Introduction to computational geometry

Convex Hull

Yann GAVET, Johan DEBAYLE

Ecole Nationale Supérieure des Mines de Saint-Etienne
Laboratoire Georges Friedel, UMR CNRS 5307

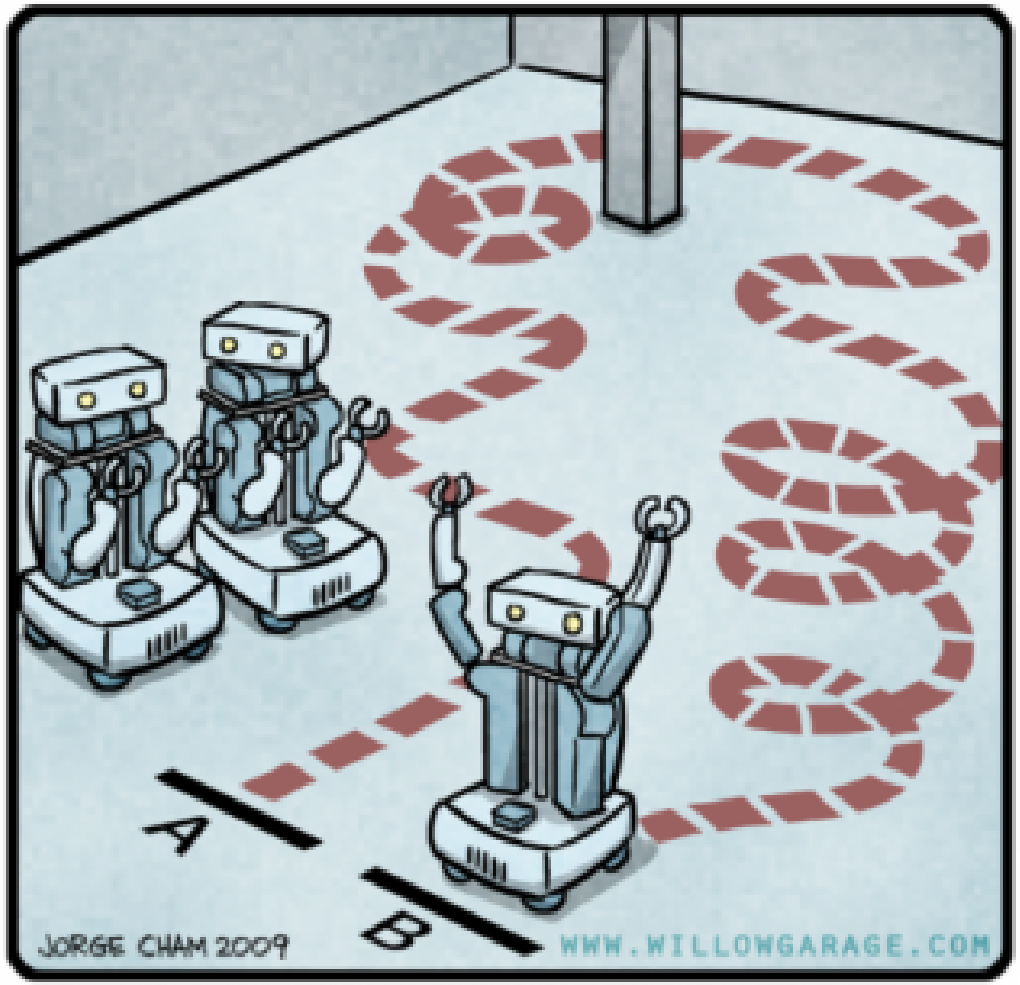
Plan

Section 1

Introduction

Use cases

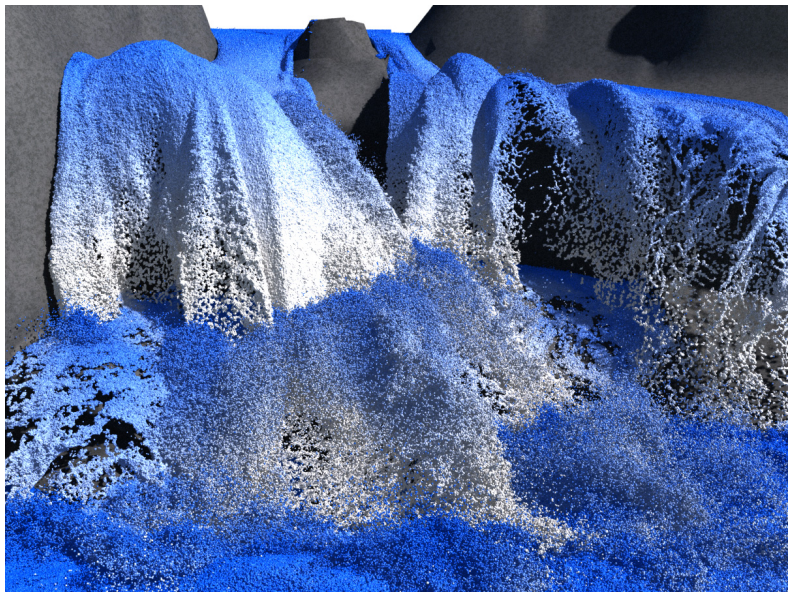
R.O.B.O.T. Comics

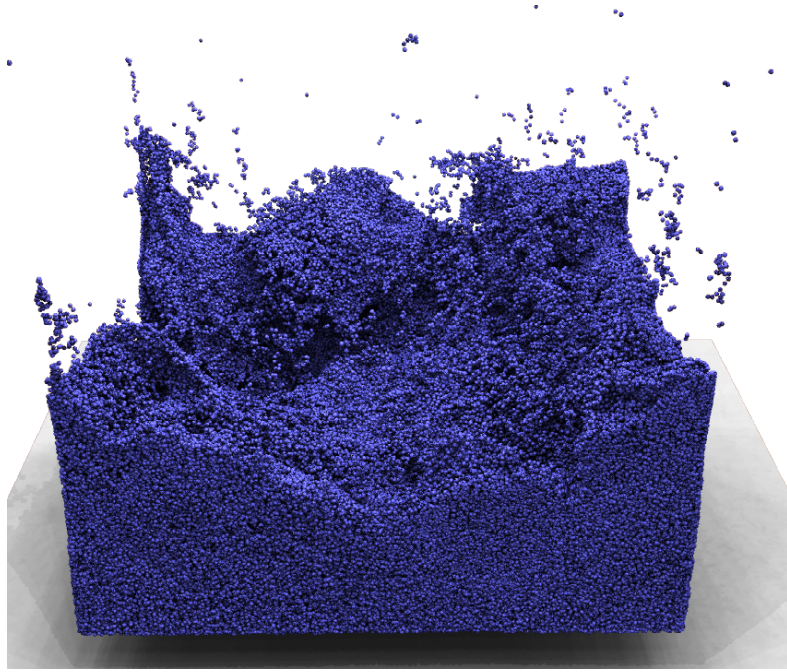


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Computational geometry

- Deals with geometry problems and the corresponding algorithms.
- Complexity is a key consideration: huge number of points/data.
- Discrete data are considered:
  - point sets
  - finite (but sometimes huge) number of data





In 1 sentence

Computational geometry is a way of ordering the data in its space.

- The key algorithm is the sorting of numbers.
- Best complexity in $O(n \cdot \log(n))$

Numerical problems

Floating numbers have limited precision

How to precisely test

- Equality?
- Alignments?

Example: sum of many terms

Classical summation

```
S=0.0;
for (j=1;j<=N;j++)
    S=S+X[j];
return S;
```

Kahan summation

```
S = 0.0; C = 0.0;
for(j=1; j<=N; j++) {
    y = X[j] - C;
    t = S + y;
    C = (t - S) - y;
    S = t;
return S;
```

Pre-requisites: asymptotic notation

Definitions

- $n \in \mathbb{Z}$
- f, g are functions defined on $\mathbb{Z}$
- $C \in \mathbb{R}$
- $n_0 \in \mathbb{Z}$

Big O

$$g(n) \in O(f(n)) \Leftrightarrow \exists C > 0 \mid g(n) \leq C \cdot f(n), \forall n \geq n_0$$

Pre-requisites: asymptotic notation

Definitions

- $n \in \mathbb{Z}$
- f, g are functions defined on $\mathbb{Z}$
- $C \in \mathbb{R}$
- $n_0 \in \mathbb{Z}$

Big Ω

$$g(n) \in \Omega(f(n)) \Leftrightarrow \exists C > 0 \mid g(n) \geq C \cdot f(n), \forall n \geq n_0$$

Pre-requisites: asymptotic notation

Definitions

- $n \in \mathbb{Z}$
- f, g are functions defined on $\mathbb{Z}$
- $C \in \mathbb{R}$
- $n_0 \in \mathbb{Z}$

theta

$$g(n) \in \theta(f(n)) \Leftrightarrow \exists C_1, C_2 > 0 \mid C_1 \cdot f(n) \leq g(n) \leq C_2 \cdot f(n), \forall n \geq n_0$$

Pre-requisites: asymptotic notation

Definitions

- $n \in \mathbb{Z}$
- f, g are functions defined on $\mathbb{Z}$
- $C \in \mathbb{R}$
- $n_0 \in \mathbb{Z}$

Little o

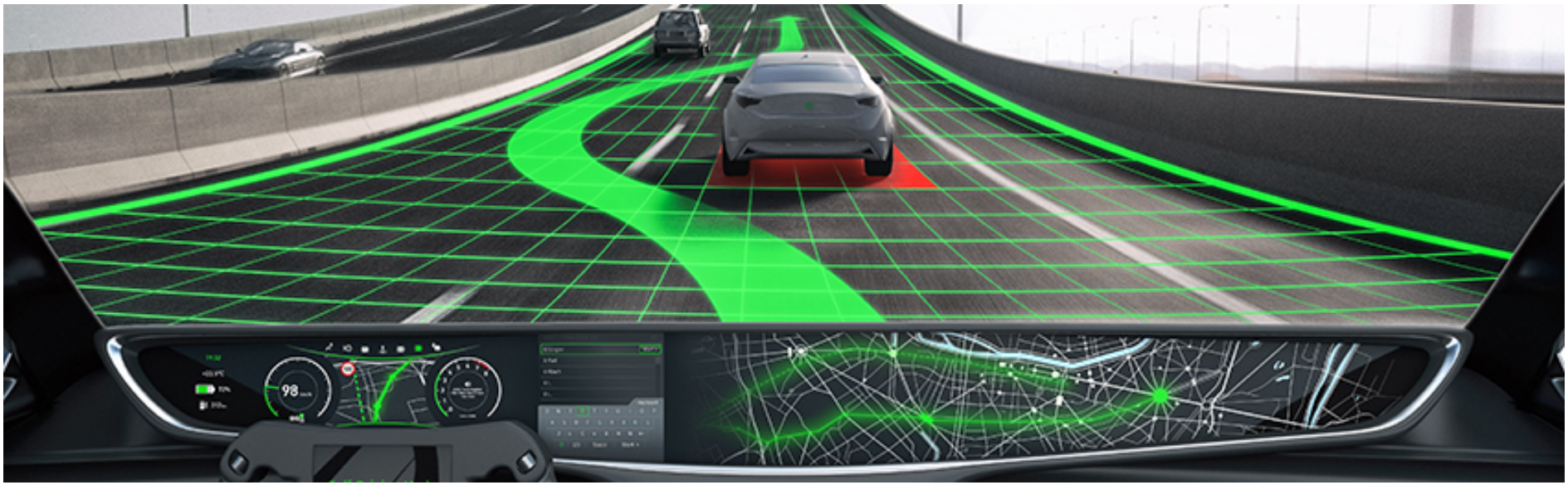
$$g(n) \in o(f(n)) \Leftrightarrow \forall C > 0 \mid g(n) \leq C \cdot f(n), \forall n \geq n_0$$

Plan of the lecture

- Introduction and convex hull
- Voronoi diagrams and Delaunay triangulation
- Nearest neighbor search
- Alpha shapes

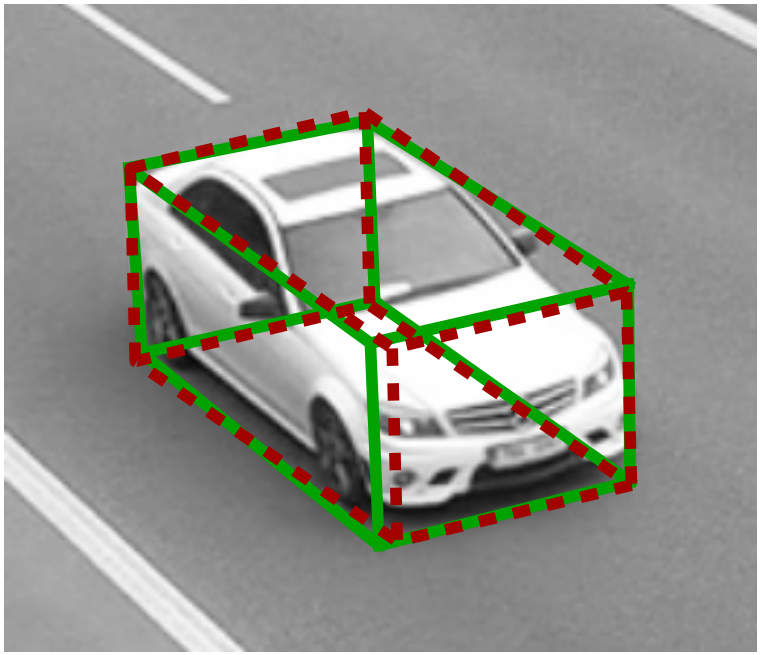
Applications

- Collision avoidance, path planning,



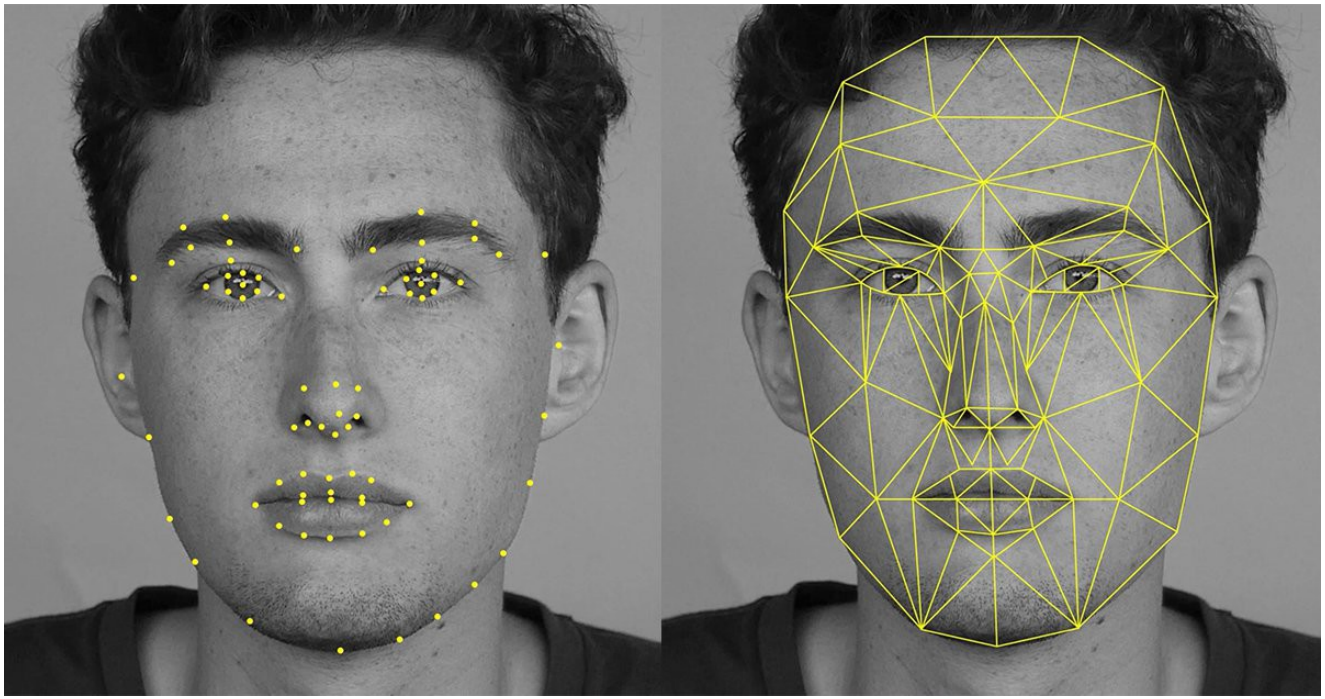
Applications

- Collision avoidance, path planning,
- Enclosing box,



Applications

- Collision avoidance, path planning,
- Enclosing box,
- Shape analysis, classifications



Section 2

Introduction to convex Hull

Notion of convexity

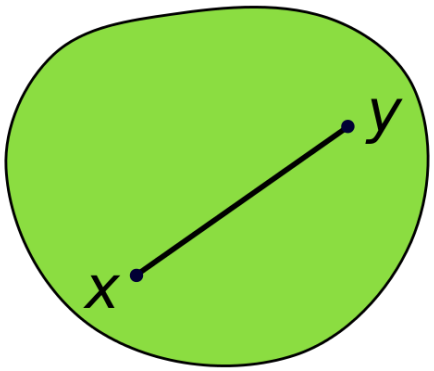
Definition

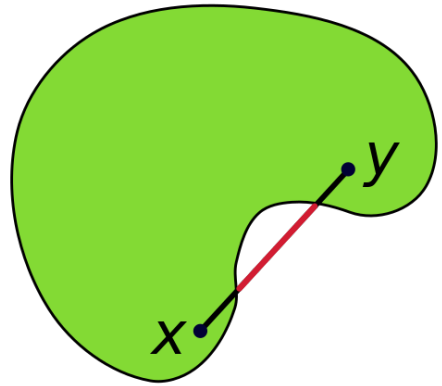
Let S be a set. S is convex if the segment joining every pair of points $(x, y) \in S$ is also in S.

Generalisation: convex combination

$$c = \sum \lambda_i p_i, p_i \in S, \lambda_i \geq 0, \sum \lambda_i = 1$$

More generally, S is convex if every convex combination $c \in S$.

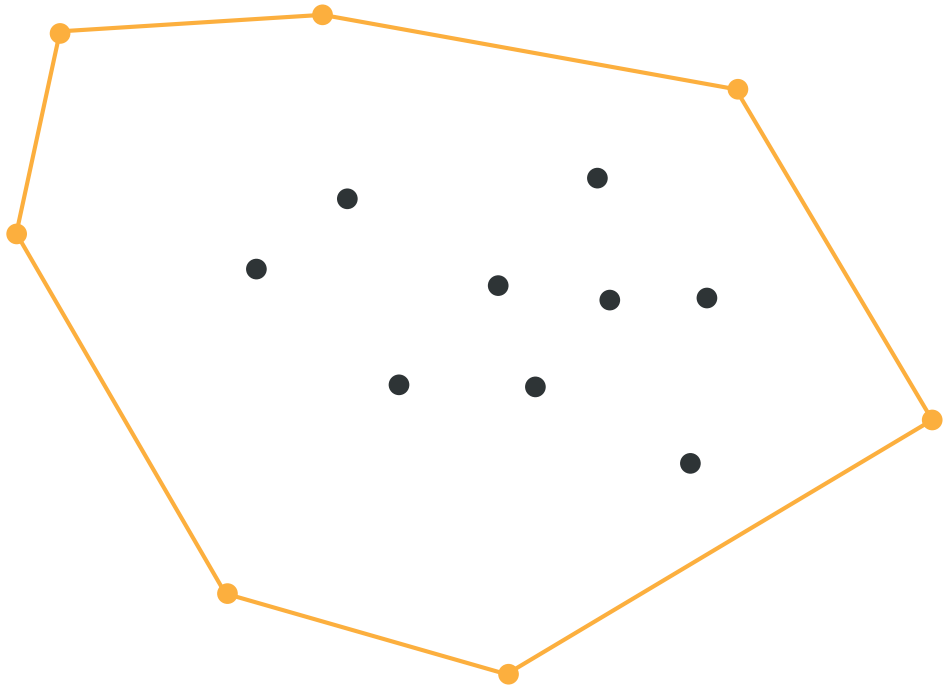




Convex hull

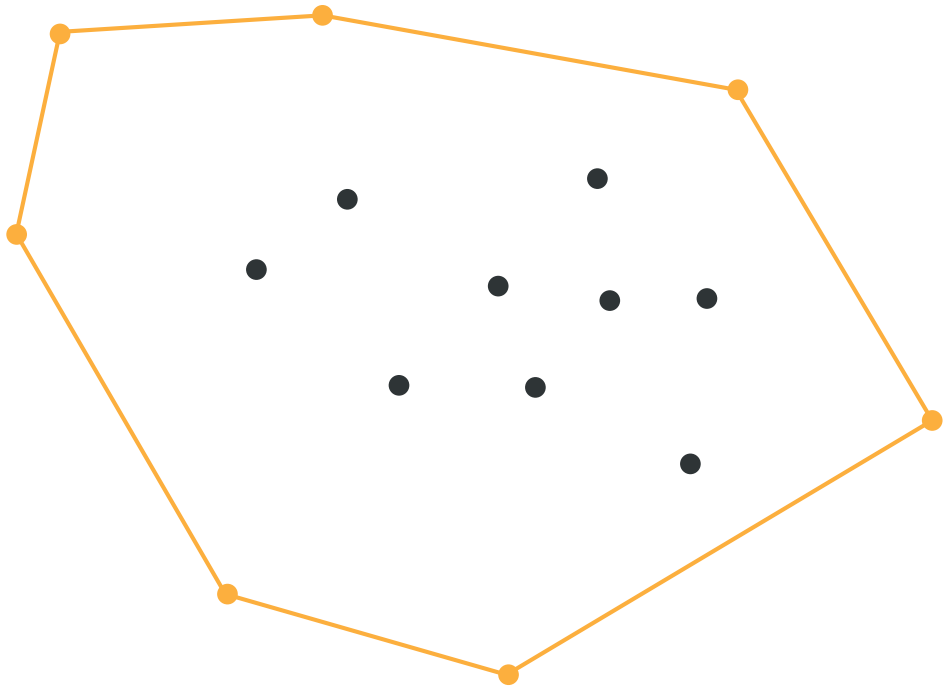
The convex hull, denoted $conv(S)$, is the intersection of all convex regions that contain S.

The convex hull of S is the set of all convex combinations of S.



Polygon

- For a finite point set S, $conv(S)$ is a polygon.
- $conv(S) \subset S$

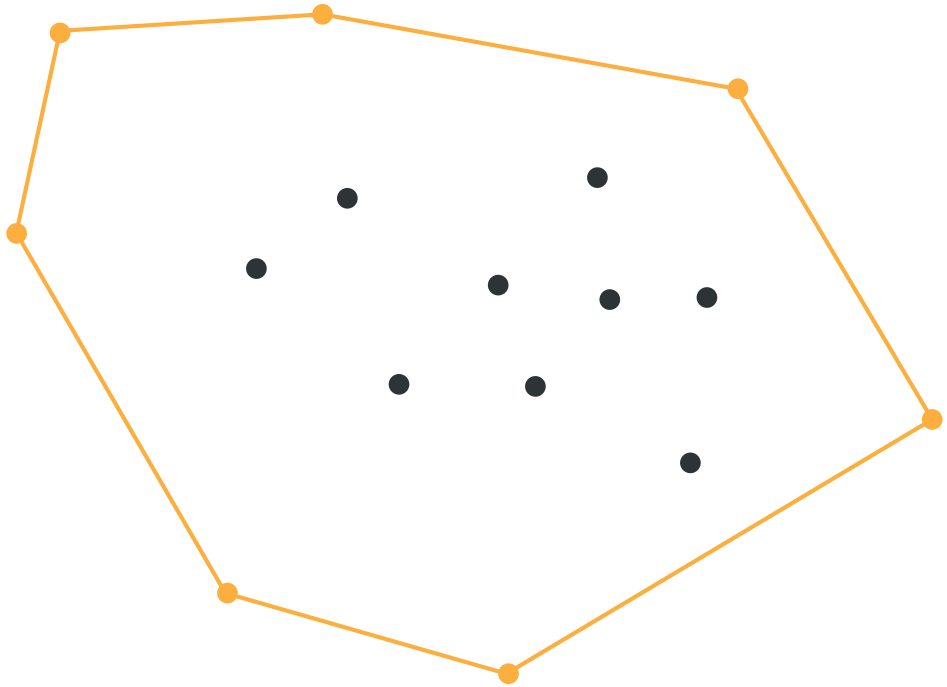


Section 3

Algorithms

How to compute the convex hull?

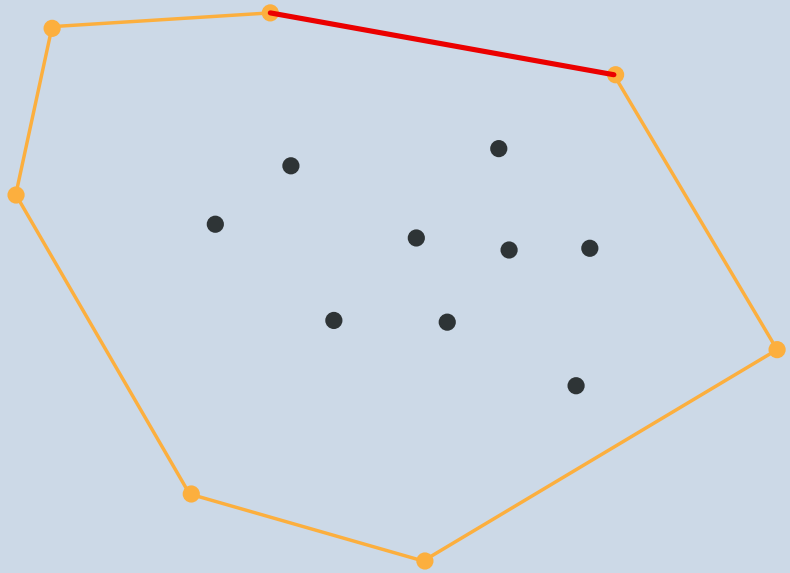
- Input: set of points
- Output: ordered set of points describing $conv(S)$



Naive algorithm

- Definition is useless
- Interesting property: $conv(S) \subset S$

- Take one edge of the polygon
- Every point of S is on the same side

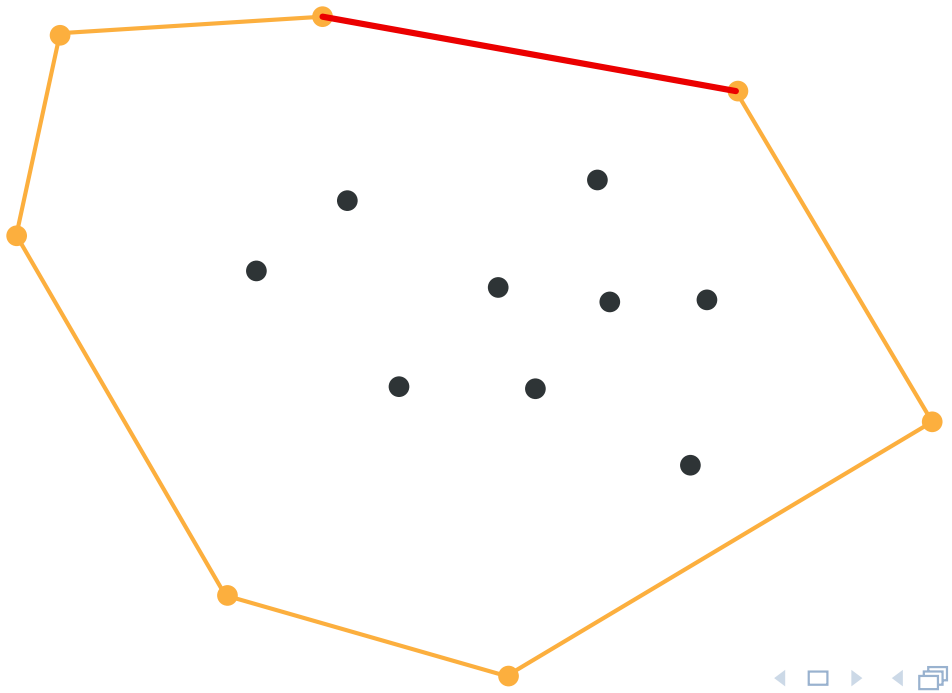


Naive algorithm

- Test all pairs of points of S
- If all other points are on left side, add this pair to an ordered list.

Complexity in $O(n^3)$

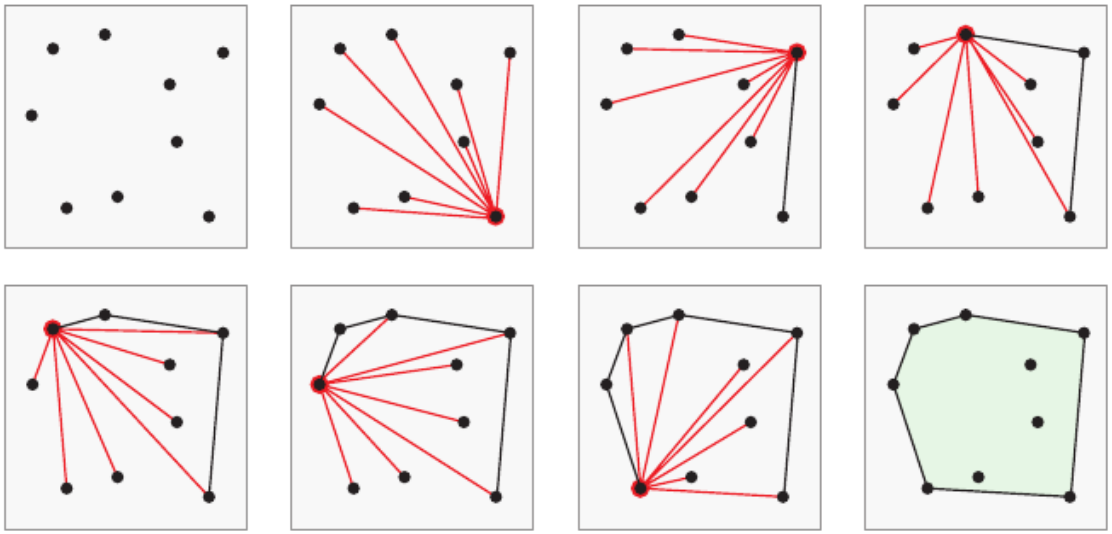
n is the number of points of S, can be really big.



Gift wrapping 1970 / Jarvis 1973

Algorithm

- Considers bottommost point.
- Evaluate angles with all other points from horizontal line.
- Choose biggest angle: this gives first edge.
- Iteratively, try to wrap all points.



Gift wrapping 1970 / Jarvis 1973

Algorithm

- Considers bottommost point.
- Evaluate angles with all other points from horizontal line.
- Choose biggest angle: this gives first edge.
- Iteratively, try to wrap all points.

Complexity

- n points in S, h points in $conv(S)$
- $O(n \cdot h)$
- worst case: $O(n^2)$ (all points in hull)
- output-sensitive (h)

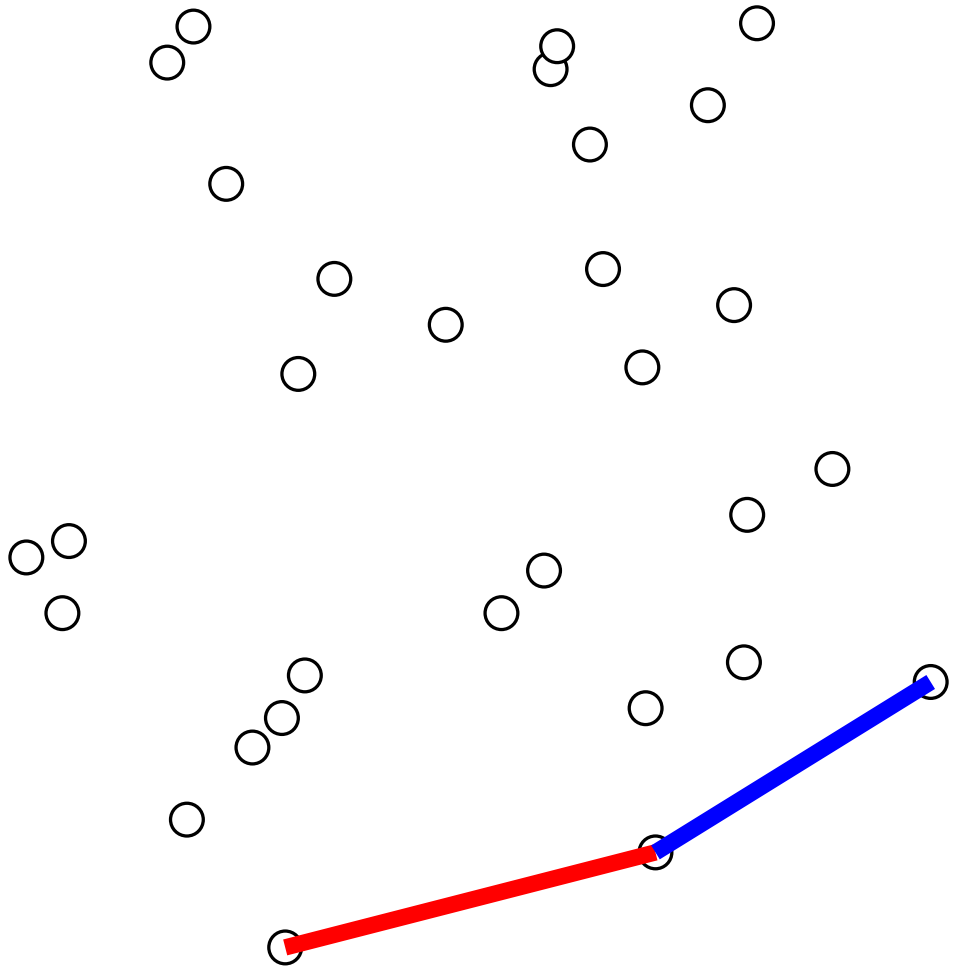
Graham scan 1972

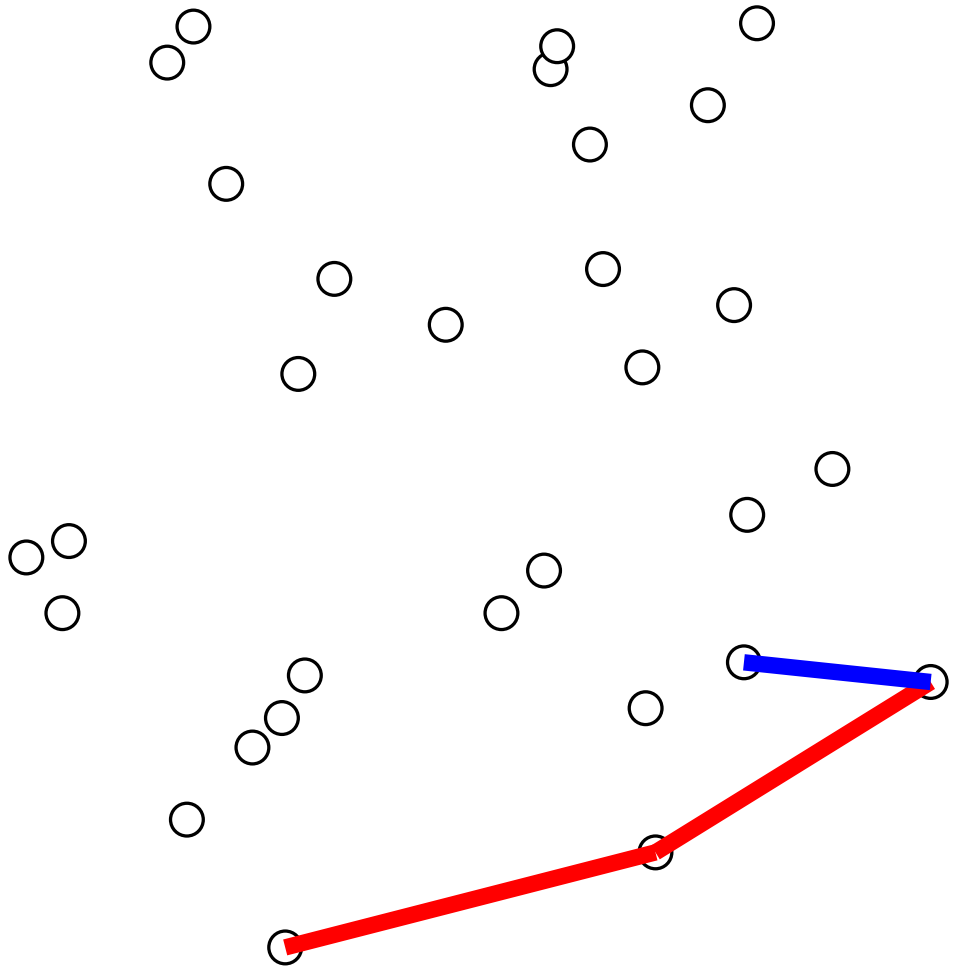
Algorithm

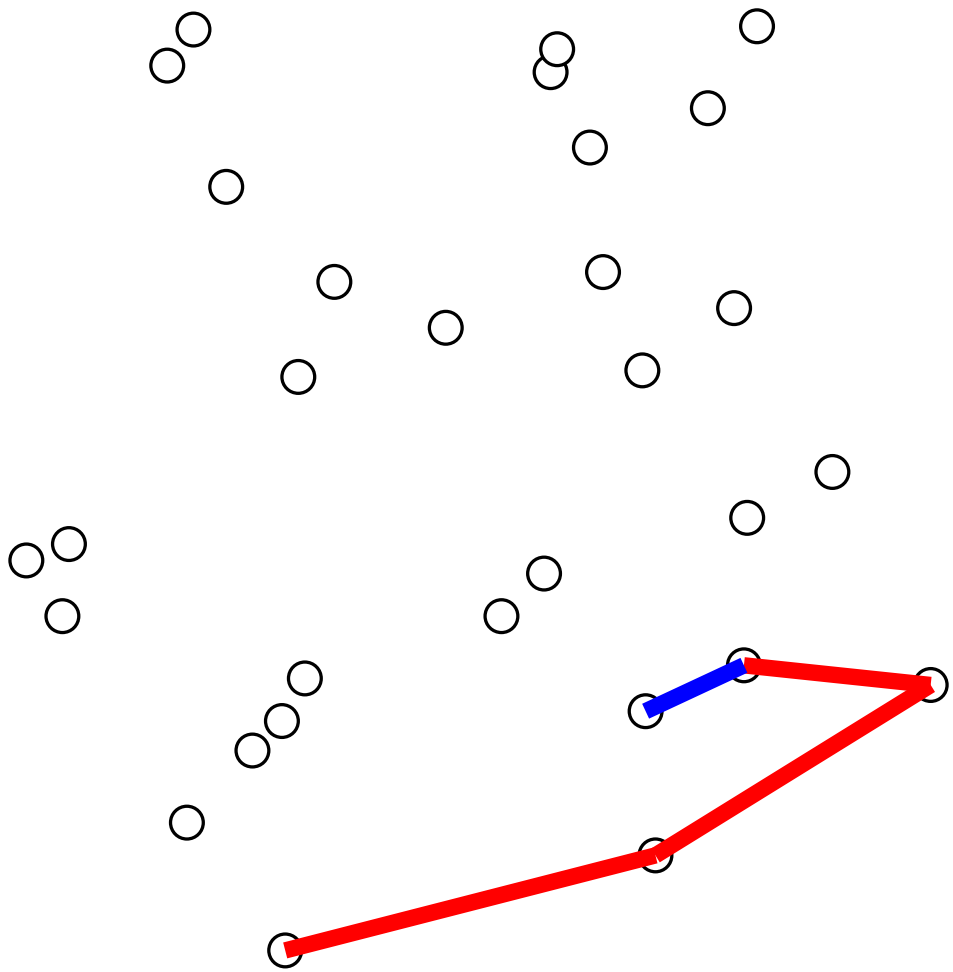
- Considers bottommost point.
- Sort all other points by angle from horizontal line.
- Consider them in this order and add to convex hull according to angle sign.

