Optimal Algorithm for the Planar Two-Center Problem

SoCG 2024

Speaker: Kyungjin Cho

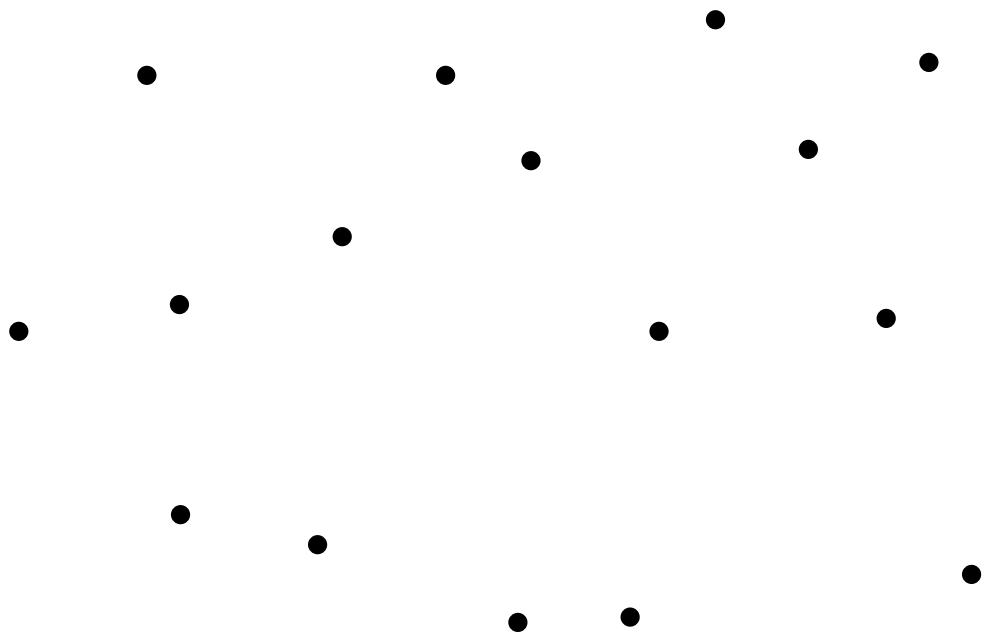
Eunjin Oh
Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)

Haitao Wang
Kahlert School of Computing
University of Utah

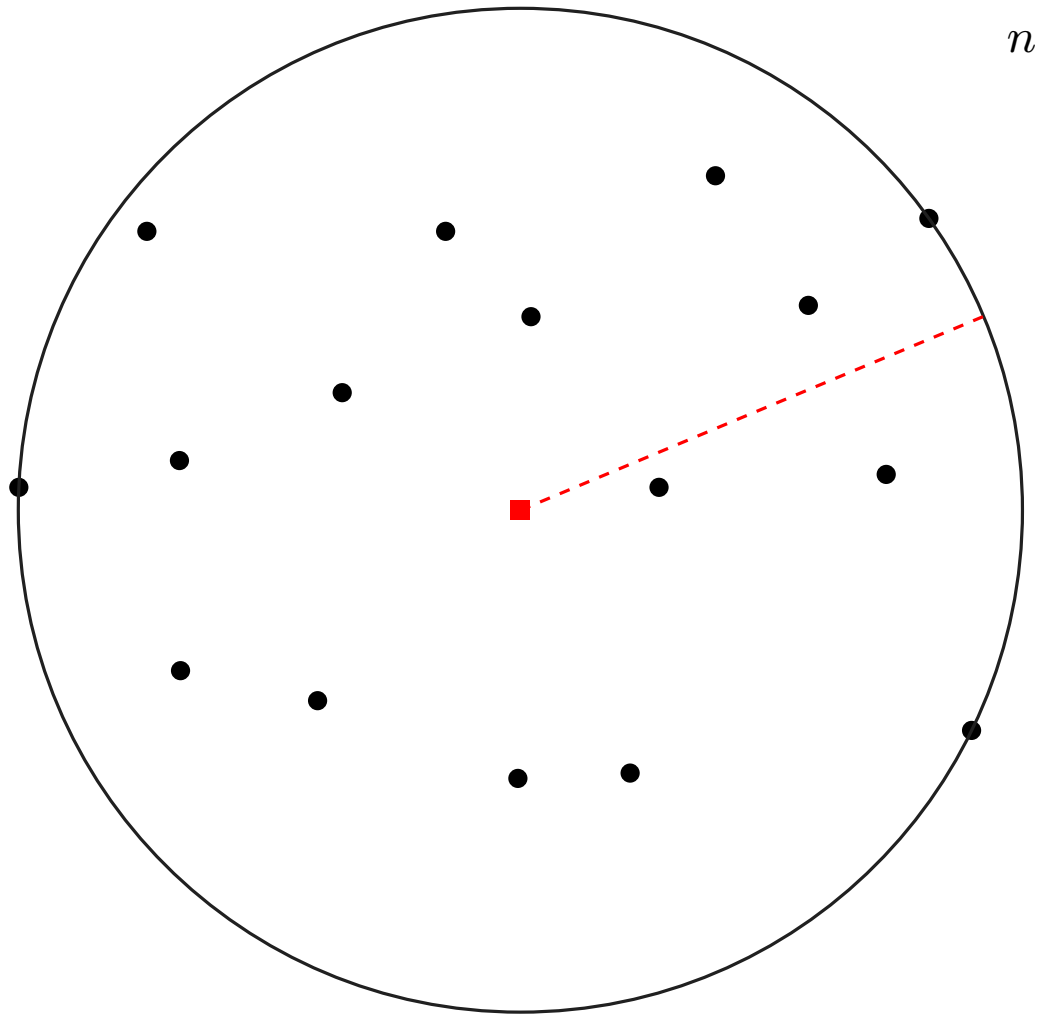
Jie Xue
Department of Computer Science
New York University Shanghai

Problem Definition

n points in $\mathbb{R}^2$



Problem Definition

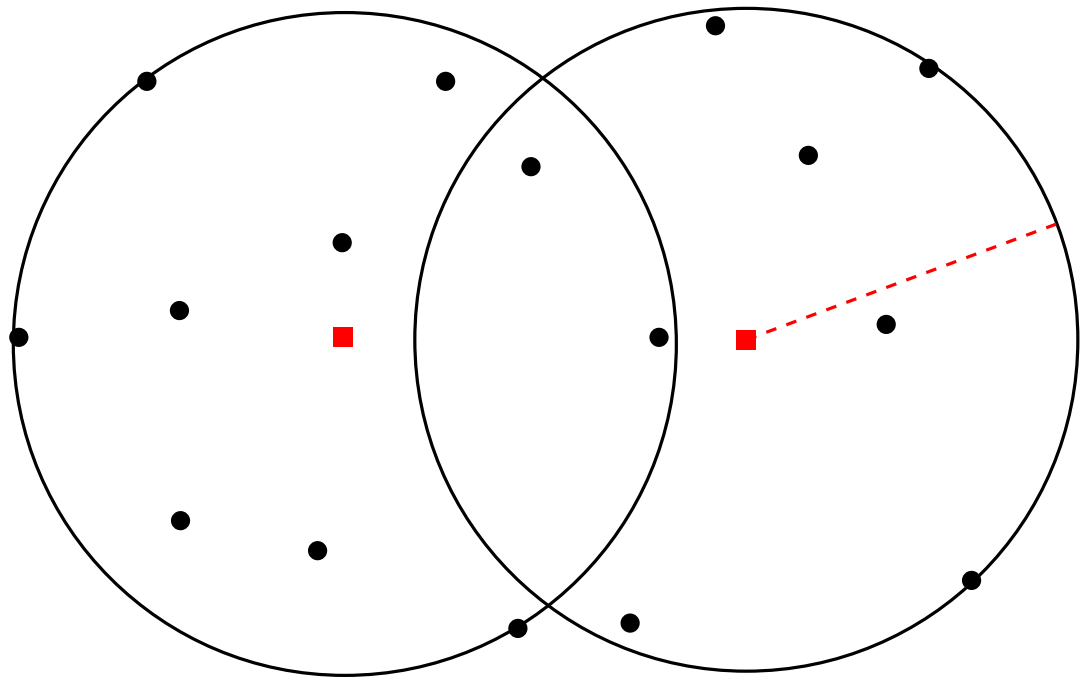


n points in $\mathbb{R}^2$

1-center problem: It requires $\Theta(n)$ time to solve in $\mathbb{R}^d$.

Problem Definition

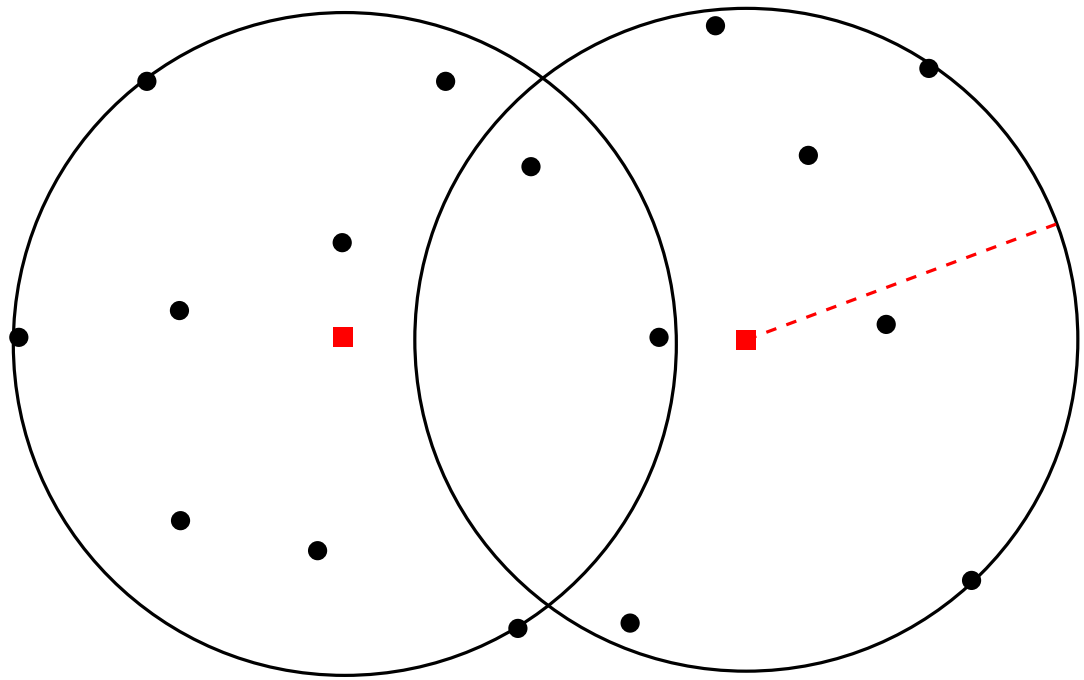
n points in $\mathbb{R}^2$



2-center problem: Finding two congruent disks D_1 and D_2 covering all points

Problem Definition

n points in $\mathbb{R}^2$

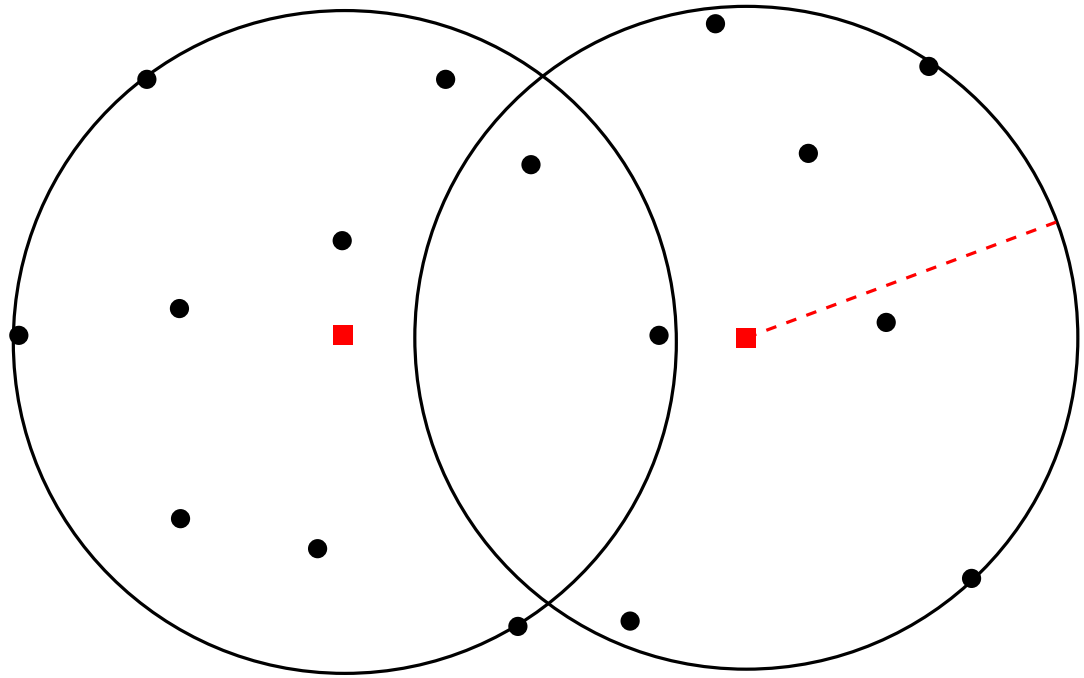


2-center problem: Finding two congruent disks D_1 and D_2 covering all points

Lowerbound: It requires $\Omega(n \log n)$ time to solve. [Eppstein SODA'97]

Problem Definition

n points in $\mathbb{R}^2$



2-center problem: Finding two congruent disks D_1 and D_2 covering all points

Lowerbound: It requires $\Omega(n \log n)$ time to solve. [Eppstein SODA'97]

Question: Is there a deterministic $O(n \log n)$ time algorithm?

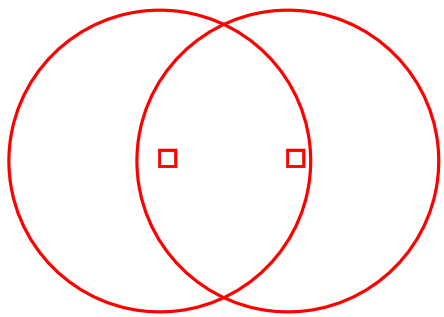
Sketch of History

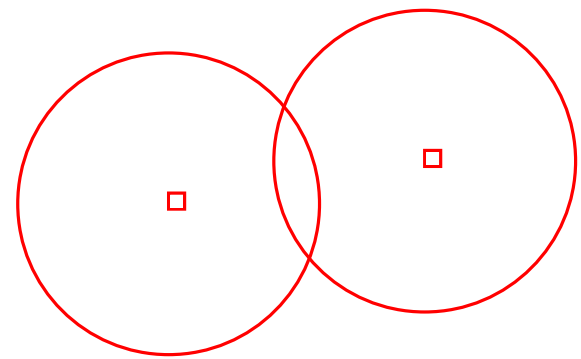
[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem





[Sharir 96] $O(n \log^9 n)$ time

[Sharir 96] $O(n \log^8 n)$ time

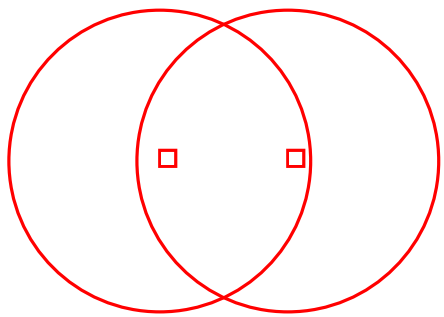
[Parametric Search]
Solving optimization using decision algorithms

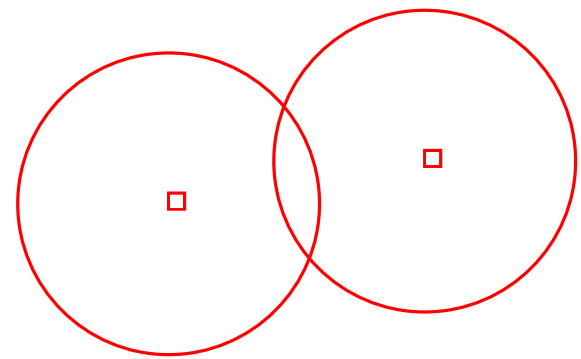
Optimization: Find a min/max value r^*
Decision: Determine $r \geq r^*$ or not for given r

Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem





[Sharir 96] $O(n \log^9 n)$ time

[Eppstein 97] $O(n \log^2 n)$ time (randomized)

[Parametric Search]
Solving optimization using decision algorithms

Optimization: Find a min/max value r^*
Decision: Determine $r \geq r^*$ or not for given r

[Sharir 96] $O(n \log^8 n)$ time

[Eppstein 97] $O(n \log^2 n)$ time

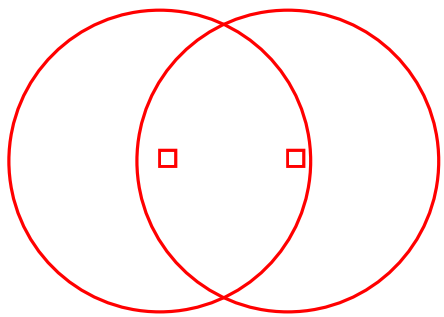
[Cole's Parametric Search 1987]
Sequential Decision in $T_S(n)$ time
Paralallel Decision in $T_P(n)$ parallel step with $P(n)$ processors.

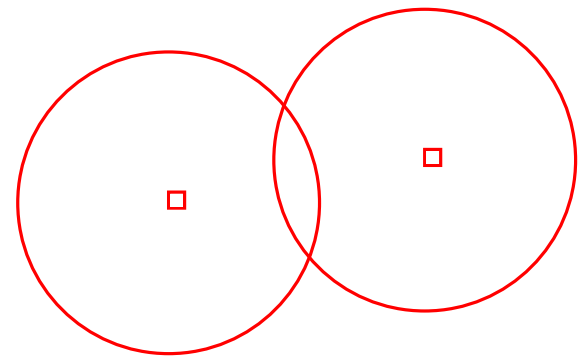
Then the optimal value r^* is returned in $O(T_S(n) \log P(n) + T_P(n)P(n))$ time.

Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem





[Sharir 96] $O(n \log^9 n)$ time

[Eppstein 97] $O(n \log^2 n)$ time (randomized)

[Wang 2020] $O(n \log n \log \log n)$ time

[Choi and Ahn 2021] $O(n \log n)$ time

[Sharir 96] $O(n \log^8 n)$ time

[Eppstein 97] $O(n \log^2 n)$ time

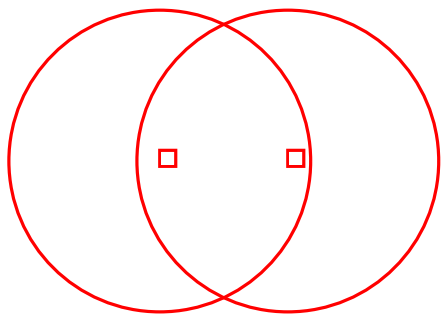
[Cole's Parametric Search 1987]
Sequential Decision in $T_S(n)$ time
Paralallel Decision in $T_P(n)$ parallel step with $P(n)$ processors.

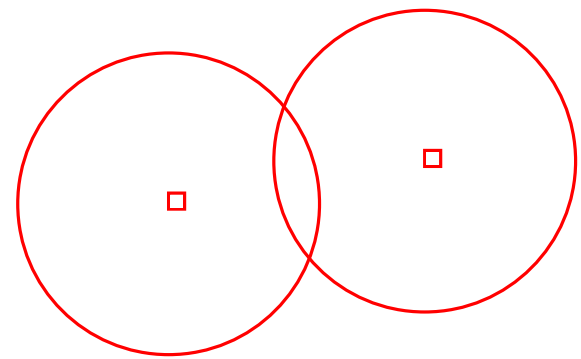
Then the optimal value r^* is returned in $O(T_S(n) \log P(n) + T_P(n)P(n))$ time.

Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem





[Sharir 96] $O(n \log^9 n)$ time

[Eppstein 97] $O(n \log^2 n)$ time (randomized)

[Wang 2020] $O(n \log n \log \log n)$ time

$\rightarrow T_S(n) = O(n)$

[Choi and Ahn 2021] $O(n \log n)$ time

$\rightarrow T_P(n) = O(\log n)$
with $P(n) = O(n)$

[Sharir 96] $O(n \log^8 n)$ time

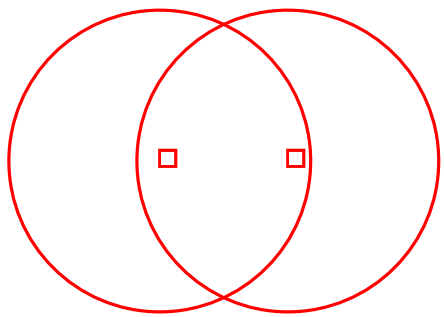
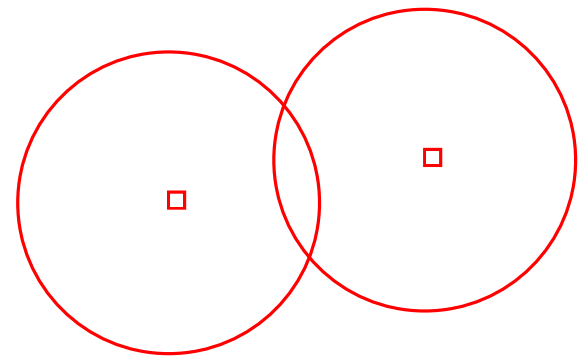
[Eppstein 97] $O(n \log^2 n)$ time

[Cole's Parametric Search 1987]
Sequential Decision in $T_S(n)$ time
Paralallel Decision in $T_P(n)$ parallel step
with $P(n)$ processors.

Then the optimal value r^* is returned in
$O(T_S(n) \log P(n) + T_P(n)P(n))$ time.

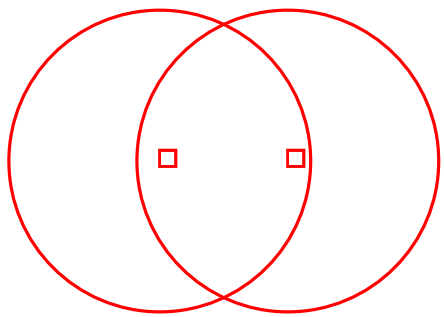
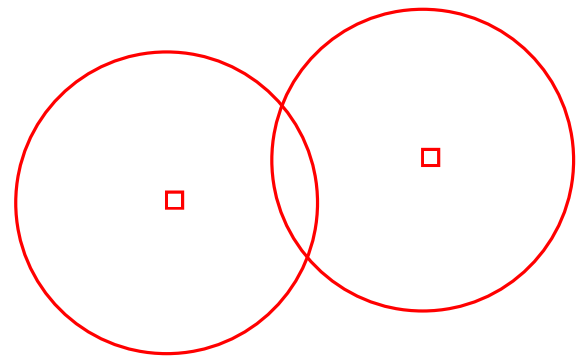
Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem
	
[Sharir 96] $O(n \log^9 n)$ time	[Sharir 96] $O(n \log^8 n)$ time
[Eppstein 97] $O(n \log^2 n)$ time (randomized)	[Eppstein 97] $O(n \log^2 n)$ time
[Wang 2020] $O(n \log n \log \log n)$ time	[COWX 24] $O(n \log n)$ time
$\rightarrow T_S(n) = O(n)$	$\rightarrow T_S(n) = O(n)$
[Choi and Ahn 2021] $O(n \log n)$ time	$T_P(n) = O(\log n)$
$\rightarrow T_P(n) = O(\log n)$ with $P(n) = O(n)$	with $P(n) = O(n)$

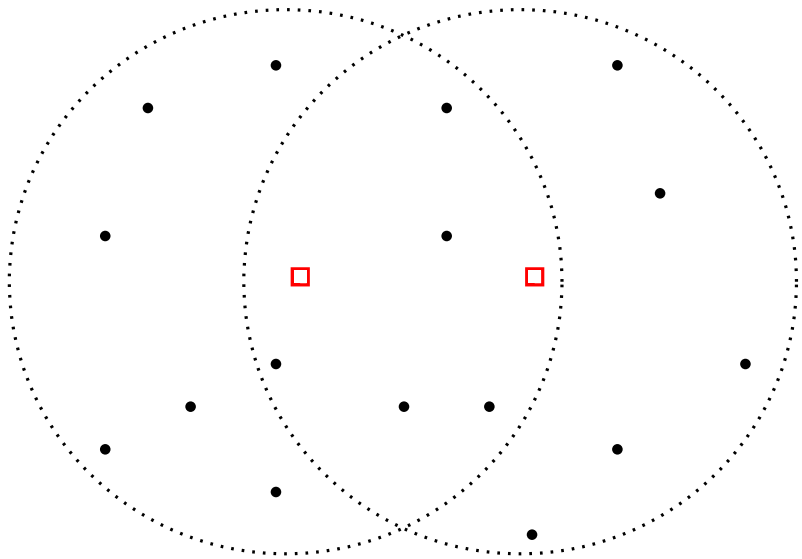
Sketch of History

[Agarwal and Sharir 94] $O(n^2 \log^3 n)$ time algorithm

Nearby 2-center problem	Distant 2-center problem
	
[Sharir 96] $O(n \log^9 n)$ time	[Sharir 96] $O(n \log^8 n)$ time
[Eppstein 97] $O(n \log^2 n)$ time (randomized)	[Eppstein 97] $O(n \log^2 n)$ time
[Wang 2020] $O(n \log n \log \log n)$ time $\rightarrow T_S(n) = O(n)$	[COWX 24] $O(n \log n)$ time $\rightarrow T_S(n) = O(n)$
[Choi and Ahn 2021] $O(n \log n)$ time $\rightarrow T_P(n) = O(\log n)$ with $P(n) = O(n)$	$T_P(n) = O(\log n)$ with $P(n) = O(n)$

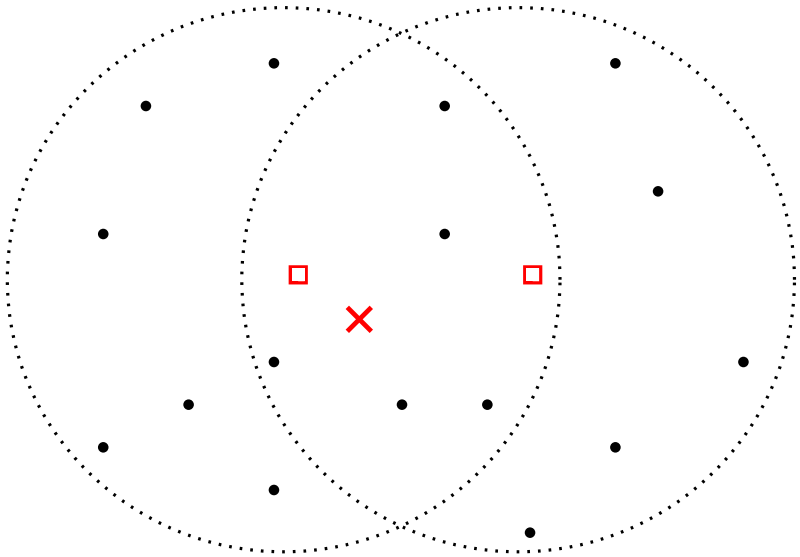
[Wang 20] Decision for Near-by-Case

Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)



[Wang 20] Decision for Near-by-Case

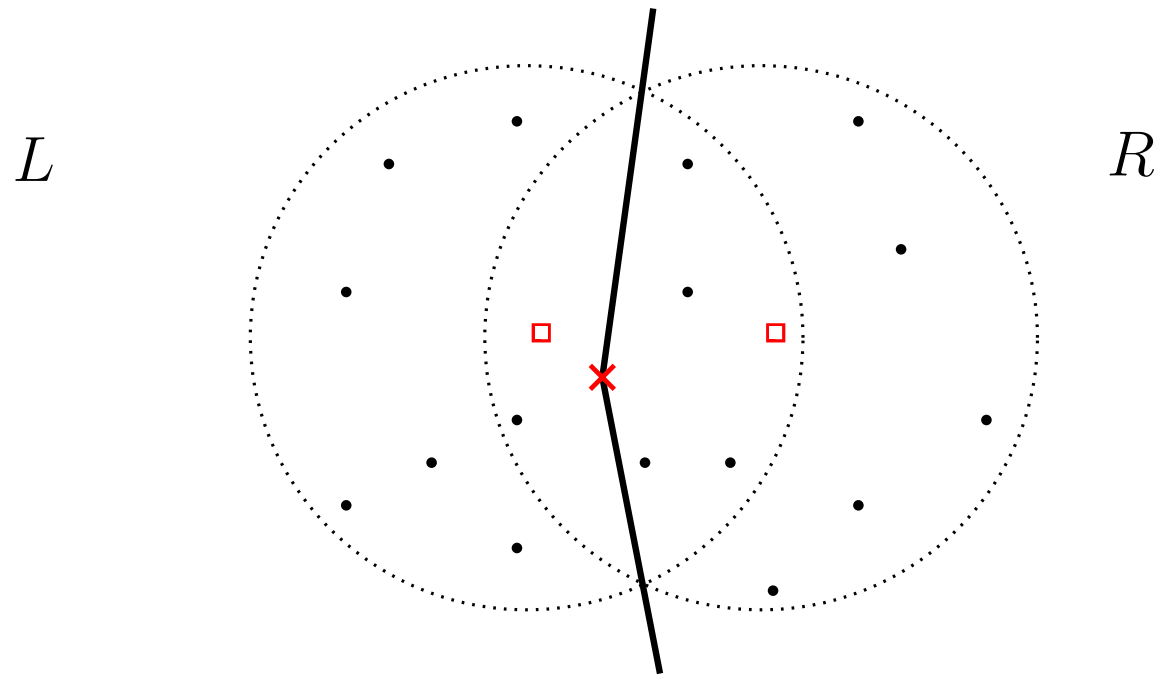
Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)



[Eppstein 97] There are $O(1)$ candidates for a point $o \in D_1 \cap D_2$ computed in $O(n)$ time.

[Wang 20] Decision for Near-by-Case

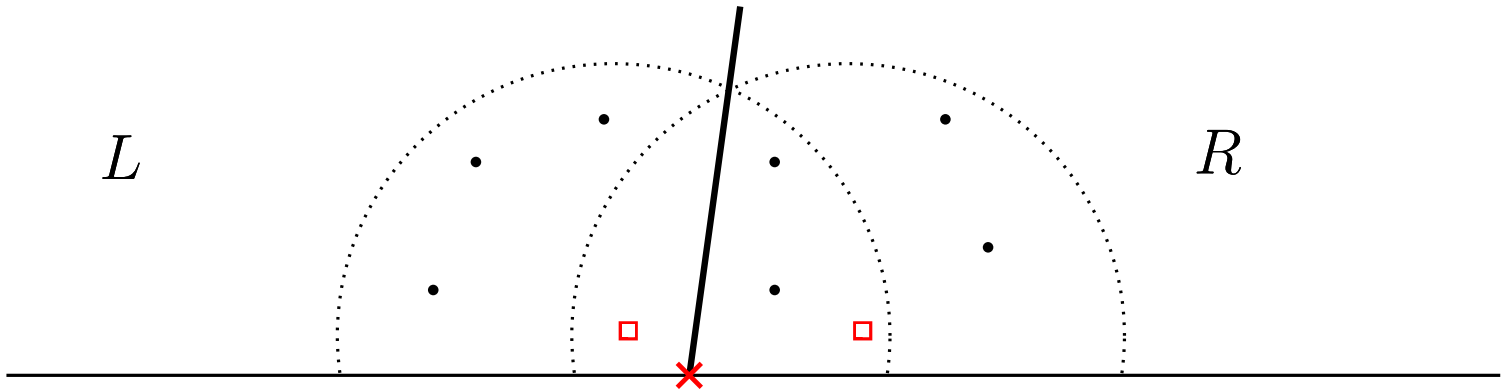
Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)



[Eppstein 97] There are $O(1)$ candidates for a point $o \in D_1 \cap D_2$ computed in $O(n)$ time.

[Wang 20] Decision for Near-by-Case

Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)

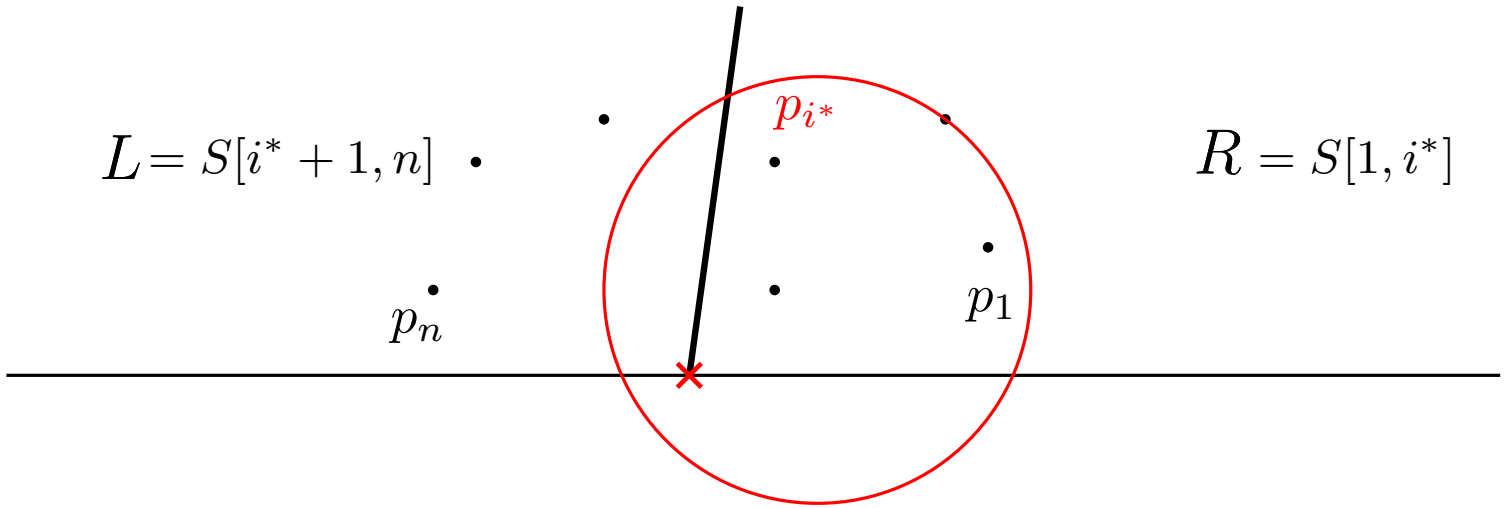


[Eppstein 97] There are $O(1)$ candidates for a point $o \in D_1 \cap D_2$ computed in $O(n)$ time.

Easy case: If all points p have $y(p) > 0$?

[Wang 20] Decision for Near-by-Case

Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)



[Eppstein 97] There are $O(1)$ candidates for a point $o \in D_1 \cap D_2$ computed in $O(n)$ time.

Easy case: If all points p have $y(p) > 0$?

