# Point Set Pattern Matching under Rigid Motion

Arijit Bishnu, Sandip Das, Subhas C. Nandy
and Bhargab B. Bhattacharya

Advanced Computing and Microelectronics Unit,
Indian Statistical Institute, Kolkata, India.

October 25, 2010

# Outline

# Outline

# Introduction

## Introduction

- A problem in pattern matching is to design efficient algorithms for testing how closely a query set $Q$ of $k$ points resembles a sample set $P$ of $n$ points, where $k \leq n$.
- In image processing, computer vision and related applications like finger print matching, point sets represent some spatial features.
- The problem has several variants [Alt and Guibas, 1999] based on:
  - class of allowable transformations
  - exact or approximate matching
  - equal cardinality, subset, or largest common point set matching

# Problem Variants

## Allowable Transformations

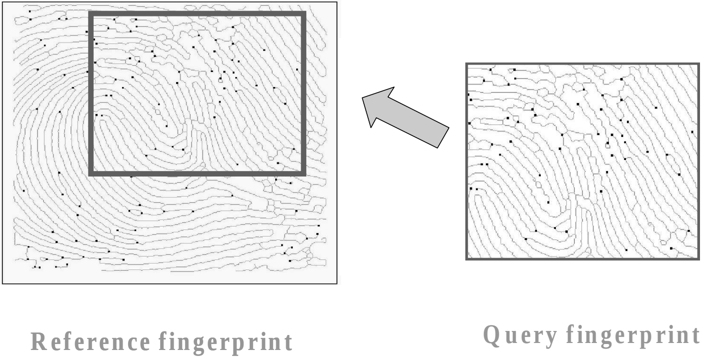
- Translation
- Translation and Rotation - Rigid Motion Transform
- Translation, Rotation and Scaling - Similarity Transform

Distances are preserved in Translation and rigid motion.

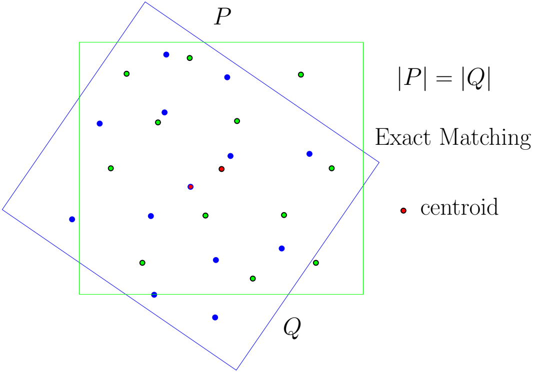
## Types of Matching

- Exact matching: points in query set match exactly with points in sample set after the requisite transform.
- Approximate matching: points in query set after the requisite transform go to a neighborhood of points in the sample set.

# Motivation



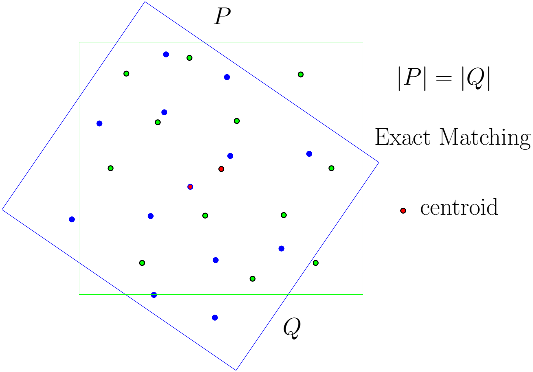
# Exact Matching under Rigid Motion




## Anchor

- Anchor centroids and match distances.
- But anchoring centroids do not help when we match $Q$ with a set of points $P' \subseteq P$.

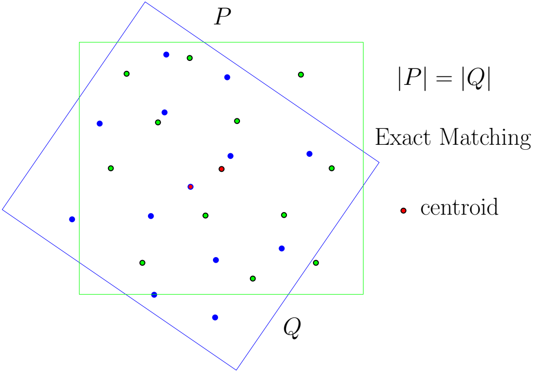
# Exact Matching under Rigid Motion




## Anchor

- Anchor centroids and match distances.
- But anchoring centroids do not help when we match $Q$ with a set of points $P' \subseteq P$.

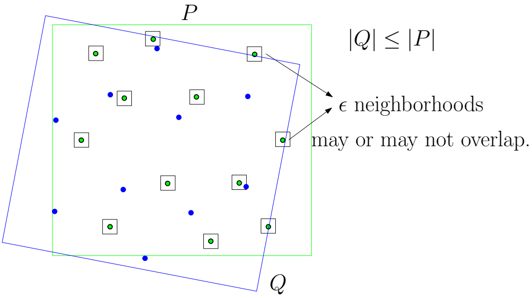
# Exact Matching under Rigid Motion




## Anchor

- Anchor centroids and match distances.
- But anchoring centroids do not help when we match $Q$ with a set of points $P' \subseteq P$.

# Approximate Partial Matching under Rigid Motion




Given two point sets $P$ and $Q$ ($|P| = |Q|$), check if there is a bijection $\ell : Q \rightarrow P$ and a congruence $T$, such that $T(q) \in U_\epsilon(\ell(q)), \forall\, q \in Q'$, where $Q' \subseteq Q$, and $U_\epsilon(p)$ denotes the closed $\epsilon$-neighborhood of a point $p \in P$.

# Outline

# Previous Work

## Exact Matching

- Rezende and Lee (1995) proposed an $O(kn^d)$ time algorithm for partial point set pattern matching, where
  $d$ (* dimension of the plane *)
  $n = |P|$ (* size of sample set *)
  $k = |Q|$ (* size of pattern set *).
  It allows translation, rotation and scaling.
- Akutsu, Tamaki and Tokuyama (1998) proposed an $O(kn^{\frac{4}{3}} + \mathcal{A})$ time algorithm for testing the congruence in 2D,
  where $\mathcal{A}$ = time complexity for locating $r$-th smallest distance among a set of $n$ points in 2D.
  = $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ (on an average).
- Akutsu, Tamaki and Tokuyama (1998) proposed a Las Vegas expected time algorithm of time complexity $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n + \min(k^{0.77} n^{1.43} \log n, n^{\frac{4}{3}} k \log n))$ using parametric search.