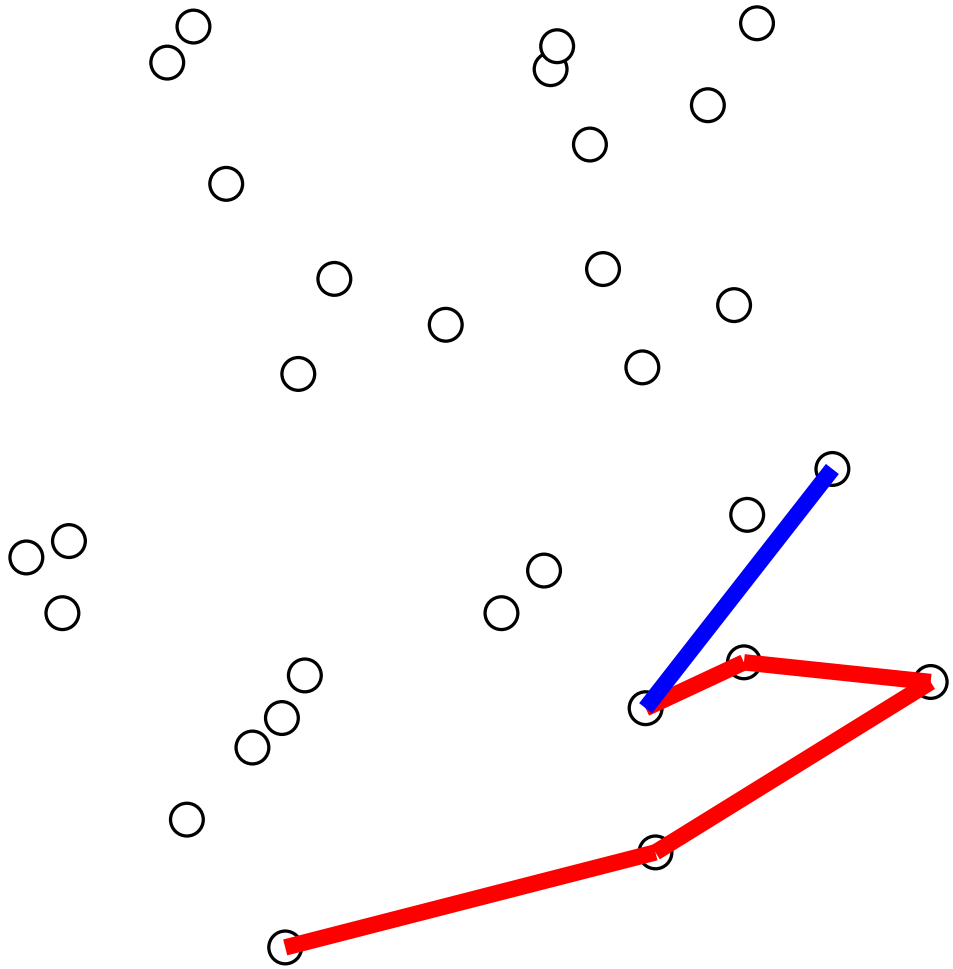
Complexity

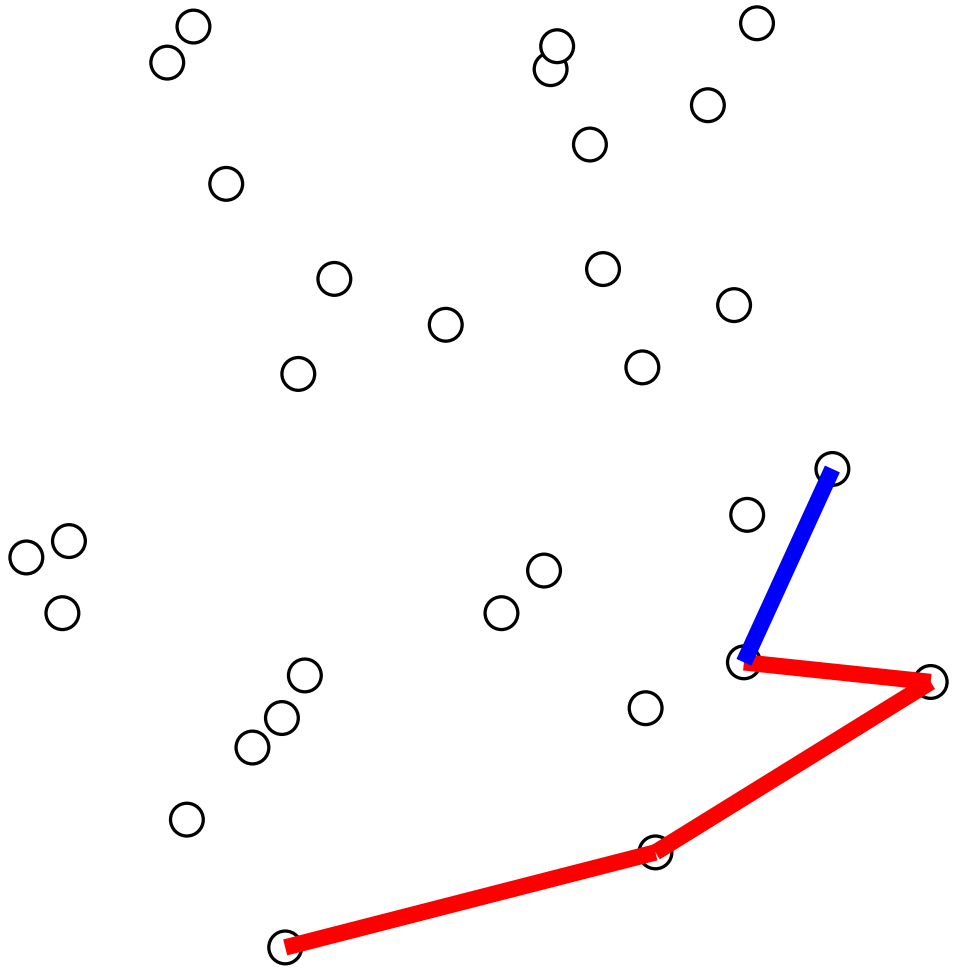
- n points in S, h points in $conv(S)$
- $O(n \cdot \log(n))$
- $O(n)$ if points are already sorted (coordinates or angles)