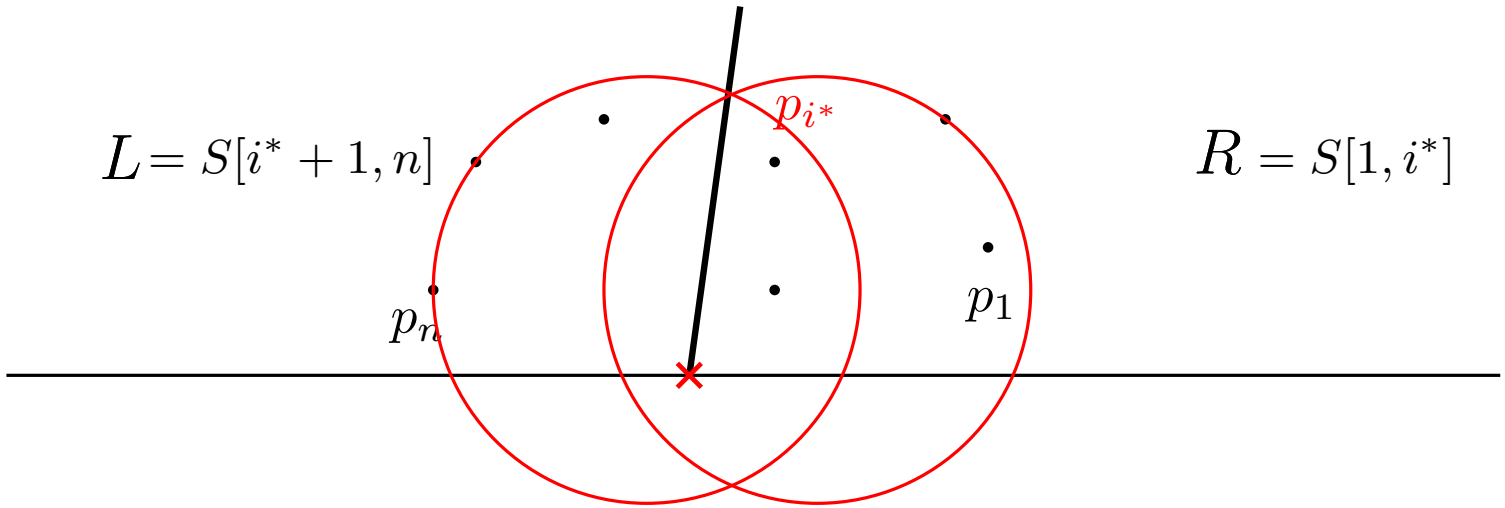
Angular sorting $S = \{p_1, \ldots, p_n\}$ (preprocessing)

Line sweeping:
Find the maximum i^* s.t. $S[1 : i^*]$ is covered by a disk of radius r

Check $S[i^* + 1 : n]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case

Decision: For a given r,decide whether $r^* \leq r$ or not (after $O(n \log n)$ preprocessing time)



[Eppstein 97] There are $O(1)$ candidates for a point $o \in D_1 \cap D_2$ computed in $O(n)$ time.

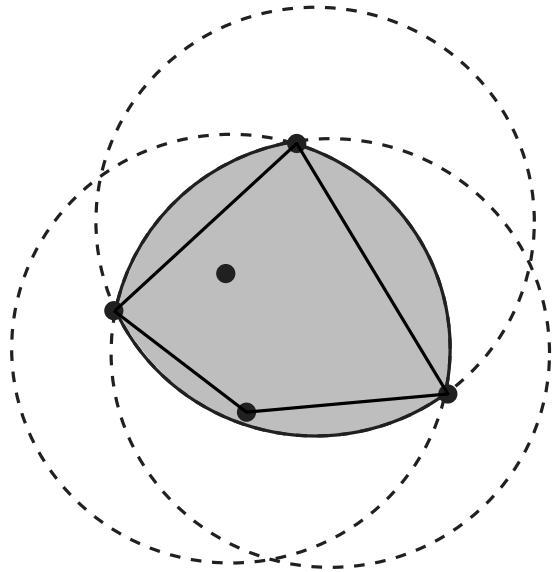
Easy case: If all points p have $y(p) > 0$?

Angular sorting $S = \{p_1, \ldots, p_n\}$ (preprocessing)

Line sweeping:
Find the maximum i^* s.t. $S[1 : i^*]$ is covered by a disk of radius r

Check $S[i^* + 1 : n]$ is covered by a disk of radius r

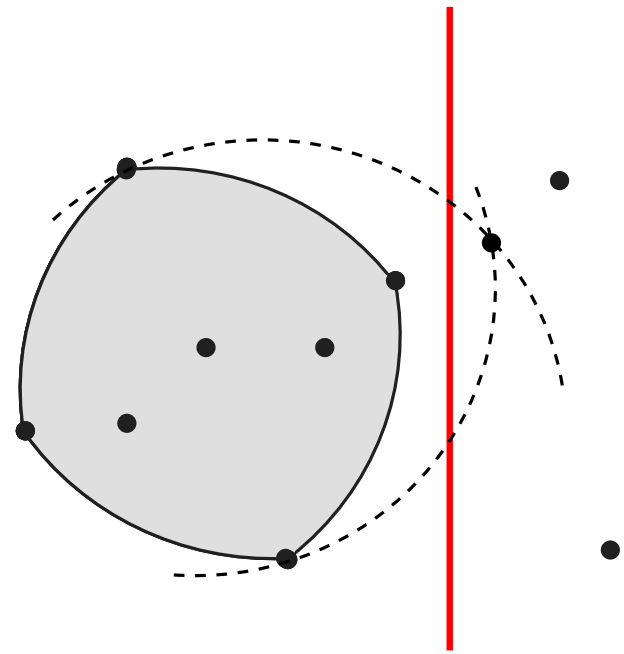
[Wang 20] Dynamic r-Circular Hull



convex hull $\mathrm{CH}(S)$: intersection of all convex sets containing S

r-**circular hull** $\alpha_r(S)$: intersection of **all disks of radius** r containing S

[Wang 20] Dynamic r-Circular Hull



convex hull $\mathrm{CH}(S)$: intersection of all convex sets containing S

- If points are sorted, $\mathrm{CH}(S)$ can be computed in $O(n)$ time.
  $\rightarrow$ a line sweeping algorithm updates $\mathrm{CH}(S[1:i])$ in $O(1)$ amortized time.

r-**circular hull** $\alpha_r(S)$: intersection of **all disks of radius** r containing S

- If points are sorted, $\alpha_r(S)$ can be computed in $O(n)$ time.
  $\rightarrow$ a line sweeping algorithm updates $\alpha_r(S[1:i])$ in $O(1)$ amortized time.

[Wang 20] Dynamic r-Circular Hull : Insertion

Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

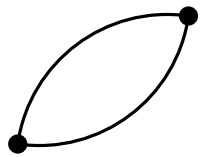
Insert the right most point to S

[Wang 20] Dynamic r-Circular Hull : Insertion

Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

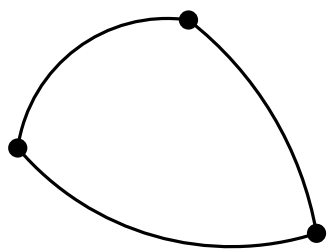
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

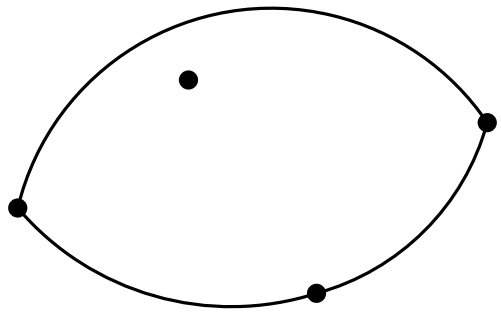
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

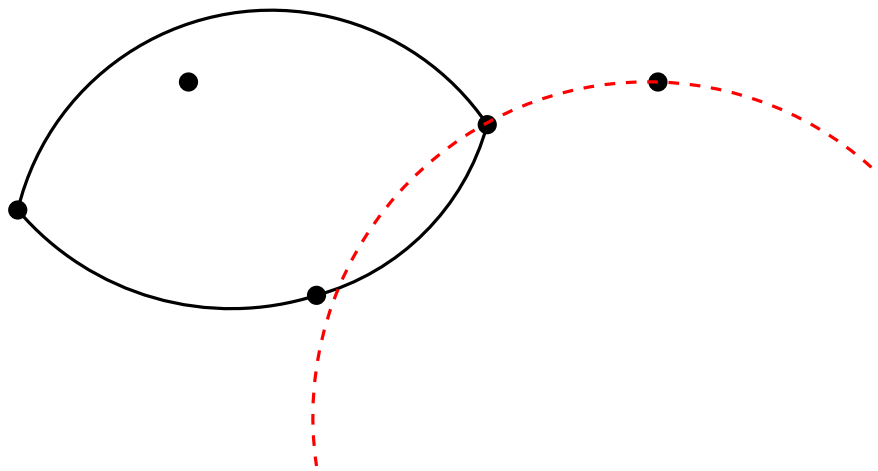
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

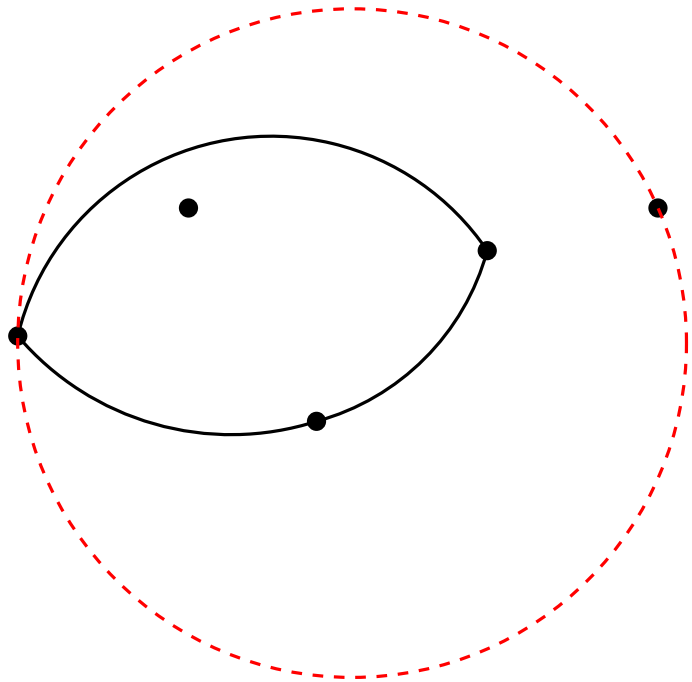
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

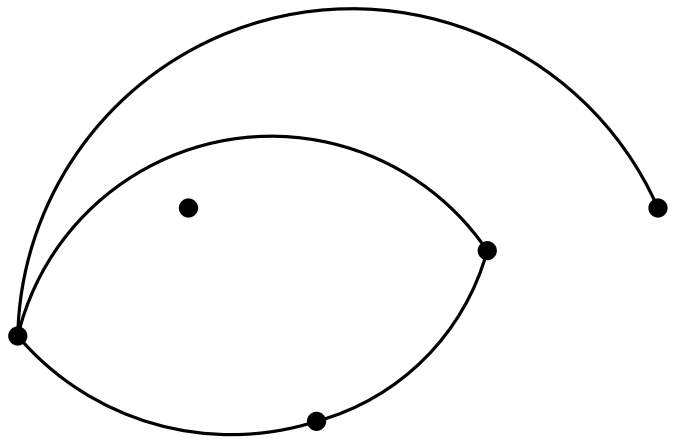
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

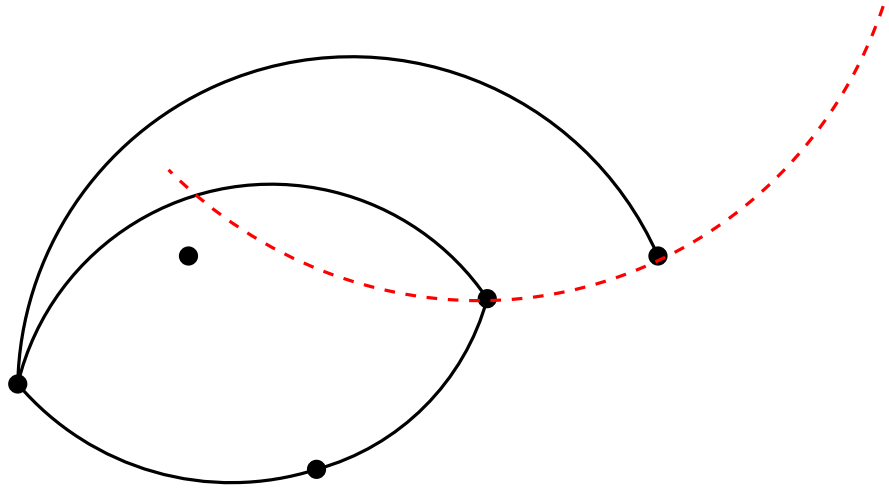
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

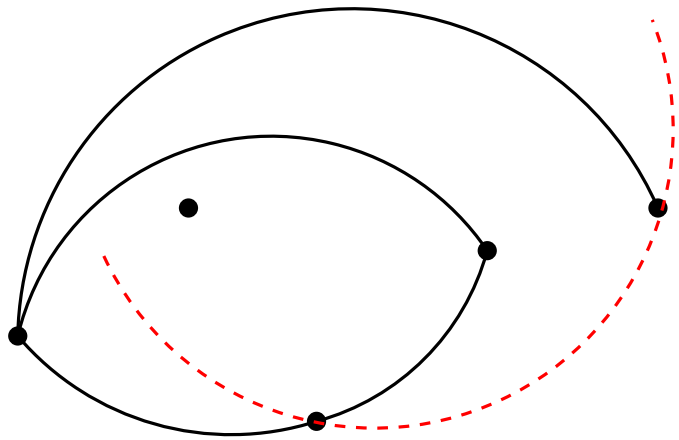
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

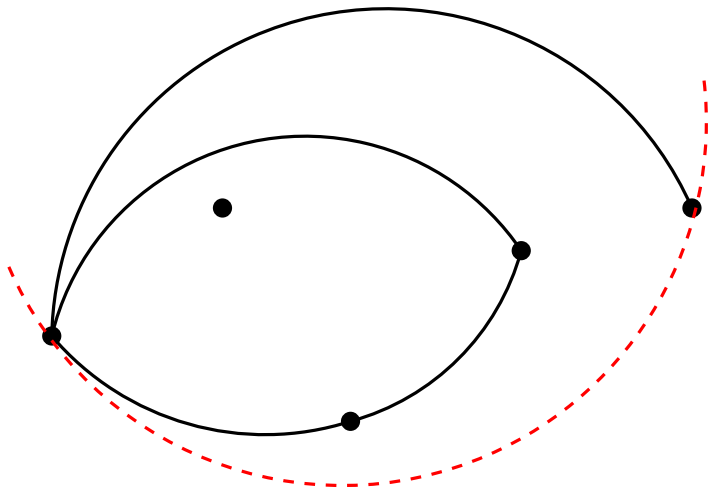
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

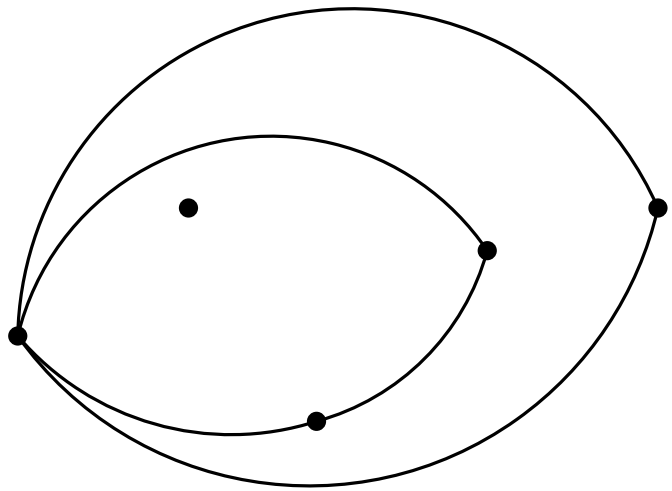
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

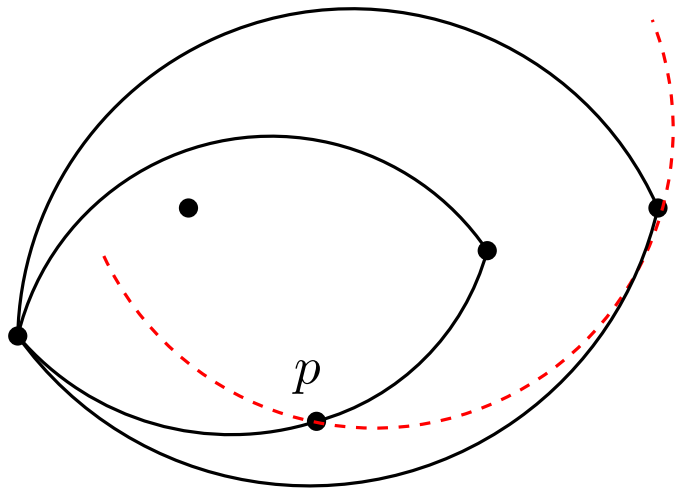
[Wang 20] Dynamic r-Circular Hull : Insertion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

[Wang 20] Dynamic r-Circular Hull : Insertion



- p is a vertex of $\alpha_r(S)$
- p is lying inside $\alpha_r(S)$

  $\rightarrow$ p cannot be a vertex or boundary of α_r

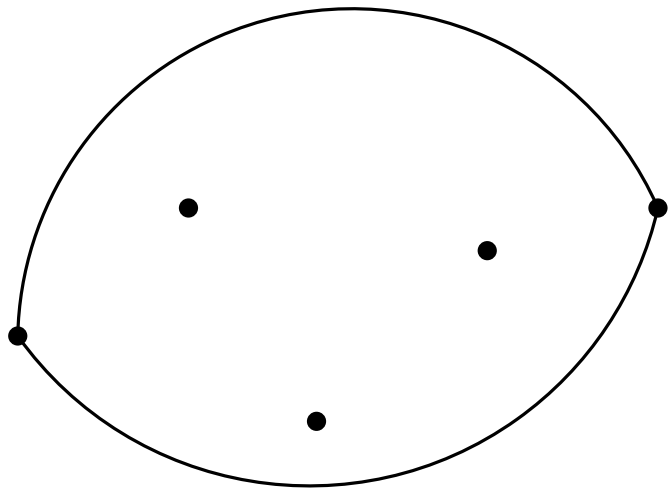
Inserting n vertices to S takes $O(n)$ time in total.

