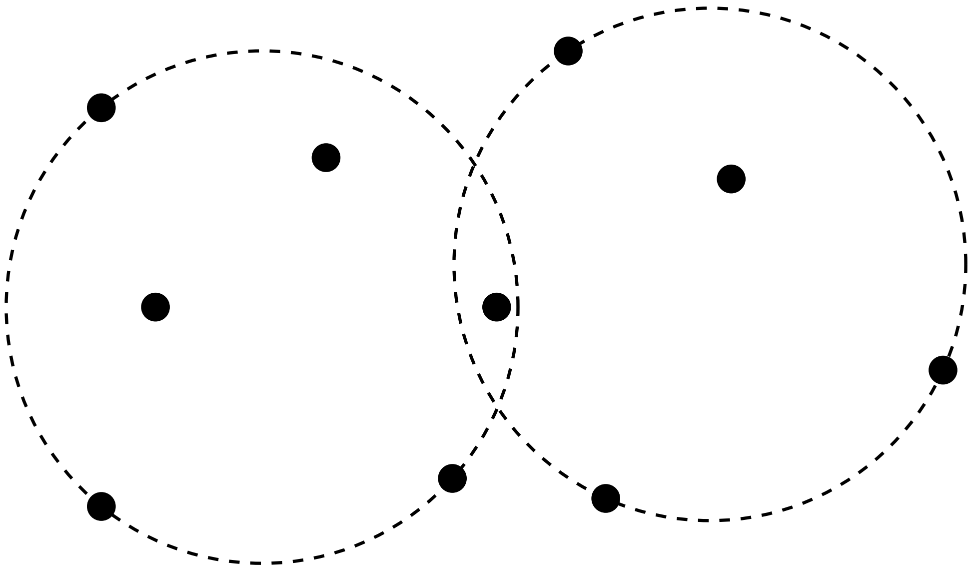
Efficient k-center algorithms for planar points in convex position

Jongmin Choi Jaegun Lee Hee-Kap Ahn

Problem Statement: Given a set of n points in plane, cover the points using k balls so that the maximum radius of a ball is minimized

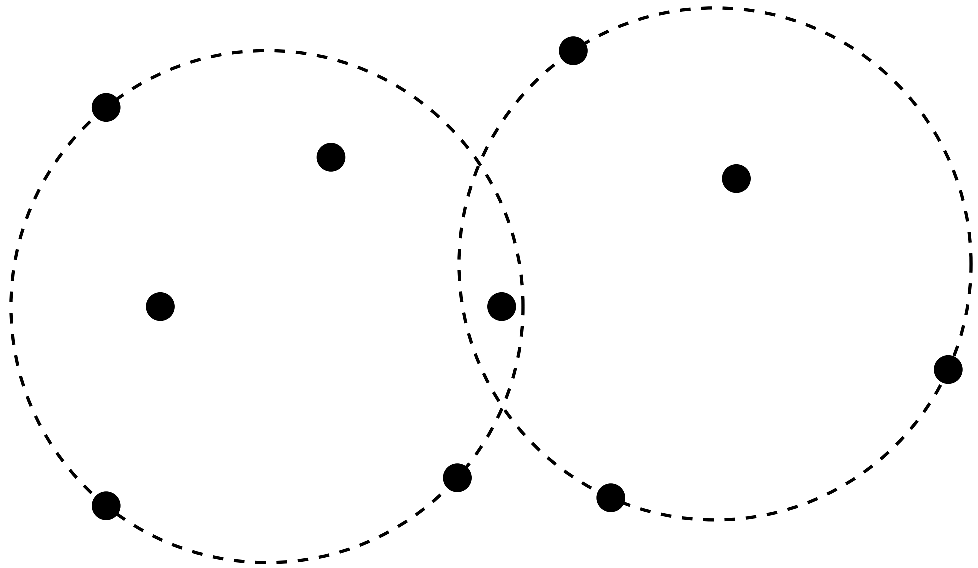
- k-center problem for arbitrary points: $n^{O(\sqrt{k})}$ time
- 2-center problem for arbitrary points: $O(n \log n)$ time



k-center problem

Problem Statement: Given a set of n points in plane, cover the points using k balls so that the maximum radius of a ball is minimized

- k-center problem for arbitrary points: $n^{O(\sqrt{k})}$ time
- 2-center problem for arbitrary points: $O(n \log n)$ time

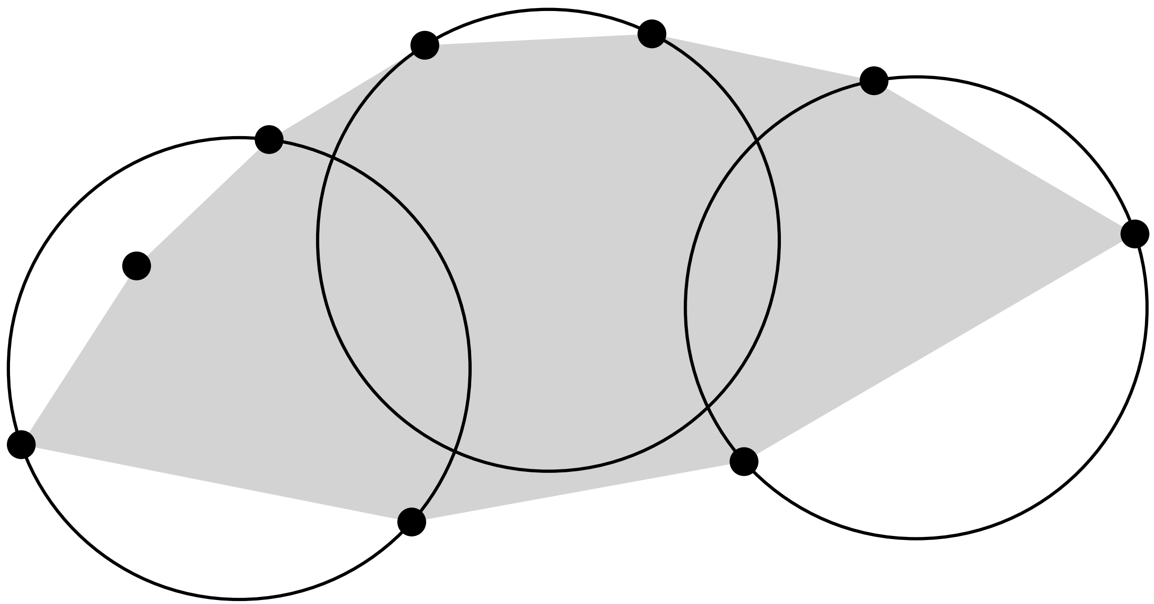


- 2-center problem for points in convex position: $O(n \log n)$ time
- 3-center problem for points in convex position: $O(n^2 \log^3 n)$ time

Our results

First efficient algorithm for planar k-center problem for points in convex position

- $O(n^2 \min\{k, \log n\} \log n + k^2 n \log n)$-time algorithm
- For $k = 3$, $O(n^2 \log^3 n) \Rightarrow O(n^2 \log n)$.



Preliminaries

$P = \langle p_0, \dots, p_{n-1} \rangle$: (cyclic) sequence of points in clockwise order

$P(i, t) = \langle p_i, p_{i+1}, \dots, p_{i+t-1} \rangle$

substring: subsequence that consists of a consecutive run of elements

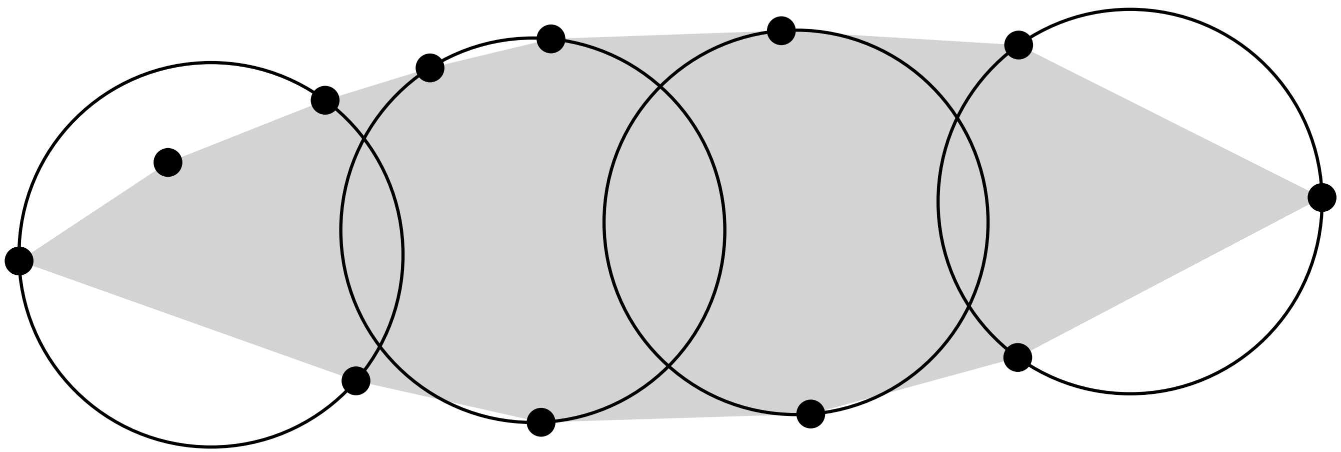
Preliminaries

$P = \langle p_0, \dots, p_{n-1} \rangle$: (cyclic) sequence of points in clockwise order

$P(i, t) = \langle p_i, p_{i+1}, \dots, p_{i+t-1} \rangle$

substring: subsequence that consists of a consecutive run of elements

(ℓ, r)-partition: partition into substrings such that set of ℓ disks with radius r can cover the partition



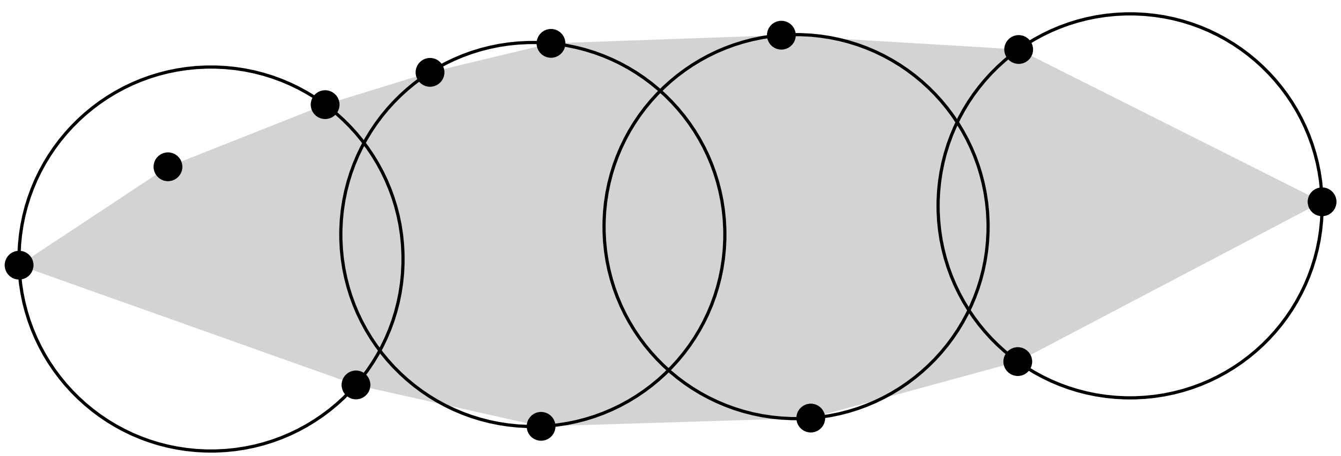
Preliminaries

$P = \langle p_0, \dots, p_{n-1} \rangle$: (cyclic) sequence of points in clockwise order

$P(i, t) = \langle p_i, p_{i+1}, \dots, p_{i+t-1} \rangle$

substring: subsequence that consists of a consecutive run of elements

(ℓ, r)-partition: partition into substrings such that set of ℓ disks with radius r can cover the partition



Lemma. If P can be covered by ℓ disks with radius r, there exists a (ℓ, r)-partition which is *line-separable* and *balanced*((ℓ, r)-cover).

