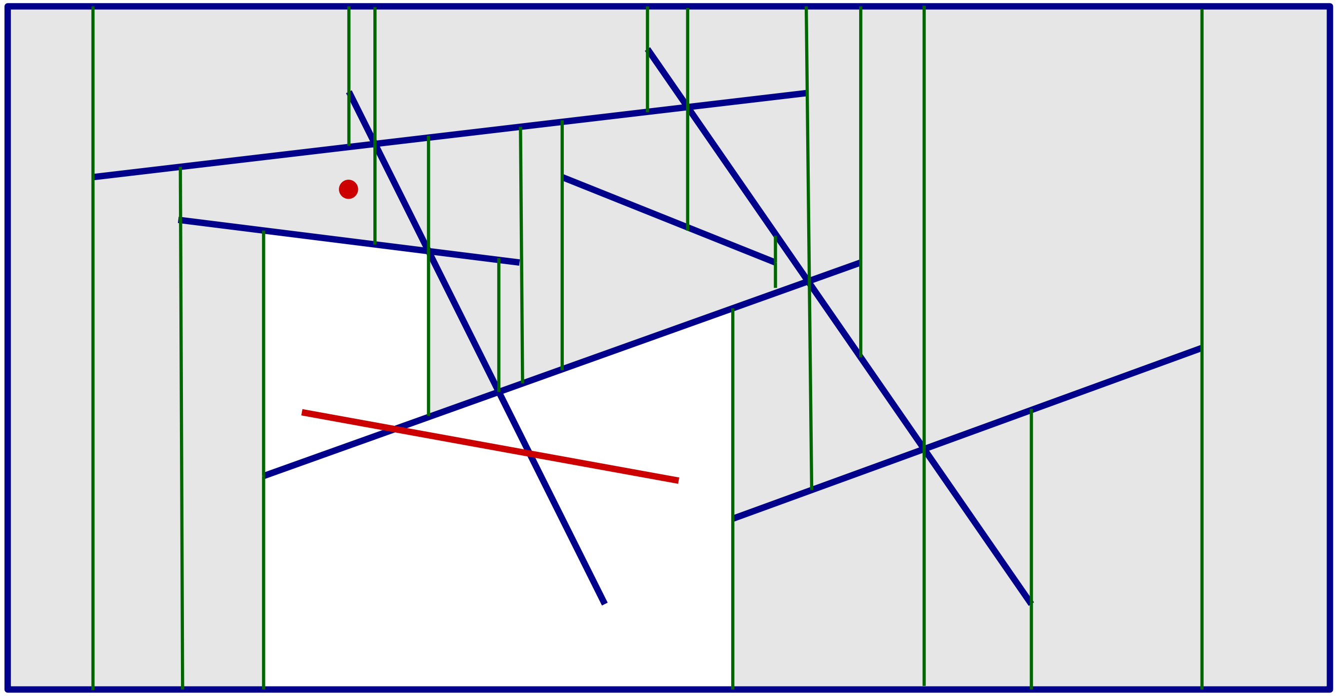


the 8-th iteration

Lazy Randomized Incremental Construction

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

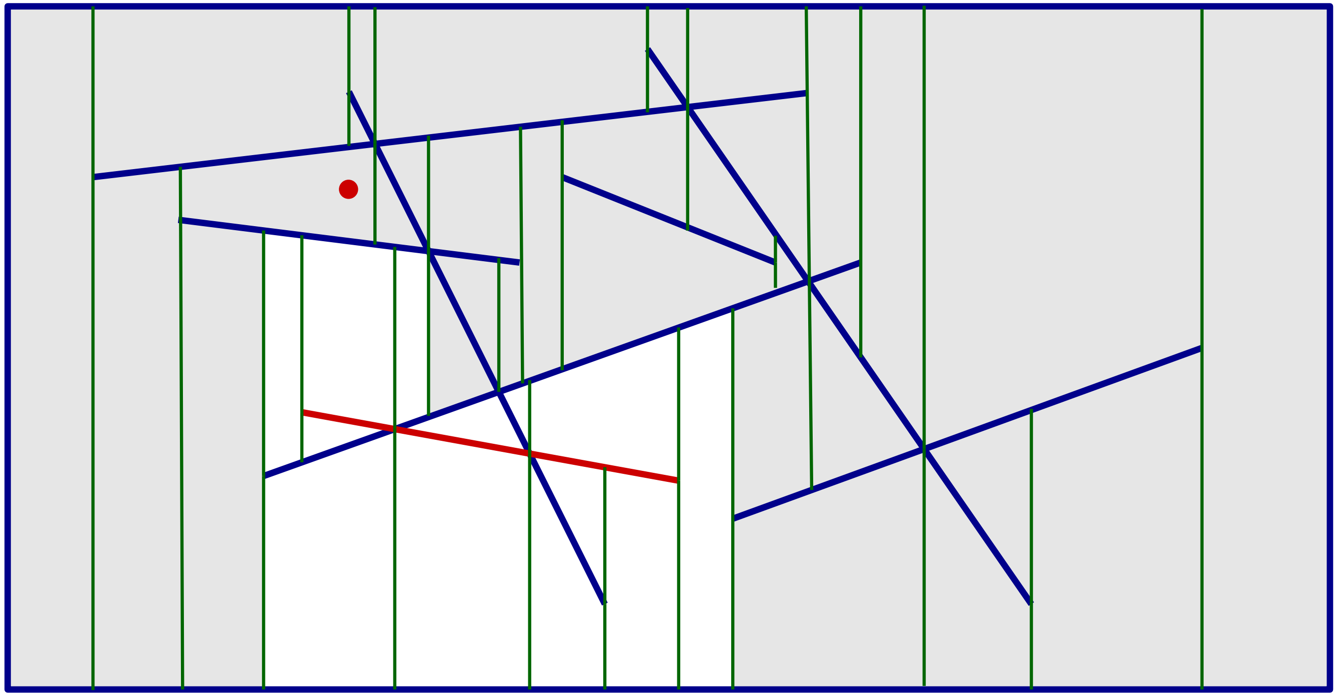
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



the 8-th iteration

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

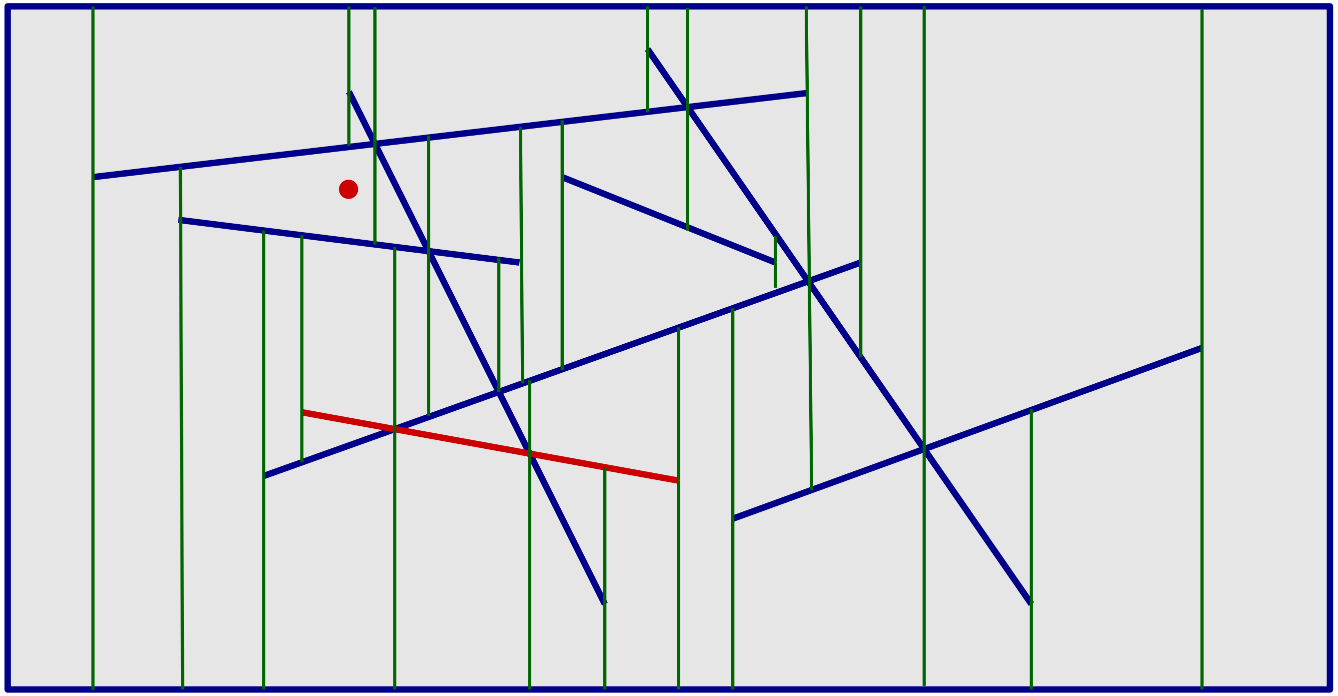
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, . . . perform a clean-up step.



the 8-th iteration

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

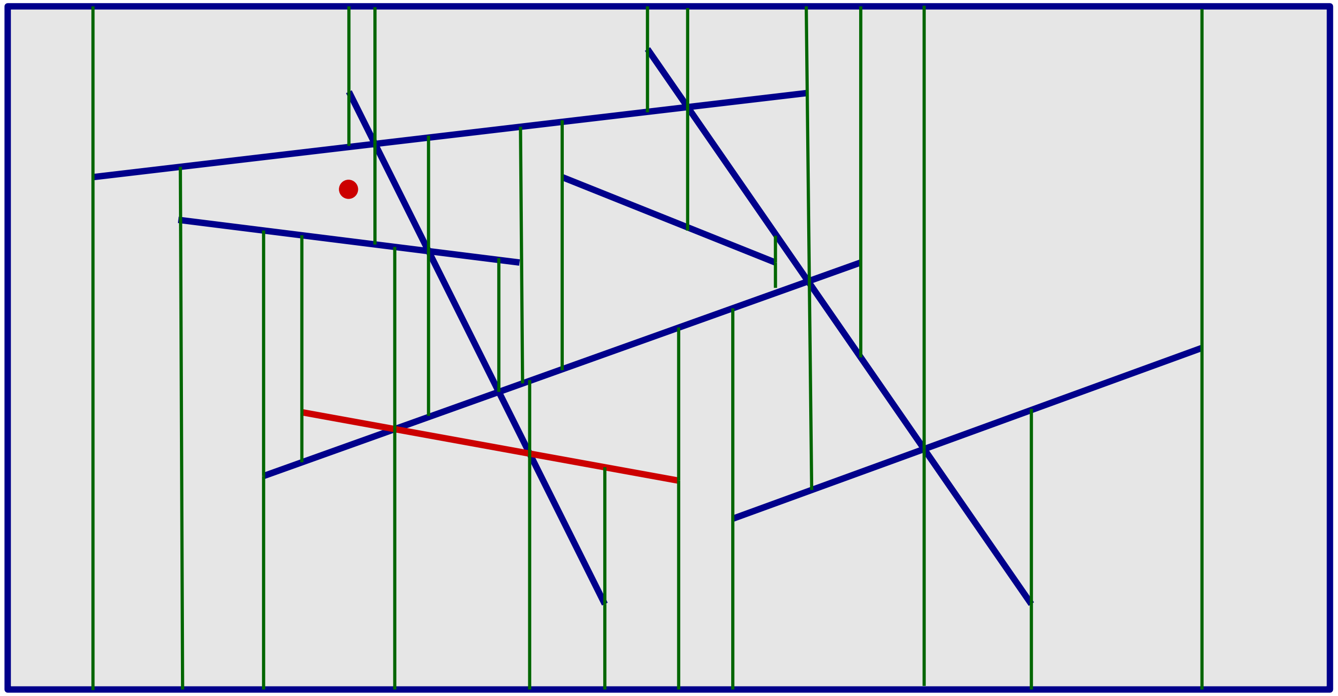
- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.



the 8-th iteration

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.

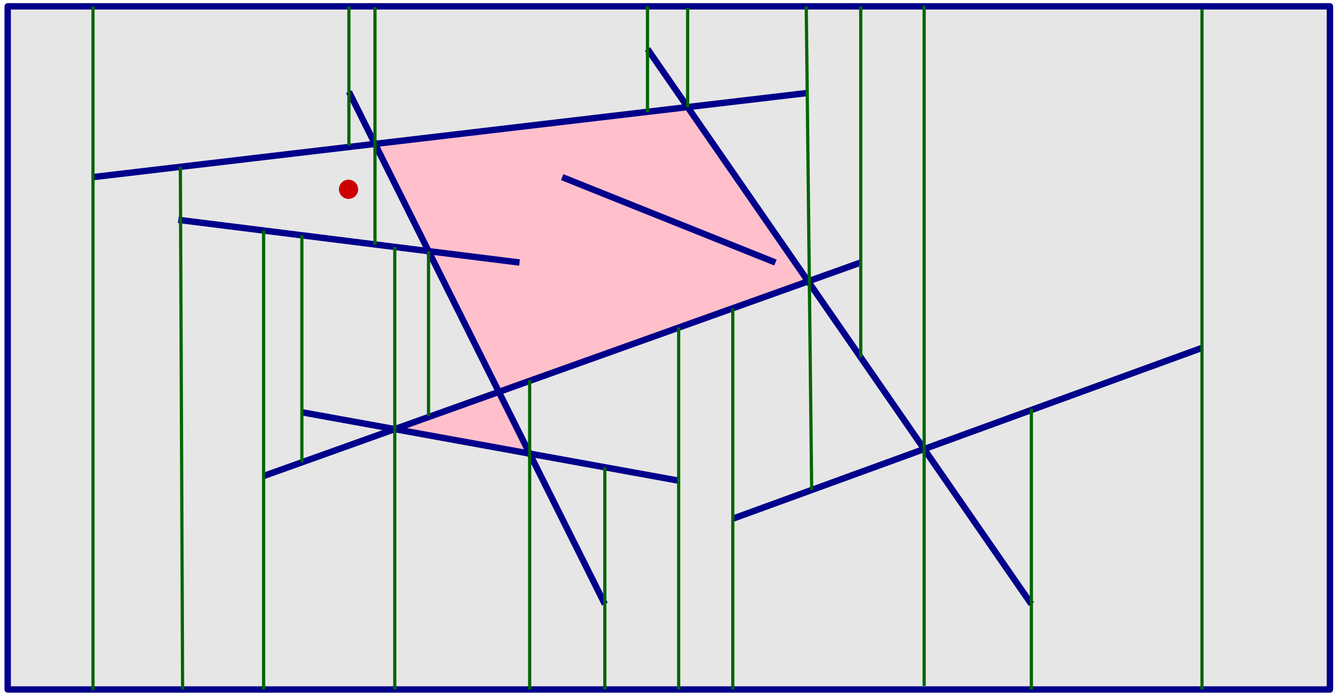


the 8-th iteration

clean-up phase:
remove trapezoids not in the cell of p

Theorem. Let S be a set of n line segments and let p be a point. Then the single cell of $\mathcal{A}(S)$ defined by p can be computed in $O(n\alpha(n)\log n)$ expected time.

