2IL76
Algorithms for Geographic Data

Spring 2015

Lecture 3: Simplification

Problem

Many algorithms handling movement data are slow, e.g.

- similarity $O(nm \log nm)$
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Problem

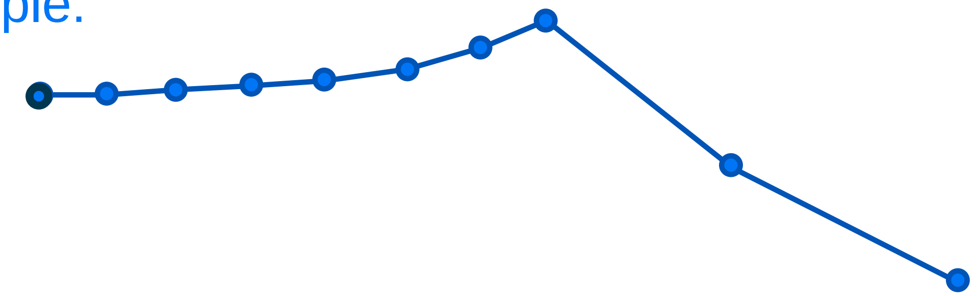
Many algorithms handling movement data are slow, e.g.

- similarity O(nm log nm)
- approximate clustering O(n^2+nml)
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

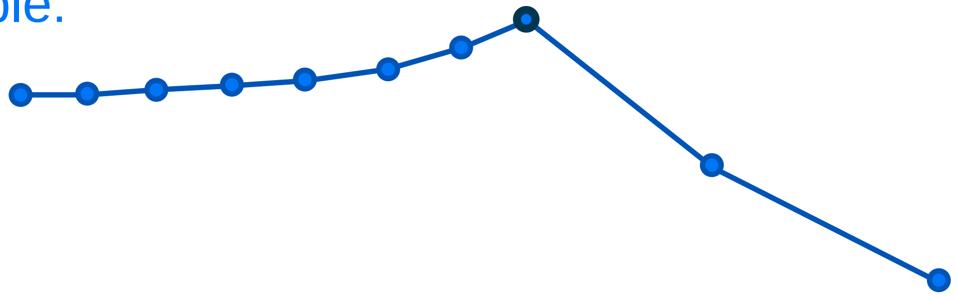
Many algorithms handling movement data are slow, e.g.

- similarity $O(nm \log nm)$
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

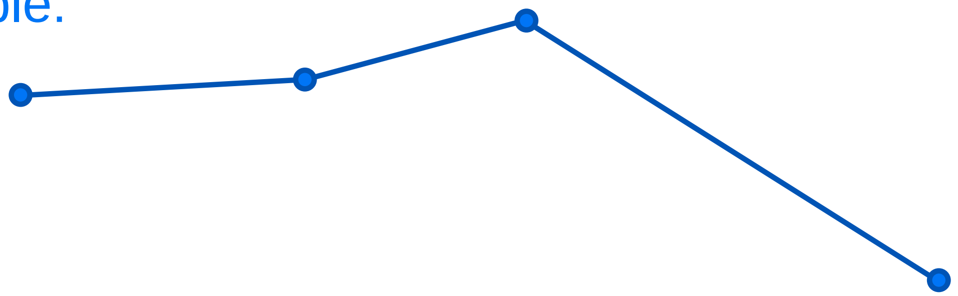
Many algorithms handling movement data are slow, e.g.

- similarity O(nm log nm)
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

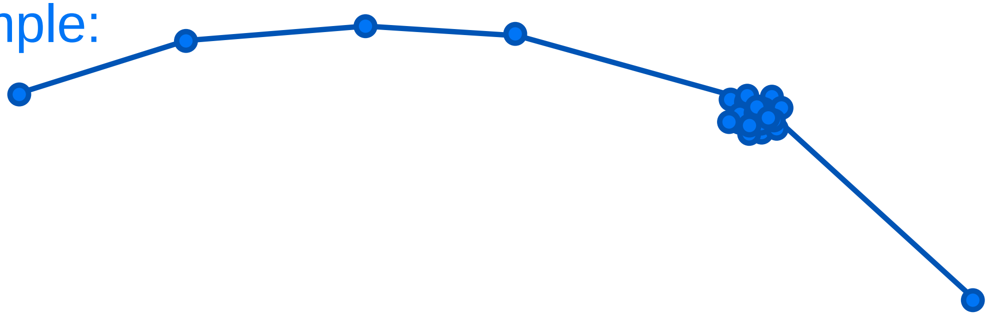
Many algorithms handling movement data are slow, e.g.

- similarity O(nm log nm)
- approximate clustering O(n^2+nml)
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

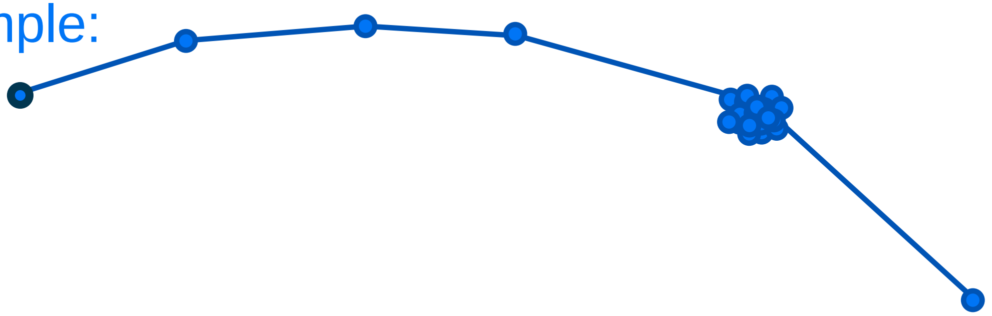
Many algorithms handling movement data are slow, e.g.

- similarity $O(nm \log nm)$
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



TU/e Technische Universiteit Eindhoven University of Technology

Problem

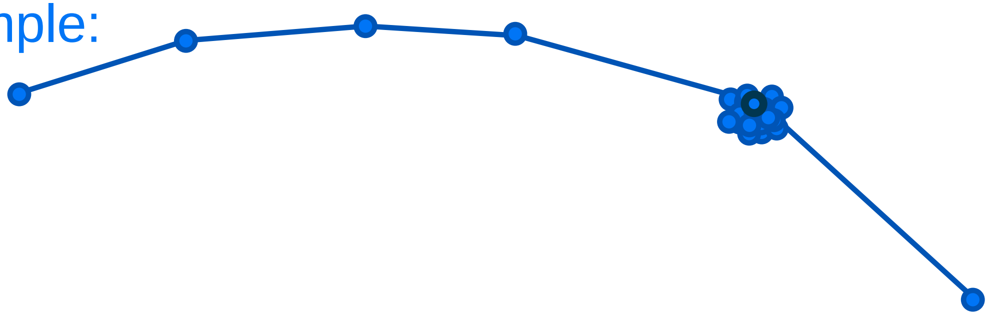
Many algorithms handling movement data are slow, e.g.

- similarity $O(nm \log nm)$
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

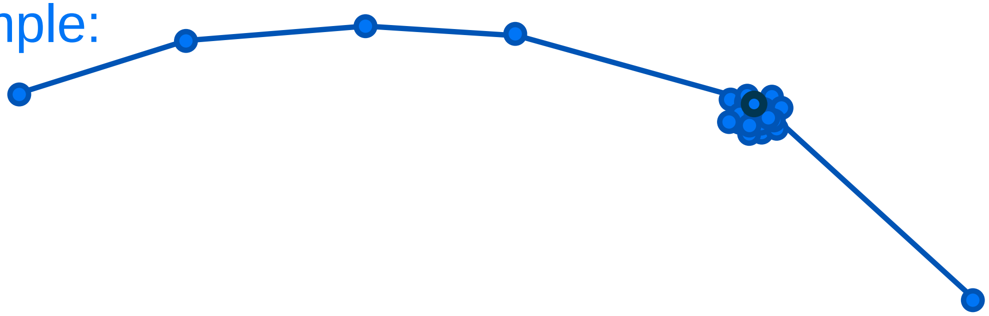
Many algorithms handling movement data are slow, e.g.

- similarity O(nm log nm)
- approximate clustering $O(n^2+nml)$
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



Problem

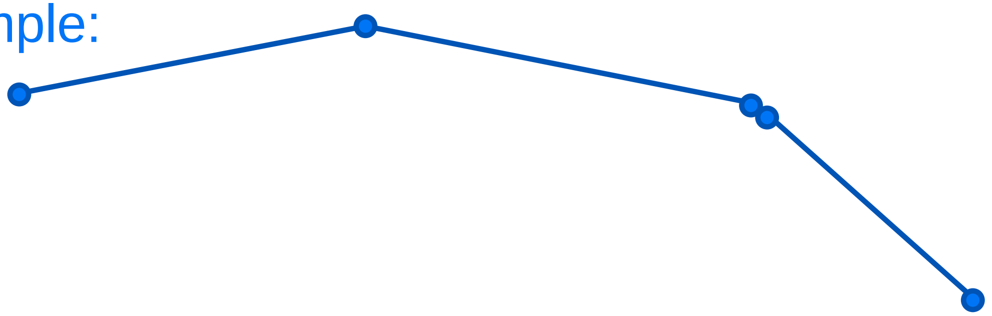
Many algorithms handling movement data are slow, e.g.

- similarity O(nm log nm)
- approximate clustering O(n^2+nml)
- …

Observation:

Many trajectories can be simplified without losing much information.

Example:



A sequence of steps to solve a problem

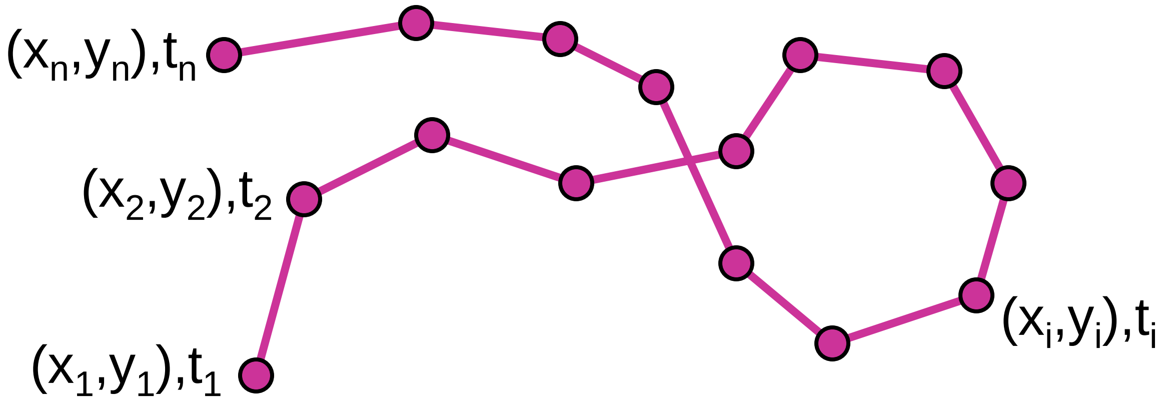
- Understand the input
- Understand what you really want from the output
- Write an input and output specification
  and double-check it!

- Find geometric properties of the desired output
- Construct an algorithm

- Verify that it actually solves the problem you specified
- Analyze the efficiency

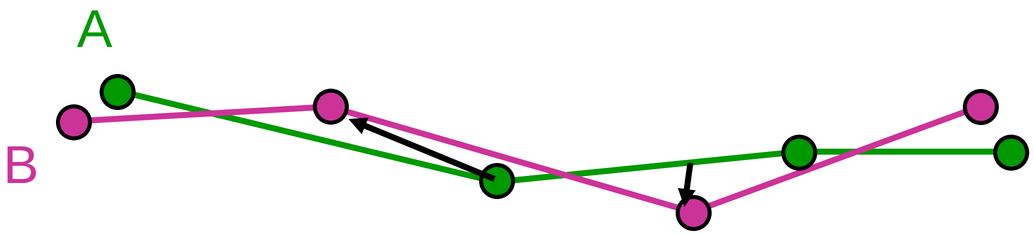
Assumptions

- Due to lack of further information: constant speed (and velocity) on each segment
  - ➡ location changes continuously, but speed/velocity is piecewise constant and changes "in jumps"



Assumptions

- We do not want to assume that …
  - sampling occurred at regular intervals
  - no data is missing
  - the object is only present at the measured locations but not in between



How much did A deviate from the route of B?

A sequence of steps to solve a problem

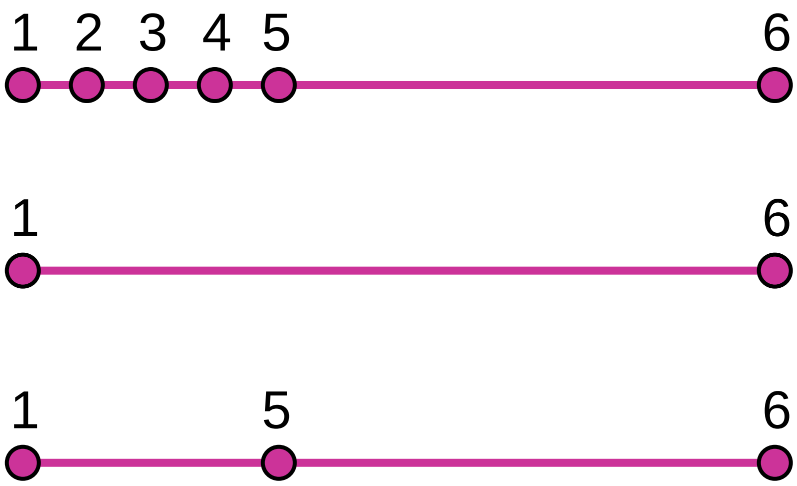
- √ Understand the input
- Understand what you really want from the output
- Write an input and output specification
  and double-check it!

- Find geometric properties of the desired output
- Construct an algorithm

- Verify that it actually solves the problem you specified
- Analyze the efficiency

Specifying the output

- We want a trajectory with the smallest number of vertices and with error at most ε
- Without thinking about the output, one could say:
  "Just use a line simplification method"



assume the numbers correspond to minutes

we have made speed uniform!

correct!

Specifying the output

- We want a trajectory with the smallest number of vertices and with error at most ε

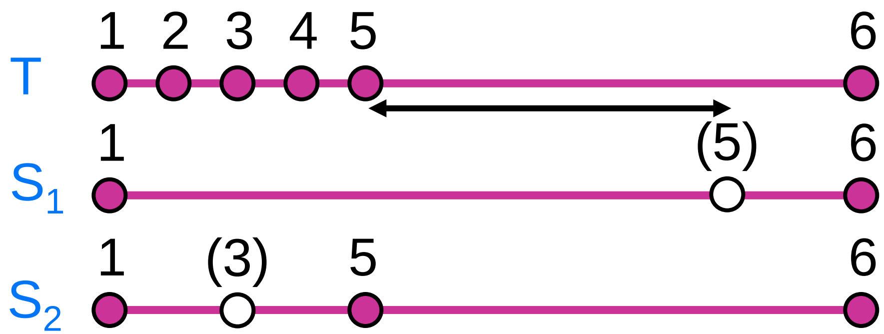
  … but what do we mean with "error" ?

Option 1: the maximum error in location for any moment of time:

$$\max_{\text{time } t} \text{distance}(T(t), S(t))$$

T input trajectory

S simplified trajectory



Specifying the output

- We want a trajectory with the smallest number of vertices and with error at most ε

  … but what do we mean with “error” ?

  Option 2: the error in speed as a multiplicative factor, for any moment

  Option 3: the error in velocity (speed and heading combined)

  … or any combination of the above

But …

- it is important to start simple
- therefore in this lecture mostly: curve simplification
- you will see in Exercise 3 an example of how to incorporate time

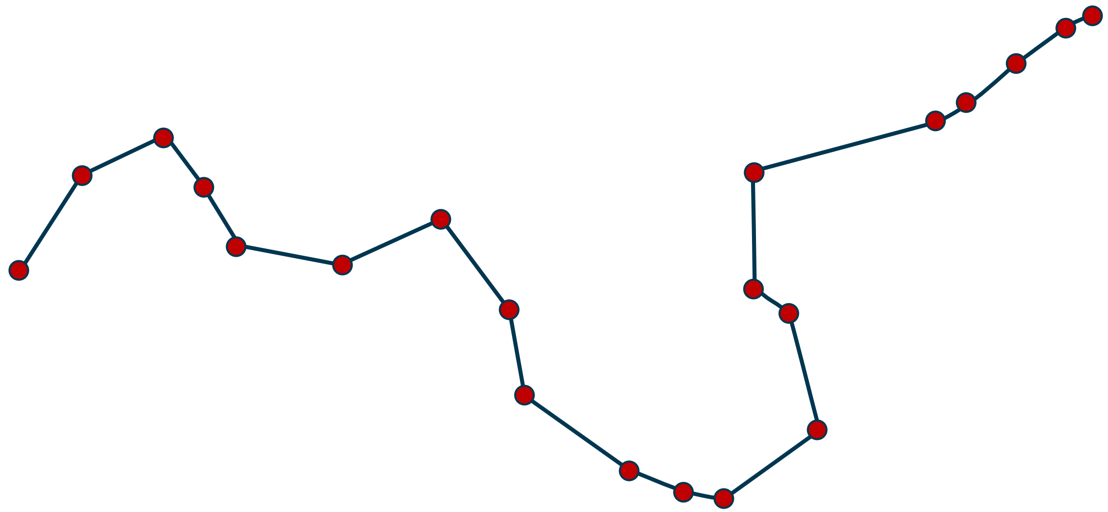
Outline

Simplifying polygonal curves

- Ramer–Douglas–Peucker, 1973
- Driemel, Har-Peled and Wenk, 2010
- Imai-Iri, 1988
- Agarwal, Har-Peled, Mustafa and Wang, 2005

Ramer-Douglas-Peucker

- 1972 by Urs Ramer and 1973 by David Douglas and Thomas Peucker
- The most successful simplification algorithm. Used in GIS, geography, computer vision, pattern recognition…



- Very easy to implement and works well in practice.

