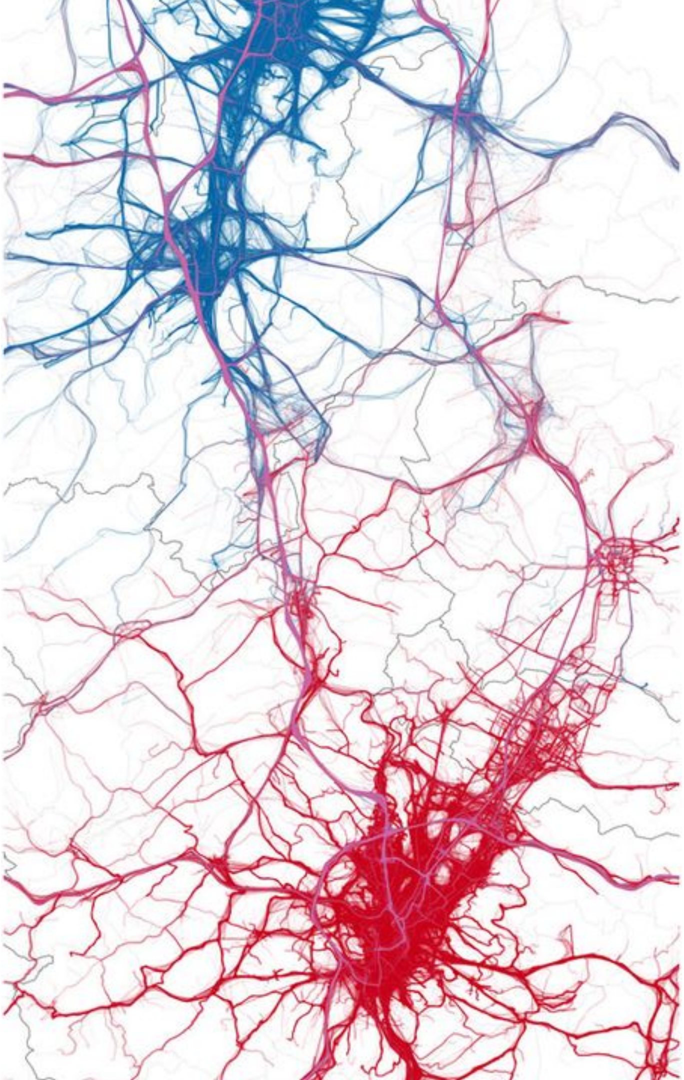


IN SUPREMÆ DIGNITATIS
1343

Preprocessing Mobility Data

Consiglio Nazionale
delle Ricerche

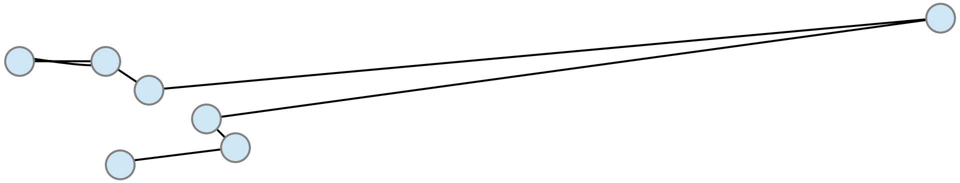
Content of this lesson

- Preprocessing trajectories – Part I
    - trajectory filtering
    - point map matching
    - route reconstruction
    - trajectory compression

Trajectory filtering

- Data points are sometimes affected by errors
- Errors can have huge effects on results