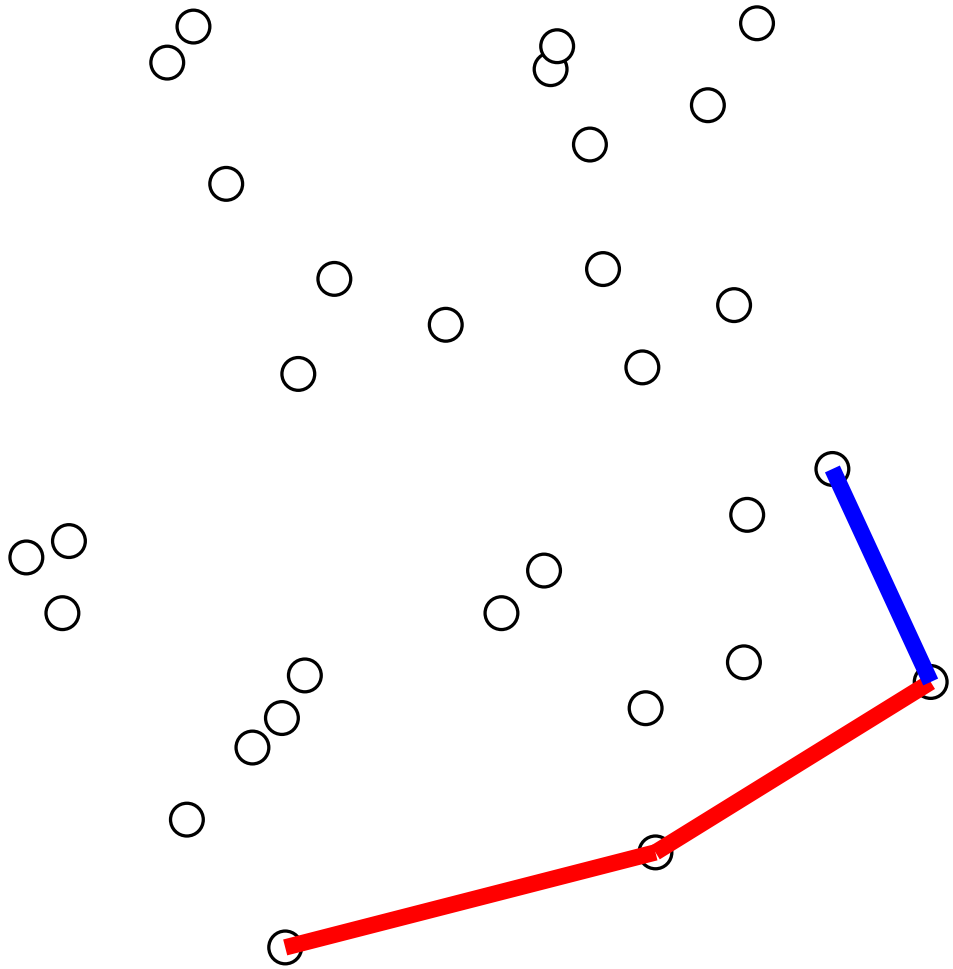


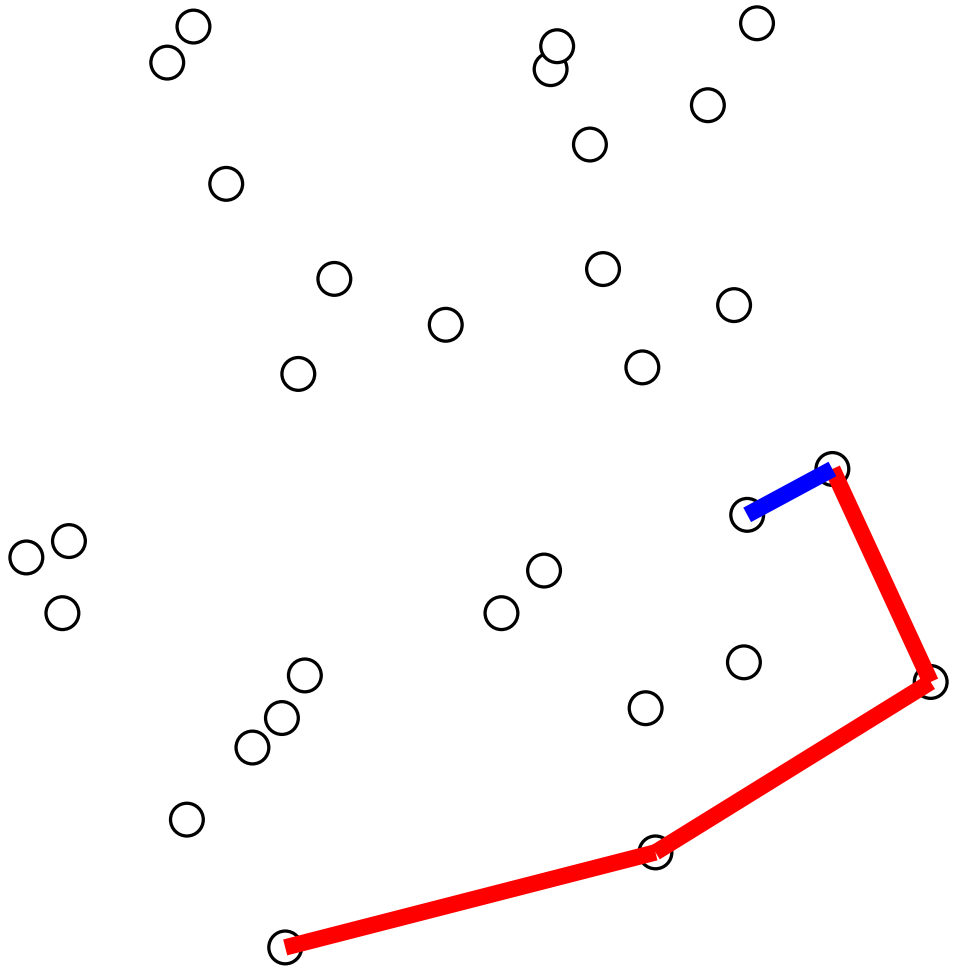


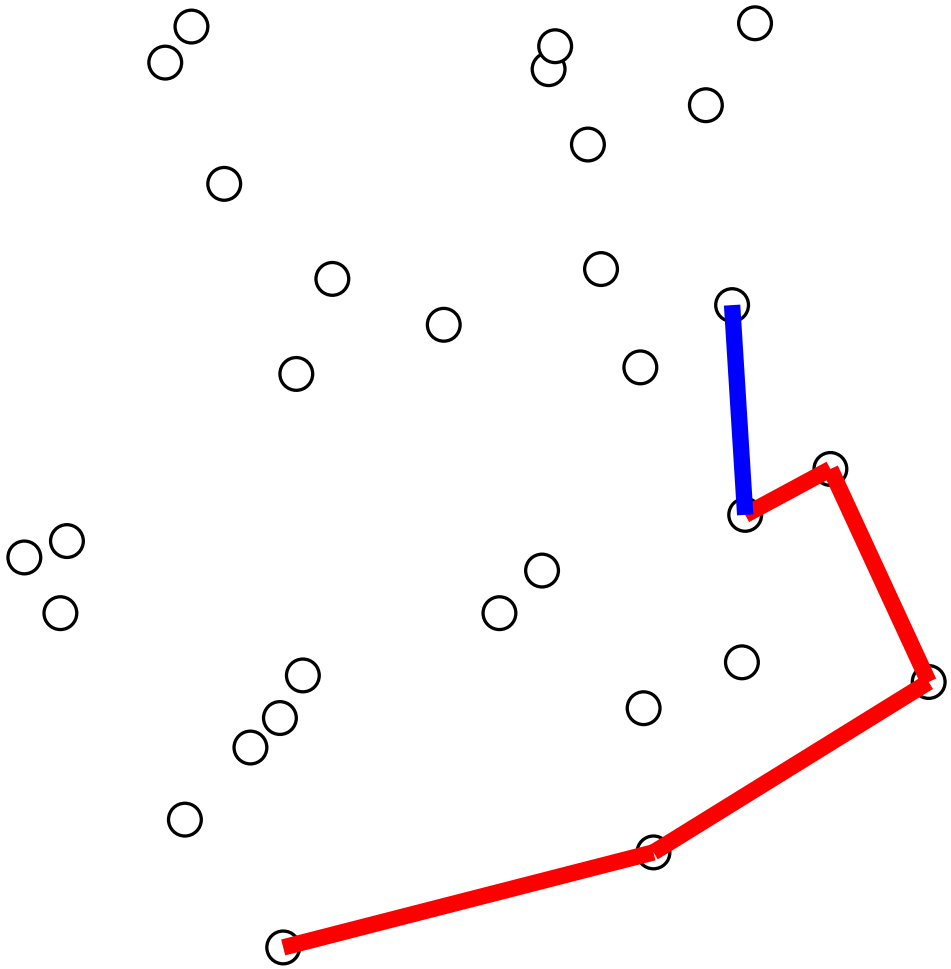


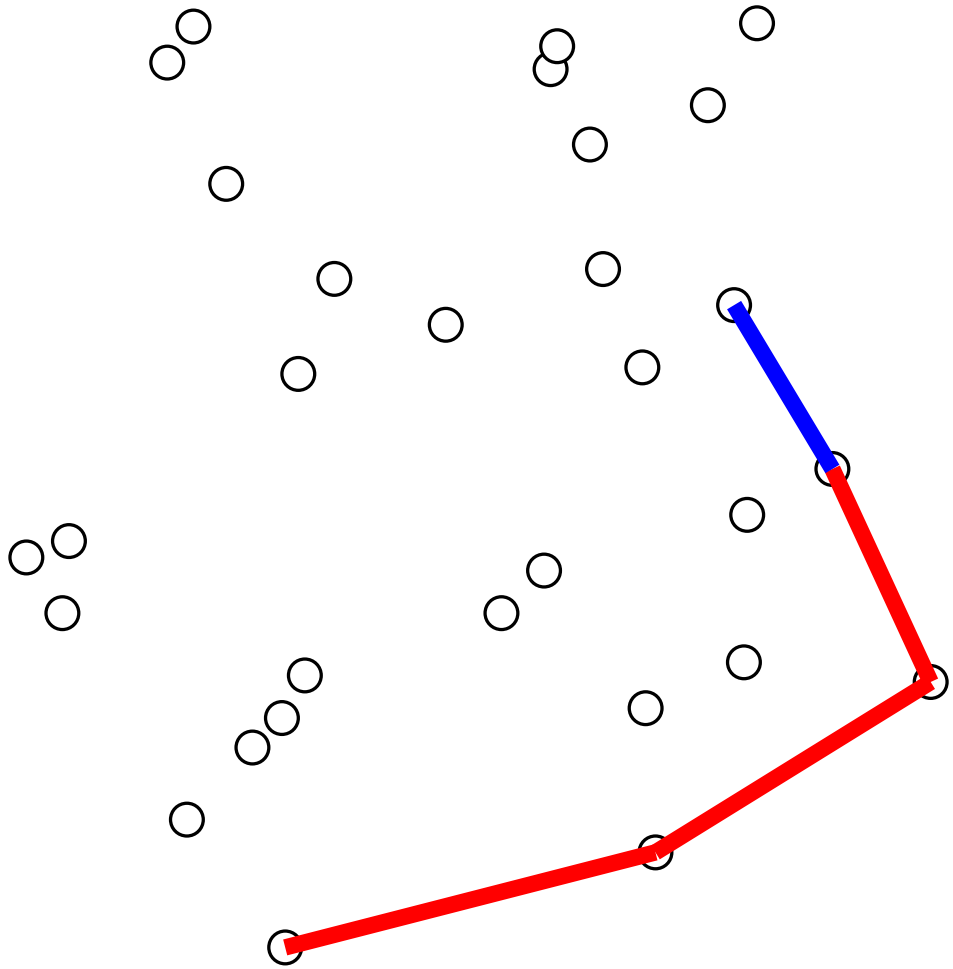


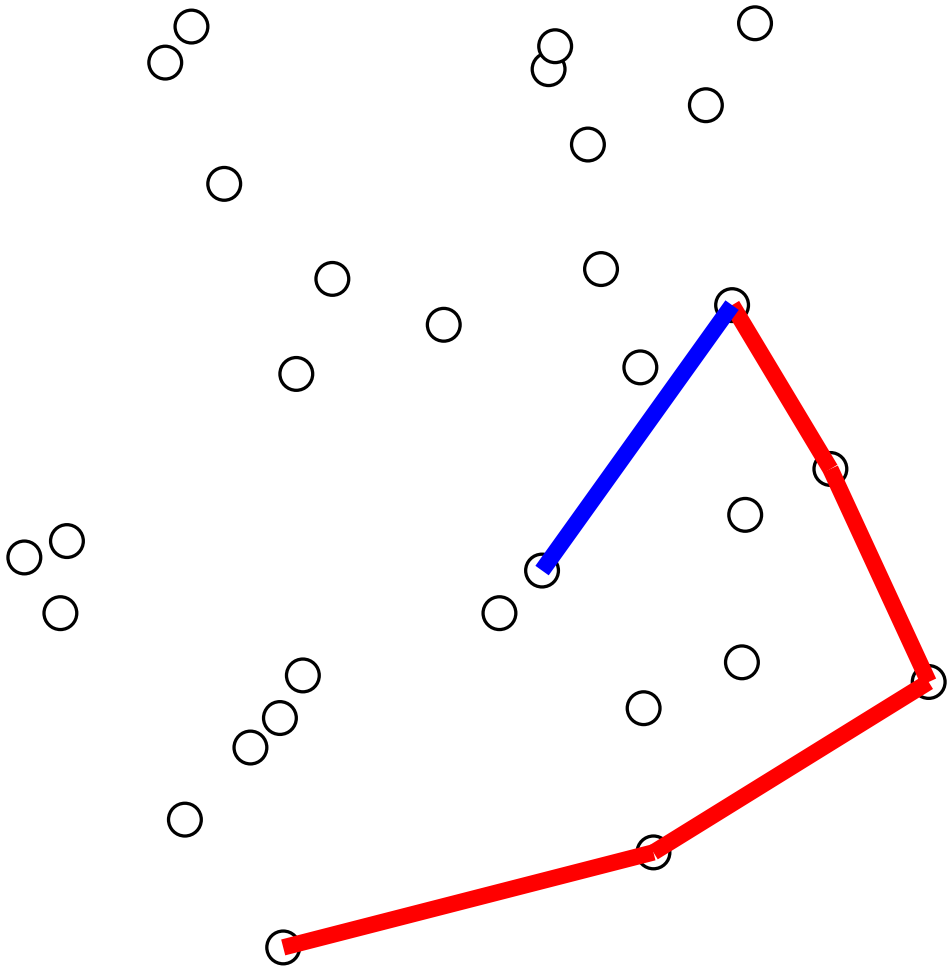


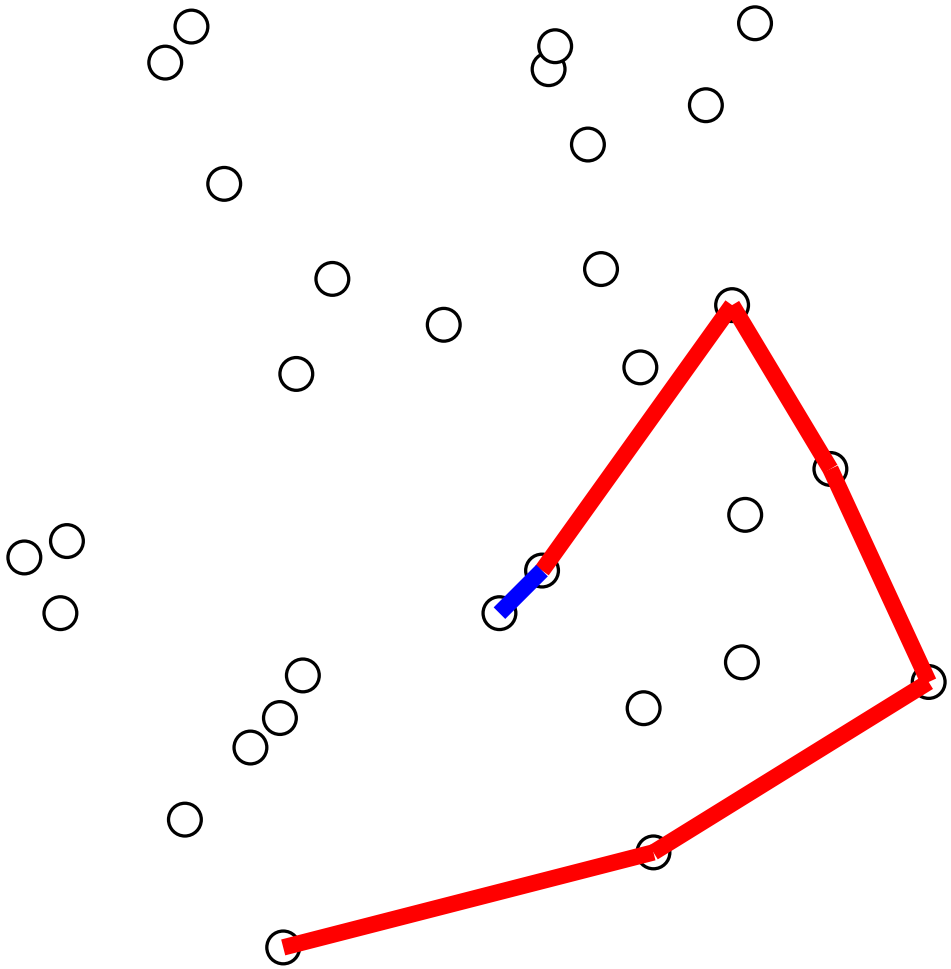


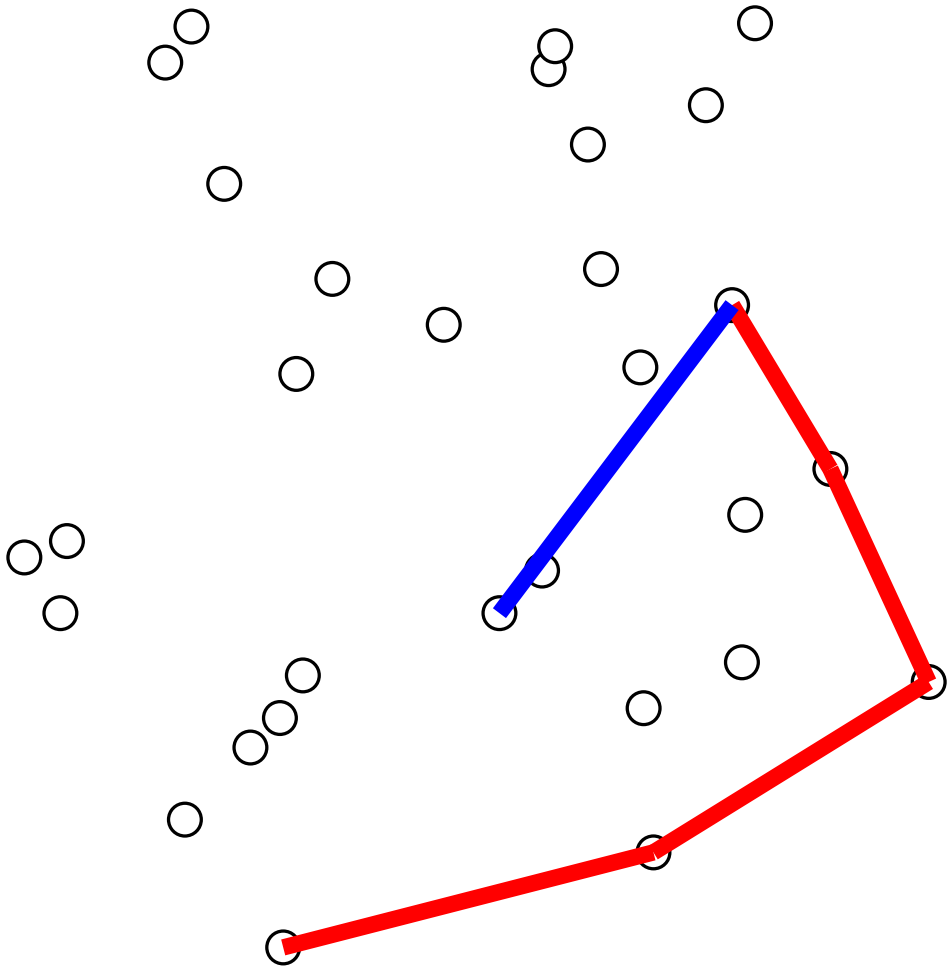


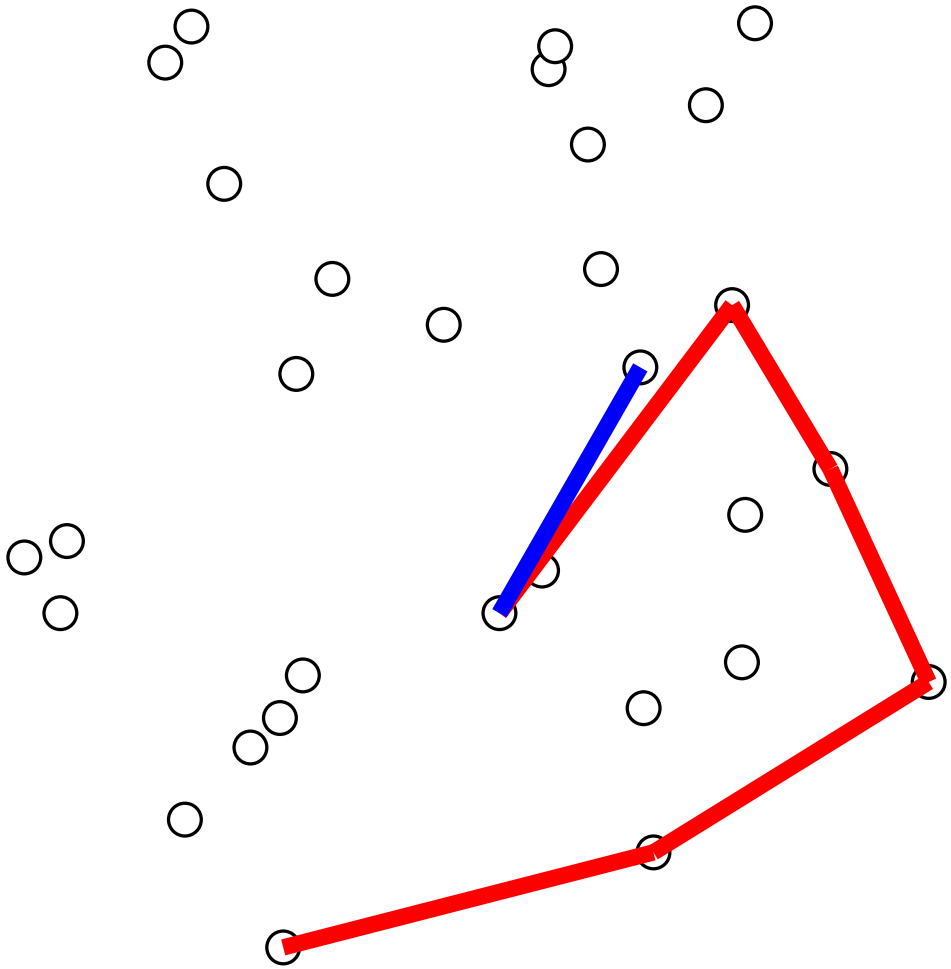


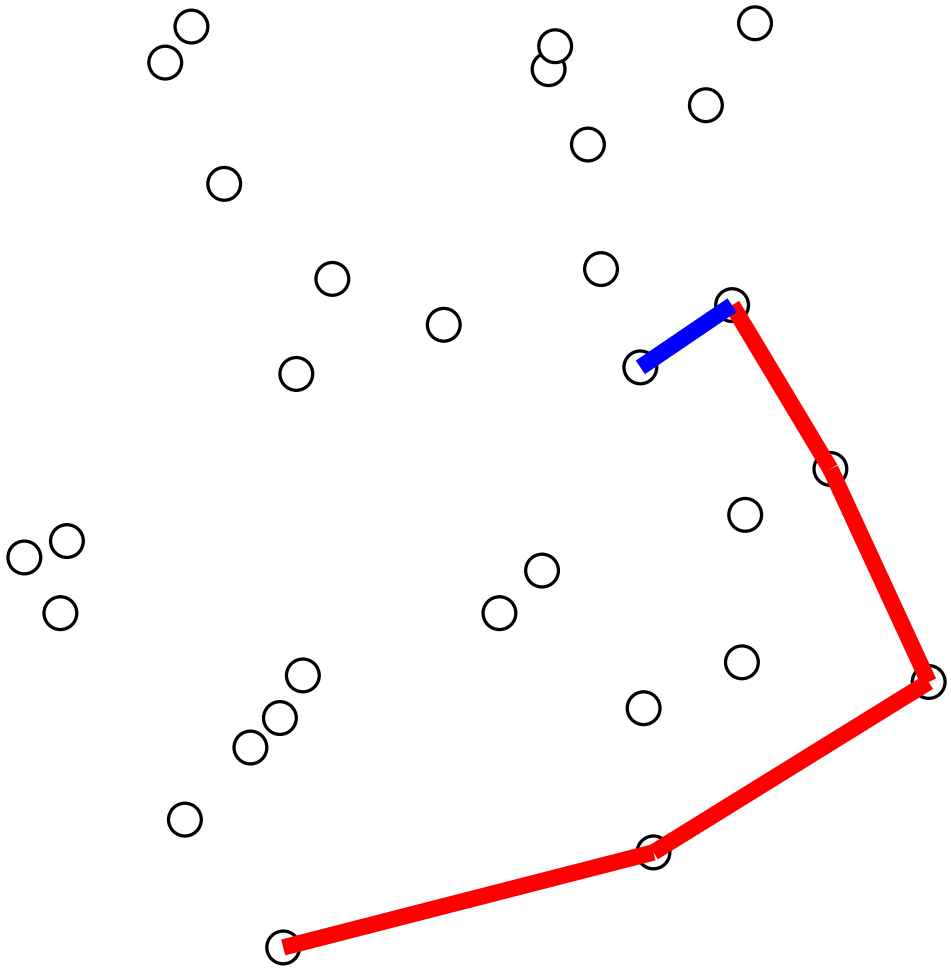


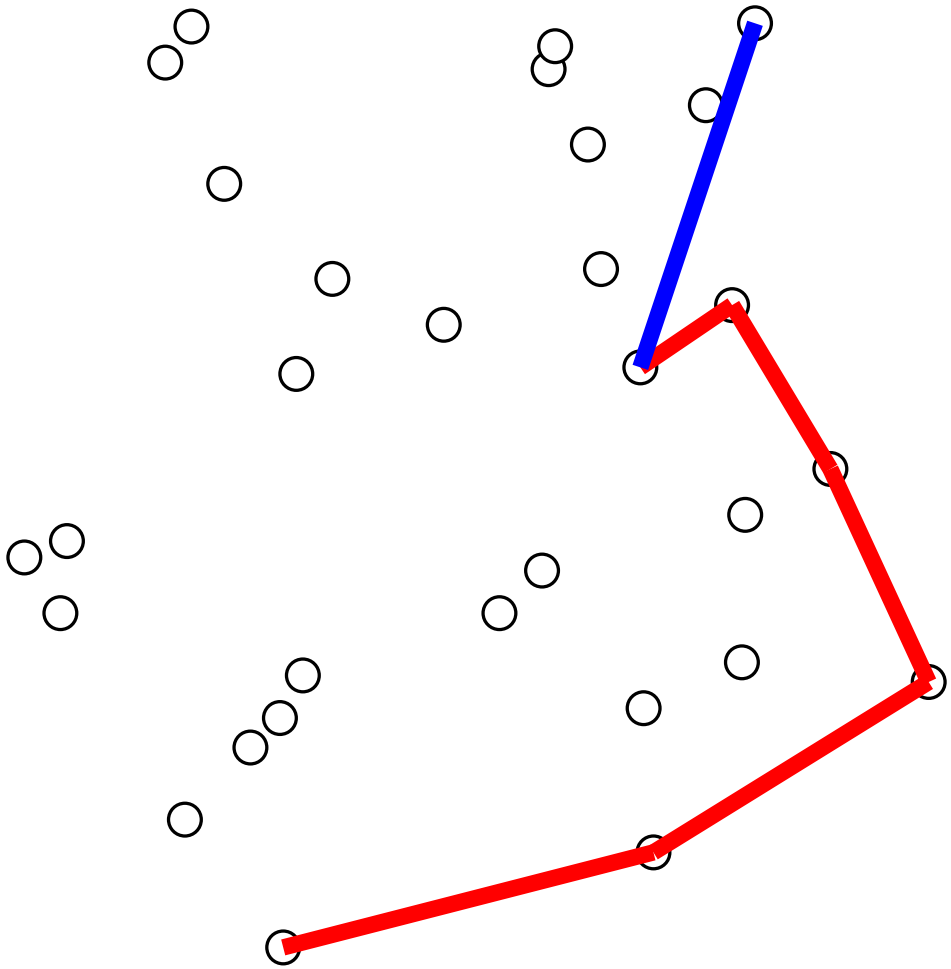


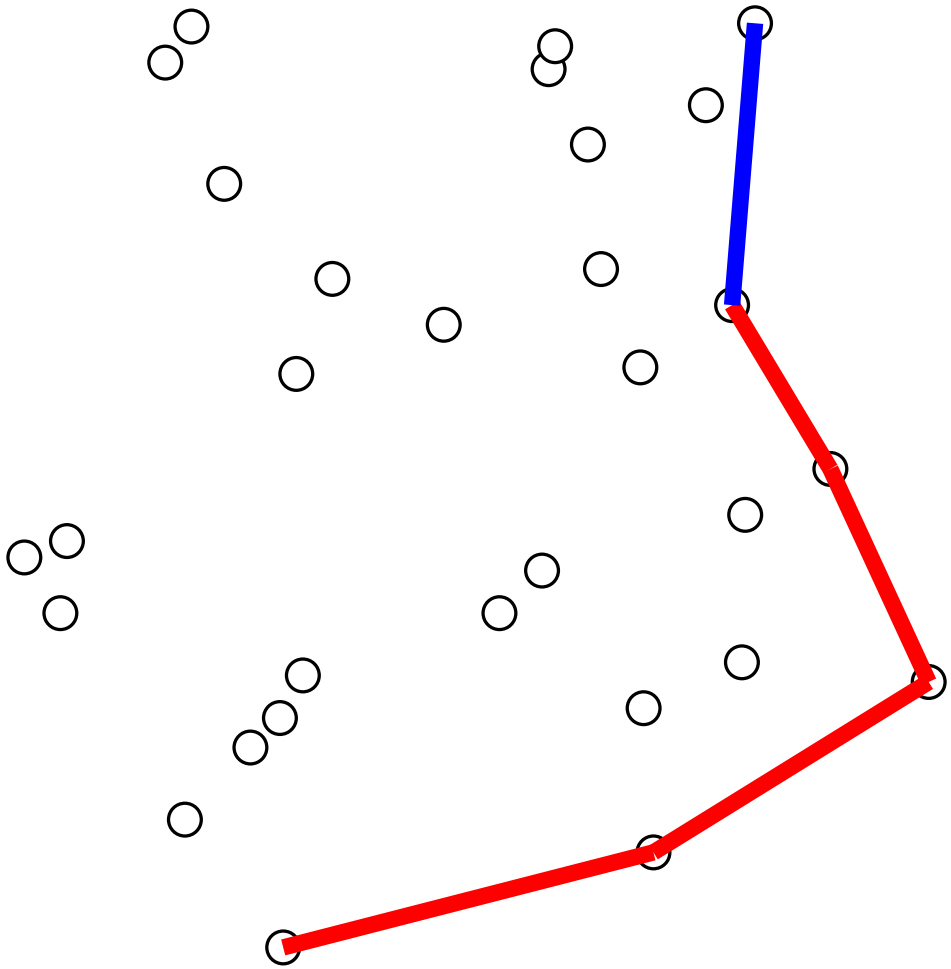


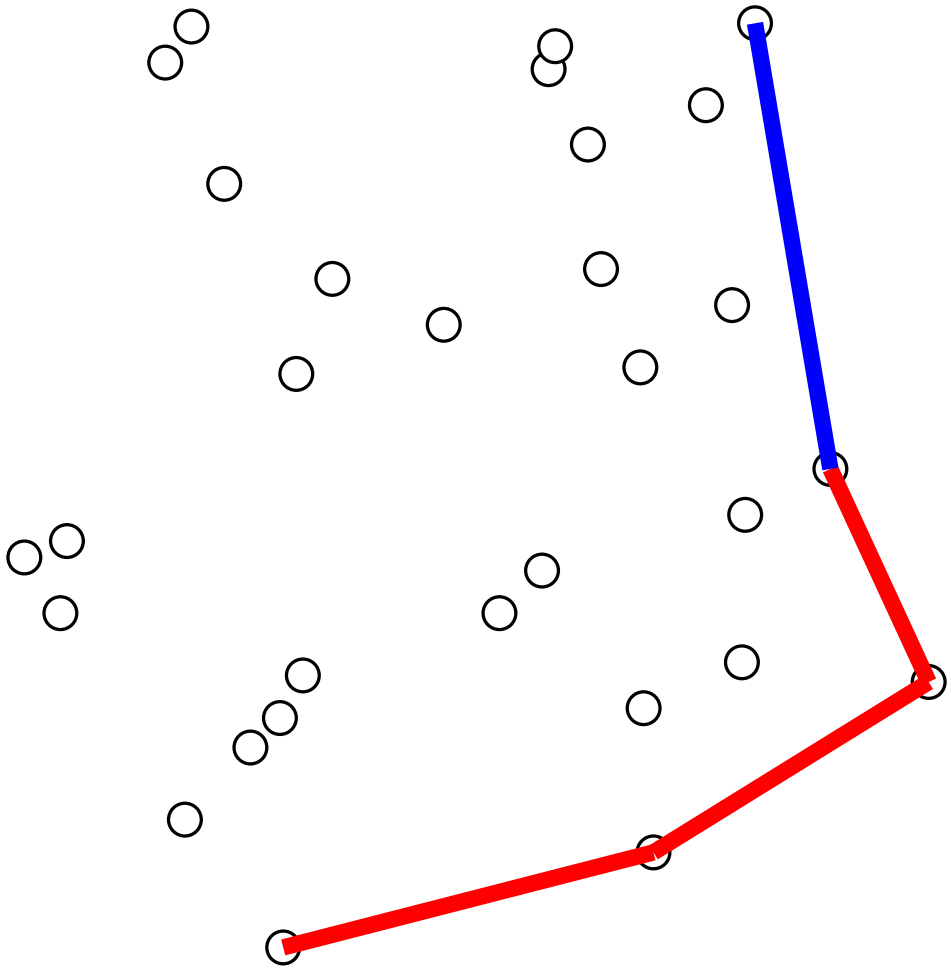


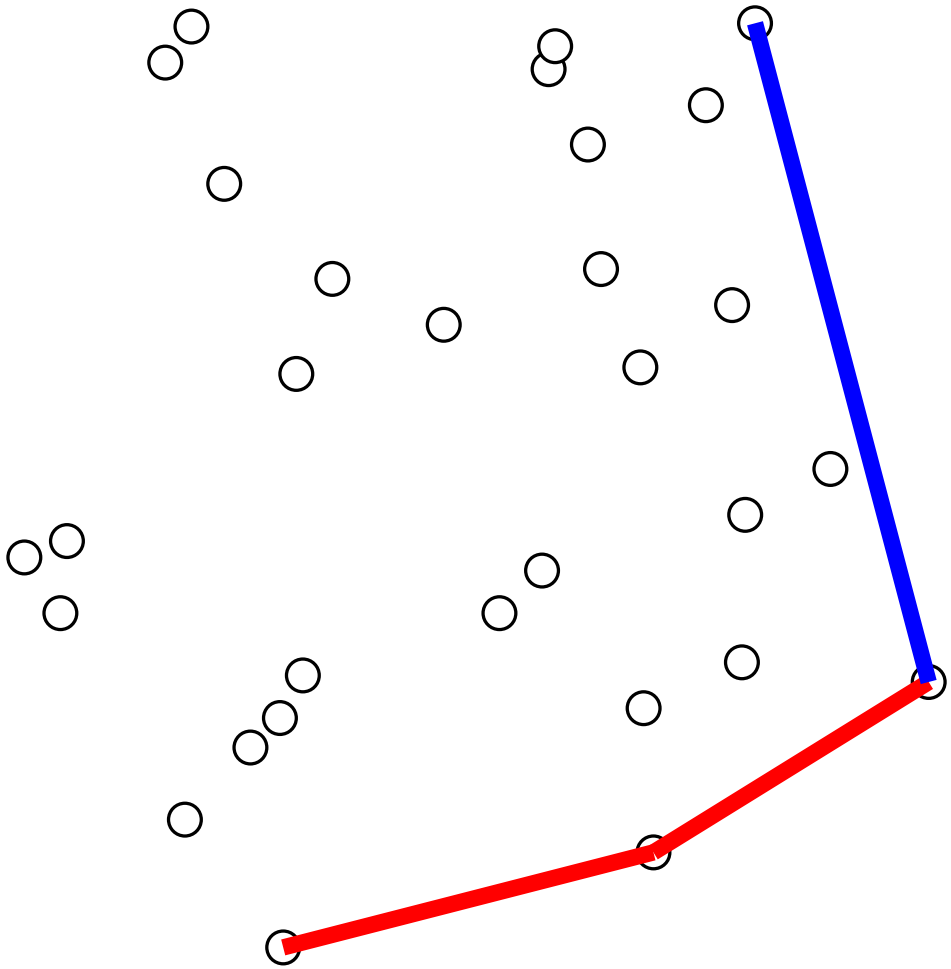


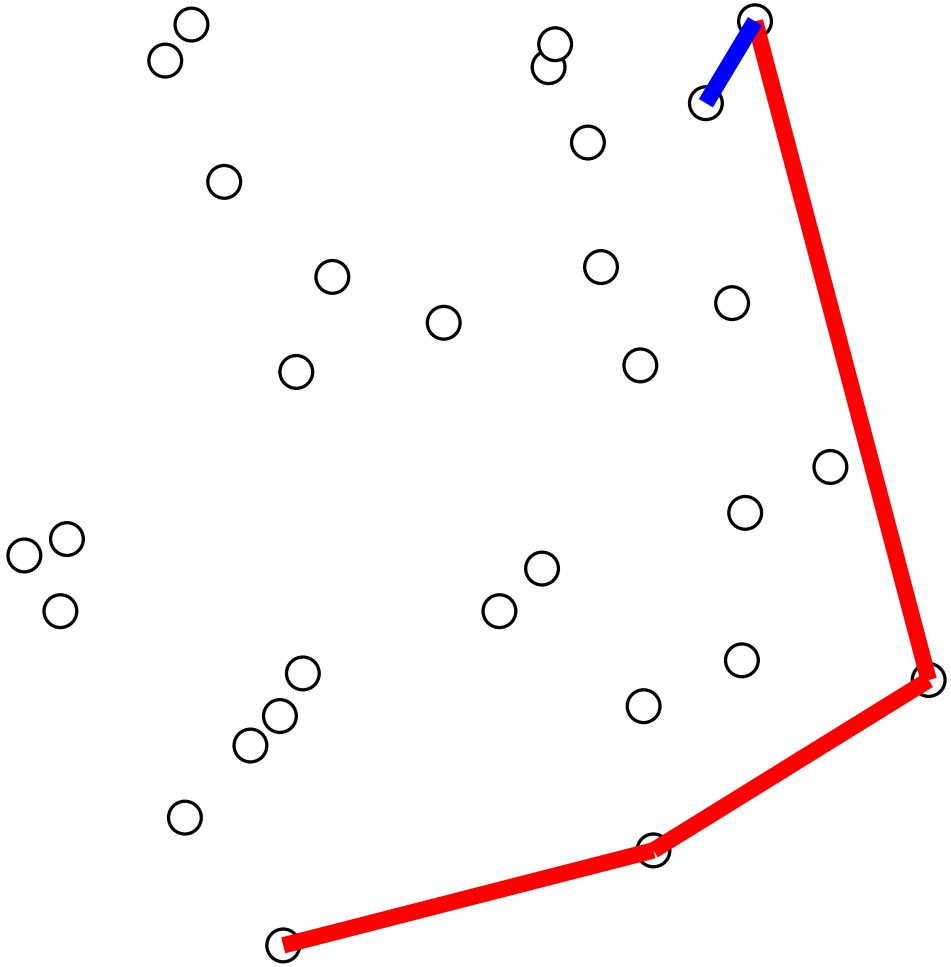


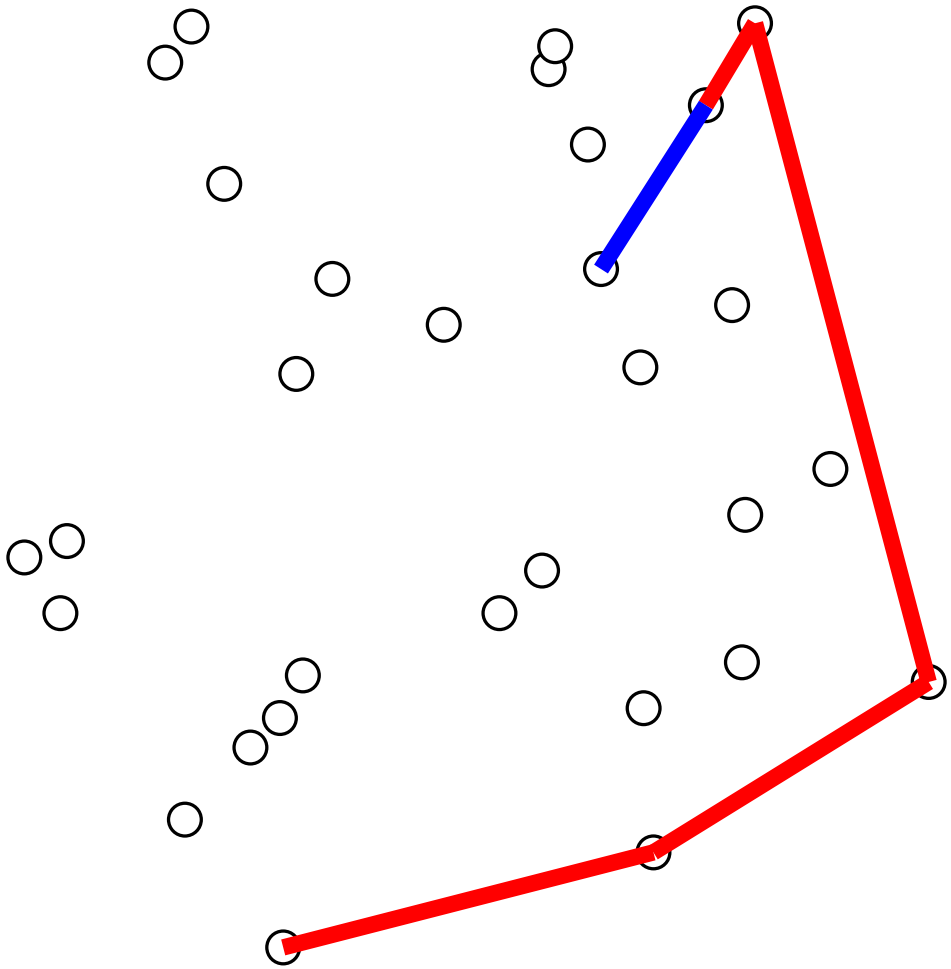


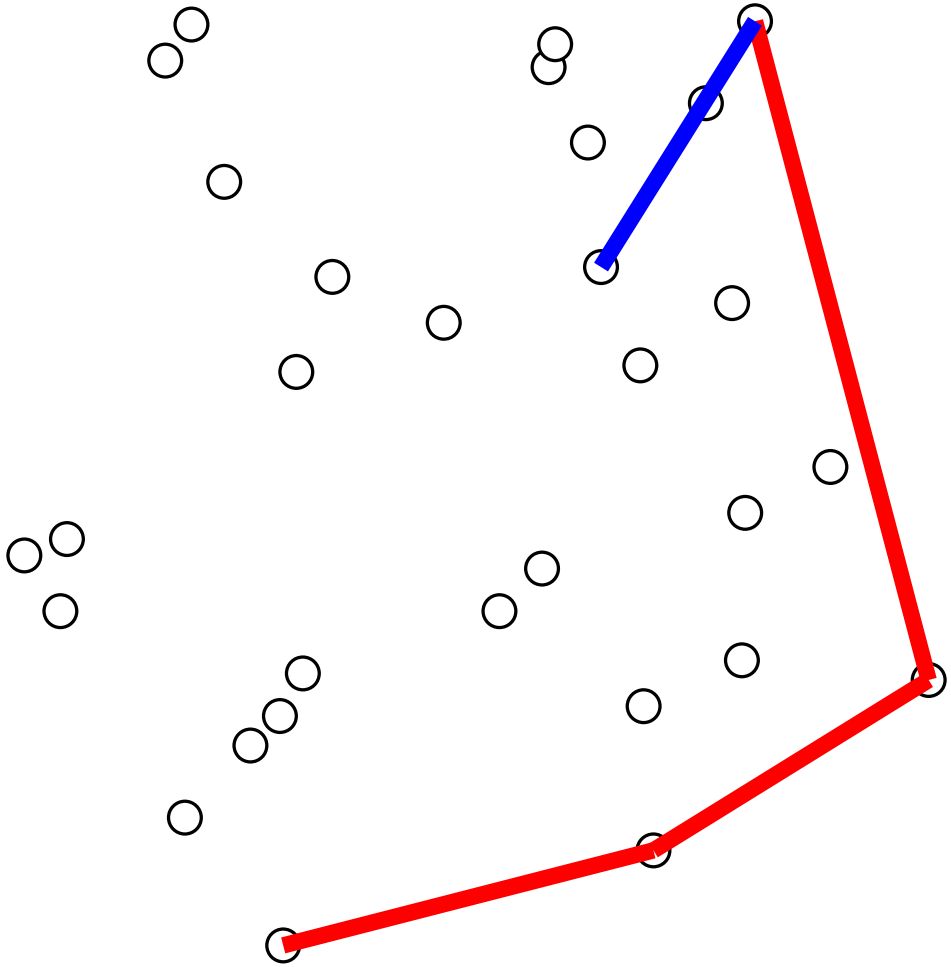


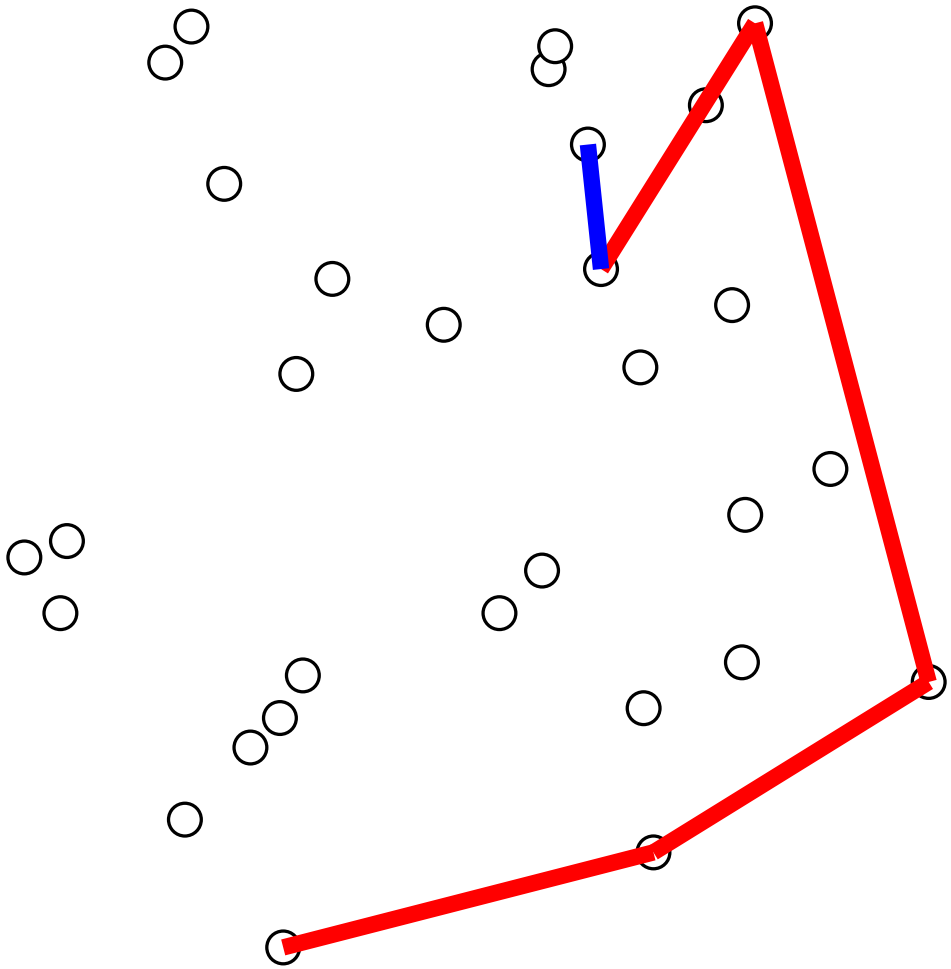


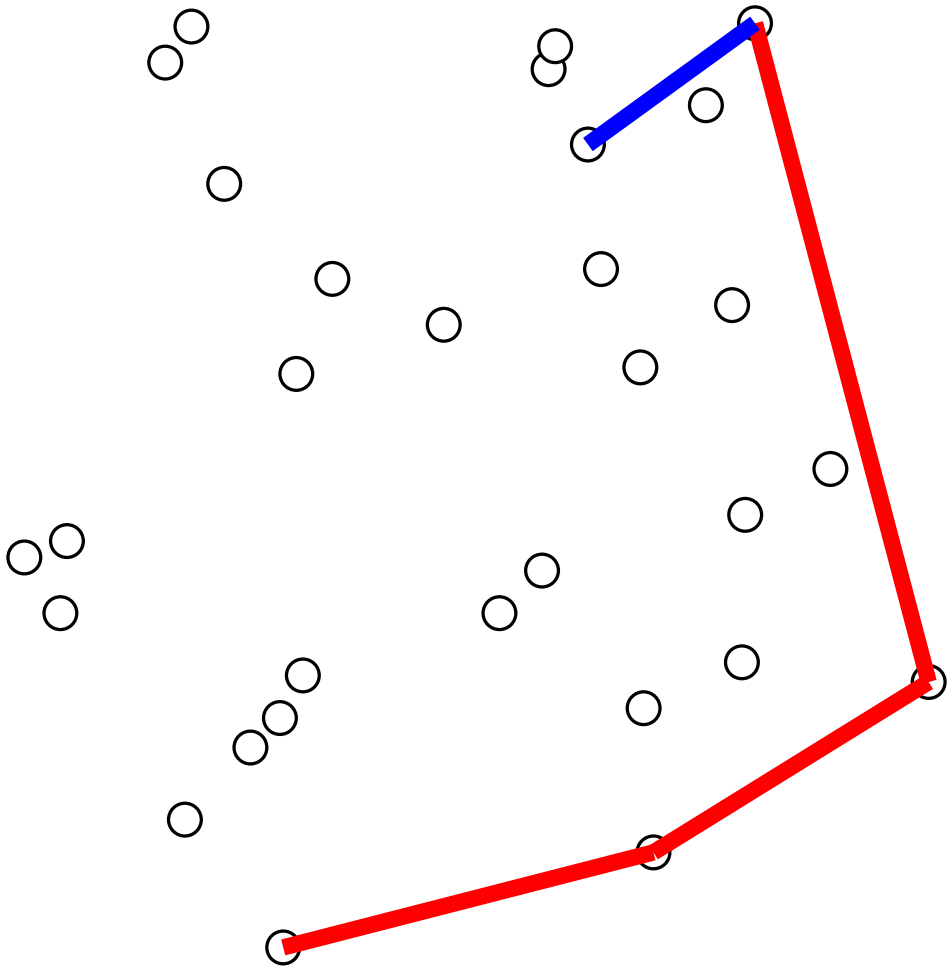


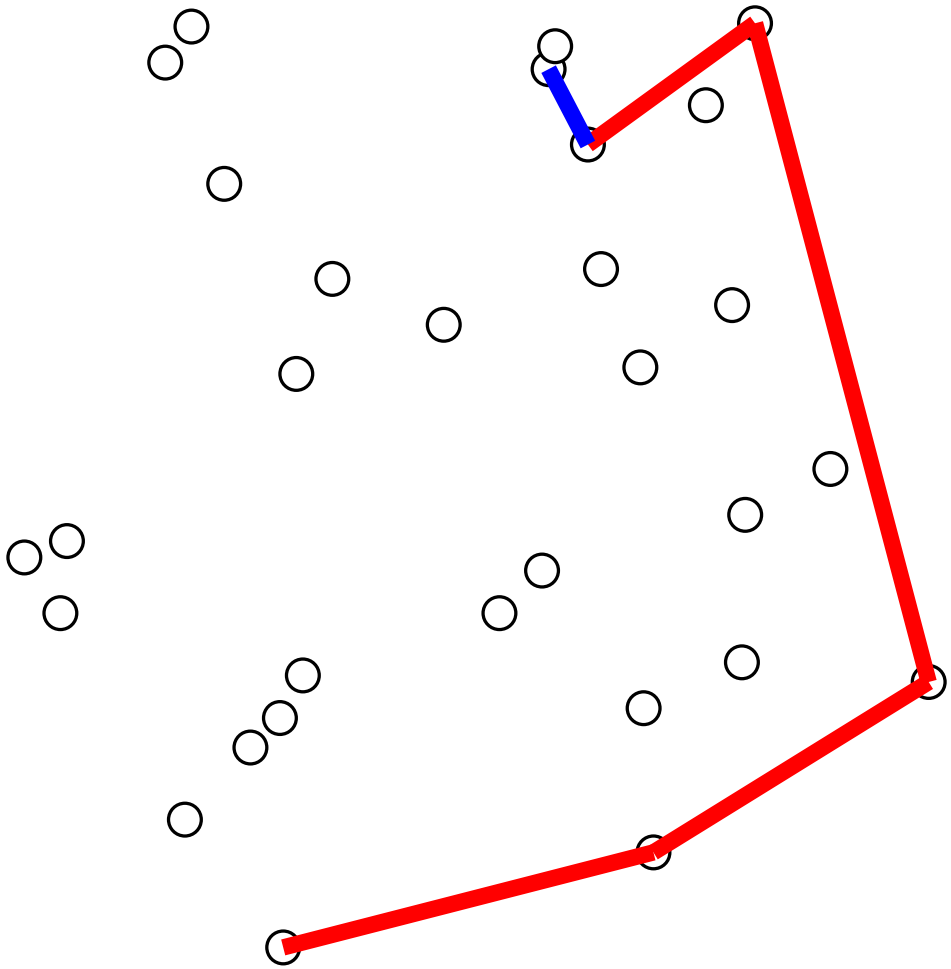


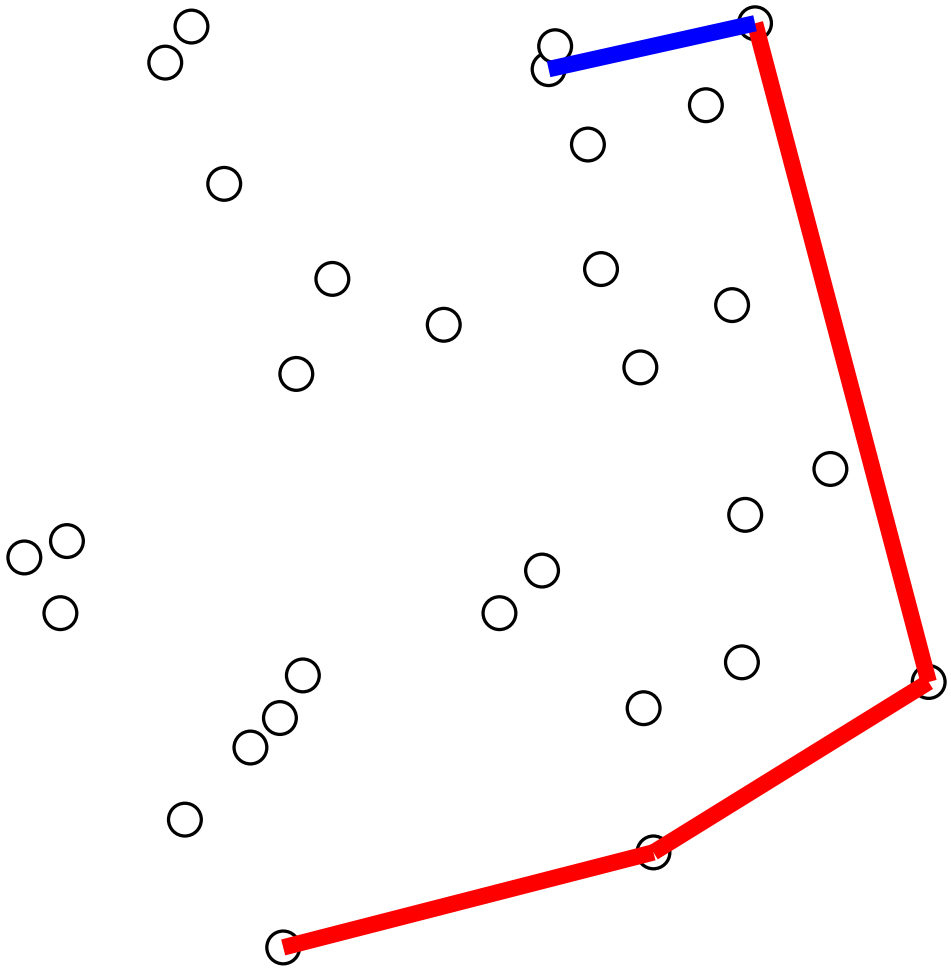


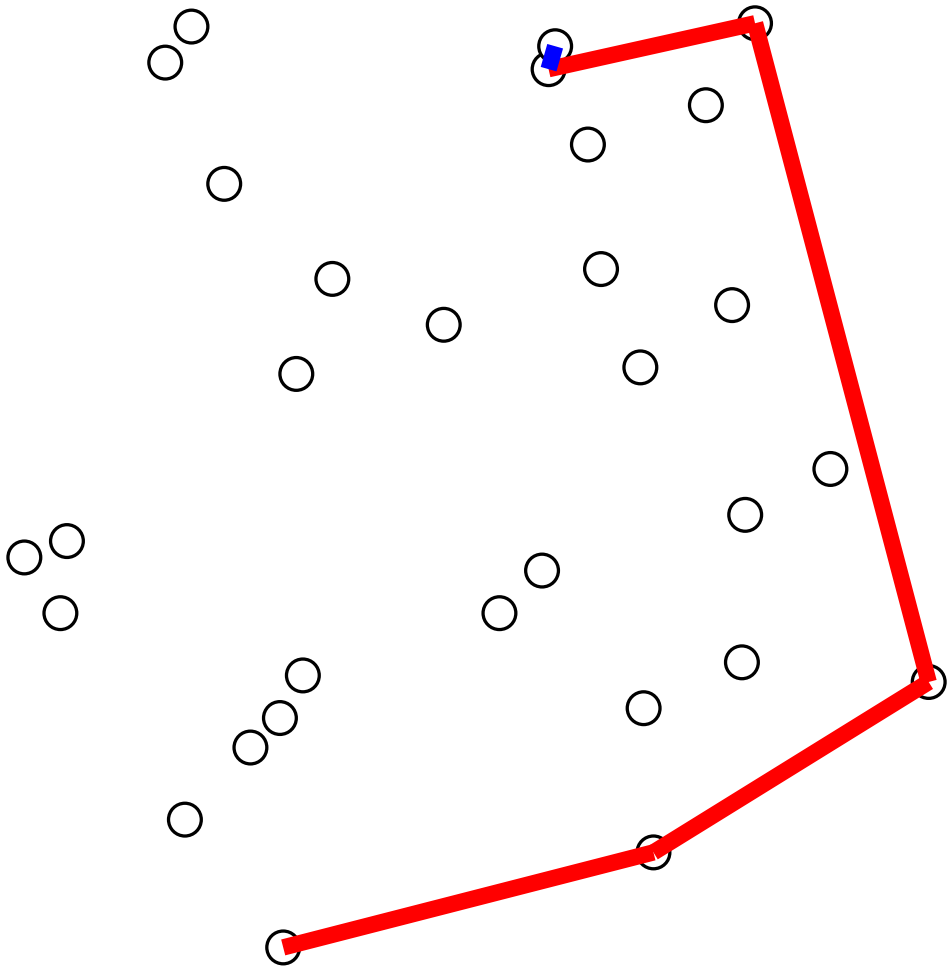


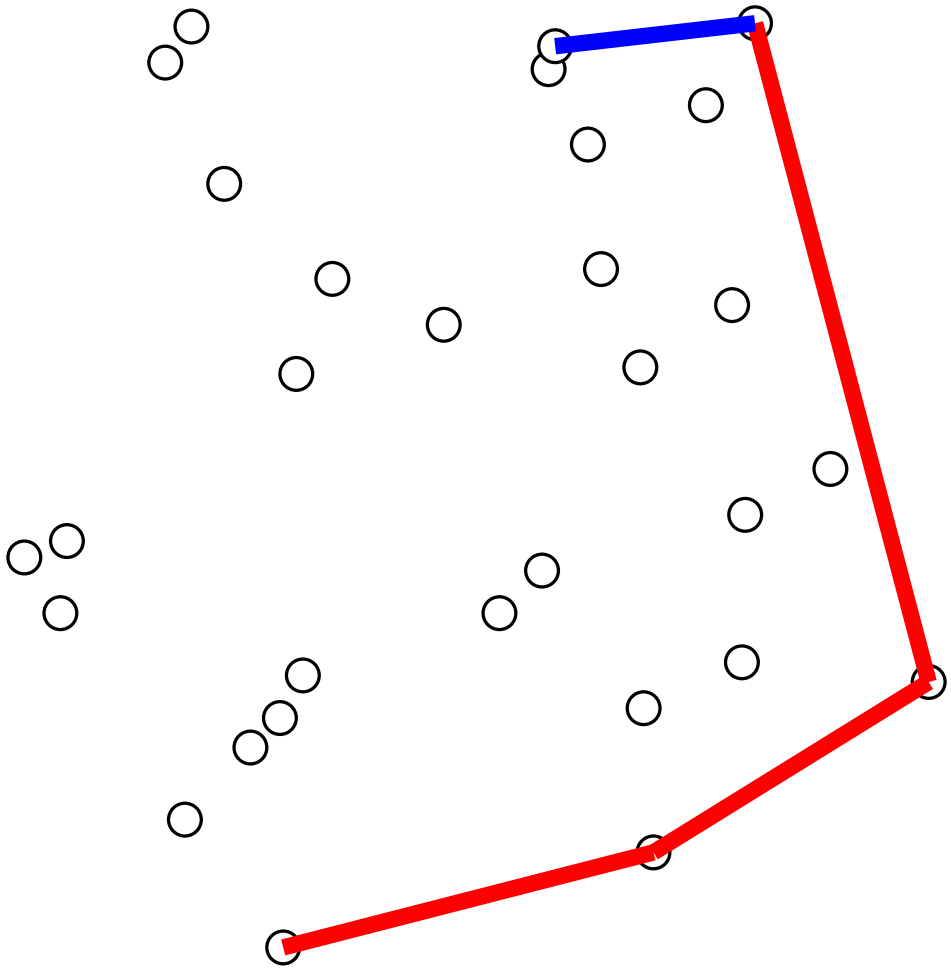


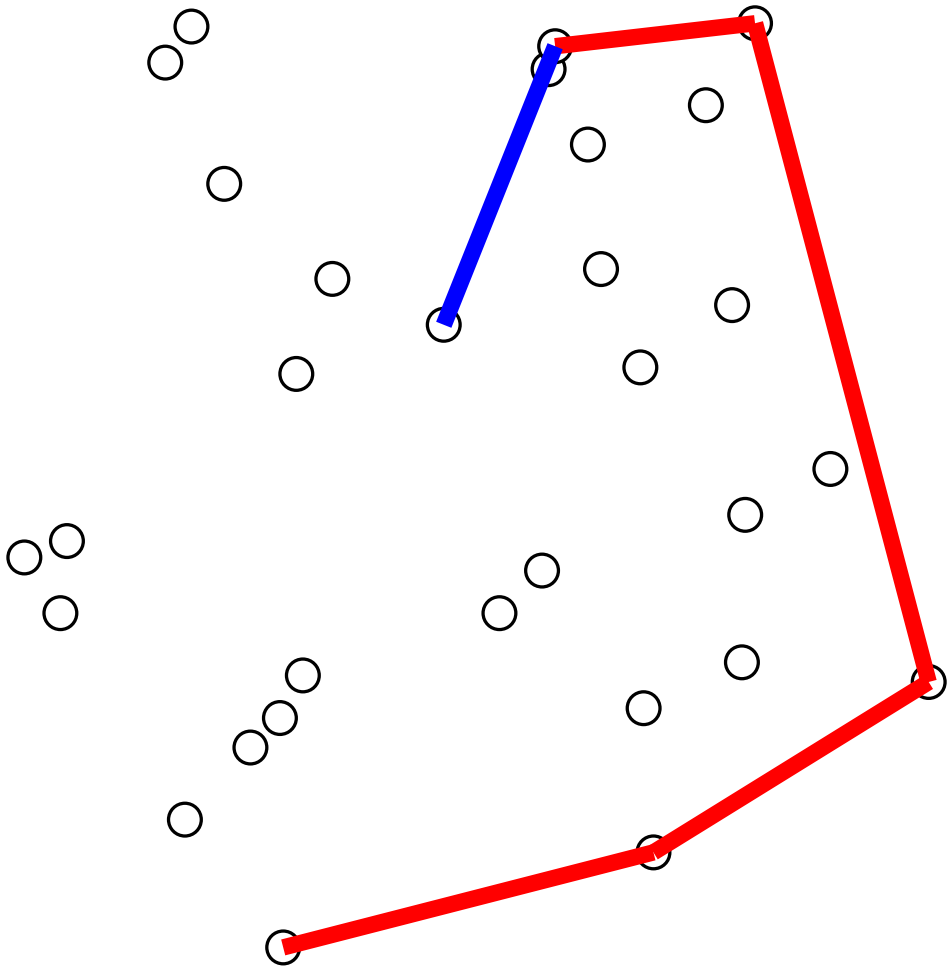


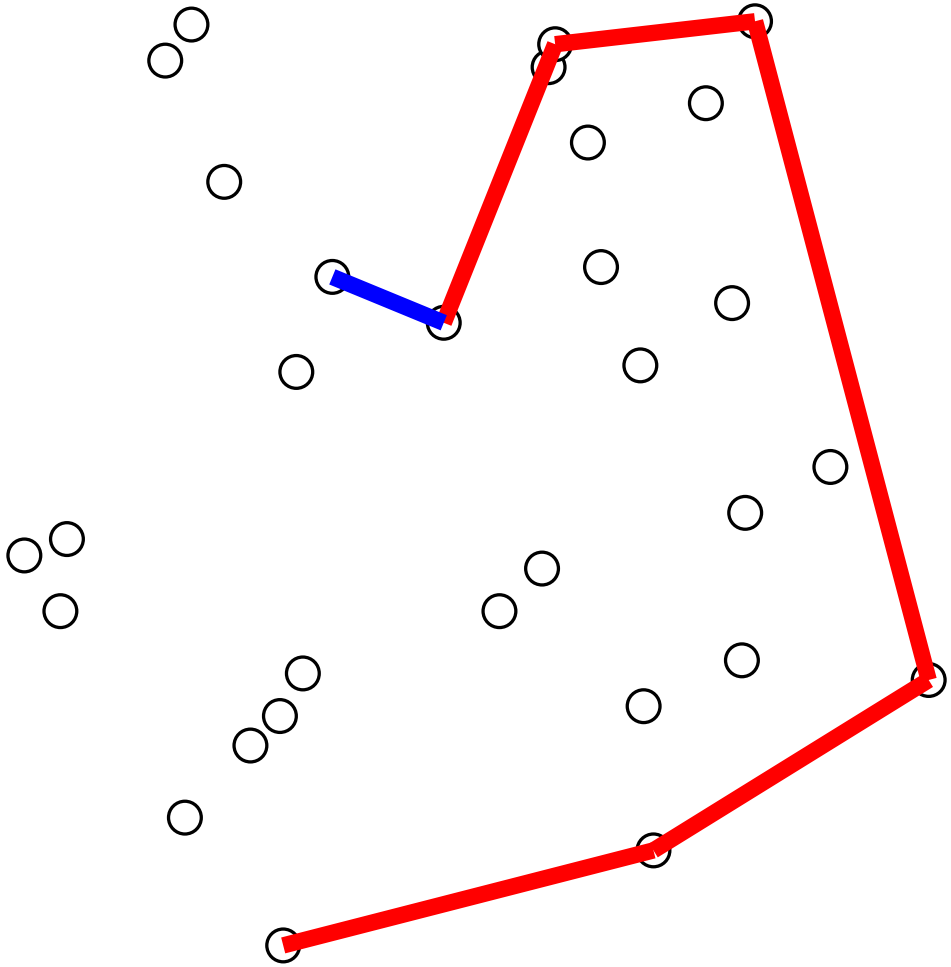


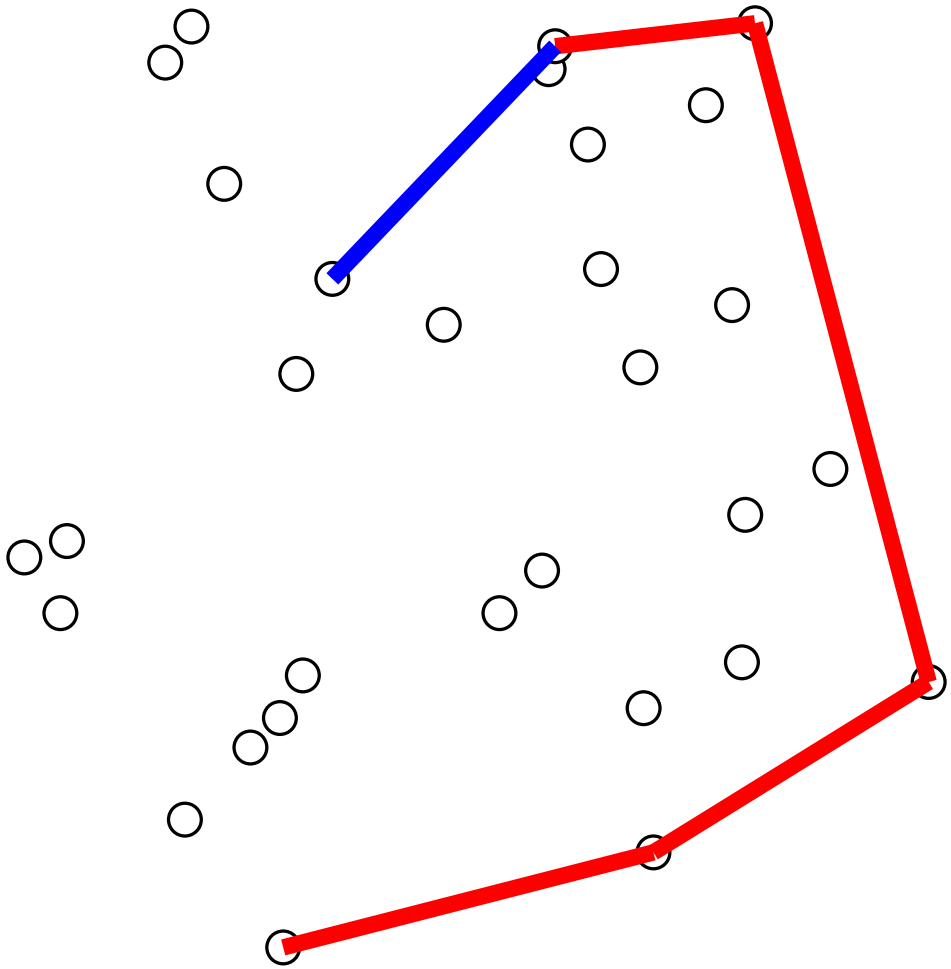


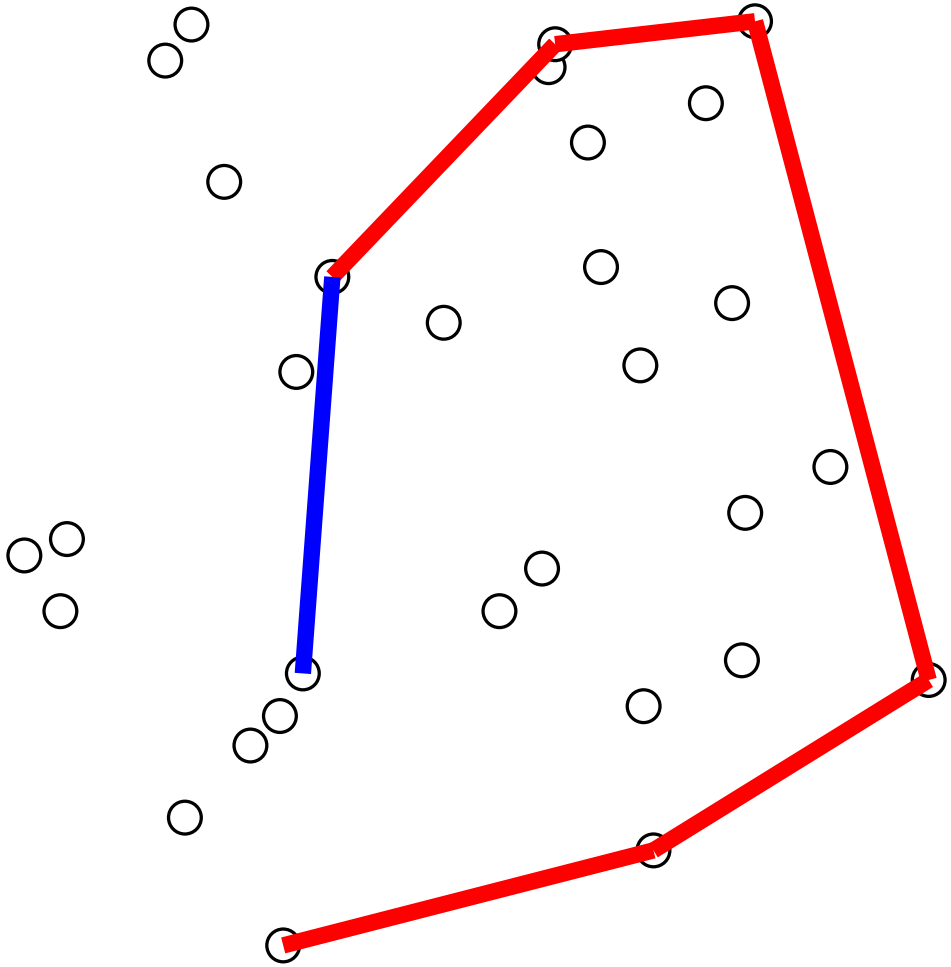


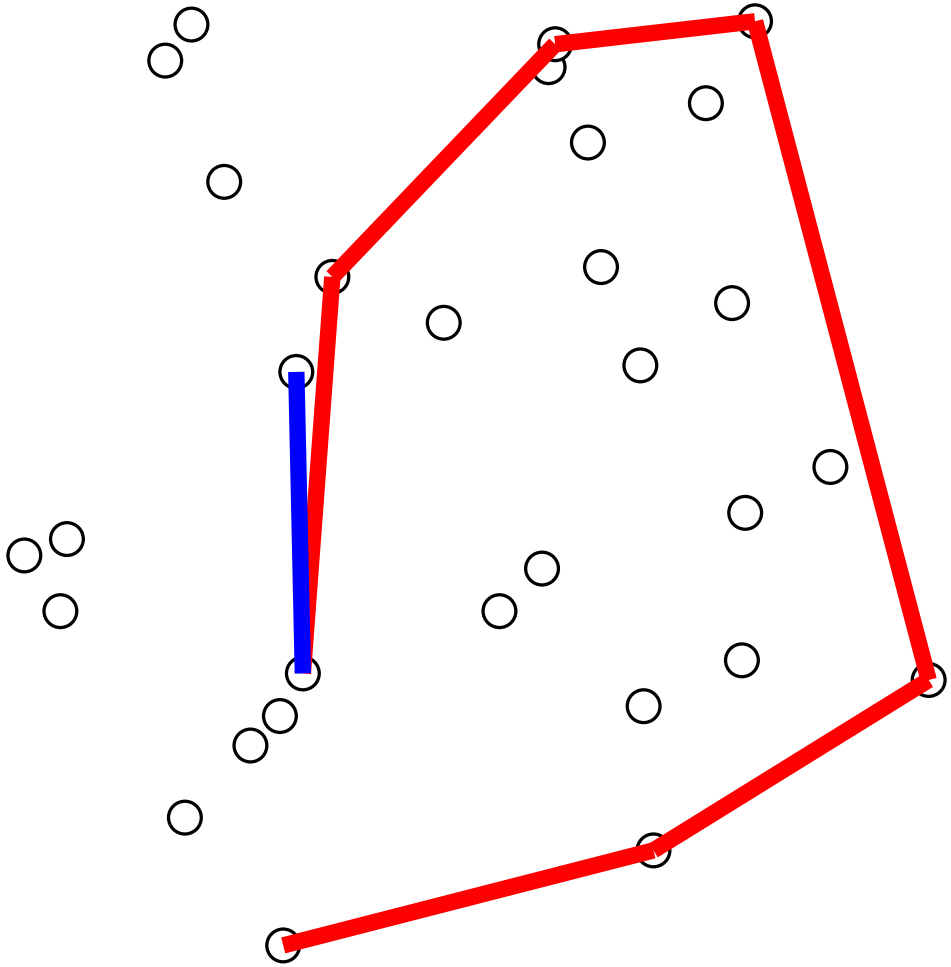


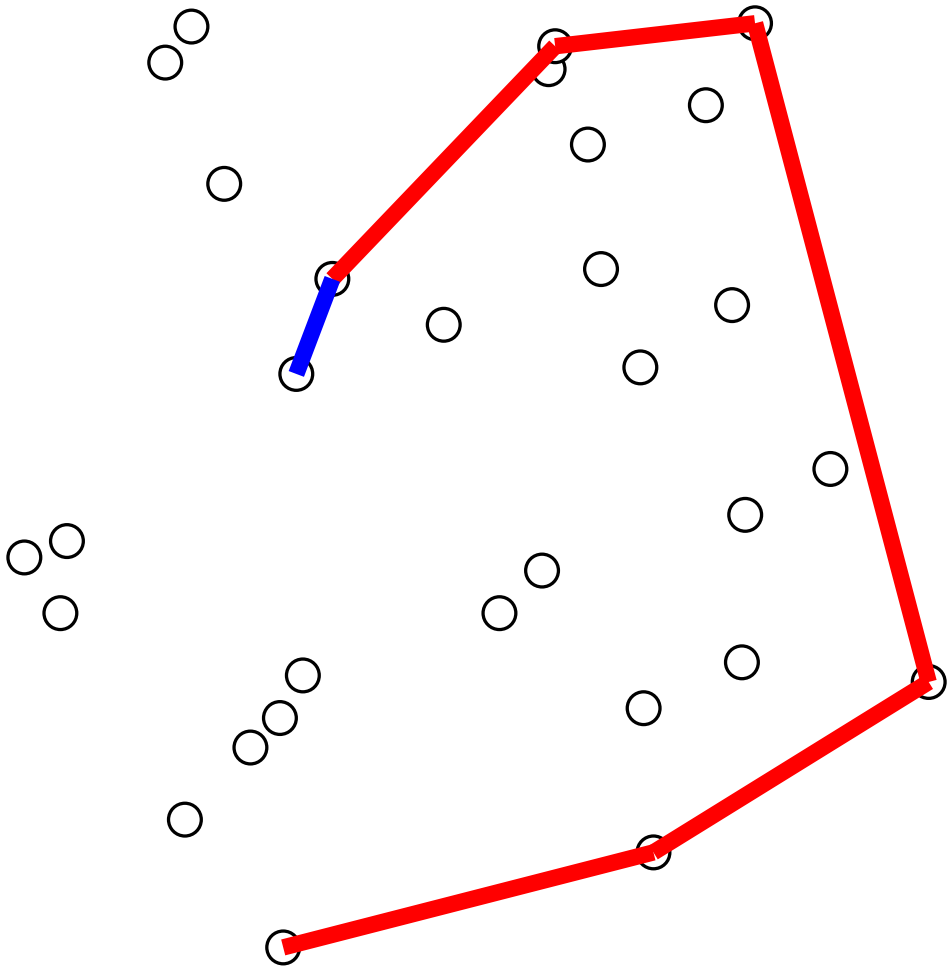


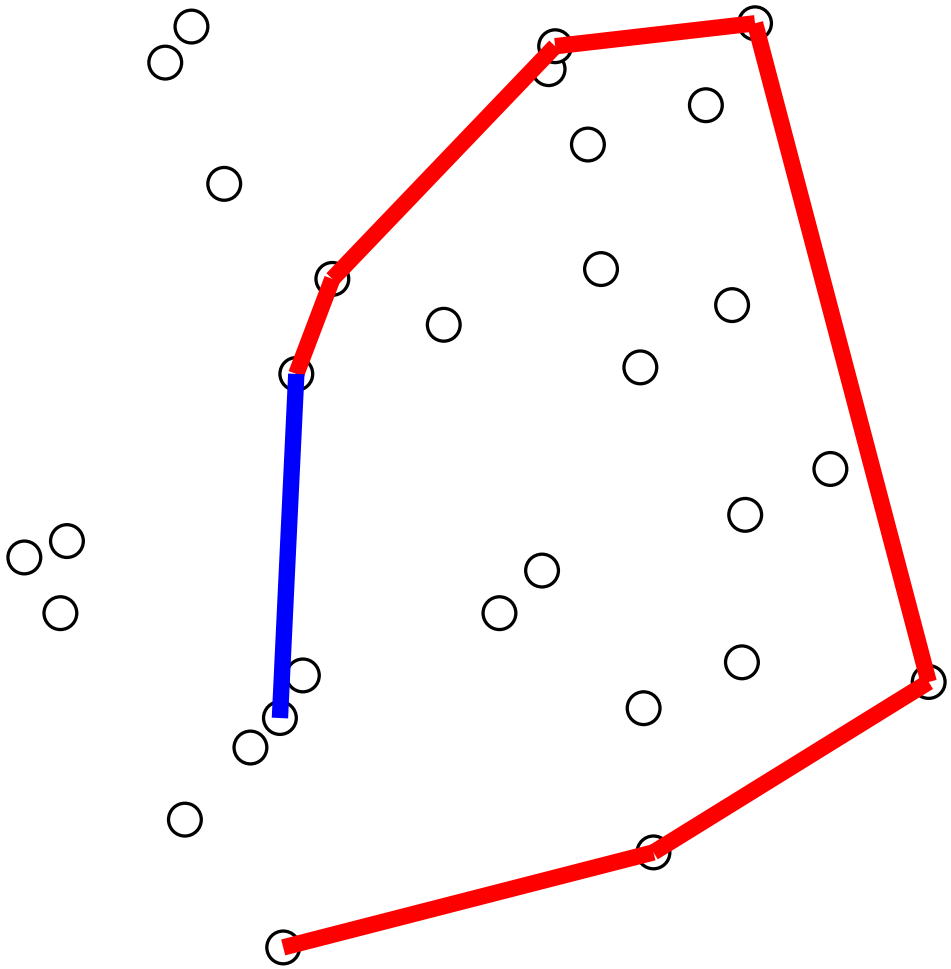


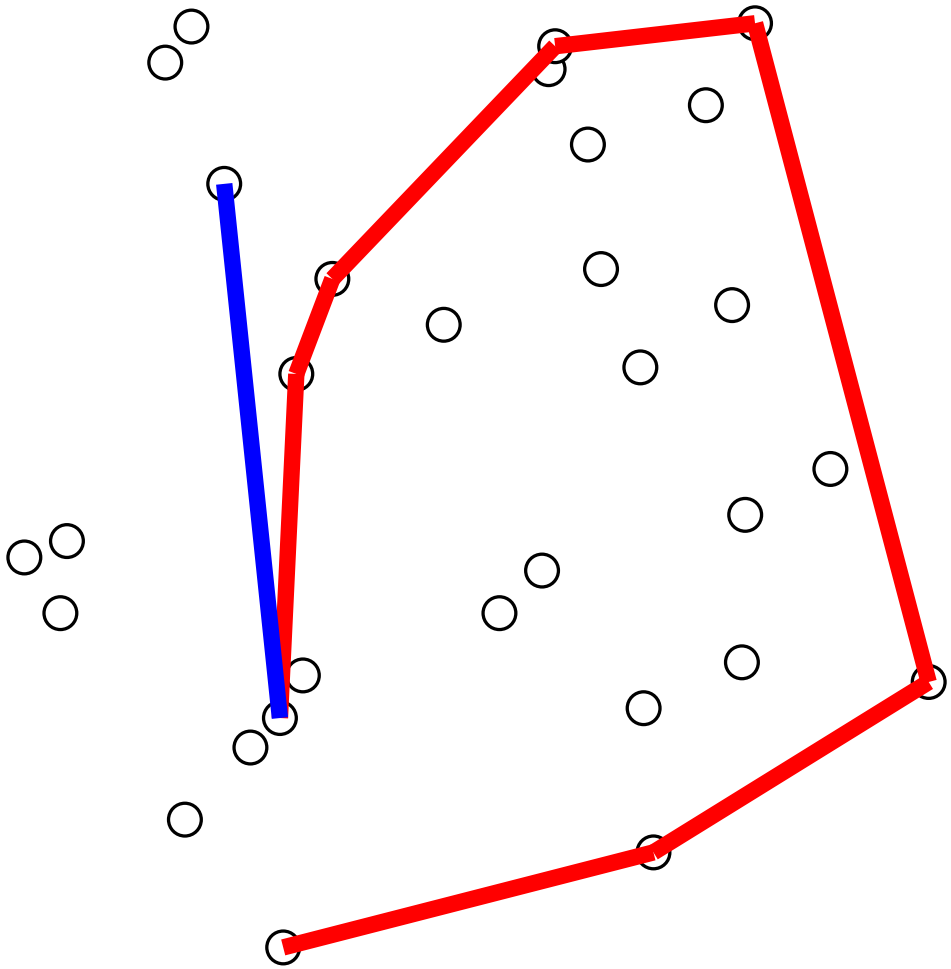


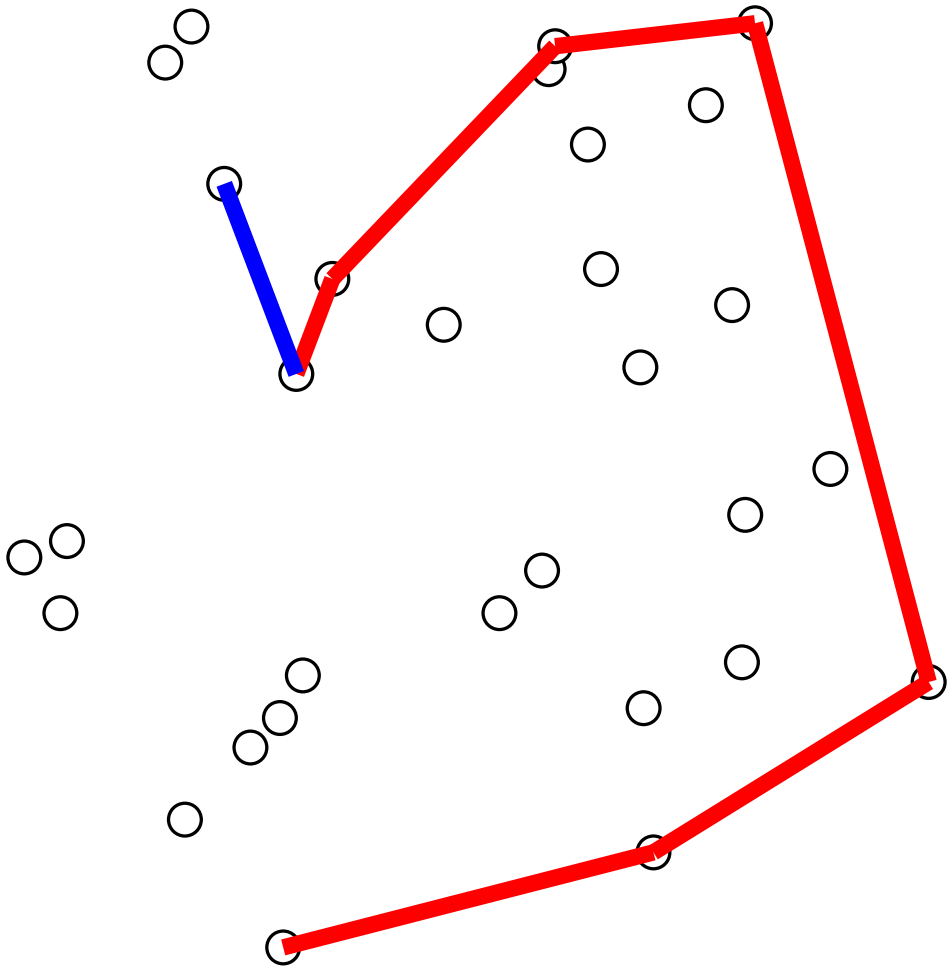


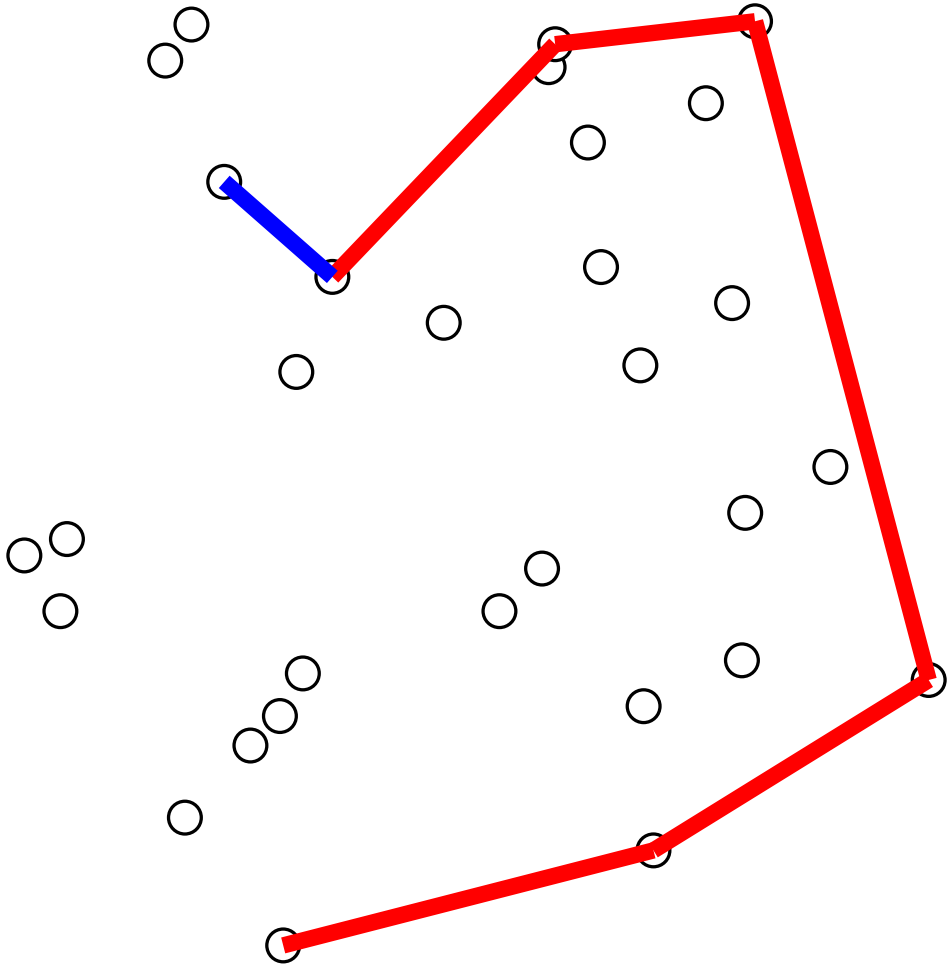


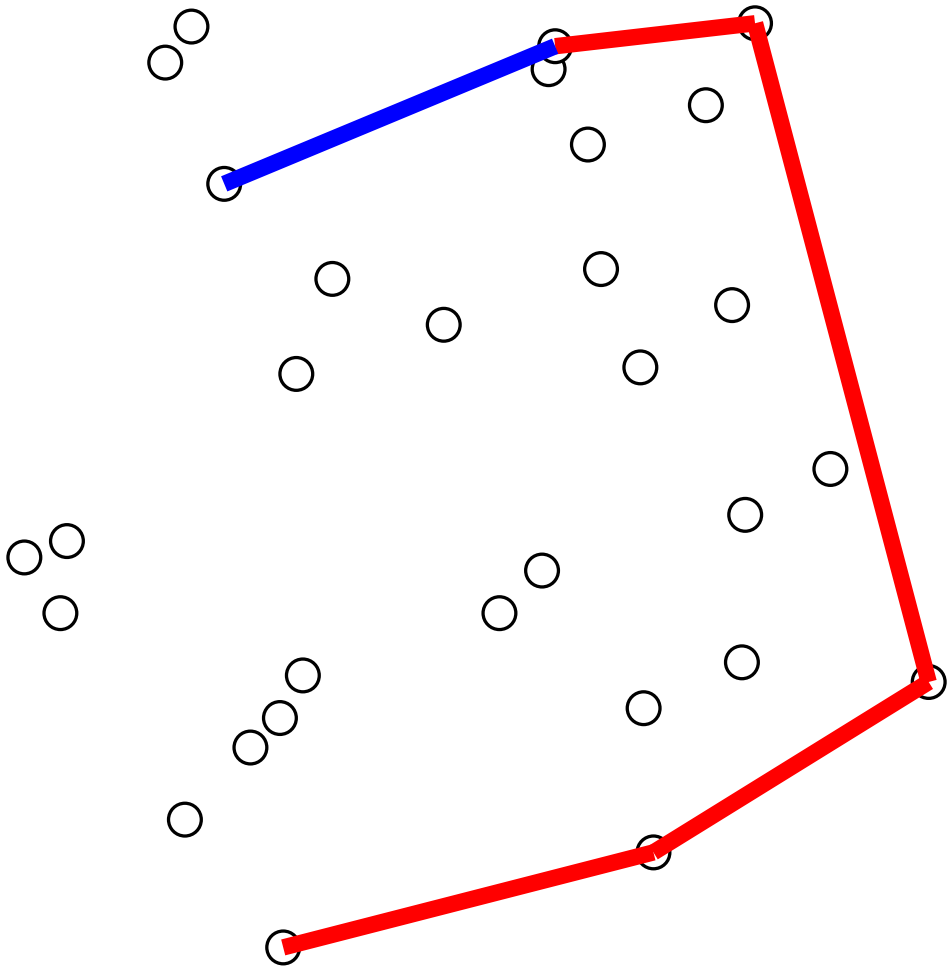


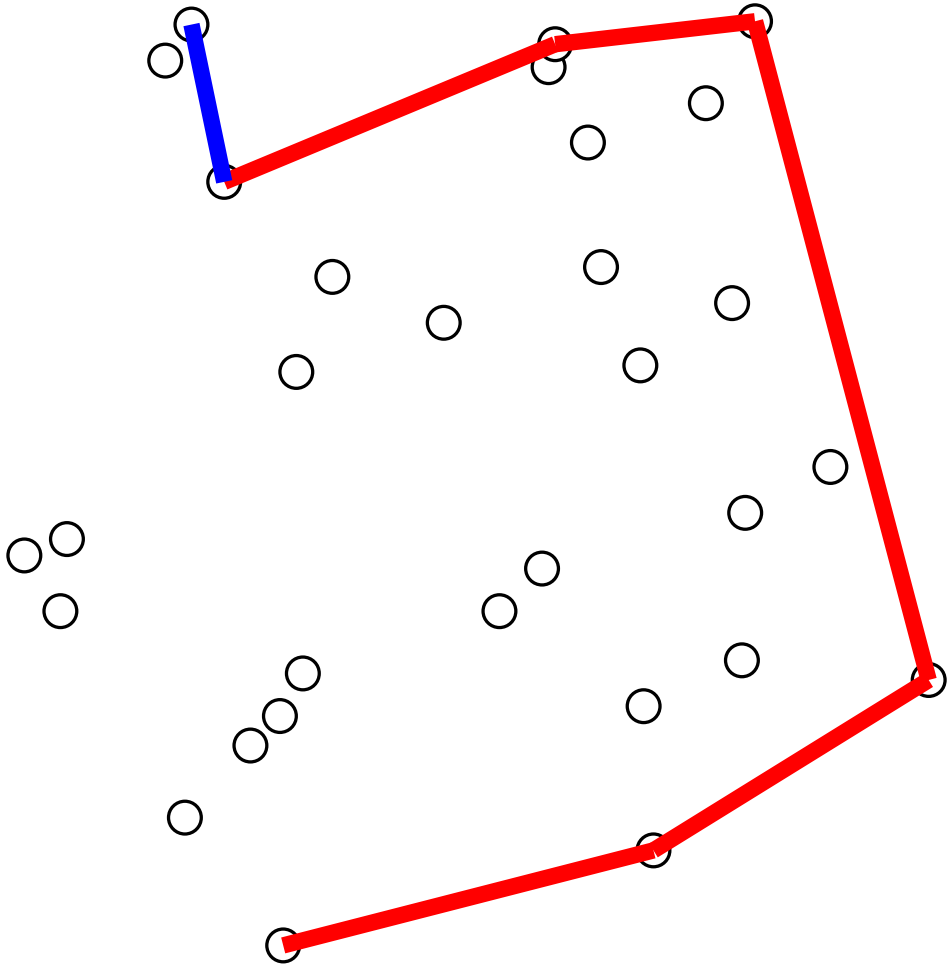


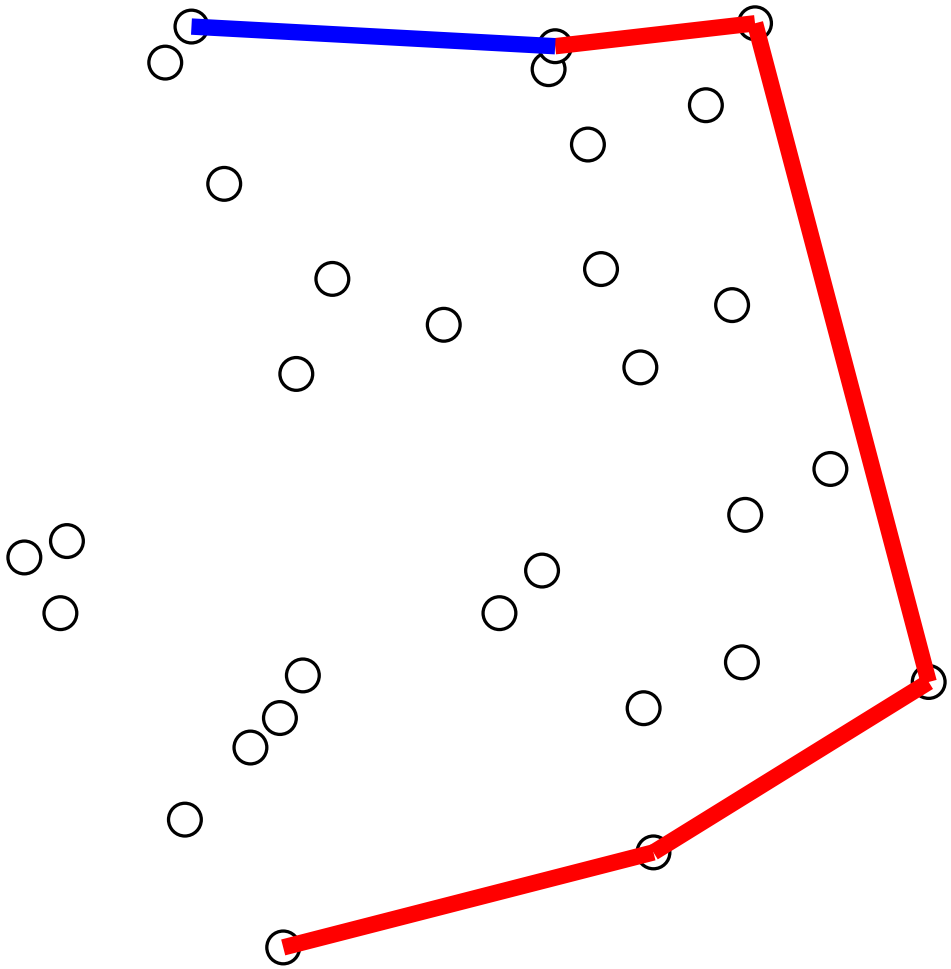


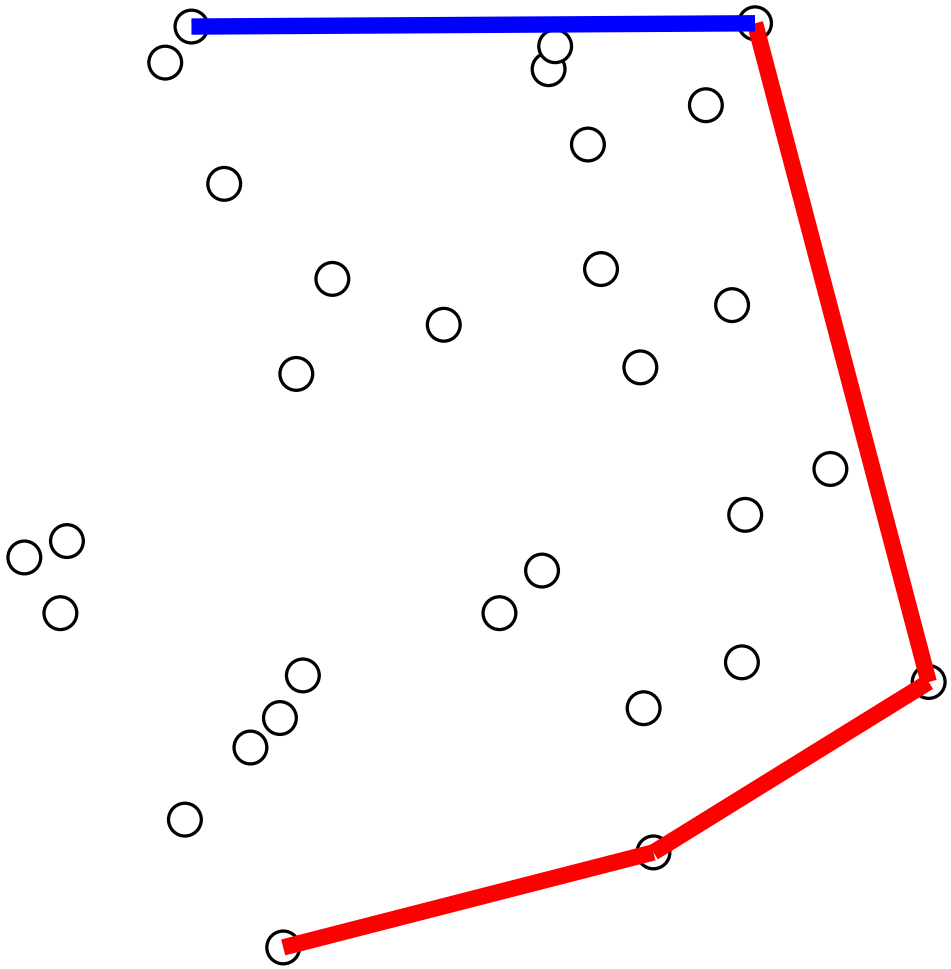


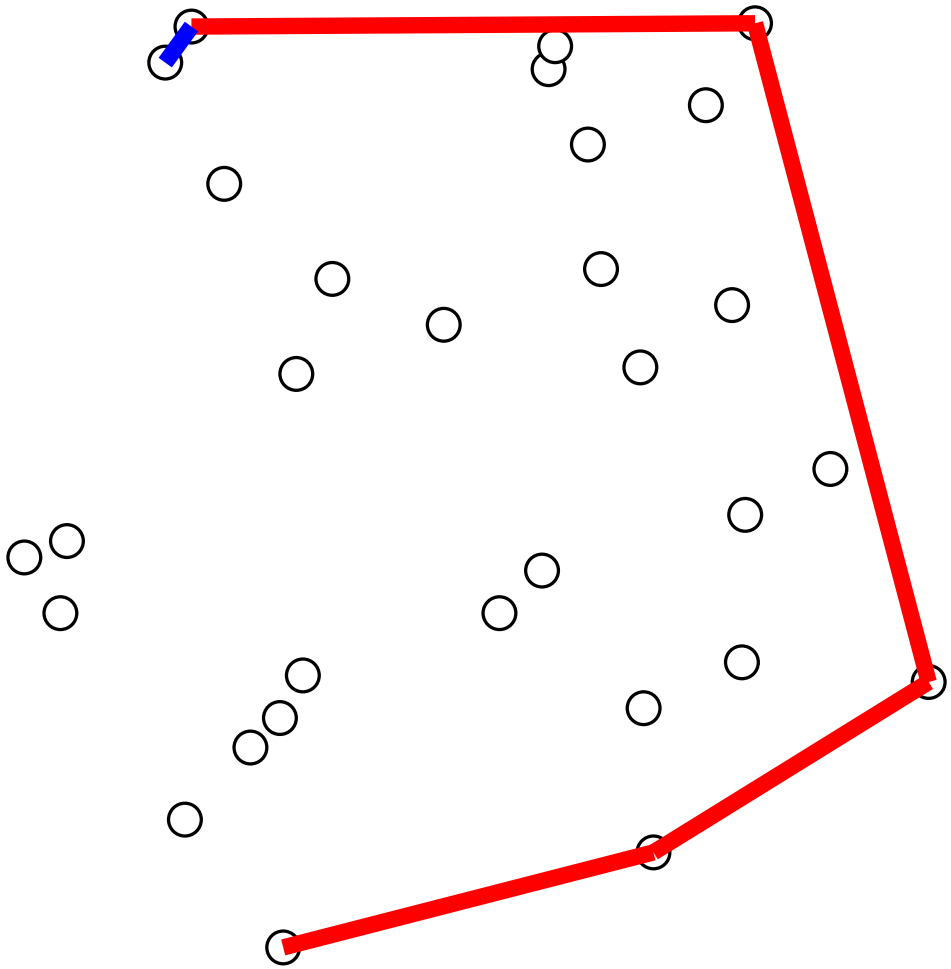


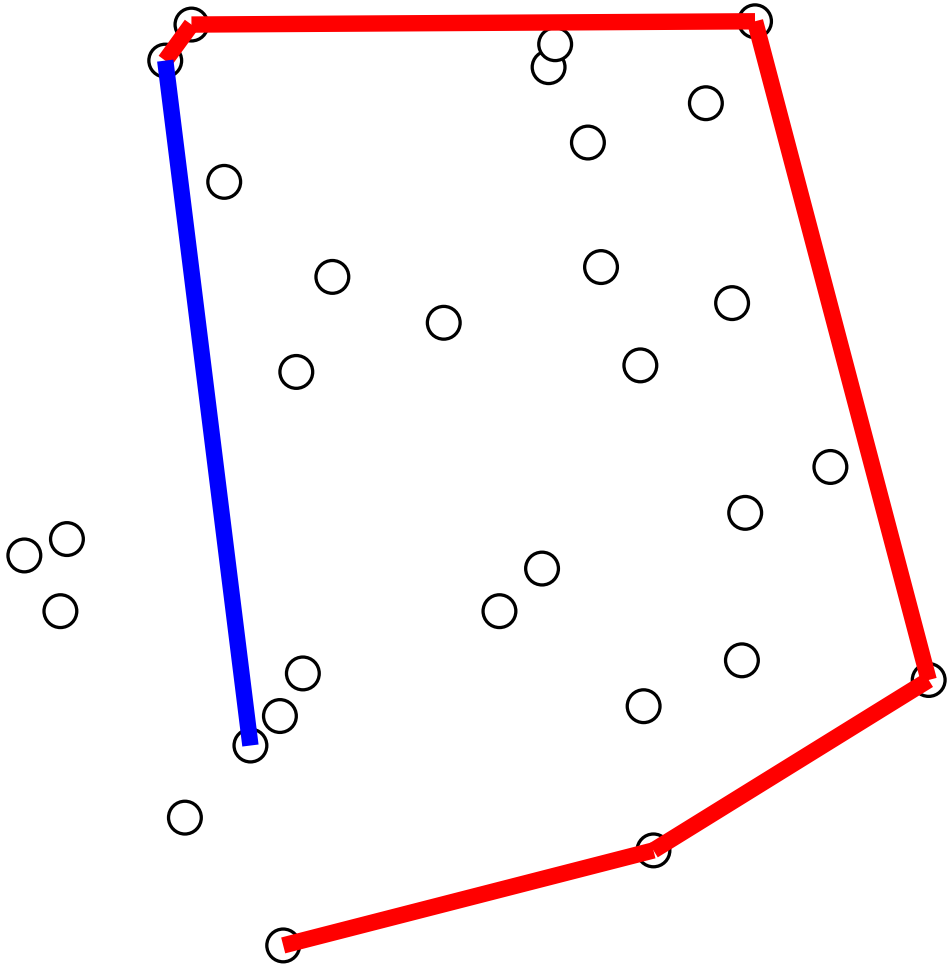


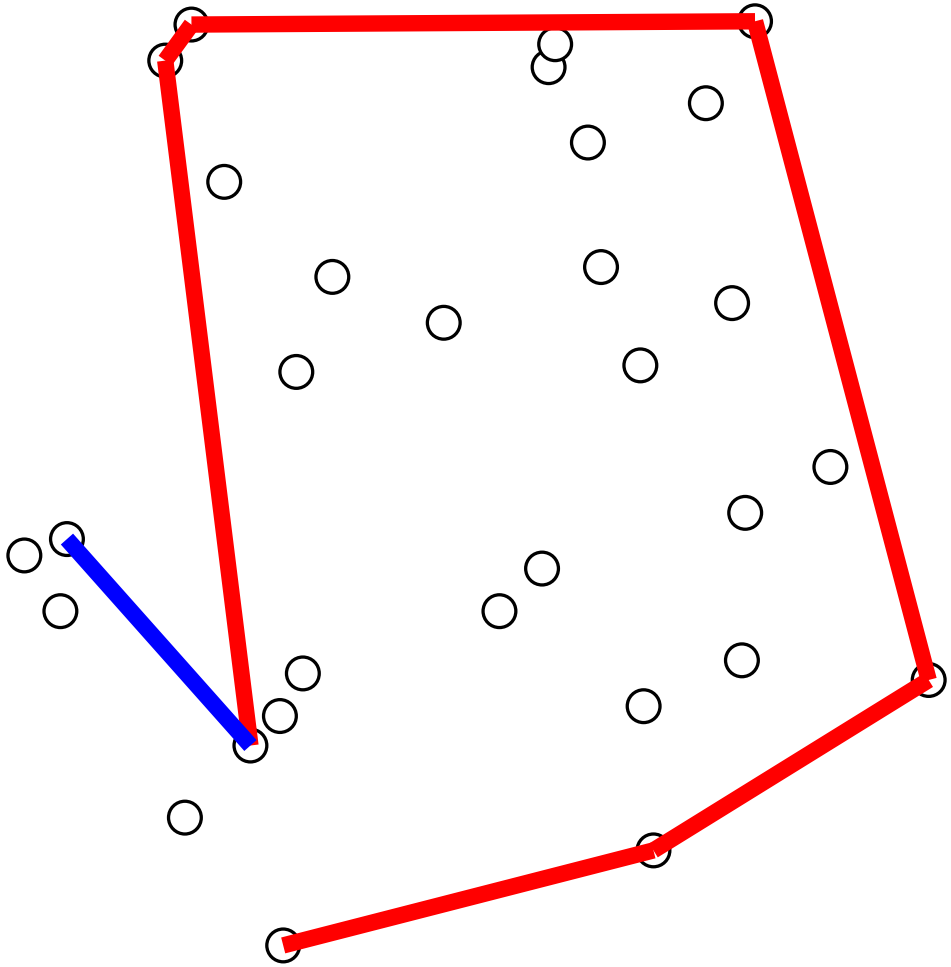