Ramer-Douglas-Peucker

Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

Initially i=1 and j=n

Algorithm DP(P,i,j)

Find the vertex v_f between p_i and p_j farthest from $p_i p_j$.

dist := the distance between v_f and $p_i p_j$.

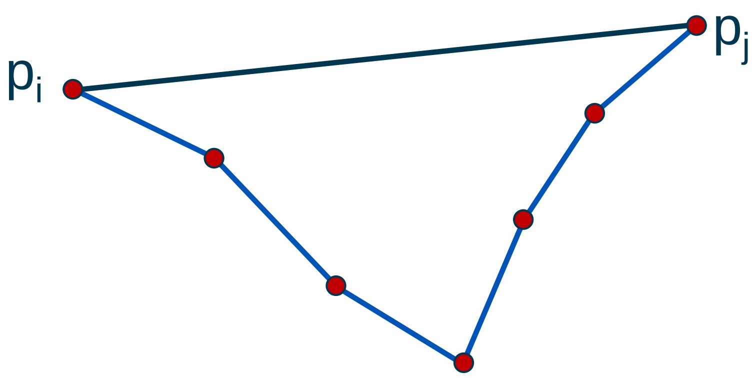
if dist > ε then

DP(P, v_i , v_f)

DP(P, v_f , v_j)

else

Output($v_i v_j$)



Ramer-Douglas-Peucker

Input polygonal path $P = \langle p_1,\ldots,p_n\rangle$ and threshold ε

Initially $i=1$ and $j=n$

Algorithm DP(P,i,j)

Find the vertex v_f between p_i and p_j farthest from p_ip_j.

dist := the distance between v_f and p_ip_j.

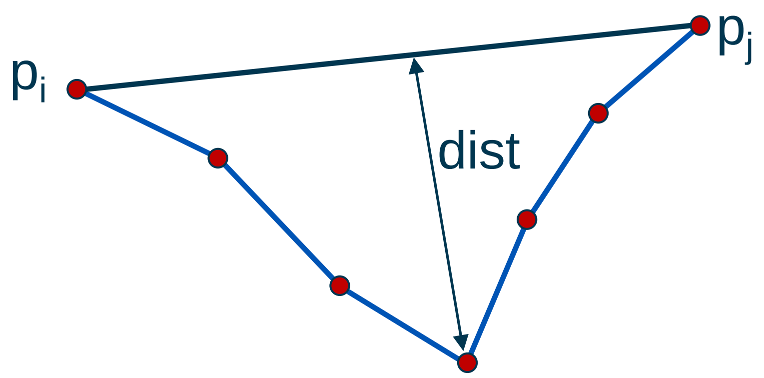
if dist > ε then

DP(P, v_i , v_f)

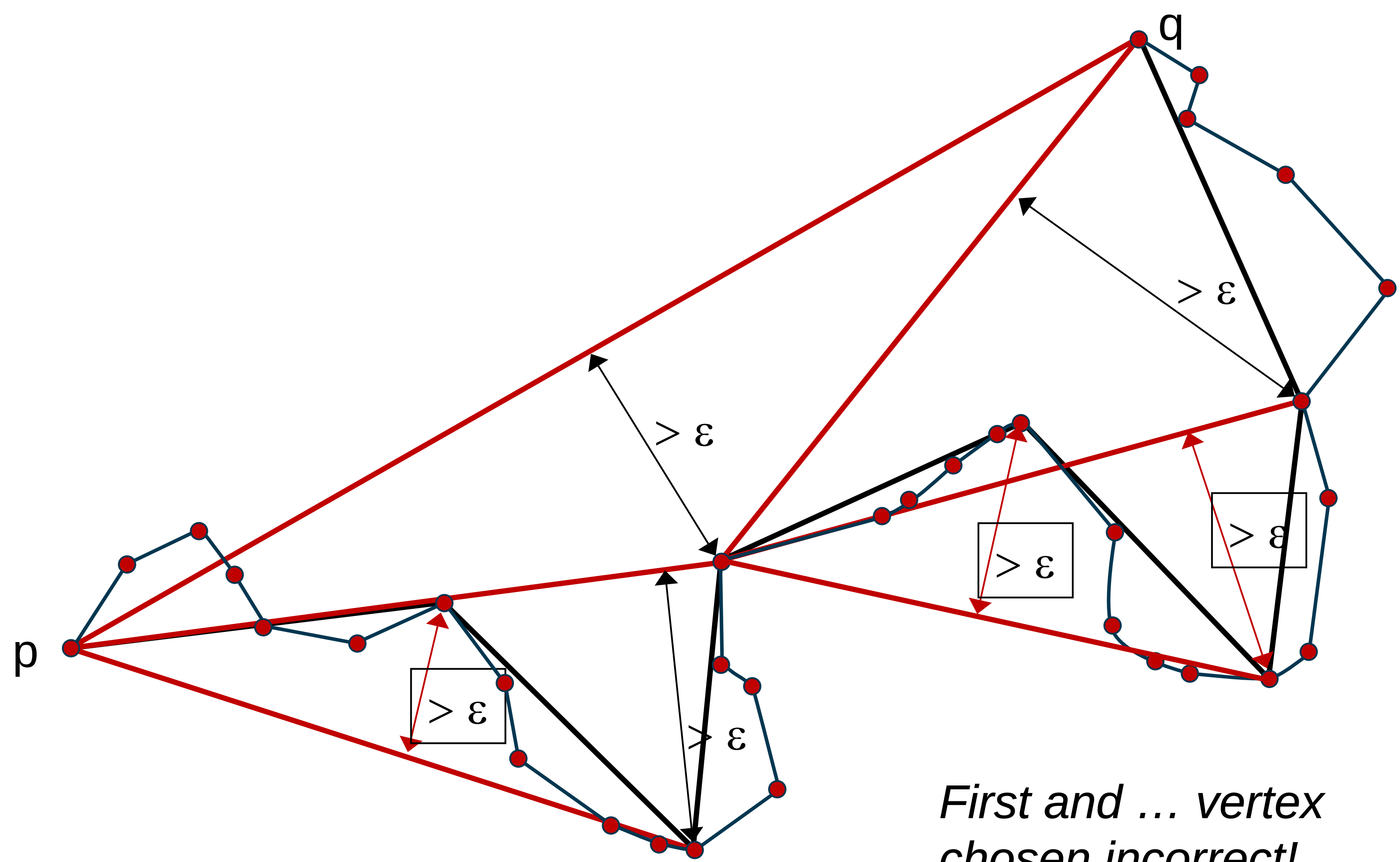
DP(P, v_f , v_j)

else

Output(v_iv_j)

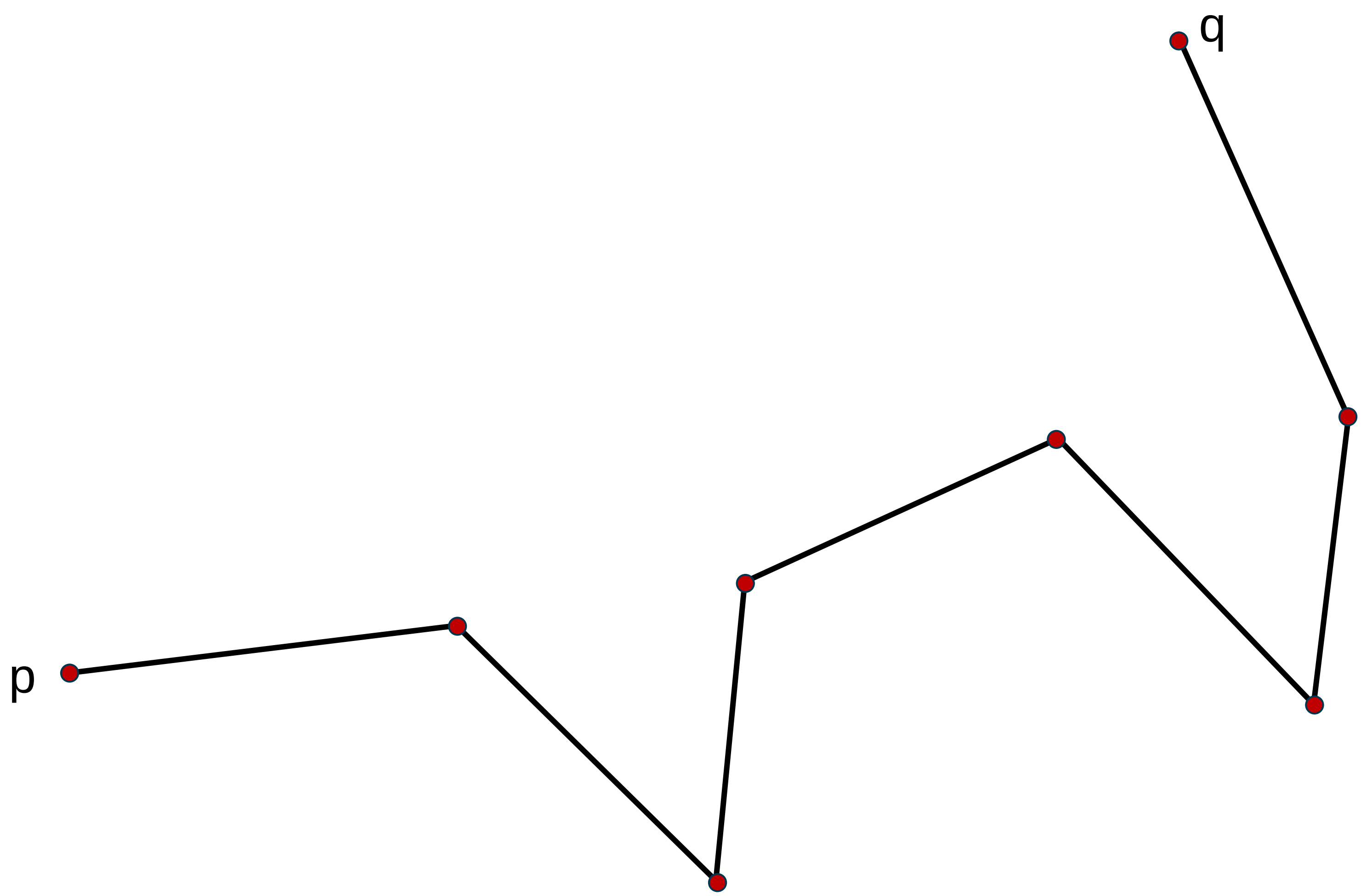


Ramer-Douglas-Peucker



First and … vertex chosen incorrect!

Ramer-Douglas-Peucker



Ramer-Douglas-Peucker

Time complexity?

Testing a shortcut between p_i and p_j takes $O(j\text{-}i)$ time.

Worst-case recursion?

DP(P, v_i , v_{i+1})

DP(P, v_{i+1} , v_j)

Time complexity

$T(n) = O(n) + T(n\text{-}1)$

$= O(n^2)$

Algorithm DP(P,i,j)

Find the vertex v_f farthest from p_ip_j.

dist := the distance between v_f and p_ip_j.

if dist > ε then

DP(P, v_i , v_f)

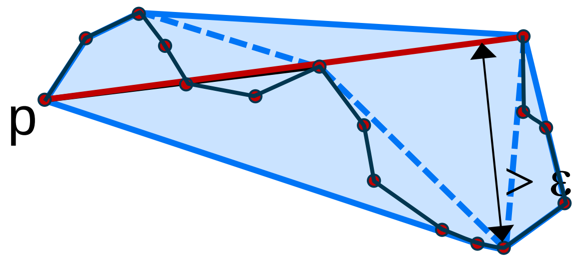
DP(P, v_f , v_j)

else

Output(v_iv_j)

Summary: Ramer-Douglas-Peucker

- Worst-case time complexity: $O(n^2)$
- In most realistic cases: $T(n) = T(n_1) + T(n_2) + O(n) = O(n \log n)$, where n_1 and n_2 are smaller than n/c for some constant c.
- If the curve is in 2D and it does not self-intersect then the algorithm can be implemented in O(n log* n) time.



[Hershberger & Snoeyink'98]

- Does not give any bound on the complexity of the simplification!

Driemel et al.

Simple simplification $(P = \langle p_1,\ldots,p_n\rangle, \varepsilon)$

$P' := \langle p_1 \rangle$

$i := 1$

while $i<n$ do

  $q := p_i$

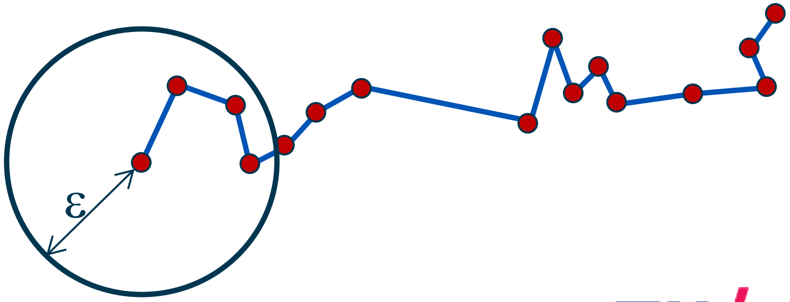
  $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$

  if no such vertex then set $i:=n$

  add p_i to P'

end

return P'



Driemel et al.

Simple simplification $(P = \langle p_1,\ldots,p_n \rangle, \varepsilon)$

$P' := \langle p_1 \rangle$

$i := 1$

while $i<n$ do

$\quad q := p_i$

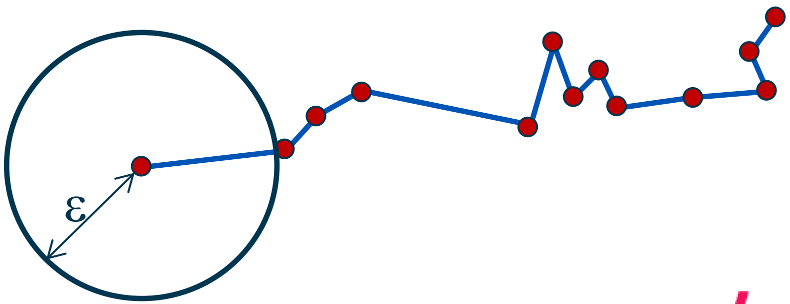
$\quad p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$

$\quad$ if no such vertex then set $i:=n$

$\quad$ add p_i to P'

end

return P'



Driemel et al.

Simple simplification $(P = \langle p_1,\ldots,p_n\rangle, \varepsilon)$

$P' := \langle p_1 \rangle$

$i := 1$

while $i<n$ do

  $q := p_i$

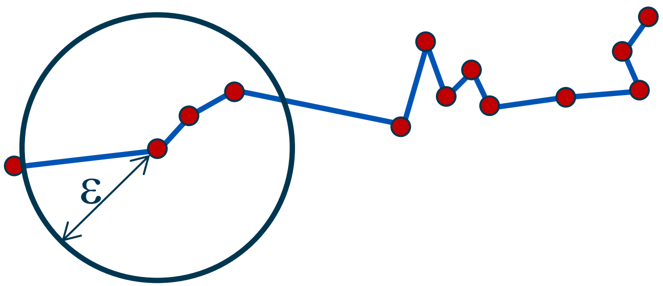
  $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$

  if no such vertex then set $i:=n$

  add p_i to P'

end

return P'



Driemel et al.

Simple simplification $(P = \langle p_1, \ldots, p_n \rangle, \varepsilon)$

$P' := \langle p_1 \rangle$

$i := 1$

while $i < n$ do

    $q := p_i$

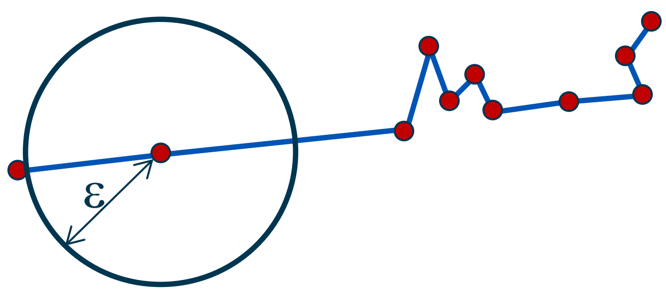
    $p_i :=$ first vertex p_i in $\langle q, \ldots, p_n \rangle$ s.t. $|q - p_i| > \varepsilon$

    if no such vertex then set $i := n$

    add p_i to P'

end

return P'



Driemel et al.

Simple simplification ($P = \langle p_1,\ldots,p_n\rangle$, ε)

$P' := \langle p_1 \rangle$

$i:=1$

while $i<n$ do

  $q := p_i$

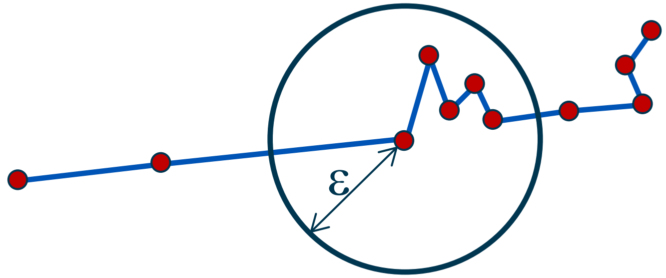
  $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$

  if no such vertex then set $i:=n$

  add p_i to P'

end

return P'



Driemel et al.