$\rightarrow$ amortized $O(1)$ update time

Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: $S = \emptyset$

Insert the right most point to S

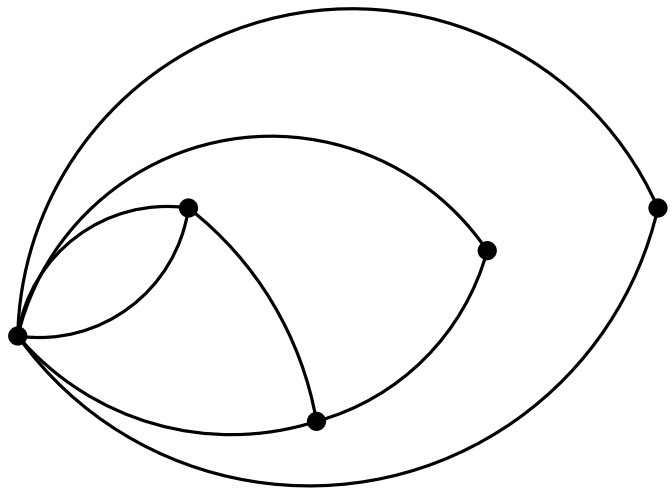
[Wang 20] Dynamic r-Circular Hull : Deletion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S

Delete the right most point to S

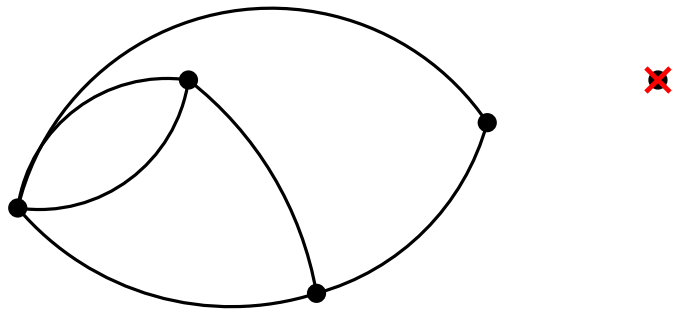
[Wang 20] Dynamic r-Circular Hull : Deletion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Delete the right most point to S

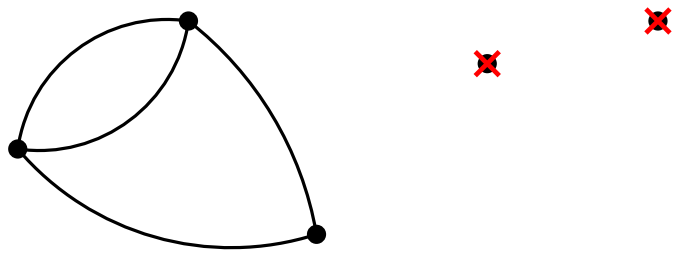
[Wang 20] Dynamic r-Circular Hull : Deletion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Delete the right most point to S

[Wang 20] Dynamic r-Circular Hull : Deletion



Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Delete the right most point to S

[Wang 20] Dynamic r-Circular Hull : Deletion

Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Delete the right most point to S

[Wang 20] Dynamic r-Circular Hull : Deletion

Initializing $O(n)$ time

Deleting a point takes $O(1)$ time.

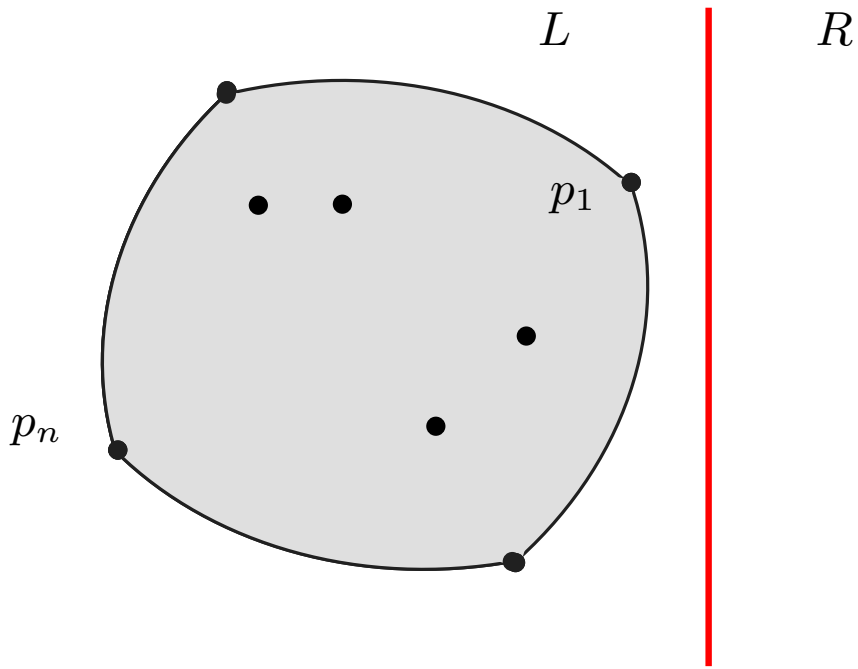
$\rightarrow$ amortized $O(1)$ update time

Dynamic r-Circular Hull: Maintaining $\alpha_r(S)$
Initial: S

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

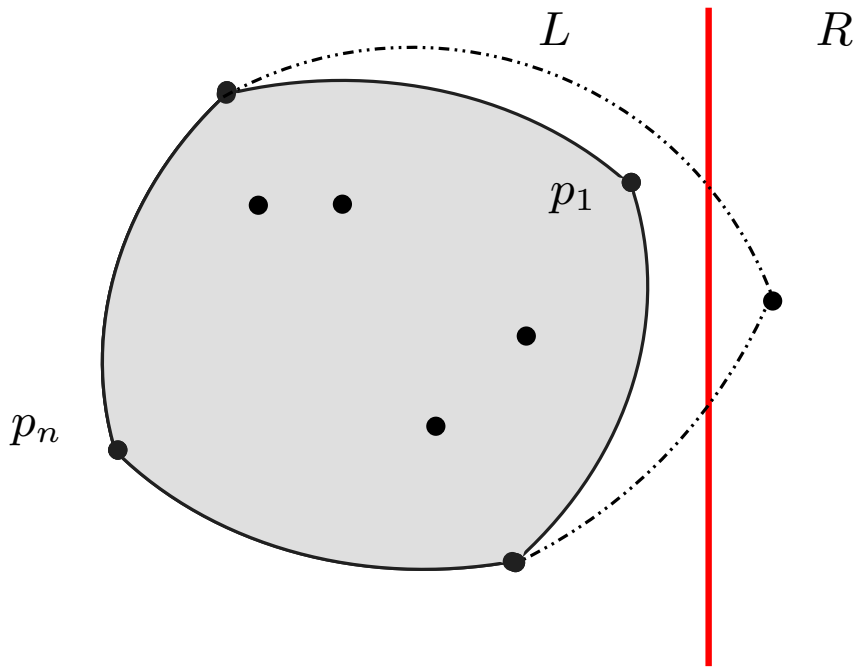
Delete the right most point to S

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

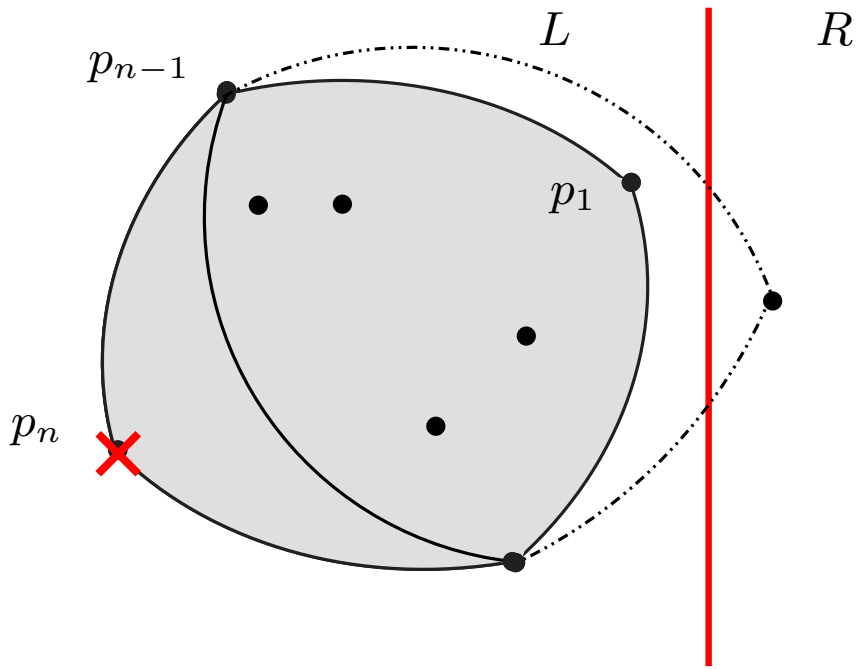
[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Insert the right most point q to R

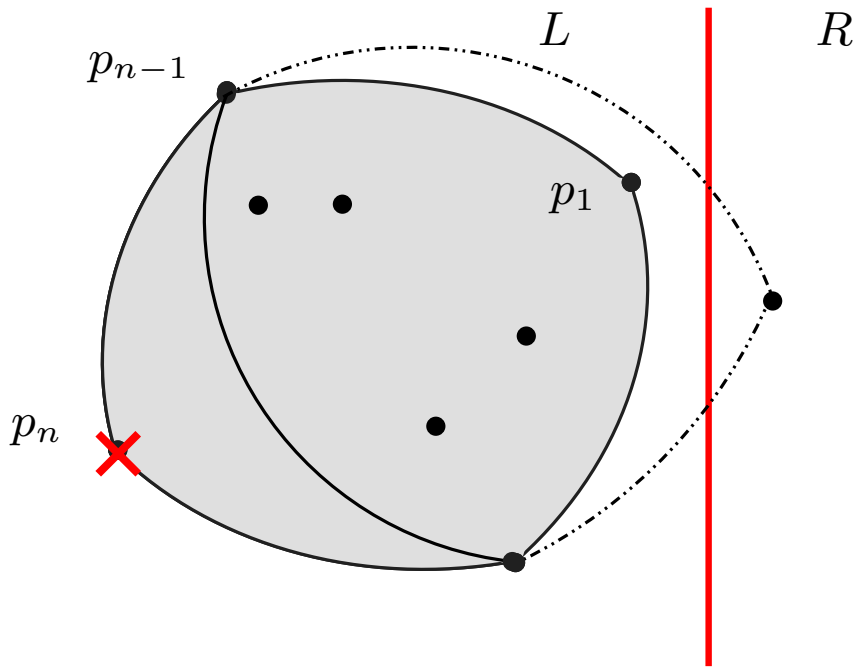
[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$
We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Insert the right most point q to R
Delete the left most point p from L

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

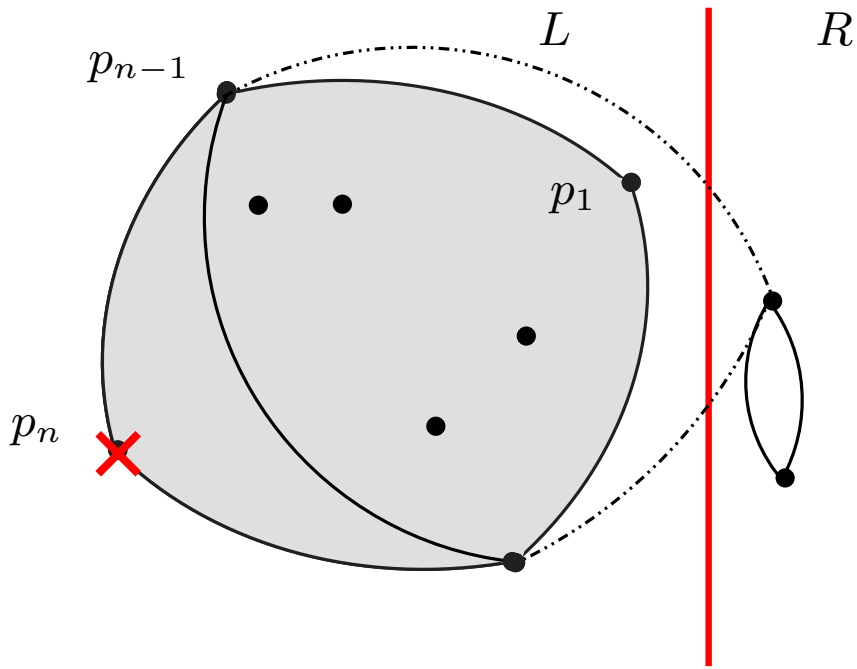
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

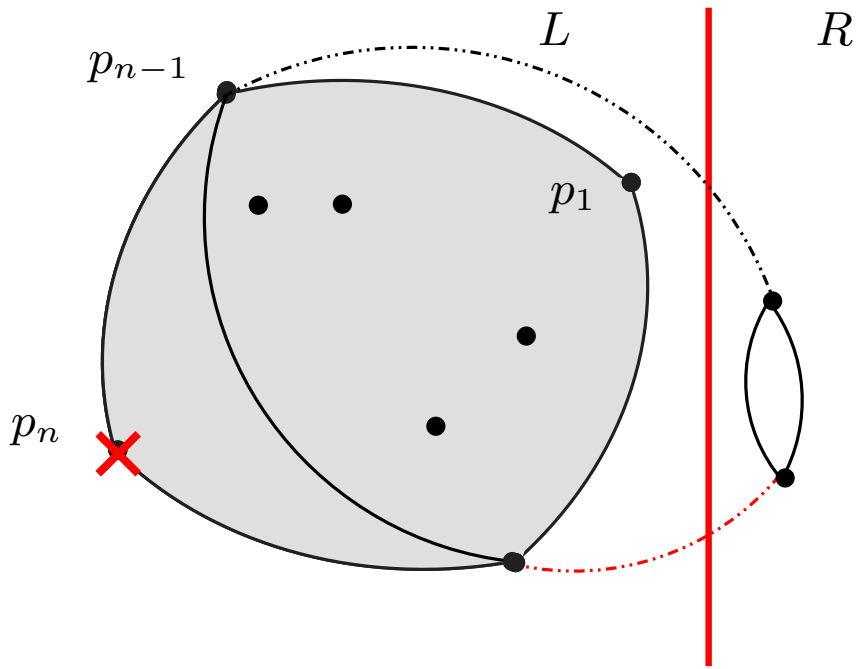
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

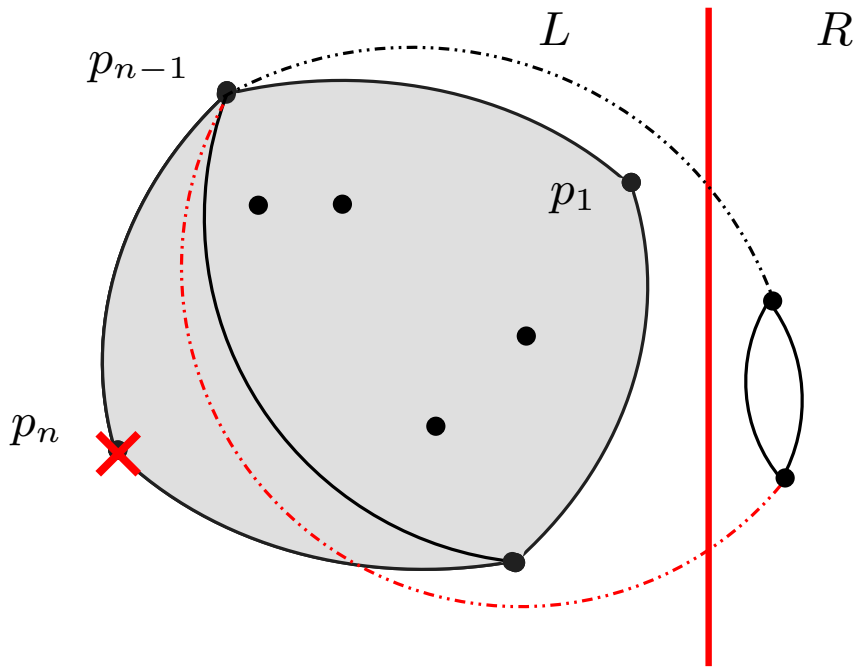
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

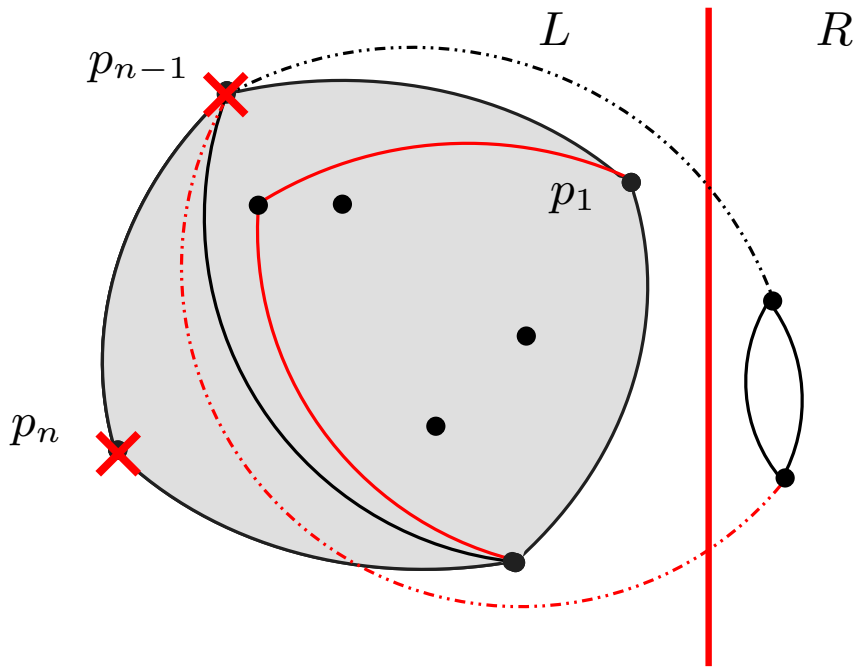
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