# Previous Work

## Exact Matching

- Rezende and Lee (1995) proposed an $O(kn^d)$ time algorithm for partial point set pattern matching, where
  $d$ (* dimension of the plane *)
  $n = |P|$ (* size of sample set *)
  $k = |Q|$ (* size of pattern set *).
  It allows translation, rotation and scaling.
- Akutsu, Tamaki and Tokuyama (1998) proposed an $O(kn^{\frac{4}{3}} + \mathcal{A})$ time algorithm for testing the congruence in 2D,
  where $\mathcal{A}$ = time complexity for locating $r$-th smallest distance among a set of $n$ points in 2D.
  = $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ (on an average).
- Akutsu, Tamaki and Tokuyama (1998) proposed a Las Vegas expected time algorithm of time complexity $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n + \min(k^{0.77} n^{1.43} \log n, n^{\frac{4}{3}} k \log n))$ using parametric search.

# Previous Work

## Exact Matching

- Rezende and Lee (1995) proposed an $O(kn^d)$ time algorithm for partial point set pattern matching, where
  $d$ (* dimension of the plane *)
  $n = |P|$ (* size of sample set *)
  $k = |Q|$ (* size of pattern set *).
  It allows translation, rotation and scaling.
- Akutsu, Tamaki and Tokuyama (1998) proposed an $O(kn^{\frac{4}{3}} + \mathcal{A})$ time algorithm for testing the congruence in 2D,
  where $\mathcal{A}$ = time complexity for locating $r$-th smallest distance among a set of $n$ points in 2D.
  = $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ (on an average).
- Akutsu, Tamaki and Tokuyama (1998) proposed a Las Vegas expected time algorithm of time complexity $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n + \min(k^{0.77} n^{1.43} \log n, n^{\frac{4}{3}} k \log n))$ using parametric search.

# Previous Work

## Exact Matching

- Rezende and Lee (1995) proposed an $O(kn^d)$ time algorithm for partial point set pattern matching, where
  $d$ (* dimension of the plane *)
  $n = |P|$ (* size of sample set *)
  $k = |Q|$ (* size of pattern set *).
  It allows translation, rotation and scaling.
- Akutsu, Tamaki and Tokuyama (1998) proposed an $O(kn^{\frac{4}{3}} + \mathcal{A})$ time algorithm for testing the congruence in 2D,
  where $\mathcal{A}$ = time complexity for locating $r$-th smallest distance among a set of $n$ points in 2D.
  = $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ (on an average).
- Akutsu, Tamaki and Tokuyama (1998) proposed a Las Vegas expected time algorithm of time complexity
  $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n + \min(k^{0.77} n^{1.43} \log n, n^{\frac{4}{3}} k \log n))$ using parametric search.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

## Approximate Matching (for $k = n$ case)

- Alt et al. (1988) designed a general algorithm of $O(n^8)$ time that works for overlapping and non-overlapping $\epsilon$-circles and $\epsilon$-boxes.
- They use a geometric fact and bipartite graph matching.
- A valid matching of $Q$ with $P$ exists iff there is a matching where $q_i, q_j \in Q$ are matched exactly to the boundaries of $U_\epsilon(p_\alpha)$, $U_\epsilon(p_\beta)$ of two points $p_\alpha, p_\beta \in P$.
- The algorithms are of high time complexity and involve computing the intersection of complex algebraic curves.
- Heffernan and Schirra (1994) show that this $O(n^8)$ algorithm is indeed optimal for $\epsilon$-circles.

# Previous Work

**Approximate Matching (for $k \neq n$ case)**

- Chew et al. (1997) proposed an $(n^2k^3 \log^2 kn)$ time algorithm for approximate partial point set pattern matching under rigid motion.

# Previous Work

## Approximate Matching (for $k \neq n$ case)

- Chew et al. (1997) proposed an $(n^2k^3 \log^2 kn)$ time algorithm for approximate partial point set pattern matching under rigid motion.

# Outline

# Exact Point Set Pattern Matching

## The Problem

**Input:** $P = \{p_1, p_2, \ldots, p_n\}$ (* Sample Set *)
$Q = \{q_1, q_2, \ldots, q_k\}$ (* Query Set *),
$k \leq n$.

**Output:** A subset of points in $P$ that are matched with the points in $Q$ if match exists under rigid motion.

## Trivial Algorithm

Choose all possible $\binom{n}{k}$ subset of $P$ and test for a match under rigid motion.

# Exact Point Set Pattern Matching

## The Problem

**Input:** $P = \{p_1, p_2, \ldots, p_n\}$ (* Sample Set *)
$Q = \{q_1, q_2, \ldots, q_k\}$ (* Query Set *),
$k \leq n$.

**Output:** A subset of points in $P$ that are matched with the points in $Q$ if match exists under rigid motion.

## Trivial Algorithm

Choose all possible $\binom{n}{k}$ subset of $P$ and test for a match under rigid motion.

# Exact Point Set Pattern Matching

## An obvious improvement

Preprocessing: Use circular sorting to create a data structure attached with each point in $P$.

Space: $O(n^2)$

Time: $O(n^2)$.

Query:
- Sort the query point set angularly.
- Anchor a pair of query point with each pair of sample point.
- It determines the rotation angle and scaling.
- Match the other query points with a subset of sample points.

Time: $O(k \log k + kn(n-1))$

# Exact Point Set Pattern Matching

## An obvious improvement

Preprocessing: Use circular sorting to create a data structure attached with each point in $P$.

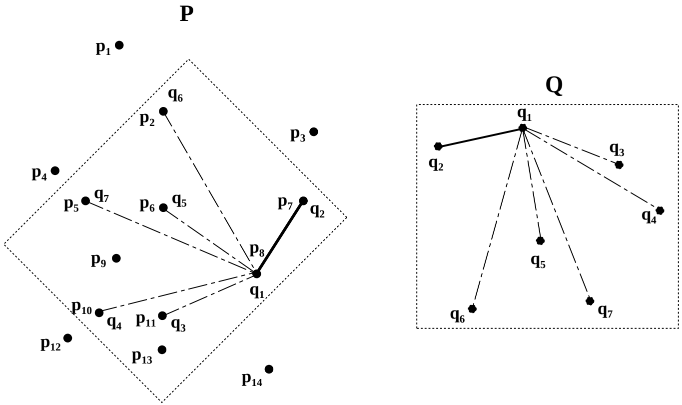
Space: $O(n^2)$

Time: $O(n^2)$.