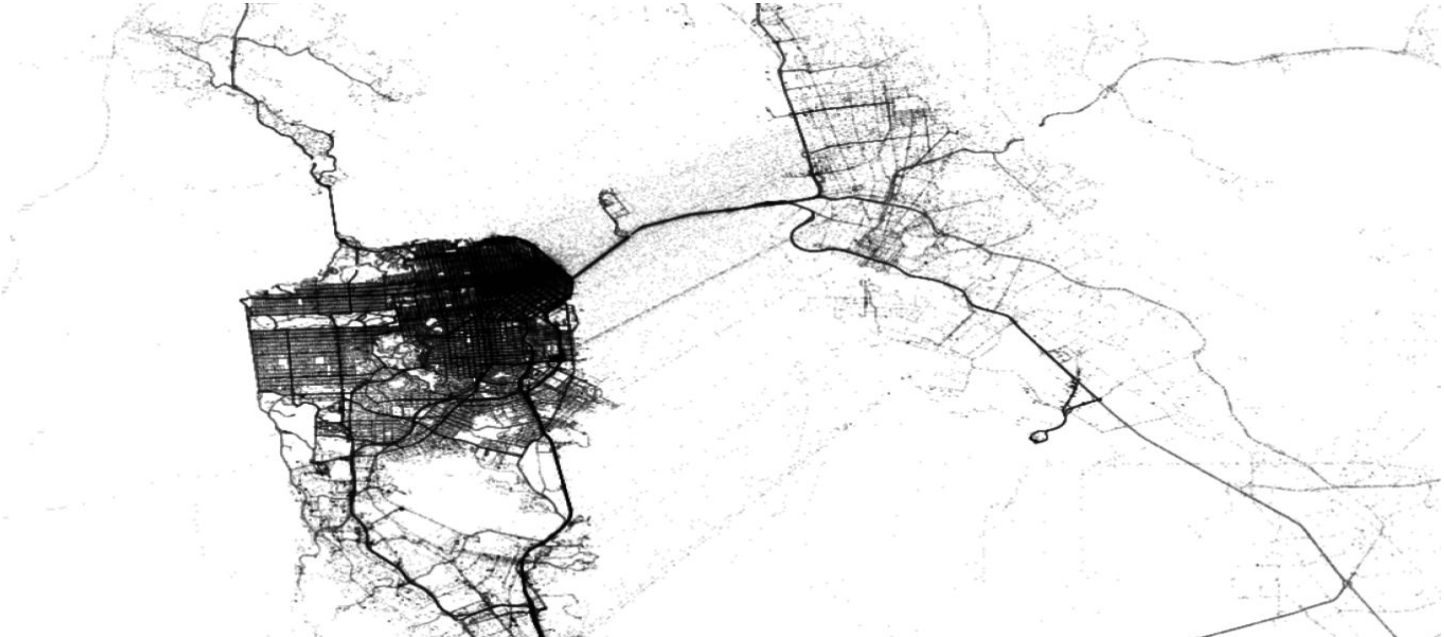
What is the real length of this trip?



- Two families of approaches:
  - Context-based filtering
  - Movement-based filtering