- Apply standard RIC approach to construct trapezoidal decomposition of the whole arrangement.
- After iterations 1, 2, 4, 8, ... perform a clean-up step.

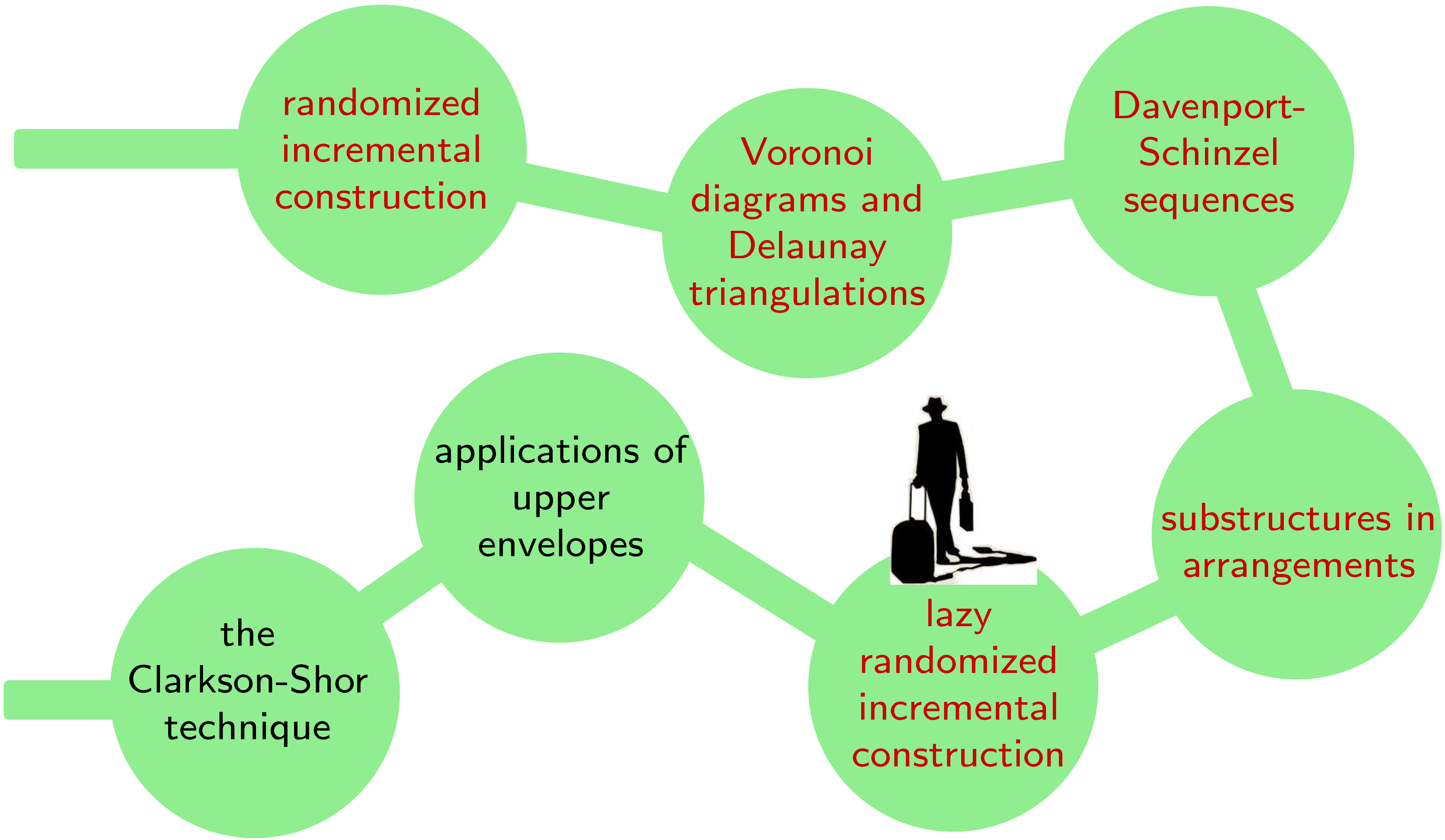


the 8-th iteration

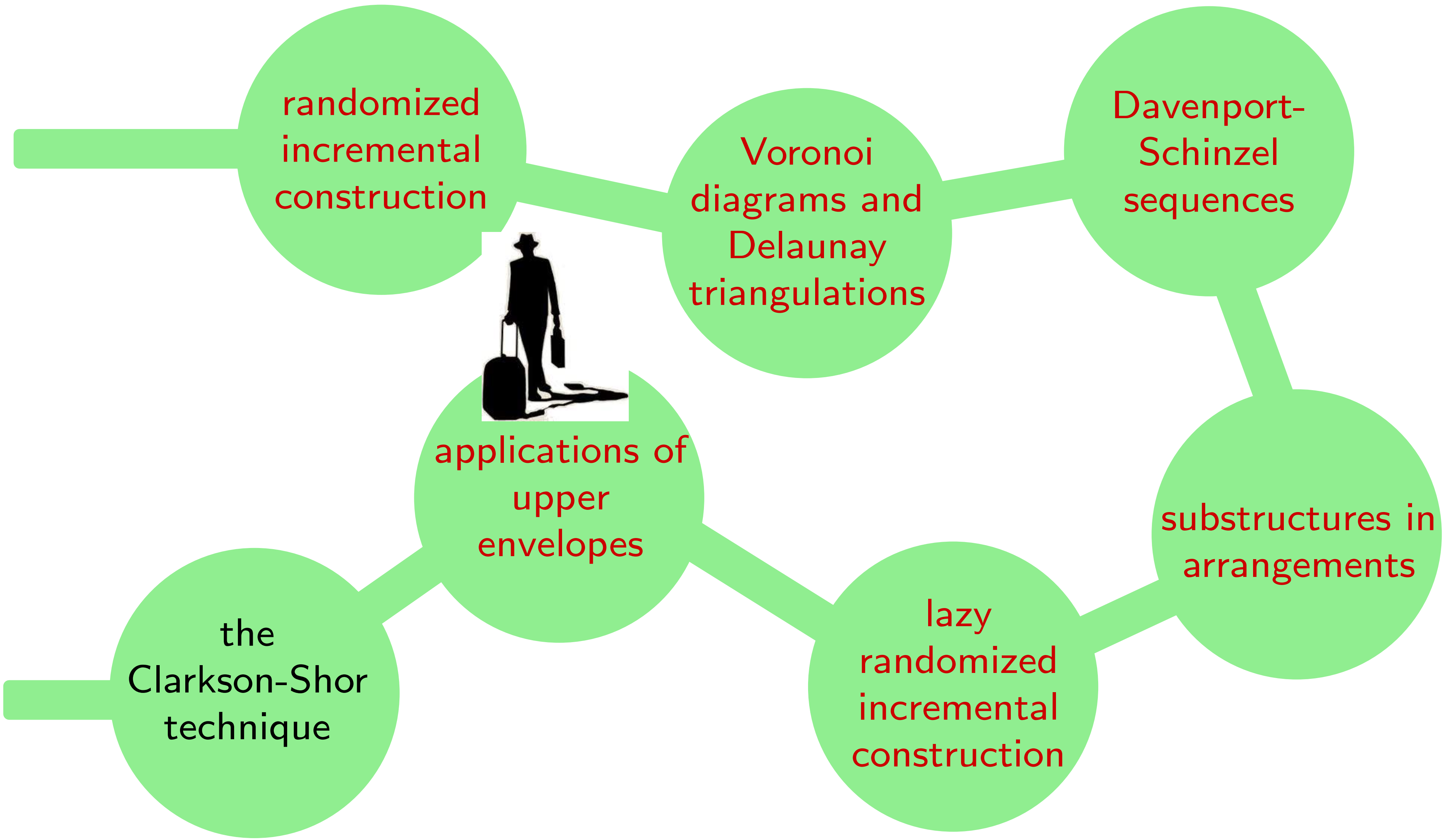
clean-up phase:
remove trapezoids not in the cell of p

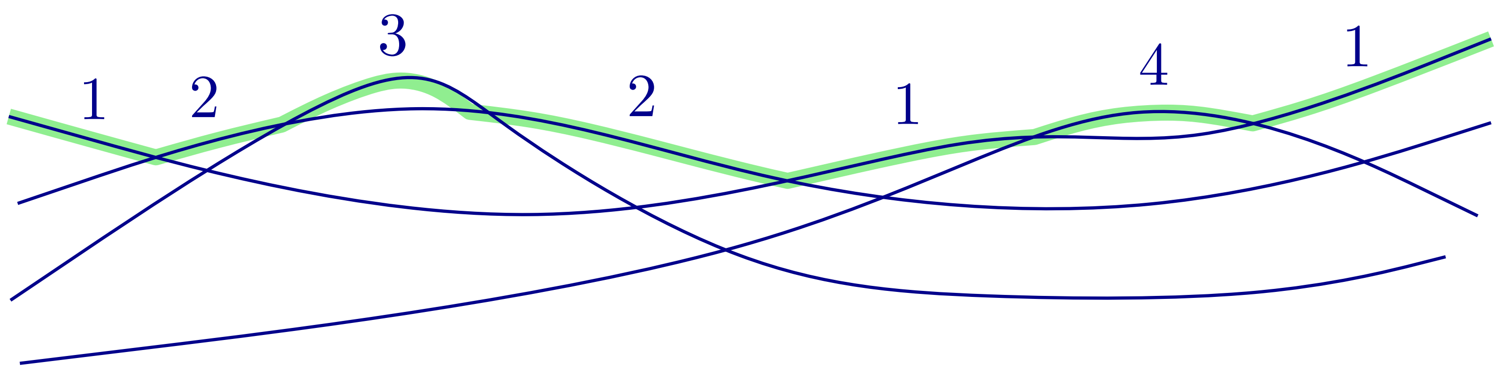
- Resulting algorithm has same performance bounds as when one could magically remove cells not in cell of p after each iteration
- Approach can also be formulated using abstract framework
- Can also be used to compute single cell in arrangement of triangles in $\mathbb{R}^3$, of zone of set of hyperplanes in $\mathbb{R}^d$, and more

Course Overview



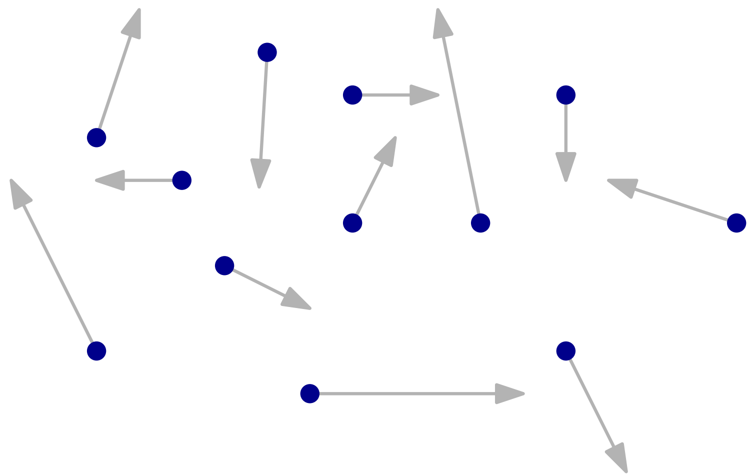
Course Overview





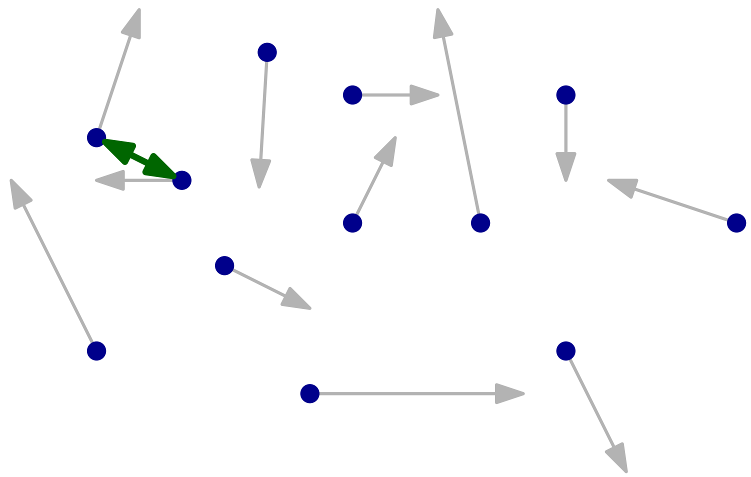
- n monotone curves with at most s intersections per pair
  - complexity of upper envelope is near-linear
  - infinite curves $O(\mathrm{DS}_s(n))$, finite curves $O(\mathrm{DS}_s(n))$
- n constant-degree algebraic surfaces in $\mathbb{R}^d$
  - complexity of upper envelope is $O(n^{d-1+\varepsilon})$, for any fixed $\varepsilon > 0$

P: set of n points in $\mathbb{R}^2$ that move linearly

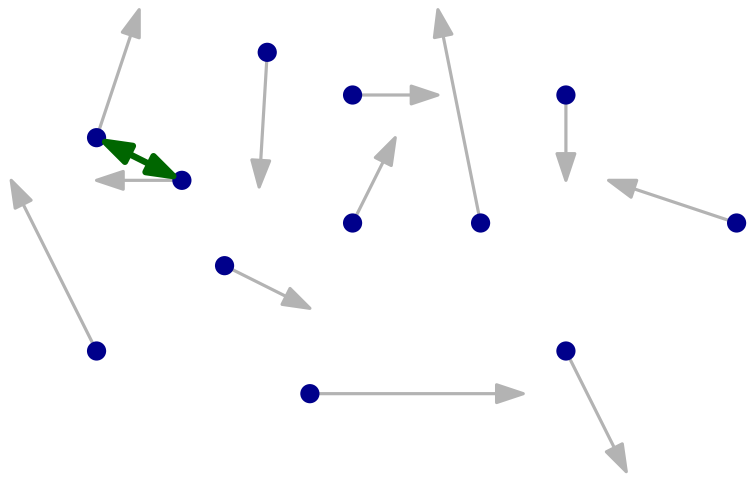


- How often can the closest pair change, in the worst case?
- How often can the convex hull change, in the worst case?
- How often can the Delaunay triangulation change, in the worst case?