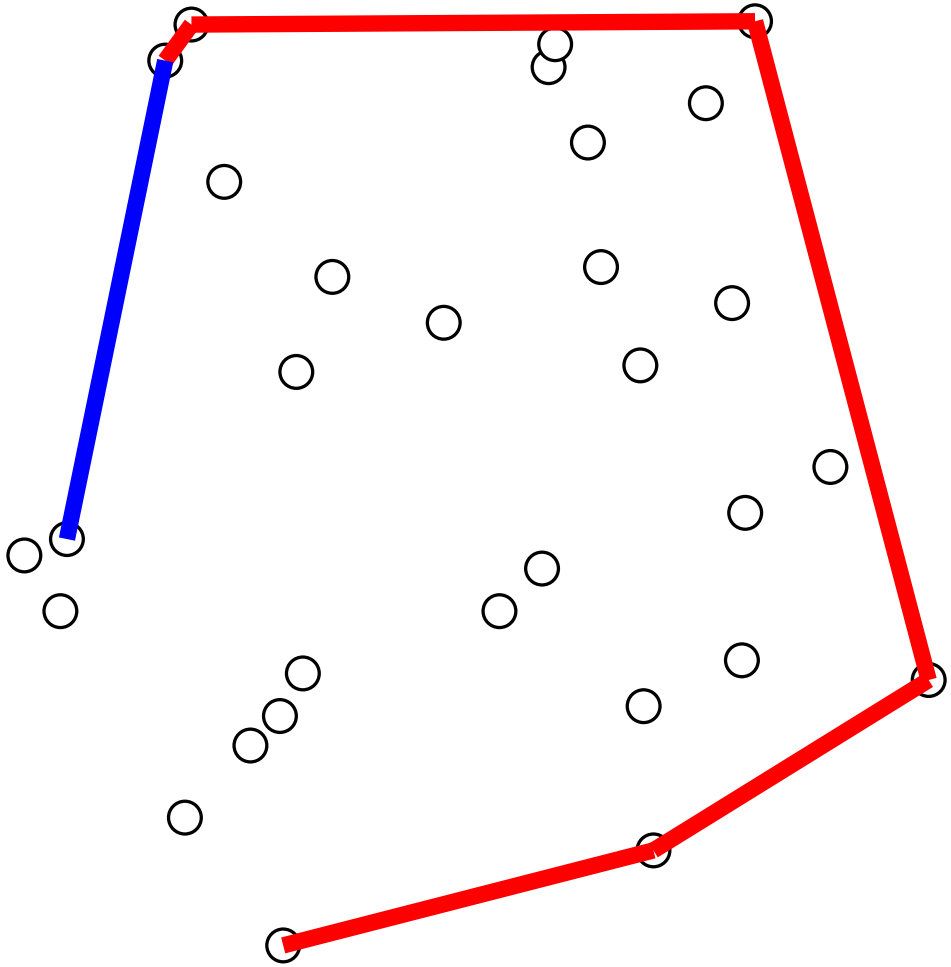


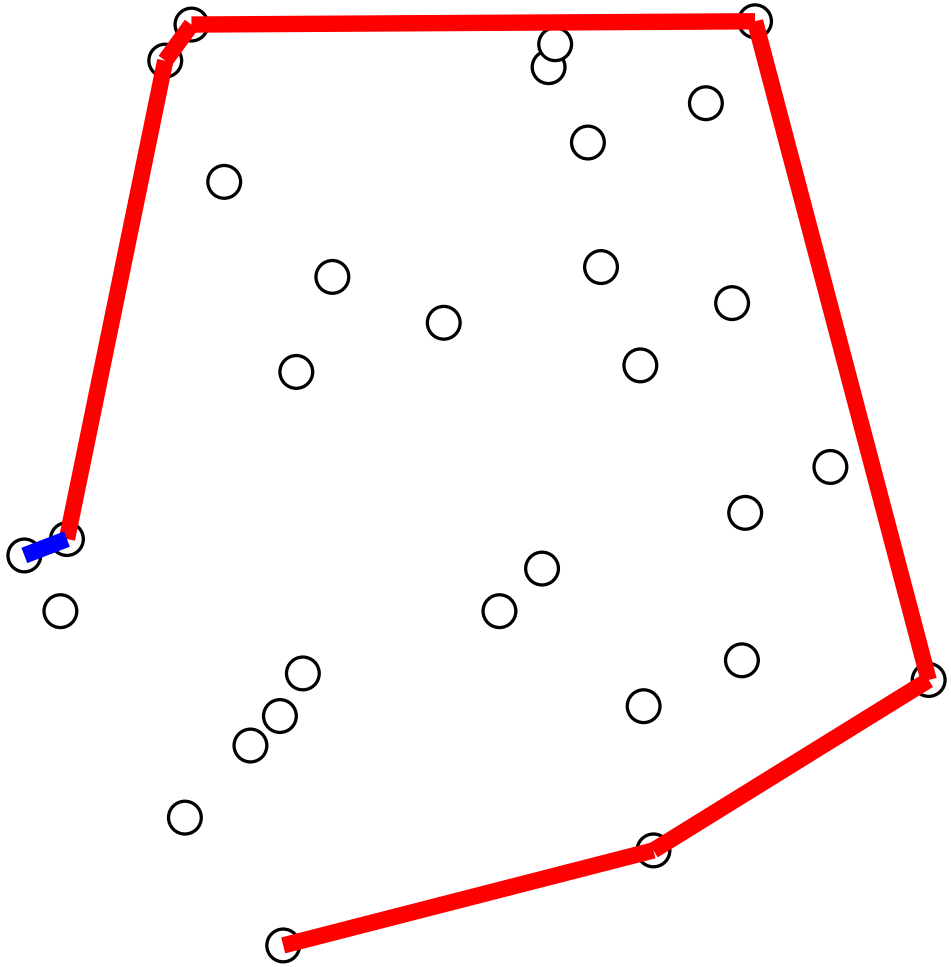


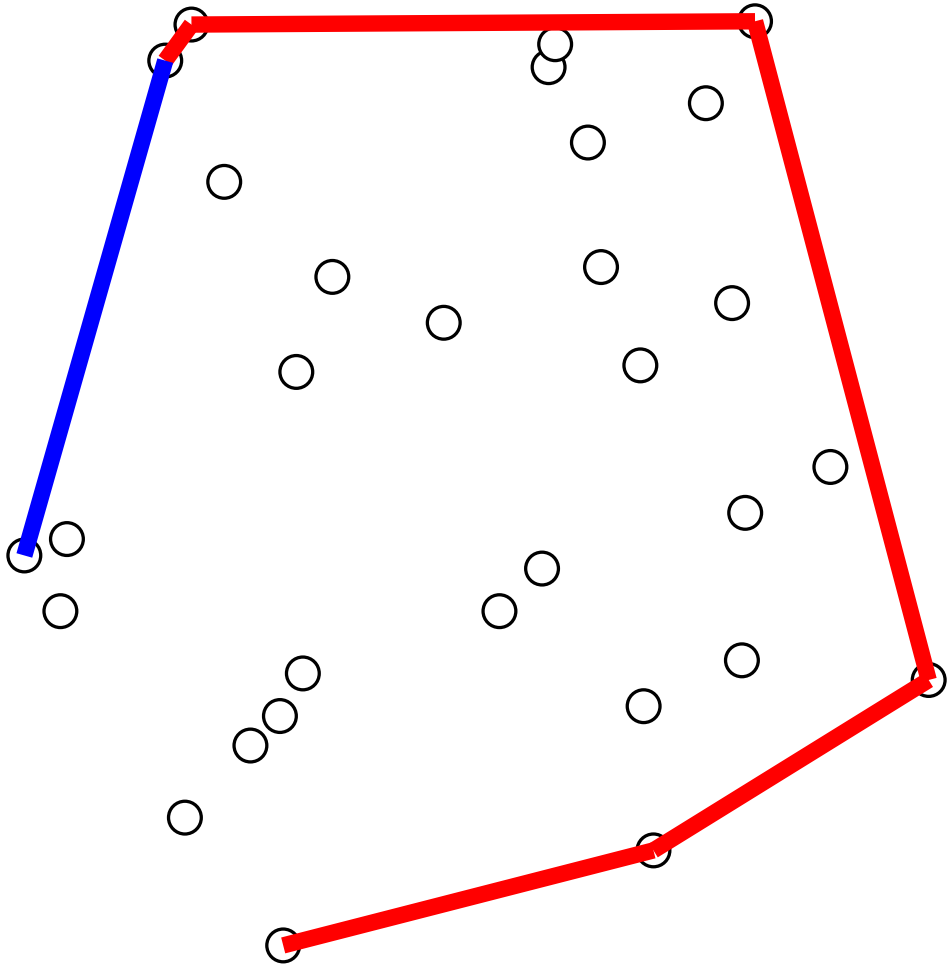


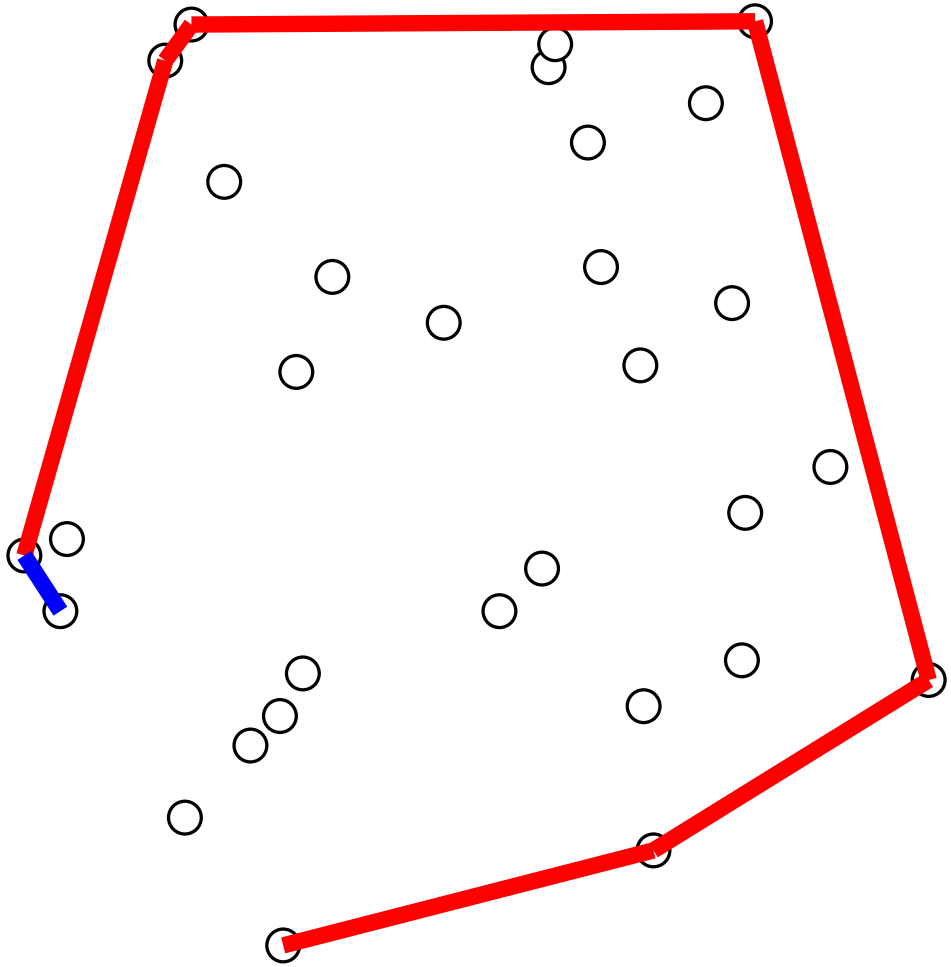


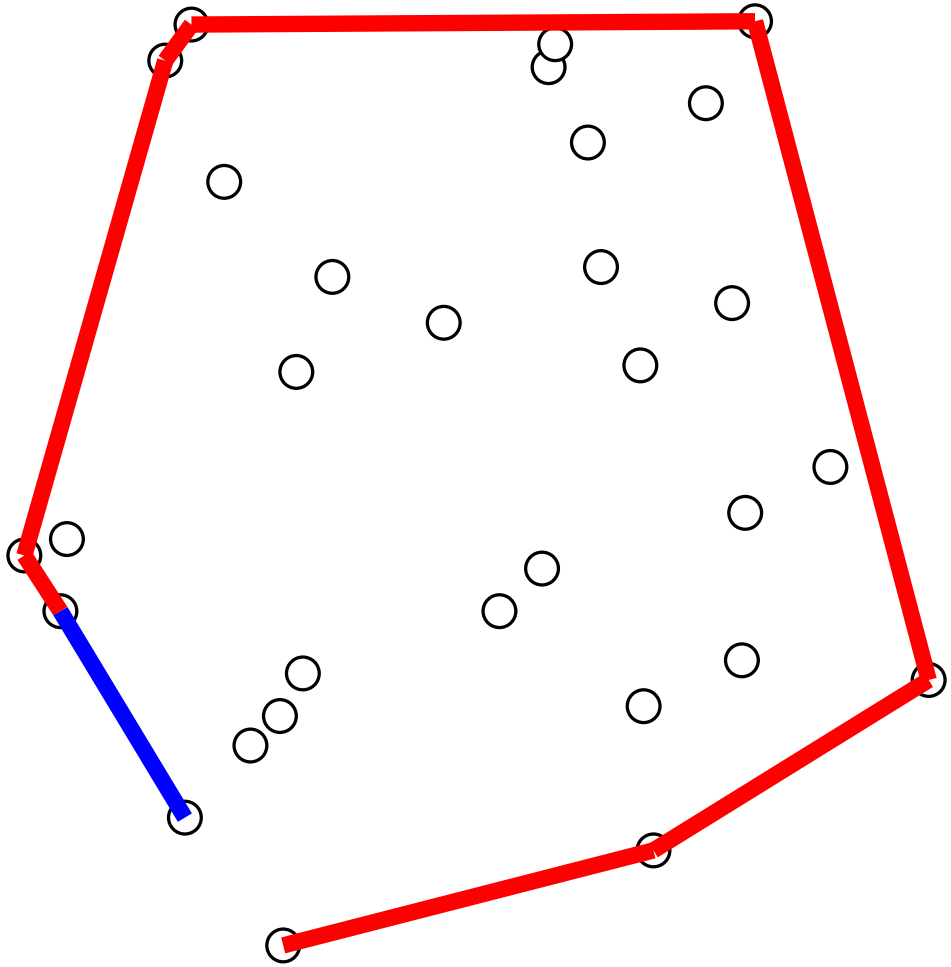


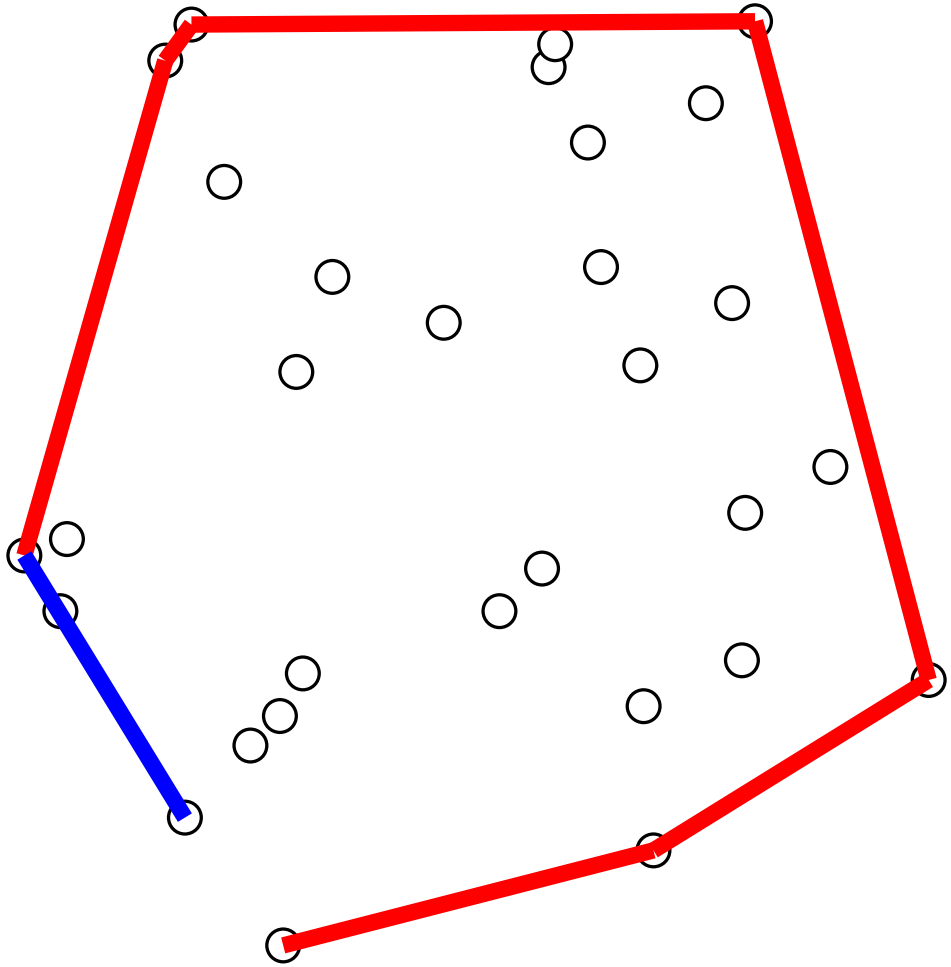


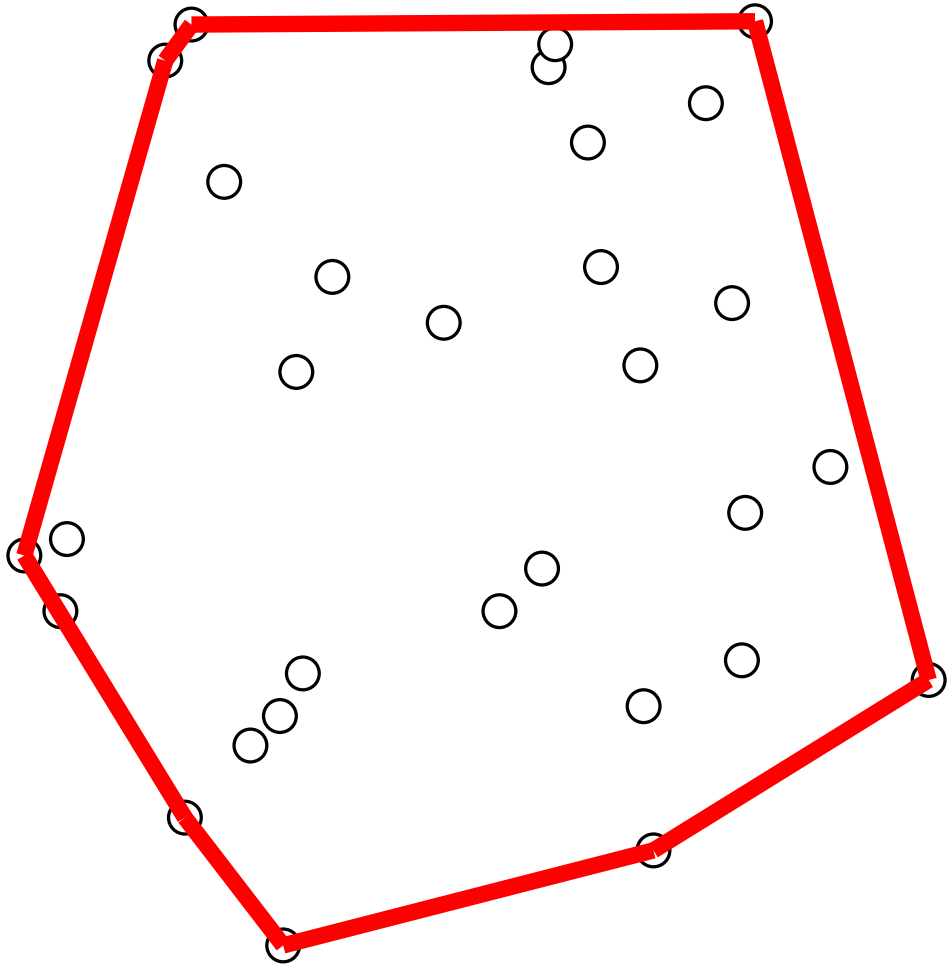






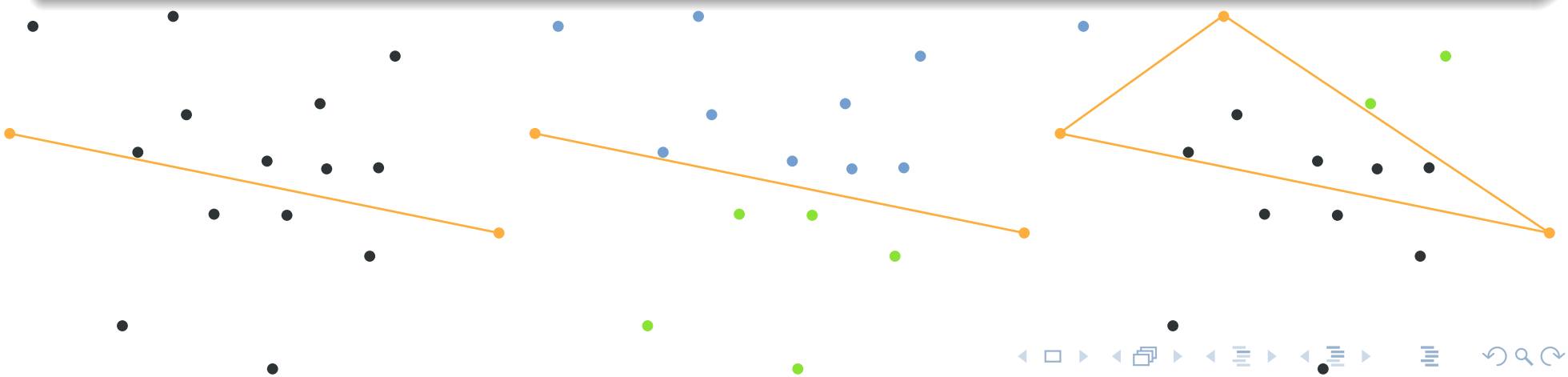






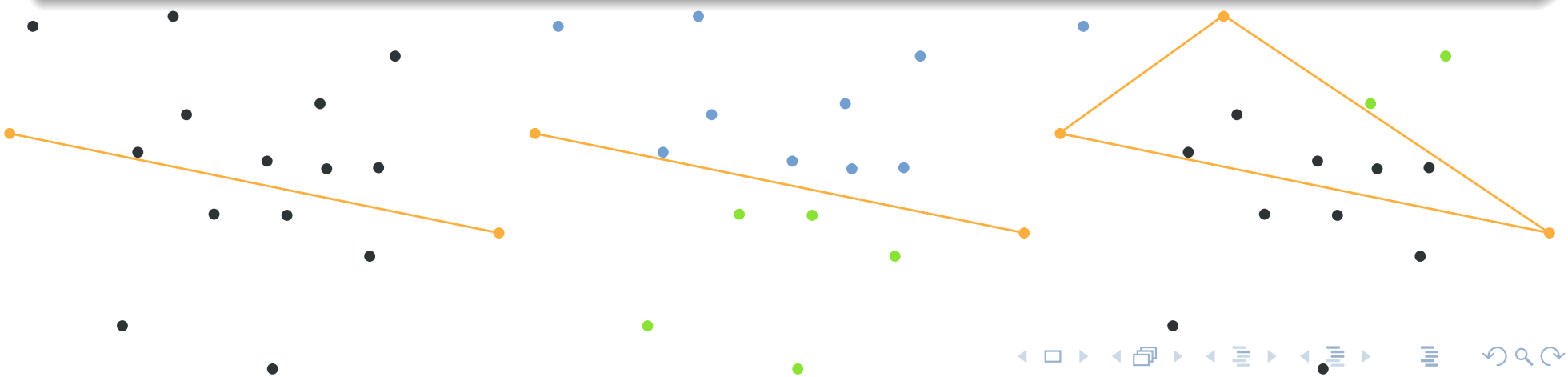
QuickHull 1977: divide and conquer strategy

- Find rightmost a and leftmost points b



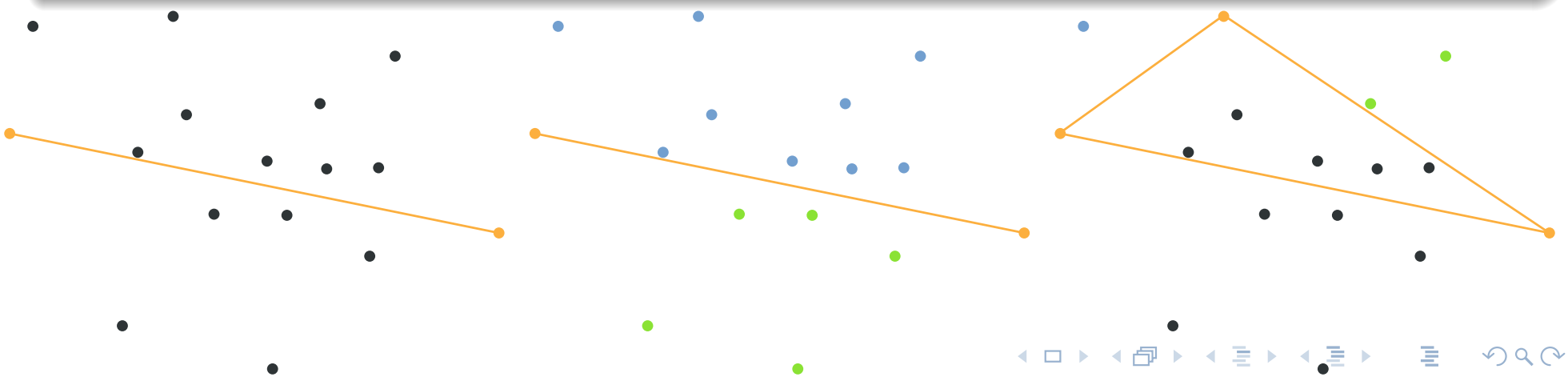
QuickHull 1977: divide and conquer strategy

- Find rightmost a and leftmost points b
- Use the line (ab) to divide the sets in to subsets



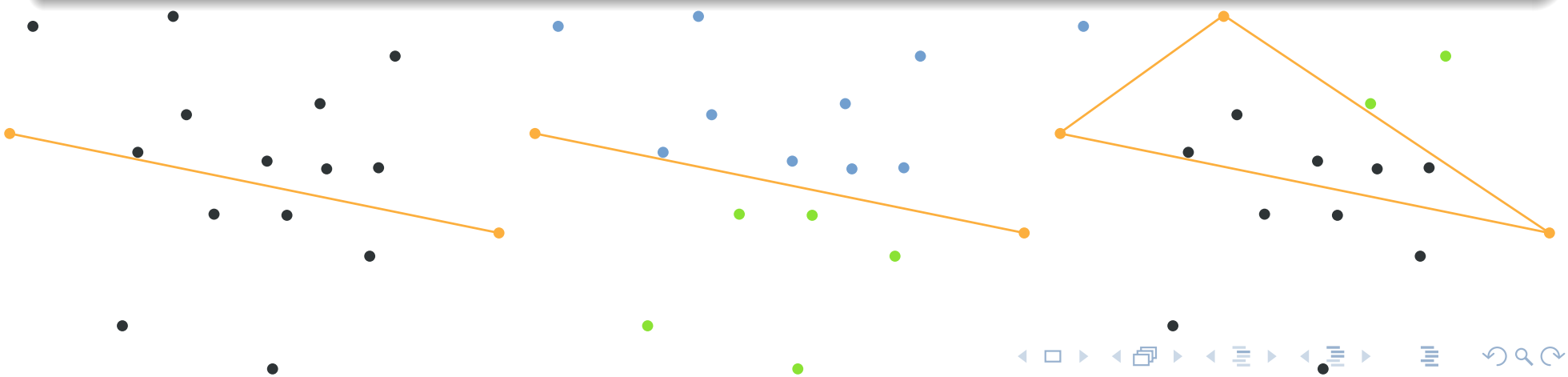
QuickHull 1977: divide and conquer strategy

- Find rightmost a and leftmost points b
- Use the line (ab) to divide the sets in to subsets
- On each subset, determine the point c with the maximum distance from (ab).



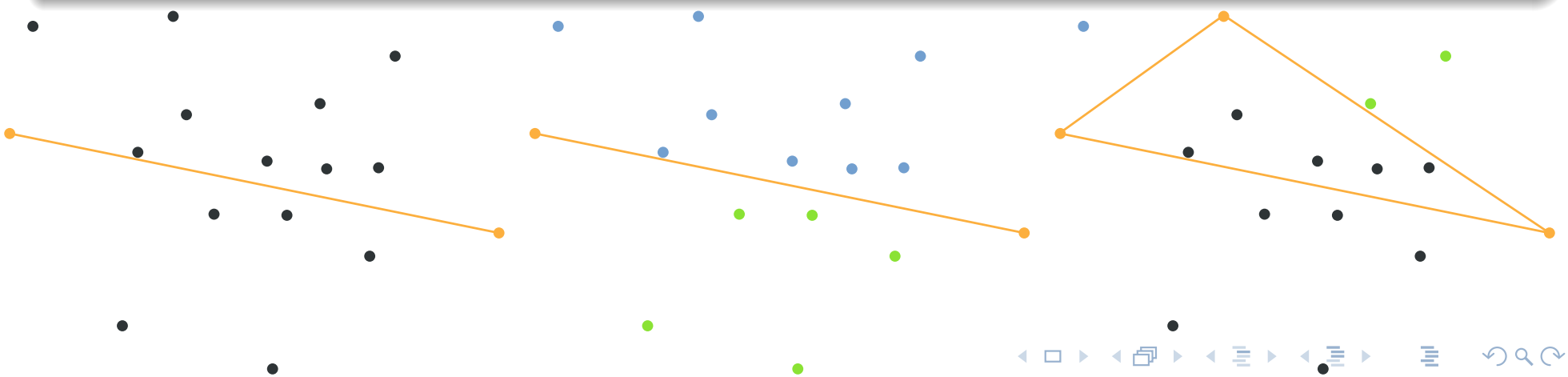
QuickHull 1977: divide and conquer strategy

- Find rightmost a and leftmost points b
- Use the line (ab) to divide the sets in to subsets
- On each subset, determine the point c with the maximum distance from (ab).
- The points lying inside triangle (abc) can be ignored.



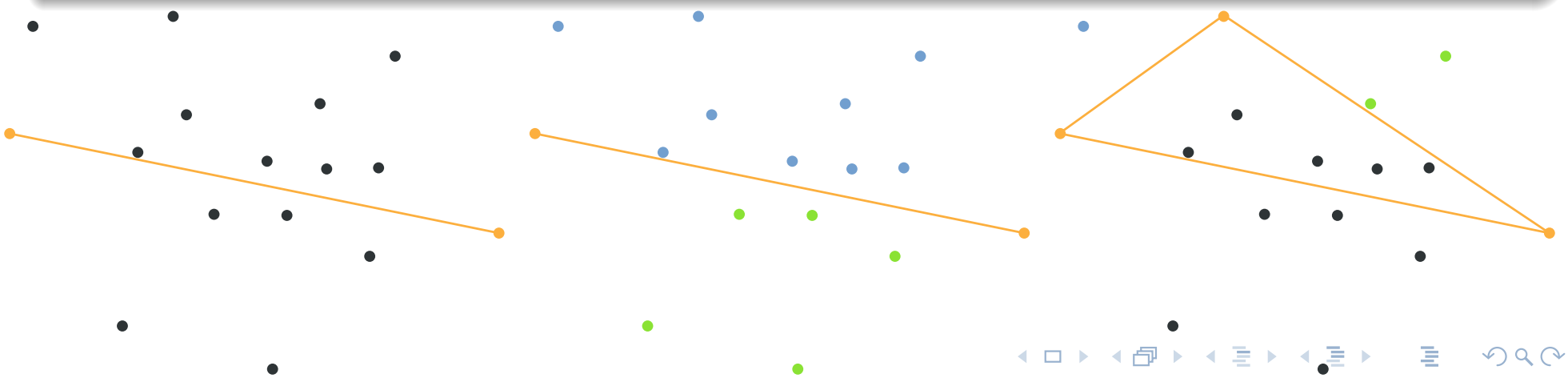
QuickHull 1977: divide and conquer strategy

- Find rightmost a and leftmost points b
- Use the line (ab) to divide the sets in to subsets
- On each subset, determine the point c with the maximum distance from (ab).
- The points lying inside triangle (abc) can be ignored.
- Repeat the previous two steps on the two lines formed by the triangle.



QuickHull 1977: divide and conquer strategy