Query:
- Sort the query point set angularly.
- Anchor a pair of query point with each pair of sample point.
- It determines the rotation angle and scaling.
- Match the other query points with a subset of sample points.

Time: $O(k \log k + kn(n-1))$

# Demonstration



# An Efficient Algorithm for Rigid Motion

**Fact 1**

All the $\binom{k}{2}$ distances must occur in the $\binom{n}{2}$ distances in $P$

**Fact 2**

[Szekely 1997] In a sample set $P$ of size $n$, the maximum number of equidistant pairs of points is $O(n^{\frac{4}{3}})$ in the worst case.

# An Efficient Algorithm for Rigid Motion

**Fact 1**

All the $\binom{k}{2}$ distances must occur in the $\binom{n}{2}$ distances in $P$

**Fact 2**

[Szekely 1997] In a sample set $P$ of size $n$, the maximum number of equidistant pairs of points is $O(n^{\frac{4}{3}})$ in the worst case.

# An Efficient Algorithm for Rigid Motion

## Preprocessing

- Sort the $\binom{n}{2}$ distances of $P$ distances in $P$.
- Create a height balanced binary tree $\mathcal{T}$ with distinct distances.
- Attach an array $\chi_\delta$ with each element $\delta$ of the tree. Its each element is a triple $(p_i, p_j, \psi_{ij})$, where $\psi_{ij}$ is the angle of the line $(p_i, p_j)$ with $x$-axis.
- Each point $p_i \in P$ is attached with a height-balanced binary tree $\mathcal{S}_i$. Its elements are tuples $(r_{ij}, \theta_{ij})$.

Time Complexity: $O(n^2 \log n)$
Space Complexity: $O(n^2)$

# An Efficient Algorithm for Rigid Motion

## Preprocessing

- Sort the $\binom{n}{2}$ distances of $P$ distances in $P$.
- Create a height balanced binary tree $\mathcal{T}$ with distinct distances.
- Attach an array $\chi_\delta$ with each element $\delta$ of the tree. Its each element is a triple $(p_i, p_j, \psi_{ij})$, where $\psi_{ij}$ is the angle of the line $(p_i, p_j)$ with $x$-axis.
- Each point $p_i \in P$ is attched with a height-balanced binary tree $\mathcal{S}_i$. Its elements are tuples $(r_{ij}, \theta_{ij})$.

Time Complexity: $O(n^2 \log n)$
Space Complexity: $O(n^2)$

# An Efficient Algorithm for Rigid Motion

## Query

- Take two points $(q_1, q_2)$, and check whether $\lambda(q_1, q_2) \in \mathcal{T}$. If not, report no match found.
- Let $\lambda(q_1, q_2) = \delta$.
- We consider each member $\lambda(p_i, p_j) \in \chi_\delta$.
- Anchor $(q_1, q_2)$ with $(p_i, p_j)$, and search in $\mathcal{S}_i$ for the presence of a match.

## Time Complexity

For each $\lambda(p_i p_j) = \delta$, searching for a match needs $O(k \log n)$ time. So,
Time Complexity: $O(n^{\frac{4}{3}} k \log n)$

# An Efficient Algorithm for Rigid Motion

## Query

- Take two points $(q_1, q_2)$, and check whether $\lambda(q_1, q_2) \in \mathcal{T}$. If not, report no match found.
- Let $\lambda(q_1, q_2) = \delta$.
- We consider each member $\lambda(p_i, p_j) \in \chi_\delta$.
- Anchor $(q_1, q_2)$ with $(p_i, p_j)$, and search in $\mathcal{S}_i$ for the presence of a match.

## Time Complexity

For each $\lambda(p_i p_j) = \delta$, searching for a match needs $O(k \log n)$ time. So,
Time Complexity: $O(n^{\frac{4}{3}} k \log n)$

# An Important Note

Though the worst case number of equidistant pairs in a point set of size $n$ is $O(n^{\frac{4}{3}})$, in a random instance actually the number is very less.

| | | | | |
|---|---|---|---|---|
| Number of points (n) | 100 | 200 | 500 | 1000 |
| $\lceil n^{\frac{4}{3}} \rceil$ | 465 | 1170 | 3969 | 10001 |
| Maximum number of Equidistant pairs observed | 4 | 6 | 20 | 53 |

# Experimental Results

| No. of points in sample | No. of points in query | No. of anchoring | CPU time for Rezende and Lee | CPU time for our Algorithm | % savings |
|---|---|---|---|---|---|
| 200 | 50 | 1 | 730.0 | 22.0 | 97.0 |
| | 100 | 1 | 807.4 | 23.4 | 97.0 |
| | 150 | 2 | 867.6 | 31.2 | 96.0 |
| 500 | 100 | 2 | 1905.0 | 32.2 | 98.0 |
| | 200 | 1 | 1937.2 | 23.0 | 99.0 |
| | 400 | 2 | 1952.8 | 32.0 | 98.0 |
| 1000 | 300 | 12 | 4037.6 | 129.0 | 97.0 |
| | 500 | 7 | 4054.0 | 76.0 | 98.0 |
| | 800 | 11 | 4098.0 | 11.0 | 97.0 |

# Outline

# Approximate Point Set Pattern Matching

## Our Effort

We assume that

- the $\epsilon$-neighborhoods are axis-parallel squares of side length $\epsilon$.
- $P$ is well separated, i.e. each pair of points $p, p' \in P$ satisfy either $|x(p) - x(p')| \geq \epsilon$ or $|y(p) - y(p')| \geq 3\epsilon$ or both.

# Approximate Point Set Pattern Matching

## Our Effort

We assume that

- the $\epsilon$-neighborhoods are axis-parallel squares of side length $\epsilon$.
- $P$ is well separated, i.e. each pair of points $p, p' \in P$ satisfy either $|x(p) - x(p')| \geq \epsilon$ or $|y(p) - y(p')| \geq 3\epsilon$ or both.