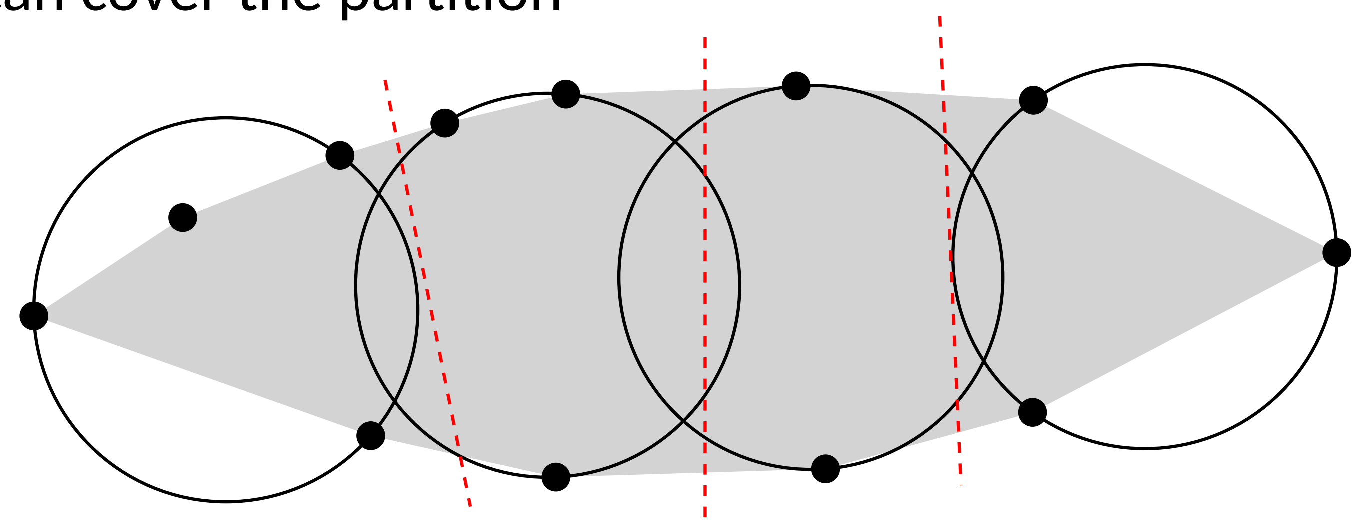
- *balanced*: each group of substring paired by a disk consists of at most two nonconsecutive substrings

Preliminaries

$P = \langle p_0, \dots, p_{n-1} \rangle$: (cyclic) sequence of points in clockwise order
$P(i, t) = \langle p_i, p_{i+1}, \dots, p_{i+t-1} \rangle$

substring: subsequence that consists of a consecutive run of elements

(ℓ, r)-partition: partition into substrings such that set of ℓ disks with radius r can cover the partition



Lemma. If P can be covered by ℓ disks with radius r, there exists a (ℓ, r)-partition which is *line-separable* and *balanced*((ℓ, r)-cover).

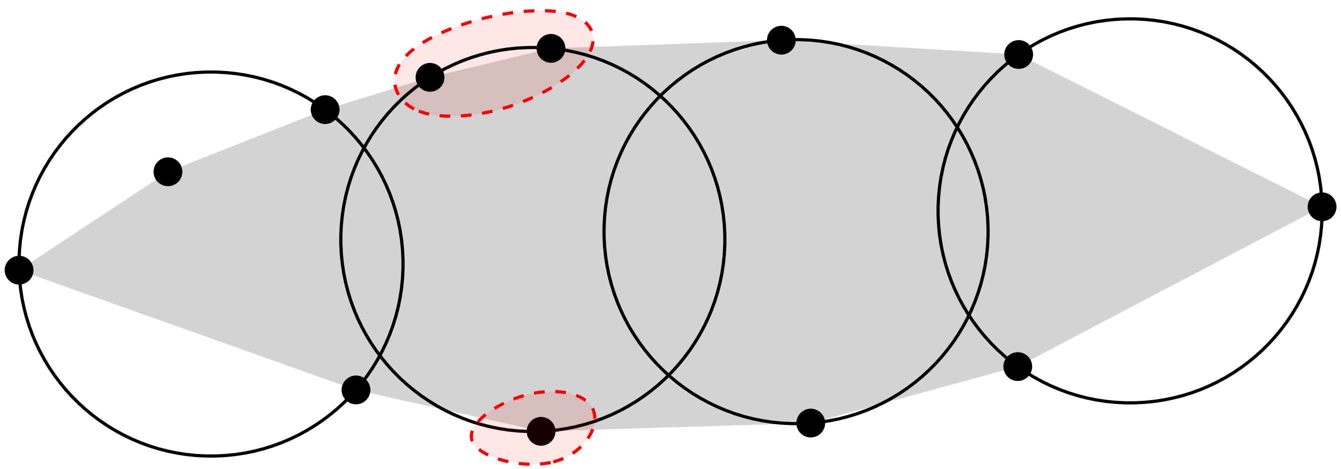
- *balanced*: each group of substring paired by a disk consists of at most two nonconsecutive substrings

Preliminaries

$P = \langle p_0, \dots, p_{n-1} \rangle$: (cyclic) sequence of points in clockwise order
$P(i, t) = \langle p_i, p_{i+1}, \dots, p_{i+t-1} \rangle$

substring: subsequence that consists of a consecutive run of elements

(ℓ, r)-partition: partition into substrings such that set of ℓ disks with radius r can cover the partition



Lemma. If P can be covered by ℓ disks with radius r, there exists a (ℓ, r)-partition which is *line-separable* and *balanced*((ℓ, r)-cover).

- *balanced*: each group of substring paired by a disk consists of at most two nonconsecutive substrings

Decision Algorithm

Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

Decision Algorithm

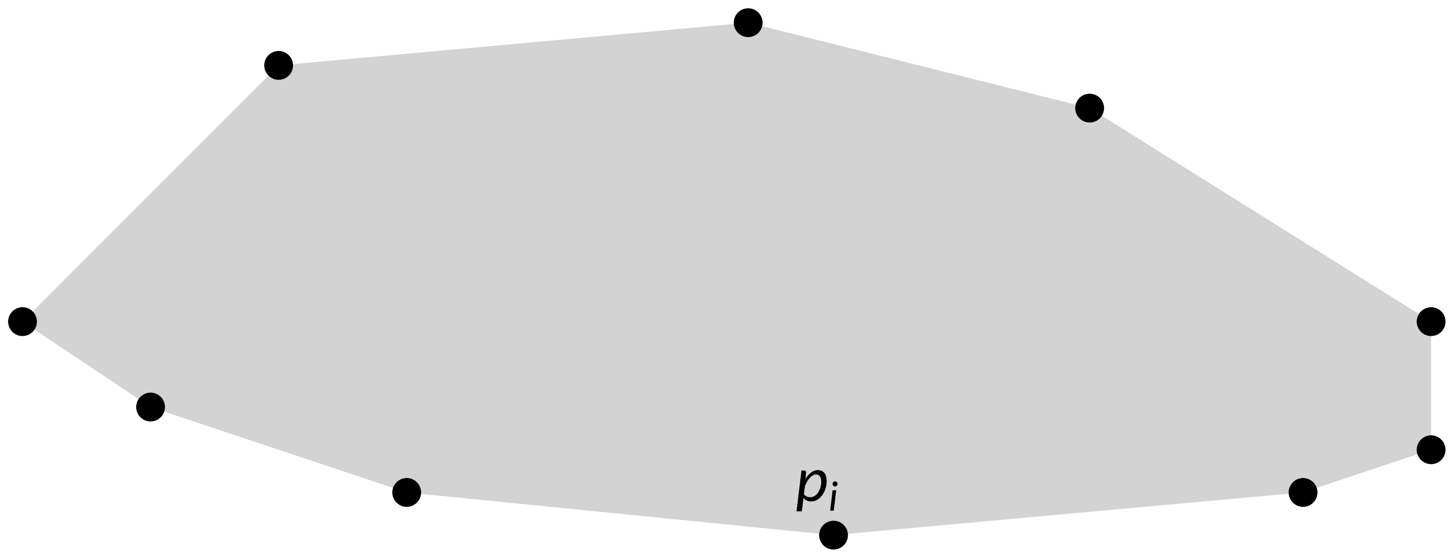
Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

$f(i, \ell)$: length of the longest substring of P from p_i that admits an (ℓ, r)-cover

Decision Algorithm

Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

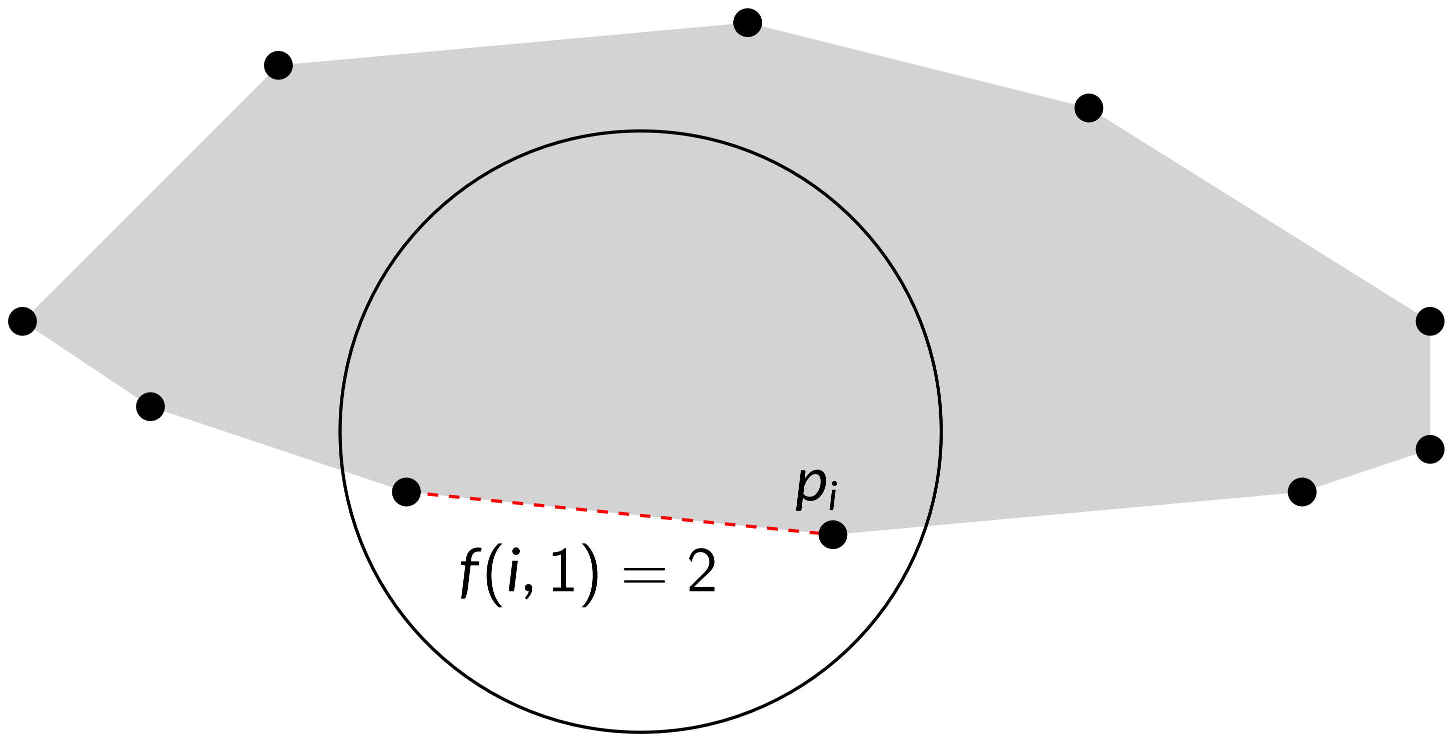
$f(i, \ell)$: length of the longest substring of P from p_i that admits an (ℓ, r)-cover



Decision Algorithm

Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

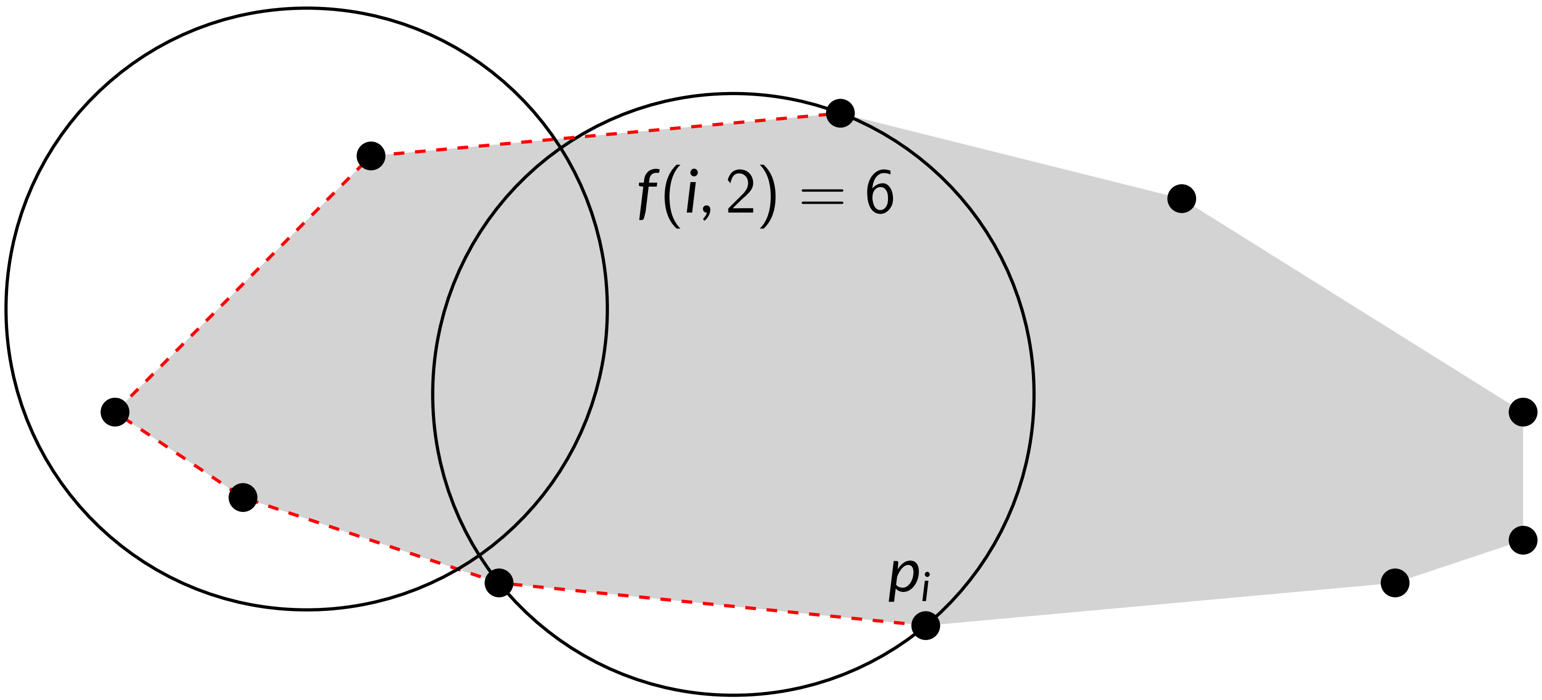
$f(i, \ell)$: length of the longest substring of P from p_i that admits an (ℓ, r)-cover