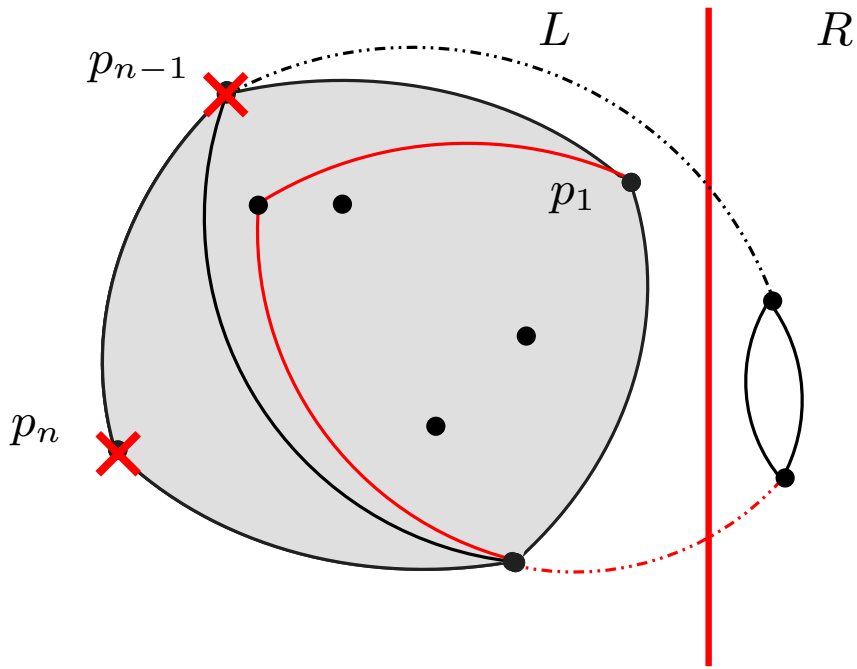
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

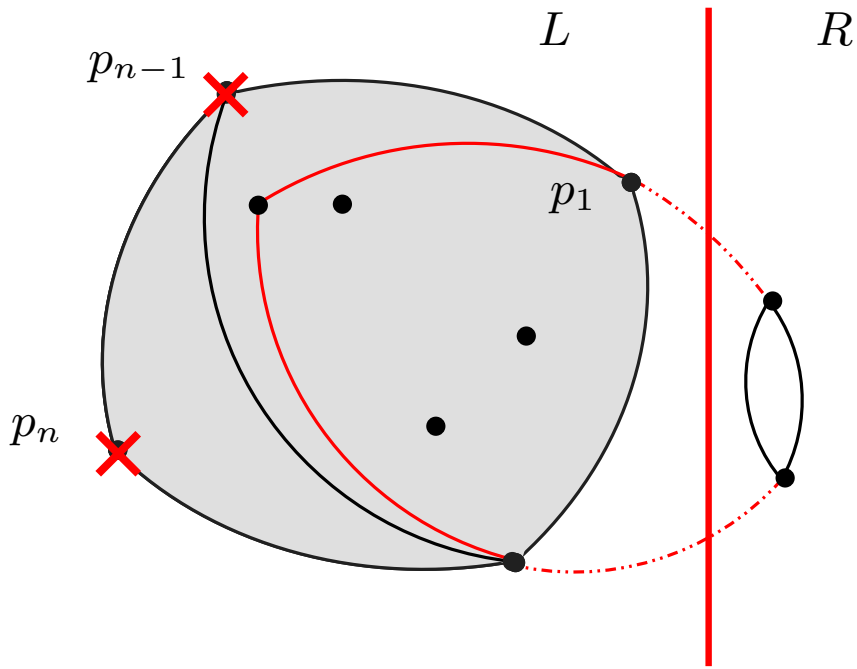
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

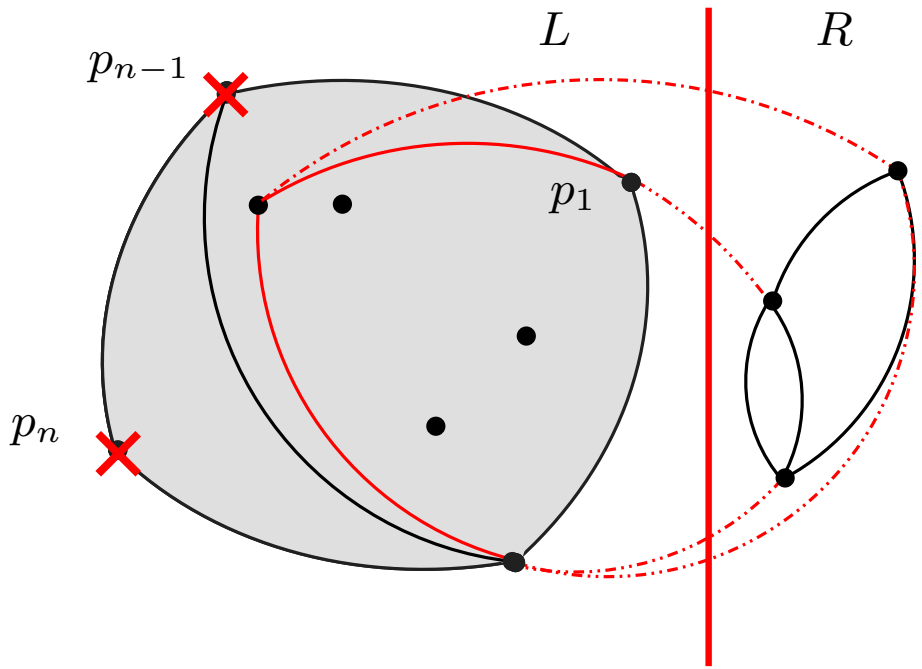
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Fully Dynamic r-Circular Hull



Dynamic r-Circular Hull: Maintaining $\alpha_r(L \cup R)$
Initial: $L = S$ and $R = \emptyset$

We store $\alpha_r(S[1:1])$, $\alpha_r(S[1:2]), \ldots, \alpha_r(S[1:n])$

Initializing takes $O(n)$ time.

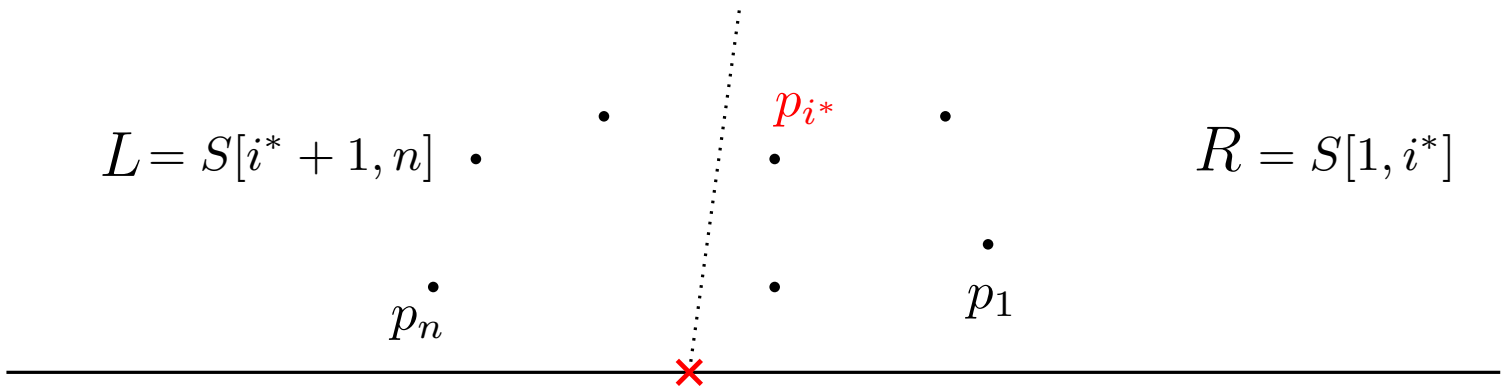
Insert the right most point q to R
Delete the left most point p from L

updating takes $O(1)$ amortized time.

Inserting m vertices to R and removing n vertices from L takes $O(n+m)$ time in total.

$\rightarrow$ updating takes $O(1)$ amortized time.

[Wang 20] Decision for Near-by-Case



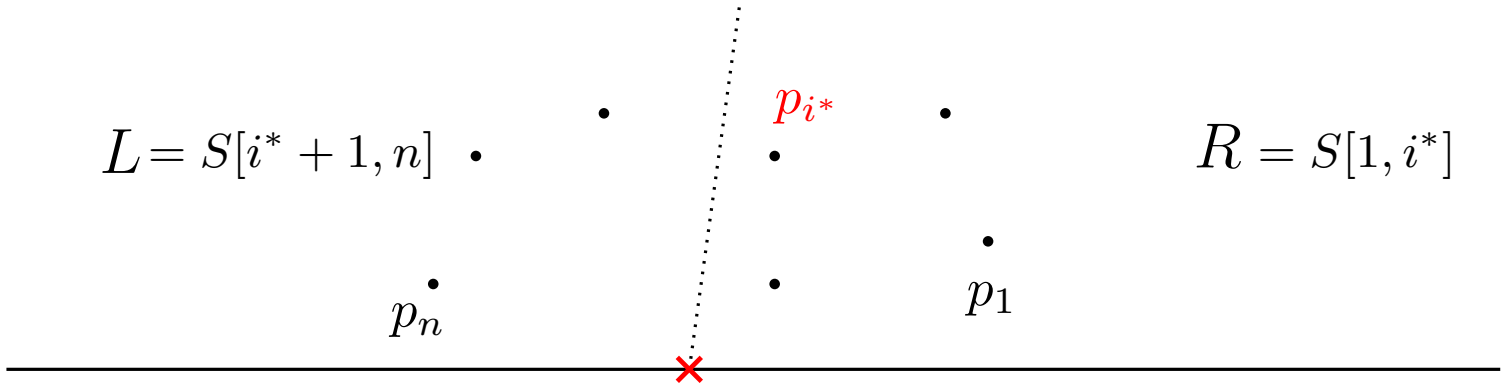
Easy case: If all points p have $y(p) > 0$?

Angular sorting $S = \{p_1, \ldots, p_n\}$ (preprocessing)

Line sweeping:
Find the maximum i^* s.t. $S[1 : i^*]$ is covered by a disk of radius r

Check $S[i^* + 1 : n]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



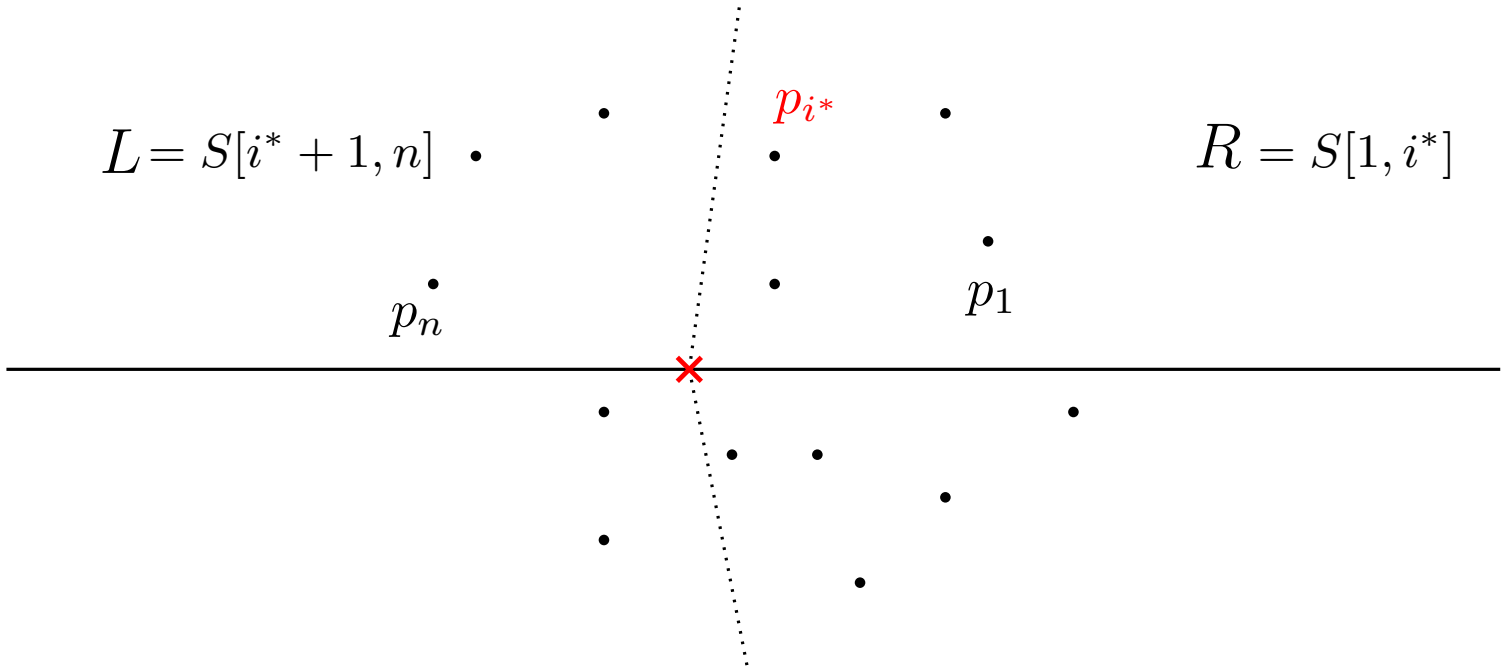
Easy case: If all points p have $y(p) > 0$?

Angular sorting $S = \{p_1, \ldots, p_n\}$ (preprocessing)

Line sweeping:
Find the maximum i^* s.t. $\alpha_r(S[1 : i^*])$ is not empty

Check $S[i^* + 1 : n]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



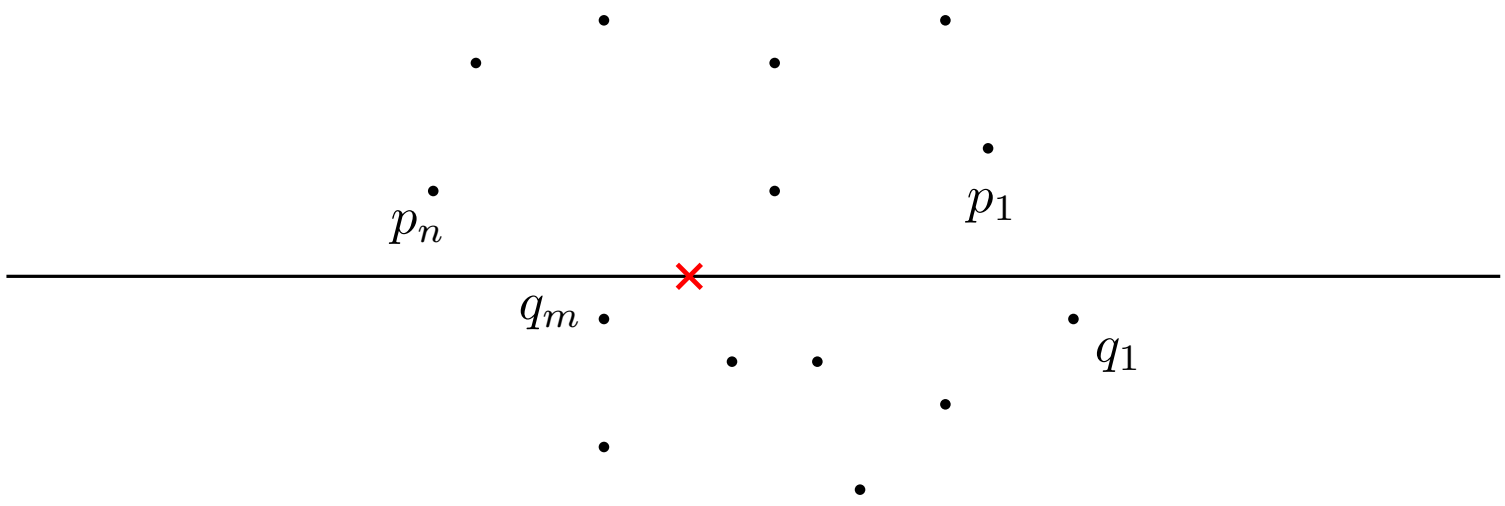
General case ?