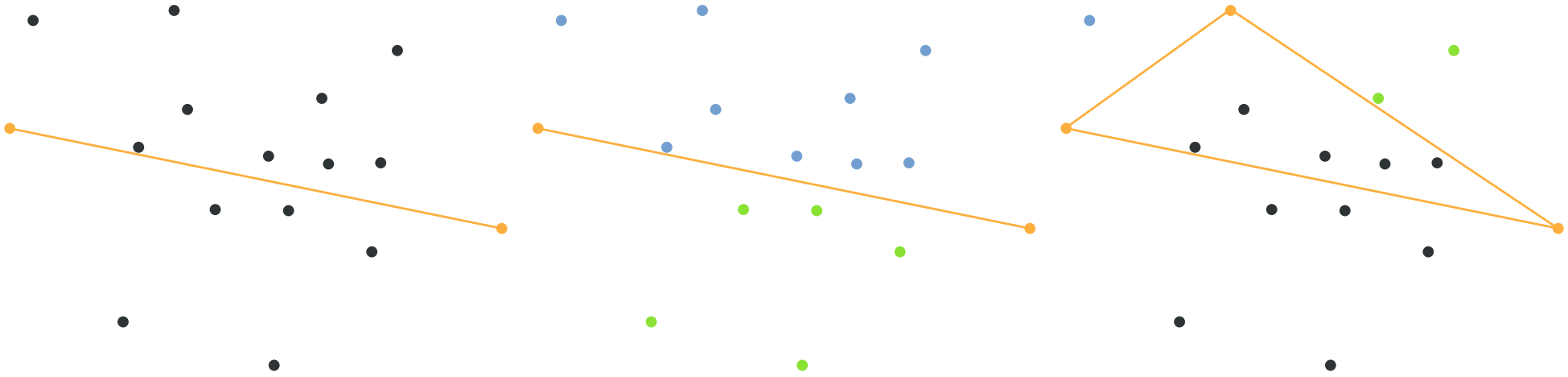
- Find rightmost a and leftmost points b
- Use the line (ab) to divide the sets in to subsets
- On each subset, determine the point c with the maximum distance from (ab).
- The points lying inside triangle (abc) can be ignored.
- Repeat the previous two steps on the two lines formed by the triangle.
- Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.



QuickHull: divide and conquer strategy

Complexity

- n points in S, h points in $conv(S)$
- $\theta(n \cdot \log(n))$ in average
- worst case $O(n^2)$



Section 4

Geometric problems

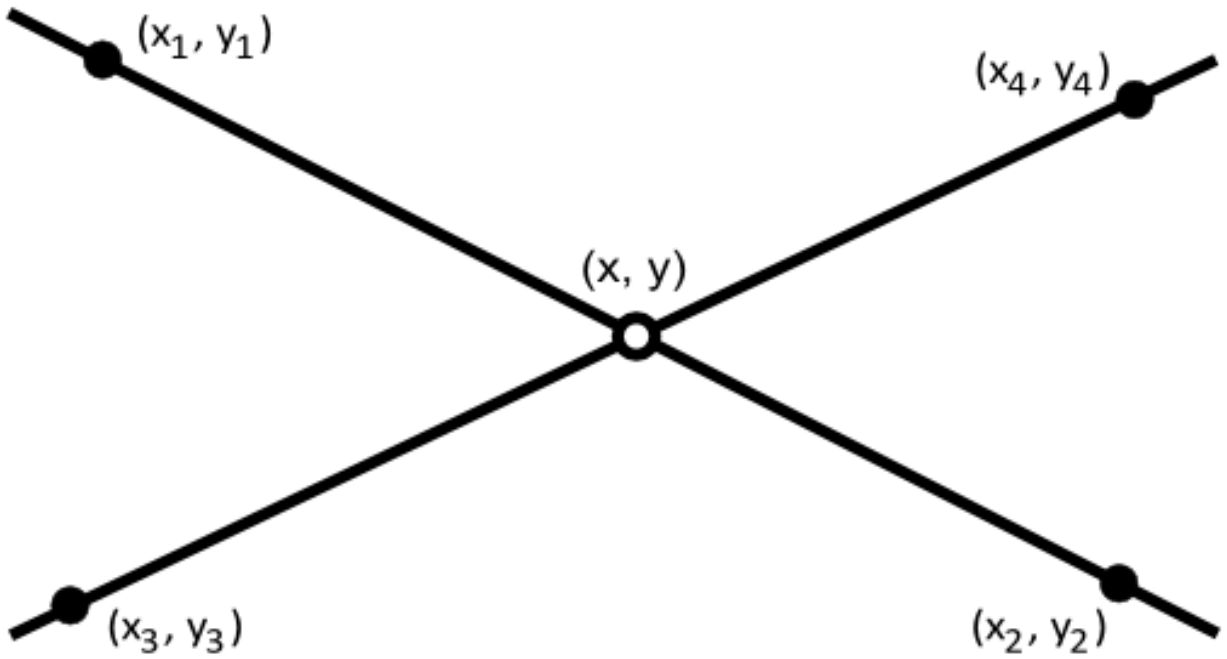
Intersection of 2 2D lines

Performed with determinants

2 lines L_1 and L_2 in 2D space

- (x_1, y_1) and (x_2, y_2) on L_1
- (x_3, y_3) and (x_4, y_4) on L_2

The intersection P of line L_1 and L_2 can be defined using determinants.



Intersection of 2 2D lines

Performed with determinants

2 lines L_1 and L_2 in 2D space

- (x_1, y_1) and (x_2, y_2) on L_1
- (x_3, y_3) and (x_4, y_4) on L_2

The intersection P of line L_1 and L_2 can be defined using determinants.

$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

Segments

Modifiy slightly this formula.

Line-Segment intersection in 2D I

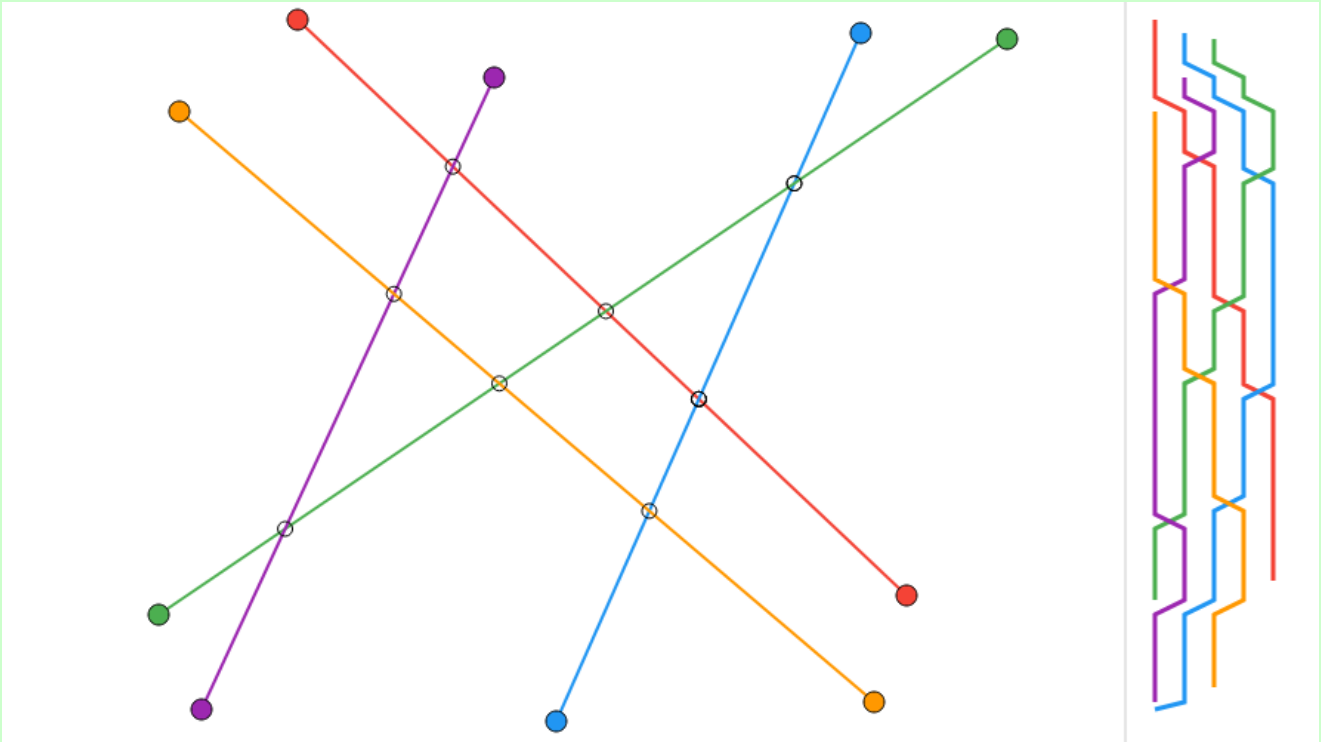
Problem: large number of segments

- Simple algorithms examine each pair of segments: $O(n^2)$ (n number of segments)

Solution

- Shamos and Hoey (1976), sweep line algorithm
- $O(n \log(n))$

Line-Segment intersection in 2D II



- Basis algorithm for polygon intersection, Voronoi diagram
- Beware of degenerate cases: colinear segments, endpoints inside segments, and shared endpoints, vertical/horizontal segments.