Simple simplification ($P = \langle p_1,\ldots,p_n \rangle, \varepsilon$)

$P' := \langle p_1 \rangle$

$i := 1$

while $i < n$ do

$q := p_i$

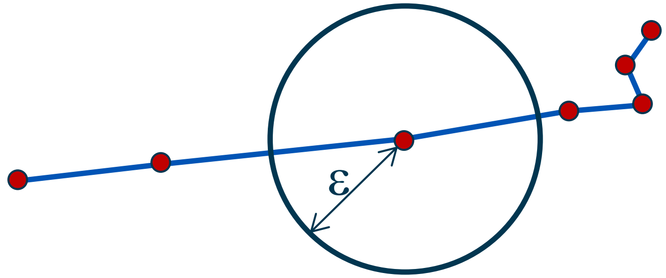
$p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$

if no such vertex then set $i := n$

add p_i to P'

end

return P'



Driemel et al.

Simple simplification ($P = \langle p_1,\ldots,p_n \rangle, \varepsilon$)

$P' := \langle p_1 \rangle$

$i := 1$

while $i<n$ do

  $q := p_i$

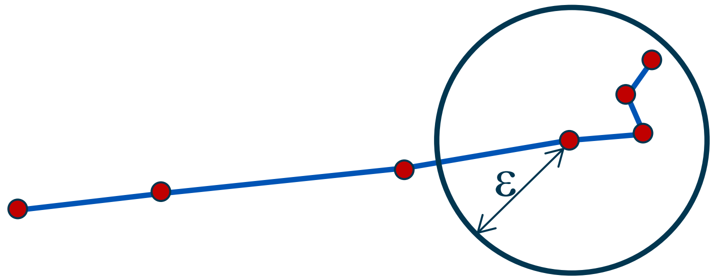
  p_i := first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$

  if no such vertex then set $i:=n$

  add p_i to P'

end

return P'



Driemel et al.

Simple simplification ($P = \langle p_1,\ldots,p_n\rangle, \varepsilon$)

$P' := \langle p_1 \rangle$

$i := 1$

while $i<n$ do

  $q := p_i$

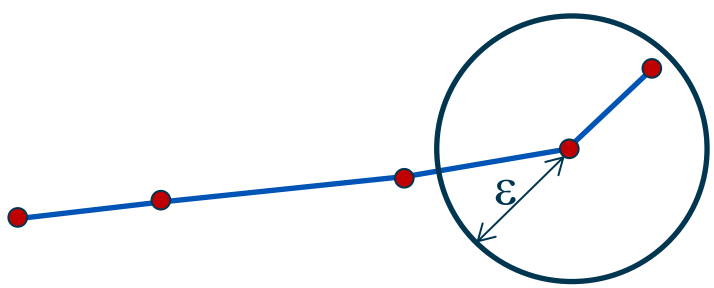
  $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$

  if no such vertex then set $i:=n$

  add p_i to P'

end

return P'



Driemel et al.

Simple simplification $(P = \langle p_1,\ldots,p_n \rangle, \varepsilon)$

P':= $\langle p_1 \rangle$

i:=1

while i<n do

    q := p_i

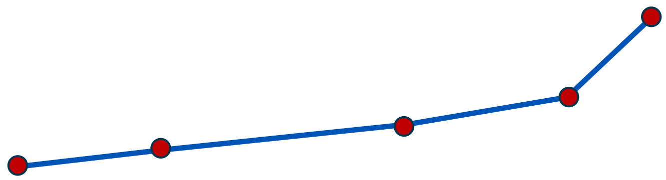
    p_i := first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$

    if no such vertex then set i:=n

    add p_i to P'

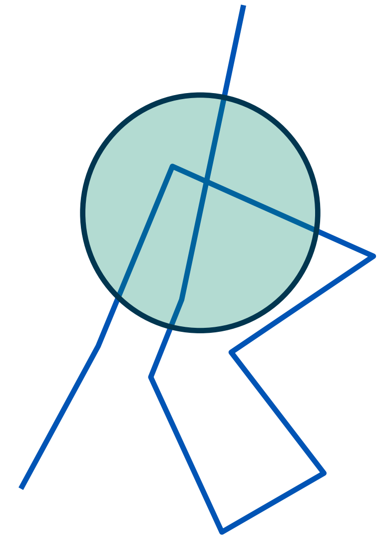
end

return P'



Driemel et al.

Simple simplification $(P = \langle p_1, \ldots, p_n \rangle, \varepsilon)$



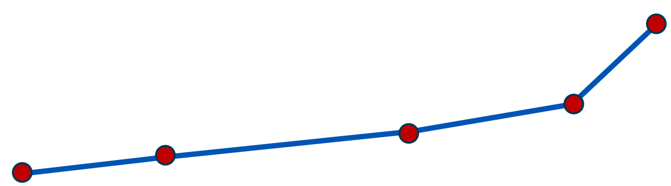
Property 1:

All edges (except the last one) have length at least ε.

Property 2: $\delta_F(P,P') \leq \varepsilon$

Definition: A curve P is c-packed, if has finite length, and for any ball $b(p,r)$ it holds $|P \cap b(p,r)| < cr$.

Running time: $O(n)$



Simplification maintains packedness,

but gives no bound on size.

Imai-Iri

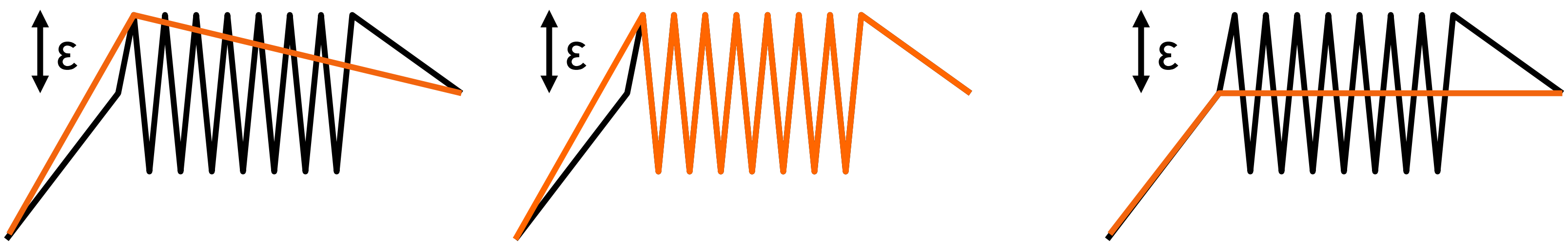
- Both previous algorithms are simple and fast but do not give a bound on the complexity of the simplification!

- Examples for which they perform poorly?

Imai-Iri

- Both previous algorithms are simple and fast but do not give a bound on the complexity of the simplification!

- Examples for which they perform poorly?



Douglas-Peucker

Optimal

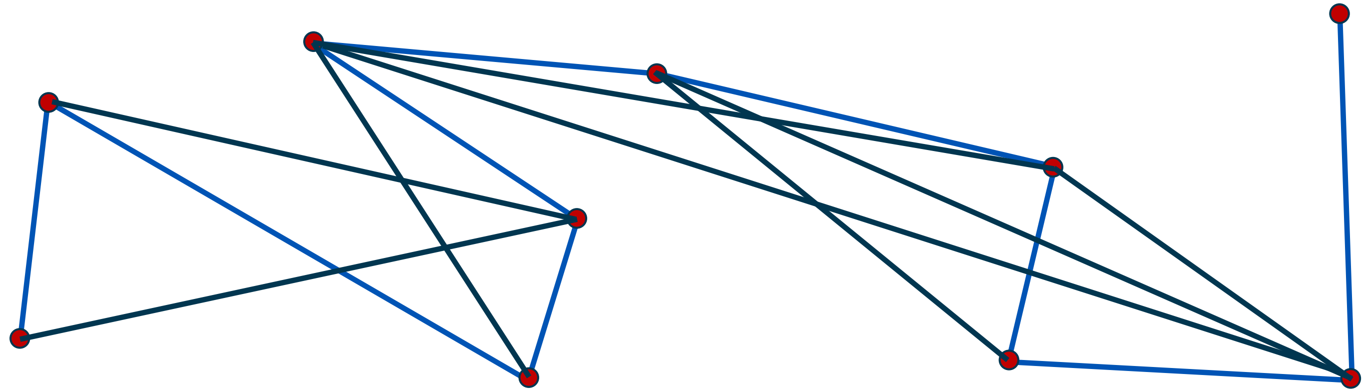
Imai-Iri

- Both previous algorithms are simple and fast but do not give a bound on the complexity of the simplification!
- Examples for which they perform poorly?
- Imai-Iri 1988 gave an algorithm that produces an ε-simplification with the minimum number of links.
- Generally, two variants
  - Min-vertices (for given ε): this is the one we mostly consider
  - Min-ε (for given length of simplification): often uses binary search with Min-vertices as subroutine
- Another distinction: Does the simplification only use vertices of the input or not? In lecture only input vertices, but see assignment

Imai-Iri

Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

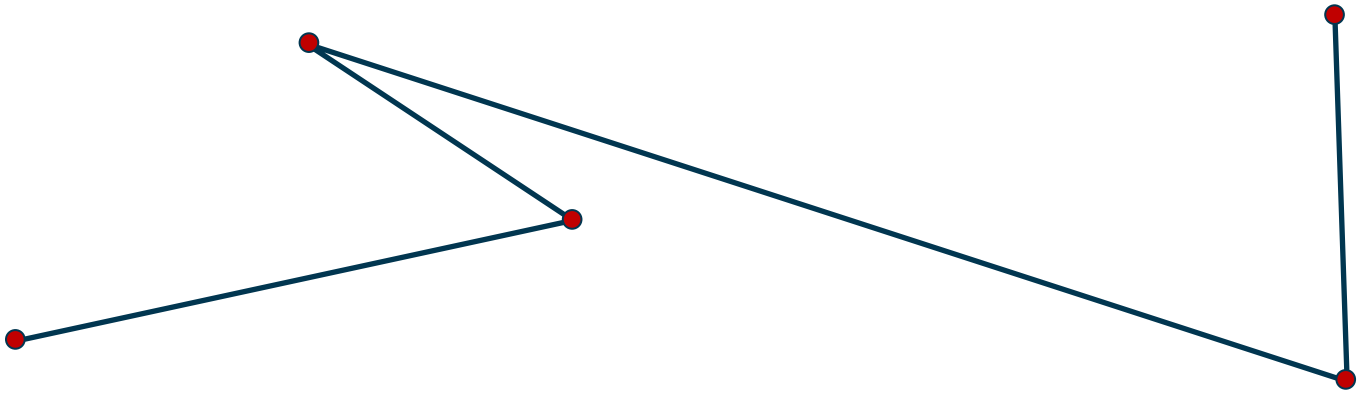
1. Build a graph G containing all valid shortcuts.
2. Find a minimum link path from p_1 to p_n in G



Imai-Iri

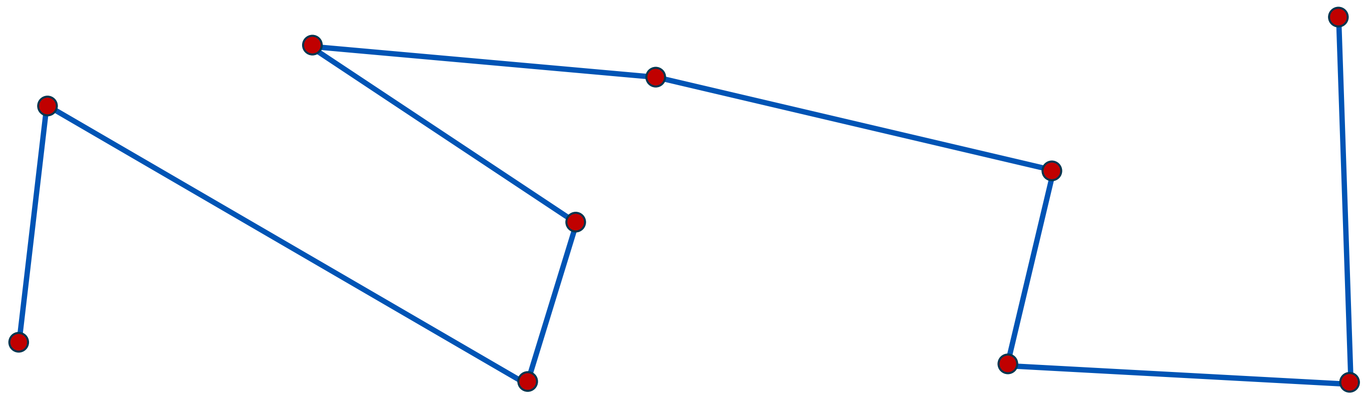
Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