Angular sorting $S = \{p_1, \ldots, p_n\}$ (preprocessing)

Line sweeping:
Find the maximum i^* s.t. $\alpha_r(S[1 : i^*])$ is not empty

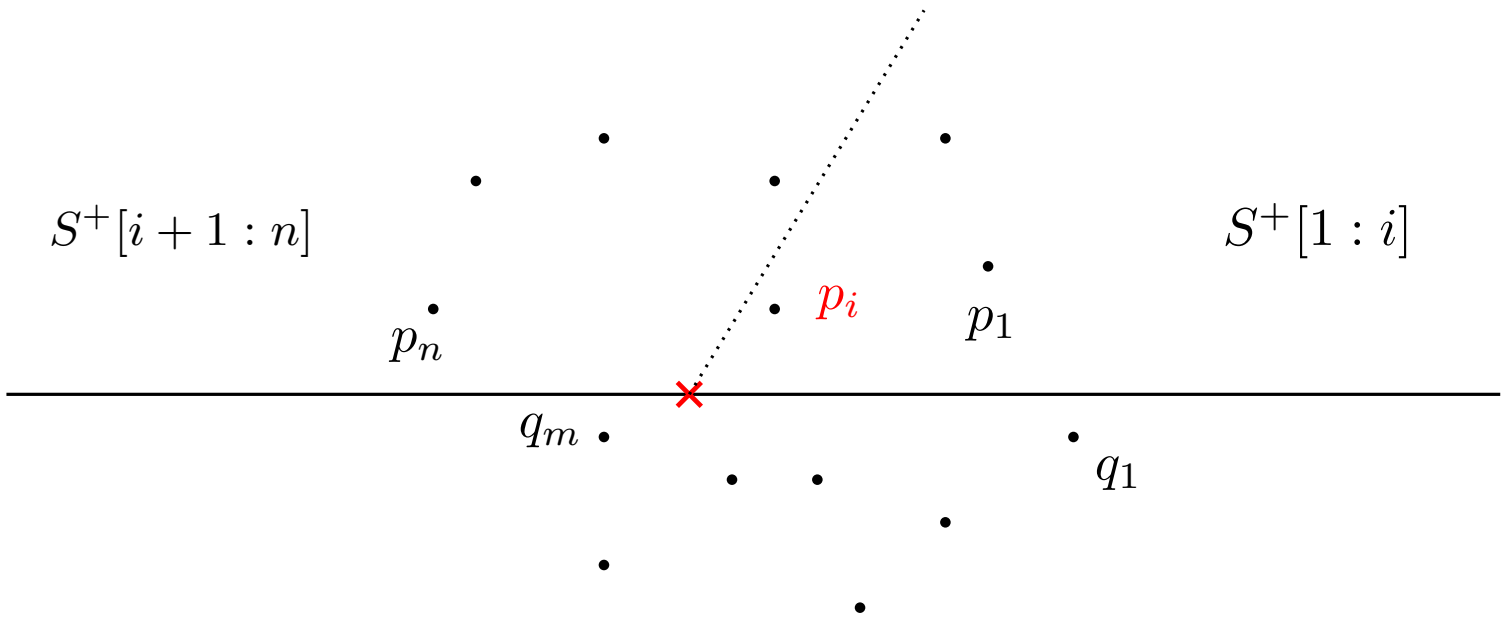
Check $S[i^* + 1 : n]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

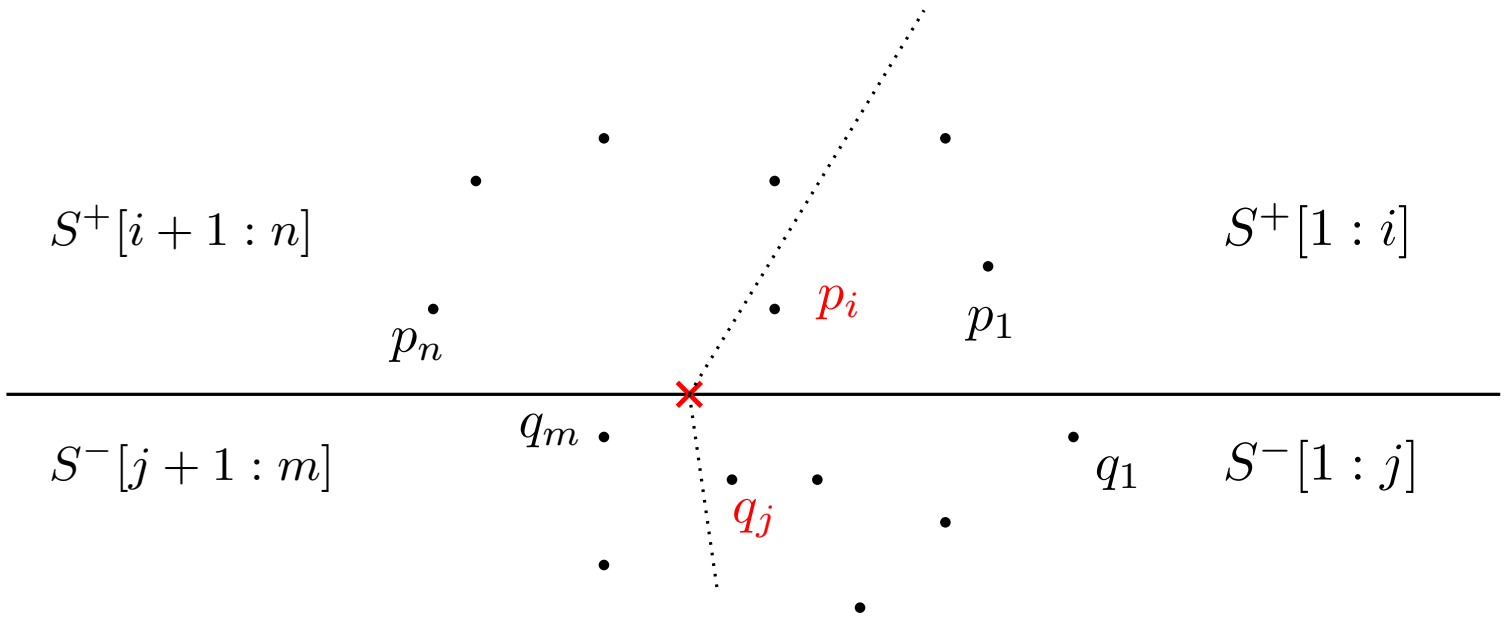
[Wang 20] Decision for Near-by-Case



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Fixing i

[Wang 20] Decision for Near-by-Case



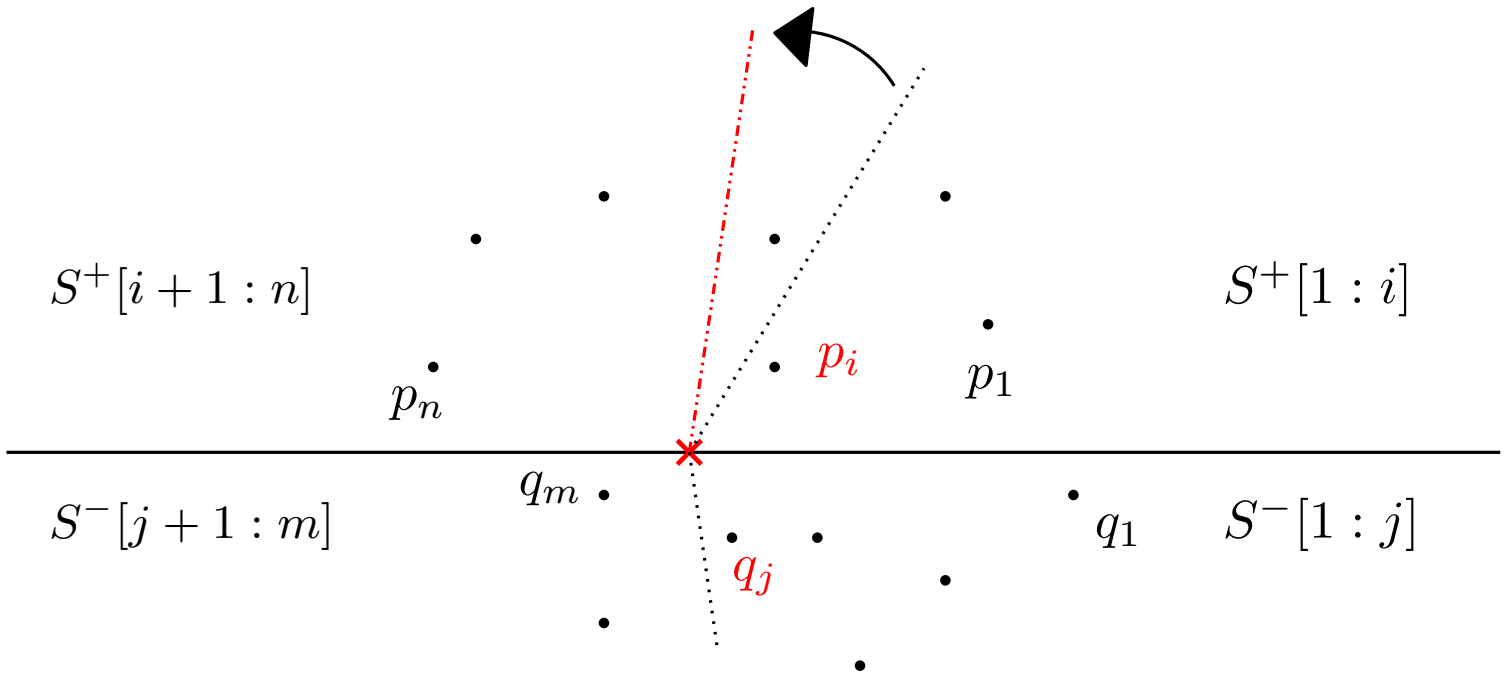
Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Fixing i

Find the maximum j s.t. $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



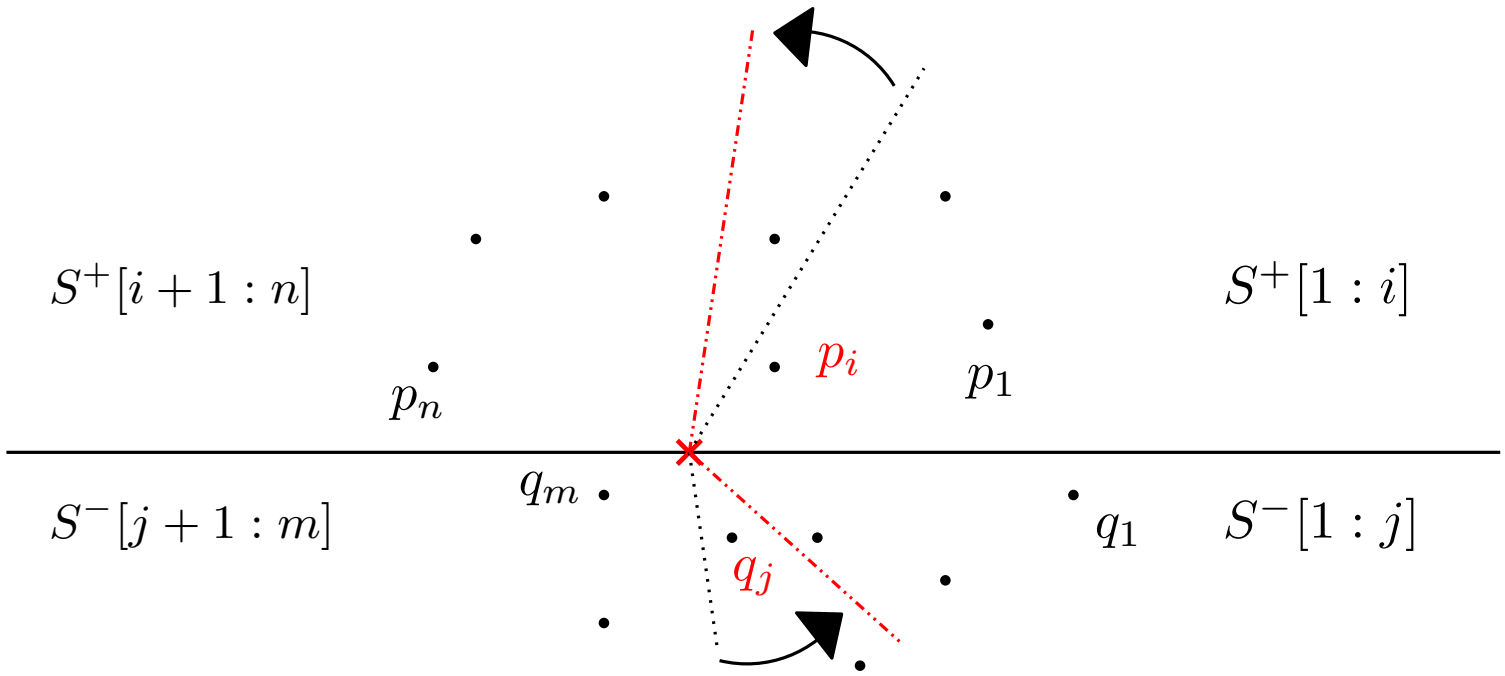
Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Fixing i

Find the maximum j s.t. $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



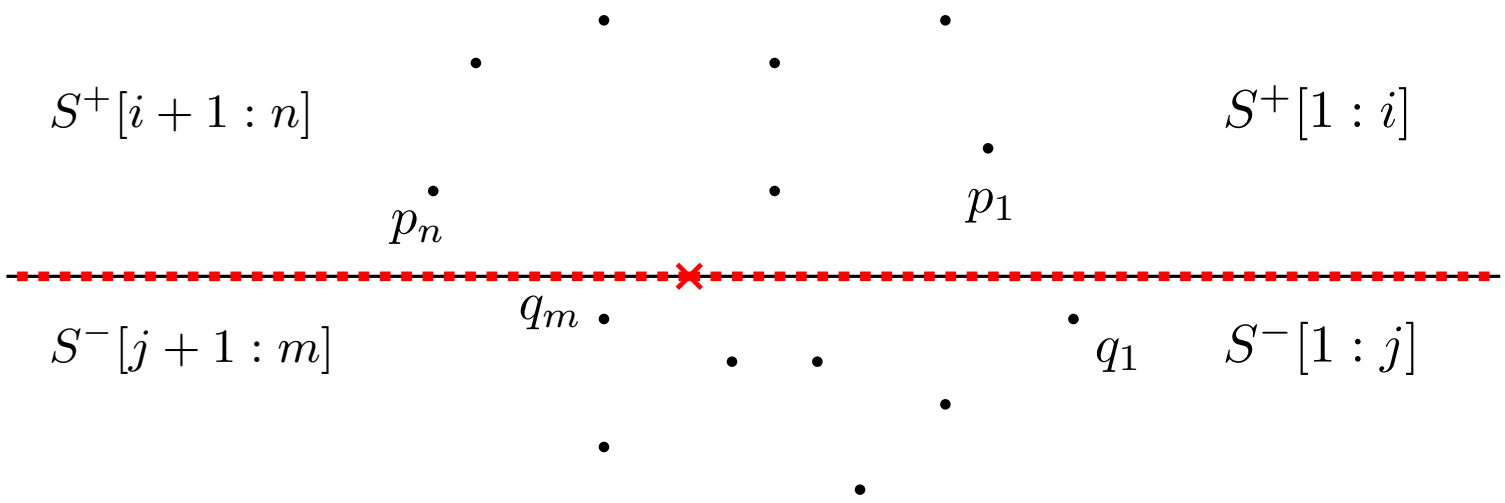
Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Fixing i

Find the maximum j s.t. $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

[Wang 20] Decision for Near-by-Case



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Starting from $i = 0$ and $j = n$

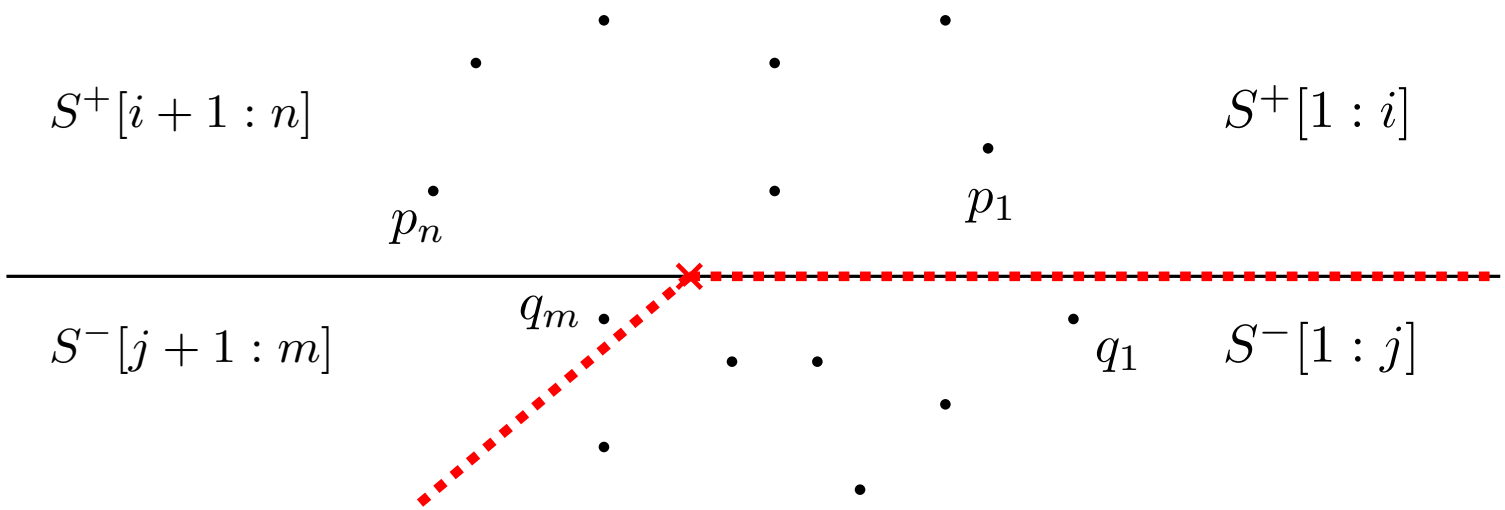
Check $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

If not, $j \leftarrow j - 1$

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

If not, $i \leftarrow i + 1$

[Wang 20] Decision for Near-by-Case



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Starting from $i = 0$ and $j = n$