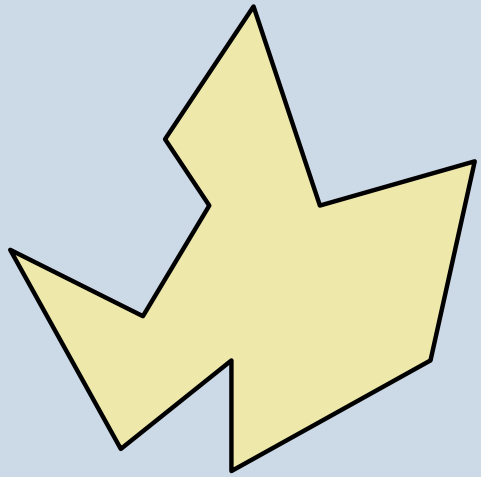
Section 5

Polygons

Polygons in 2D

Definition: polygon

- Closed region
- bounded by line segments
- no intersection



Definition: triangulation

A decomposition of P into triangles by a maximal set of noncrossing diagonals.

- Maximal: no additional diagonal can be added without crossing one.

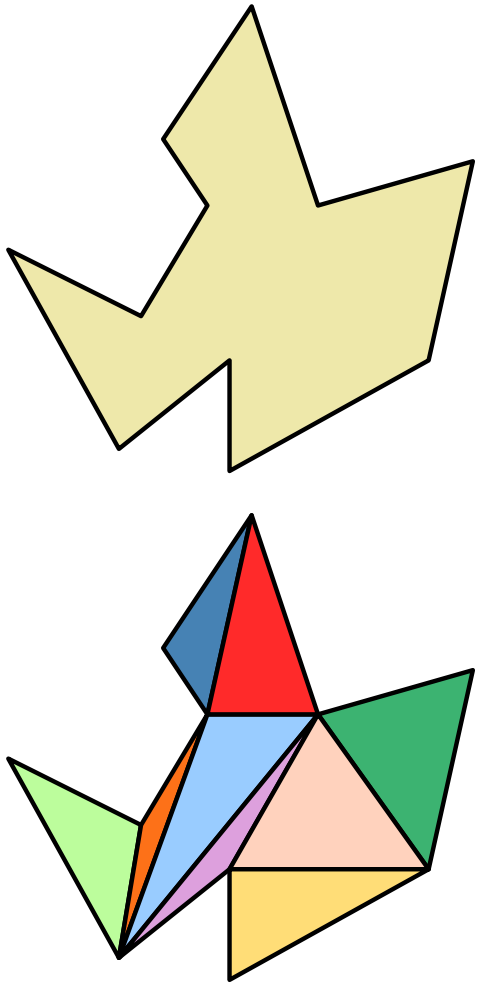
Theorem

Every polygon P has a triangulation.

Theorem

Every triangulation of P with n vertices has:

- $n - 2$ triangles
- $n - 3$ diagonals



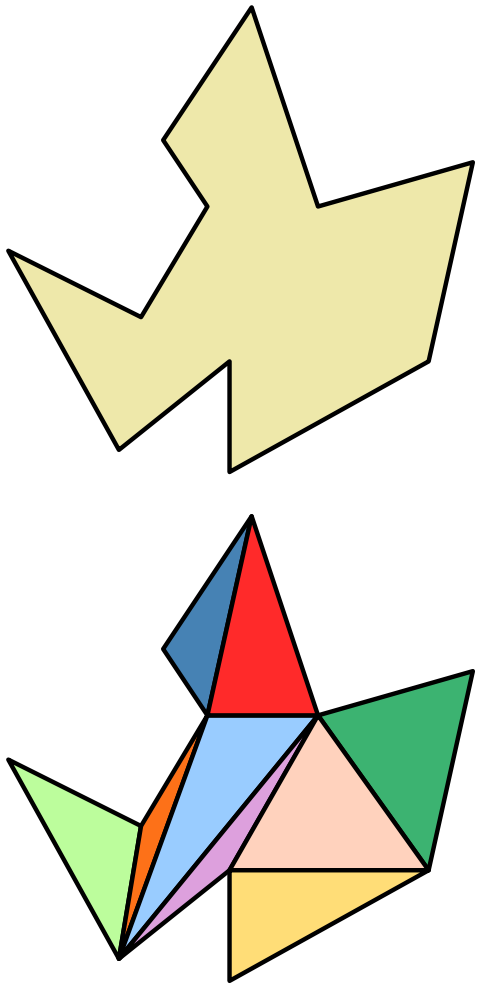
Theorem

The number of triangulations of a convex polygon with $n+2$ vertices is the Catalan number:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Theorem

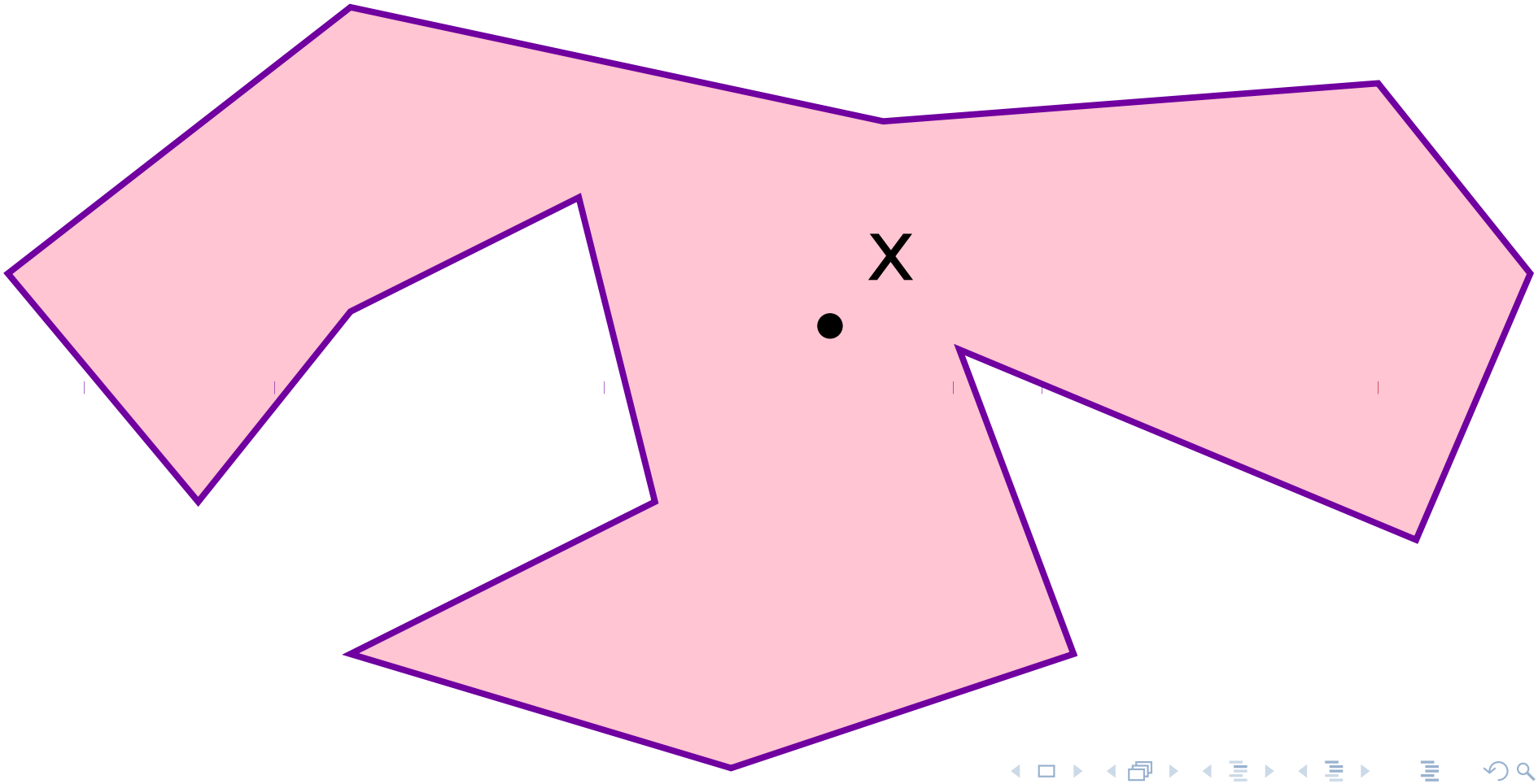
The number of triangulations of a polygon with $n+2$ vertices is between 1 and C_n.



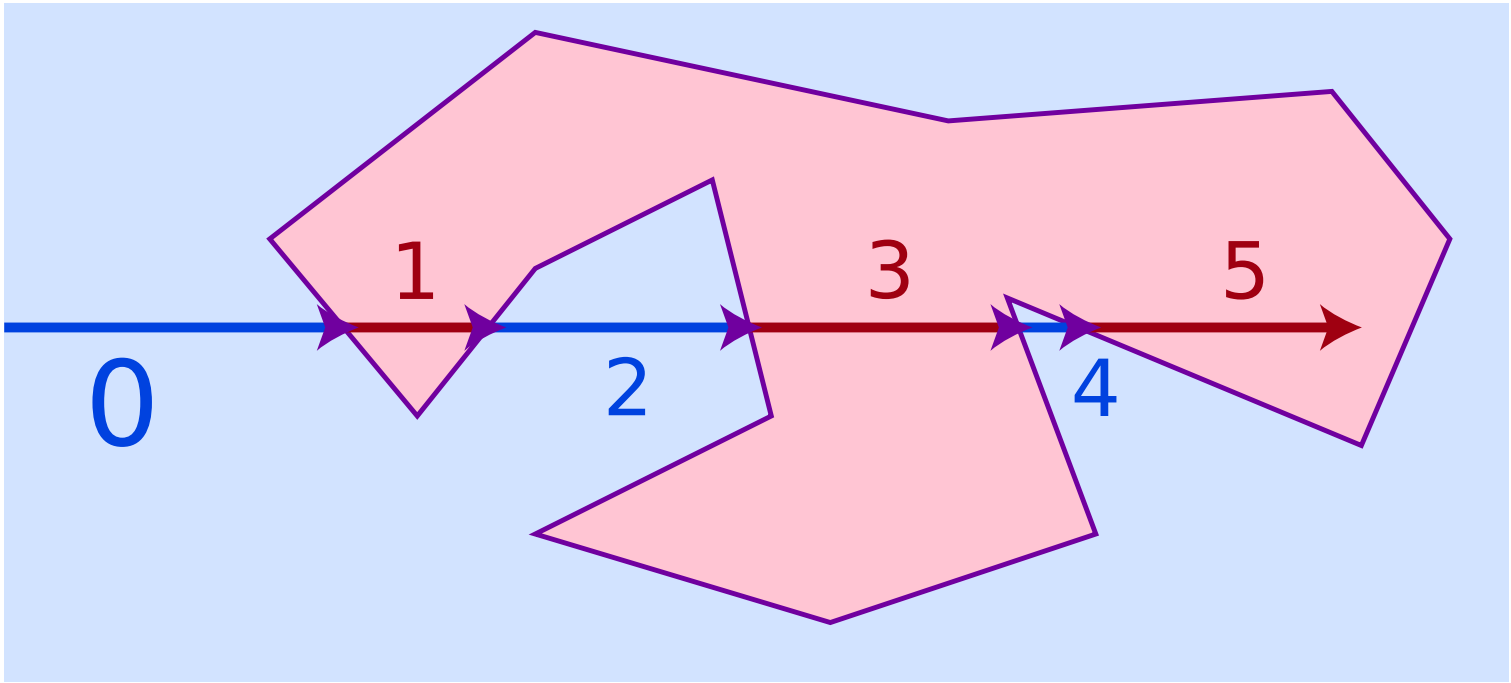
Point in polygon

Problem

Does a point lay inside a polygon P?

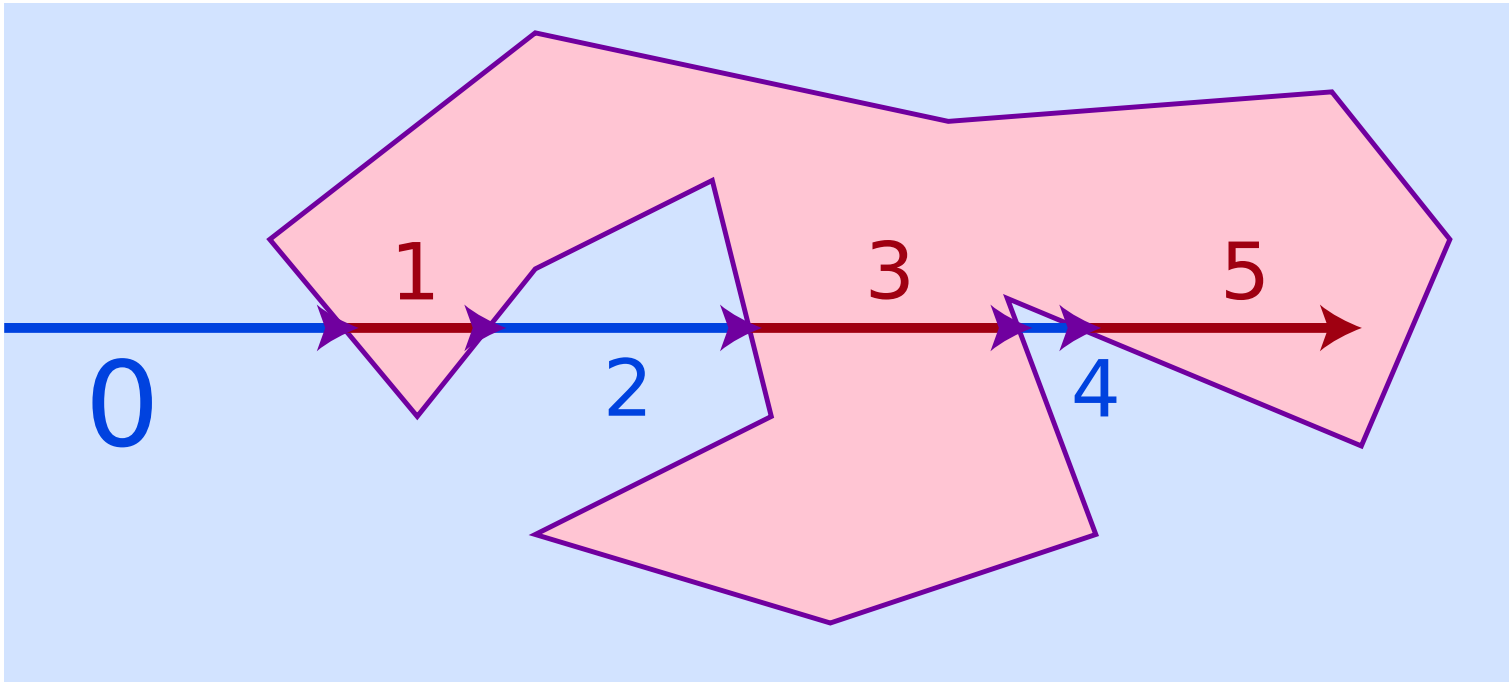


Point in polygon: ray casting algorithm (aka even-odd rule)



- A ray goes from infinity to the probe point.
- It crosses the boundary several times.
- Counts this number of times.

Point in polygon: ray casting algorithm (aka even-odd rule)



Numerical problems

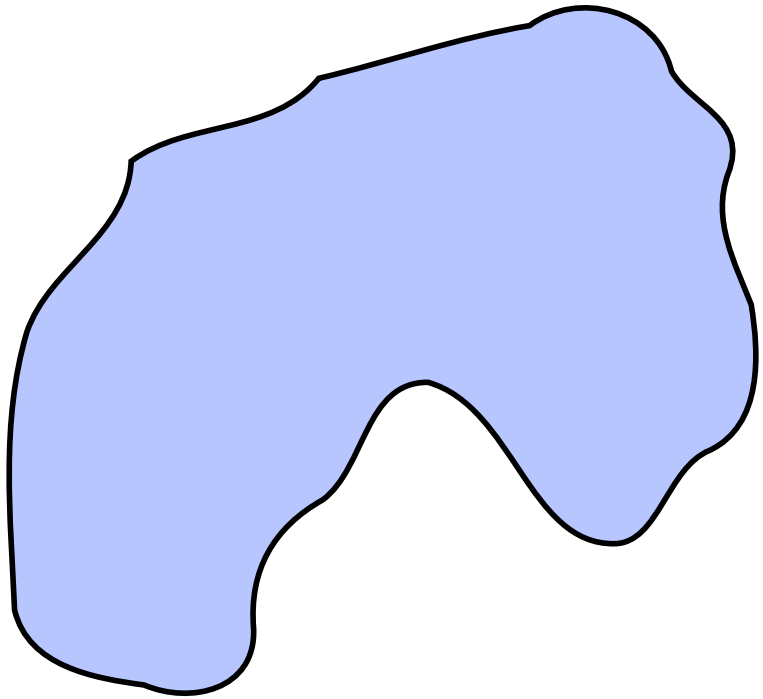
When ray goes through

- a vertex
- en entire segment

Element of proof: Jordan curve theorem

Theorem

A Jordan curve (simple closed curve) in $\mathbb{R}^2$ partitions the plane into two parts: the interior (bounded) and the exterior (unbounded).

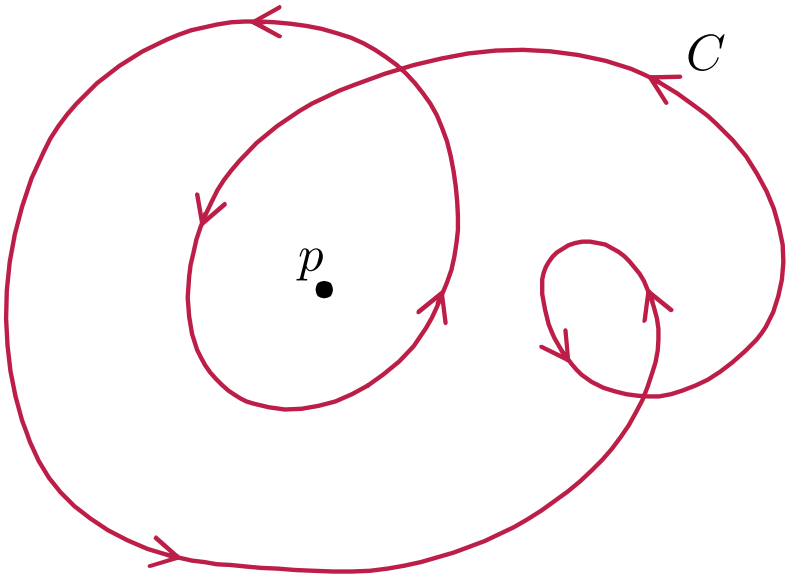


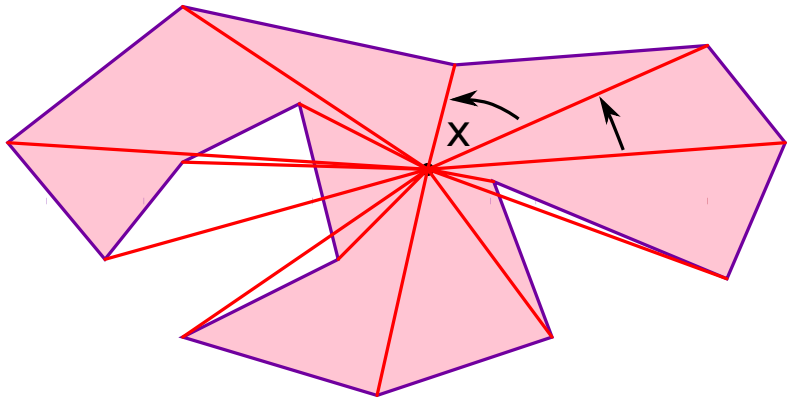
Point in polygon: winding number

Definition

The number of times a closed curve travels counterclockwise around a point.

Application to polygons: sum up the angles sustended by each side of the polygon (use dot product...).





Ray casting or winding number?

- both work for simple polygons (non intersecting lines)
- for complex polygons (overlapping parts), **fast signed crossing WN**.
- for triangles, use dot product.

Warning

- As always: floating operations precision
- degenerate cases

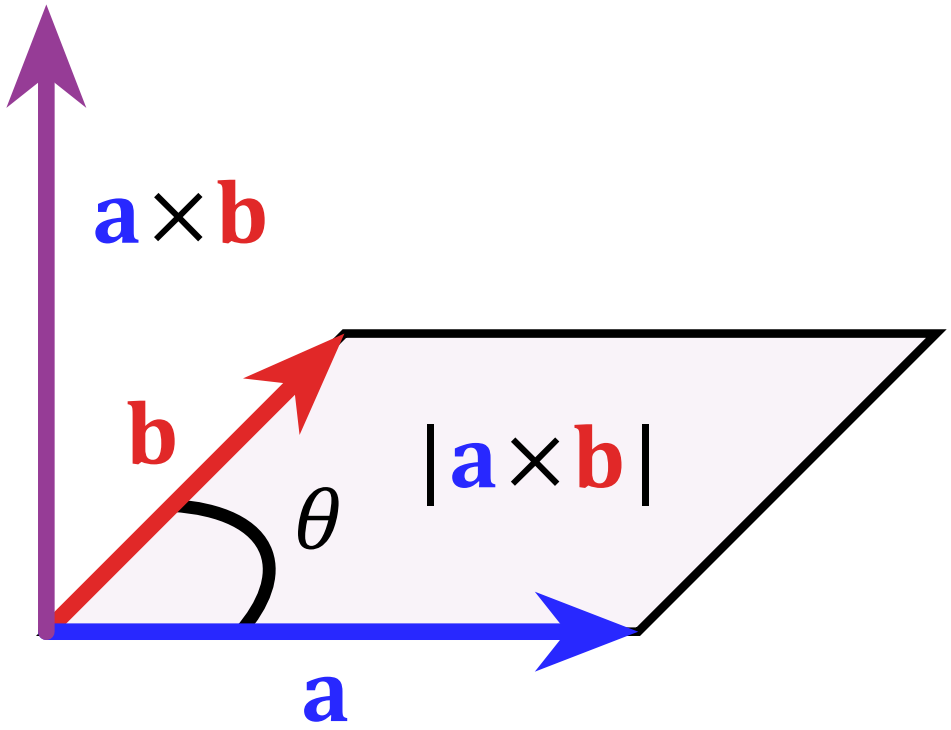
2D planes in 3D

Perform query on a projection.

Area of a triangle

Vector product

$$\mathcal{A} = \frac{1}{2}|a \times b|$$



Area of a simple polygon

- A simple polygon P defined by its vertices $V_i = (x_i, y_i), i \in [O; n]$ ($V_n = V_0$).
- Choose a point x, define triangle $\Delta_i = xV_iV_{i+1}$.
- $\mathcal{A}$ is the signed area (oriented).
- $\mathcal{A}(P) = \sum_0^{n-1} \mathcal{A}(\Delta_i)$

Sum the (oriented) areas of all triangles

$$\mathcal{A} = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

Also known with lengths of sides and exterior angles.

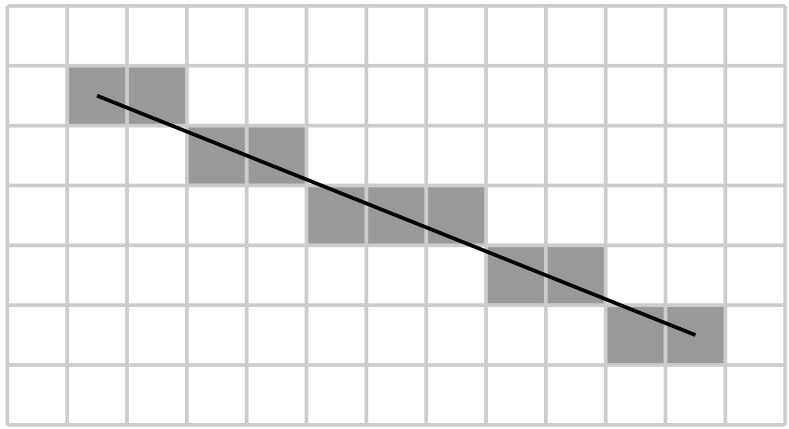
Center of mass of a polygon

$$C_x = \frac{1}{6\mathcal{A}} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$$

$$C_y = \frac{1}{6\mathcal{A}} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$$

Line drawing: Bresenham algorithm, 1962, IBM

- Determination of pixels in a line.
- Not restricted to 2D.
- Discrete approximation, available in integer arithmetic.



Bresenham line: algorithm

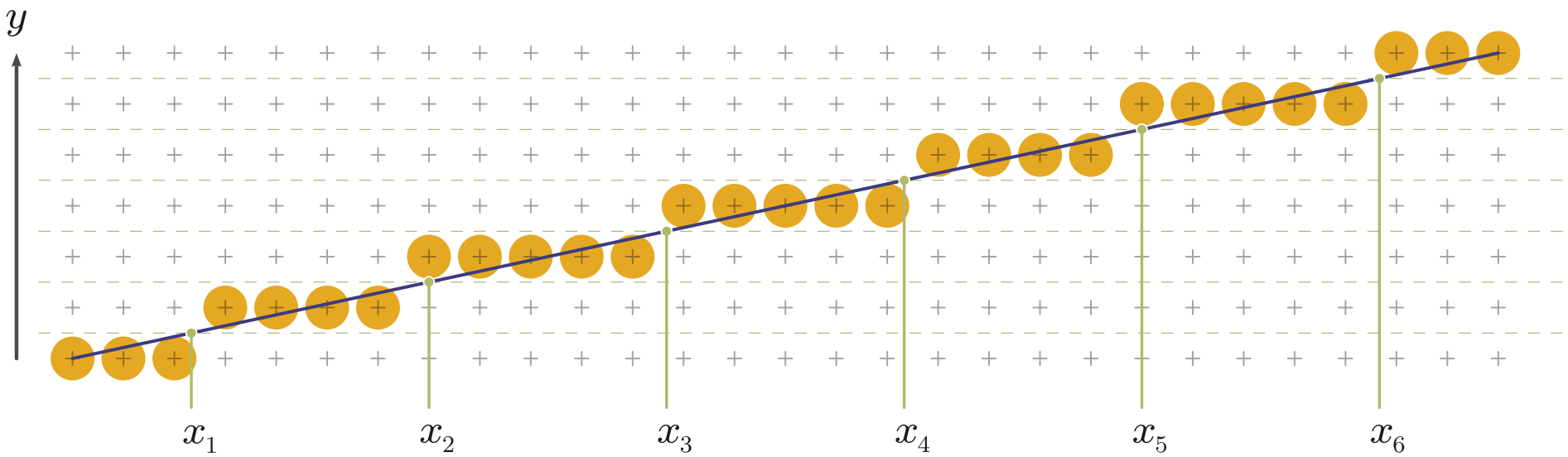
Error propagation algorithm

Discrete version ($\dot{x}$ and $\dot{y}$):

$$\dot{y} = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \cdot (\dot{x} - x_1) + y_1 + 0,5 \right\rfloor$$

Continuous version:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1)$$



Bresenham line: algorithm

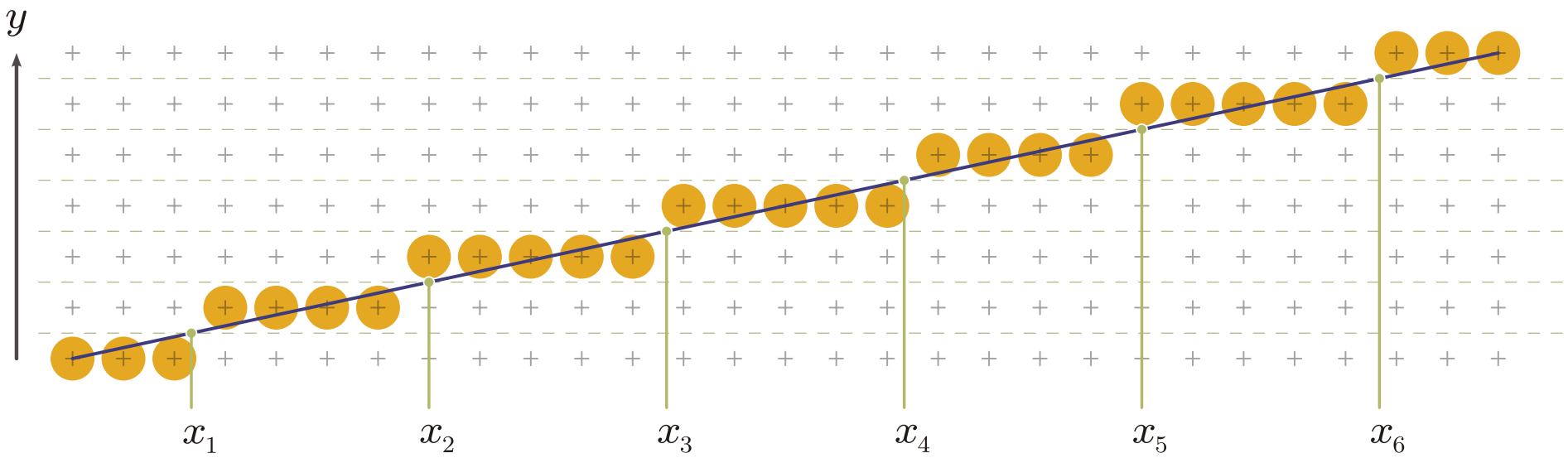
Error propagation algorithm

Discrete version ($\dot{x}$ and $\dot{y}$):

$$\dot{y} = \left\lfloor \frac{y_2 - y_1}{x_2 - x_1} \cdot (\dot{x} - x_1) + y_1 + 0,5 \right\rfloor$$

Error to be propagated:

$$e_{(1,0)} = \frac{y_2 - y_1}{x_2 - x_1}$$



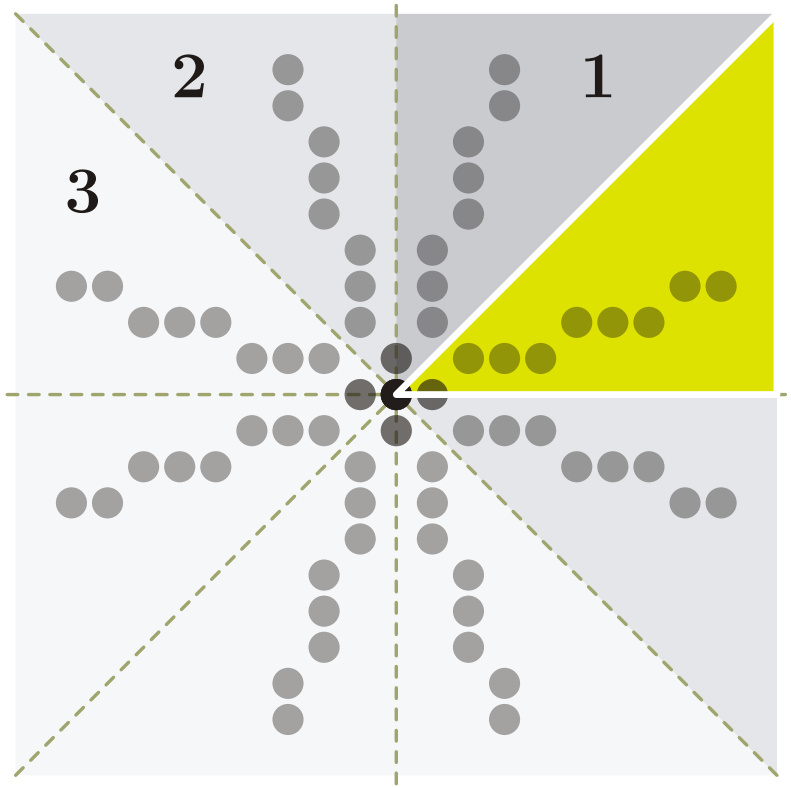
Bresenham line: algorithm

```
1  function line(x0, y0, x1, y1)
2      real deltax ← x1 − x0
3      real deltay ← y1 − y0
4      real deltaerr ← abs(deltay / deltax)
5          // Assume deltax != 0 (line is not vertical),
6          // note that this division needs to be done in a way
               that preserves the fractional part
7      real error ← 0.0 // No error at start
8      int y ← y0
9      for x from x0 to x1
10         plot(x,y)
11         error ← error + deltaerr
12         if error ≥ 0.5 then
13             y ← y + sign(deltay) × 1
14             error ← error − 1.0
```

Bresenham line algorithm with integer arithmetic

```
1   function plotLine(x0,y0,
        x1,y1)
2     dx ← x1 − x0
3     dy ← y1 − y0
4     D ← 2dy − dx
5     y ← y0
6
7     for x from x0 to x1
8       plot(x,y)
9       if D > 0
10        y ← y + 1
11        D ← D − 2dx
12      end if
13      D ← D + 2dy
```

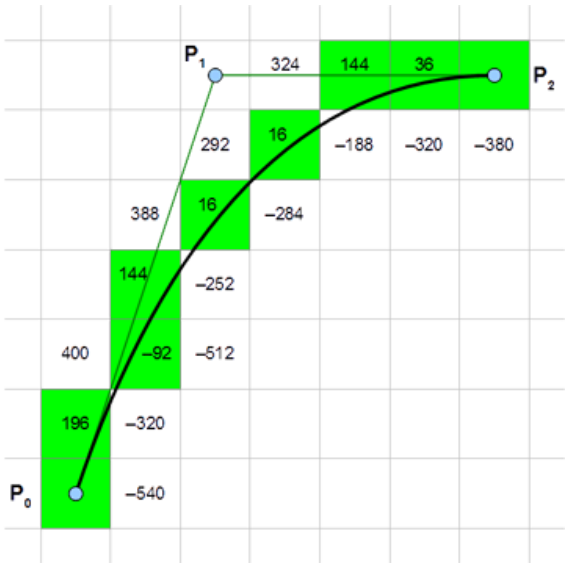
Use symmetry and octants for general position drawing

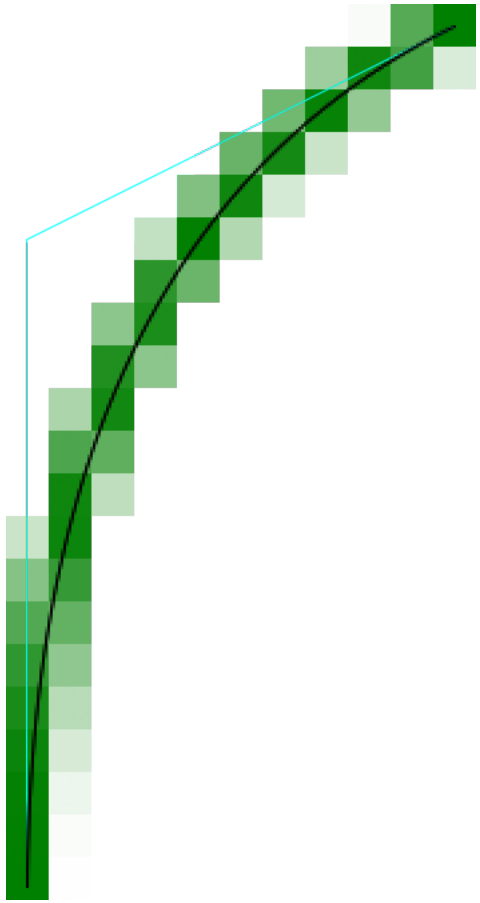


Bresenham algorithm

Can be adapted to

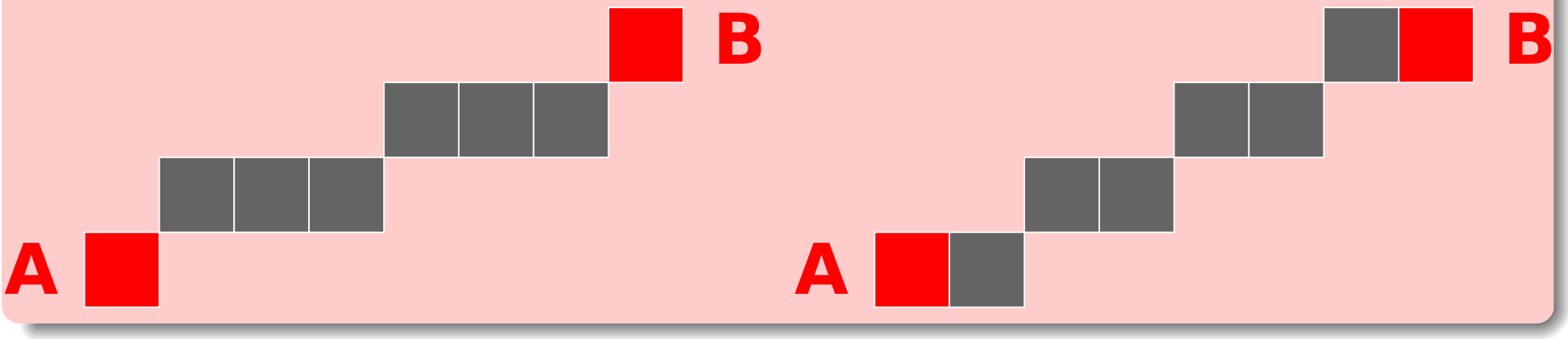
- circles and ellipses
- Bézier curve
- anti-aliased curves
- thickness





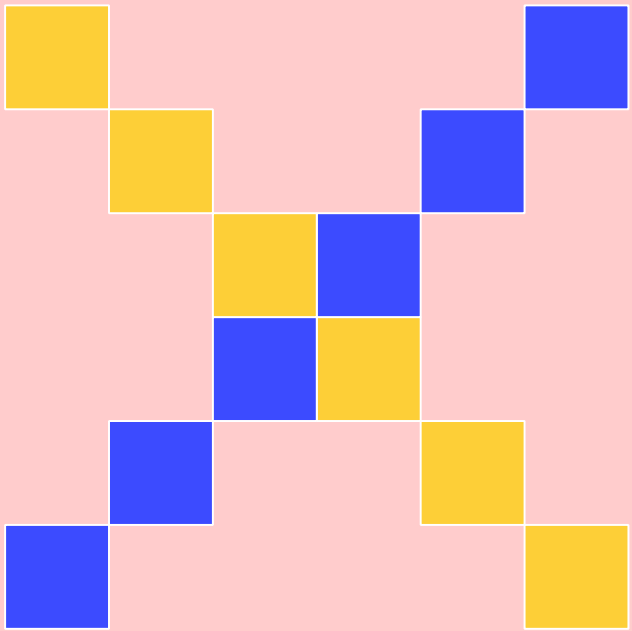
Problems

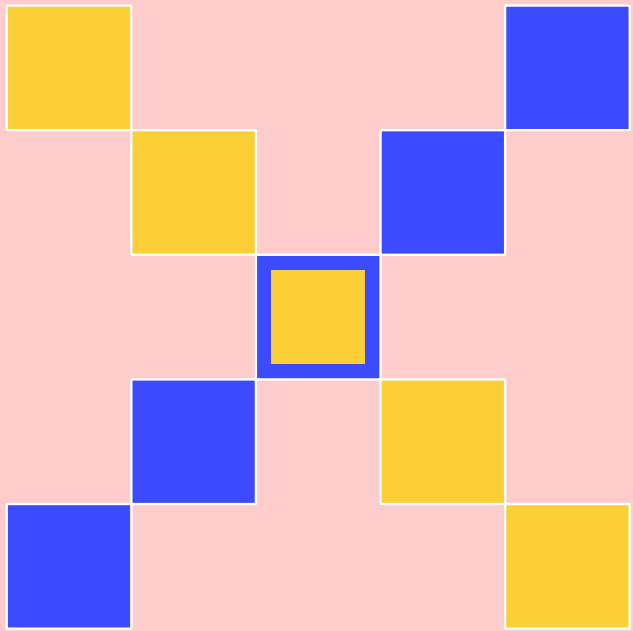
Unicity?