Decision Algorithm

Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

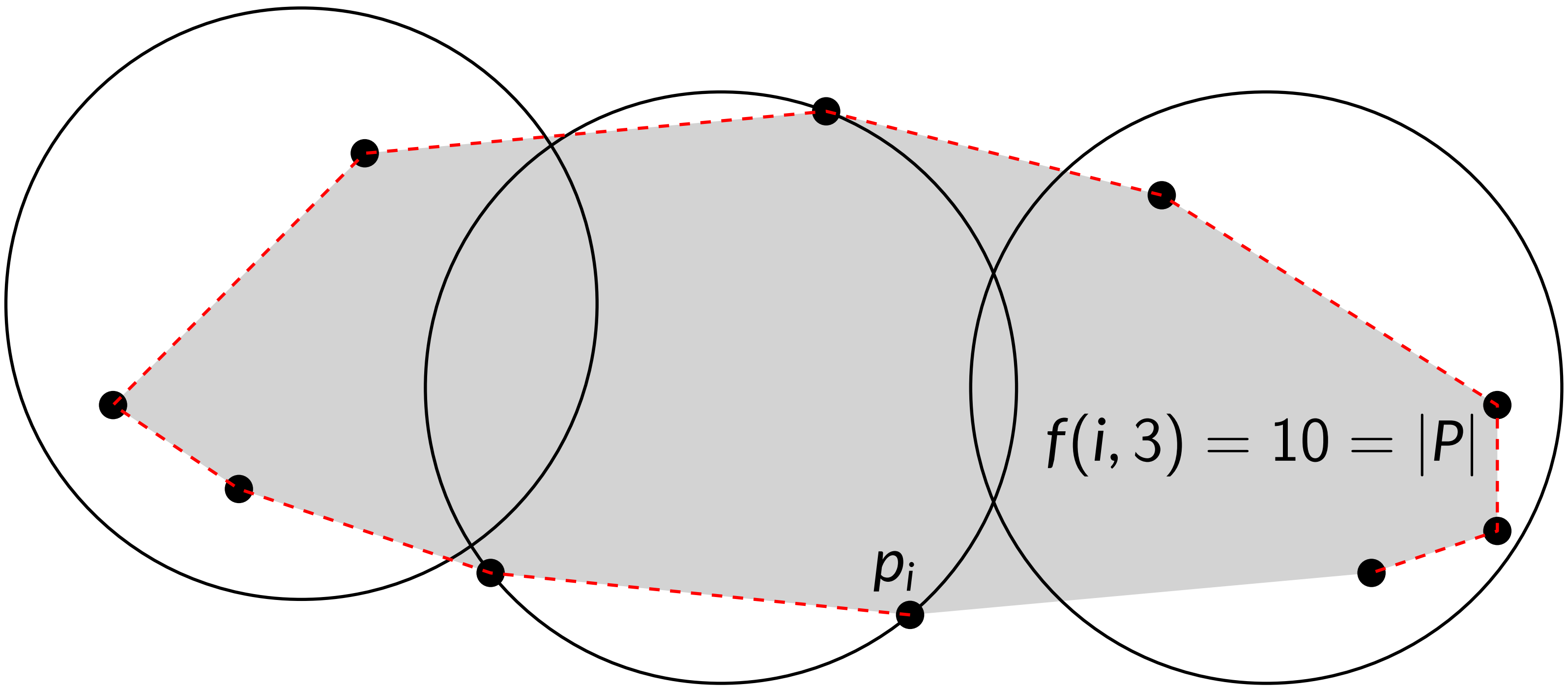
$f(i, \ell)$: length of the longest substring of P from p_i that admits an (ℓ, r)-cover



Decision Algorithm

Statement: Given set of points P, integer k, and a real value r, determine whether P admits (k, r)-cover.

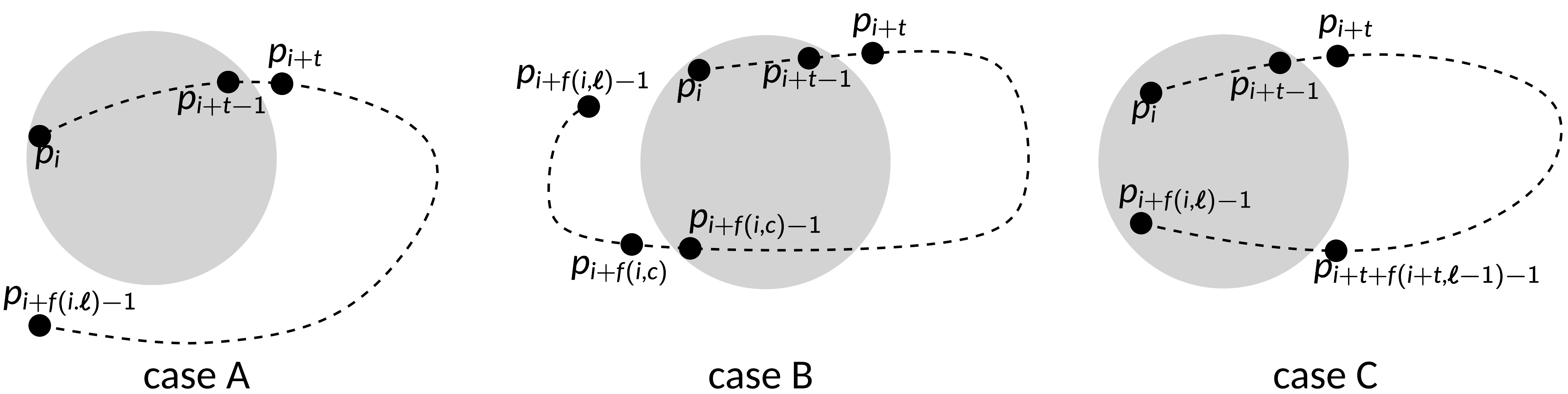
$f(i, \ell)$: length of the longest substring of P from p_i that admits an (ℓ, r)-cover



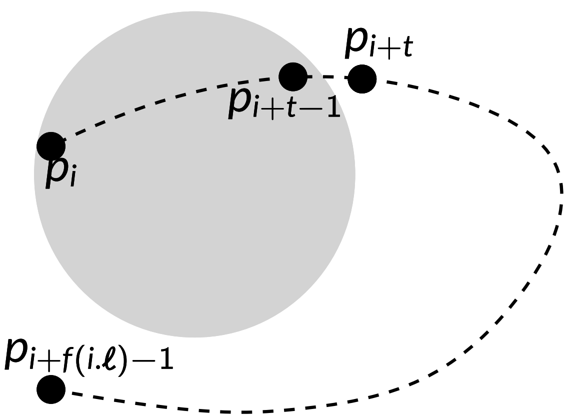
- P admits (k, r)-cover if $f(i, k) \geq |P|$ for some i

Decision Algorithm

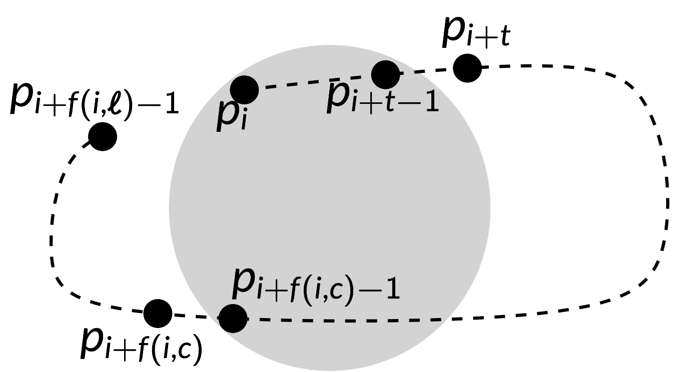
For a substring $P(i, f(i, \ell))$, there are three cases for the group that contains the first substring:



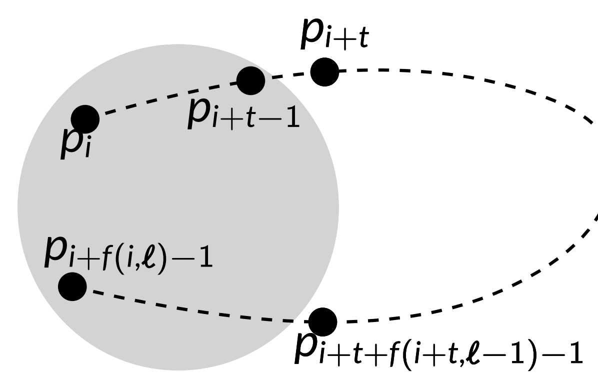
For a substring $P(i, f(i,\ell))$, there are three cases for the group that contains the first substring:



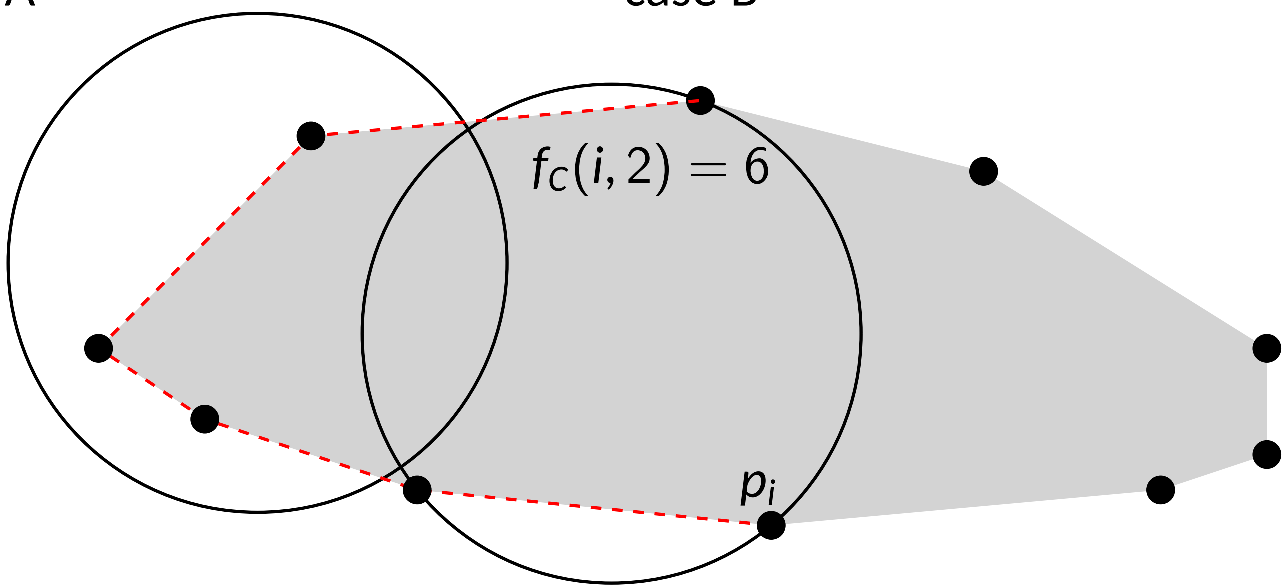
case A



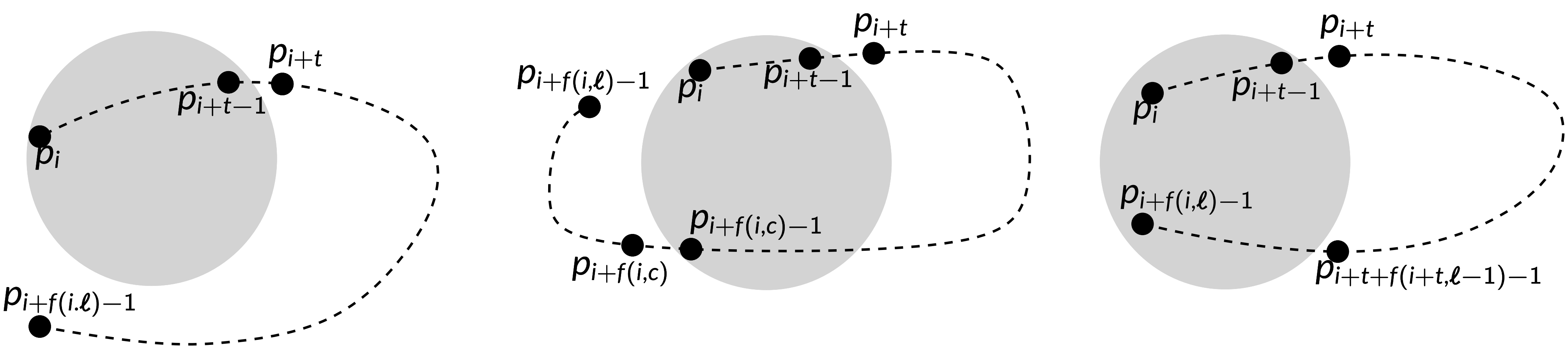
case B



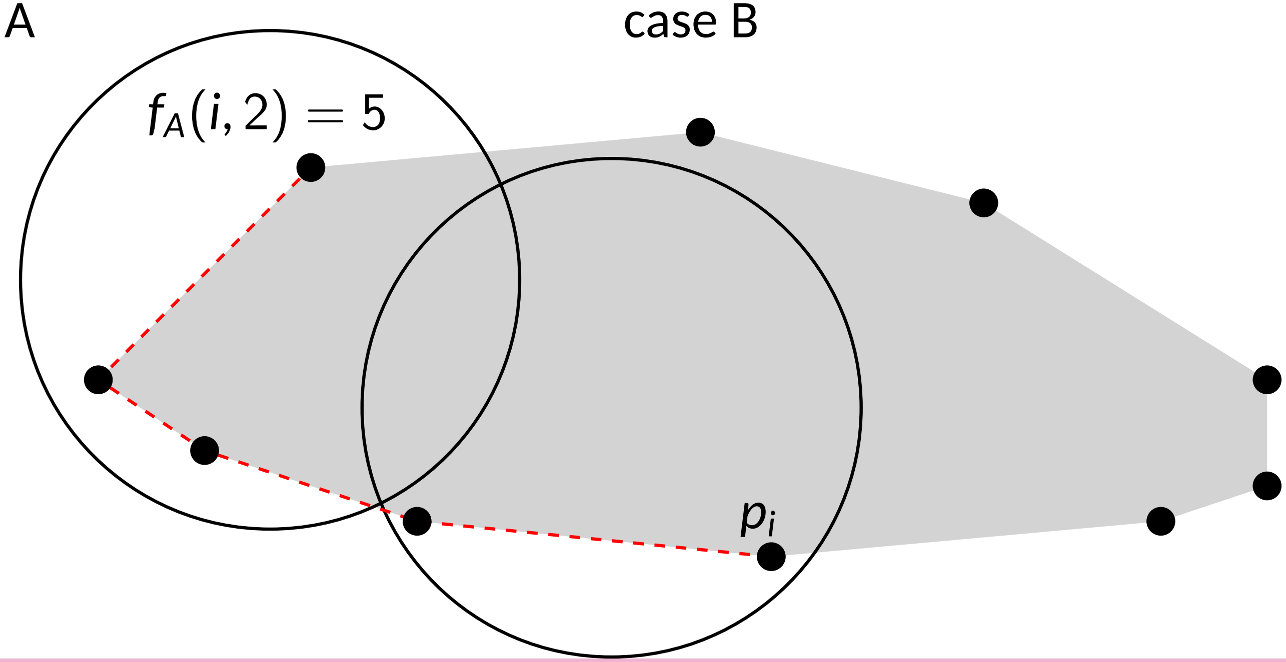
case C



For a substring $P(i, f(i, \ell))$, there are three cases for the group that contains the first substring:

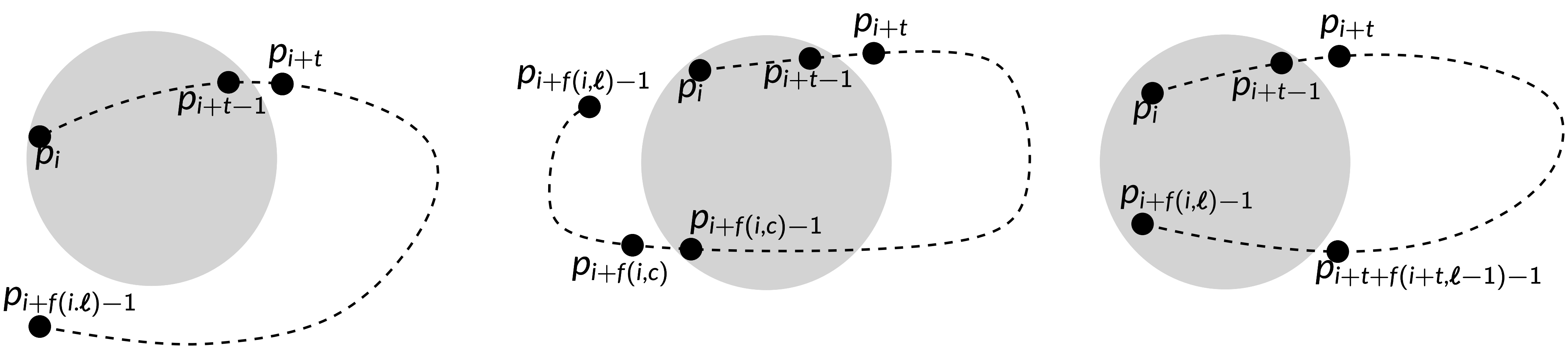


case A case B case C

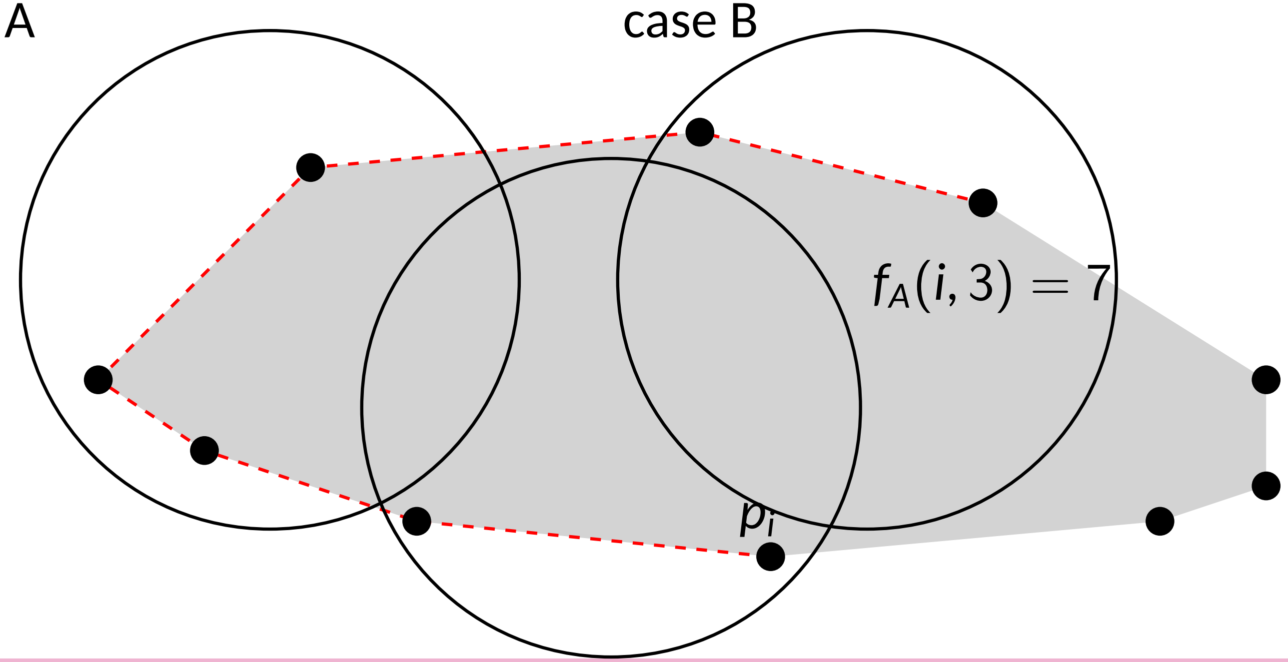


Decision Algorithm

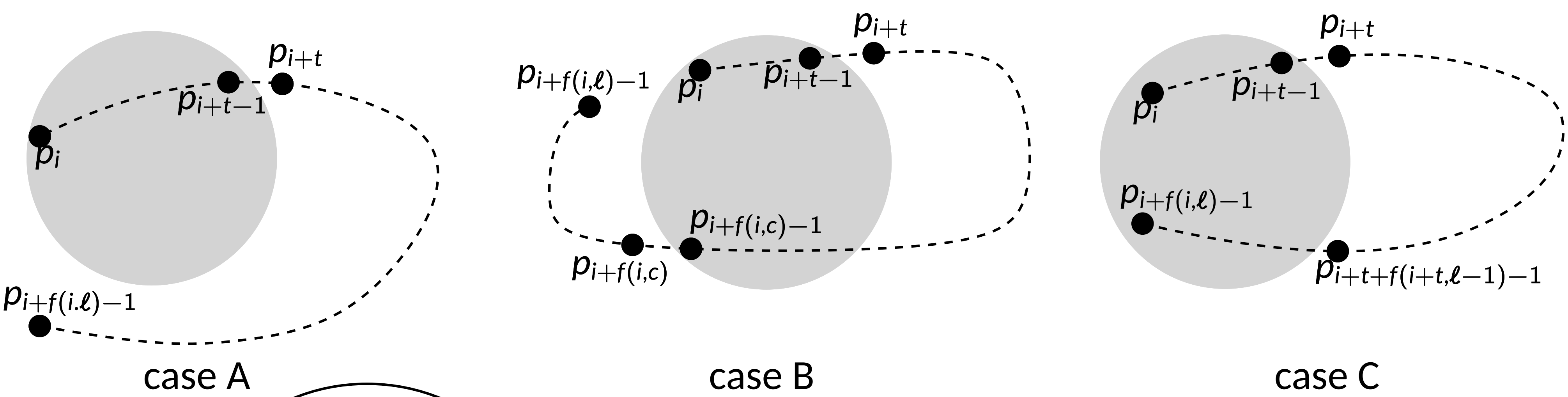
For a substring $P(i, f(i, \ell))$, there are three cases for the group that contains the first substring:

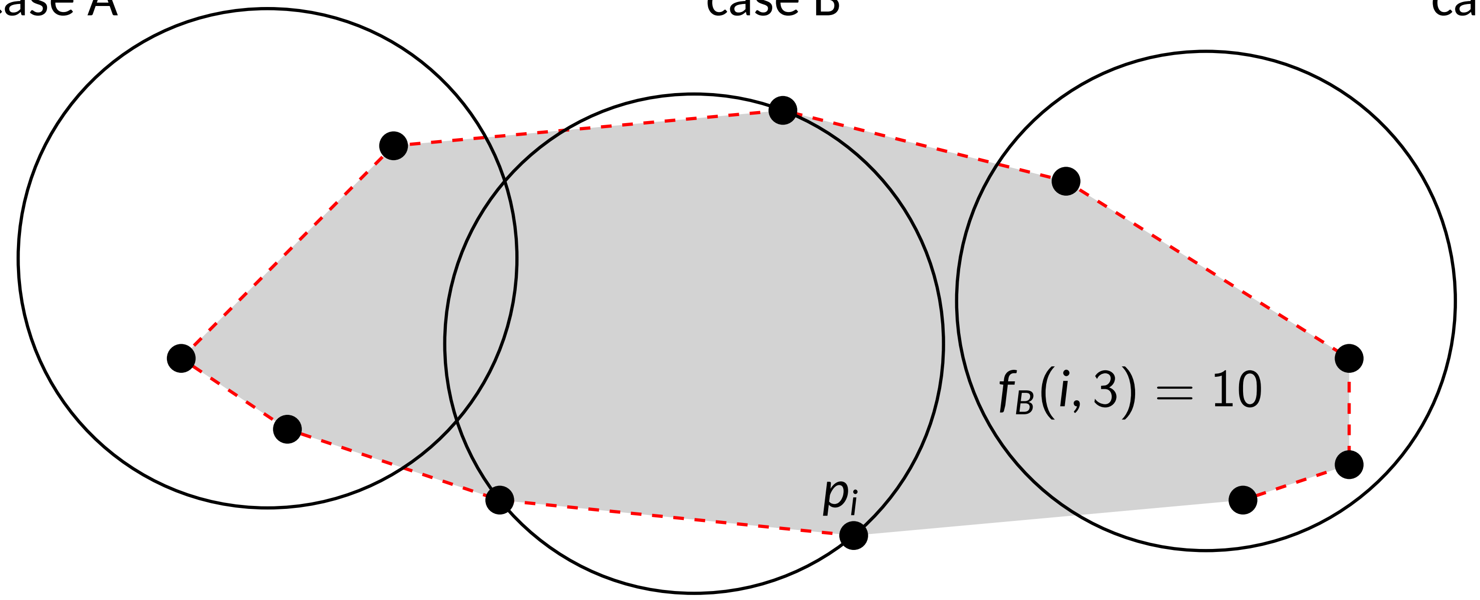


case A case B case C



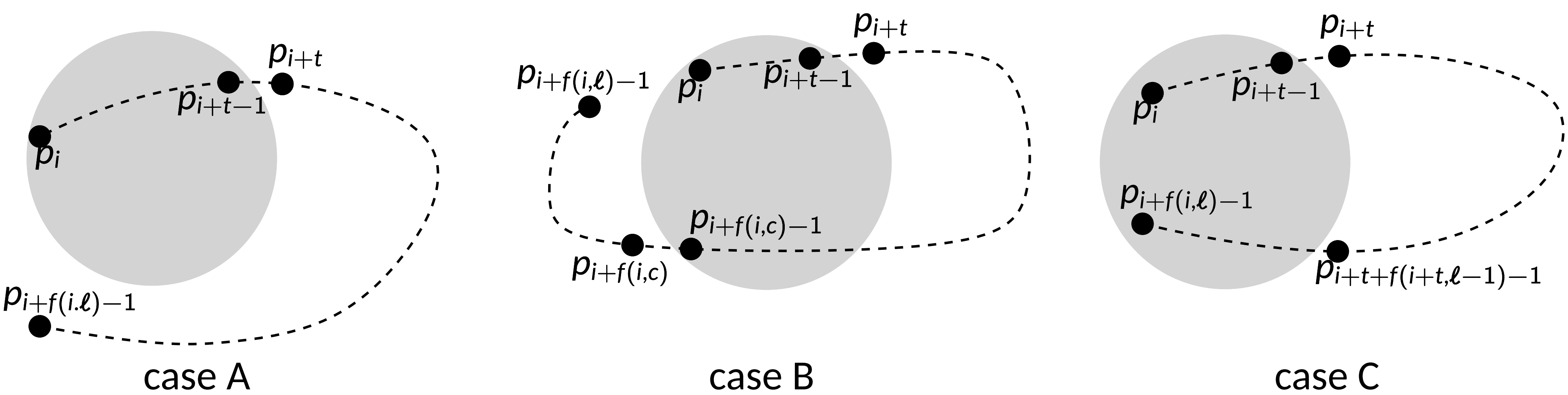
For a substring $P(i, f(i,\ell))$, there are three cases for the group that contains the first substring:





Decision Algorithm

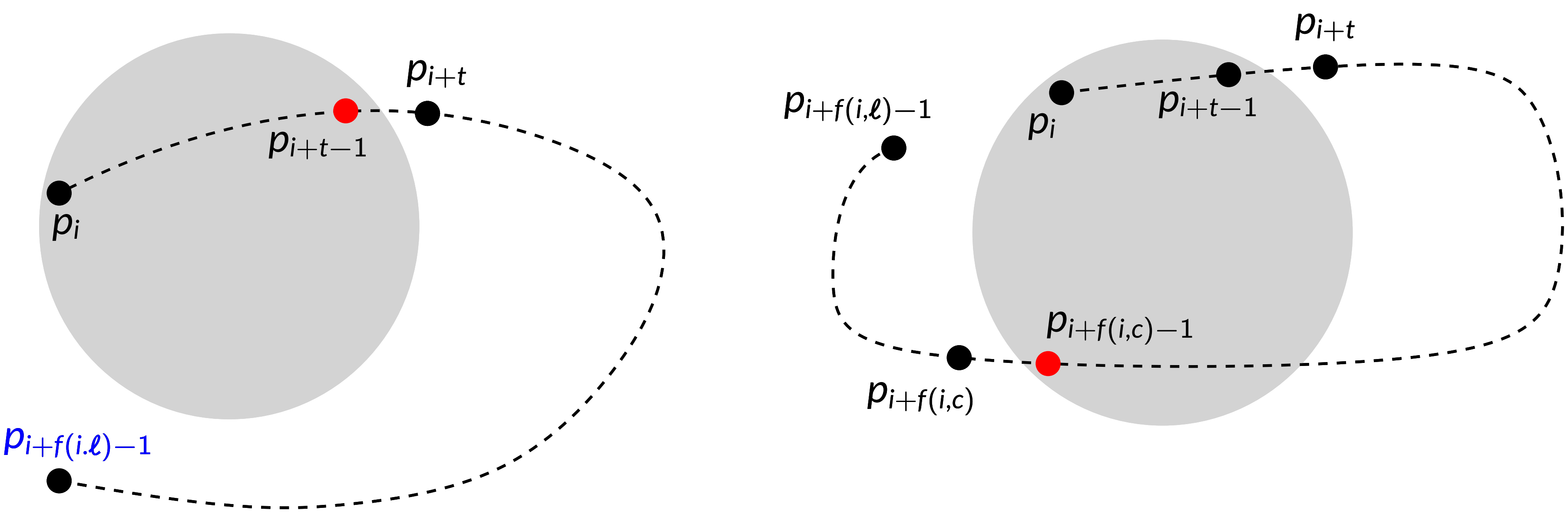
For a substring $P(i, f(i, \ell))$, there are three cases for the group that contains the first substring:



$$f(i, \ell) = \max\{f_A(i, \ell), f_B(i, \ell), f_C(i, \ell)\}$$

Decision Algorithm

For case A & B: $O(k^2 n)$-time algorithm



Observation. The value $f(i,\ell)+i$ is nondecreasing while i and ℓ is increasing.

$$f_A(i,\ell) = f(i,1) + f(i+f(i,1),\ell-1)$$

$$\max\{f_A(i,\ell), f_B(i,\ell)\} = \max_{1\leqslant\gamma\leqslant\ell-1}\{f(i,\gamma)+f(i+f(i,\gamma),\ell-\gamma)\}$$

Decision Algorithm

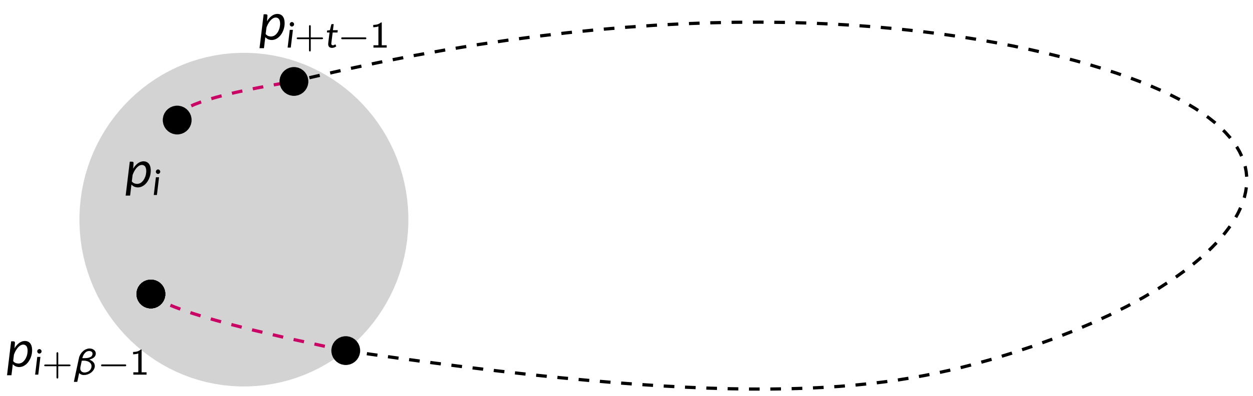
For case C: $O(n^2 \log n)$-time / $O(kn^2)$-time algorithm

Decision Algorithm

For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$

Decision Algorithm

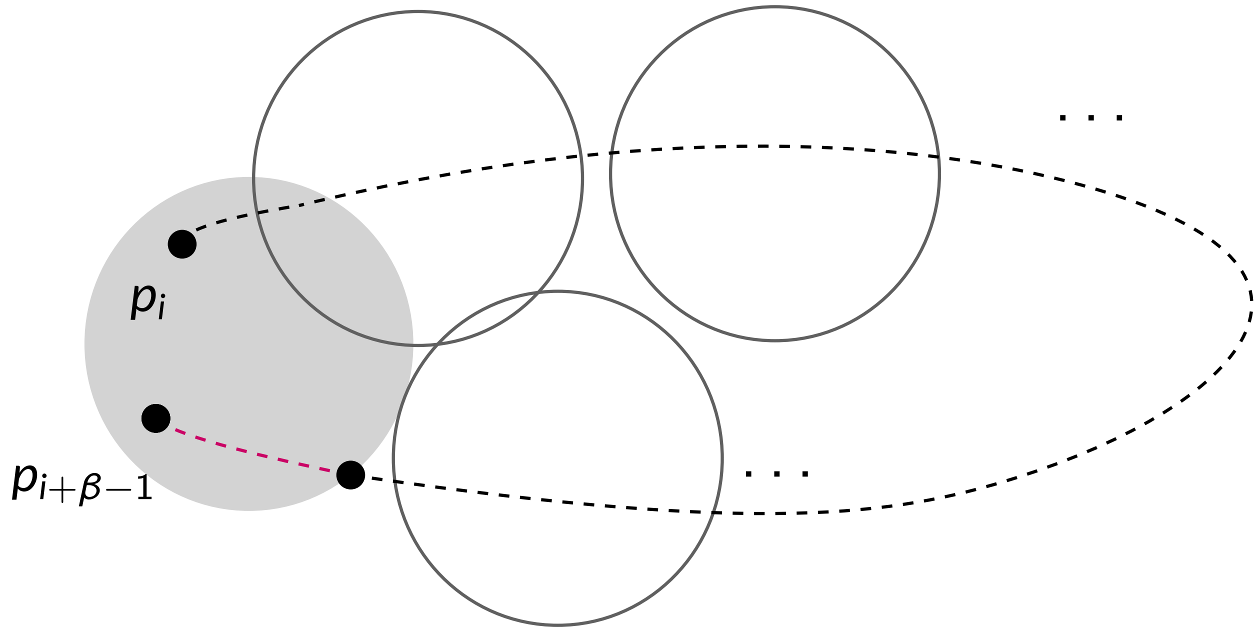
For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$



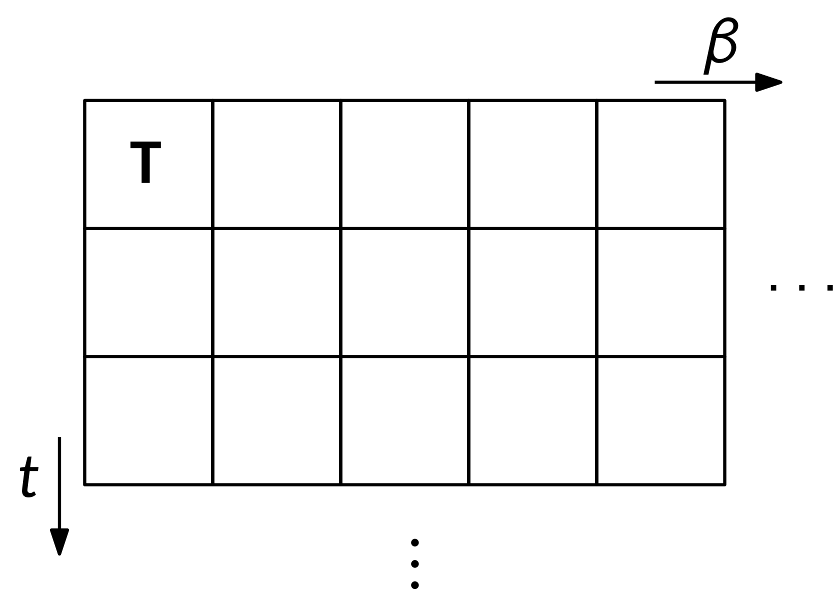
Decision Algorithm

For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$

$t = 1$



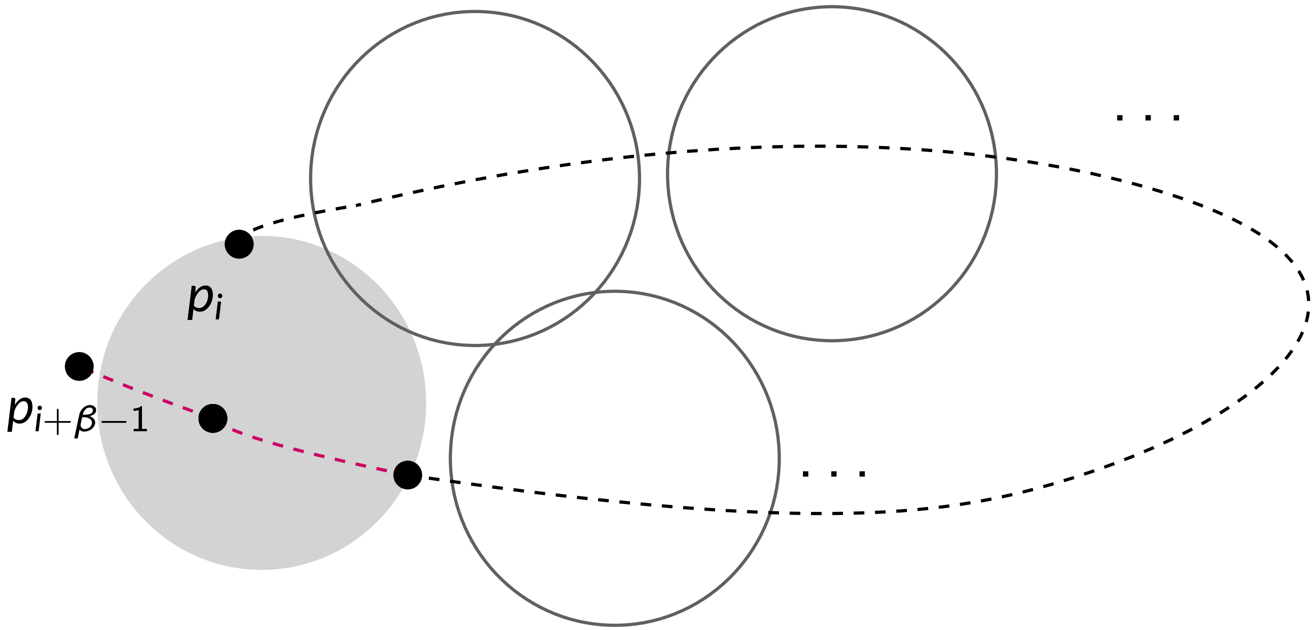
Initially, $t = 1$, $\beta = f_A(i, \ell)$