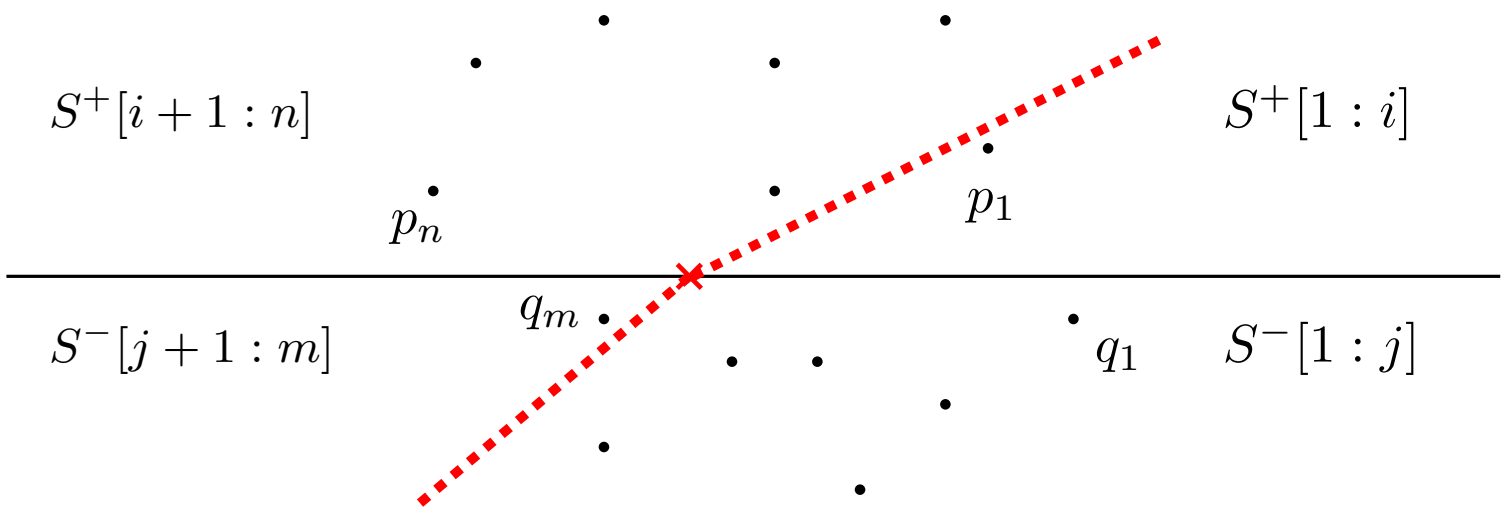
Check $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

If not, $j \leftarrow j - 1$

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

If not, $i \leftarrow i + 1$

[Wang 20] Decision for Near-by-Case



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Starting from $i = 0$ and $j = n$

Check $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

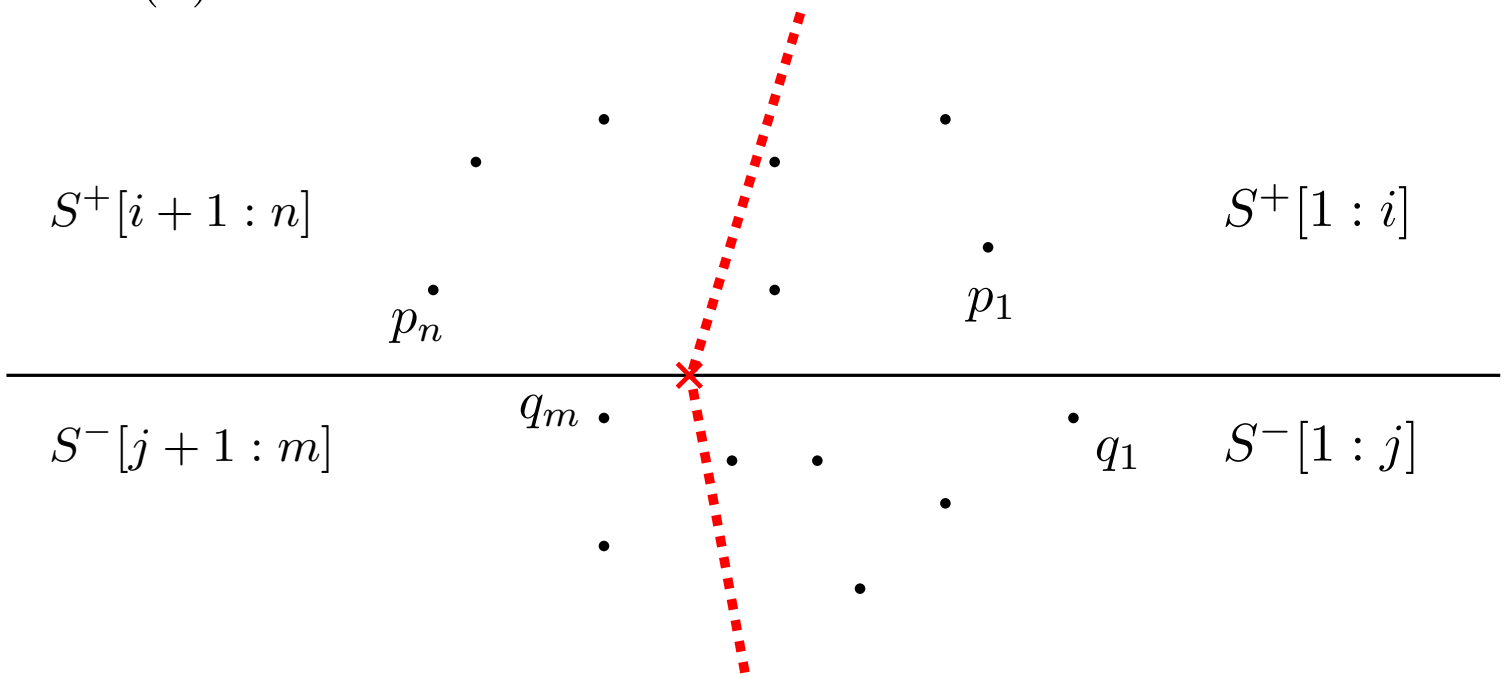
If not, $j \leftarrow j - 1$

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

If not, $i \leftarrow i + 1$

[Wang 20] Decision for Near-by-Case

Decision takes $O(n)$ time



Angular sorting $S^+ = \{p_1, \ldots, p_n\}$ and $S^- = \{q_1, \ldots, q_m\}$ (preprocessing)

Starting from $i = 0$ and $j = n$

Check $S^+[1:i] \cup S^-[1:j]$ is covered by a disk of radius r

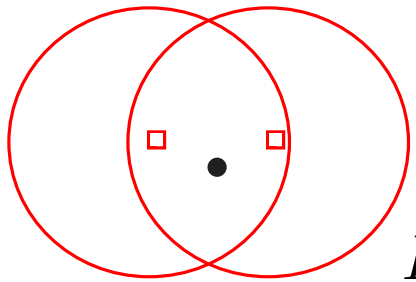
If not, $j \leftarrow j - 1$

Check $S^+[i+1:n] \cup S^-[j+1:m]$ is covered by a disk of radius r

If not, $i \leftarrow i + 1$

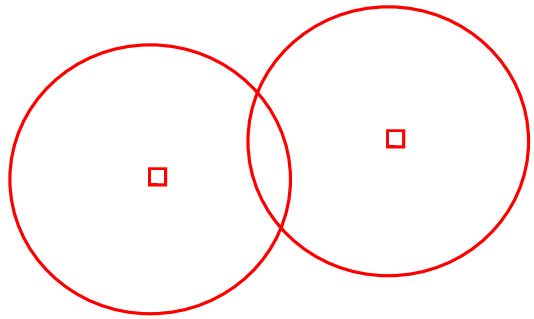
[COWX 24] Approach

Nearby 2-center problem	Distant 2-center problem



$D_1 \cap D_2 \ni p$

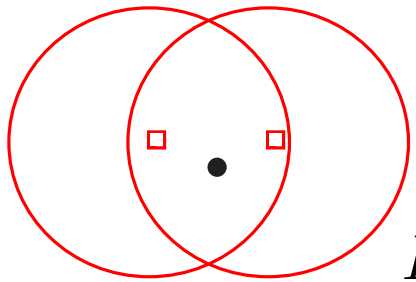
[Choi and Ahn 2021] $O(n \log n)$ time



[Eppstein 1997] $O(n \log^2 n)$ time

[COWX 24] Approach

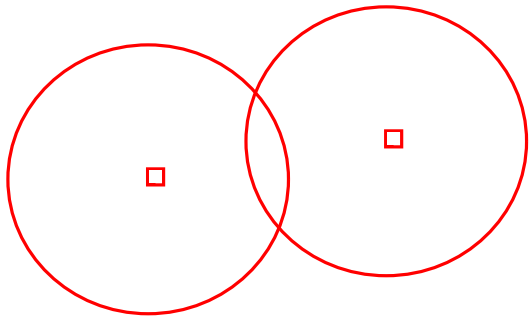
Nearby 2-center problem



$D_1 \cap D_2 \ni p$

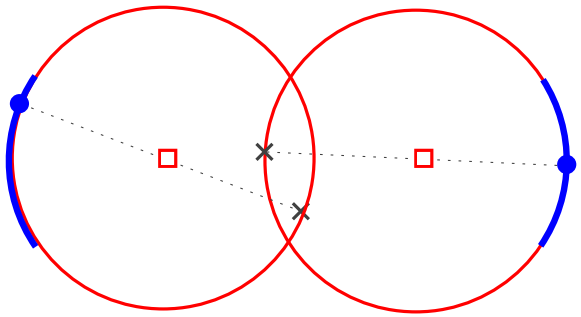
[Choi and Ahn 2021] $O(n \log n)$ time

Distant 2-center problem



[Eppstein 1997] $O(n \log^2 n)$ time

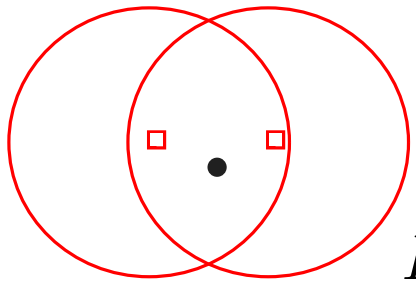
Antipodal case



$\partial(D_1)$ (and $\partial(D_2)$) has a point so that its **antipodal point** lies in $D_1 \cap D_2$

[COWX 24] Approach

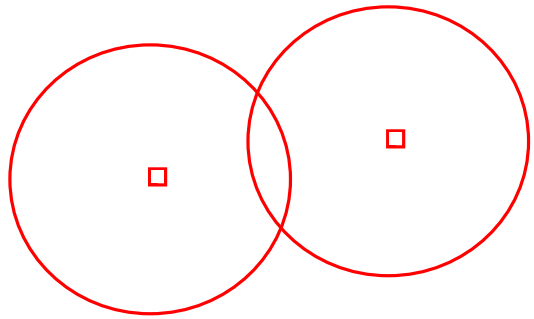
Nearby 2-center problem



$D_1 \cap D_2 \ni p$

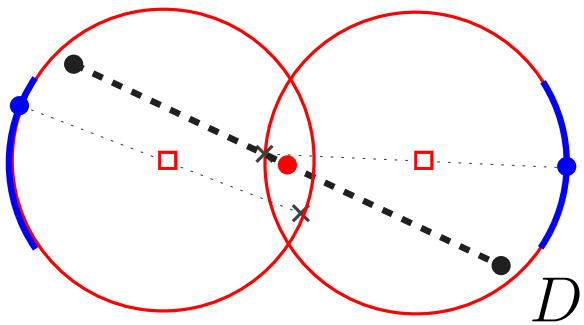
[Choi and Ahn 2021] $O(n \log n)$ time

Distant 2-center problem



[Eppstein 1997] $O(n \log^2 n)$ time

Antipodal case

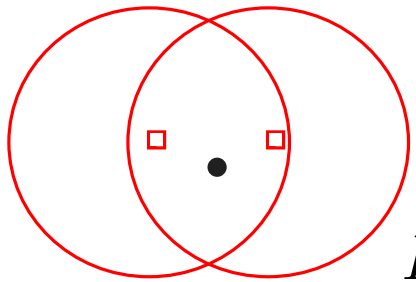


$D_1 \cap D_2 \ni o$

$\partial(D_1)$ (and $\partial(D_2)$) has a point so that its **antipodal point** lies in $D_1 \cap D_2$

[COWX 24] Approach

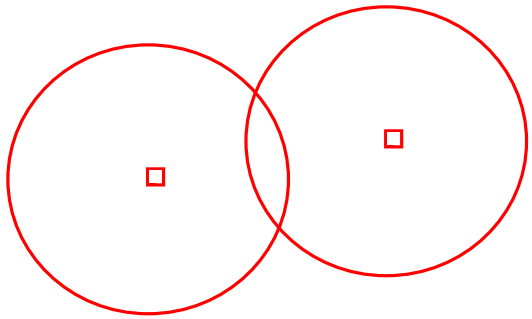
Nearby 2-center problem



$D_1 \cap D_2 \ni p$

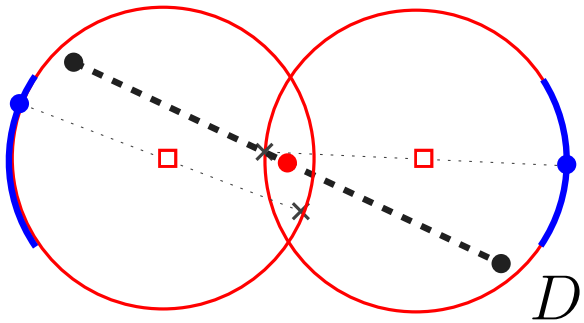
[Choi and Ahn 2021] $O(n \log n)$ time

Distant 2-center problem



[Eppstein 1997] $O(n \log^2 n)$ time

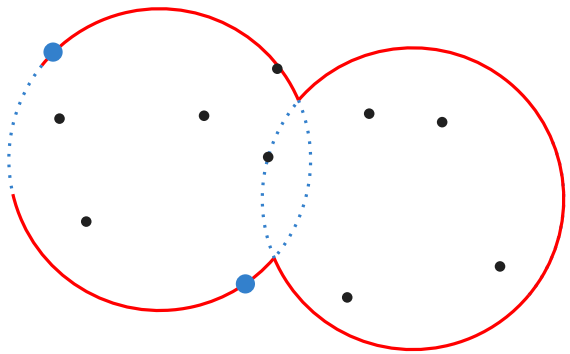
Antipodal case



$D_1 \cap D_2 \ni o$

$\partial(D_1)$ (and $\partial(D_2)$) has a point so that its **antipodal point** lies in $D_1 \cap D_2$

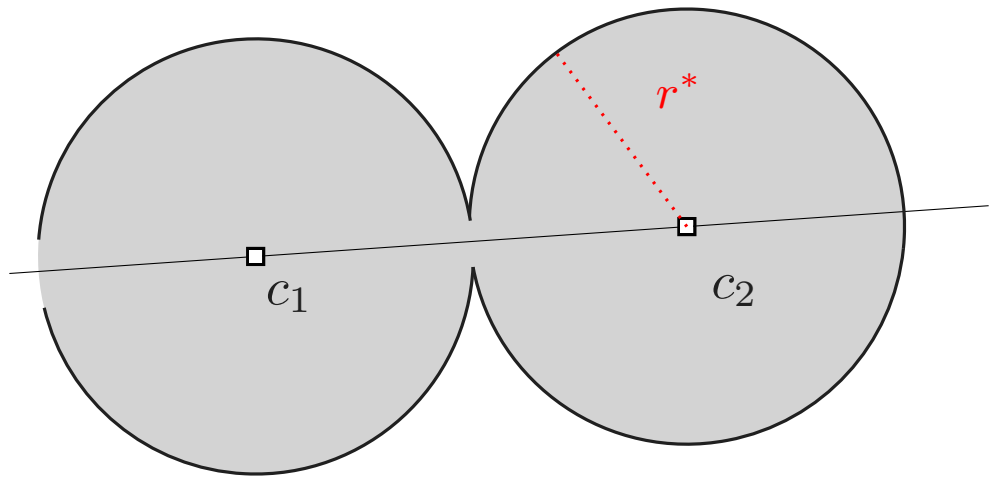
Otherwise?