How often can the closest pair change, in the worst case?

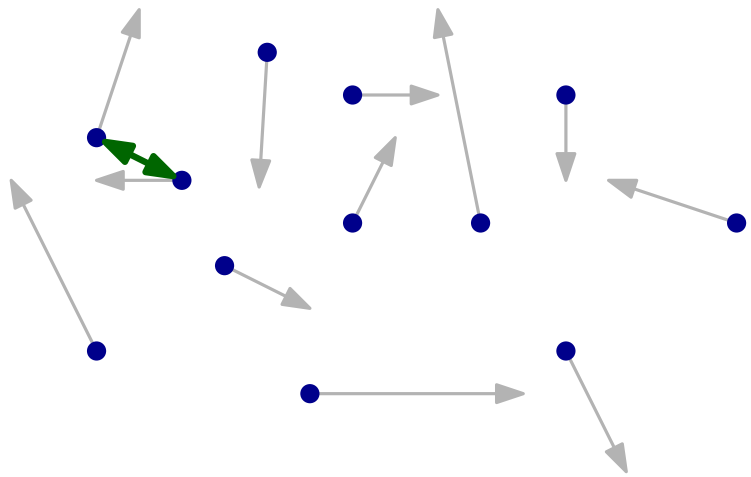


How often can the closest pair change, in the worst case?



Lower bound

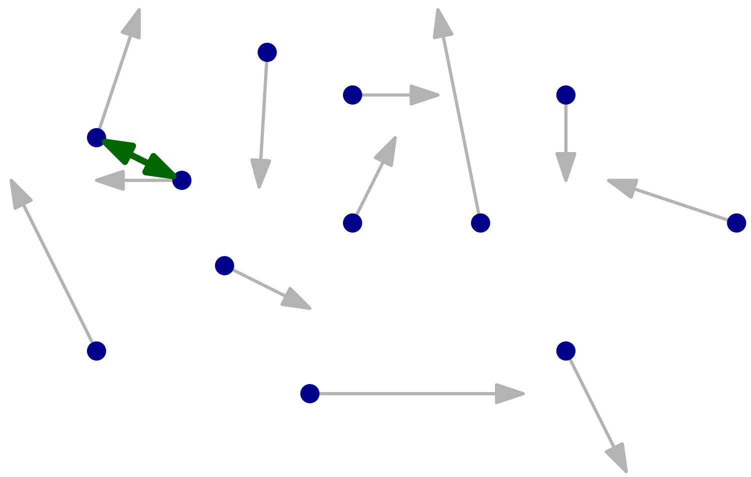
How often can the closest pair change, in the worst case?



Lower bound

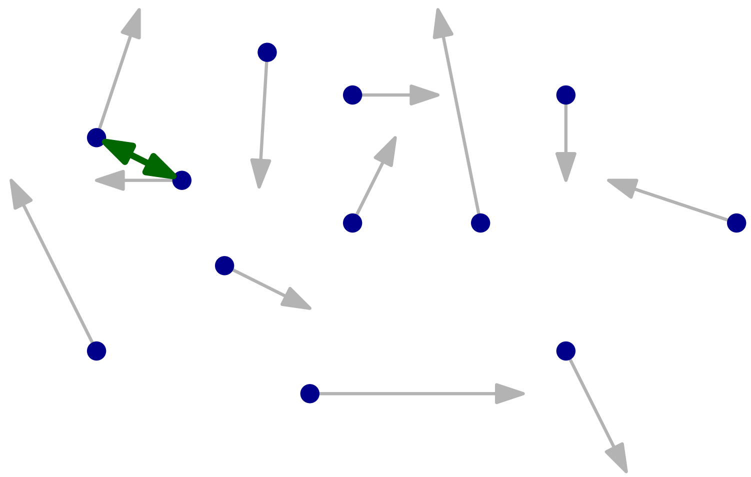
$\Omega(n^2)$ changes

How often can the closest pair change, in the worst case?



Upper bound

How often can the closest pair change, in the worst case?

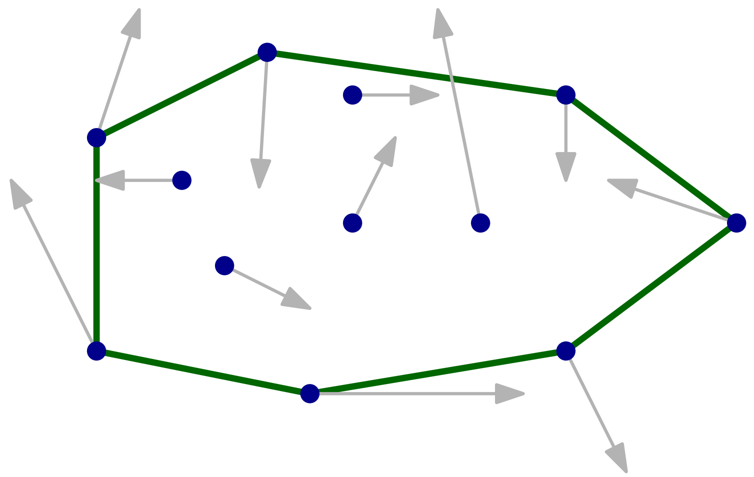


Upper bound

- for each pair p, q define $f_{pq}(t) :=$ distance between p and q at time t
- number of changes = complexity of lower envelope of n^2 functions

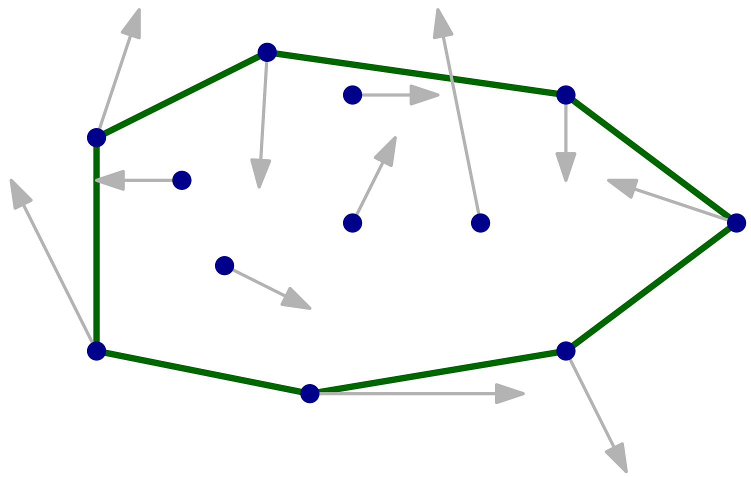
  $\approx O(n^2)$

How often can the convex hull change, in the worst case?

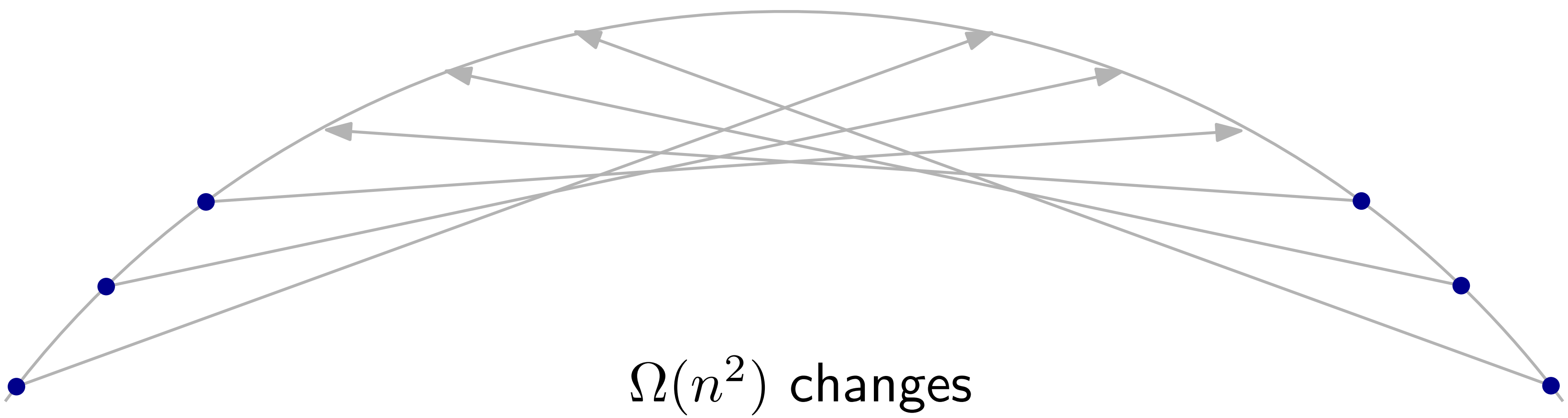


Lower bound

How often can the convex hull change, in the worst case?

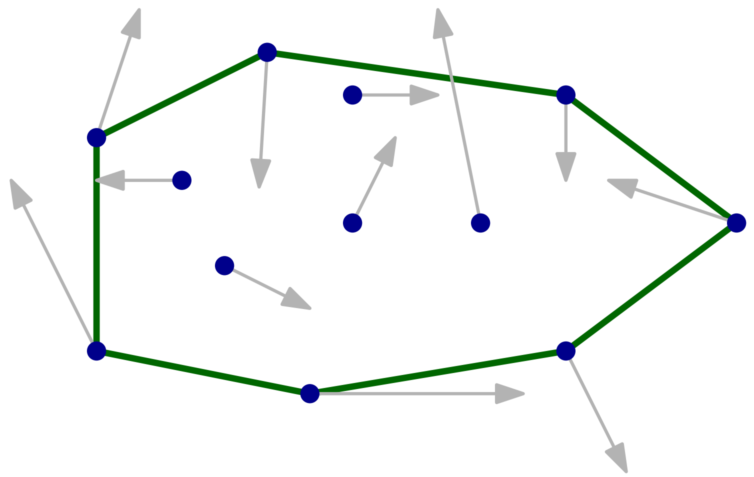


Lower bound



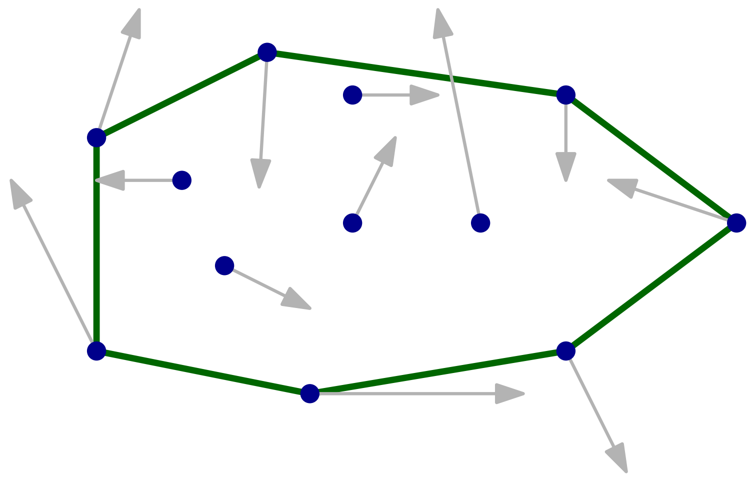
$\Omega(n^2)$ changes

How often can the convex hull change, in the worst case?



Trivial upper bound

How often can the convex hull change, in the worst case?