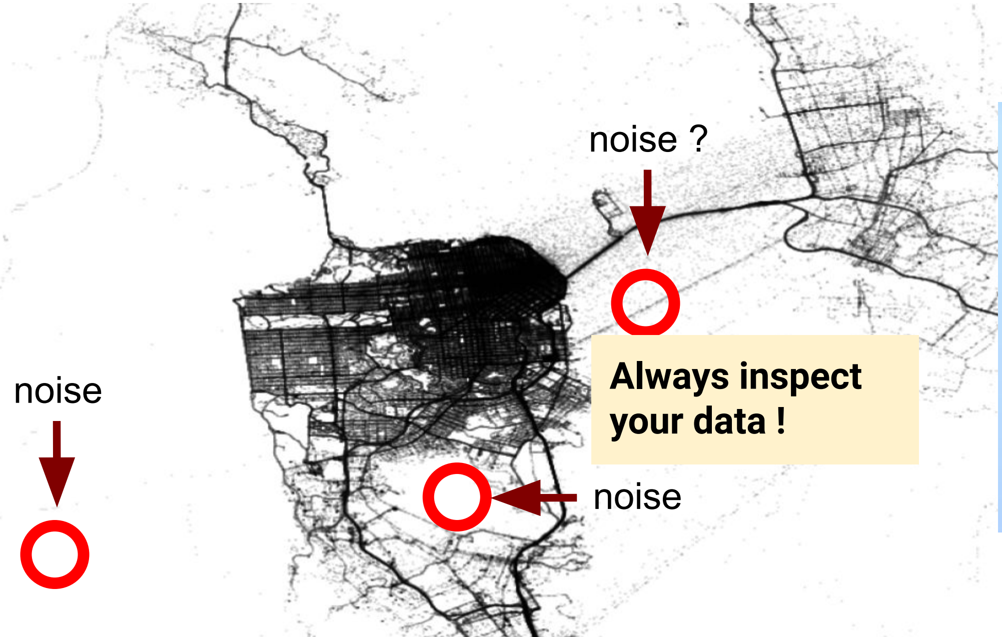
Context-based filtering

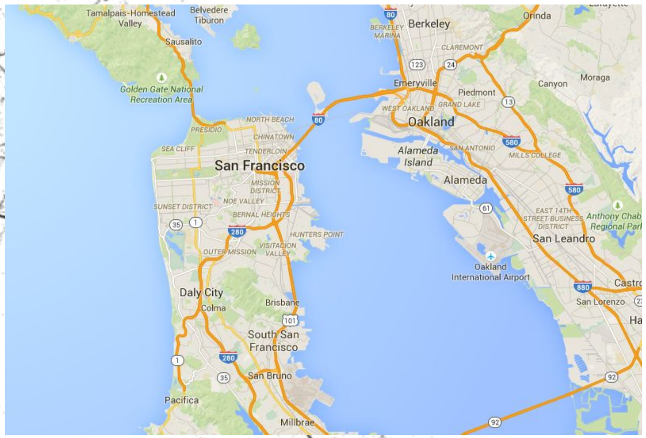
- Single points might contain errors of various kinds



Context-based filtering

- Single points might contain errors of various kinds
- **Map-based detection**: cars on the water or out of roads are noise
    - Caution: do you trust 100% your map?



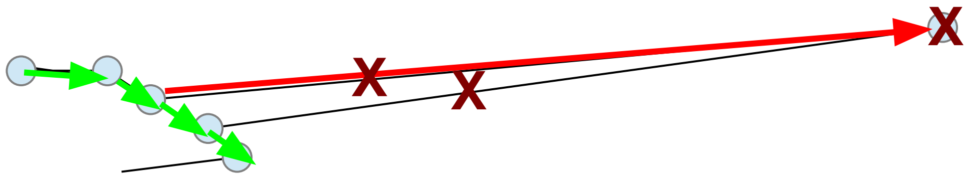


Movement-based filtering

- No context is used, just the geometry / dynamics of movement

- **Speed-based** noise filtering approach:
  - The first point of the trajectory is set as valid
  - Scan all remaining points “p” of the trajectory (time order)
    - Compute “v” = average straight-line speed between point “p” and the previous valid one
    - If “v” is huge (e.g. larger than 400 km/h)
      => remove “p” from trajectory (“p” will not be used next to estimate speeds…)