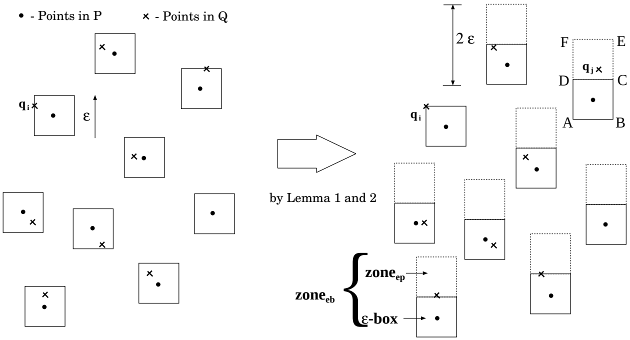
# Approximate Point Set Pattern Matching

## Our Effort

We assume that

- the $\epsilon$-neighborhoods are axis-parallel squares of side length $\epsilon$.
- $P$ is well separated, i.e. each pair of points $p, p' \in P$ satisfy either $|x(p) - x(p')| \geq \epsilon$ or $|y(p) - y(p')| \geq 3\epsilon$ or both.

# A Necessary Characterization for a Matching



**Lemma (an iff condition)**

If $\exists$ a transformation $T(Q)$ for the said match, then $\exists$ another transformation $T'(Q)$, such that one point of $Q$ lies on the left boundary of the $\epsilon$-box of a point in $P$, and one point of $Q$ lies on the top boundary of the $\epsilon$-box of a point in $P$.

**Definition**

Consider an $\epsilon$-box $ABCD$ around $p \in P$. The *extended $\epsilon$-box* of $p$ is a $\epsilon \times 2\epsilon$ box formed by attaching another $\epsilon \times \epsilon$ square $CDFE$ above the $\epsilon$-box $ABCD$. $CDFE$ is called the *extended portion* of the $\epsilon$-box.

### Lemma (an iff condition)

If $\exists$ a transformation $T(Q)$ for the said match, then $\exists$ another transformation $T'(Q)$, such that one point of $Q$ lies on the left boundary of the $\epsilon$-box of a point in $P$, and one point of $Q$ lies on the top boundary of the $\epsilon$-box of a point in $P$.

### Definition

Consider an $\epsilon$-box *ABCD* around $p \in P$. The *extended $\epsilon$-box* of $p$ is a $\epsilon \times 2\epsilon$ box formed by attaching another $\epsilon \times \epsilon$ square *CDFE* above the $\epsilon$-box *ABCD*. *CDFE* is called the *extended portion* of the $\epsilon$-box.

## Lemma (an if condition)

If $\exists$ a transformation $T(Q)$ for the said match, $\exists$ another transformation $T'(Q)$, such that

- a point $q \in Q$ lies at the top-left corner of the $\epsilon$-box of a point in $P$.
- at least one point lies in the extended portion of the $\epsilon$-box
- each of the remaining members in $Q$ lie in the extended $\epsilon$-box

# Anchorings for Finding the Transformation

## Anchoring for Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

- Two unknown parameters, $t_x$ and $t_y$. So, one anchoring is needed.

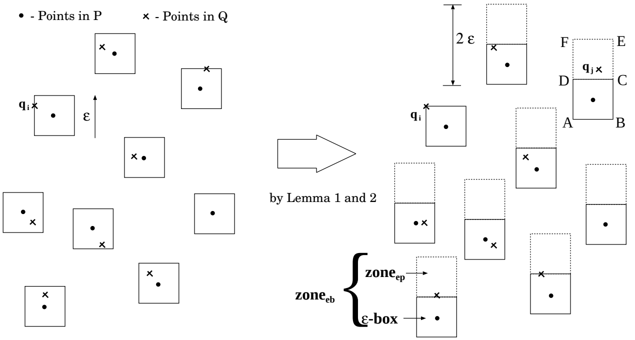
## Anchoring for Rigid Motion

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

- Three unknown parameters, $t_x$, $t_y$ and $\theta$. So, two anchorings are needed.

# Outline

# Matching under Translation



# Translation

## The Algorithm

- Position $q$ at the top-left corner of the $\epsilon$-box of $p$, and check whether each point in $Q \setminus \{q\}$ lies inside the extended box of some point of $P$.
- If the above checking returns false, then no match exists with $q$ at the top-left corner of the $\epsilon$-box of $p$.
- If it returns true, then the points in $Q \setminus \{q\}$ can be partitioned into $Q_1$ and $Q_2$. Each $q_1 \in Q_1$ lies in the $\epsilon$-box of some point in $P$. Each $q_2 \in Q_2$ lies in the extended portion of $\epsilon$-box of some member in $P$.
- Push points in $Q_2$ down such that points in $Q_1$ do not go out.

# Translation

## The Algorithm

- Position $q$ at the top-left corner of the $\epsilon$-box of $p$, and check whether each point in $Q \setminus \{q\}$ lies inside the extended box of some point of $P$.
- If the above checking returns false, then no match exists with $q$ at the top-left corner of the $\epsilon$-box of $p$.
- If it returns true, then the points in $Q \setminus \{q\}$ can be partitioned into $Q_1$ and $Q_2$. Each $q_1 \in Q_1$ lies in the $\epsilon$-box of some point in $P$. Each $q_2 \in Q_2$ lies in the extended portion of $\epsilon$-box of some member in $P$.
- Push points in $Q_2$ down such that points in $Q_1$ do not go out.

# Translation

## The Algorithm

- Position $q$ at the top-left corner of the $\epsilon$-box of $p$, and check whether each point in $Q \setminus \{q\}$ lies inside the extended box of some point of $P$.
- If the above checking returns false, then no match exists with $q$ at the top-left corner of the $\epsilon$-box of $p$.
- If it returns true, then the points in $Q \setminus \{q\}$ can be partitioned into $Q_1$ and $Q_2$. Each $q_1 \in Q_1$ lies in the $\epsilon$-box of some point in $P$. Each $q_2 \in Q_2$ lies in the extended portion of $\epsilon$-box of some member in $P$.
- Push points in $Q_2$ down such that points in $Q_1$ do not go out.

# Translation

## The Algorithm

- Position $q$ at the top-left corner of the $\epsilon$-box of $p$, and check whether each point in $Q \setminus \{q\}$ lies inside the extended box of some point of $P$.
- If the above checking returns false, then no match exists with $q$ at the top-left corner of the $\epsilon$-box of $p$.
- If it returns true, then the points in $Q \setminus \{q\}$ can be partitioned into $Q_1$ and $Q_2$. Each $q_1 \in Q_1$ lies in the $\epsilon$-box of some point in $P$. Each $q_2 \in Q_2$ lies in the extended portion of $\epsilon$-box of some member in $P$.
- Push points in $Q_2$ down such that points in $Q_1$ do not go out.