Problems

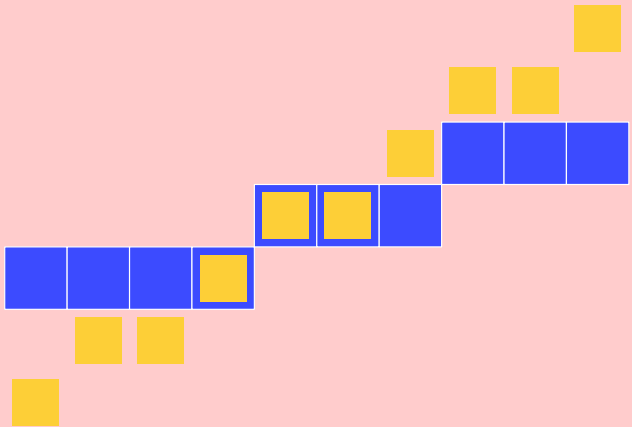
Intersection?

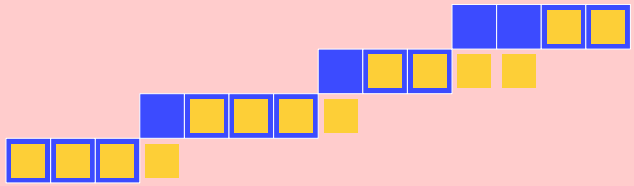




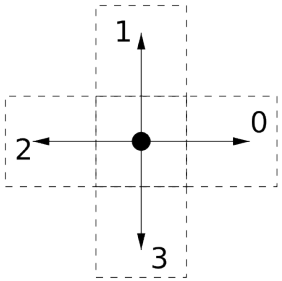
Problems

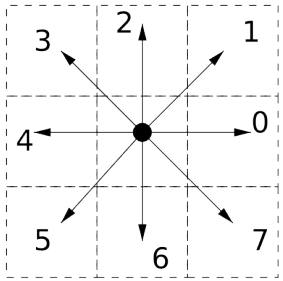
Connected intersection?

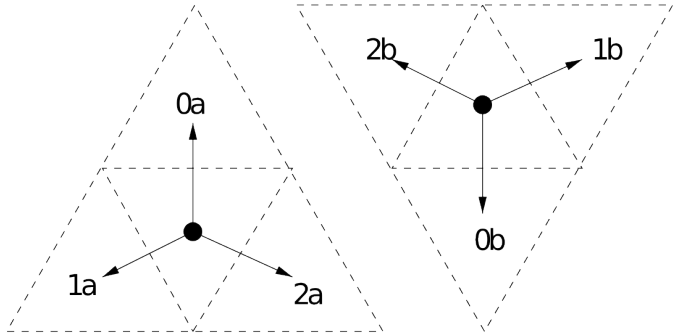


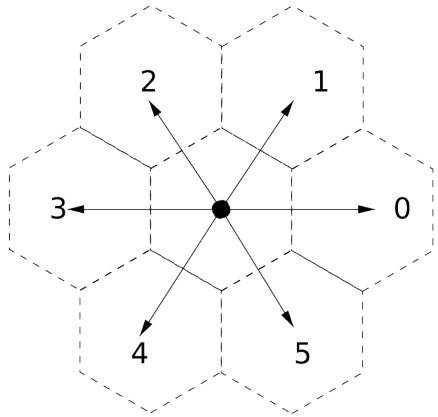


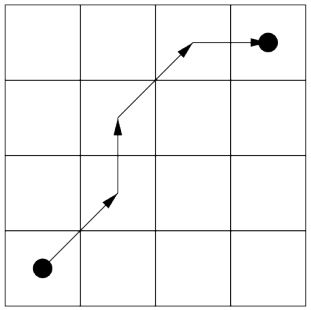
Freeman chain code



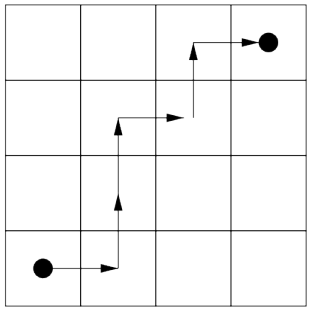




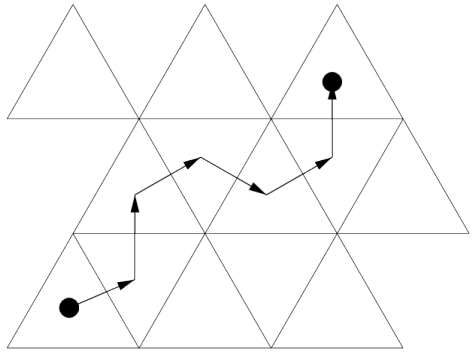




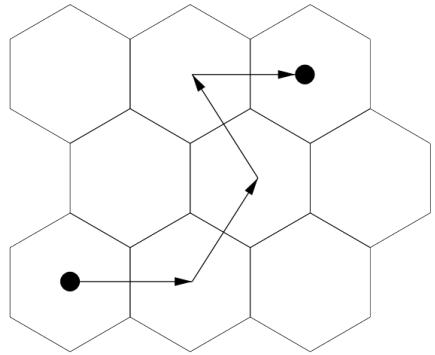
1–2–1–0



0–1–1–0–1–0

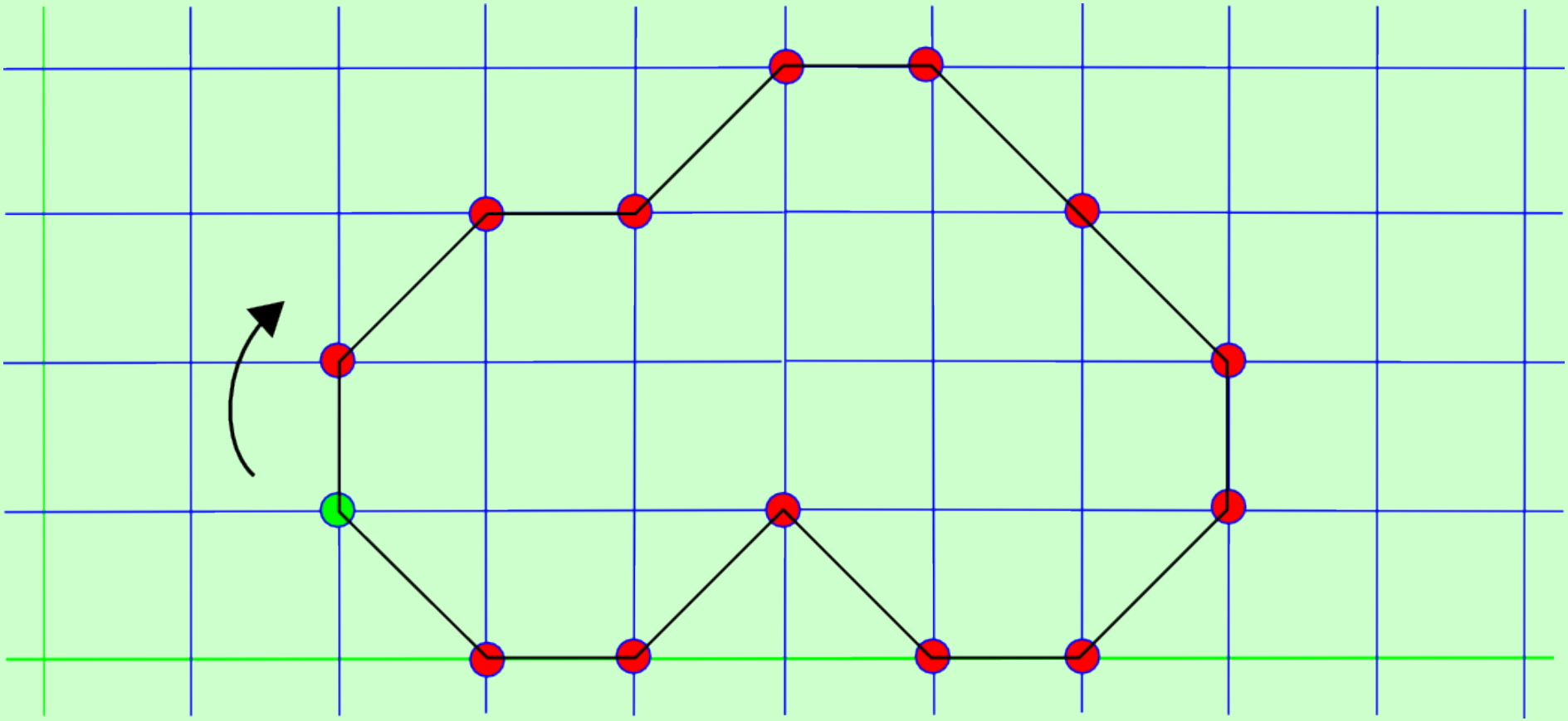


1b–0a–1b–2a–1b–0a



0–1–2–0

Freeman chain code



$$P_0 = (2,1), C = \{2,1,0,1,0,7,7,6,5,4,3,5,4,3\}$$

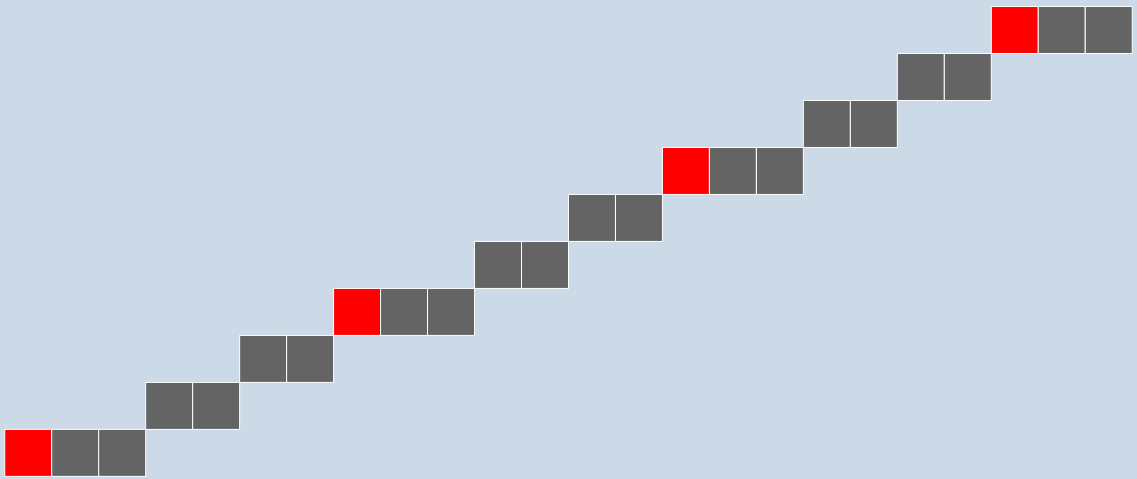
Bresenham line and periodicity

Chord property [Rosenfeld]

any point of the digital line is at a maximum distance of 1 of the real line.

Arithmetic results and more

- link with Diophantine equations $ax + by = c,\ a, b, c \in \mathbb{Z}$
- periodicity in representation (invariance by translation)
  - true for Freeman code
  - to go further, look at Farey Series



Polygonalization problem

Problem

Convert a digital contour into a polygon such that its digitization is the input digital set

Formalization of the problem

- $\mathcal{P} = \{p_i\}_{i \in [1;n]}$ denote a polygonal chain of n vertices
- Ojective: $\mathcal{A} = \{a_j\}_{j \in [1;m]}$, where
  - $\forall j, a_j \in \mathcal{P}$
  - $a_1 = p_1$ and $a_m = p_n$
  - $m \leq n$
  - Minimization of an error $E(\mathcal{P})$
- Min-ϵ problem: minimizes E with a given number of edges
- Min-# problem: minimizes the number of edges for $E < \epsilon$

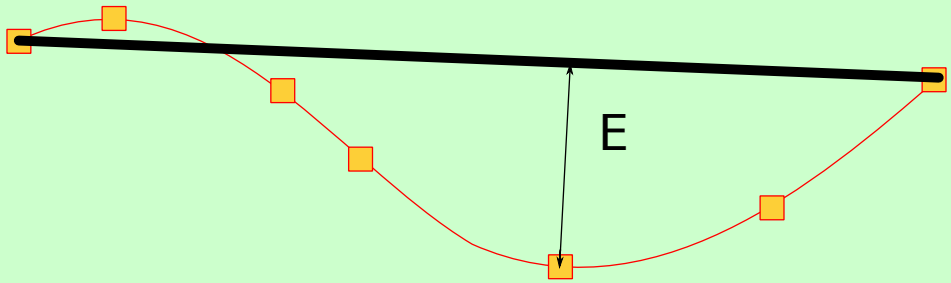
Polygonalization problem

Problem

Convert a digital contour into a polygon such that its digitization is the input digital set

Error examples

- Consider all points of $\mathcal{P}$
- Use classical distances L_1, L_2, L_∞



Polygonalization problem

Problem

Convert a digital contour into a polygon such that its digitization is the input digital set

2 classes of algorithms

- Classical: sequential algorithms, split, merge, split-and- merge, dominant points detection, relaxation labeling
- K-means, tabu search, genetic algorithms, ant colony optimization methods, and vertex adjustment methods

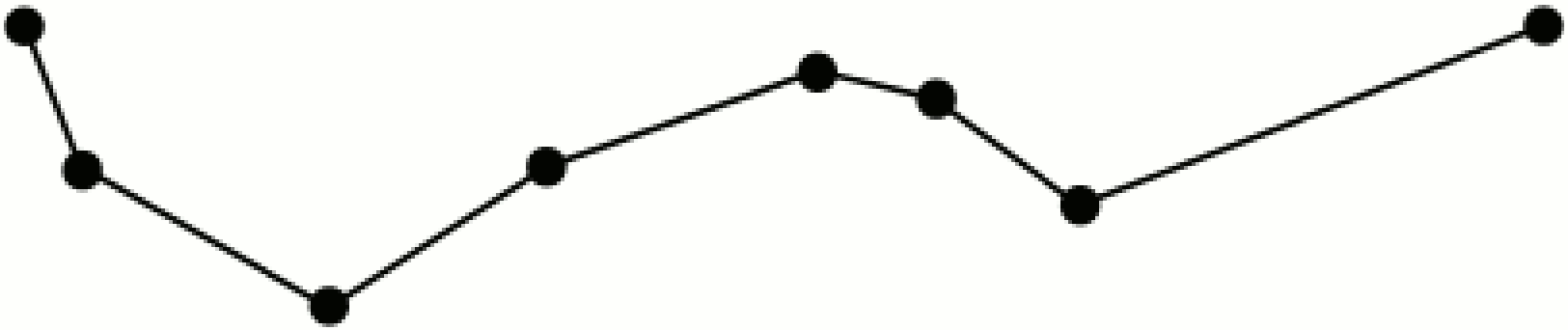
Sequential tracing approach

- Start from one point
- Consider all points in sequence order
- Keep last point before deviation is too high

Split: Ramer–Douglas–Peucker algorithm

Principle

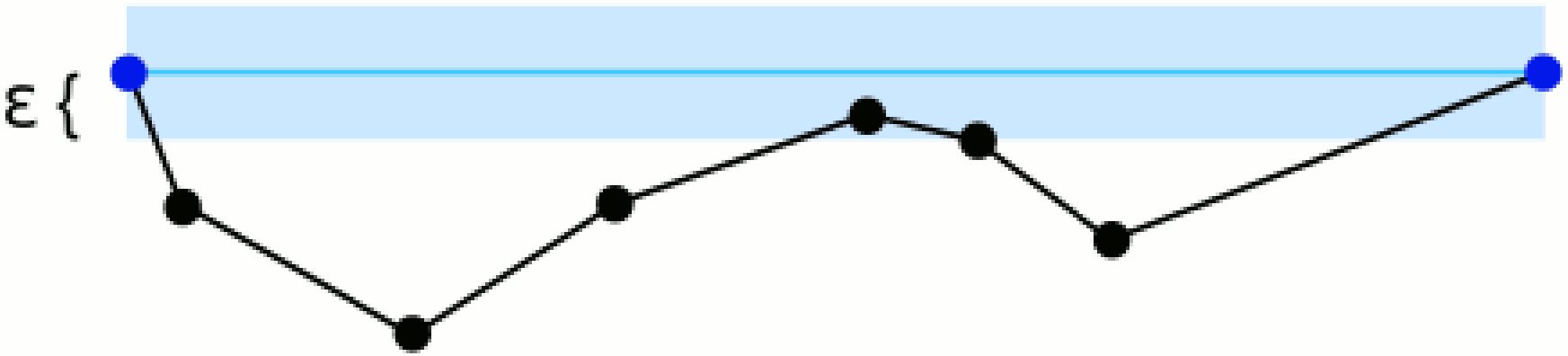
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

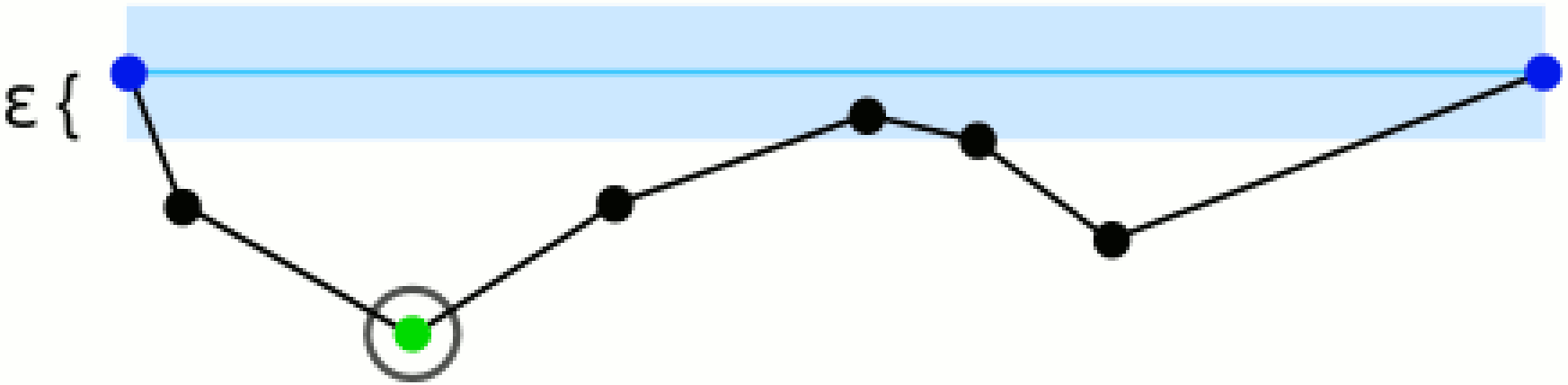
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

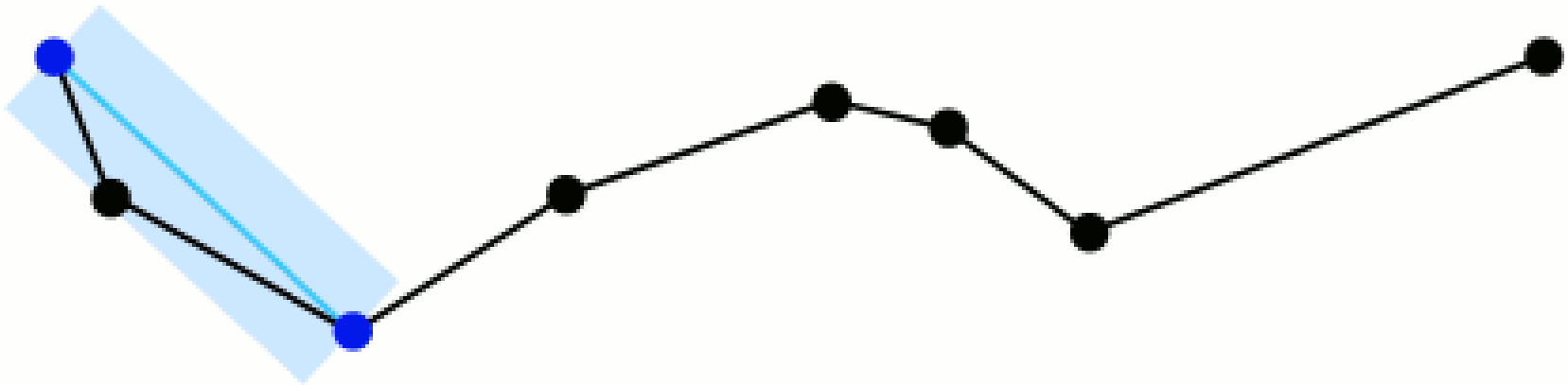
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

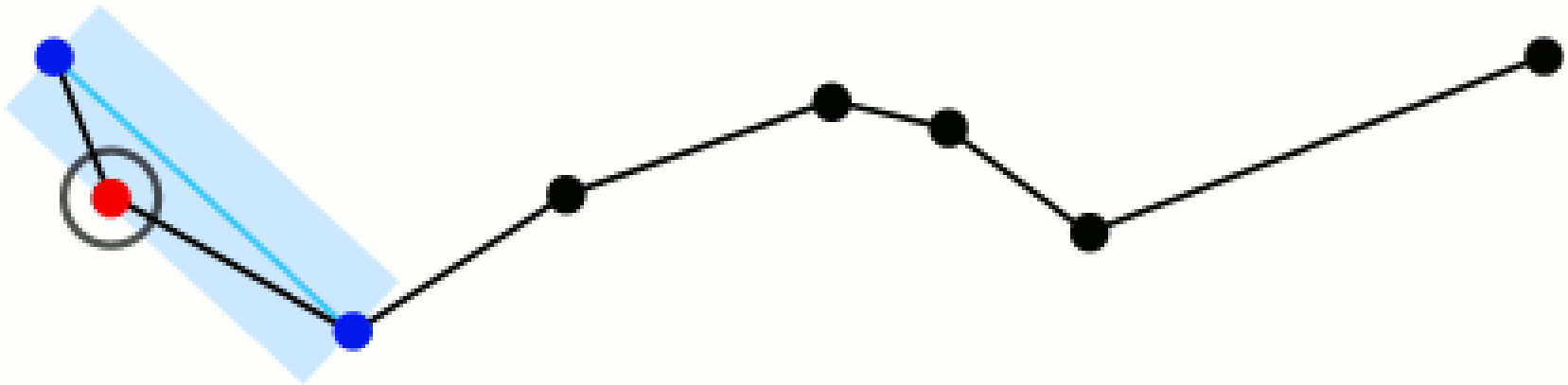
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

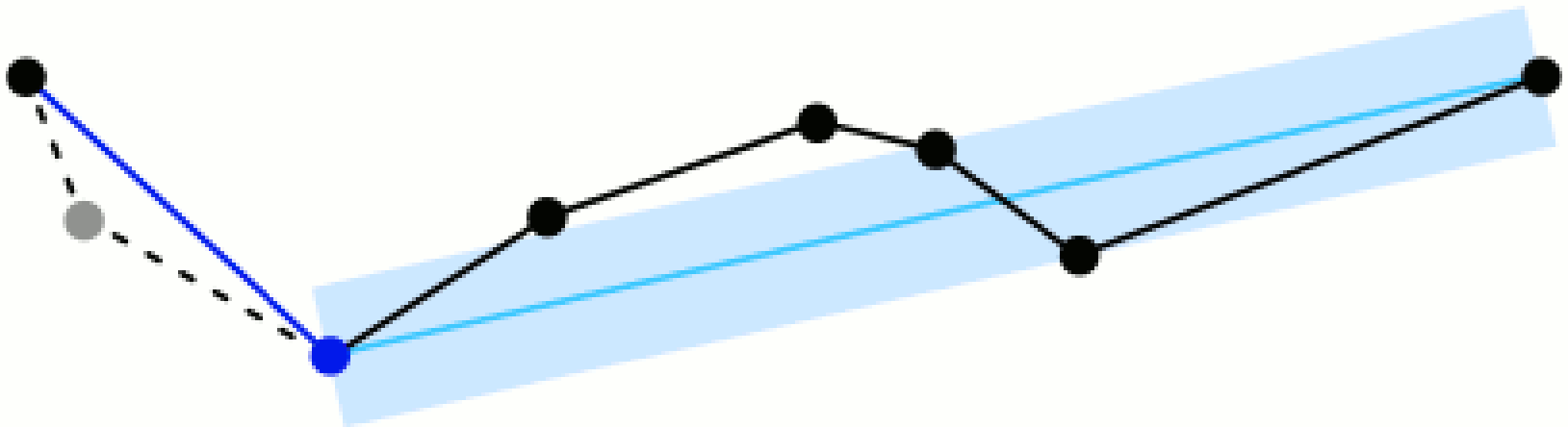
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

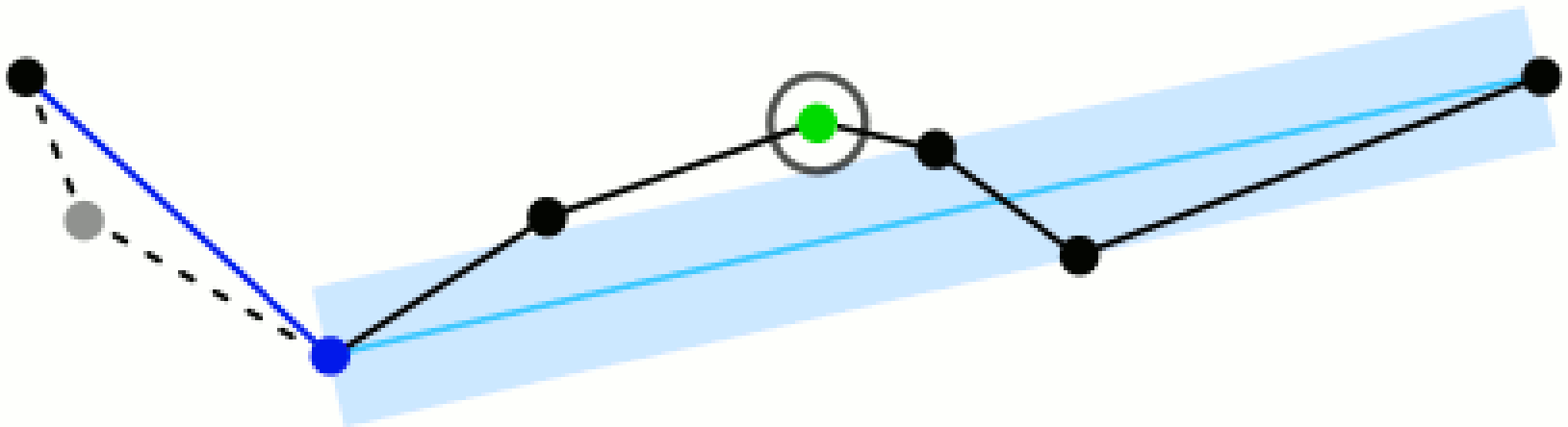
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

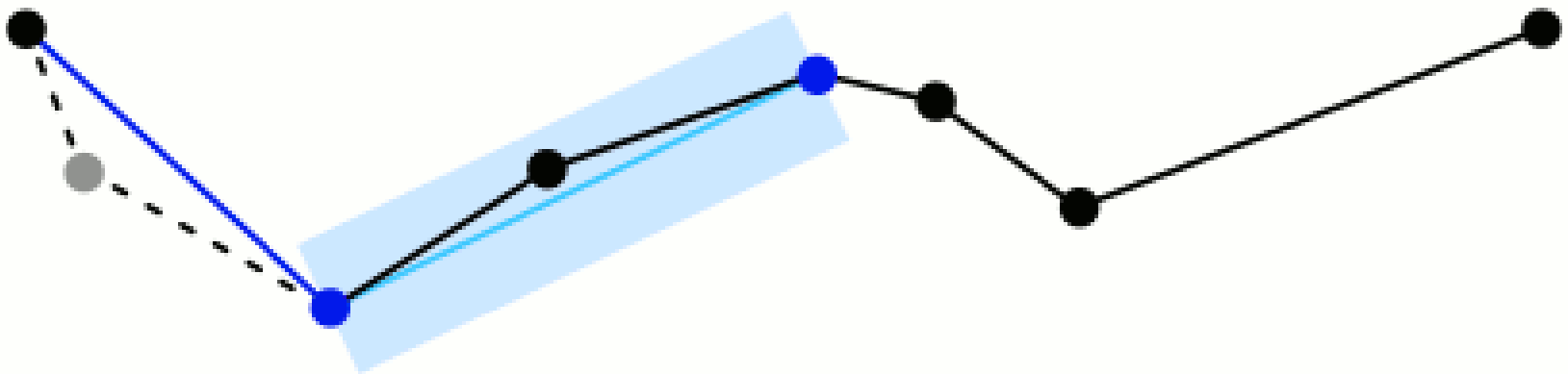
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

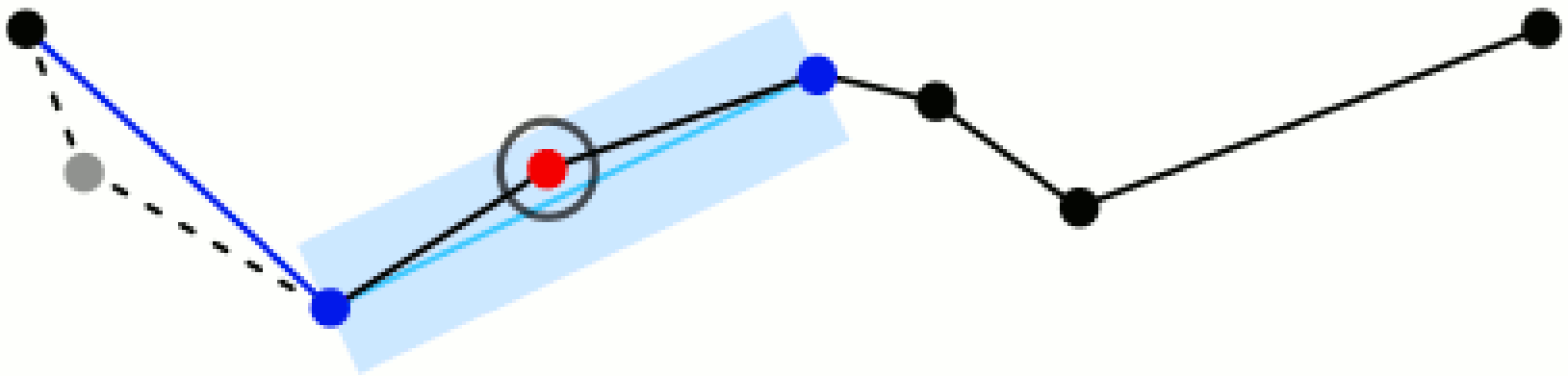
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

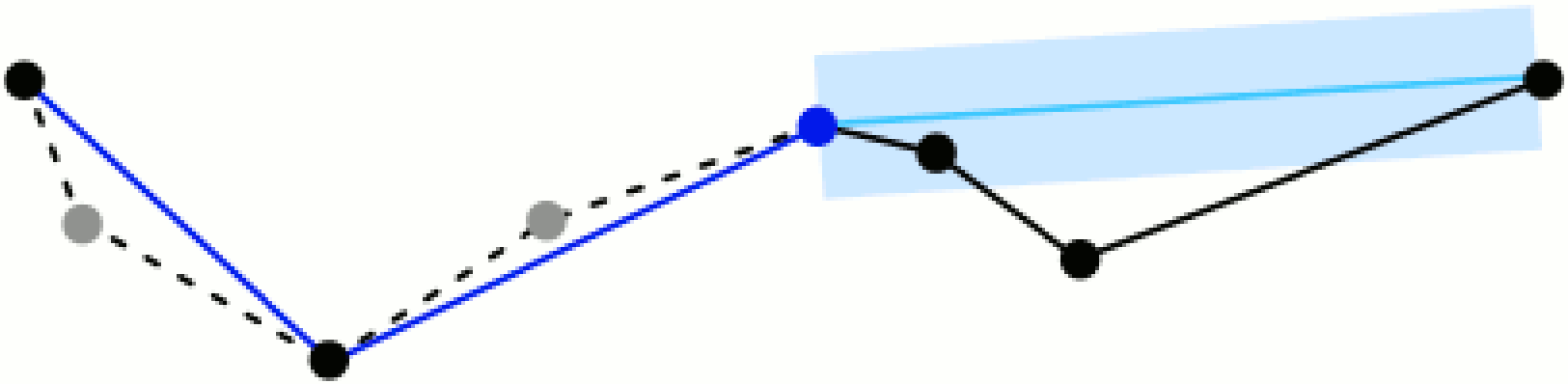
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

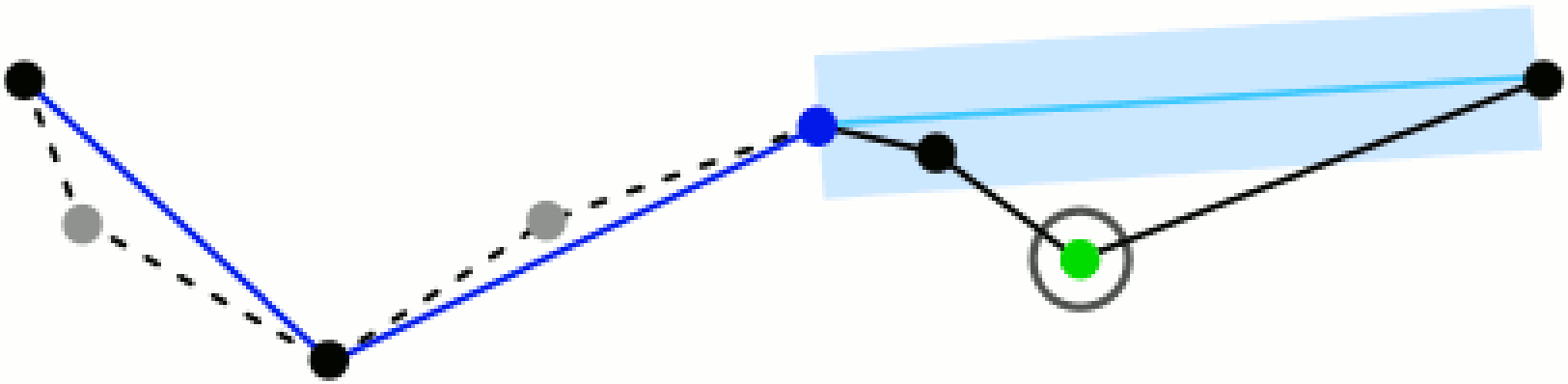
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

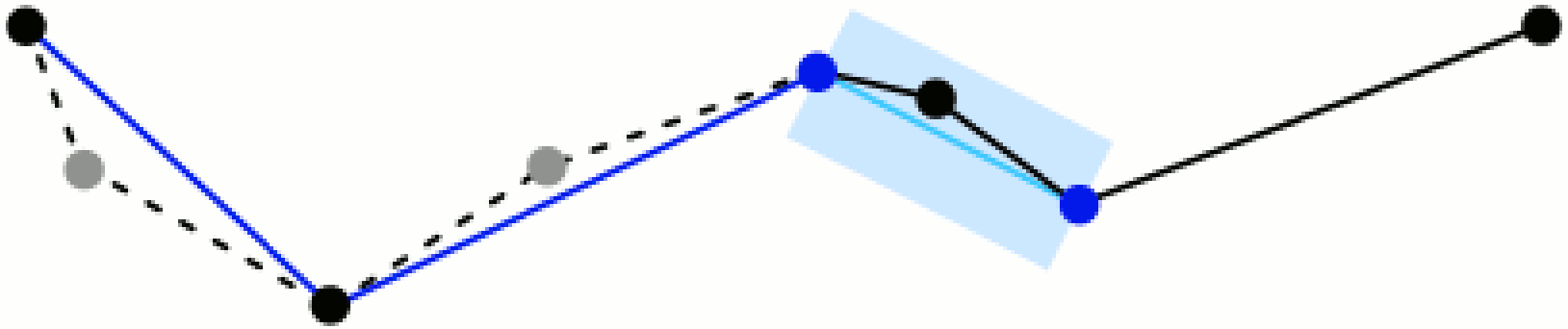
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

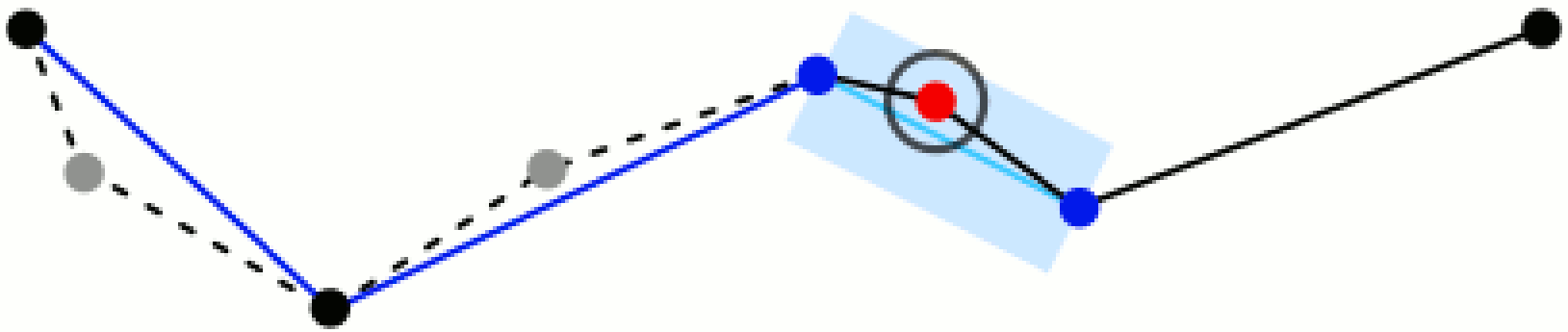
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

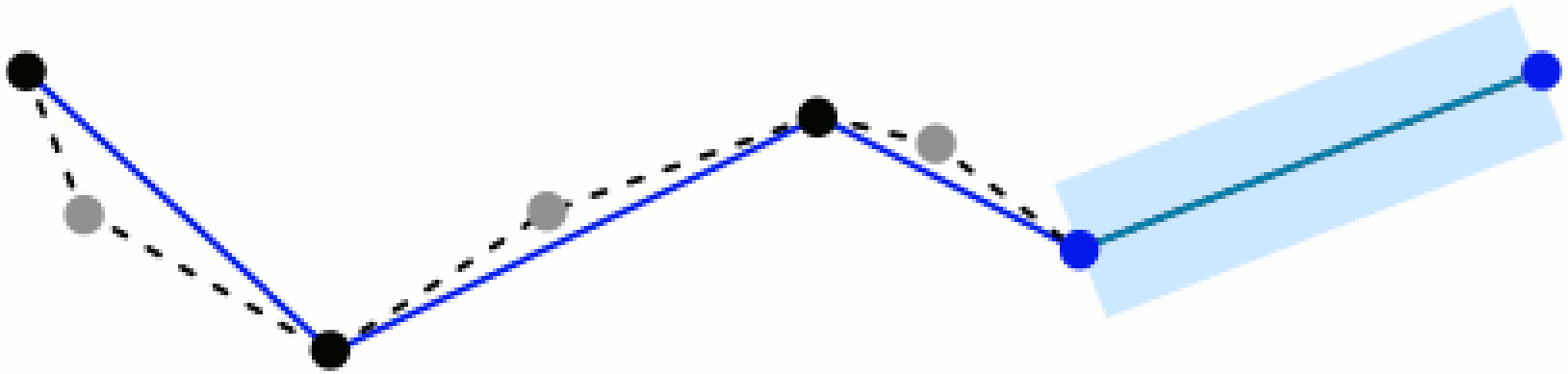
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

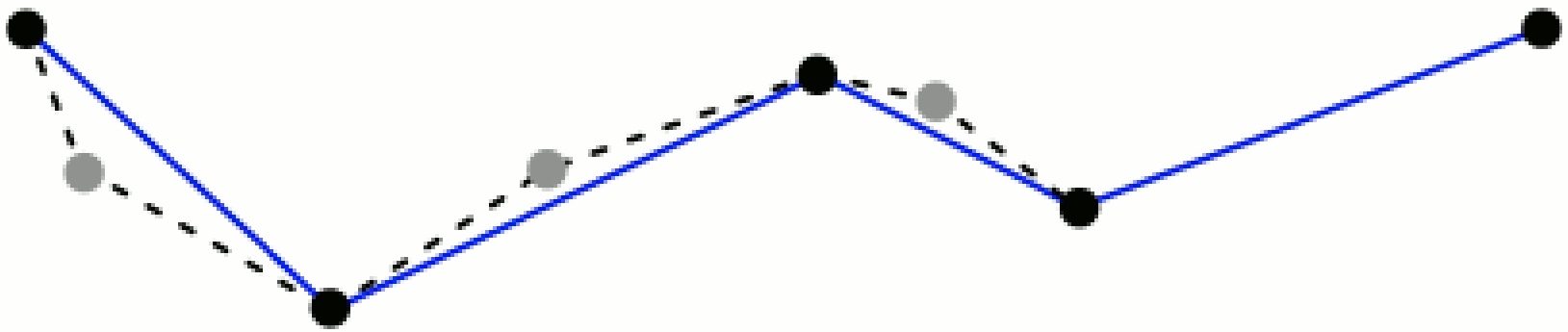
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

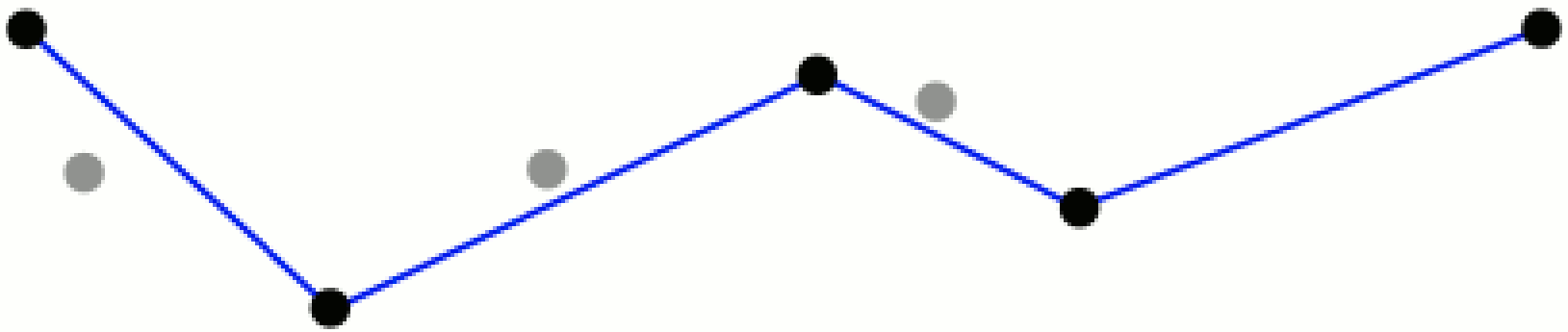
- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Split: Ramer–Douglas–Peucker algorithm

Principle

- Search for points far from a line
- $O(n^2)$ worst case, $O(n \log n)$ in average.