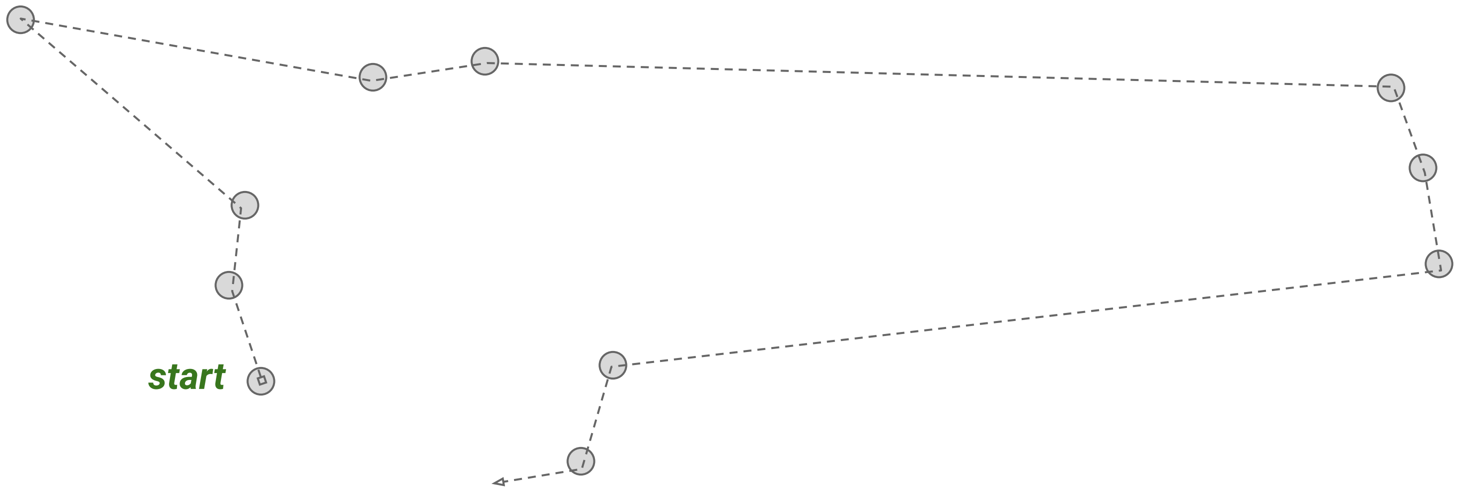
      else
      => set “p” as valid



Movement-based filtering

Exercise

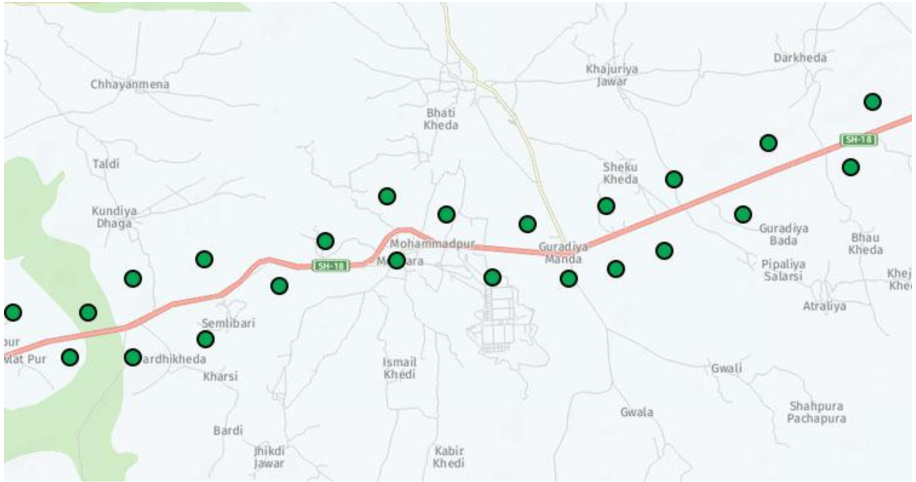
- What happens in this situation? (Multiple noisy points)



Point map matching

- Points can be aligned to the road network
  - Objective 1: improve accuracy of position
  - Objective 2: remove extreme cases (ref. filtering)
  - Objective 3: translate trajectories to sequences of road IDs

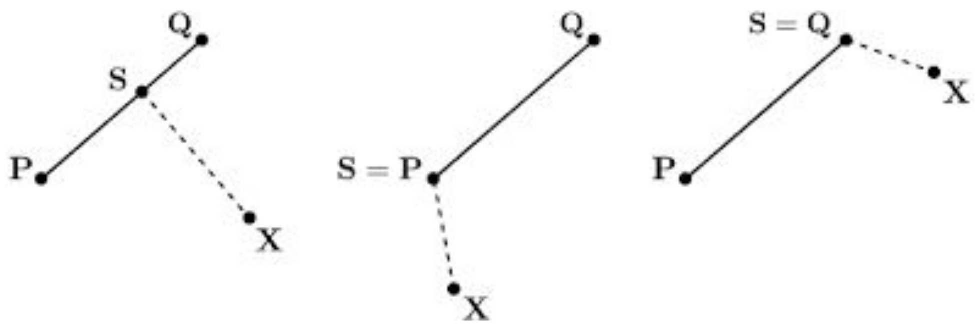
- Idea: project the point to the close location in the network
  - Usually there is a maximum threshold
  - Points farther than the threshold from any road are removed as noise