# The Analysis

## The Algorithm

- Maintain a planar straight line graph (PSLG) data structure with the $\epsilon$-boxes around each point in $P$.
- Anchoring a point $q \in Q$ with the top-left corner of an $\epsilon$-box, we perform $k$ point location queries in the PSLG. This needs $O(k \log n)$ time.
- Pushing points down take another $O(k)$ time.
- There can be $O(nk)$ anchorings in the worst case.

## Theorem

The worst case time complexity of the approximate matching of $Q$ with a $k$-subset of $P$ in 2D when only translation is considered is $O(nk^2 \log n)$.

# The Analysis

## The Algorithm

- Maintain a planar straight line graph (PSLG) data structure with the $\epsilon$-boxes around each point in $P$.
- Anchoring a point $q \in Q$ with the top-left corner of an $\epsilon$-box, we perform $k$ point location queries in the PSLG. This needs $O(k \log n)$ time.
- Pushing points down take another $O(k)$ time.
- There can be $O(nk)$ anchorings in the worst case.

## Theorem

The worst case time complexity of the approximate matching of $Q$ with a $k$-subset of $P$ in 2D when only translation is considered is $O(nk^2 \log n)$.

# The Analysis

## The Algorithm

- Maintain a planar straight line graph (PSLG) data structure with the $\epsilon$-boxes around each point in $P$.
- Anchoring a point $q \in Q$ with the top-left corner of an $\epsilon$-box, we perform $k$ point location queries in the PSLG. This needs $O(k \log n)$ time.
- Pushing points down take another $O(k)$ time.
- There can be $O(nk)$ anchorings in the worst case.

## Theorem

The worst case time complexity of the approximate matching of $Q$ with a $k$-subset of $P$ in 2D when only translation is considered is $O(nk^2 \log n)$.

# The Analysis

## The Algorithm

- Maintain a planar straight line graph (PSLG) data structure with the $\epsilon$-boxes around each point in $P$.
- Anchoring a point $q \in Q$ with the top-left corner of an $\epsilon$-box, we perform $k$ point location queries in the PSLG. This needs $O(k \log n)$ time.
- Pushing points down take another $O(k)$ time.
- There can be $O(nk)$ anchorings in the worst case.

## Theorem

The worst case time complexity of the approximate matching of $Q$ with a $k$-subset of $P$ in 2D when only translation is considered is $O(nk^2 \log n)$.

# The Analysis

## The Algorithm

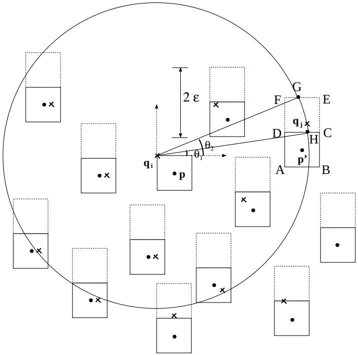
- Maintain a planar straight line graph (PSLG) data structure with the $\epsilon$-boxes around each point in $P$.
- Anchoring a point $q \in Q$ with the top-left corner of an $\epsilon$-box, we perform $k$ point location queries in the PSLG. This needs $O(k \log n)$ time.
- Pushing points down take another $O(k)$ time.
- There can be $O(nk)$ anchorings in the worst case.

## Theorem

The worst case time complexity of the approximate matching of $Q$ with a $k$-subset of $P$ in 2D when only translation is considered is $O(nk^2 \log n)$.

# Outline

# Matching under Rigid Motion



## The Algorithm

- Consider the $k - 1$ concentric circles $\mathcal{C}_{ij}\ \forall\ q_j \in Q \setminus \{q_i\}$. Each circle $\mathcal{C}_{ij}$ intersects some extended $\epsilon$-boxes.
- As $P$ is well-separated, these intersections contribute a set of non-overlapping arcs that define a circular arc graph $G$.
- Each circle $\mathcal{C}_{ij}$ may intersect $O(n)$ boxes making $O(nk)$ nodes in $G$. So, there can be $O(nk)$ cliques of size $k - 1$.
- Each clique $\chi$ corresponding to an anchoring of $q$ represents an angular interval $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ such that all points in $Q \setminus q$ lie in some extended $\epsilon$-box.

## The Algorithm

- Consider the $k-1$ concentric circles $\mathcal{C}_{ij}\ \forall\ q_j \in Q \setminus \{q_i\}$. Each circle $\mathcal{C}_{ij}$ intersects some extended $\epsilon$-boxes.
- As $P$ is well-separated, these intersections contribute a set of non-overlapping arcs that define a circular arc graph $G$.
- Each circle $\mathcal{C}_{ij}$ may intersect $O(n)$ boxes making $O(nk)$ nodes in $G$. So, there can be $O(nk)$ cliques of size $k-1$.
- Each clique $\chi$ corresponding to an anchoring of $q$ represents an angular interval $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ such that all points in $Q \setminus q$ lie in some extended $\epsilon$-box.

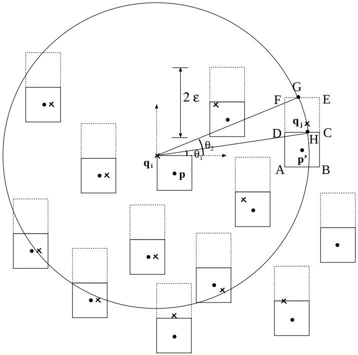
## The Algorithm

- Consider the $k-1$ concentric circles $\mathcal{C}_{ij}\ \forall\ q_j \in Q \setminus \{q_i\}$. Each circle $\mathcal{C}_{ij}$ intersects some extended $\epsilon$-boxes.
- As $P$ is well-separated, these intersections contribute a set of non-overlapping arcs that define a circular arc graph $G$.
- Each circle $\mathcal{C}_{ij}$ may intersect $O(n)$ boxes making $O(nk)$ nodes in $G$. So, there can be $O(nk)$ cliques of size $k-1$.
- Each clique $\chi$ corresponding to an anchoring of $q$ represents an angular interval $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ such that all points in $Q \setminus q$ lie in some extended $\epsilon$-box.