1. Build a graph G containing all valid shortcuts.
2. Find a minimum link path from p_1 to p_n in G



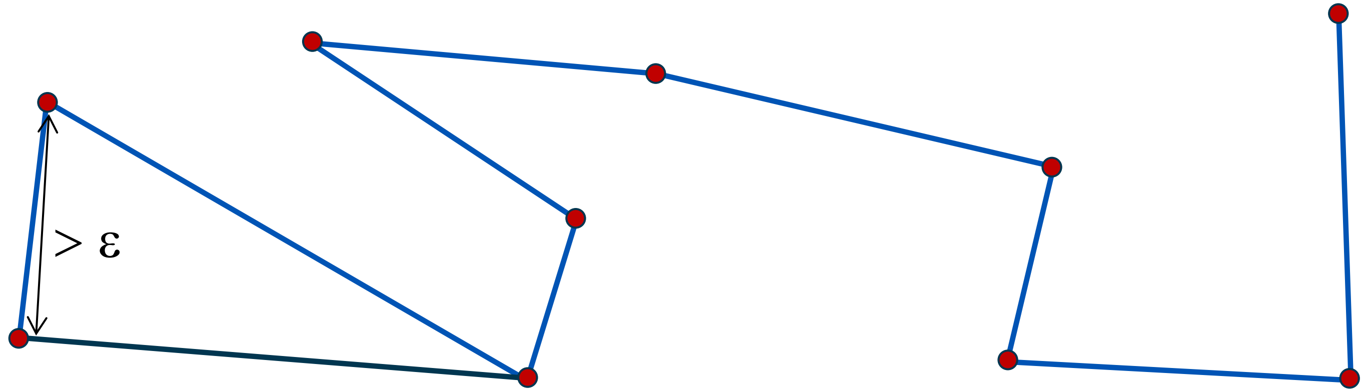
Imai-Iri

Find all possible valid shortcuts



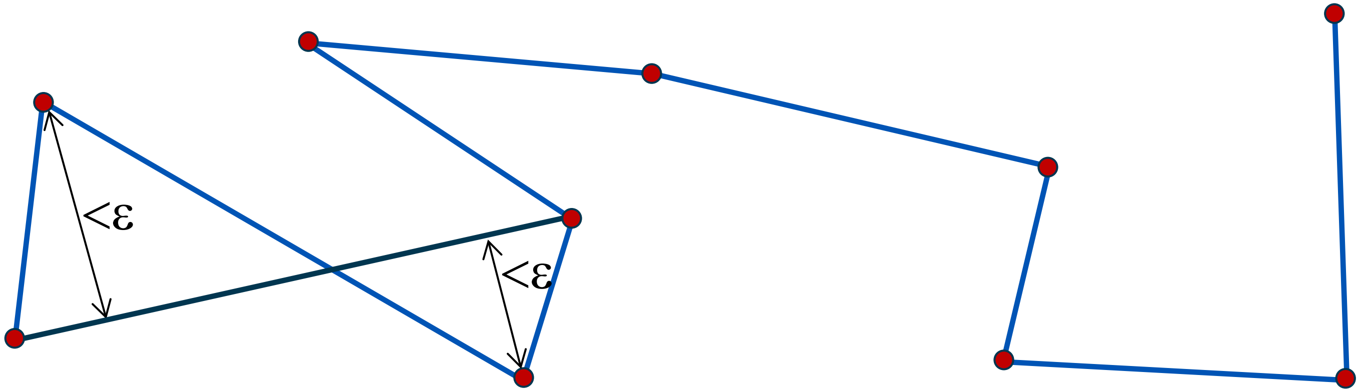
Imai-Iri

Find all possible valid shortcuts



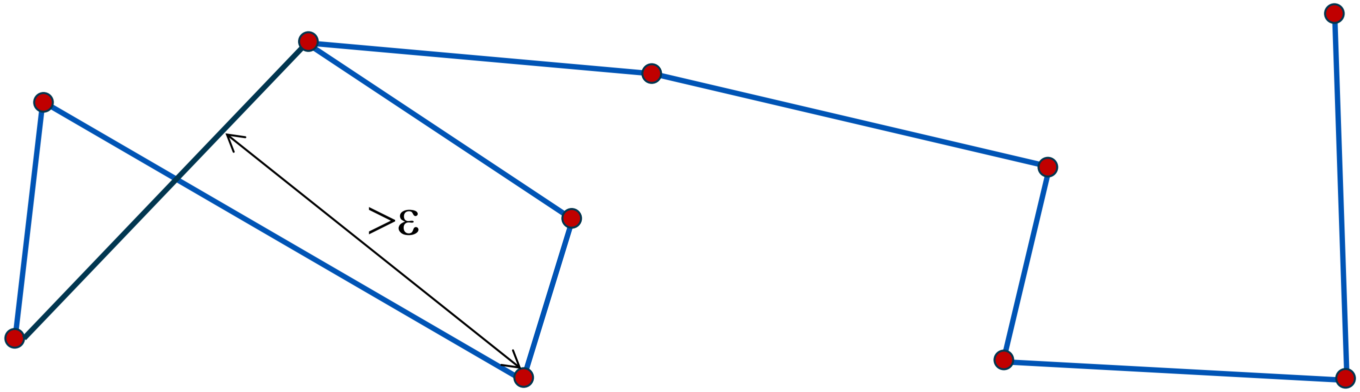
Imai-Iri

Find all possible valid shortcuts



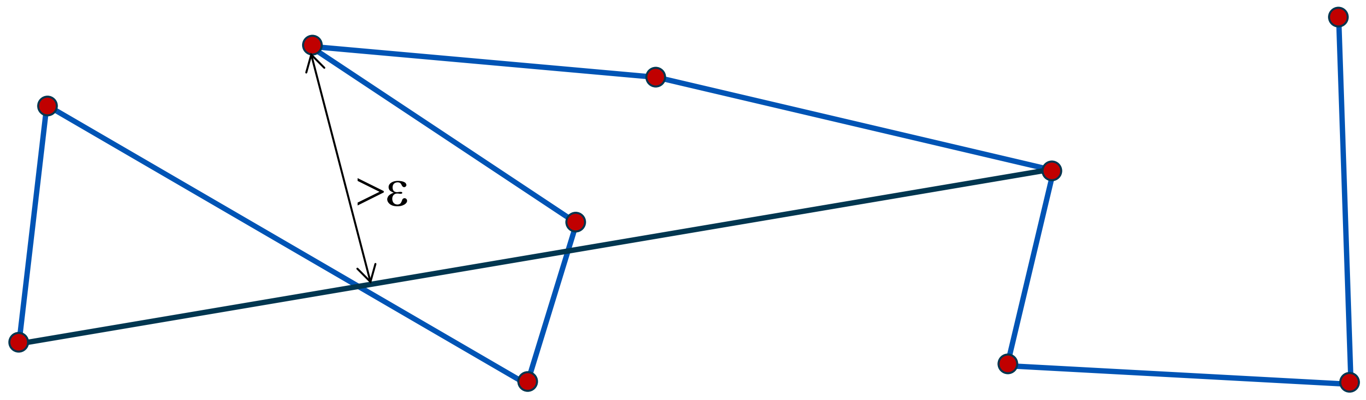
Imai-Iri

Find all possible valid shortcuts



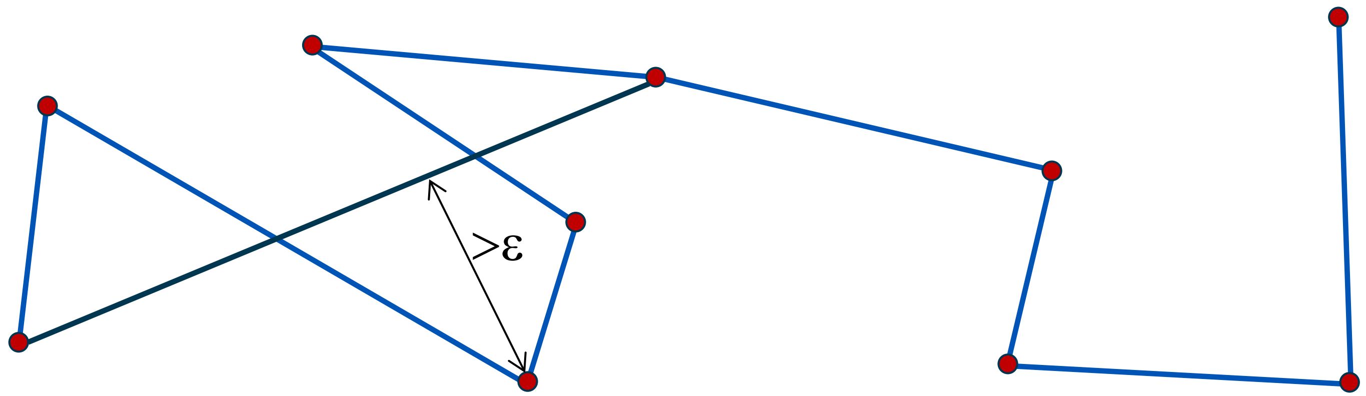
Imai-Iri

Find all possible valid shortcuts



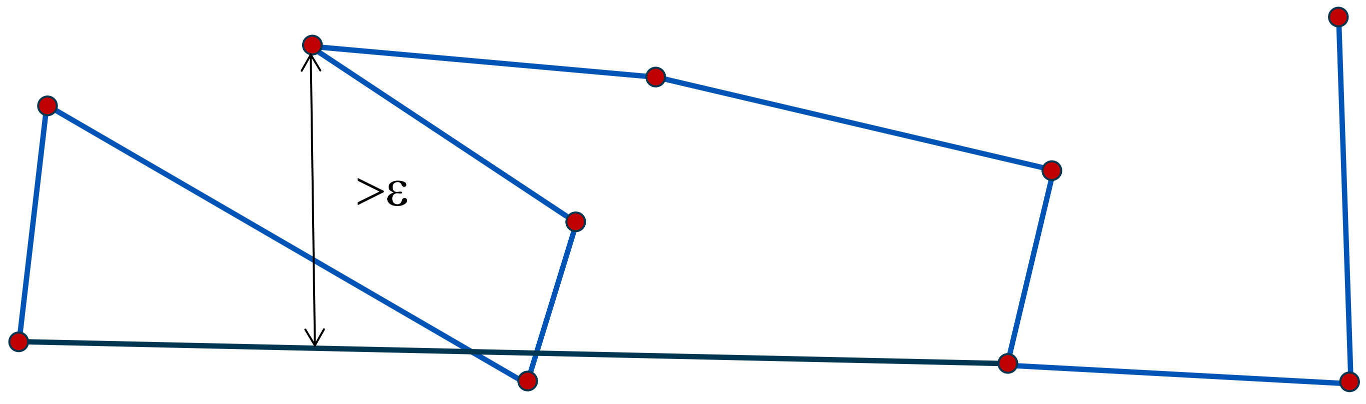
Imai-Iri

Find all possible valid shortcuts



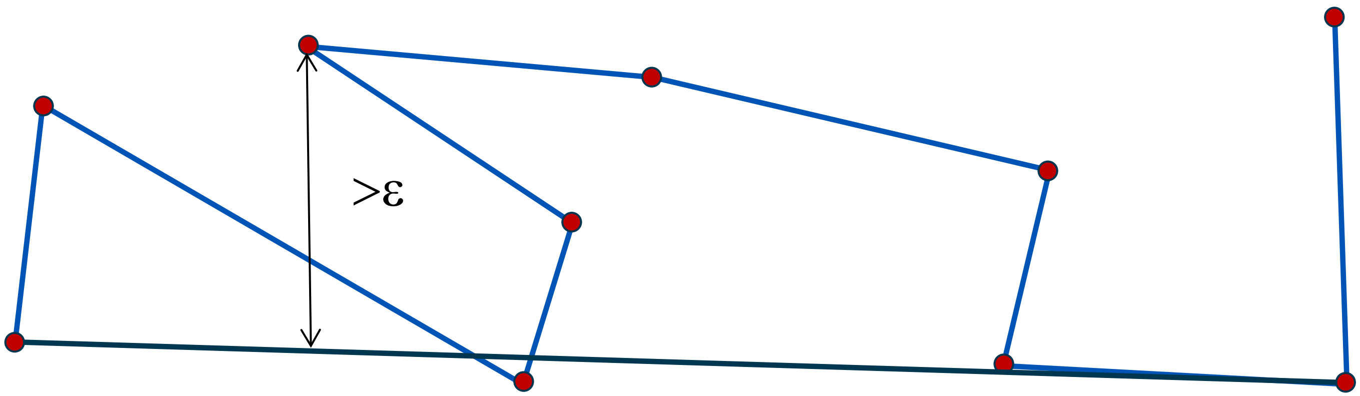
Imai-Iri

Find all possible valid shortcuts



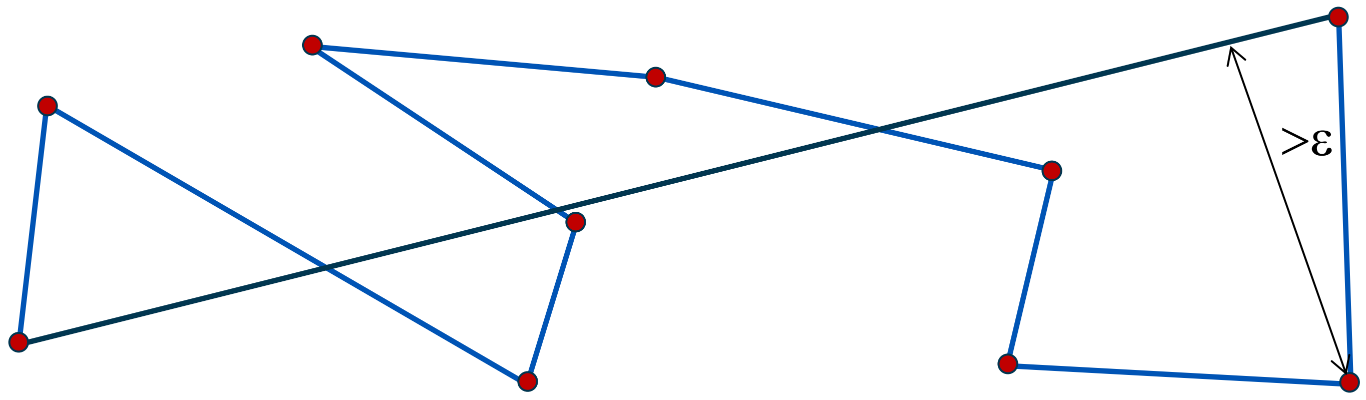
Imai-Iri

Find all possible valid shortcuts



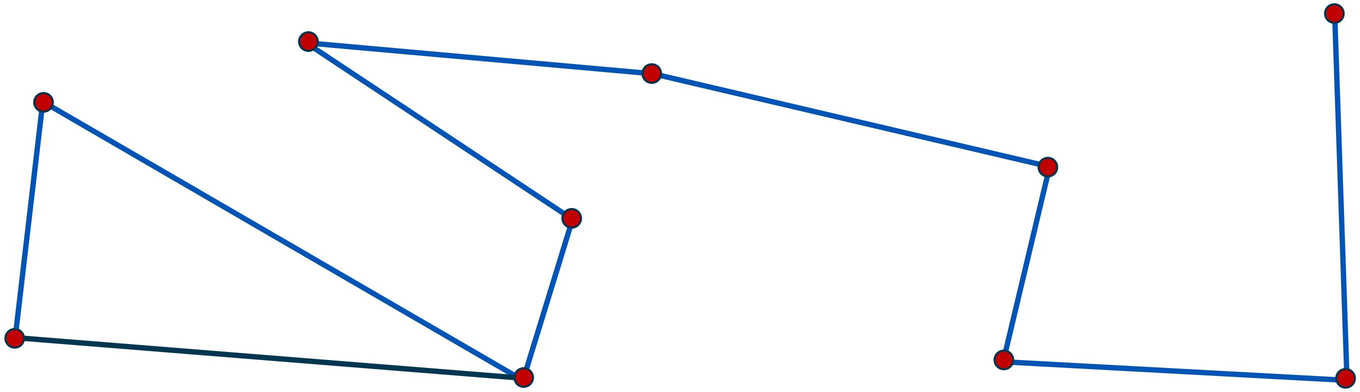
Imai-Iri

Find all possible valid shortcuts



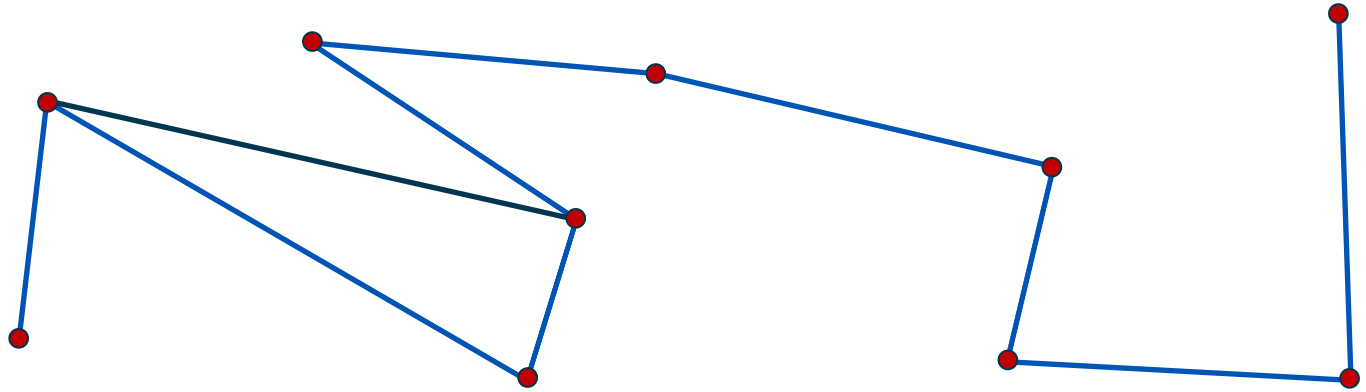
Imai-Iri

Find all possible valid shortcuts



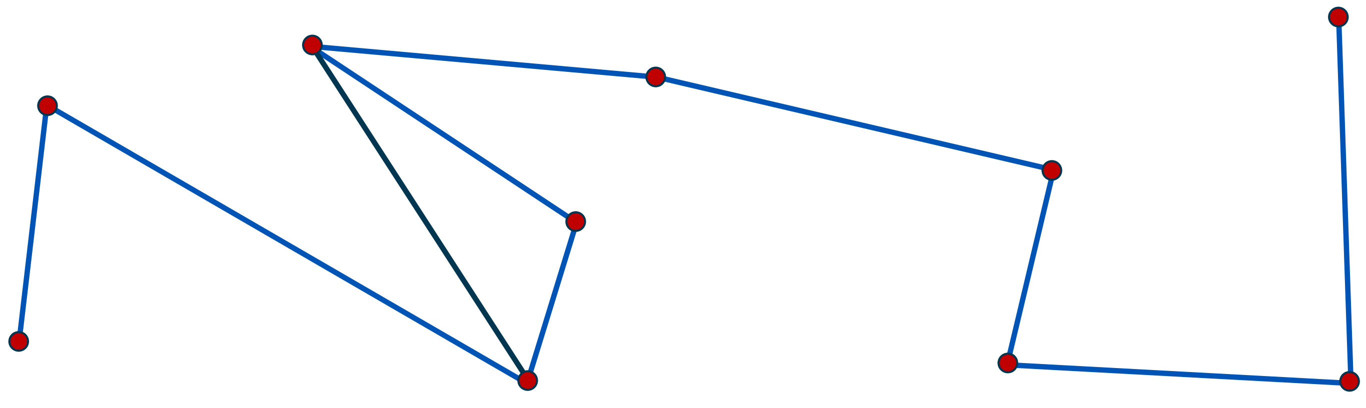
Imai-Iri

Find all possible valid shortcuts



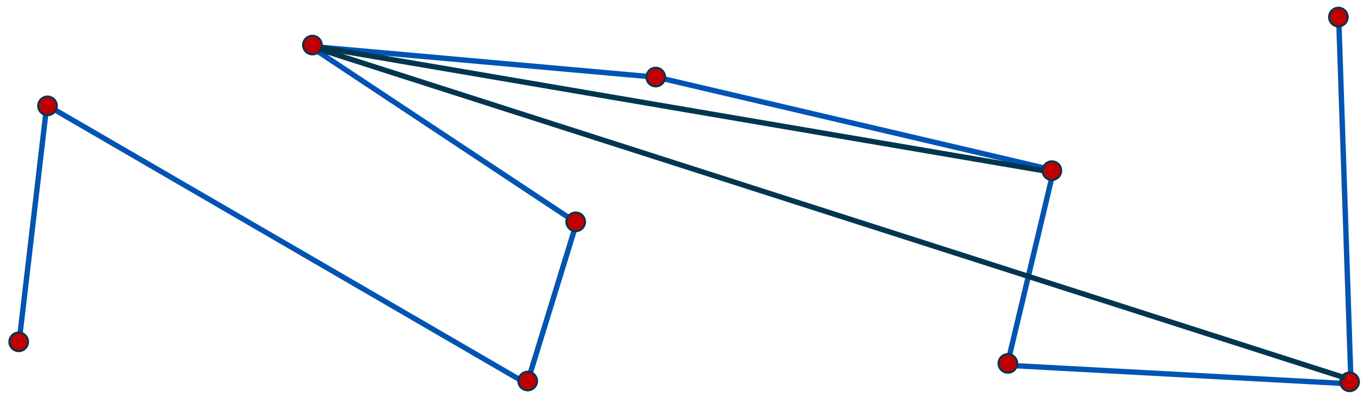
Imai-Iri

Find all possible valid shortcuts



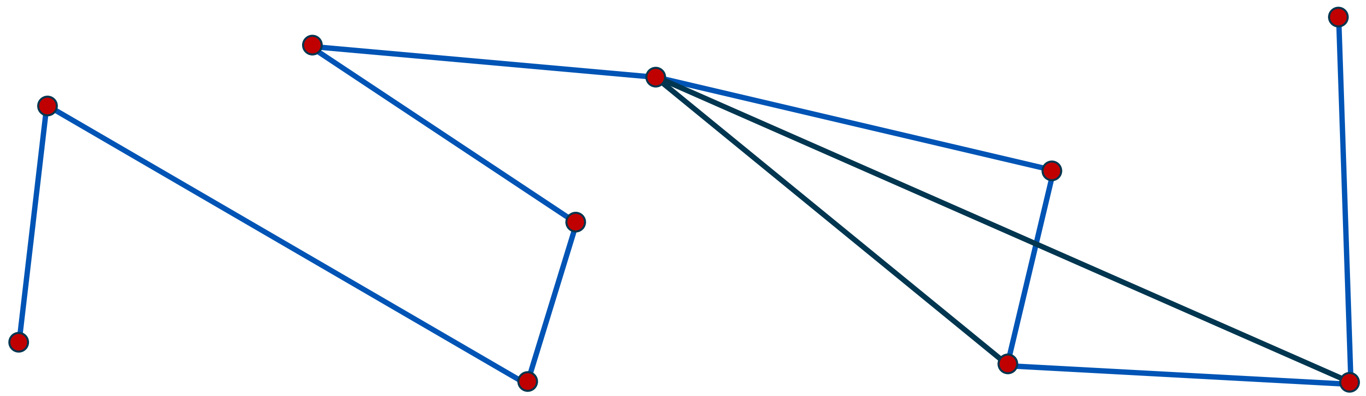
Imai-Iri

Find all possible valid shortcuts



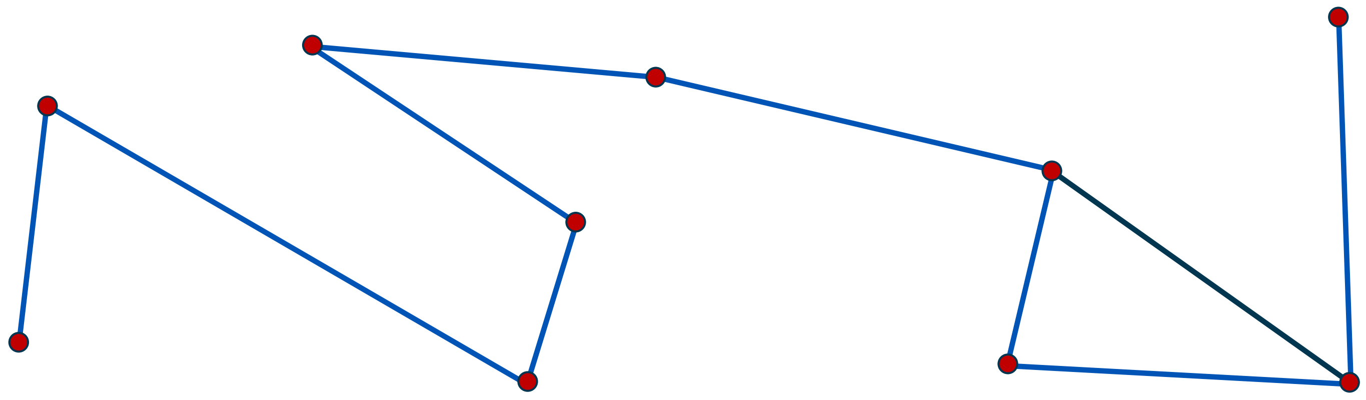
Imai-Iri

Find all possible valid shortcuts



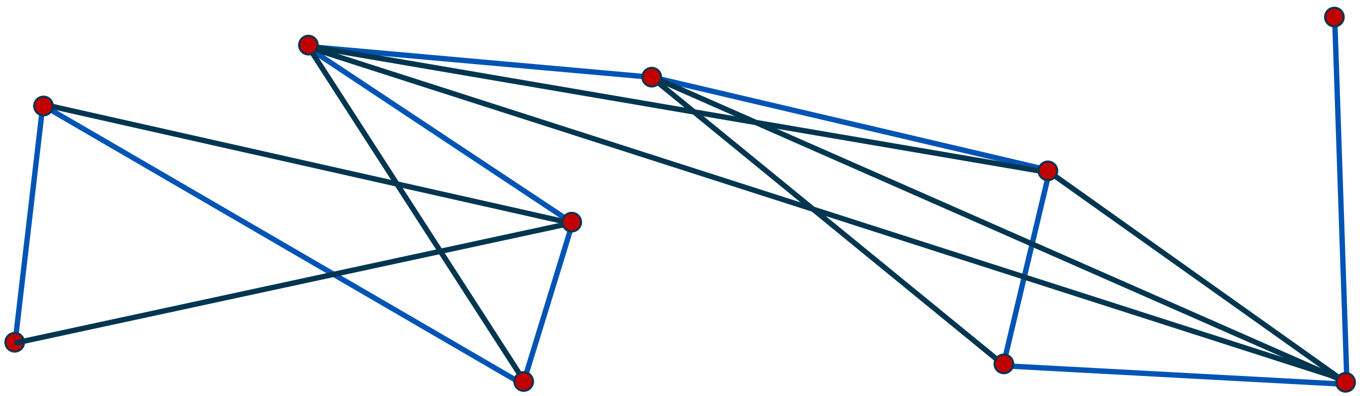
Imai-Iri

Find all possible valid shortcuts



Imai-Iri

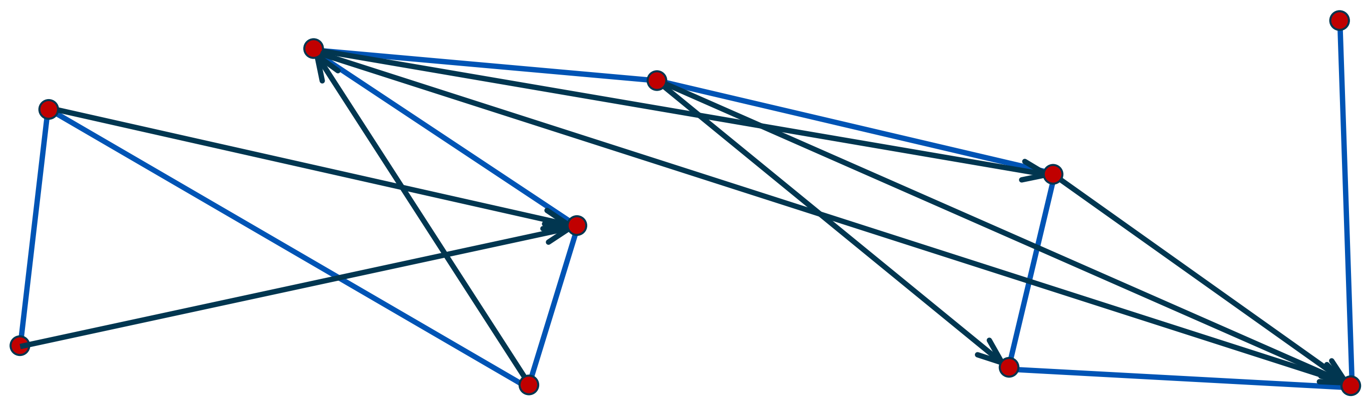
All possible shortcuts!



Imai-Iri

1. Build a directed graph of valid shortcuts.

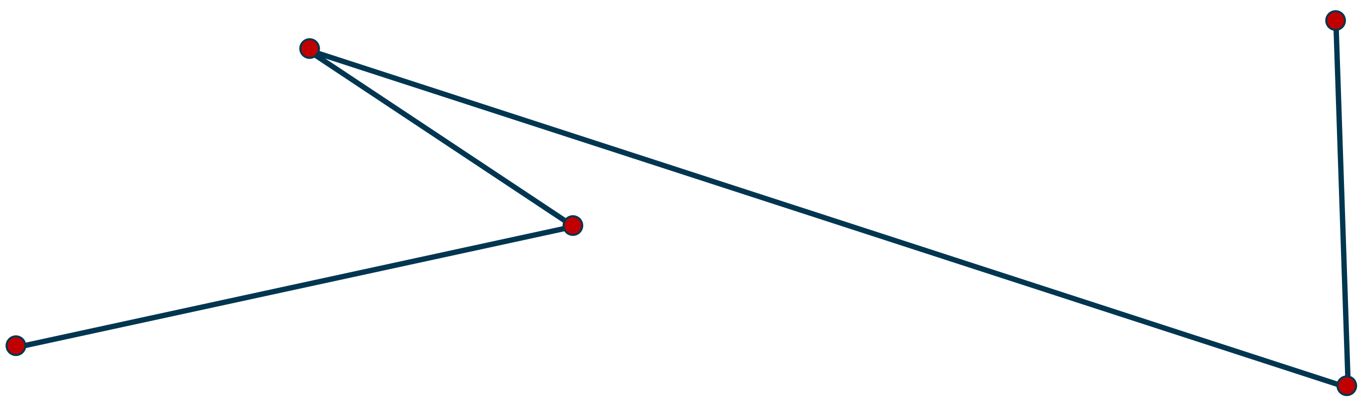
2. Compute a shortest path from p_1 to p_n using breadth-first search.



Imai-Iri

1. Build a directed graph of valid shortcuts.

2. Compute a shortest path from p_1 to p_n using breadth-first search.



Imai-Iri

Brute force running time: ?

#possible shortcuts ?

Analysis: Imai-Iri

Brute force running time: *?*

#possible shortcuts ?

Running time: $O(n^3)$

O(n^2) possible shortcuts

O(n) per shortcut $\Rightarrow$ $O(n^3)$ to build graph

O(n^2) BFS in the graph

Output: A path with minimum number of edges