[COWX 24] Non-Antipodal Case

$\partial(D_1)$ has **no** point so that its **antipodal point** lies in $D_1 \cap D_2$

A **tight solution** $\{D_1, D_2\}$: smallest disks covering P while minimizing $|c_1 c_2|$

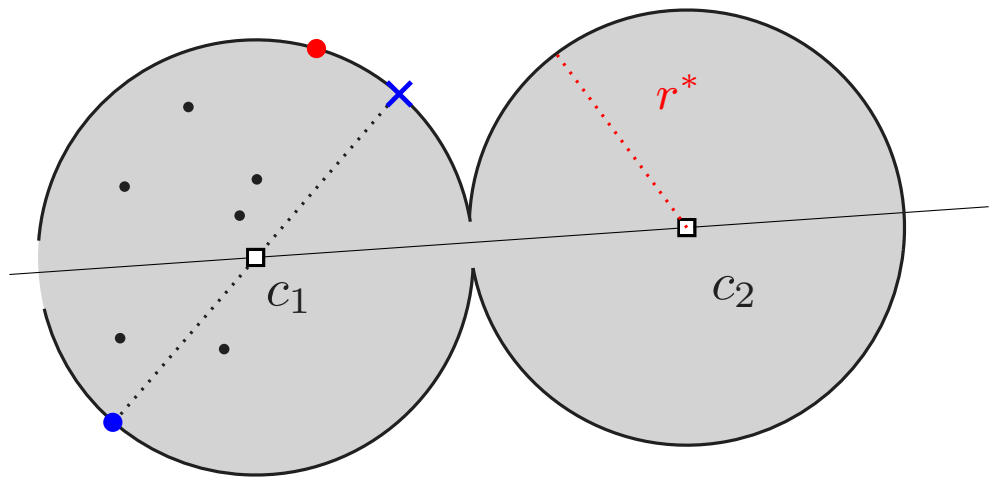


P: n points in $\mathbb{R}^2$

[COWX 24] Non-Antipodal Case

$\partial(D_1)$ has **no** point so that its **antipodal point** lies in $D_1 \cap D_2$

A **tight solution** $\{D_1, D_2\}$: smallest disks covering P while minimizing $|c_1 c_2|$

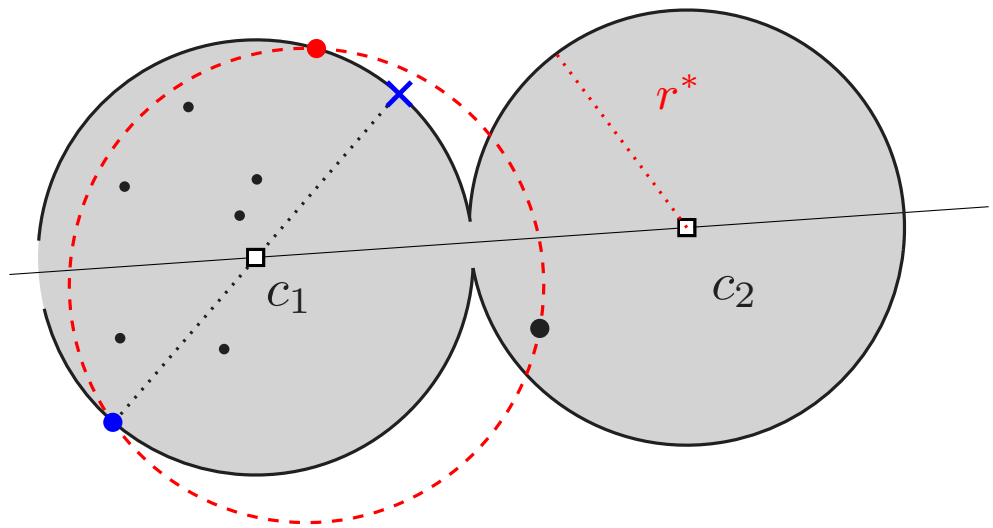


P: n points in $\mathbb{R}^2$

[COWX 24] Non-Antipodal Case

$\partial(D_1)$ has **no** point so that its **antipodal point** lies in $D_1 \cap D_2$

A **tight solution** $\{D_1, D_2\}$: smallest disks covering P while minimizing $|c_1 c_2|$



P: n points in $\mathbb{R}^2$

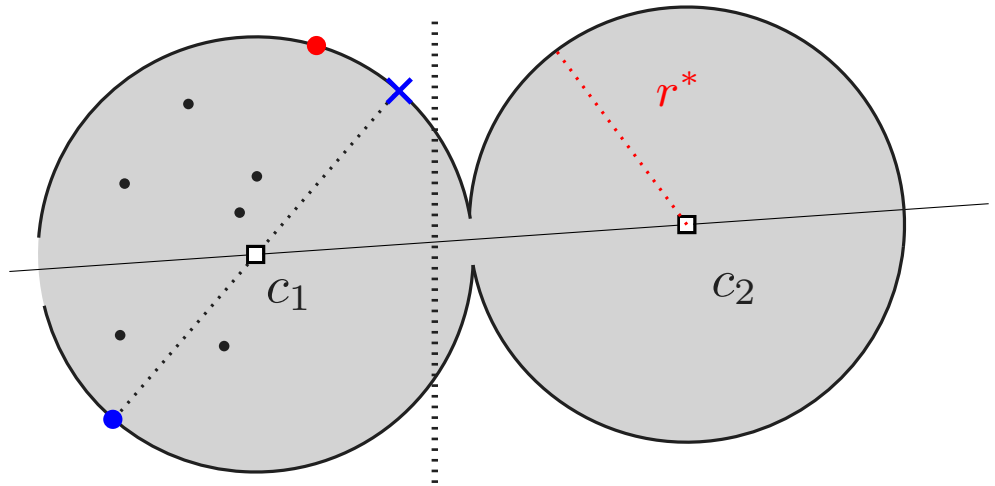
critical points: a point in $D_2 \setminus D_1$ **cannot** be in a radius r^* disk along with the points

[COWX 24] Non-Antipodal Case

$\partial(D_1)$ has **no** point so that its **antipodal point** lies in $D_1 \cap D_2$

A **tight solution** $\{D_1, D_2\}$: smallest disks covering P while minimizing $|c_1 c_2|$

axis assumption

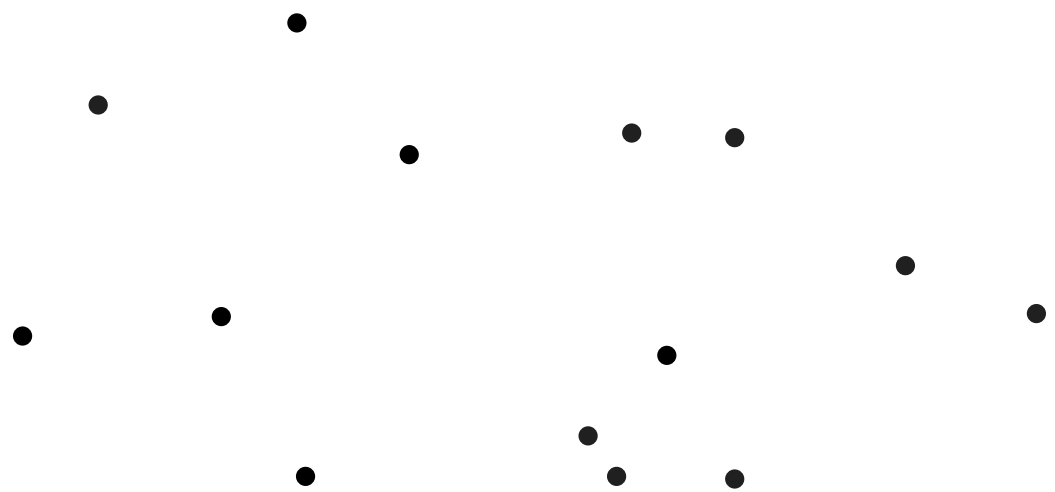


P: n points in $\mathbb{R}^2$

critical points: a point in $D_2 \setminus D_1$ **cannot** be in a radius r^* disk along with the points

[COWX 24] Decision for Non-Antipodal Case

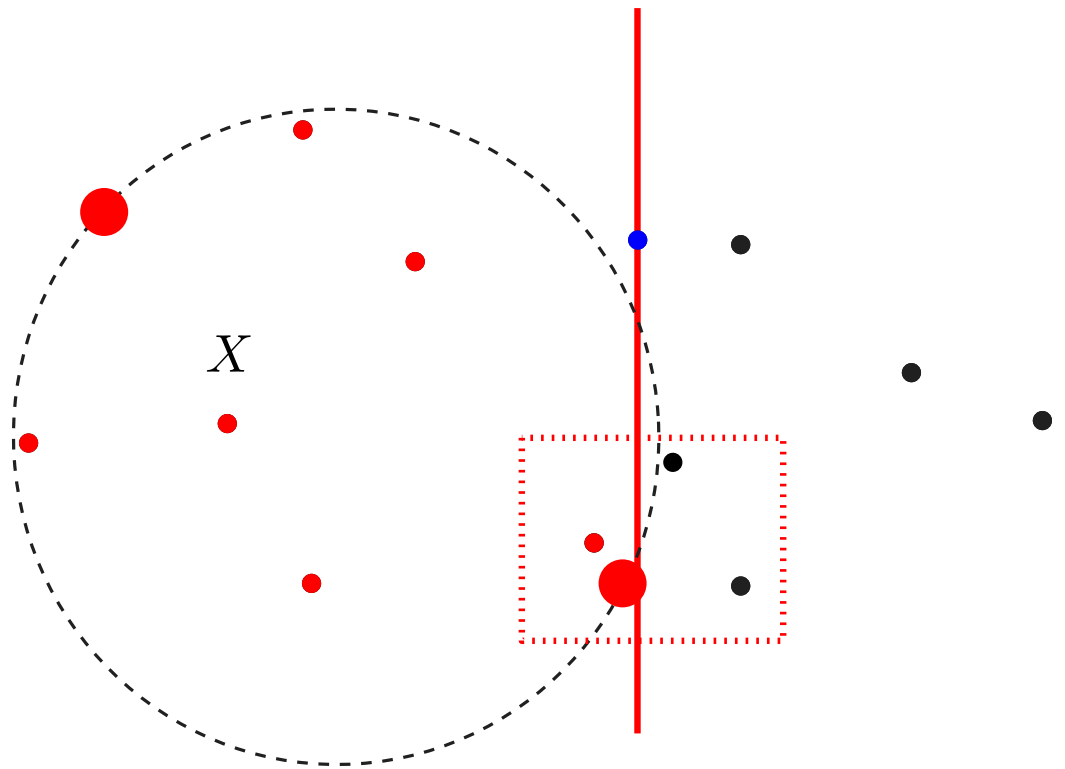
For given $r > 0$,
deciding whether $r \geq r^*$ or not in $O(n)$ time (after $O(n \log n)$ preprocessing time)



[COWX 24] Decision for Non-Antipodal Case

For given $r > 0$,
deciding whether $r \geq r^*$ or not in $O(n)$ time (after $O(n \log n)$ preprocessing time)

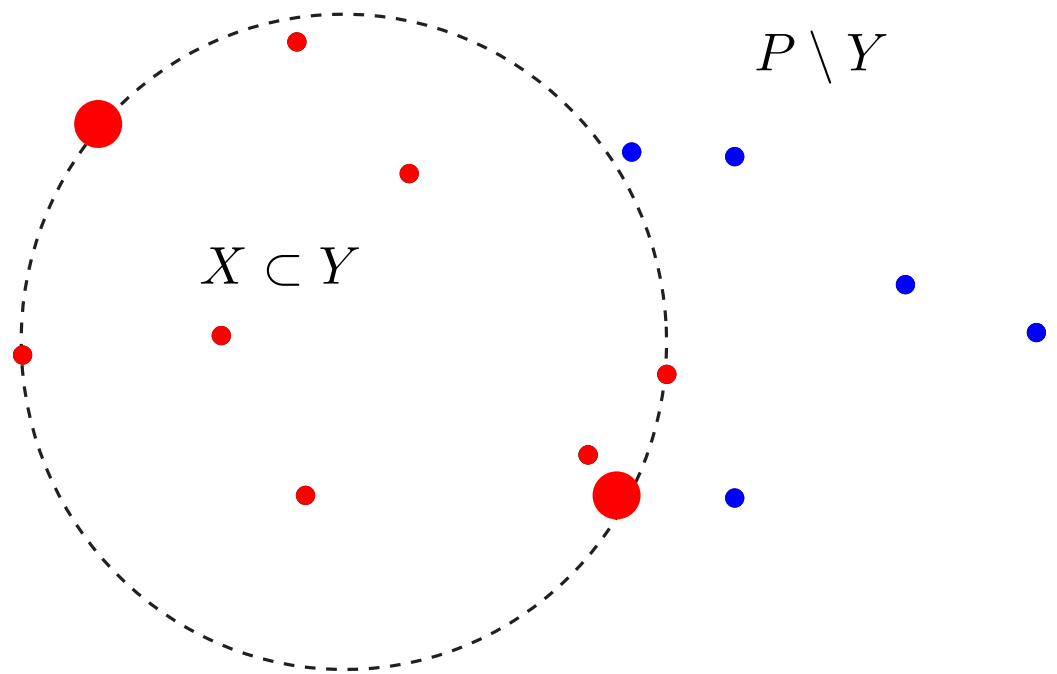
maximal r-prefix X of P



[COWX 24] Decision for Non-Antipodal Case

For given $r > 0$,
deciding whether $r \geq r^*$ or not in $O(n)$ time (after $O(n \log n)$ preprocessing time)

maximal r-prefix X of P

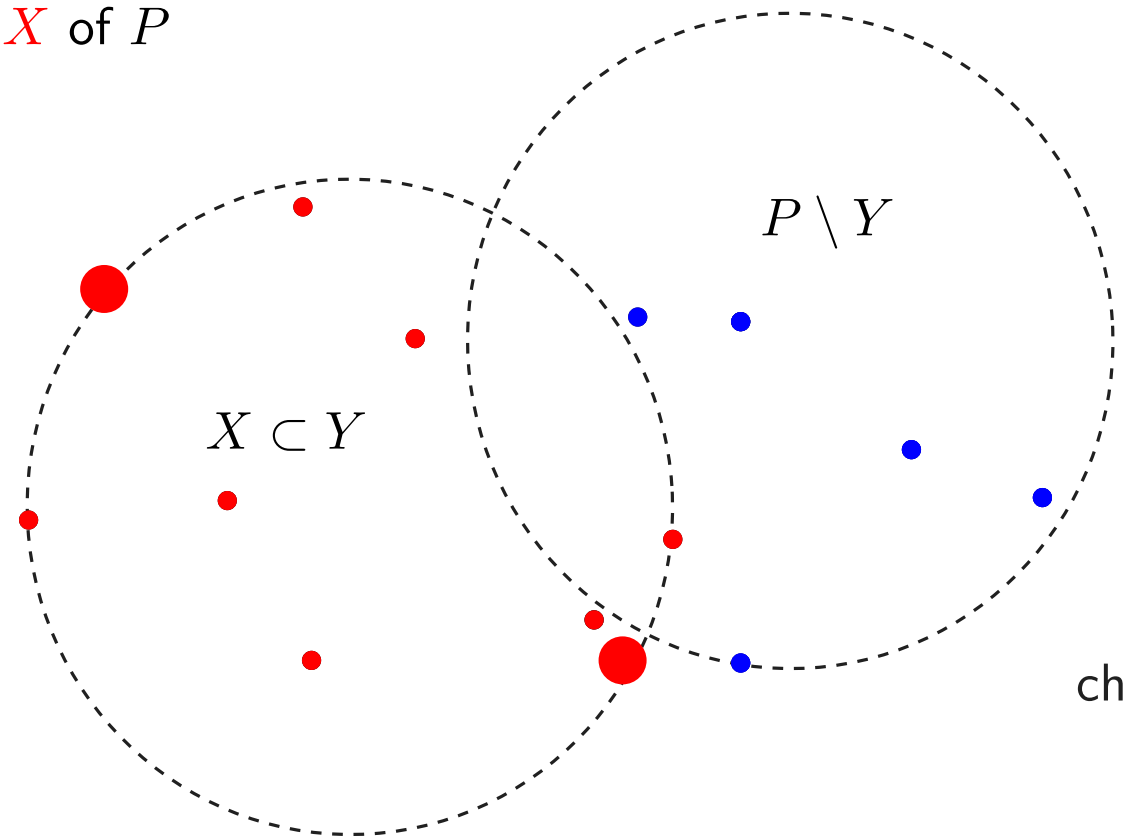


prunning $Y \subset P$: a point $q \in Y$ can be in a radius r disk along with X

[COWX 24] Decision for Non-Antipodal Case

For given $r > 0$,
deciding whether $r \geq r^*$ or not in $O(n)$ time (after $O(n \log n)$ preprocessing time)

maximal r-prefix X of P



checking Y and $P \setminus Y$

prunning $Y \subset P$: a point $q \in Y$ can be in a radius r disk along with X

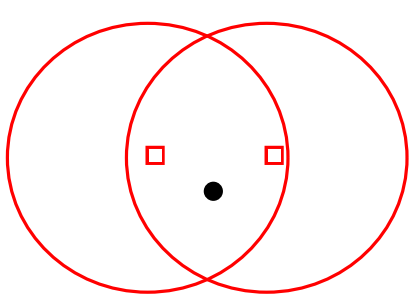
Summary

P: n points in the plane

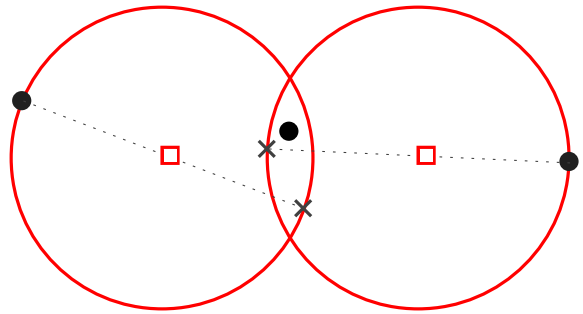
Preprocessing. $O(n \log n)$ time

- r_n: the optimal solution of the **nearby 2-center** problem of P
- r_a: the optimal solution of the **antipodal 2-center** problem of P
- P_γ: sorted P w.r.t each γ of 100 directions evenly splitting the plane

nearby 2-center problem	antipodal 2-center problem



$o \in D_1 \cap D_2$



Summary

P: n points in the plane

Preprocessing. $O(n \log n)$ time

- r_n: the optimal solution of the **nearby 2-center** problem of P
- r_a: the optimal solution of the **antipodal 2-center** problem of P
- P_γ: sorted P w.r.t each γ of 100 directions evenly splitting the plane

Decision for $r \geq$ OPT(P) or not. $O(n)$ time

- If $r \geq r_n, r_a$, then **True**.
- Otherwise, for each sorted P_γ,

Decision $r \geq r^*$ or not

ep 1) Find the maximal prefix X of P_γ

ep 2) Prune $Y \subset P$ w.r.t X

ep 3) Check Y and $P \setminus Y$

Summary

P: n points in the plane

Preprocessing. $O(n \log n)$ time

- r_n: the optimal solution of the **nearby 2-center** problem of P
- r_a: the optimal solution of the **antipodal 2-center** problem of P
- P_γ: sorted P w.r.t each γ of 100 directions evenly splitting the plane

Decision for $r \geq$ OPT(P) or not. $O(n)$ time

- If $r \geq r_n, r_a$, then **True**.
- Otherwise, for each sorted P_γ,

Using the Cole's parameteric search

The planar 2-center problem can be solved in $O(n \log n)$ time