## The Algorithm

- Consider the $k - 1$ concentric circles $\mathcal{C}_{ij}\ \forall\ q_j \in Q \setminus \{q_i\}$. Each circle $\mathcal{C}_{ij}$ intersects some extended $\epsilon$-boxes.
- As $P$ is well-separated, these intersections contribute a set of non-overlapping arcs that define a circular arc graph $G$.
- Each circle $\mathcal{C}_{ij}$ may intersect $O(n)$ boxes making $O(nk)$ nodes in $G$. So, there can be $O(nk)$ cliques of size $k - 1$.
- Each clique $\chi$ corresponding to an anchoring of $q$ represents an angular interval $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ such that all points in $Q \setminus q$ lie in some extended $\epsilon$-box.

# One of the $k - 1$ concentric circles shown.



# Processing Each Clique

## Processing Cliques

- Each clique $\chi$ represents an angular interval (an arc) $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$. For any angle of rotation $\theta \in \mathcal{I}^*$, each of the $k-1$ point of $Q \setminus q_i$ lies inside $k-1$ disjoint extended $\epsilon$-boxes.
- Partition the arcs corresponding to the points in $Q \setminus \{q_i\}$ into two subsets $Q_1$ and $Q_2$. Arcs in $Q_1$ are all inside the $\epsilon$-boxes, and those in $Q_2$ are all inside the extended portion of the $\epsilon$-boxes.
- Now we need to push down the points so that a match, if it exists, can be found.
- We do this for all arcs together.

# Processing Each Clique

## Processing Cliques

- Each clique $\chi$ represents an angular interval (an arc) $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$. For any angle of rotation $\theta \in \mathcal{I}^*$, each of the $k-1$ point of $Q \setminus q_i$ lies inside $k-1$ disjoint extended $\epsilon$-boxes.
- Partition the arcs corresponding to the points in $Q \setminus \{q_i\}$ into two subsets $Q_1$ and $Q_2$. Arcs in $Q_1$ are all inside the $\epsilon$-boxes, and those in $Q_2$ are all inside the extended portion of the $\epsilon$-boxes.
- Now we need to push down the points so that a match, if it exists, can be found.
- We do this for all arcs together.

# Processing Each Clique

## Processing Cliques

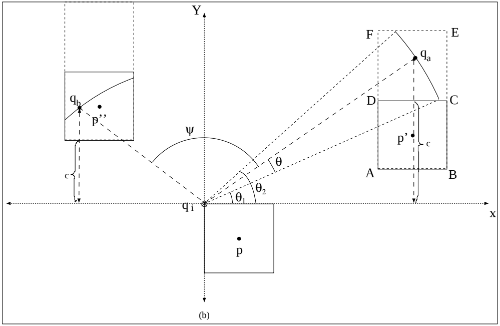
- Each clique $\chi$ represents an angular interval (an arc) $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$. For any angle of rotation $\theta \in \mathcal{I}^*$, each of the $k-1$ point of $Q \setminus q_i$ lies inside $k-1$ disjoint extended $\epsilon$-boxes.
- Partition the arcs corresponding to the points in $Q \setminus \{q_i\}$ into two subsets $Q_1$ and $Q_2$. Arcs in $Q_1$ are all inside the $\epsilon$-boxes, and those in $Q_2$ are all inside the extended portion of the $\epsilon$-boxes.
- Now we need to push down the points so that a match, if it exists, can be found.
- We do this for all arcs together.

# Processing Each Clique

## Processing Cliques

- Each clique $\chi$ represents an angular interval (an arc) $\mathcal{I}^* = [\theta_1^*, \theta_2^*]$. For any angle of rotation $\theta \in \mathcal{I}^*$, each of the $k-1$ point of $Q \setminus q_i$ lies inside $k-1$ disjoint extended $\epsilon$-boxes.
- Partition the arcs corresponding to the points in $Q \setminus \{q_i\}$ into two subsets $Q_1$ and $Q_2$. Arcs in $Q_1$ are all inside the $\epsilon$-boxes, and those in $Q_2$ are all inside the extended portion of the $\epsilon$-boxes.
- Now we need to push down the points so that a match, if it exists, can be found.
- We do this for all arcs together.

# The Pushing Down



# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q} \sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q}\sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q} \sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q} \sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q} \sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

# Homogeneous Splitting

## Homogeneous splitting of $\mathcal{I}^*$

- Consider a $\theta \in \mathcal{I}^*$.
- For each $q \in Q_1$, let $f_q^1(\theta)$ denotes the distance of $q$ from the bottom of the corresponding $\epsilon$-box.
- For each $q \in Q_2$, $f_q^2(\theta)$ denotes the distance of $q$ from the top of the corresponding $\epsilon$-box.
- The functions $f_q^i(\theta)$ $i = 1, 2$ are like $\overline{q_a q} \sin(\alpha + \theta) - c$.

## Observation

For a given $q \in Q_i$, the above functions are univariate, continuous, and unimodular.

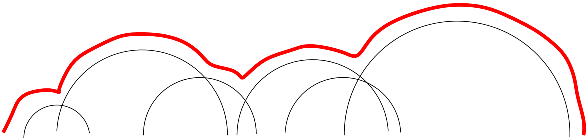
# Envelope of Functions

**Definition($\mathcal{L}(\theta)$ for $\theta \in \mathcal{I}^*$)**

$\mathcal{L}(\theta)$ denotes the lower envelope of $|Q_1|$ functions, namely $f_q^1(\theta)$, $q \in Q_1$.

**Definition($\mathcal{U}(\theta)$ for $\theta \in \mathcal{I}^*$)**

$\mathcal{U}(\theta)$ denotes the upper envelope of $|Q_2|$ functions, namely $f_q^2(\theta)$, $q \in Q_2$.



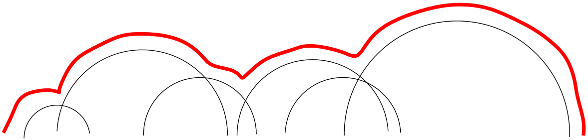
# Envelope of Functions

**Definition($\mathcal{L}(\theta)$ for $\theta \in \mathcal{I}^*$)**

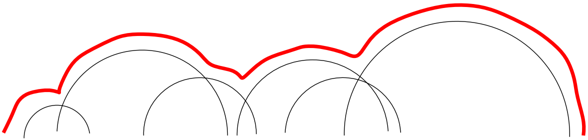
$\mathcal{L}(\theta)$ denotes the lower envelope of $|Q_1|$ functions, namely $f_q^1(\theta)$, $q \in Q_1$.