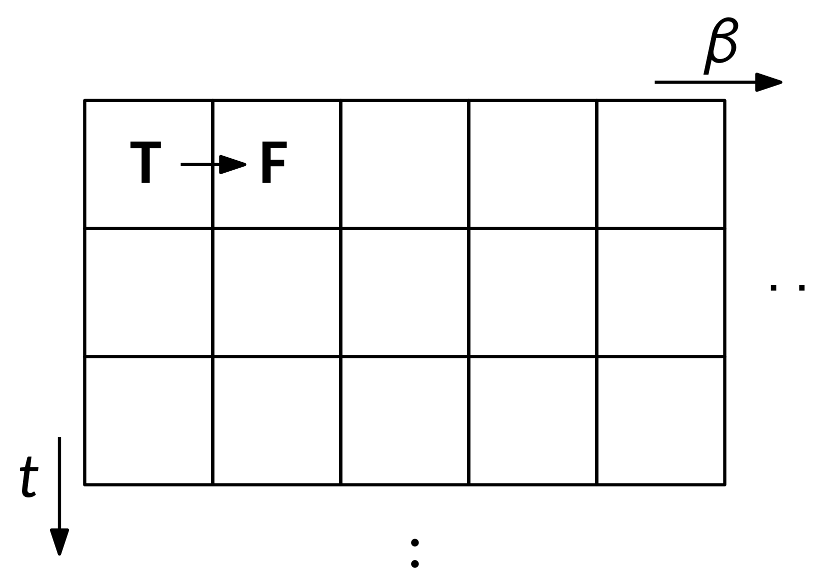
Decision Algorithm

For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$

$t = 1$

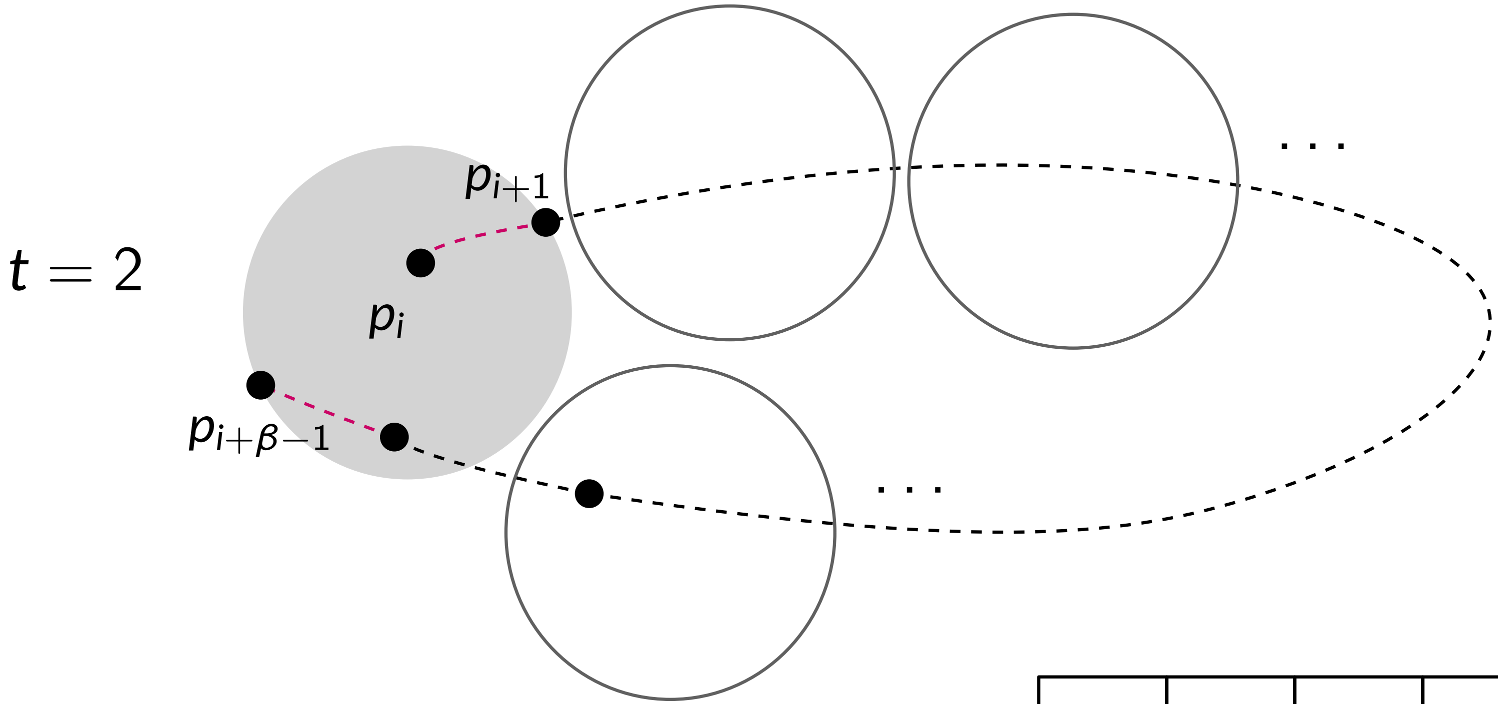


Initially, $t = 1$, $\beta = f_A(i, \ell)$

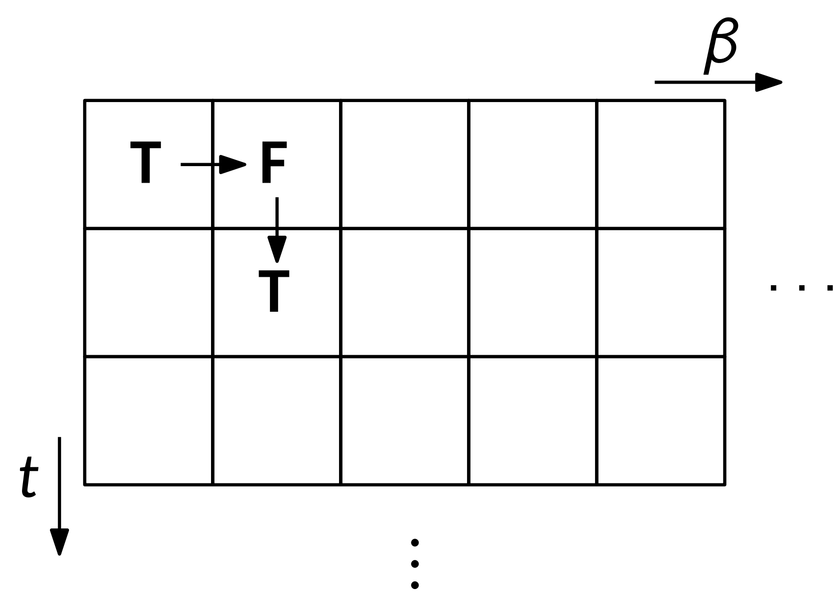


Decision Algorithm

For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$

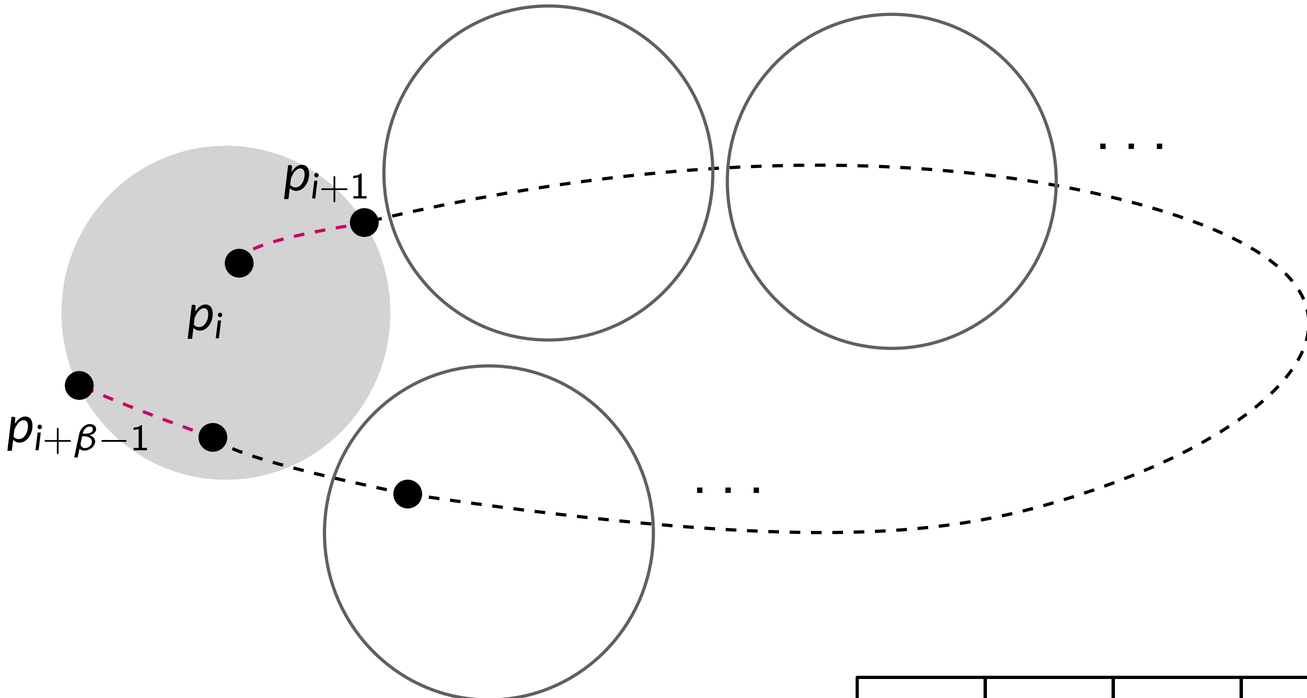


Initially, $t = 1$, $\beta = f_A(i, \ell)$



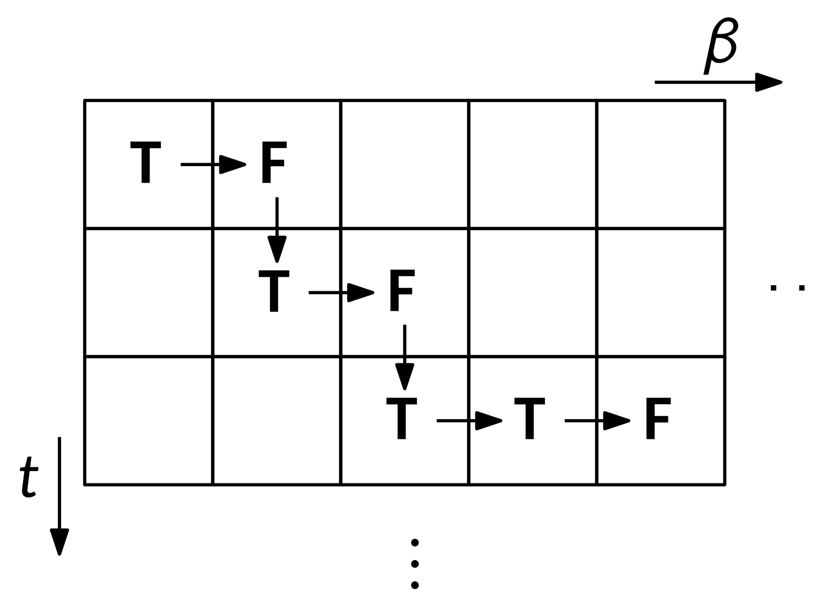
Decision Algorithm

For case C: $O(n^2 \log n)$-time / $\boxed{O(kn^2)\text{-time algorithm}}$



Initially, $t = 1$, $\beta = f_A(i, \ell)$

Each step: amortized $O(1)$ time



Decision Algorithm

Lemma. $\max\{f_A(i,\ell), f_B(i,\ell)\}$ can be computed in $O(k^2 n)$ time for all i and ℓ.

Lemma. $\max\{f_A(i,\ell), f_C(i,\ell)\}$ can be computed in $O(\min\{kn^2, n^2 \log n\})$ time for all i and ℓ.

Decision Algorithm

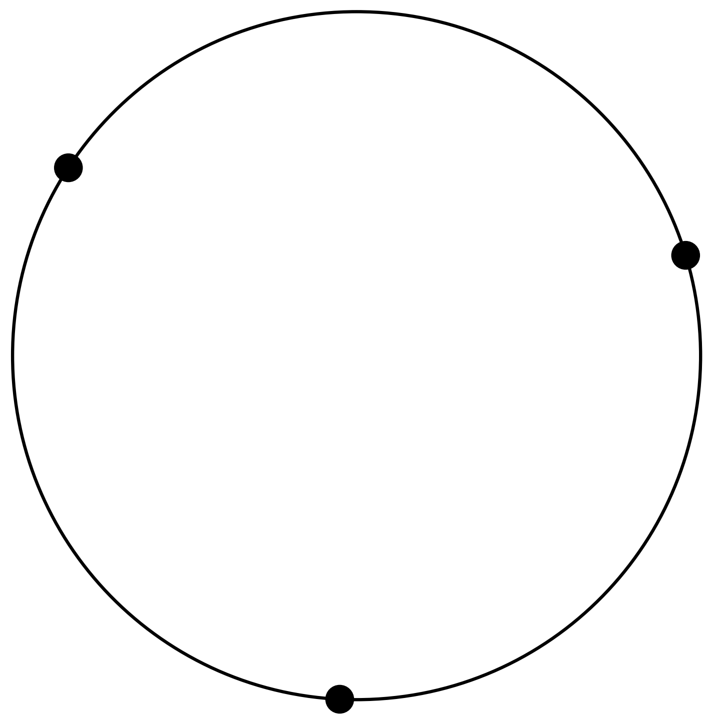
Lemma. $\max\{f_A(i,\ell), f_B(i,\ell)\}$ can be computed in $O(k^2 n)$ time for all i and ℓ.