Other algorithms

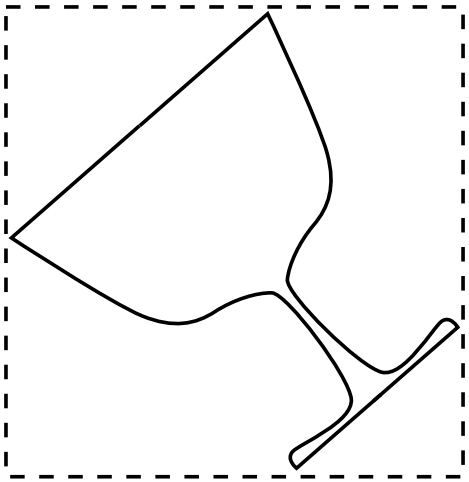
- K-means algorithm
- Visvalingam–Whyatt
- Reumann–Witkam
- Opheim simplification
- Lang simplification
- Zhao-Saalfeld

Problems and discussions

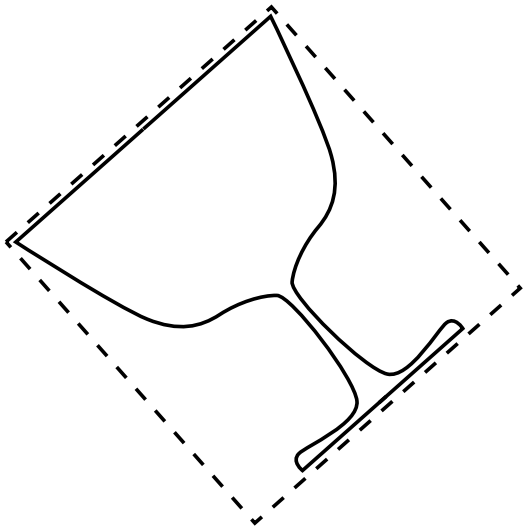
- Complexity
- Starting point
- Introduce curvature

Section 6

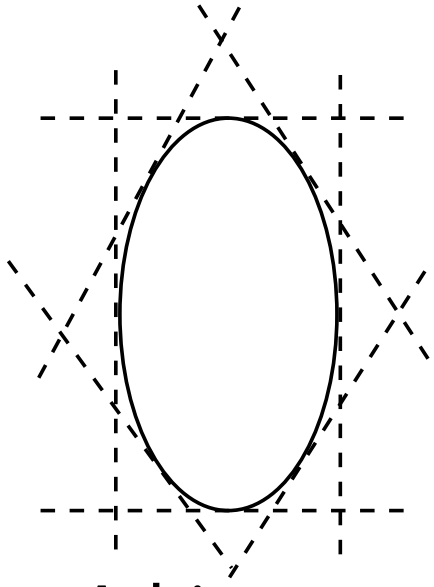
Enclosing objects



Axis-Aligned Bounding Box



Non-Axis-Aligned Bounding Box



Arbitrary Bounding Slabs

Enclosing objects

- Rectangles, hyper-rectangles
- Spheres

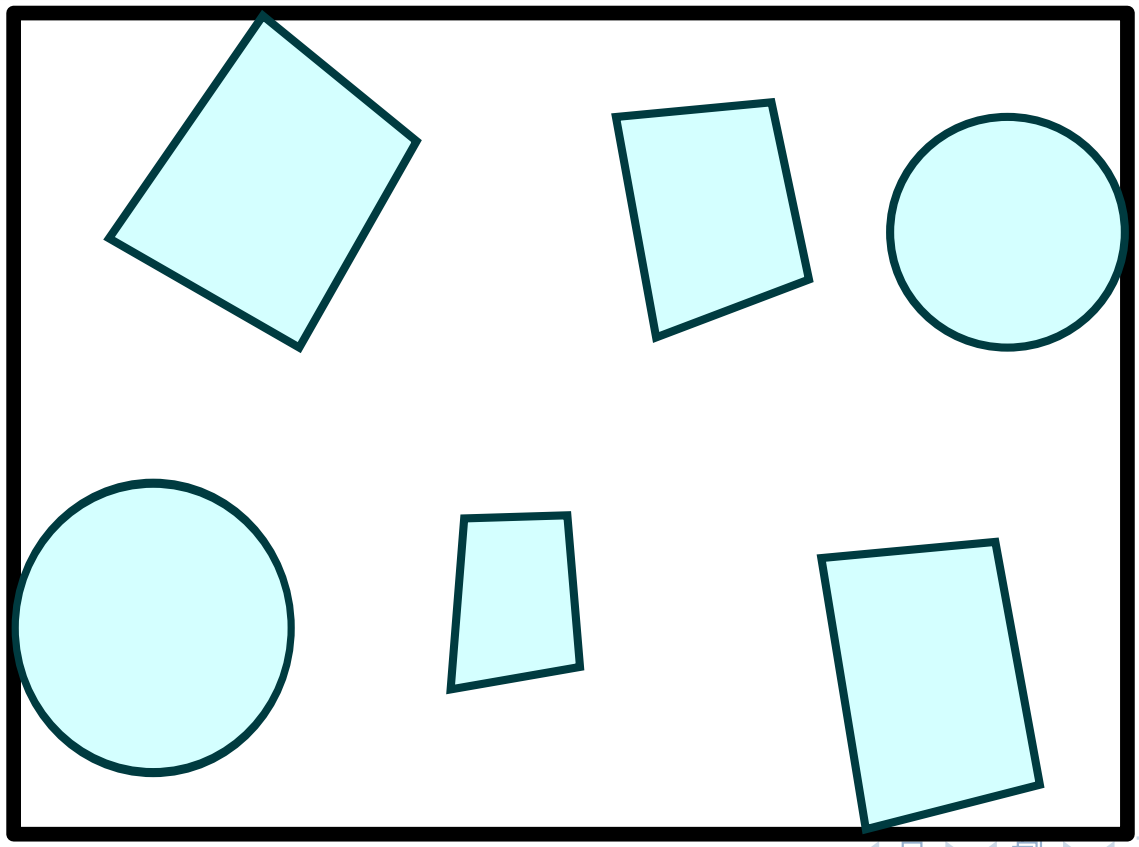
Minimum bounding box

- Box with the smallest **measure** within which all the points lie.
- **Measure** is area, perimeter, volume...

Minimum bounding rectangle

AABB: Axis Aligned Bounding Box

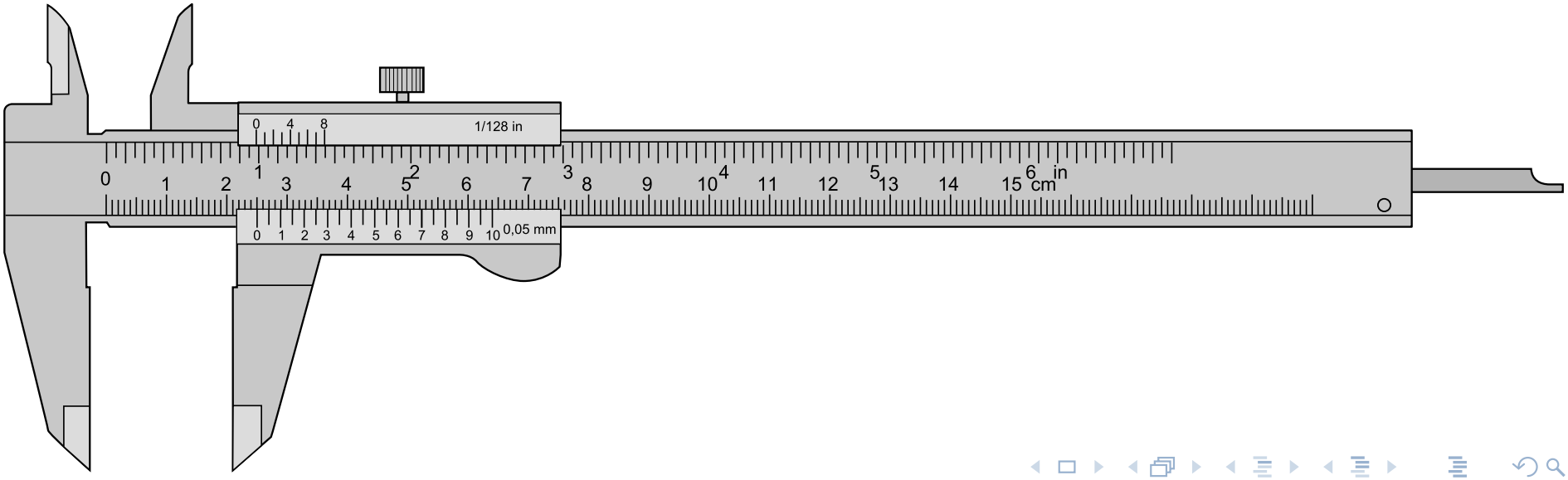
- Axis oriented.
- Use min and max of coordinates in x and y directions.



Oriented minimum bounding box

Convex polygons

- $O(n)$ algorithm for minimum area bounding BoundingBox
- Interesting property: one side is colinear to $conv(P)$
- Algorithm of *rotating calipers* (Shamos, 1978)
  - Feret diameter, width/minimum width, minimum perimeter oriented bounding box
  - Convex polygons: union, Minkowski sum, convex hull

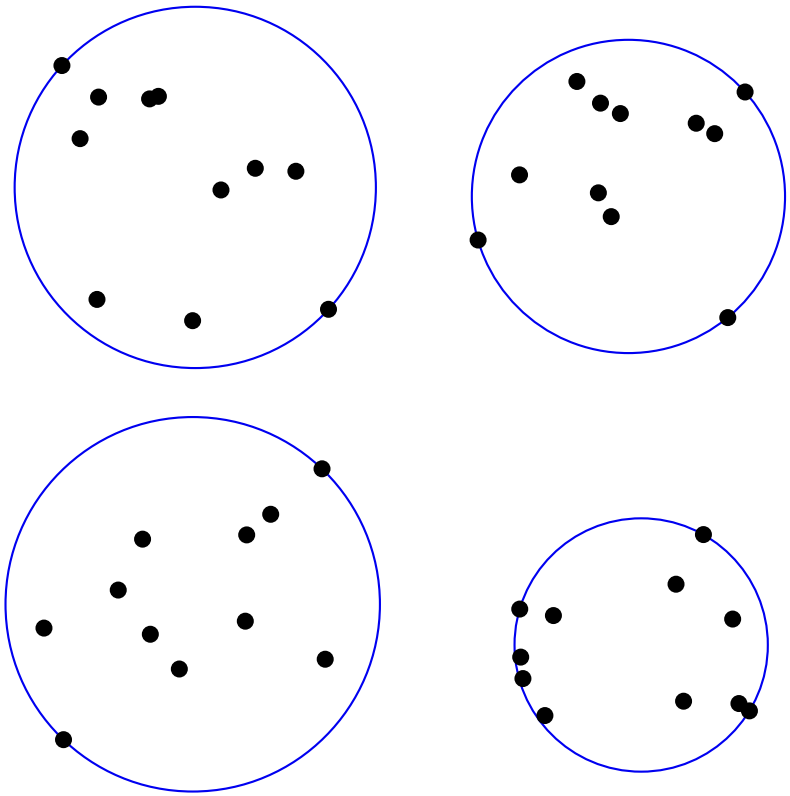


Smallest enclosing sphere

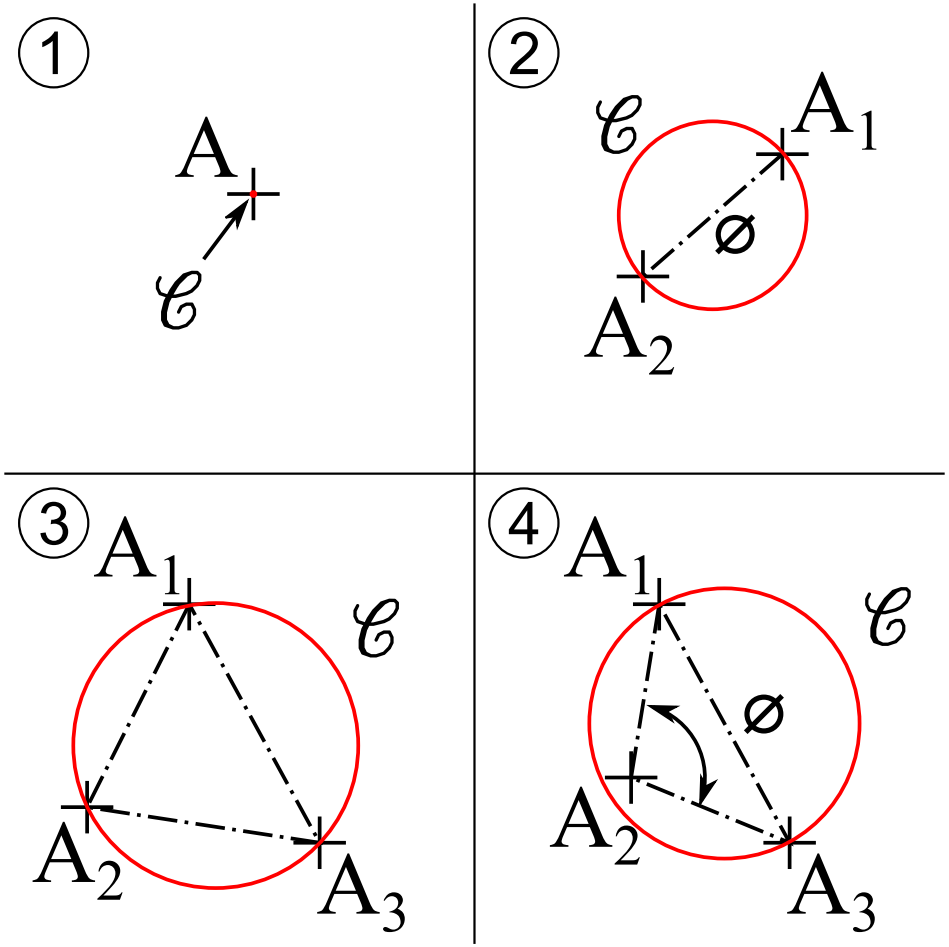
SEC: Smallest Enclosing Circle

- find the radius
- find the center (1-center problem)

Can be generalized and extended to higher dimensions.



Evident cases



Naive algorithm

Properties

- Minimal circle exists and is unique
- Except exceptionnally: it goes by 2 or 3 points.

Simple algorithm

Find the smallest circle:

- Consider the $n(n-1)/2$ circles whose diameter is given by all pairs of points
- Consider the $n(n-1)(n-2)/6$ circumcircles of 3 points among all

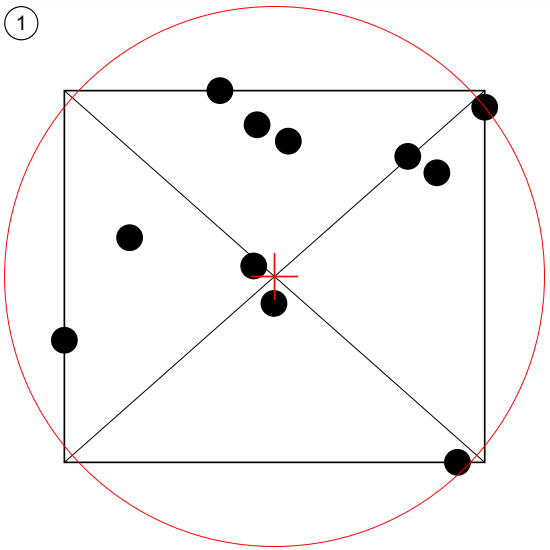
Overall complexity is in $O(n^4)$.

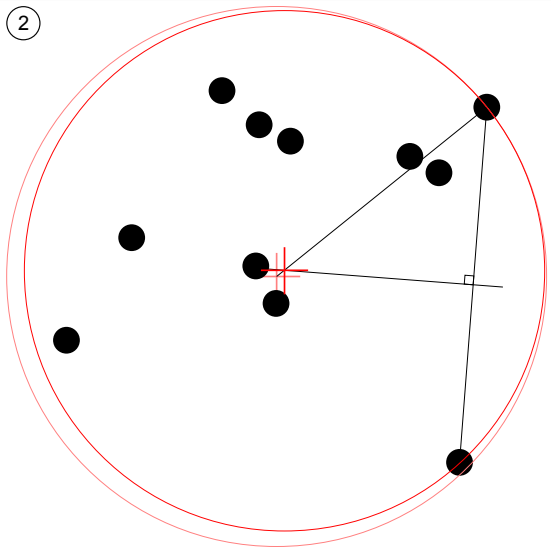
Geometric algorithm

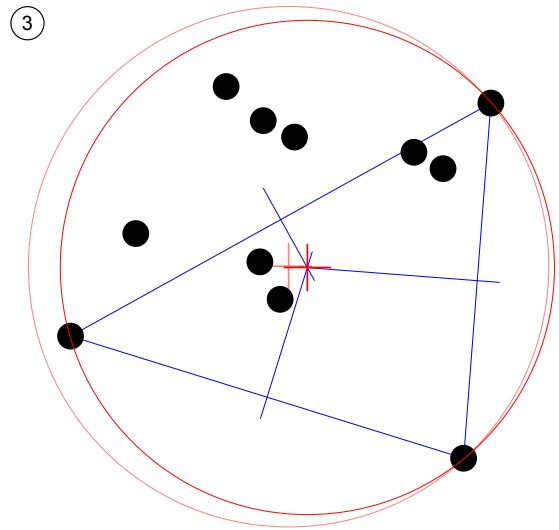
Manual approach

- Find a first guess, close to the result
- Move the center so that the circle touches another point
- Go on with an eventual 3rd point

Complexity in $O(n^2)$







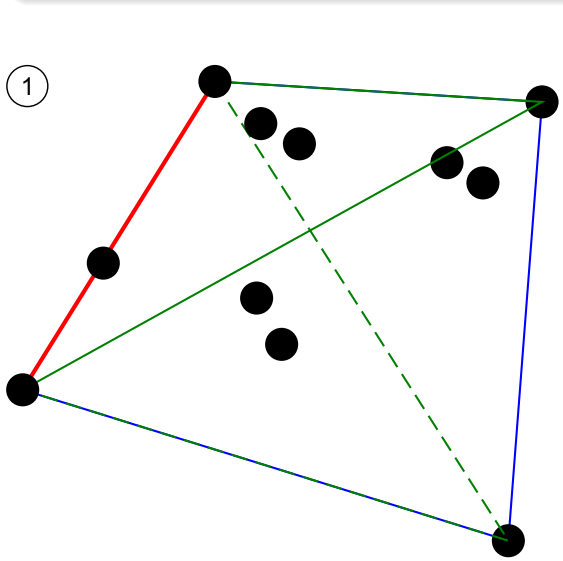
Chrystal algorithm 1885

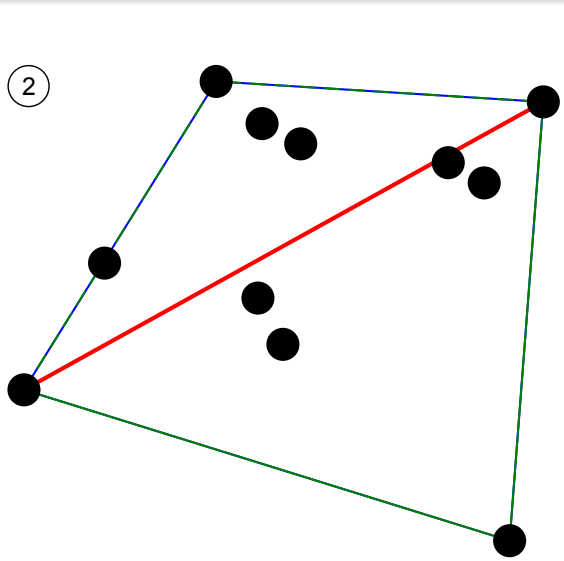
Restrain search to the m points of the convex enveloppe.

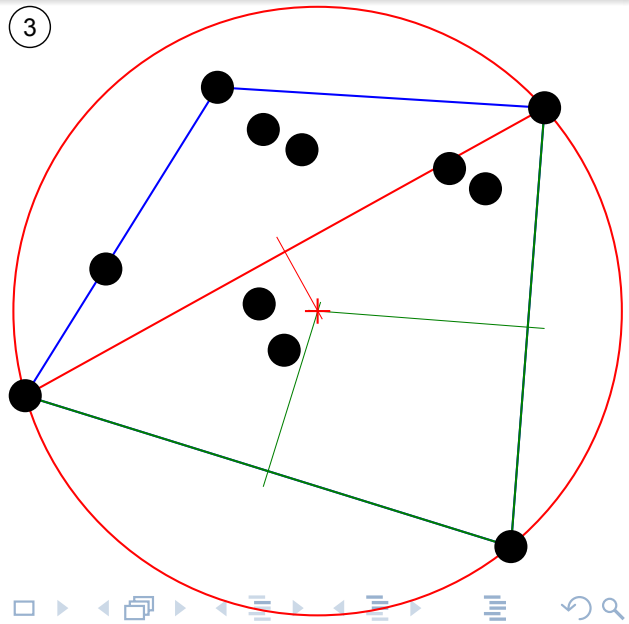
Algorithm

- Determine $conv(P)$
- Start by one side and considers all $m-2$ points

Complexity in $O(m^2)$







Shamos and Hoey, 1975

Rough description

- Use of the Voronoi diagram
- Test the vertices of the diagram

Complexity: $O(n \log n)$

Megiddo, 1983

- Prune and search algorithm
- At each step, eliminate 1/16 of the points
- Nimrod Mediddo, Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems, SIAM Journal of Computing, vol. 4, 1983, p.766-769.

Complexity: $O(n)$

Welzl, 1991

- Linear Programming, using Seidel algorithm
- Recursive

Complexity: $O(n)$

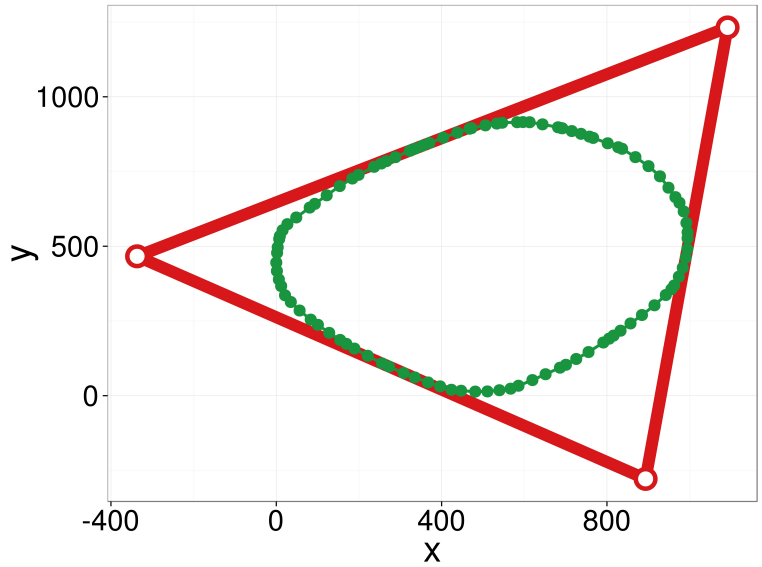
Welzl E. (1991) Smallest enclosing disks (balls and ellipsoids). In: Maurer H. (eds) New Results and New Trends in Computer Science. Lecture Notes in Computer Science, vol 555. Springer, Berlin, Heidelberg

Maximum perimeter/area polygon of k vertices

- Boyce, Dobkin, Drysdale, Guibas. $O(kn \log n + n \log^2 n)$
- The searched vertices lay on the convex hull

Minimum enclosing triangle

- Klee and Laskowski, 1985, $O(n \log^2 n)$
- O'Rourke et al., 1986, $\theta(n)$
- Inspired by rotating calipers

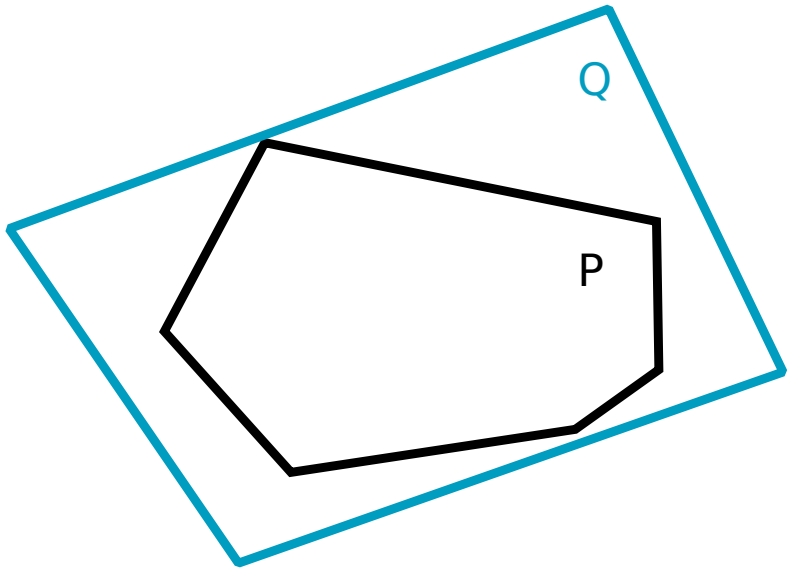


Minimum area enclosing k-gons

Problem

Find the polygon with k sides (k-gon) of minimum area enclosing a polygon of n vertices.

- Aggarwal et al., 1985, $O(n^2 \log n \log k)$

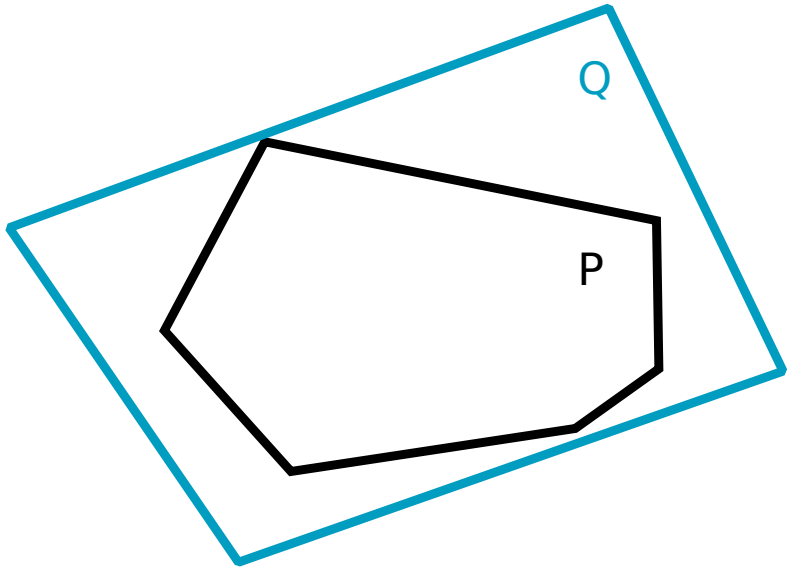


Minimum perimeter enclosing k-gons

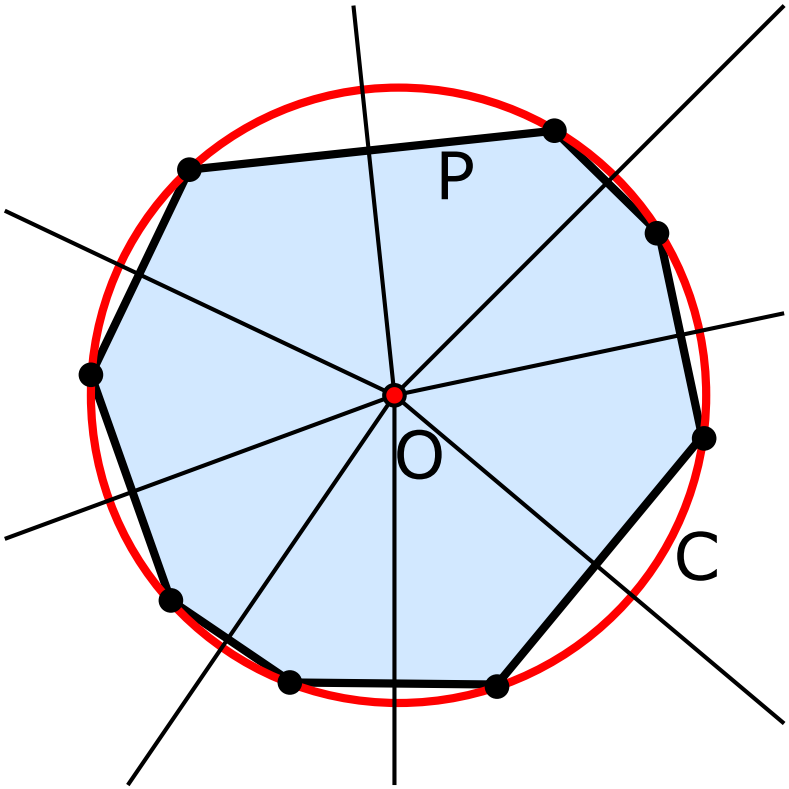
Problem

Find the polygon with k sides (k-gon) of minimum perimeter enclosing a polygon of n vertices.

- Mitchel and Polishchuk, 2006
- $O(nk \log k)$



Circumcircle



- A circle passing by all the points of the set.
- Generally different from the minimum circle.
- Does not exist for every polygon
- Such a Polygon is called cyclic

If it exists, may be different from the minimum englobing circle

All triangles are cyclic.

Other problems

Compute the smallest enclosing circle of k points among n.

Section 7

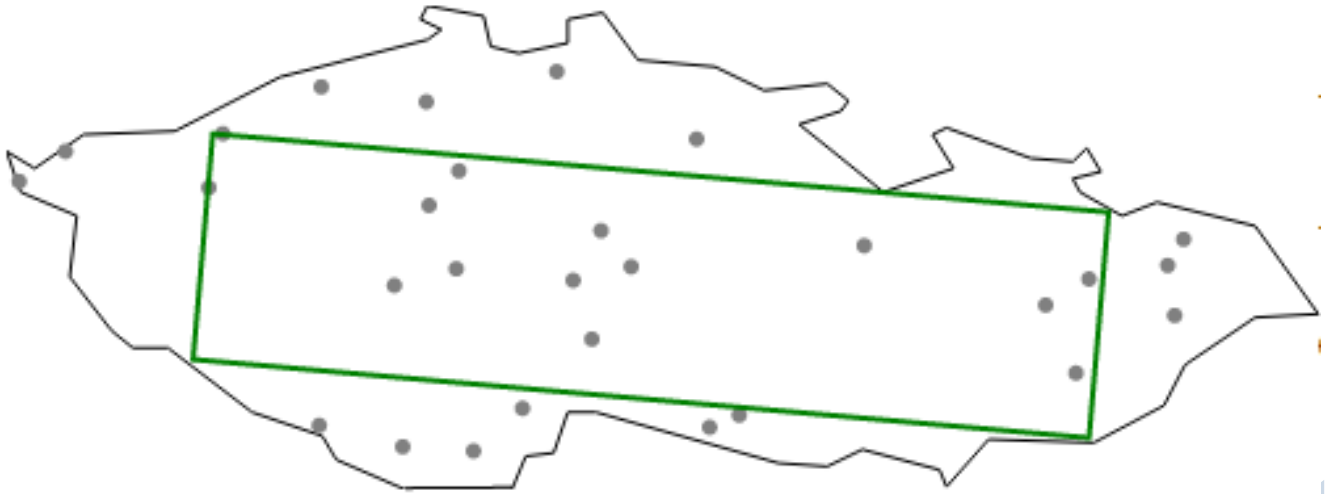
Inscribed objects

Problems

- Find largest inscribed object in polygons / point sets
- Rectangles (axis-aligned or not)
- Spheres
- K-gons

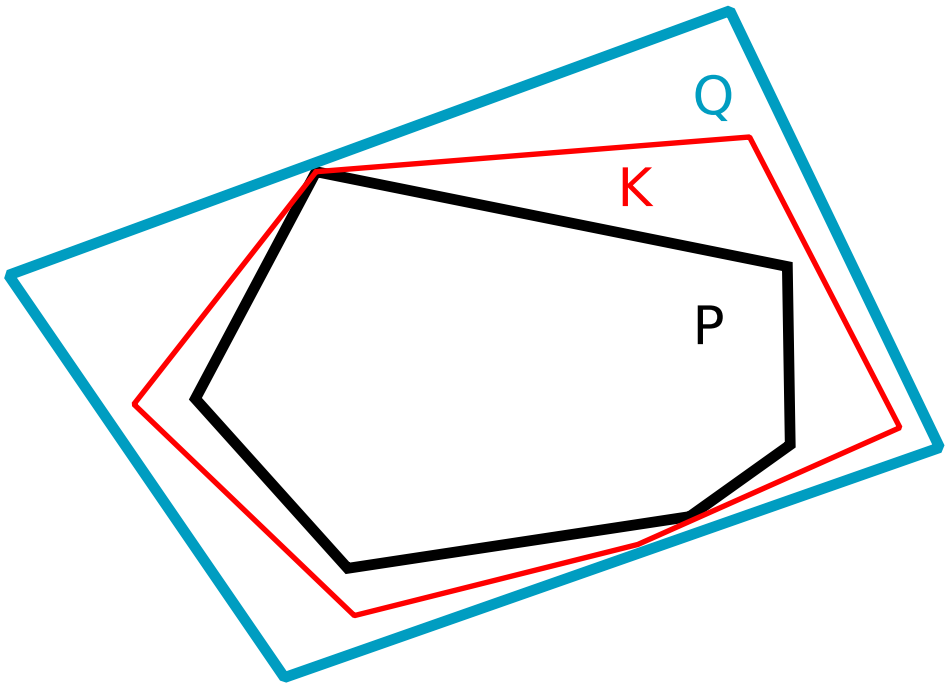
Rectangles

Author	Year	Polygon	Orientation	Complexity
Fisched and Höffgen	1994	Convex	axis-aligned	$O(\log^2 n)$
Alt et al.	1995	Convex	axis-aligned	$O(\log n)$
Daniels et al.	1997	Arbitrary	axis-aligned	$O(n \log^2 n)$
Boland and Urrutia	2001	Arbitrary	axis-aligned	$O(n \log n)$
Knauer et al.	2010	Convex	Arbitrary	$O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log n)$
Knauer et al.	2010	Arbitrary	Arbitrary	$O(\frac{1}{\epsilon} n^3 \log n)$
Molano et al.	2010	Arbitrary	Arbitrary	$O(n^3)$



Minimal convex nested polygons

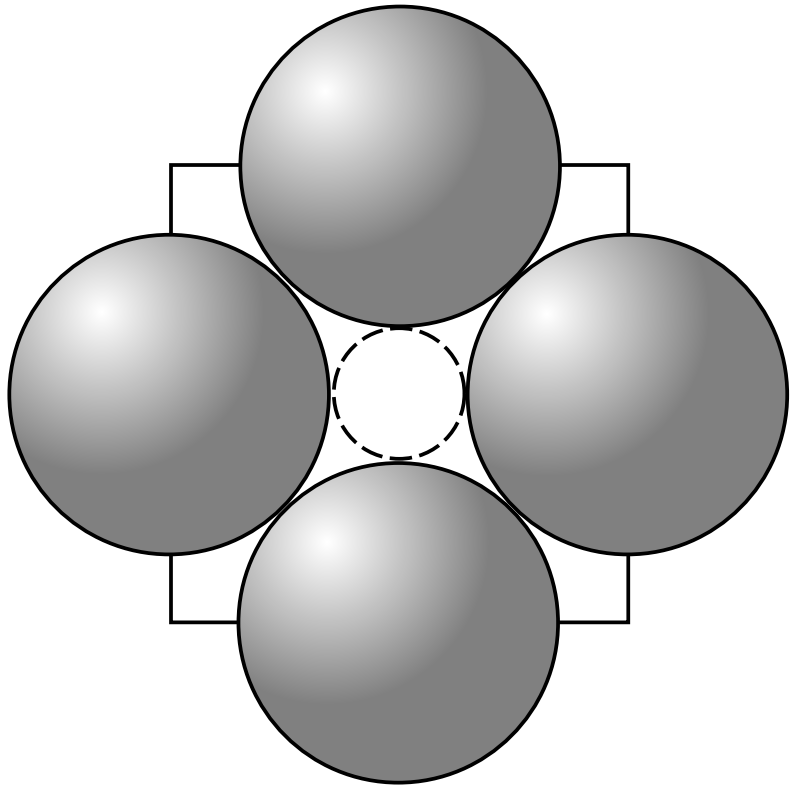
Aggarwal, booth, O'rourke, suri, 1989, $O(n \log k)$



Largest empty sphere

Problem statement

Find largest sphere that does not overlap any other object.



Conclusion

- Basic algorithms for computational geometry
- Applied mathematics and numerical problems
- Next: Voronoi diagrams, Nearest Neighbors, Alpha Shapes