**Definition($\mathcal{U}(\theta)$ for $\theta \in \mathcal{I}^*$)**

$\mathcal{U}(\theta)$ denotes the upper envelope of $|Q_2|$ functions, namely $f_q^2(\theta)$, $q \in Q_2$.



# Envelope of Functions



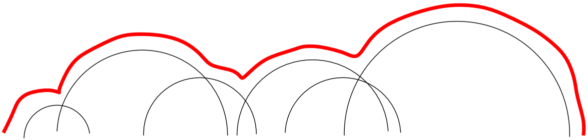
## Minimum Amount of Downward Translation for $Q_2$

At a rotation angle $\theta \in \mathcal{I}^*$, the minimum amount of downward translation required to place the points in $Q_2$ in the corresponding $\epsilon$-box is $max_{q \in Q_2} f_q^2(\theta) = \mathcal{U}(\theta)$.

## Maximum Amount of Downward Translation for $Q_1$

Similarly, the maximum amount of downward translation that may retain all the points in $Q_1$ in its corresponding $\epsilon$-box is $min_{q \in Q_1} f_q^1(\theta) = \mathcal{L}(\theta)$.

# Envelope of Functions



## Minimum Amount of Downward Translation for $Q_2$

At a rotation angle $\theta \in \mathcal{I}^*$, the minimum amount of downward translation required to place the points in $Q_2$ in the corresponding $\epsilon$-box is $max_{q \in Q_2} f_q^2(\theta) = \mathcal{U}(\theta)$.

## Maximum Amount of Downward Translation for $Q_1$

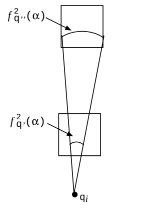
Similarly, the maximum amount of downward translation that may retain all the points in $Q_1$ in its corresponding $\epsilon$-box is $min_{q \in Q_1} f_q^1(\theta) = \mathcal{L}(\theta)$.

## Definition

A rotation angle $\theta$ is said to be a break-point in $\mathcal{L}$ if $\mathcal{L}(\theta) = f^1_{q_a}(\theta) = f^1_{q_b}(\theta)$ for $q_a, q_b \in Q_1$, $q_a \neq q_b$. Similarly, the break-points of the $\mathcal{U}$ function is defined.

## Lemma

A pair of functions $f^1_{q'}(\theta)$ and $f^1_{q''}(\theta)$ (corresponding to $q', q'' \in Q_1$, $q' \neq q''$) w.r.t. $\theta \in \mathcal{I}^*$ may intersect in at most two points. The same is true for a pair of functions $f^2_{q'}(\theta)$ and $f^2_{q''}(\theta)$ for a pair of points $q', q'' \in Q_2$.



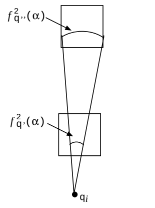
(b)

## Definition

A rotation angle $\theta$ is said to be a break-point in $\mathcal{L}$ if $\mathcal{L}(\theta) = f^1_{q_a}(\theta) = f^1_{q_b}(\theta)$ for $q_a, q_b \in Q_1$, $q_a \neq q_b$. Similarly, the break-points of the $\mathcal{U}$ function is defined.

## Lemma

A pair of functions $f^1_{q'}(\theta)$ and $f^1_{q''}(\theta)$ (corresponding to $q', q'' \in Q_1$, $q' \neq q''$) w.r.t. $\theta \in \mathcal{I}^*$ may intersect in at most two points. The same is true for a pair of functions $f^2_{q'}(\theta)$ and $f^2_{q''}(\theta)$ for a pair of points $q', q'' \in Q_2$.




(b)

# Envelope of Functions

## Observation

The collection of functions $\{f_q^1(\theta), q \in Q_1\}$ follows a $(k, 2)$-*Davenport-Schinzel sequence*. The same result is true for the collection of functions $\{f_q^2(\theta), q \in Q_2\}$.

## Lemma

The maximum number of break-points in the function $\mathcal{L}(\theta)$ is $\lambda_2(|Q_1|) = 2|Q_1| - 1$, and it can be computed in $O(|Q_1| \log |Q_1|)$ time. Similarly, for $\mathcal{U}(\theta)$.

## Moral of Homogeneous Splitting of $\mathcal{I}^*$

$\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ is split into $O(k)$ sub-intervals defined by break-points of $\mathcal{L}(\theta)$ and $\mathcal{U}(\theta)$.

# Envelope of Functions

## Observation

The collection of functions $\{f_q^1(\theta), q \in Q_1\}$ follows a $(k, 2)$-*Davenport-Schinzel sequence*. The same result is true for the collection of functions $\{f_q^2(\theta), q \in Q_2\}$.

## Lemma

The maximum number of break-points in the function $\mathcal{L}(\theta)$ is $\lambda_2(|Q_1|) = 2|Q_1| - 1$, and it can be computed in $O(|Q_1| \log |Q_1|)$ time. Similarly, for $\mathcal{U}(\theta)$.

## Moral of Homogeneous Splitting of $\mathcal{I}^*$

$\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ is split into $O(k)$ sub-intervals defined by break-points of $\mathcal{L}(\theta)$ and $\mathcal{U}(\theta)$.

# Envelope of Functions

## Observation

The collection of functions $\{f_q^1(\theta), q \in Q_1\}$ follows a $(k, 2)$-*Davenport-Schinzel sequence*. The same result is true for the collection of functions $\{f_q^2(\theta), q \in Q_2\}$.

## Lemma

The maximum number of break-points in the function $\mathcal{L}(\theta)$ is $\lambda_2(|Q_1|) = 2|Q_1| - 1$, and it can be computed in $O(|Q_1| \log |Q_1|)$ time. Similarly, for $\mathcal{U}(\theta)$.

## Moral of Homogeneous Splitting of $\mathcal{I}^*$

$\mathcal{I}^* = [\theta_1^*, \theta_2^*]$ is split into $O(k)$ sub-intervals defined by break-points of $\mathcal{L}(\theta)$ and $\mathcal{U}(\theta)$.

# Critical Angle

**Definition (Critical Angle)**

If a match is found by a rotation of $Q$ by the angle $\theta^* \in \mathcal{I}^*$ and a vertical downward shift, then $\theta^*$ is said to be a critical angle.