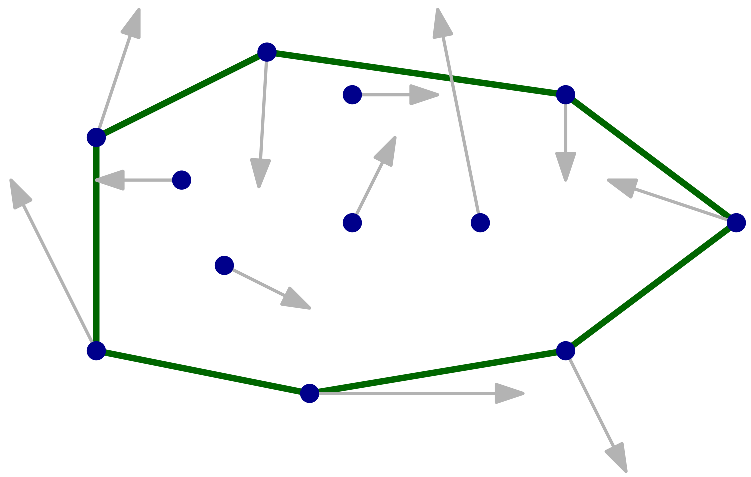
Trivial upper bound

convex hull changes $\Longrightarrow$ three points become collinear

$\Longrightarrow$ happens $O(1)$ times for each triple

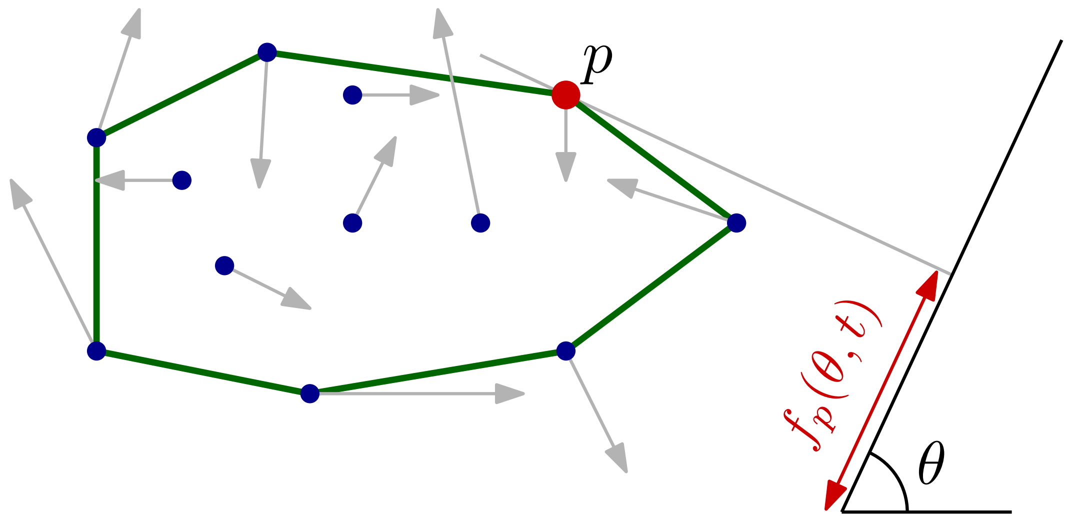
$\Longrightarrow$ $O(n^3)$ changes to convex hull

How often can the convex hull change, in the worst case?



A better bound using upper envelopes

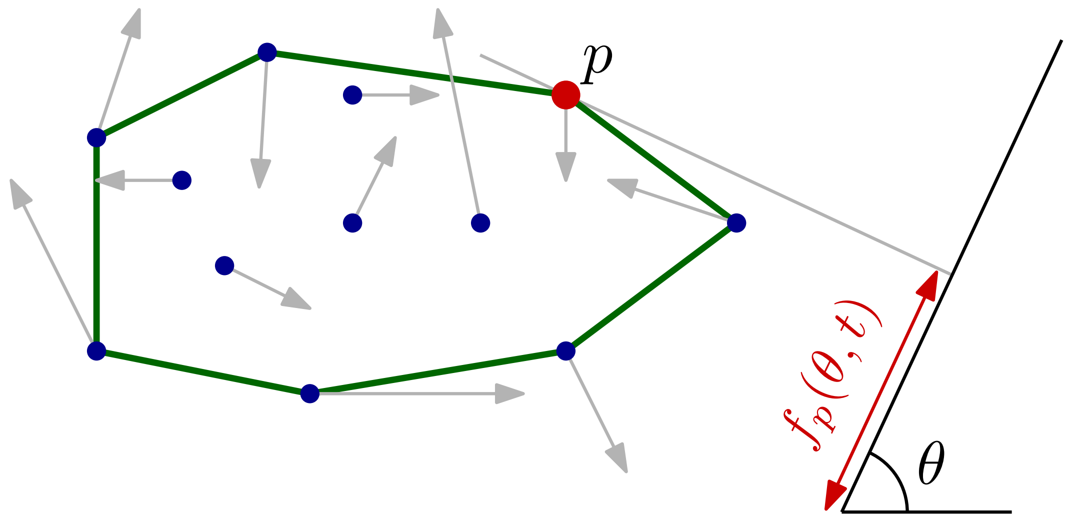
How often can the convex hull change, in the worst case?



A better bound using upper envelopes

- for each point p define function $f_p : [0, 2\pi) \times \mathbb{R}_{\geqslant 0} \to \mathbb{R}$

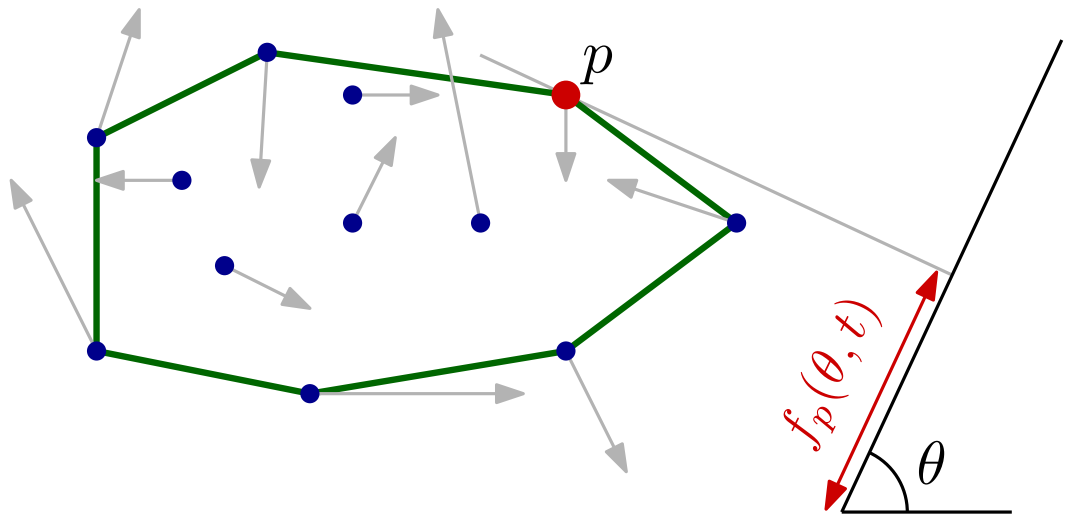
How often can the convex hull change, in the worst case?



A better bound using upper envelopes

- for each point p define function $f_p : [0, 2\pi) \times \mathbb{R}_{\geqslant 0} \to \mathbb{R}$
- p is on convex hull at time t iff $f_p(\theta, t) \geqslant f_q(\theta, t)$ for all q at time t

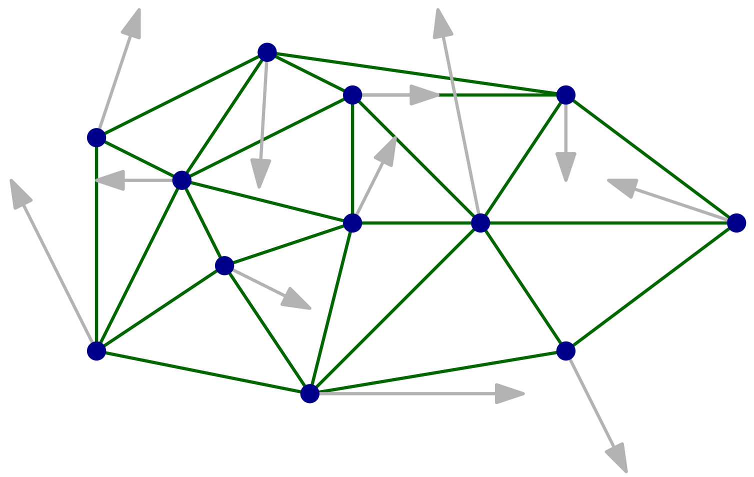
How often can the convex hull change, in the worst case?



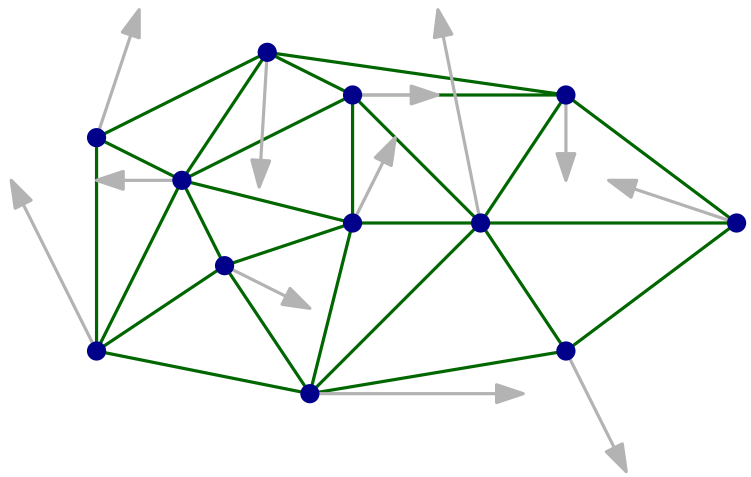
A better bound using upper envelopes

- for each point p define function $f_p : [0, 2\pi) \times \mathbb{R}_{\geqslant 0} \rightarrow \mathbb{R}$
- p is on convex hull at time t iff $f_p(\theta, t) \geqslant f_q(\theta, t)$ for all q at time t
- number of changes
  $= O($complexity of upper envelope of surfaces in $\mathbb{R}^3) = O(n^{2+\varepsilon})$

How often can the Delaunay triangulation change, in the worst case?



How often can the Delaunay triangulation change, in the worst case?

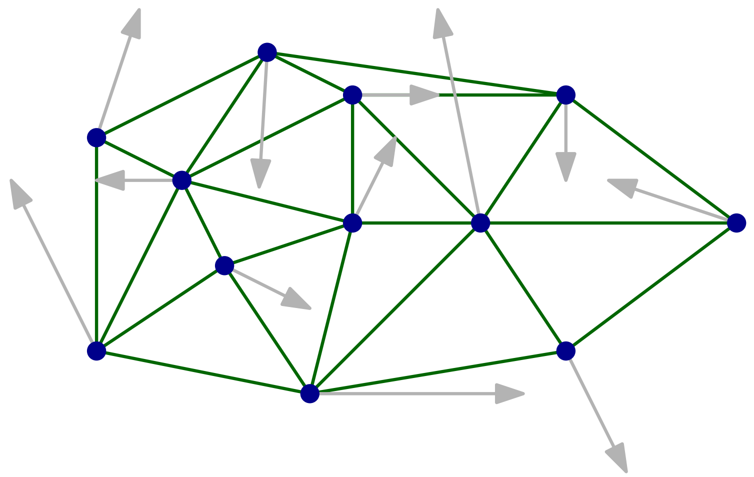


DT changes when convex hull changes $\Longrightarrow \Omega(n^2)$ changes

Exercises

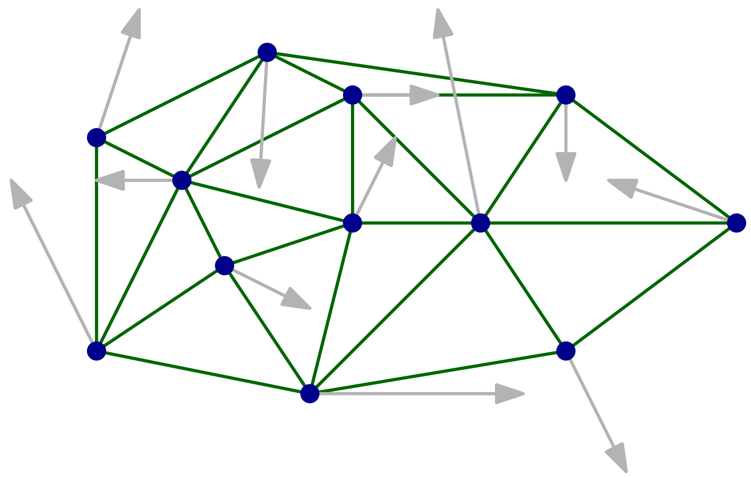
1. Give a trivial upper bound on the number of changes.
2. Give an improved upper bound using upper envelopes.

How often can the Delaunay triangulation change, in the worst case?



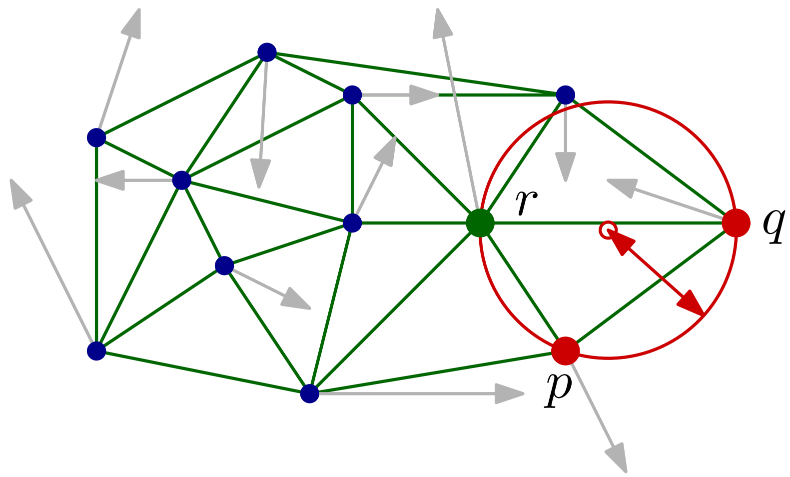
1. When DT changes, four points become co-circular $\implies$ $O(n^4)$ changes

How often can the Delaunay triangulation change, in the worst case?



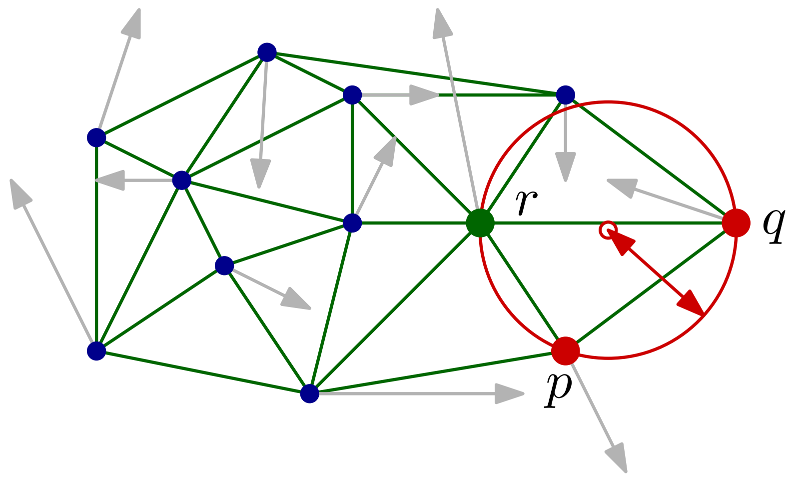
1. When DT changes, four points become co-circular $\implies$ $O(n^4)$ changes
2. When convex hull changes, DT changes $\implies$ $\Omega(n^2)$ changes

How often can the Delaunay triangulation change, in the worst case?