Point map matching

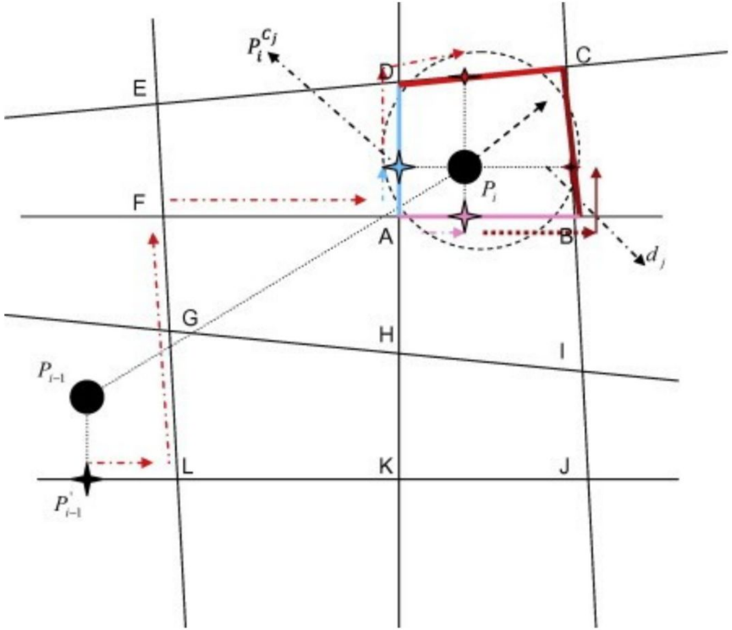
- Point projection
  - Requires to compare each point to each road segment

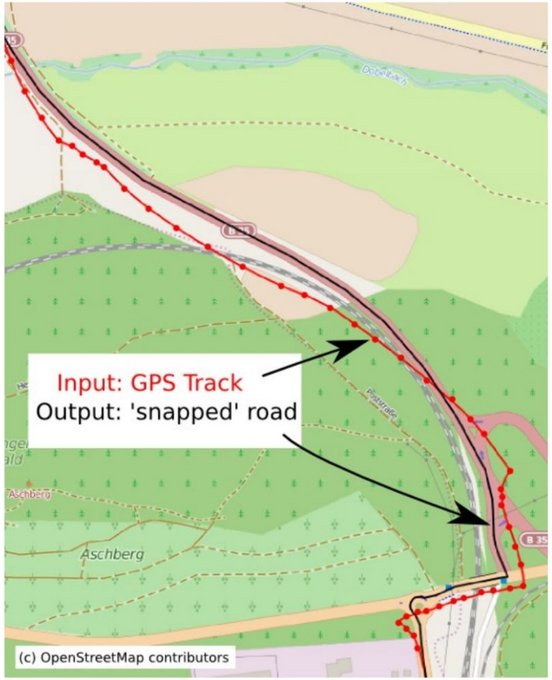
- Refresher on point-to-segment distance computation