Lemma. $\max\{f_A(i,\ell), f_C(i,\ell)\}$ can be computed in $O(\min\{kn^2, n^2 \log n\})$ time for all i and ℓ.

Theorem. Given a set of n points in convex position, an integer k, and a radius r, we can determine in $O(n^2 \min\{k, \log n\} + k^2 n)$ time whether the set admits a (k, r)-cover or not.

Let the running time of decision algorithm $T_S = O(n^2 \min\{k, \log n\} + k^2 n)$

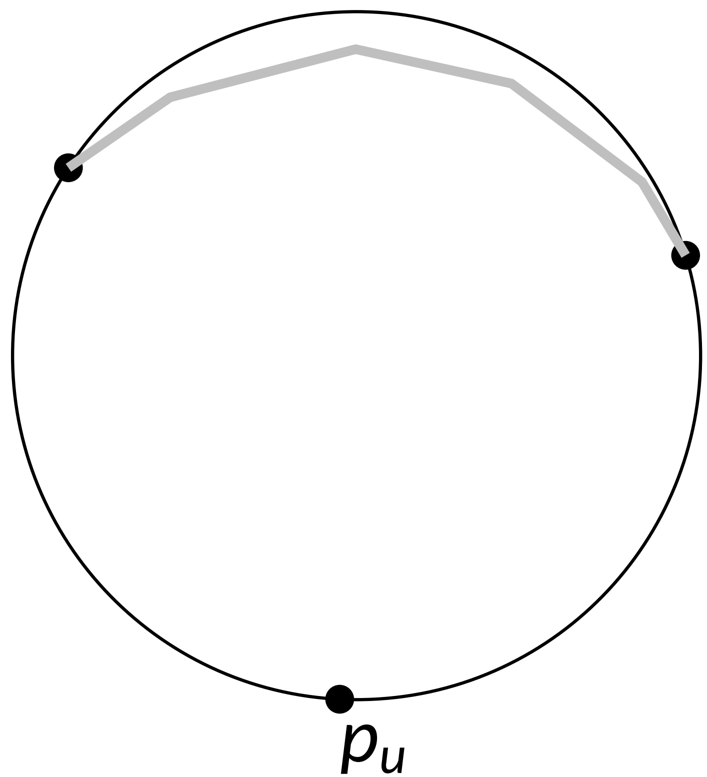
Smallest disk: defined by at most three points



In a naïve way, it takes $O(n^3 \log n)$ time by applying binary search over the set of $O(n^3)$ radii

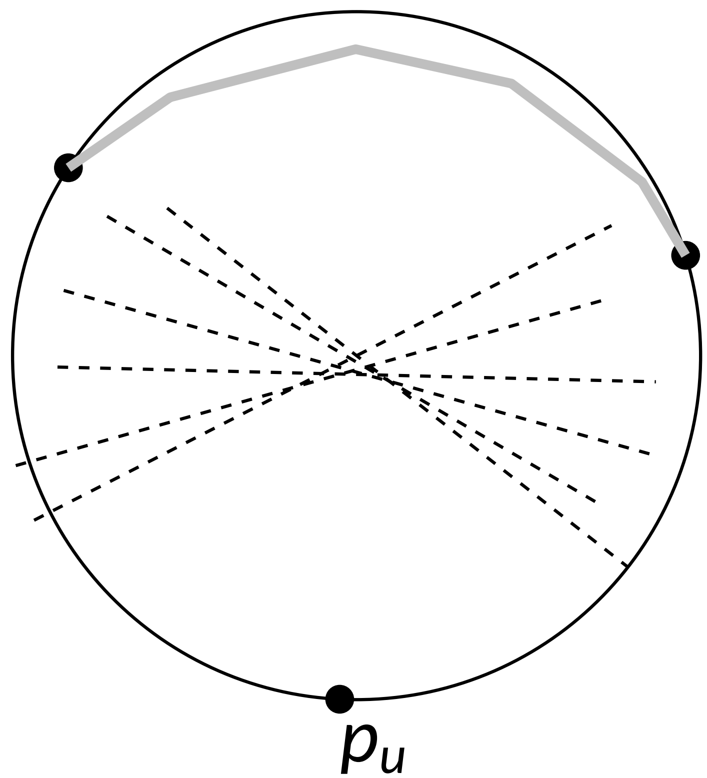
Search Algorithm

Smallest disk: defined by at most three points



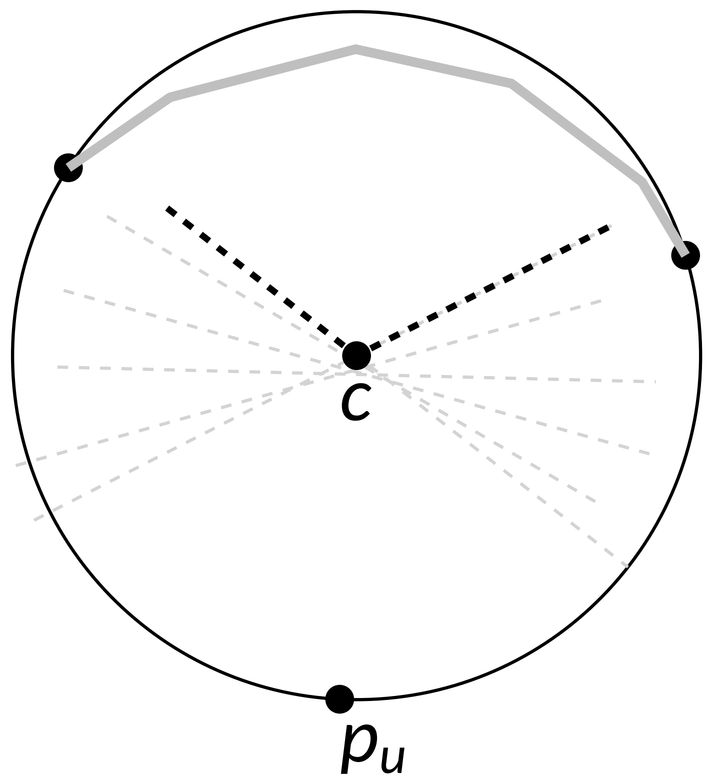
Since there are at most two substrings corresponding to a disk, the smallest disk is defined by a point and a substring.

Smallest disk: defined by at most three points



Since there are at most two substrings corresponding to a disk, the smallest disk is defined by a point and a substring.

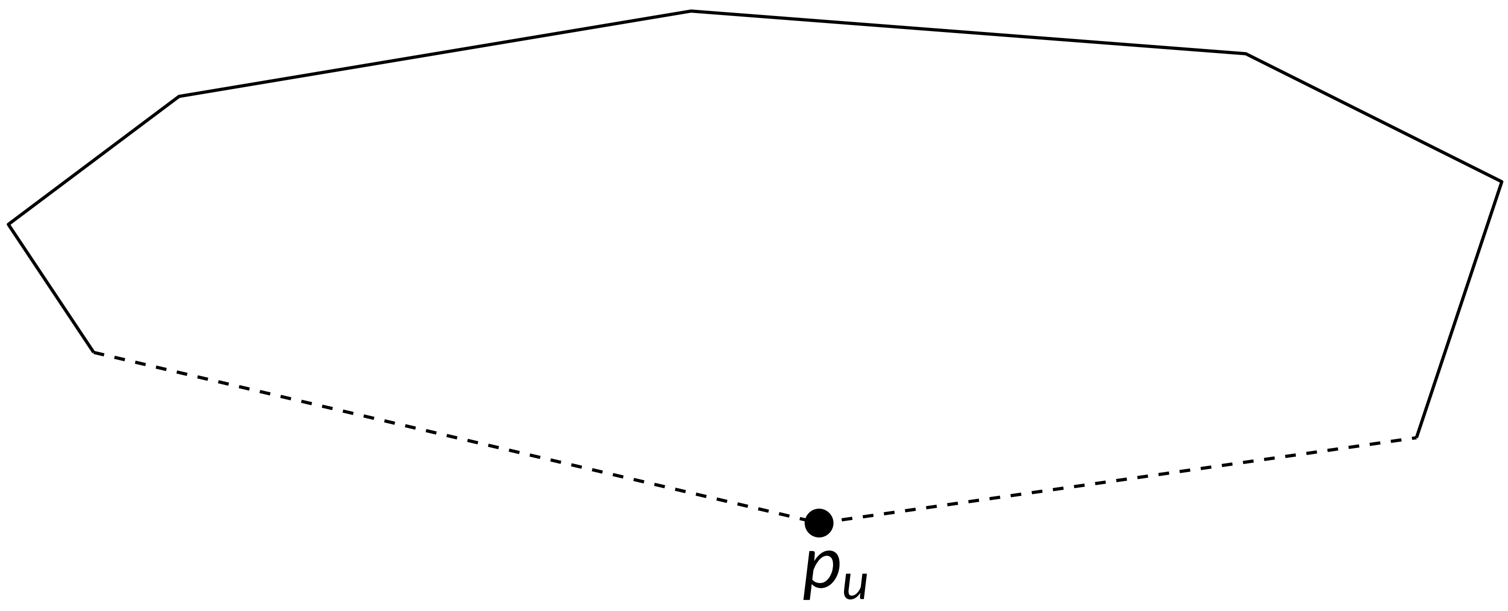
Smallest disk: defined by at most three points



Since there are at most two substrings corresponding to a disk, the smallest disk is defined by a point and a substring.

Search Algorithm

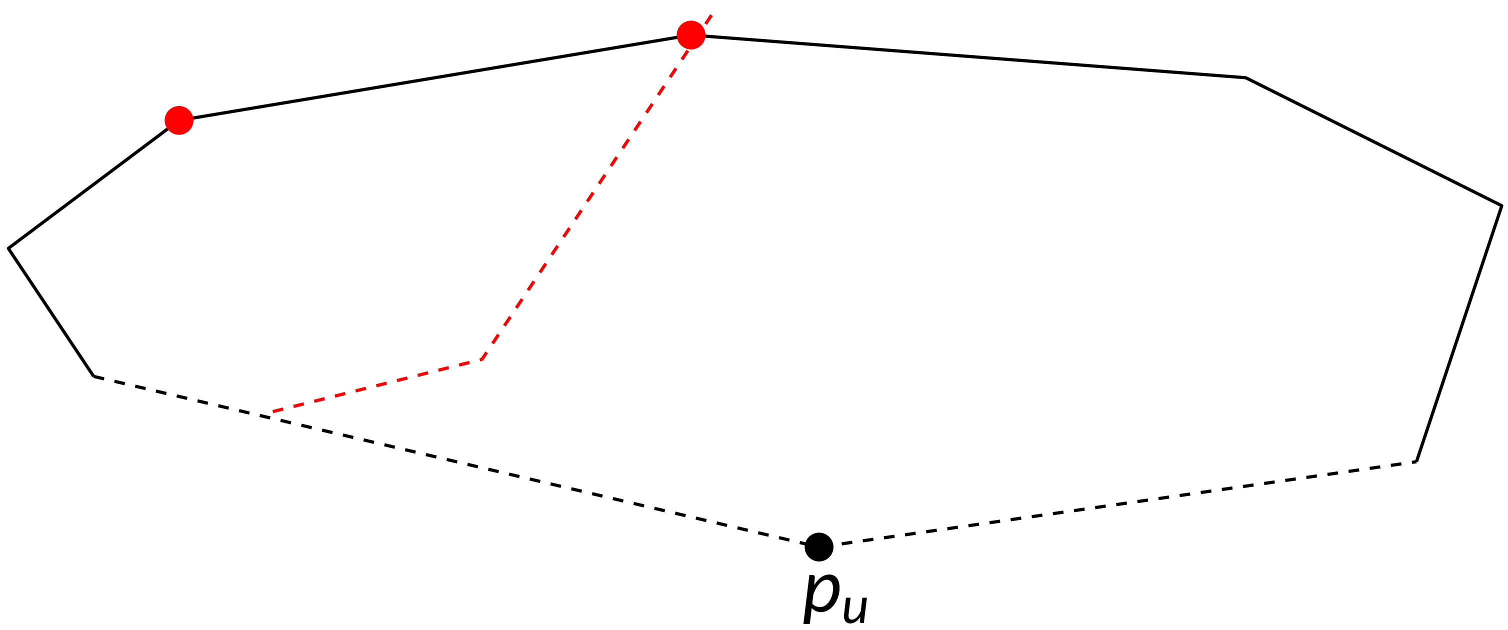
For each fixed p_u, we only consider $O(n)$ number of substrings.



$O(n^2)$ candidates of radii are computed in $O(n^2 \log n)$ time as intervals, interval $(r_L, r_U]$ that contains r^* can be computed in $O(T_S \log n)$ time.