1. When DT changes, four points become co-circular $\implies$ $O(n^4)$ changes
2. for each pair p, q, and each r, define function $f_{pq}^{(r)}(t) : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$

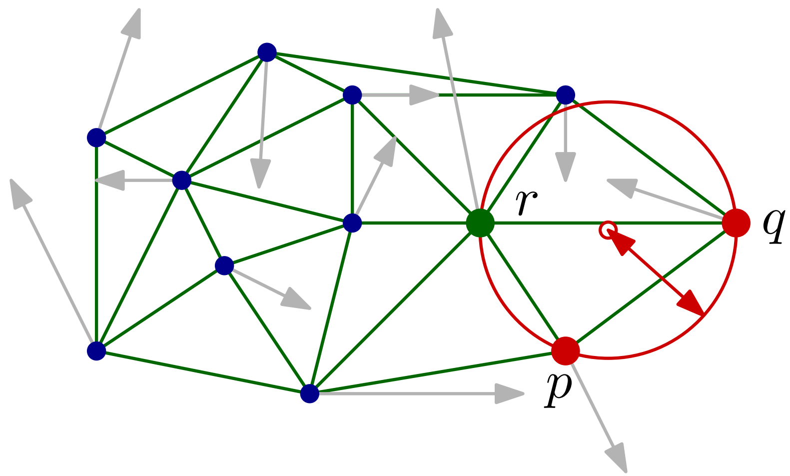
How often can the Delaunay triangulation change, in the worst case?



1. When DT changes, four points become co-circular $\implies$ $O(n^4)$ changes
2. for each pair p, q, and each r, define function $f_{pq}^{(r)}(t) : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$

   p, q, r form triangle in DT: $f_{pq}^{(r)}(t) < f_{pq}^{(r')}(t)$ for all r'

How often can the Delaunay triangulation change, in the worst case?

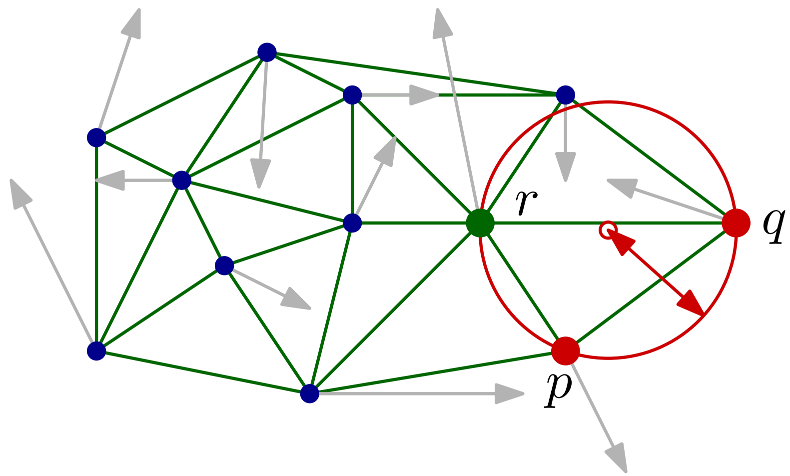


1. When DT changes, four points become co-circular $\Longrightarrow$ $O(n^4)$ changes
2. for each pair p, q, and each r, define function $f_{pq}^{(r)}(t) : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$

   p, q, r form triangle in DT: $f_{pq}^{(r)}(t) < f_{pq}^{(r')}(t)$ for all r'

number of changes $= n^2 \times$ complexity of lower envelope in $\mathbb{R}^2 \approx O(n^3)$

How often can the Delaunay triangulation change, in the worst case?



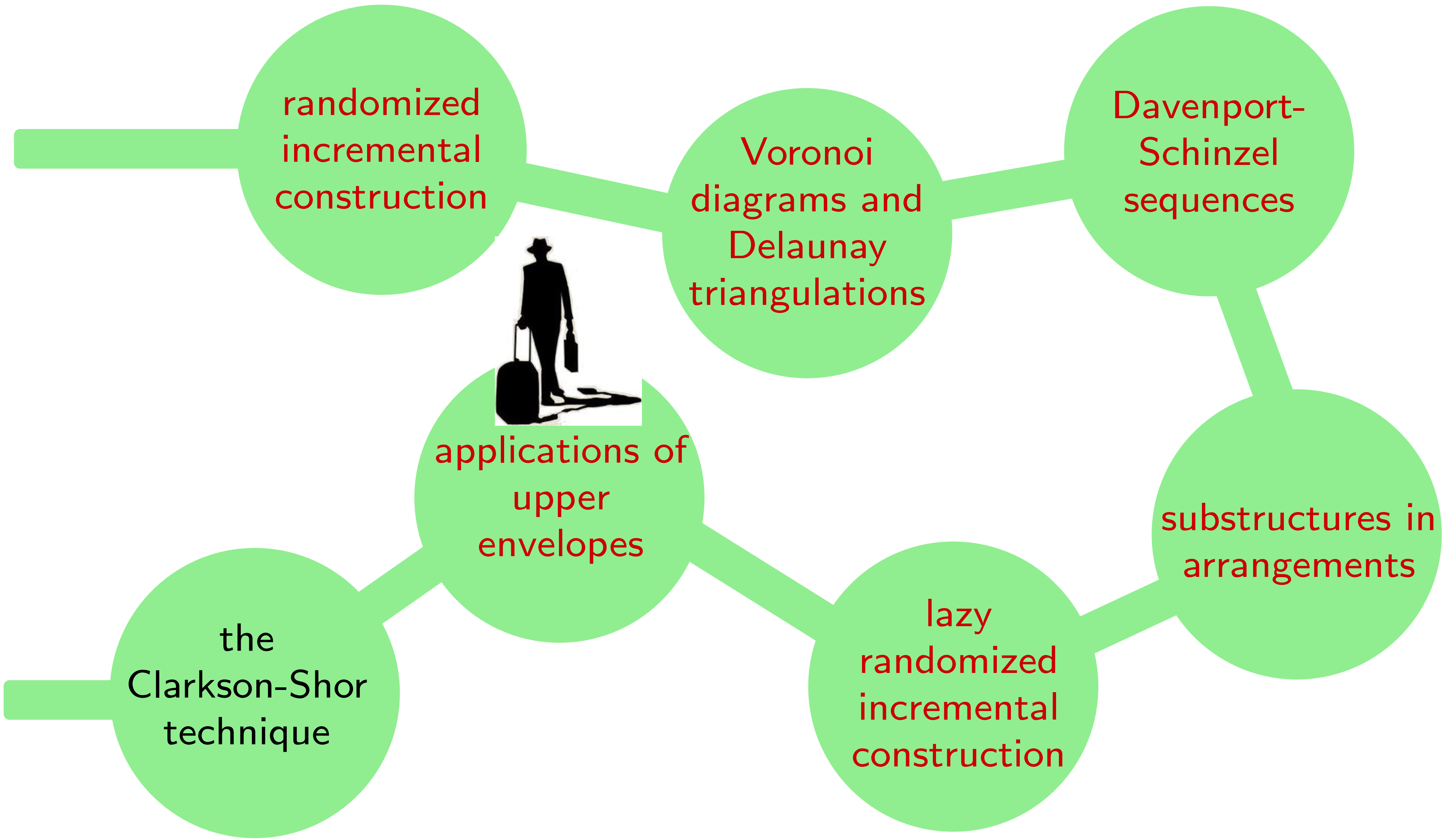
[Rubin '15; 85 pages]
for linear motions the DT changes $O(n^{2+\varepsilon})$ times

1. When DT changes, four points become co-circular $\implies$ $O(n^4)$ changes
2. for each pair p, q, and each r, define function $f_{pq}^{(r)}(t) : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$

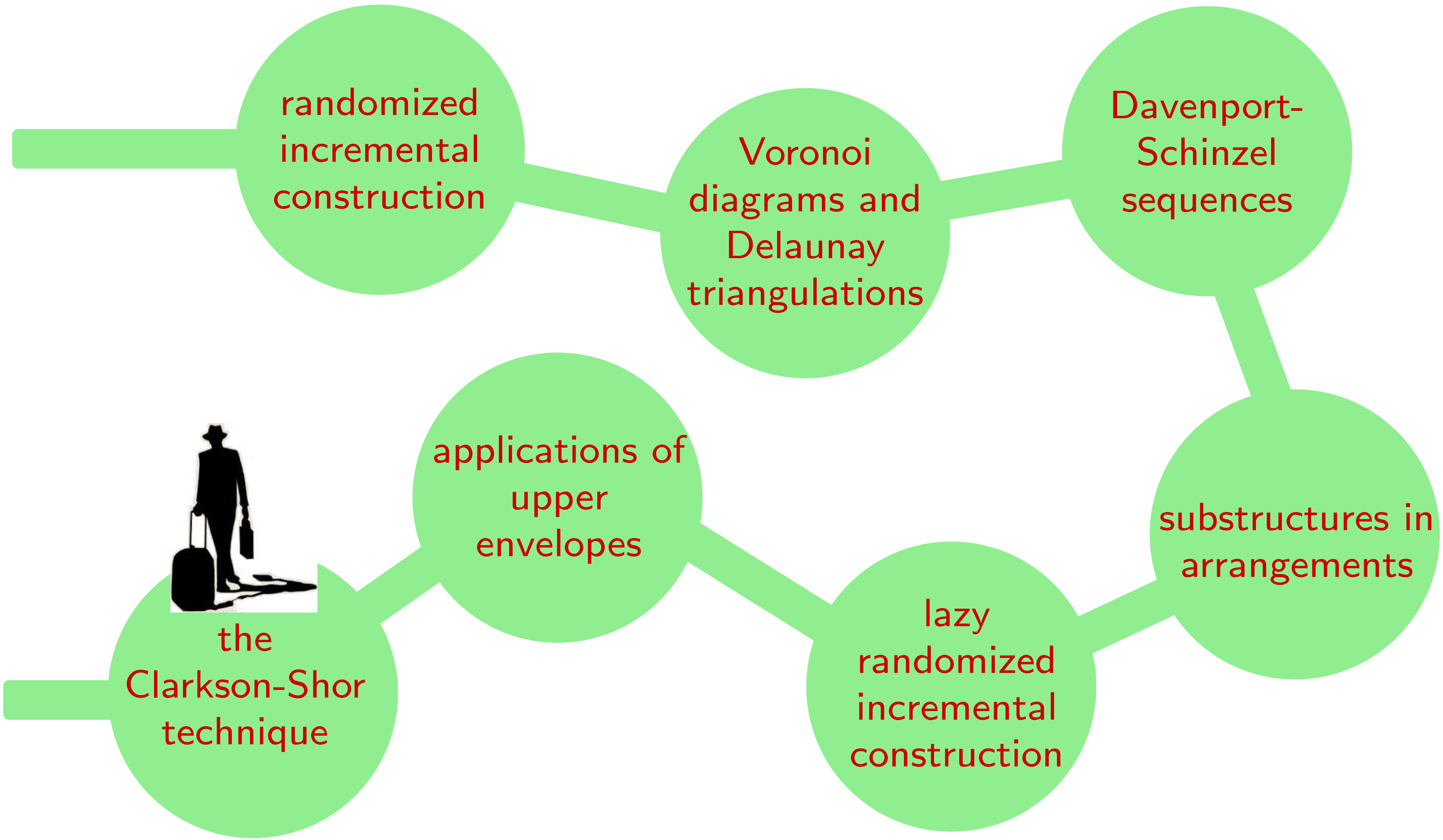
   p, q, r form triangle in DT: $f_{pq}^{(r)}(t) < f_{pq}^{(r')}(t)$ for all r'

number of changes $= n^2 \times$ complexity of lower envelope in $\mathbb{R}^2 \approx O(n^3)$