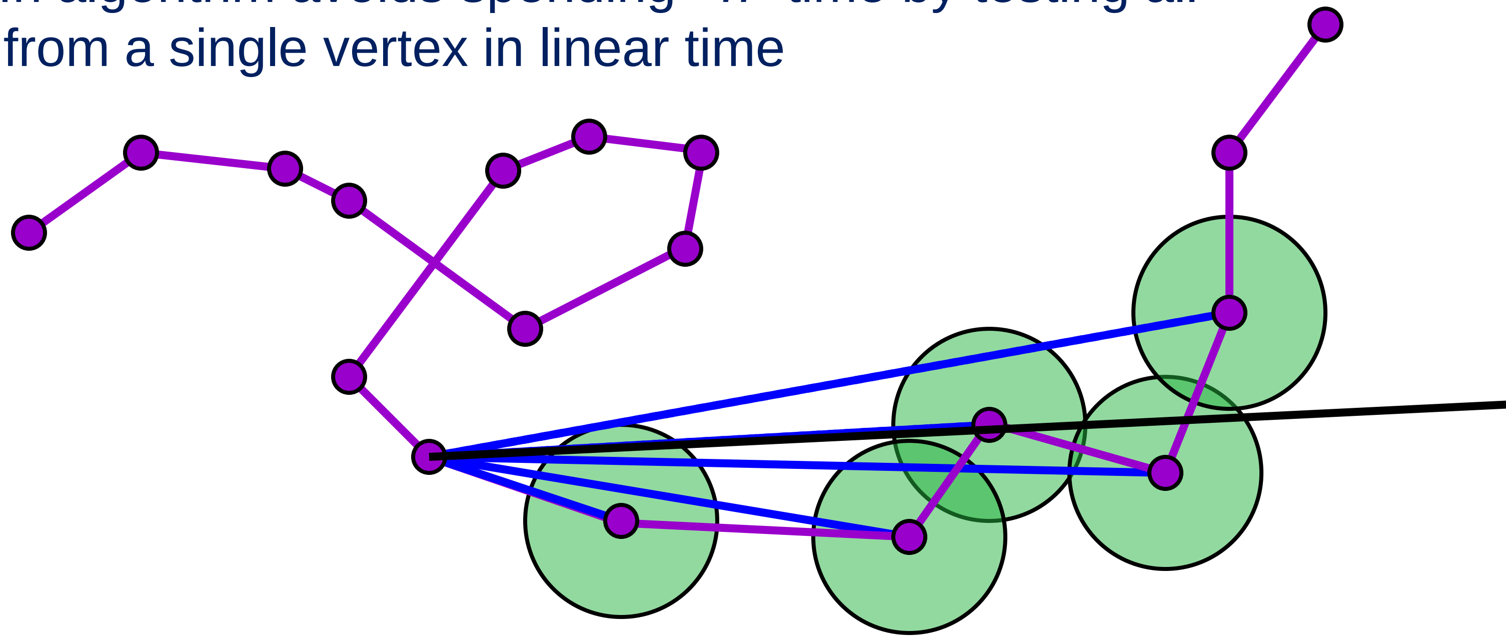
Improvements:

Chan and Chin'92: $O(n^2)$

for Min-ε: $O(n^2 \log n)$

Speeding up Imai-Iri

- The graph can have $\sim n^2$ edges; testing one shortcut takes time linear in the number of vertices in between
- The Imai-Iri algorithm avoids spending $\sim n^3$ time by testing all shortcuts from a single vertex in linear time



Speeding up Imai-Iri

A shortcut (p_i p_j) is feasible

iff it intersects all ε-disks of vertices inbetween (i+1,…,j-1)

iff both rays (from p_i through p_j and from p_j through p_i) intersect all ε-disks of vertices inbetween

Checking if a ray intersects all disks can be done efficiently:

Algorithm

for i=1,...,n

  $W_{i,i}$ = whole plane

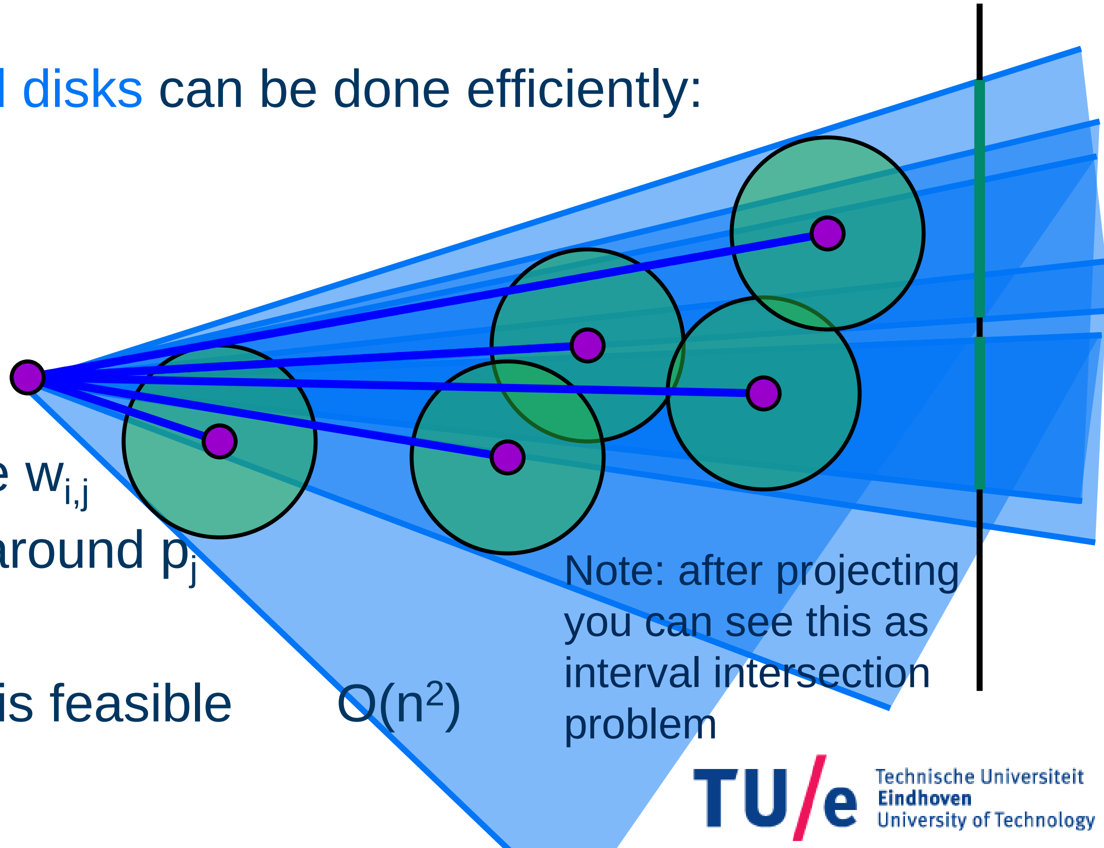
  for j = i+1,...,n

    $W_{i,j} = W_{i,j-1}$ intersect wedge $w_{i,j}$

      defined by p_i and ε-disk around p_j

    if $W_{i,j}$ = empty then break

    if ray $r_{i,j}$ in $W_{i,j}$ then (p_i p_j) is feasible $O(n^2)$



Note: after projecting you can see this as interval intersection problem

Imai-Iri with Fréchet distance

The framework of Imai-Iri can also be used for simplification under the Fréchet distance

- Min-vertices (for given ε) :

  solve decision problem for each edge: $O(n^3)$

- Min-ε (for given length of simplification):

  solve computation problem for each edge: $O(n^3 \log n)$

Results so far

- RDP: $O(n^2)$ time (simple and fast in practice)
- SimpleSimp: $O(n)$ time (simple and fast in practice)
- Output: A path with no bound on the size of the path

- Imai-Iri: $O(n^2)$ time by Chan and Chin
- Output: A path with minimum number of edges

- RDP and II use the Hausdorff error measure, SimpleSimp uses Fréchet, II can use Fréchet but slow

- Question: Can we get something that is simple, fast and has a worst-case bound using the Fréchet error measure?

Agarwal et al.

Agarwal, Har-Peled, Mustafa and Wang, 2002

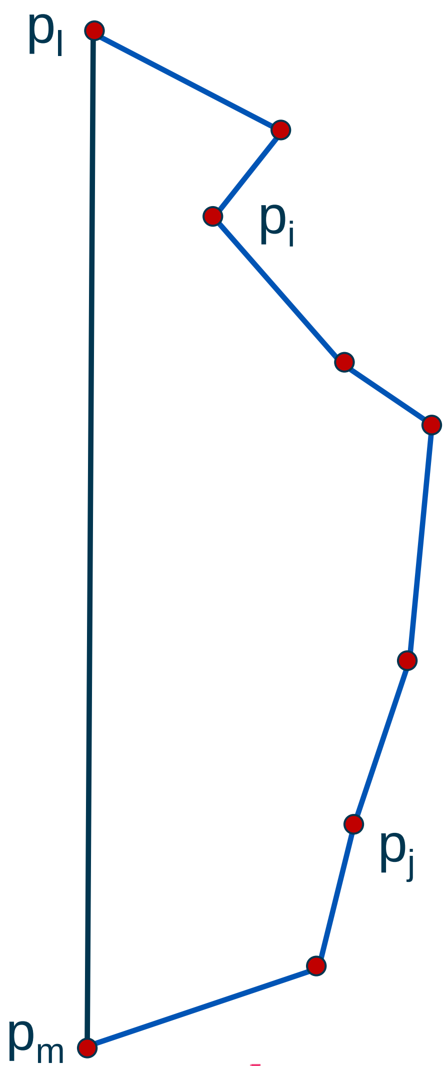
- Time: Running time O(*n log n*)

- Measure: Fréchet distance
-
- Output: Path has at most the same complexity as a minimum link (ε/2)-simplification

Analysis - Agarwal et al.

- Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve

- Notation: Let $\delta(p_i p_j)$ denote the Fréchet distance between (p_i, p_j) and the subpath $\pi(p_i, p_j)$ of P between p_i and p_j.



Algorithm - Agarwal et al.

Algorithm($P = \langle p_1,\ldots,p_n\rangle$, ε)

i := 1

P'= $\langle p_1\rangle$

while i < n do

find any j>i such that

$(\delta(p_i,p_j) \leq \varepsilon$ and $(\delta(p_i,p_{j+1}) > \varepsilon)$ or

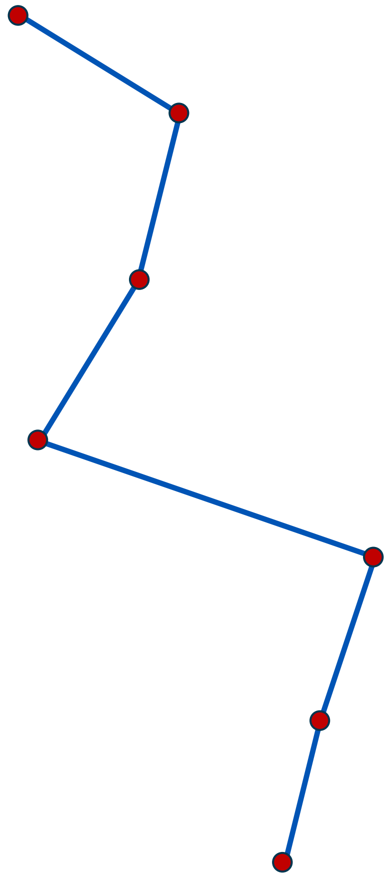
$(\delta(p_i,p_j) \leq \varepsilon$ and $j=n)$

P'= concat(P,$\langle p_j\rangle$)

i := j

end

return P'



Technische Universiteit Eindhoven University of Technology

Algorithm - Agarwal et al.

Algorithm($P = \langle p_1,\ldots,p_n\rangle$, ε)

i := 1

P'= $\langle p_1 \rangle$

while i < n do

find any j>i such that

$(\delta(p_i,p_j) \le \varepsilon$ and $(\delta(p_i,p_{j+1}) > \varepsilon)$ or

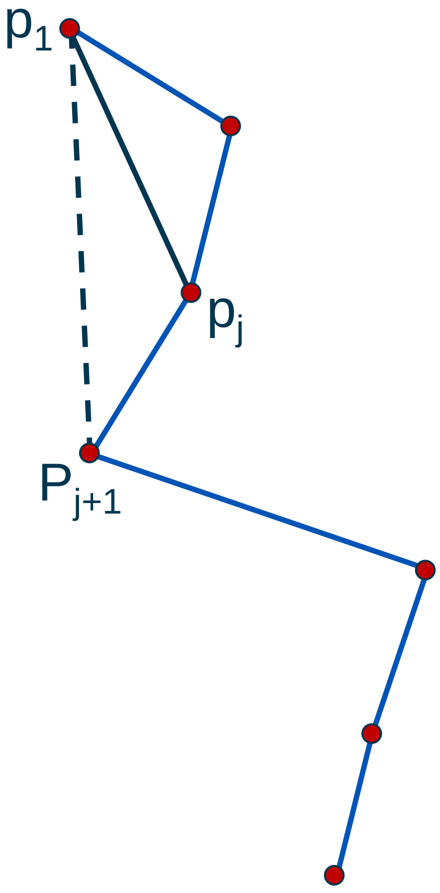
$(\delta(p_i,p_j) \le \varepsilon$ and j=n)

P'= concat(P,$\langle p_j \rangle$)

i := j

end

return P'



TU/e Technische Universiteit Eindhoven University of Technology

Algorithm - Agarwal et al.

Algorithm($P = \langle p_1,\ldots,p_n \rangle$, ε)

i := 1

P'= $\langle p_1 \rangle$

while i < n do

find any j>i such that

$(\delta(p_i,p_j) \leq \varepsilon$ and $(\delta(p_i,p_{j+1}) > \varepsilon)$ or

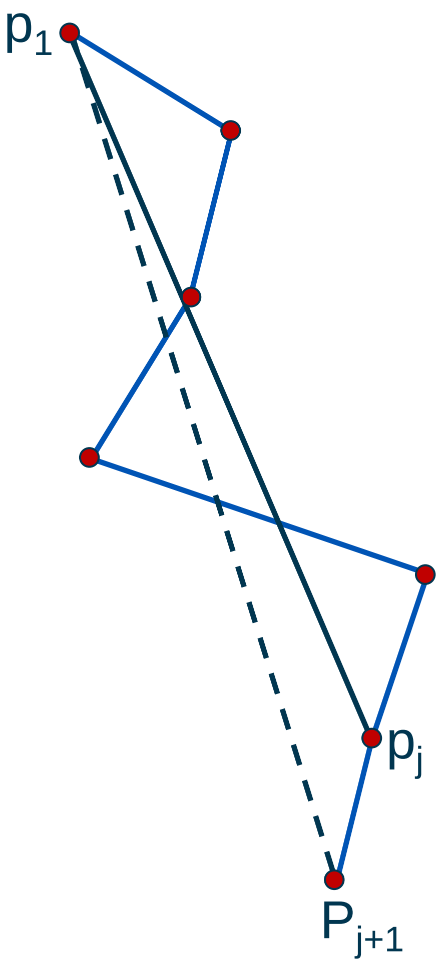
$(\delta(p_i,p_j) \leq \varepsilon$ and $j=n)$

P'= concat(P,$\langle p_j \rangle$)

i := j

end

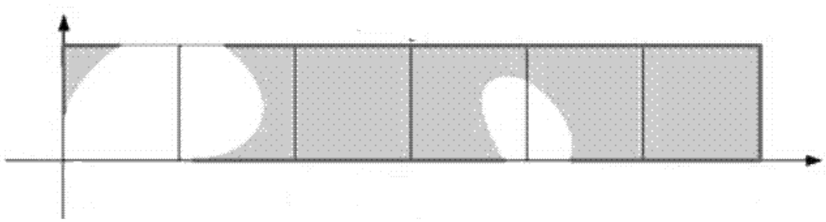
return P'

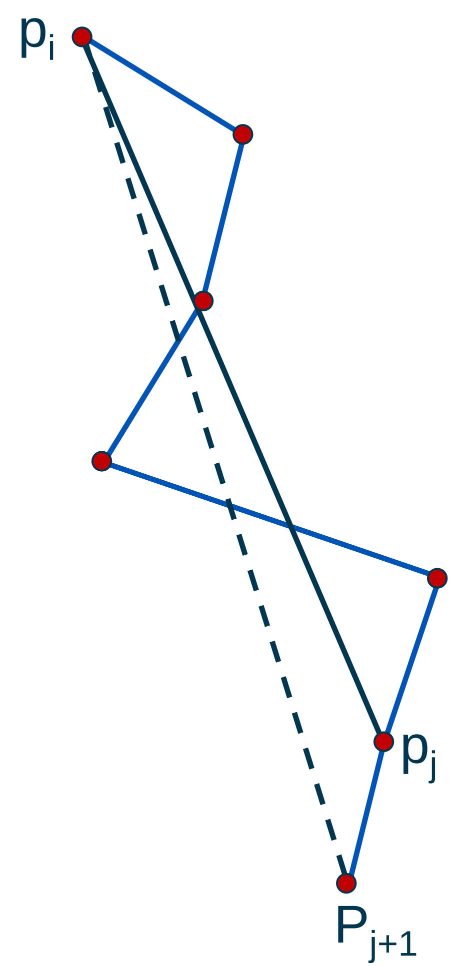


Algorithm - Agarwal et al.

- Efficient algorithm?

- Recall: Computing the Fréchet distance between (p_i,p_j) and $\pi(p_i,p_j)$ can be done in $O(j-i)$ time



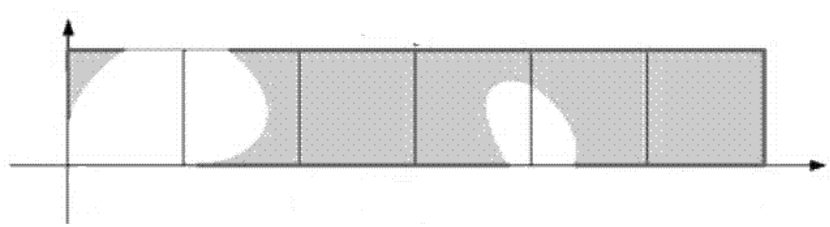


Technische Universiteit Eindhoven University of Technology

Algorithm - Agarwal et al.

- Efficient algorithm?

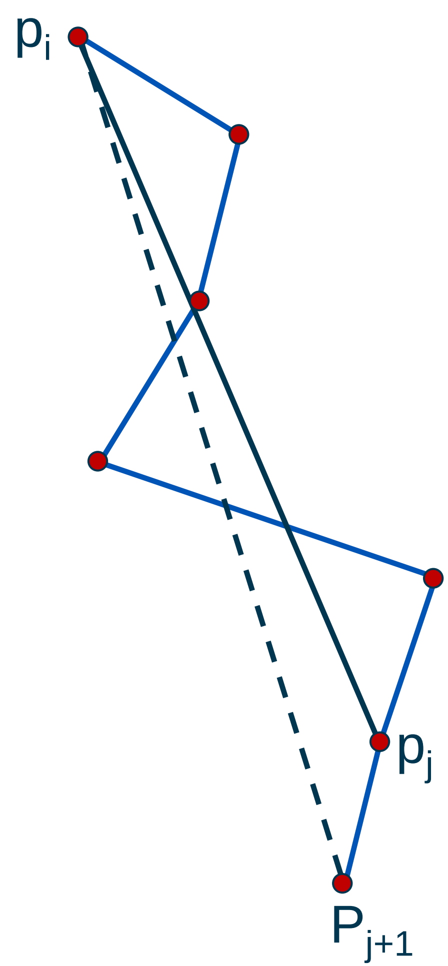
- Recall: Computing the Fréchet distance between (p_i, p_j) and $\pi(p_i, p_j)$ can be done in $O(j-i)$ time



- Finding the first j such that

  $\delta(p_i, p_j) \leq \varepsilon$ and $\delta(p_i, p_{j+1}) > \varepsilon$

  takes $O(n^2)$ time.



TU/e Technische Universiteit Eindhoven University of Technology

Algorithm - Agarwal et al.

- How can we speed up the search for p_j?

- Note that we just want to find any vertex p_j such that

$$\delta(p_i,p_j) \leq \varepsilon \text{ and } \delta(p_i,p_{j+1}) > \varepsilon$$

Idea: Search for p_j using exponential search followed by binary search!

Algorithm - Agarwal et al.

- Exponential search:

  Test p_{i+1}, p_{i+2}, p_{i+4}, p_{i+8}… until found p_{i+2^k}, such that $\delta(p_i,p_{i+2^k}) > \varepsilon$.

- Binary search:

  Search for p_j between $p_{i+2^{k-1}}$ and p_{i+2^k} s.t.

  $\delta(p_i,p_j) \leq \varepsilon$ and $\delta(p_i,p_{j+1}) > \varepsilon$.

Algorithm - Agarwal et al.

- **Exponential search:**

  Test p_{i+1}, p_{i+2}, p_{i+4}, p_{i+8}… until found p_{i+2^k}, such that $\delta(p_i,p_{i+2^k}) > \varepsilon$.

- **Binary search:**

  Search for p_j between $p_{i+2^{k-1}}$ and p_{i+2^k} s.t.

  $\delta(p_i,p_j) \leq \varepsilon$ and $\delta(p_i,p_{j+1}) > \varepsilon$.

if $\leq \varepsilon$ then

search right

$\leq \varepsilon$ | | $> \varepsilon$
$i+2^{k-1}$ | | $i+2^k$

Algorithm - Agarwal et al.

- **Exponential search:**

  Test p_{i+1}, p_{i+2}, p_{i+4}, p_{i+8}… until found p_{i+2^k}, such that $\delta(p_i,p_{i+2^k}) > \varepsilon$.

- **Binary search:**

  Search for p_j between $p_{i+2^{k-1}}$ and p_{i+2^k} s.t.

  $\delta(p_i,p_j) \leq \varepsilon$ and $\delta(p_i,p_{j+1}) > \varepsilon$.

if $\leq \varepsilon$ then

search right

$\leq \varepsilon$

$> \varepsilon$

$i+2^{k-1}$

$i+2^k$

if $> \varepsilon$ then

search left

Algorithm - Agarwal et al.

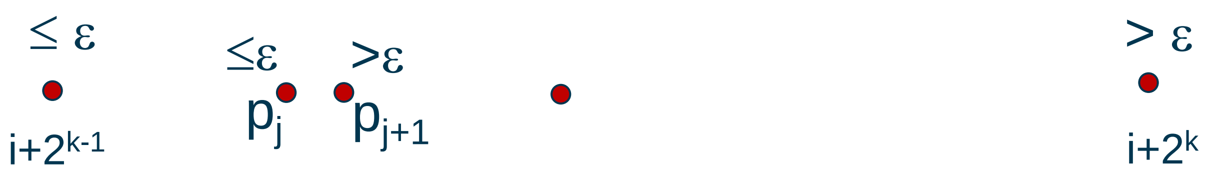
- **Exponential search:**

  Test p_{i+1}, p_{i+2}, p_{i+4}, p_{i+8}… until found p_{i+2^k}, such that $\delta(p_i,p_{i+2^k}) > \varepsilon$.

- **Binary search:**

  Search for p_j between $p_{i+2^{k-1}}$ and p_{i+2^k} s.t.

  $$\delta(p_i,p_j) \leq \varepsilon \text{ and } \delta(p_i,p_{j+1}) > \varepsilon.$$



Algorithm - Agarwal et al.

- Exponential search:

  Test p_{i+1}, p_{i+2}, p_{i+4}, p_{i+8}… until found p_{i+2^k}, such that $\delta(p_i,p_{i+2^k}) > \varepsilon$.

  Iterations: $O(\mathit{log}\ n)$

- Binary search:

  Search for p_j between $p_{i+2^{k-1}}$ and p_{i+2^k} s.t.

  $\delta(p_i,p_j) \leq \varepsilon$ and $\delta(p_i,p_{j+1}) > \varepsilon$.

  Iterations: $O(\mathit{log}\ n)$

- The algorithm computes indices $i_j = i_{j-1} + r_j$ with

  r_j in $[2^l,2^{l+1}]$ and $\sum r_j = n$.

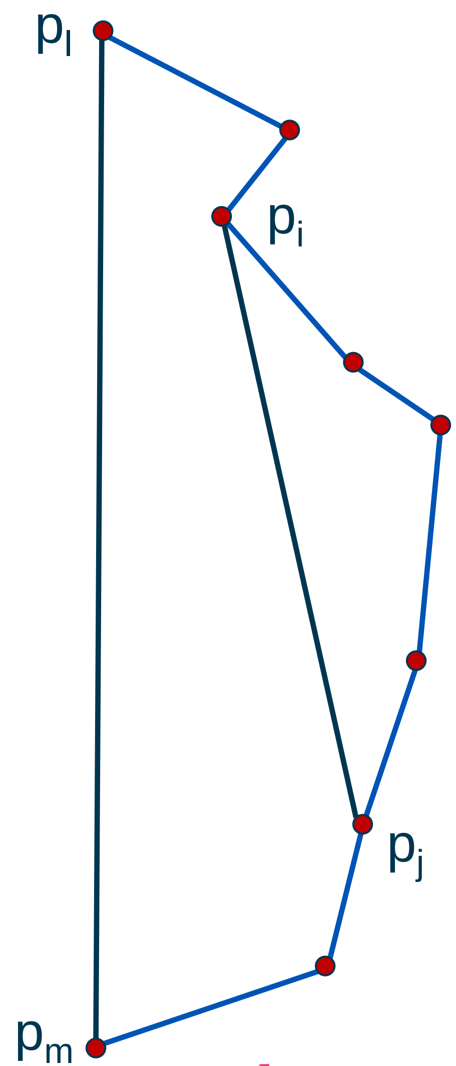
- To compute i_j requires $l = O(\log n)$ iterations of both searches, costing $O(l \cdot 2^l)$: sums to overall $O(n \log n)$

Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

For $l \leq i \leq j \leq m$, $\delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.