Search Algorithm

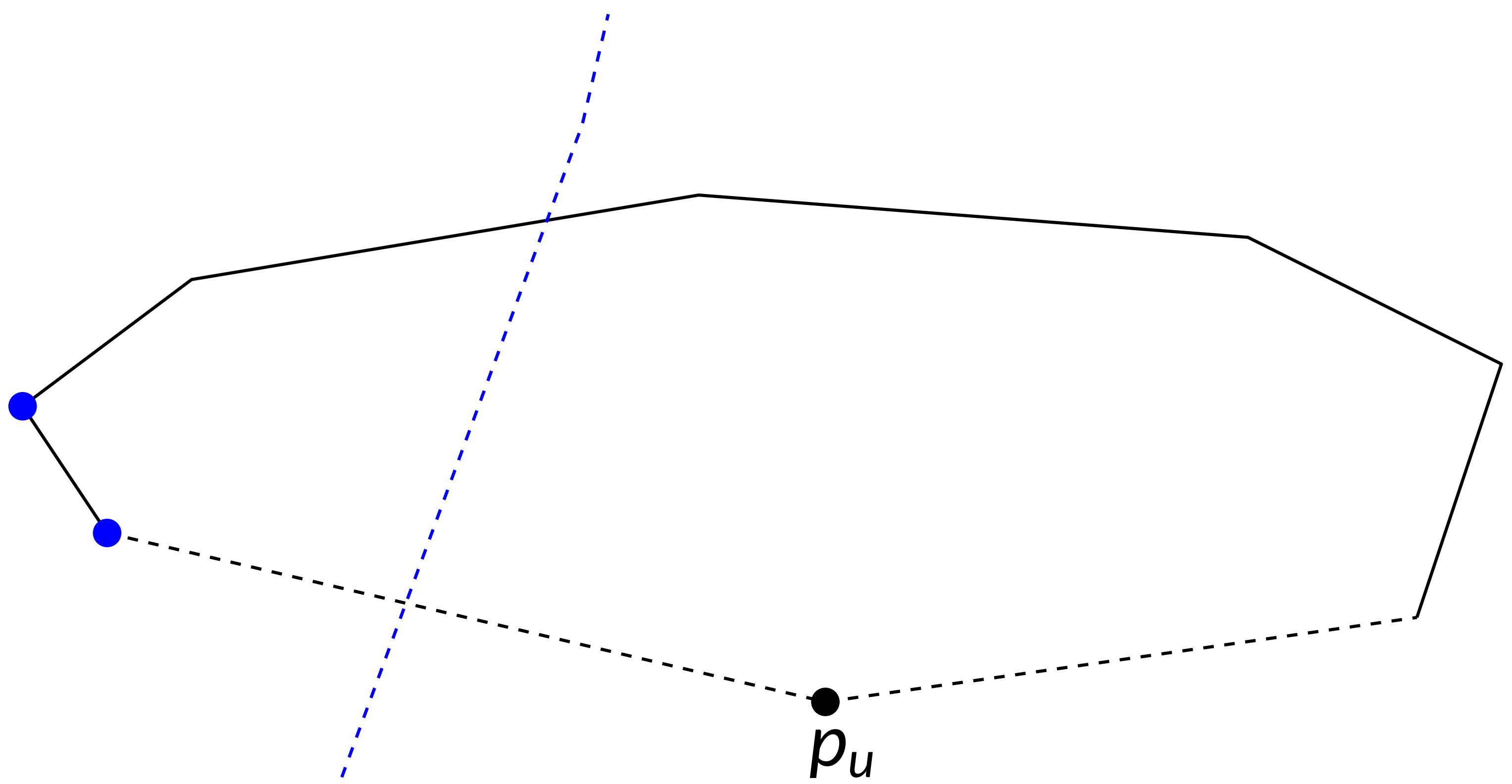
For each fixed p_u, we only consider $O(n)$ number of substrings.



$O(n^2)$ candidates of radii are computed in $O(n^2 \log n)$ time as intervals, interval $(r_L, r_U]$ that contains r^* can be computed in $O(T_S \log n)$ time.

Search Algorithm

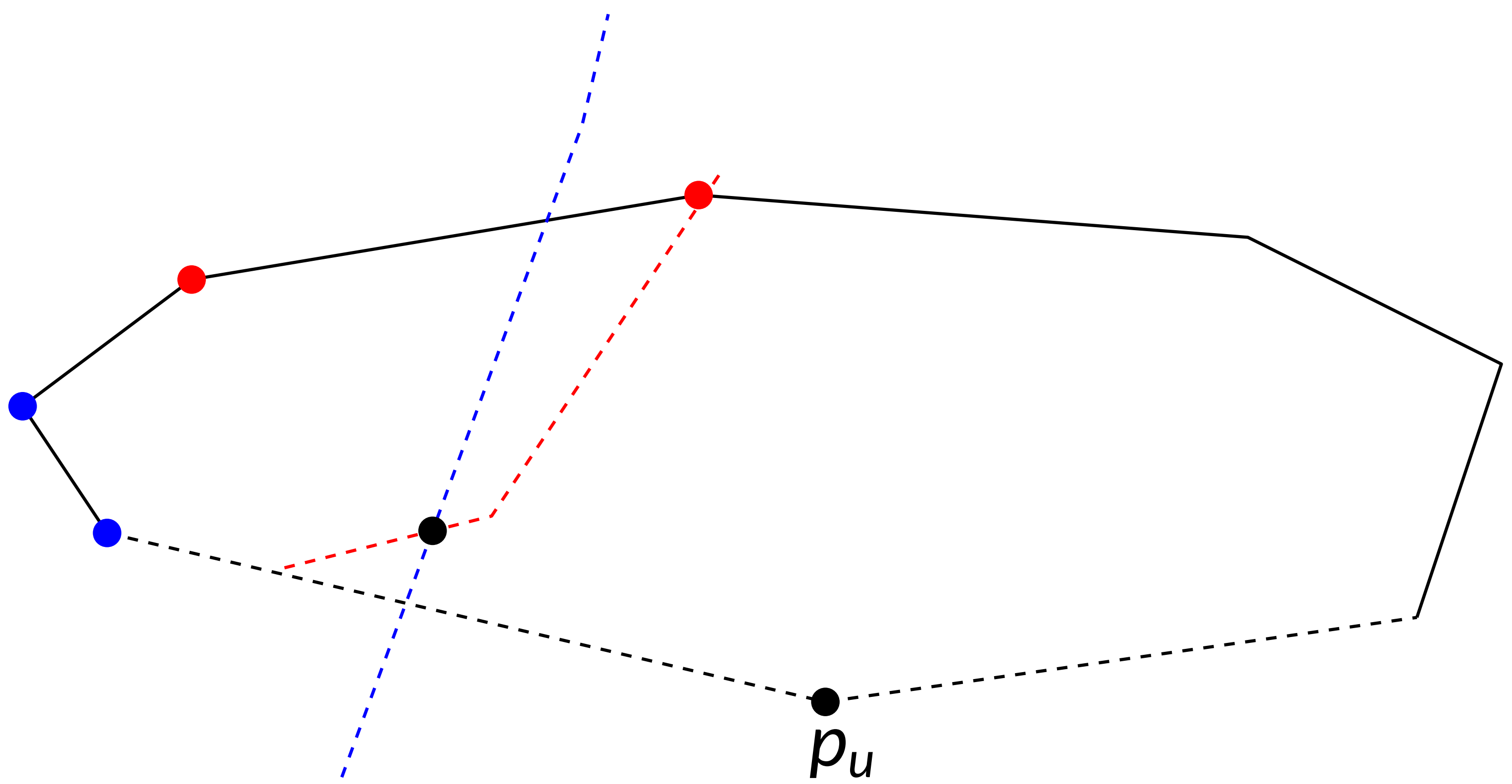
For each fixed p_u, we only consider $O(n)$ number of substrings.



$O(n^2)$ candidates of radii are computed in $O(n^2 \log n)$ time as intervals,
interval $(r_L, r_U]$ that contains r^* can be computed in $O(T_S \log n)$ time.

Search Algorithm

For each fixed p_u, we only consider $O(n)$ number of substrings.

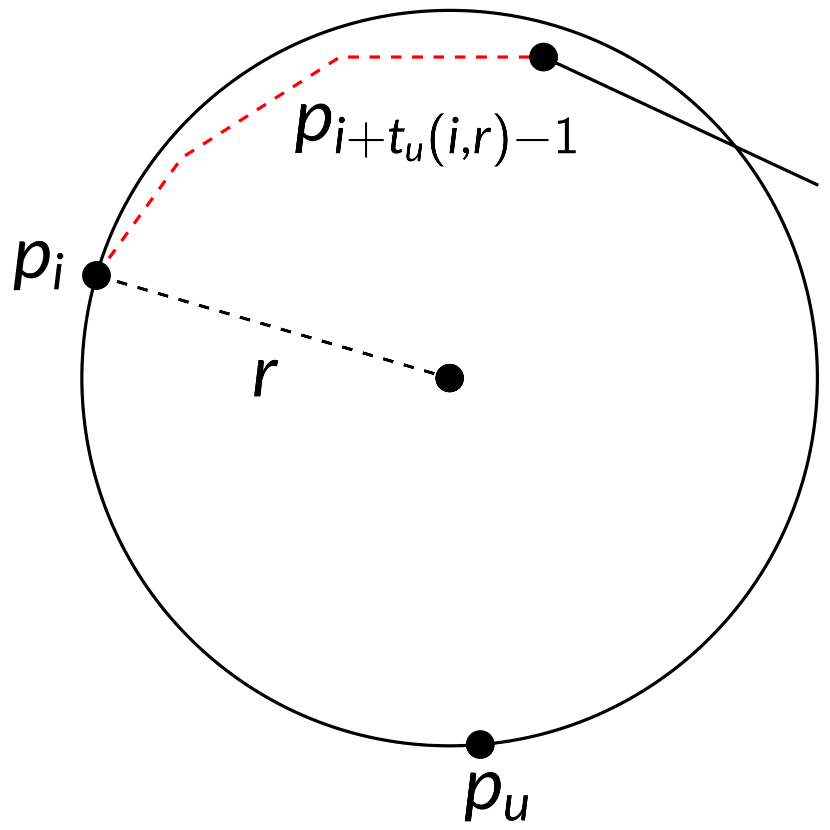


$O(n^2)$ candidates of radii are computed in $O(n^2 \log n)$ time as intervals, interval $(r_L, r_U]$ that contains r^* can be computed in $O(T_S \log n)$ time.

Search Algorithm

$r_u(i,t)$: radius of the smallest disk covering $P(i,t)$ and p_u

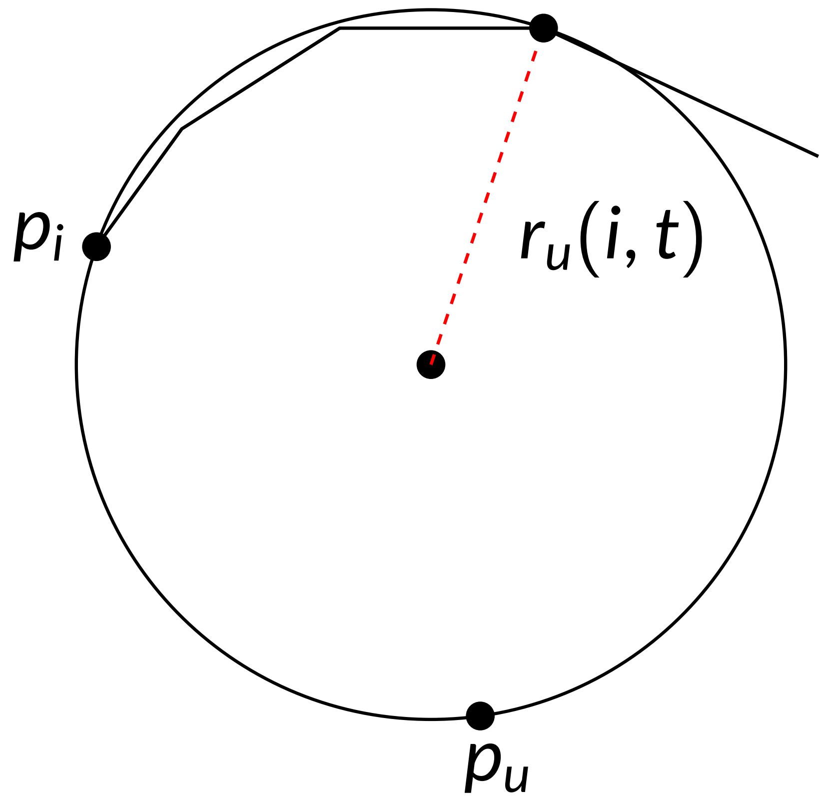
$t_u(i,r) = \max\{t | r_u(i,t) \leqslant r\}$



Search Algorithm

$r_u(i,t)$: radius of the smallest disk covering $P(i,t)$ and p_u

$t_u(i,r) = \max\{t | r_u(i,t) \leqslant r\}$



Lemma. For any fixed integers u and i and any fixed $r \in (r_L, r_U]$, we can compute $t_u(i,r)$ in $O(\log n)$ time after $O(n)$-time preprocessing.

Search Algorithm

Using Cole's parametric search with $O(n^2)$ processors,

we can compute $t_u(i, r^*)$ for all i and u in $O(T_S \log n)$ time

$r_u(i, t_u(i, r^*))$ for all i and u in $O(n^2 \log n)$ time

From these $O(n^2)$ candidates, find r^* using binary search in $O(T_S \log n)$ time.

Also works under the Minkowski distance of order p for any fixed integer p

Thank You!