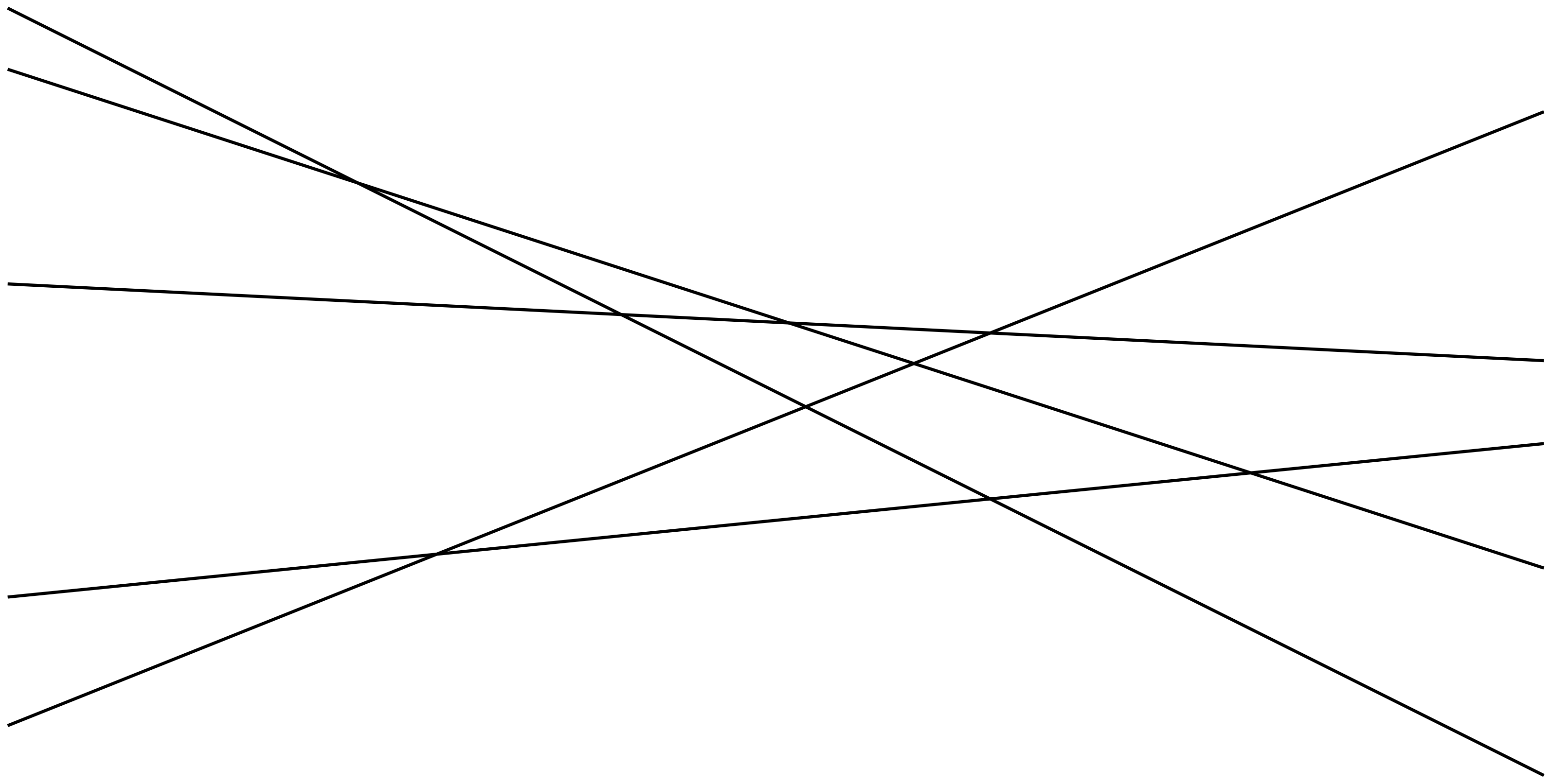
Course Overview

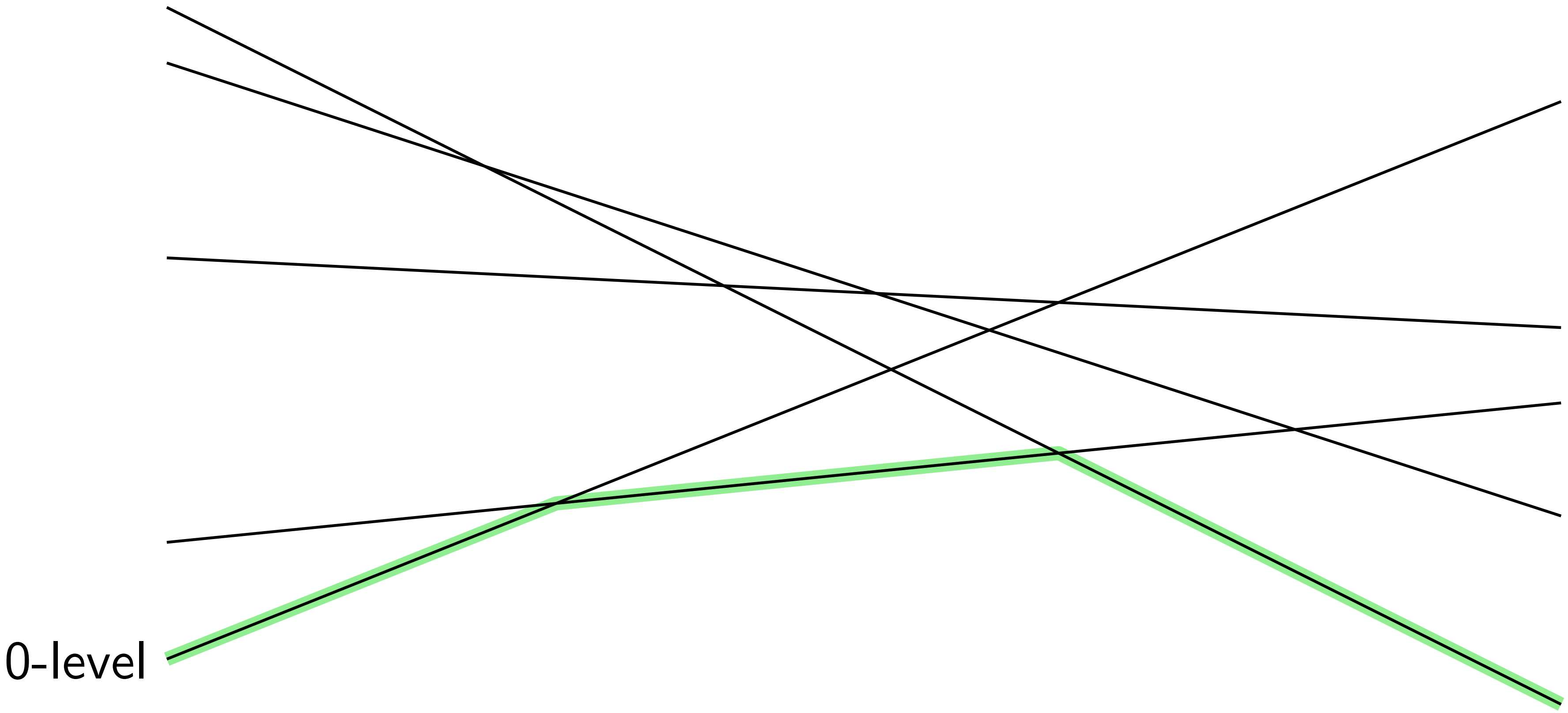


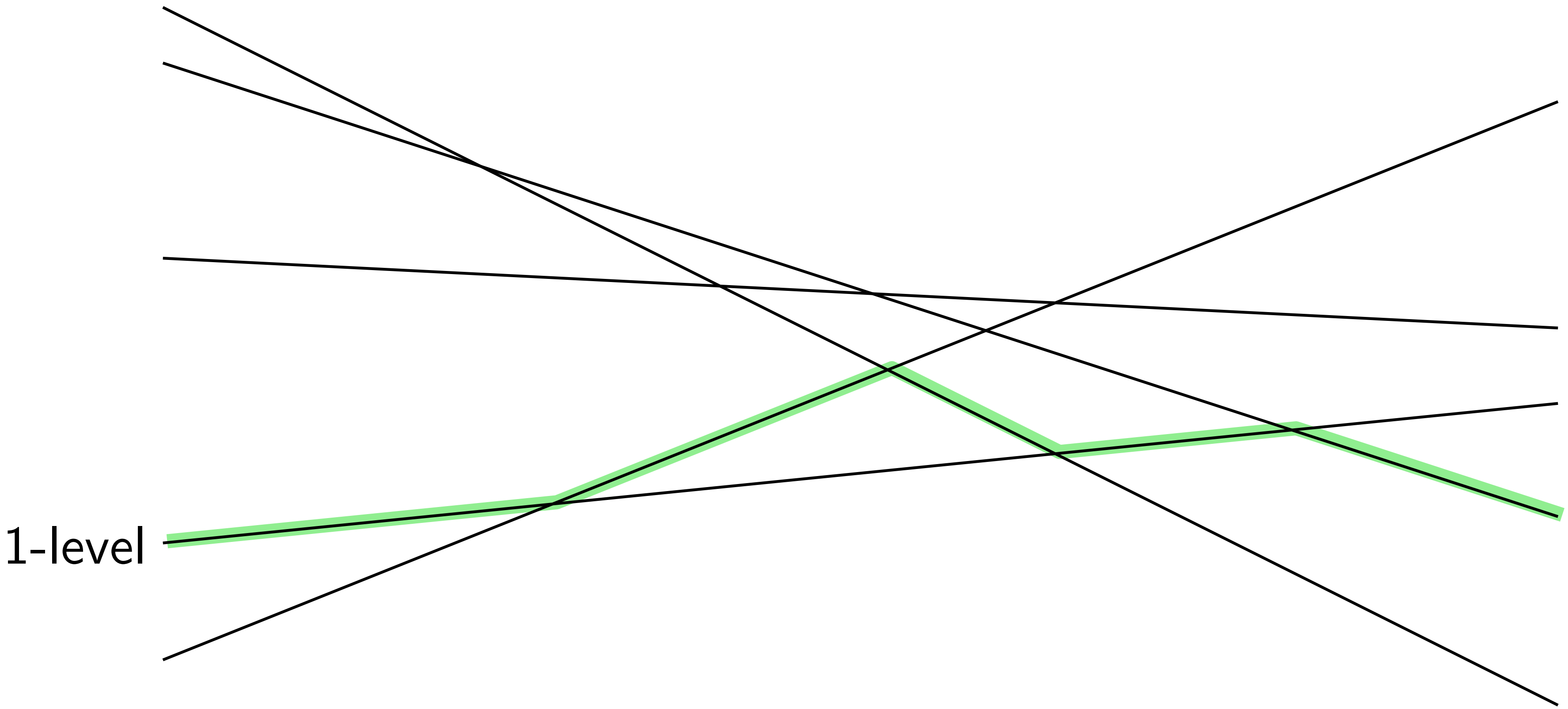
Course Overview

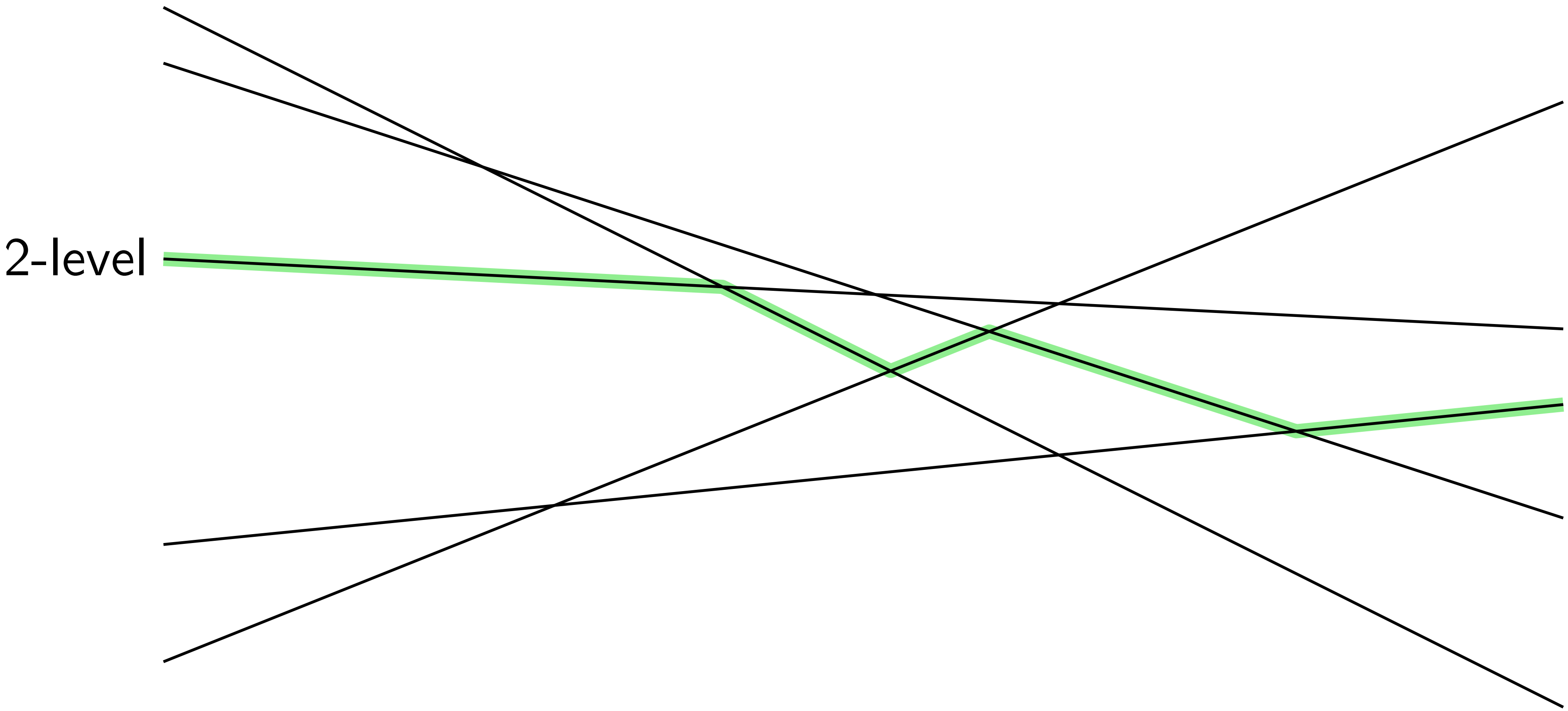


Levels in arrangements

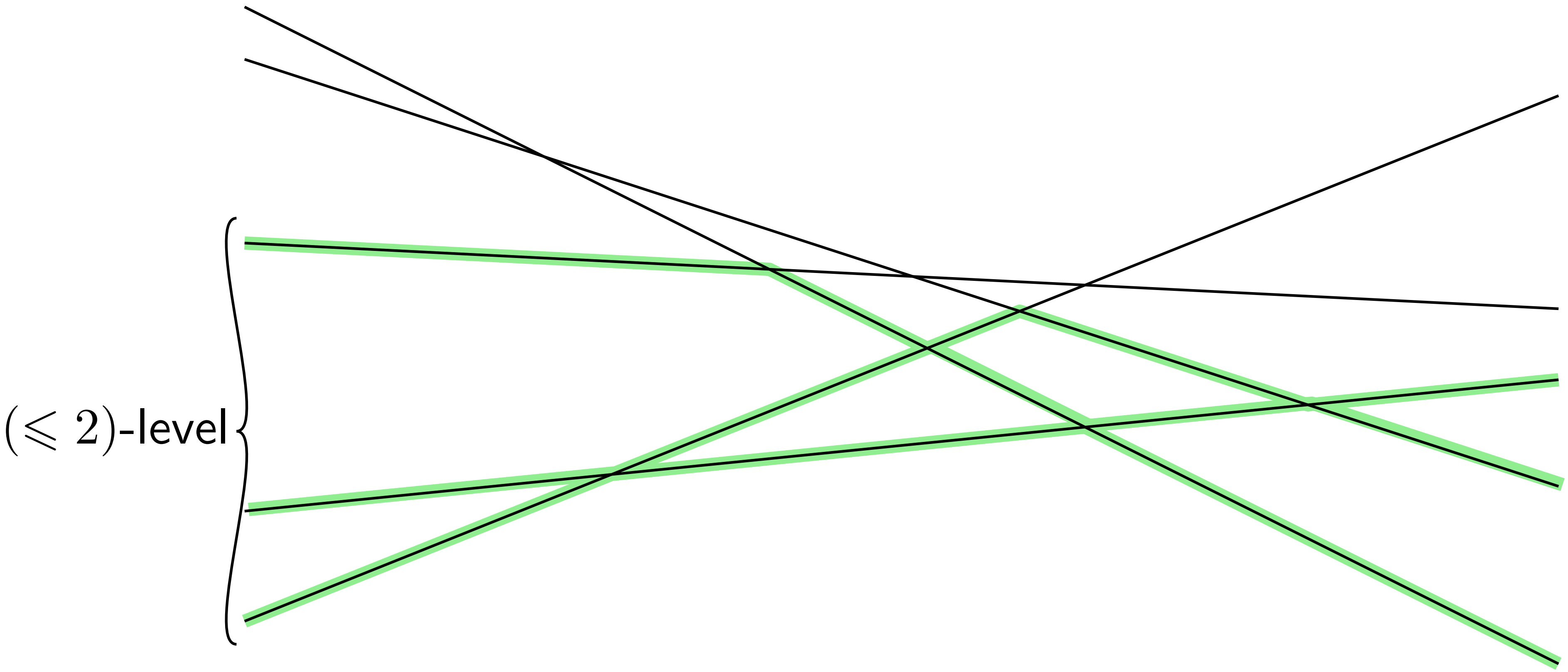



0-level

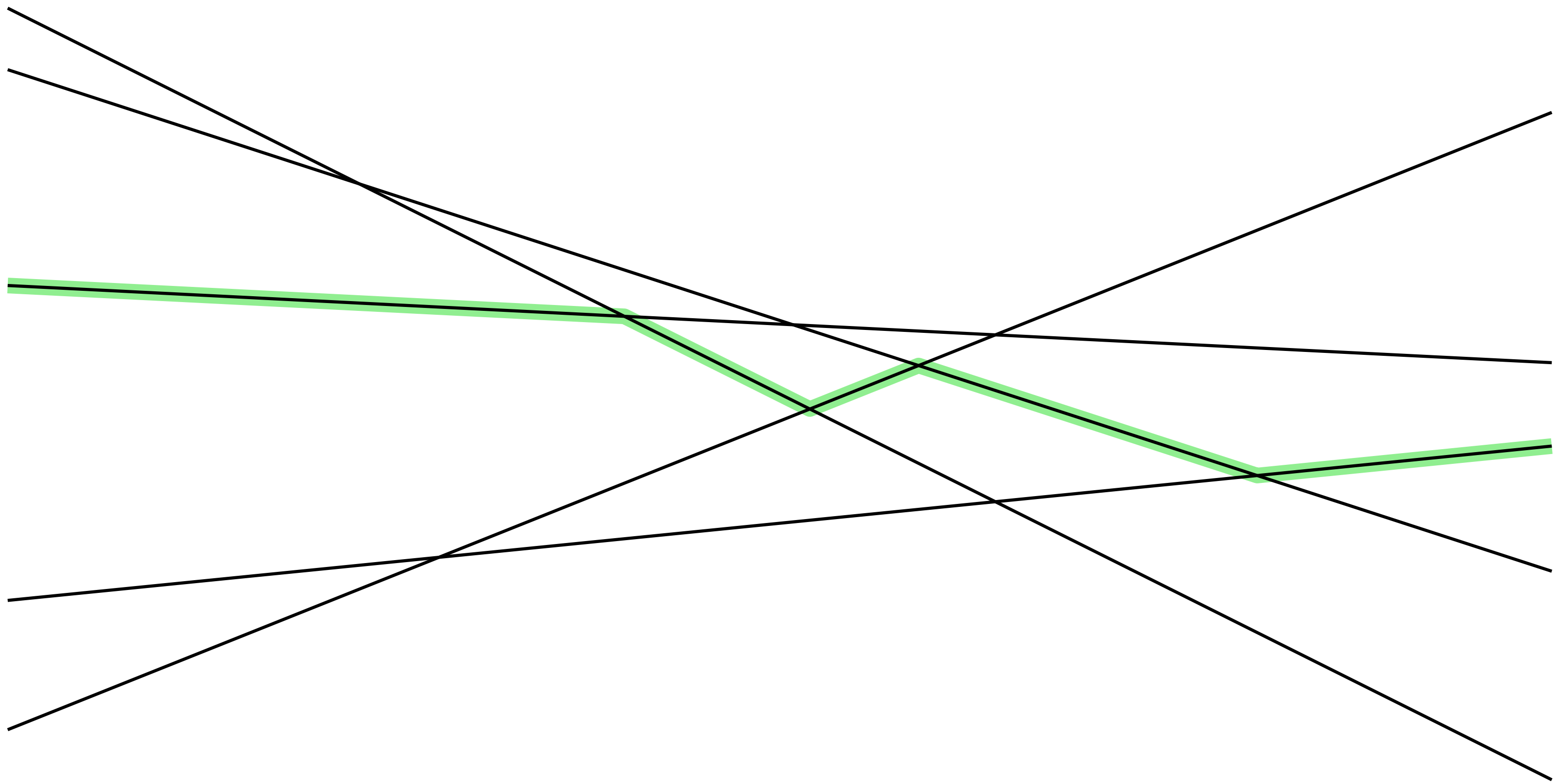

1-level


2-level

Levels in arrangements



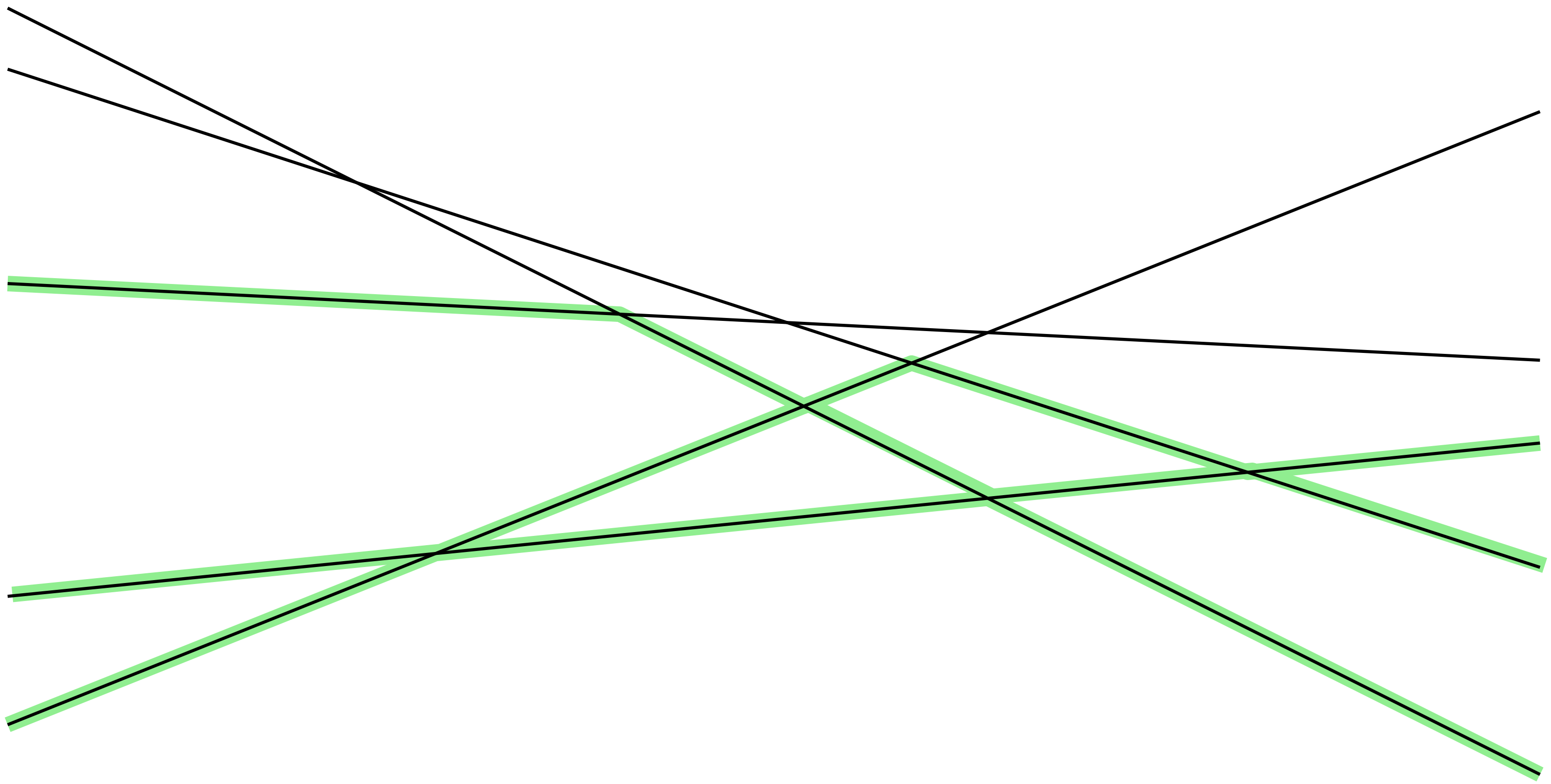
Levels in arrangements