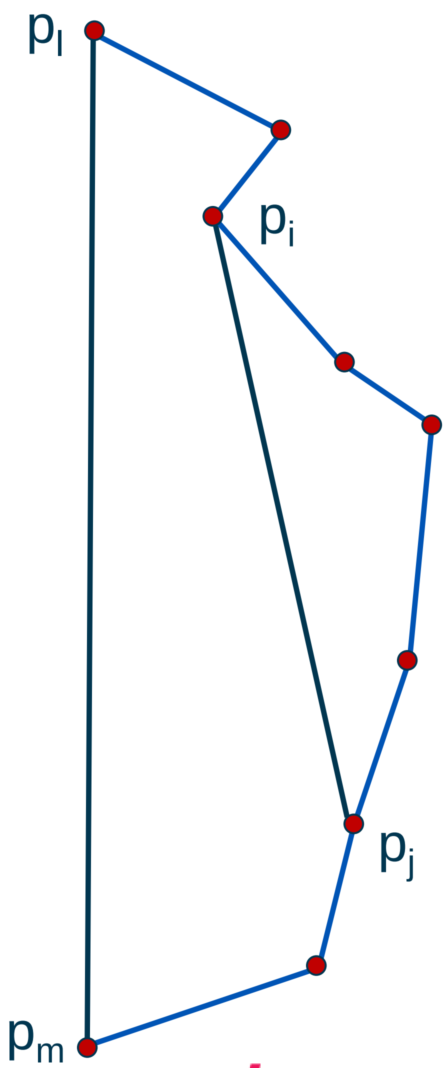


Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

For $l \leq i \leq j \leq m$, $\delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.



Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

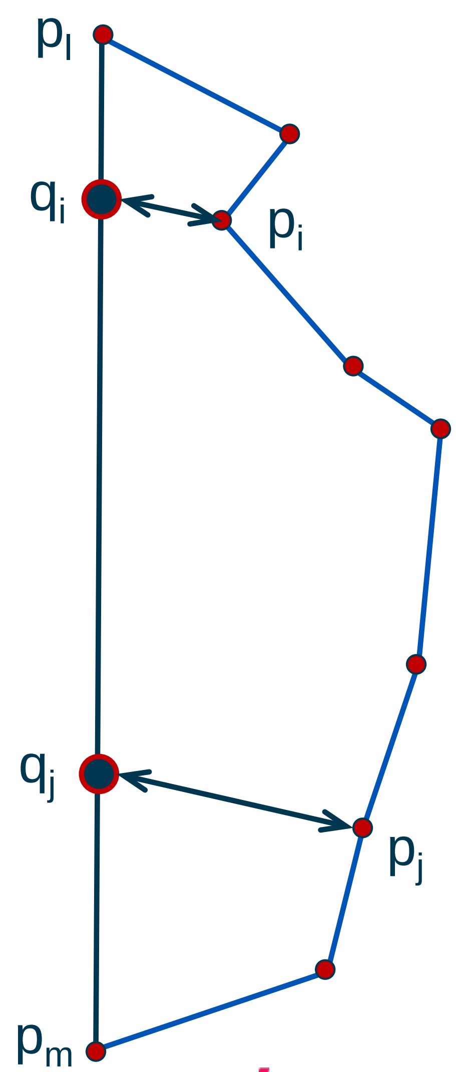
For $l \leq i \leq j \leq m$, $\delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.

Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

1. $\delta(\pi(p_i p_j),(q_i q_j)) \leq \lambda$



Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \dots, p_n \rangle$ be a polygonal curve.

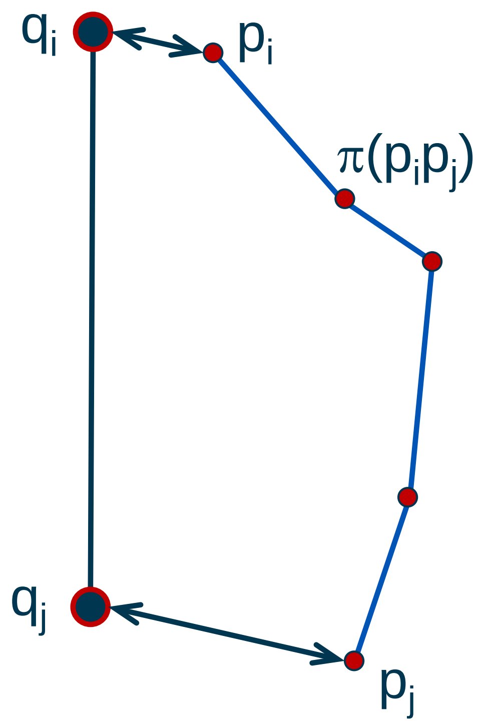
For $l \le i \le j \le m$, $\delta(p_i p_j) \le 2 \cdot \delta(p_l p_m)$.

Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

1. $\delta(\pi(p_i p_j), (q_i q_j)) \le \lambda$

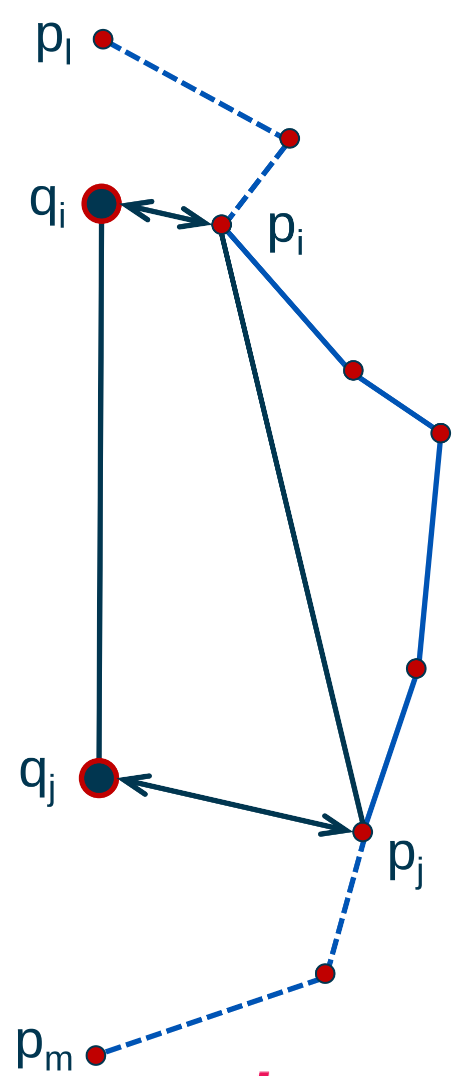


Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

For $l \leq i \leq j \leq m$, $\quad \delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.



Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

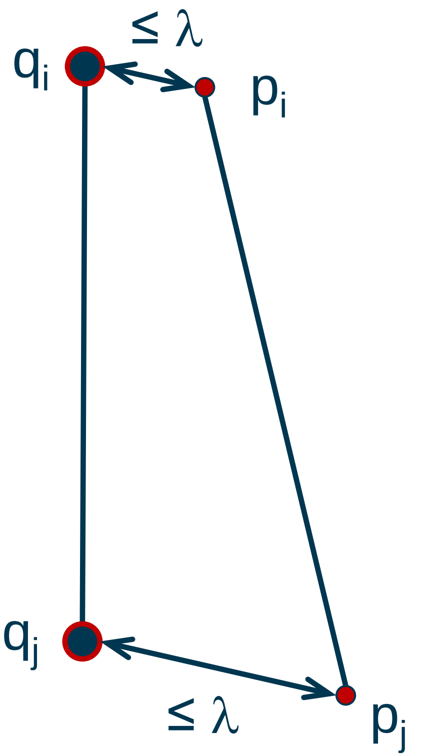
1. $\delta(\pi(p_i p_j),(q_i q_j)) \leq \lambda$
2. $\delta((q_i q_j),(p_i p_j)) \leq \lambda$

Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

For $l \leq i \leq j \leq m$, $\delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.



Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

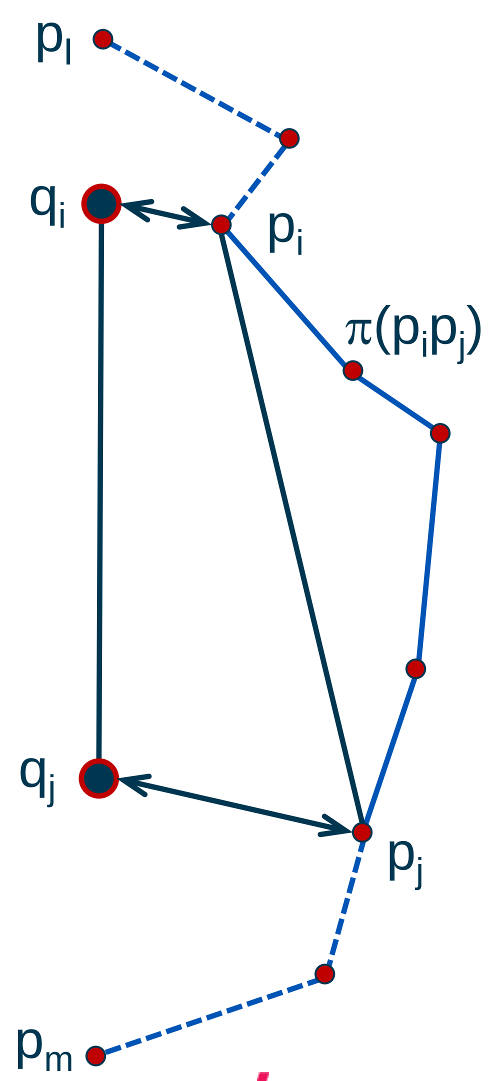
1. $\delta(\pi(p_i p_j),(q_i q_j)) \leq \lambda$
2. $\delta((q_i q_j),(p_i p_j)) \leq \lambda$

Analysis - Agarwal et al.

Lemma 1:

Let $P = \langle p_1, p_2, \ldots, p_n \rangle$ be a polygonal curve.

For $l \leq i \leq j \leq m$, $\delta(p_i p_j) \leq 2 \cdot \delta(p_l p_m)$.



Proof:

Fix matching that realises $\delta(p_l p_m)$.

Set $\lambda = \delta(p_l p_m)$.

1. $\delta(\pi(p_i p_j),(q_i q_j)) \leq \lambda$
2. $\delta((q_i q_j),(p_i p_j)) \leq \lambda$

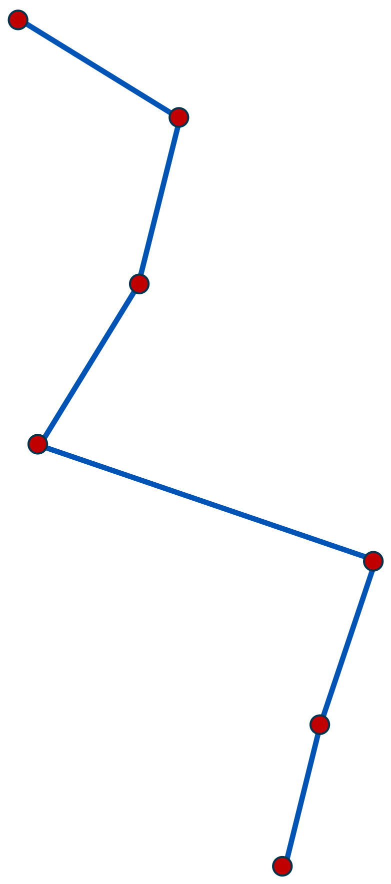
$\delta(\pi(p_i p_j),p_i p_j) \leq \delta(\pi(p_i p_j),(q_i q_j)) + \delta((q_i q_j),(p_i p_j)) \leq 2\lambda$ $\square$

Analysis - Agarwal et al.

Theorem: $\delta(P,P') \leq \varepsilon$ and $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

- $\delta(P,P') \leq \varepsilon$ follows by construction.



Analysis - Agarwal et al.

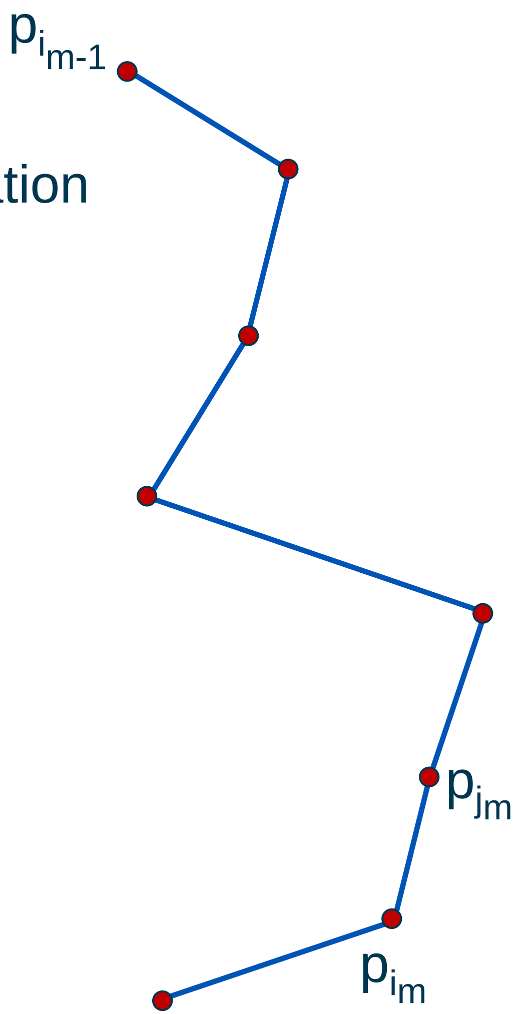
Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Prove (by induction) that $i_m \geq j_m$, $\forall m \geq 1$.



Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

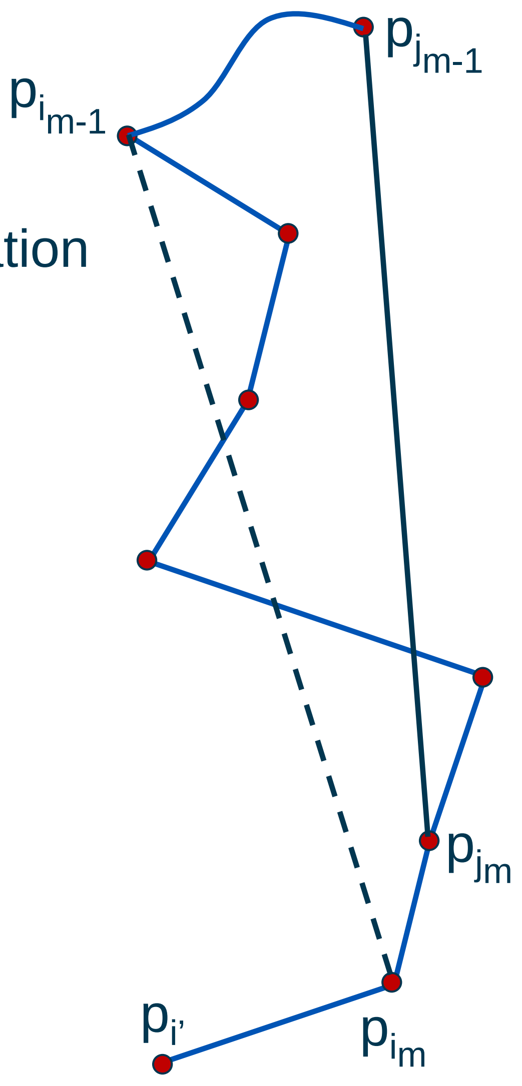
Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Prove (by induction) that $i_m \geq j_m$, $\forall m \geq 1$.

- Assume $i_{m-1} \geq j_{m-1}$ and let $i' = i_m + 1$.



Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

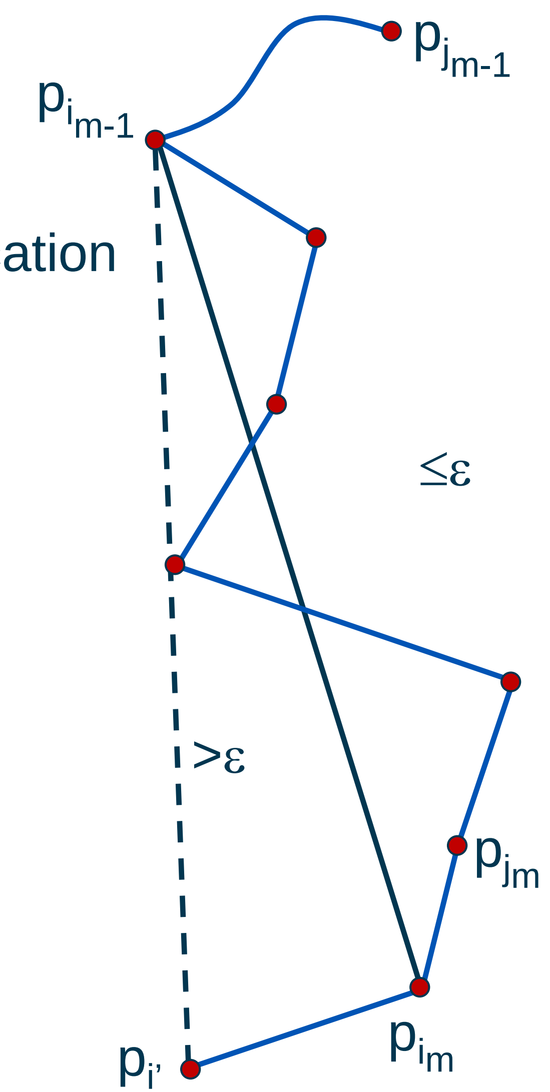
$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Prove (by induction) that $i_m \geq j_m$, $\forall m \geq 1$.

- Assume $i_{m-1} \geq j_{m-1}$ and let $i' = i_m+1$.
- By construction we have:

$$\delta(p_{i_{m-1}} p_{i'}) > \varepsilon \quad \text{and} \quad \delta(p_{i_{m-1}} p_{i'-1}) \leq \varepsilon$$

If $i' > j_m$ then we are done!



Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

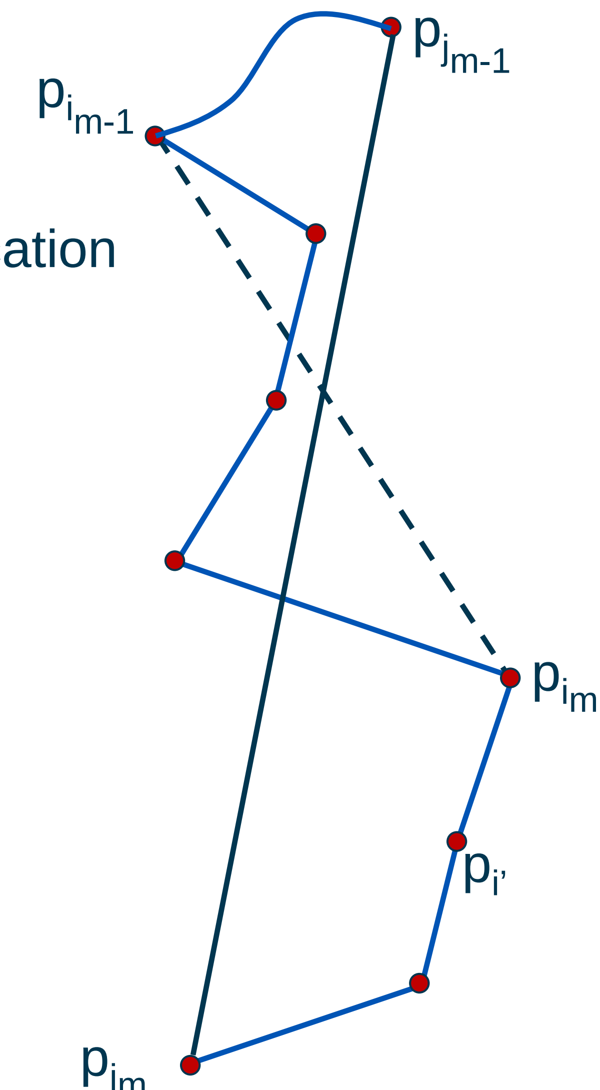
$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Prove (by induction) that $i_m \geq j_m$, $\forall m \geq 1$.

- Assume $i_{m-1} \geq j_{m-1}$ and let $i' = i_m + 1$.
- By construction we have:

$$\delta(p_{i_{m-1}} p_{i'}) > \varepsilon \quad \text{and} \quad \delta(p_{i_{m-1}} p_{i'-1}) \leq \varepsilon$$

Assume the opposite i.e. $i' \leq j_m$



Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

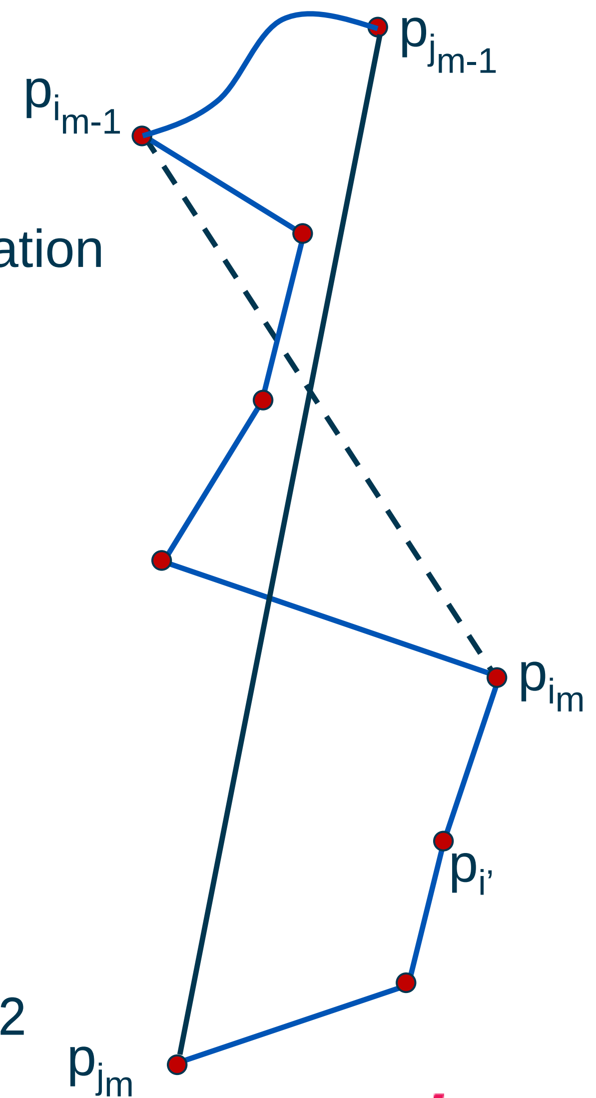
Prove (by induction) that $i_m \geq j_m$, $\forall m \geq 1$.

- Assume $i_{m-1} \geq j_{m-1}$ and let $i' = i_m + 1$.
- By construction we have:

$$\delta(p_{i_{m-1}} p_{i'}) > \varepsilon \quad \text{and} \quad \delta(p_{i_{m-1}} p_{i'-1}) \leq \varepsilon$$

Assume the opposite i.e. $i' \leq j_m$

Q is an $(\varepsilon/2)$-simplification $\Rightarrow \delta(p_{j_{m-1}} p_{j_m}) \leq \varepsilon/2$



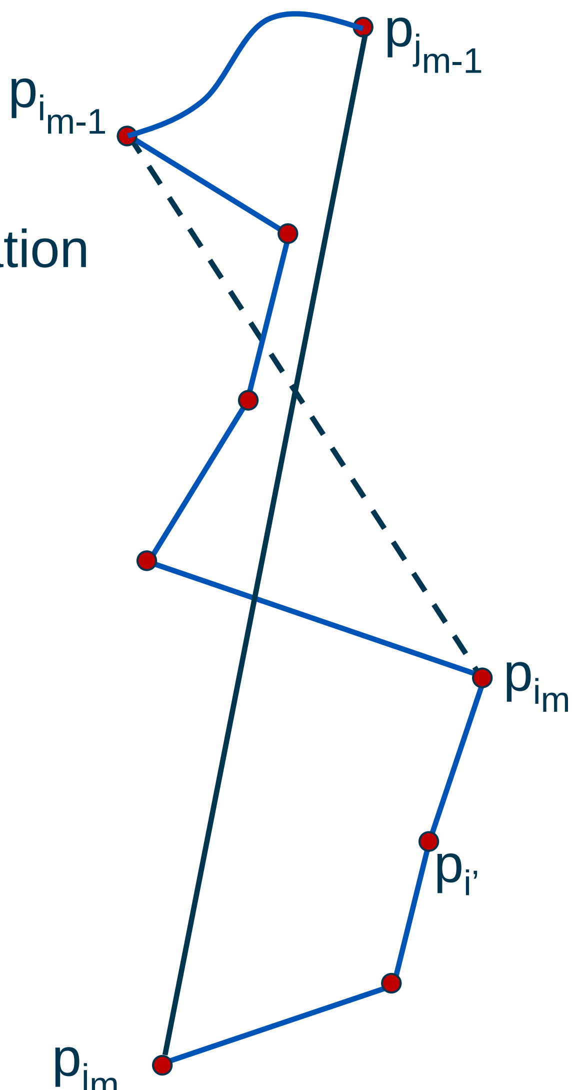
Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal ($\varepsilon/2$)-simplification

$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$



Assume the opposite i.e. $i' \leq j_m$

Q is an ($\varepsilon/2$)-simplification $\Rightarrow \delta(p_{j_{m-1}} p_{j_m}) \leq \varepsilon/2$

Analysis - Agarwal et al.

Theorem: $|P'| \leq P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

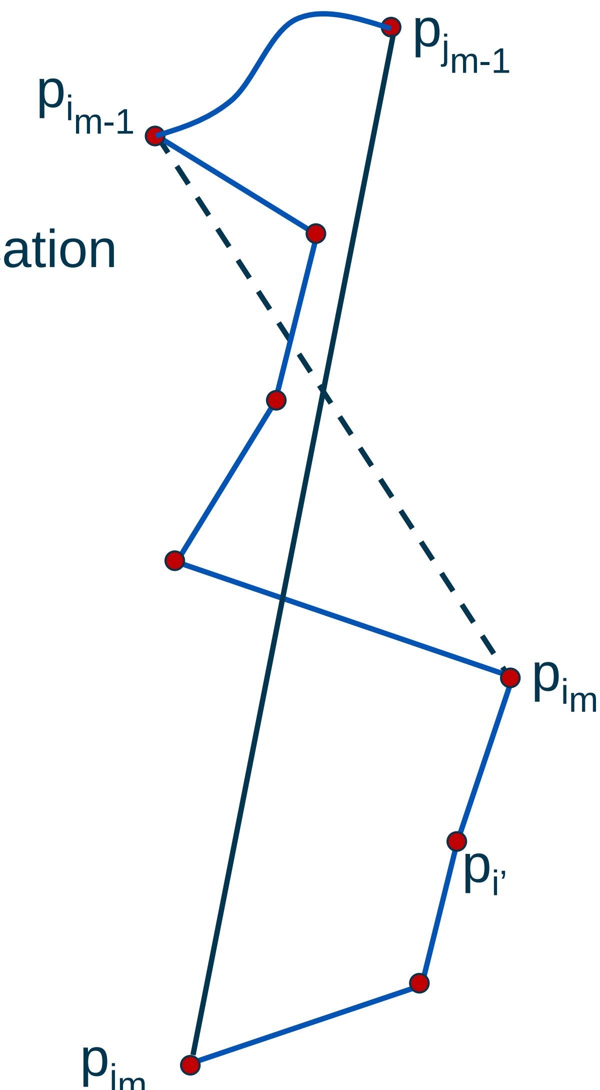
$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Assume the opposite i.e. $i' \leq j_m$

Q is an $(\varepsilon/2)$-simplification $\Rightarrow \delta(p_{j_{m-1}} p_{j_m}) \leq \varepsilon/2$

According to Lemma 1:

$$\delta(p_{i_{m-1}} p_{i'}) \leq 2\, \delta(p_{j_{m-1}} p_{j_m}) \leq \varepsilon$$



Analysis - Agarwal et al.

Theorem: $|P'| \le P_{opt}(\varepsilon/2)$

Proof:

$Q = \langle p_1 = p_{j_1}, \ldots, p_{j_l} = p_n \rangle$ - optimal $(\varepsilon/2)$-simplification

$P' = \langle p_1 = p_{i_1}, \ldots, p_{i_k} = p_n \rangle$

Assume the opposite i.e. $i' \le j_m$

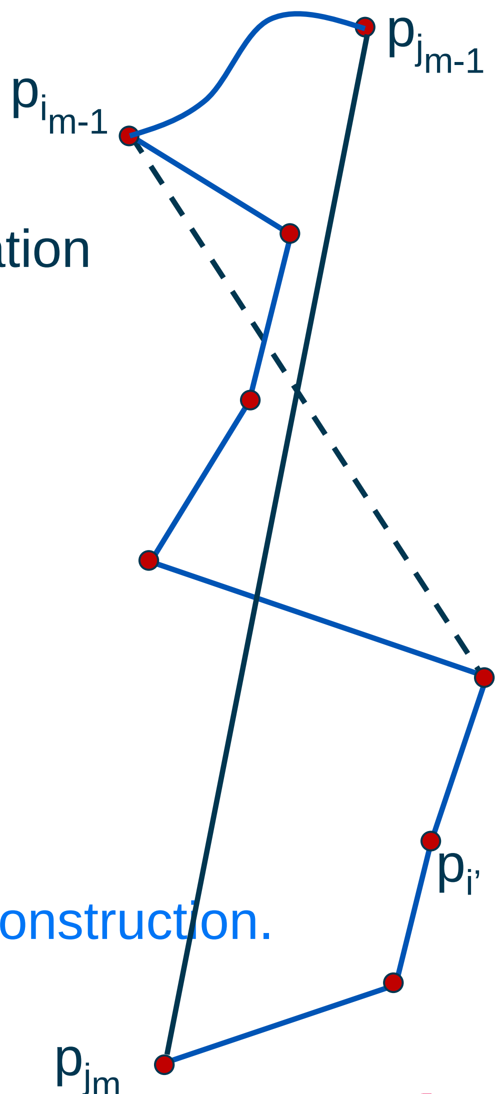
Q is an $(\varepsilon/2)$-simplification $\Rightarrow \delta(p_{j_{m-1}} p_{j_m}) \le \varepsilon/2$

According to Lemma 1:

$$\delta(p_{i_{m-1}} p_{i'}) \le 2\,\delta(p_{j_{m-1}} p_{j_m}) \le \varepsilon$$

This is a contradiction since $\delta(p_{j_{m-1}} p_{j_m}) > \varepsilon$ by construction.

$\Rightarrow i' > j_m$ and we are done!



Summary - Agarwal et al.

Theorem:

Given a polygonal curve P in R^d and a parameter $\varepsilon \geq 0$, an ε-simplification of P of size at most $P_{opt}(\varepsilon/2)$ can be constructed in $O(n \log n)$ time and $O(n)$ space.

Summary

Ramer–Douglas–Peucker 1973

- Used very often in practice. Easy to implement and fast in practice. Hausdorff error

Simple Simplification

- $O(n)$ time - simple and fast
- Several nice properties. Fréchet error

Imai-Iri 1988

- Gives an optimal solution but slow. Hausdorff/ Fréchet error

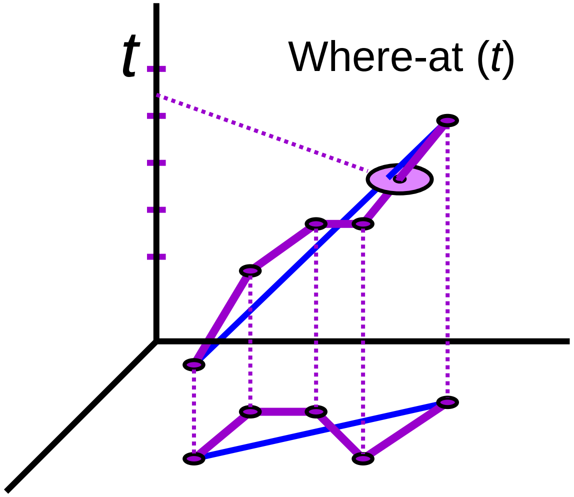
Agarwal, Har-Peled, Mustafa and Wang 2005

- Fast and easy to implement. Gives a worst-case performance guarantee. Fréchet error

More Algorithms

Simplifying trajectories

- Time has to be incorporated, possibly as third dimension
- Gudmundsson et al. 2009
- based on Ramer–Douglas–Peucker
- Exercise 3: Imai-Iri

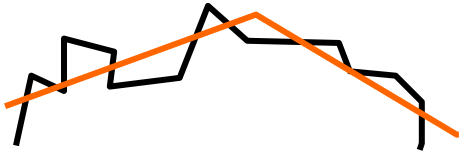


Streaming setting

- Abam, de Berg, Hachenberger, Zarei 2007

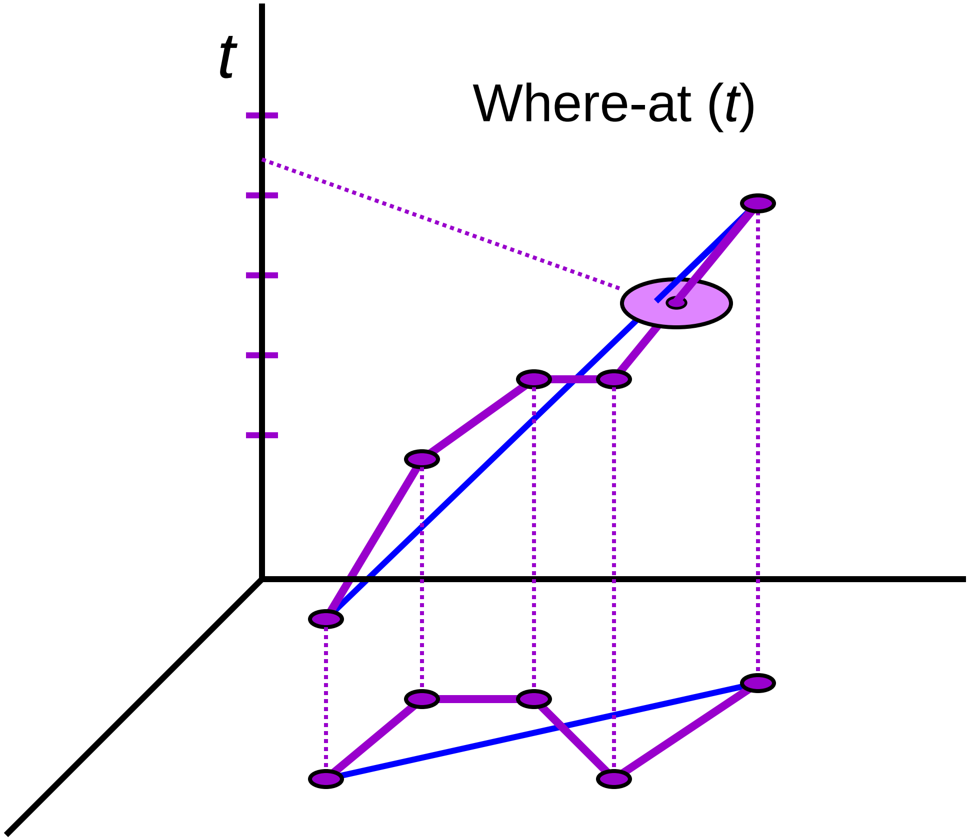
Non-vertex constrained

- Guibas, Hershberger, Mitchell, Snoeyink, 1993



… and more, e.g. topological or geometric constraints

Where-at (*t*)



References

- A. Driemel, S. Har-Peled and C. Wenk. Approximating the Fréchet Distance for Realistic Curves in Near Linear Time. *Discrete & Computational Geometry* , 2012.
- W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *Proceedings of the 3rd International Symposium on Algorithms and Computation*, 1992.
- D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 1973.
- P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplication in two and three dimensions. *Algorithmica*, 2005.