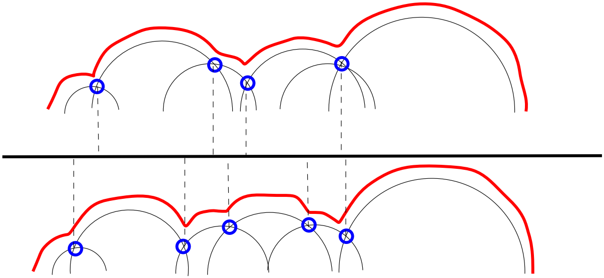
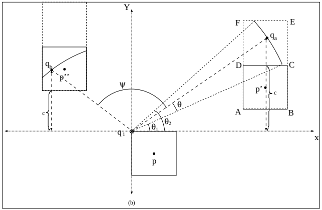


Figure: Envelopes and break points.

# Computation of Critical Angle



(b)

## Critical Angle Computation

For $\theta \in [\theta_1, \theta_2]$, the vertical downward shift will be determined

# Computation of Critical Angle

## Critical Angle Computation

$\Delta_1$ = Minimum amount of downward shift required to bring $q_a$ inside the $\epsilon$-box of $p'$. Thus, $\Delta_1 = \delta(q_i, q_\alpha) sin(\theta + \theta_1) - (y(p') + \frac{\epsilon}{2})$.

$\Delta_2$ = Maximum amount of permissible downward shift keeping $q_b$ inside the $\epsilon$-box of $p''$. Thus, $\Delta_2 = \delta(q_i, q_\beta) sin(\theta + \theta_1 + \psi) - (y(p'') - \frac{\epsilon}{2})$.

- A feasible solution $\theta$ (if it exists) must satisfy $\Delta_1 \leq \Delta_2$.

# Computation of Critical Angle

## Critical Angle Computation

$\Delta_1$ = Minimum amount of downward shift required to bring $q_a$ inside the $\epsilon$-box of $p'$. Thus, $\Delta_1 = \delta(q_i, q_\alpha) sin(\theta + \theta_1) - (y(p') + \frac{\epsilon}{2})$.

$\Delta_2$ = Maximum amount of permissible downward shift keeping $q_b$ inside the $\epsilon$-box of $p''$. Thus, $\Delta_2 = \delta(q_i, q_\beta) sin(\theta + \theta_1 + \psi) - (y(p'') - \frac{\epsilon}{2})$.

- A feasible solution $\theta$ (if it exists) must satisfy $\Delta_1 \leq \Delta_2$.

# Computation of Critical Angle

## Critical Angle Computation

$\Delta_1$ = Minimum amount of downward shift required to bring $q_a$ inside the $\epsilon$-box of $p'$. Thus, $\Delta_1 = \delta(q_i, q_\alpha) sin(\theta + \theta_1) - (y(p') + \frac{\epsilon}{2})$.

$\Delta_2$ = Maximum amount of permissible downward shift keeping $q_b$ inside the $\epsilon$-box of $p''$. Thus, $\Delta_2 = \delta(q_i, q_\beta) sin(\theta + \theta_1 + \psi) - (y(p'') - \frac{\epsilon}{2})$.

- A feasible solution $\theta$ (if it exists) must satisfy $\Delta_1 \leq \Delta_2$.

# Complexity Analysis

## Complexity Analysis

- Each point of $Q$ needs to be anchored at the top-left corner of the $\epsilon$-box of each point in $P$.
- The nodes of the circular arc graph $G$ are obtained in $O(nk)$ time and the cliques of $G$ can be obtained in $O(nk \log n)$ time.
- While processing a clique, computation of functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k \log k)$ time.
- Number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case.
- The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time.

# Complexity Analysis

## Complexity Analysis

- Each point of $Q$ needs to be anchored at the top-left corner of the $\epsilon$-box of each point in $P$.
- The nodes of the circular arc graph $G$ are obtained in $O(nk)$ time and the cliques of $G$ can be obtained in $O(nk\log n)$ time.
- While processing a clique, computation of functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k\log k)$ time.
- Number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case.
- The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time.

# Complexity Analysis

## Complexity Analysis

- Each point of $Q$ needs to be anchored at the top-left corner of the $\epsilon$-box of each point in $P$.
- The nodes of the circular arc graph $G$ are obtained in $O(nk)$ time and the cliques of $G$ can be obtained in $O(nk\log n)$ time.
- While processing a clique, computation of functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k\log k)$ time.
- Number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case.
- The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time.

# Complexity Analysis

## Complexity Analysis

- Each point of $Q$ needs to be anchored at the top-left corner of the $\epsilon$-box of each point in $P$.
- The nodes of the circular arc graph $G$ are obtained in $O(nk)$ time and the cliques of $G$ can be obtained in $O(nk\log n)$ time.
- While processing a clique, computation of functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k\log k)$ time.
- Number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case.
- The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time.

# Complexity Analysis

## Complexity Analysis

- Each point of $Q$ needs to be anchored at the top-left corner of the $\epsilon$-box of each point in $P$.
- The nodes of the circular arc graph $G$ are obtained in $O(nk)$ time and the cliques of $G$ can be obtained in $O(nk\log n)$ time.
- While processing a clique, computation of functions $\mathcal{U}$ and $\mathcal{L}$ needs $O(k\log k)$ time.
- Number of elements in $\mathcal{U} \cup \mathcal{L}$ is $O(k)$ in the worst case.
- The algebraic computation for processing each element in $\mathcal{U} \cup \mathcal{L}$ needs $O(1)$ time.

# The Final Result

**Theorem**

The time complexity of the proposed algorithm for $\epsilon$-approximate matching of $Q$ with a subset of $P$ where the neighbourhood around a point (in $P$) is defined as an $\epsilon$-box, is $O(n^2k^2(\log n + k \log k))$.

# Further reading I


H. Alt and L. J. Guibas,
*Discrete geometric shapes: matching, interpolation, and approximation*,
in Handbook of Computational Geometry, Eds. Sack, J.-R., and Urrutia, J., 1999..

H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl,
*Congruence, similarity and symmetries of geometric objects*,
Discrete Computational Geometry, vol. 3, 237-256, 1988.

E. M. Arkin, K. Kedem, J. S. B. Mitchell, J. Sprinzak and M. Werman,
*Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms*,
ORSA Journal on Computing, vol. 4, 375-386, 1992.

P. J. Heffernan and S. Schirra,
*Approximate decision algorithms for point set congruence*,
Computational Geometry: Theory and Applications, vol. 4, no. 3, pp. 137-156, 1994.

Thank You!