What is the max complexity of the k-level in an arrangement of n lines?

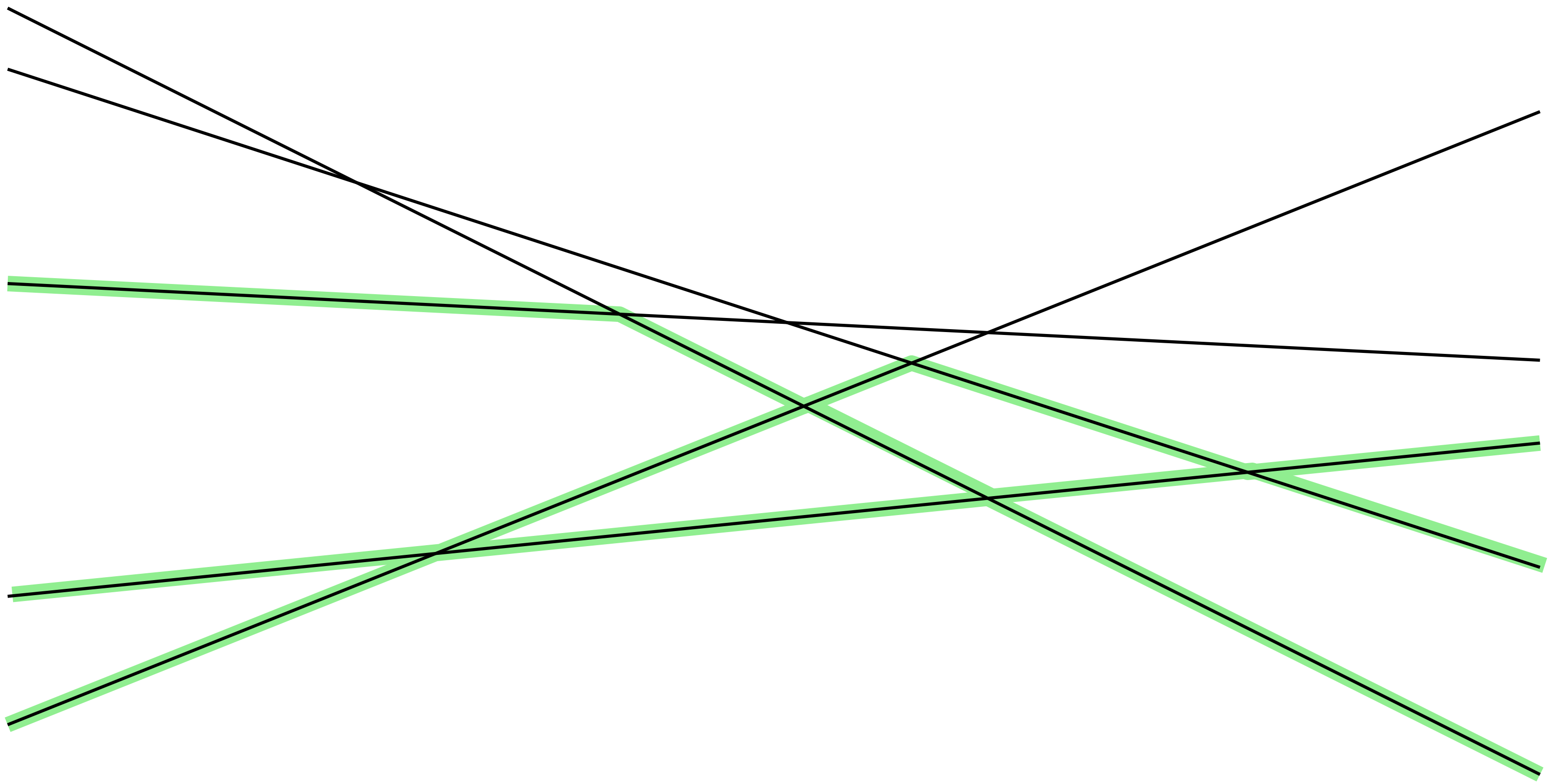
- 0-level $=$ lower envelope $\implies$ complexity $\leqslant n$
- $k \geqslant 1$: complexity is $n2^{\Omega(\sqrt{\log k})}$ and $O(nk^{1/3})$

The Clarkson-Shor Technique: Application to $(\leqslant k)$-levels

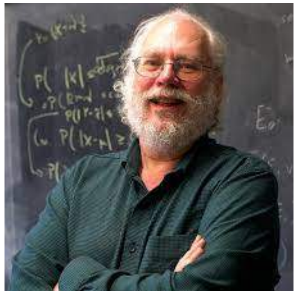


What is the max complexity of the $(\leqslant k)$-level in an arrangement of n lines?

The Clarkson-Shor Technique: Application to $(\leqslant k)$-levels



What is the max complexity of the $(\leqslant k)$-level in an arrangement of n lines?



Clarkson-Shor '89: $\Theta(nk)$

The Clarkson-Shor Technique: Application to $(\leqslant k)$-levels

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Take sample $R \subset L$ by picking each line $\ell \in L$ with probability $1/k$.

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Take sample $R \subset L$ by picking each line $\ell \in L$ with probability $1/k$.

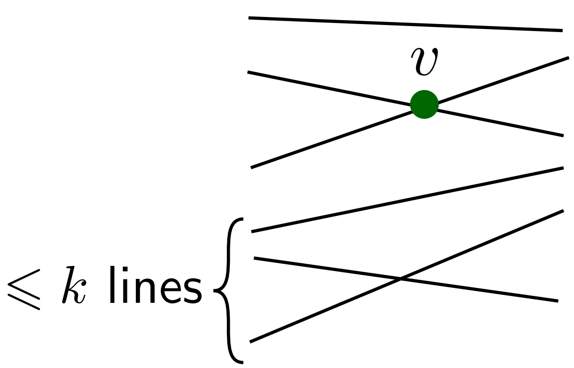
$$\mathbb{E}\big[\text{complexity of 0-level of } R\big] \;\leqslant\; \mathbb{E}\big[|R|\big] \;=\; n/k$$

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Take sample $R \subset L$ by picking each line $\ell \in L$ with probability $1/k$.

$$\mathbb{E}\big[\text{complexity of 0-level of } R\big] \leqslant \mathbb{E}\big[|R|\big] = n/k$$



vertex of k-level of L shows up on 0-level of R iff

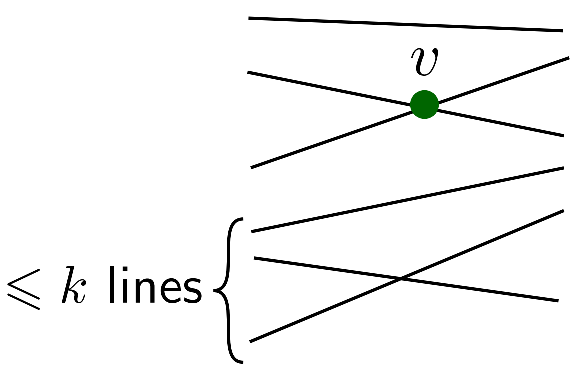
- both lines defining v are in R
- none of the at most k lines below v are in R

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Take sample $R \subset L$ by picking each line $\ell \in L$ with probability $1/k$.

$$\mathbb{E}\big[\text{complexity of 0-level of } R\big] \leqslant \mathbb{E}\big[|R|\big] = n/k$$



vertex of k-level of L shows up on 0-level of R iff

- both lines defining v are in R
- none of the at most k lines below v are in R

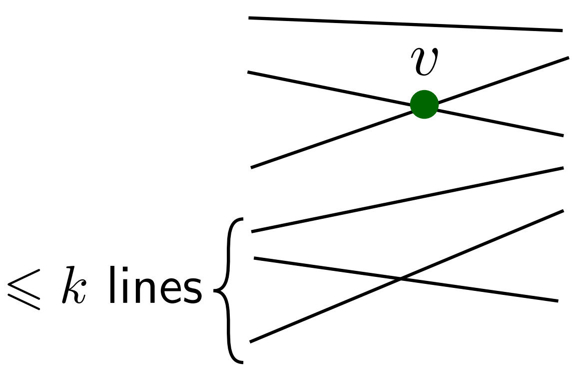
$$\text{prob} \geqslant \left(\tfrac{1}{k}\right)^2 \cdot \left(1 - \tfrac{1}{k}\right)^k \geqslant \left(\tfrac{1}{k}\right)^2 \cdot \tfrac{1}{e}$$

Theorem. The max complexity of the $(\leqslant k)$-level in an arrangement induced by a set L of n lines in the plane is $O(nk)$.

Proof.

Take sample $R \subset L$ by picking each line $\ell \in L$ with probability $1/k$.

$$\mathbb{E}\big[\text{complexity of 0-level of } R\big] \;\leqslant\; \mathbb{E}\big[|R|\big] \;=\; n/k$$



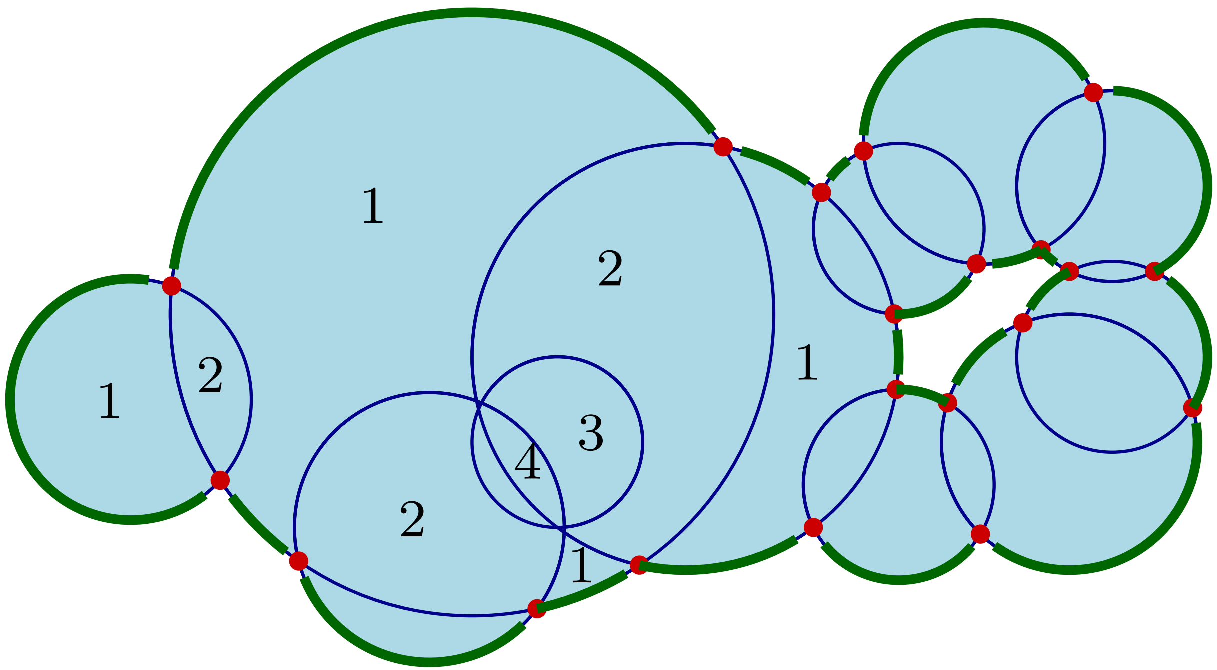
vertex of k-level of L shows up on 0-level of R iff

- both lines defining v are in R
- none of the at most k lines below v are in R

$$\text{prob} \geqslant \left(\tfrac{1}{k}\right)^2 \cdot \left(1 - \tfrac{1}{k}\right)^k \geqslant \left(\tfrac{1}{k}\right)^2 \cdot \tfrac{1}{e}$$

$$\mathbb{E}\big[\text{complexity of 0-level of } R\big] \;\geqslant\; (\text{complexity of } k\text{-level in } L) \cdot \left(\tfrac{1}{k}\right)^2 \cdot \tfrac{1}{e}$$

Another application: Depth in Disk Arrangements

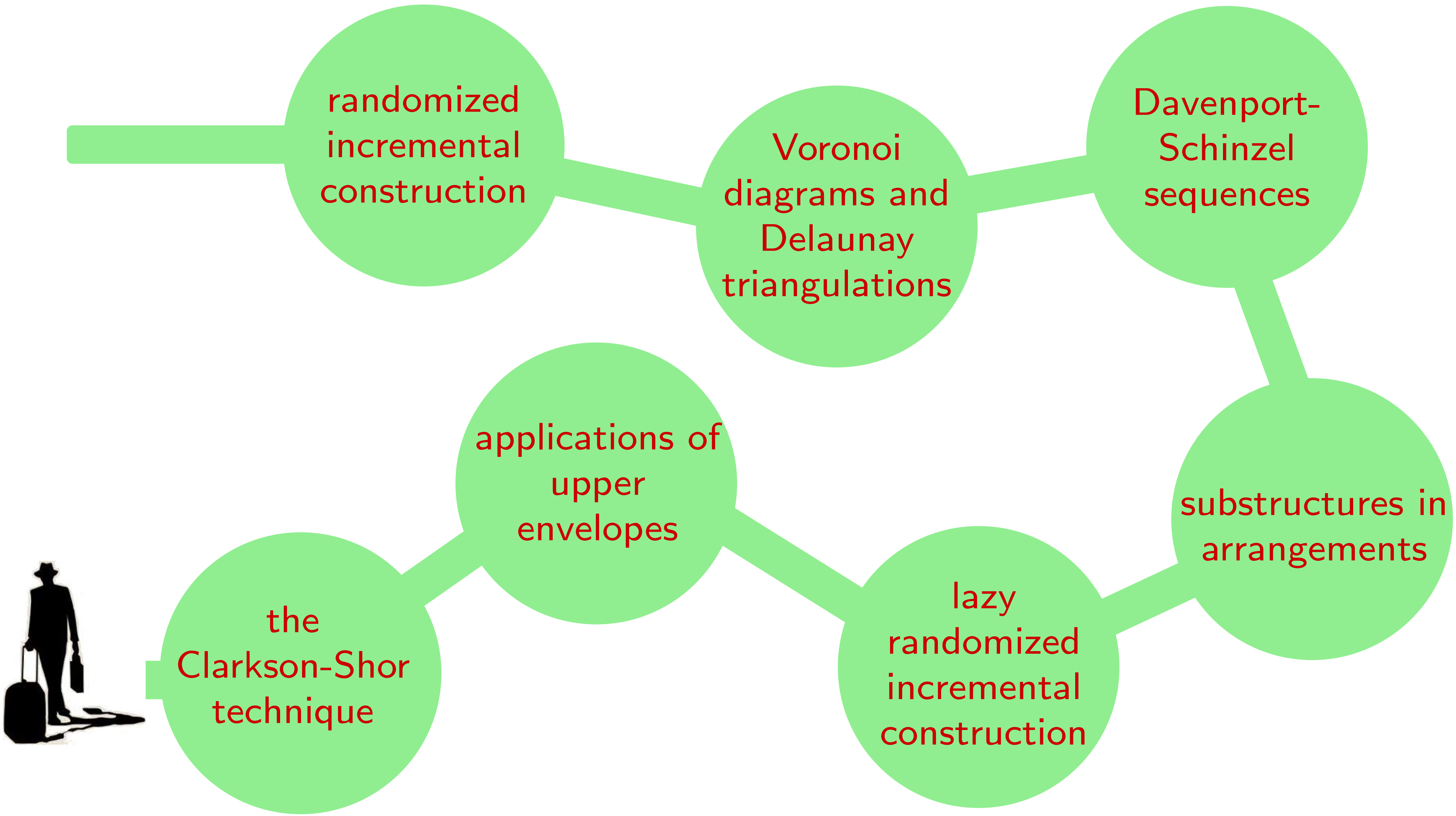


Exercises

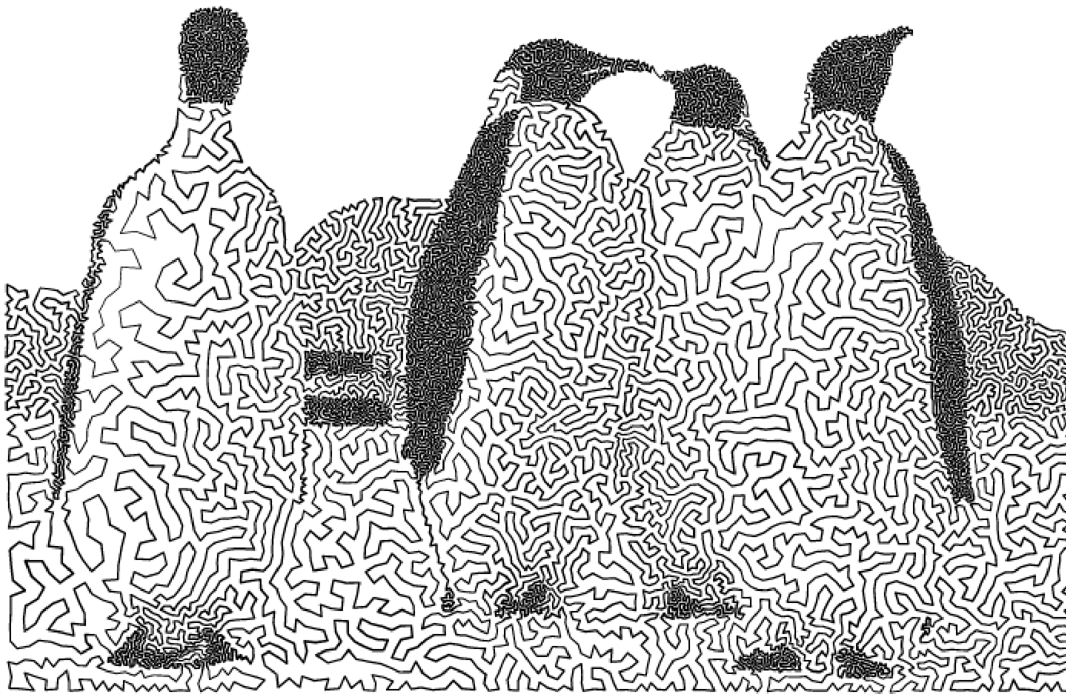
1. Prove that the total number of vertices on the union boundary is $O(n)$. Hint: Define a suitable planar graph whose nodes are disk centers.

2. Prove that the total number of regions of depth at most k is $O(nk)$.

Course Overview



Thanks for your attention!



TSP Art by Carig Kaplan and Robert Bosch