Point map matching

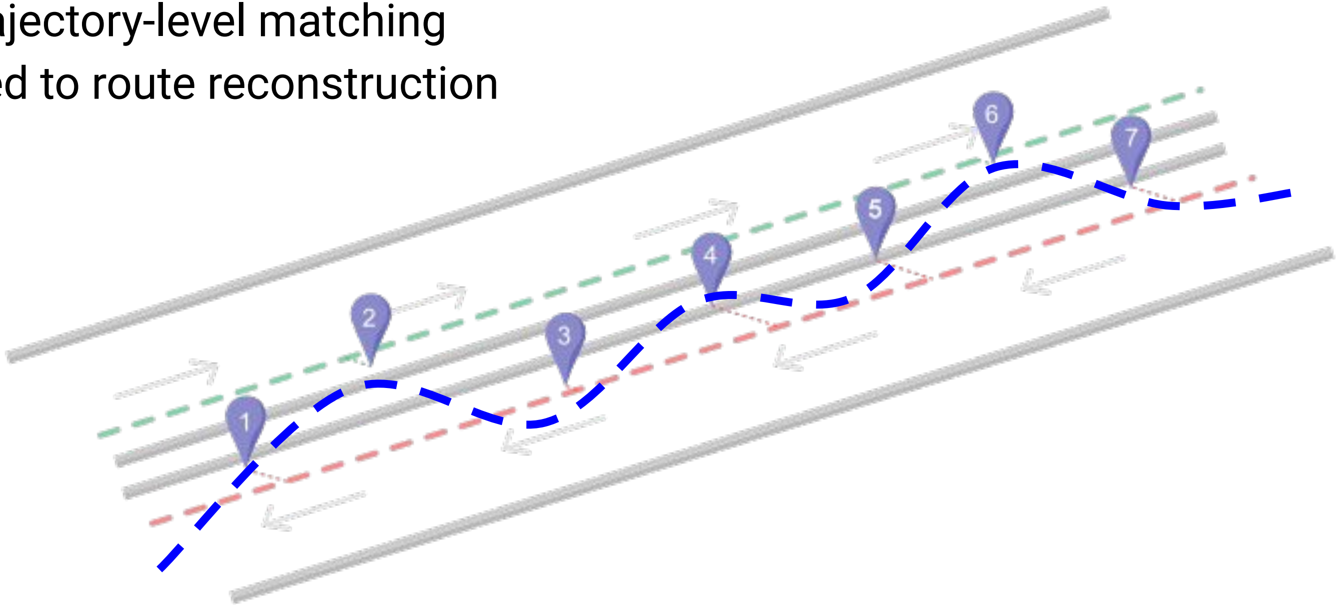
- In some contexts there can be multiple choices





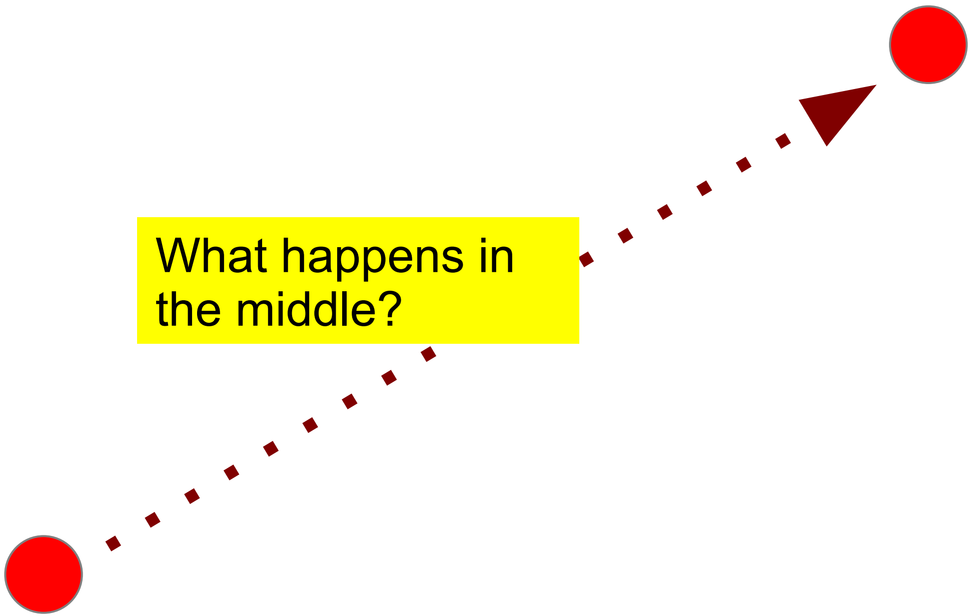
Point map matching

- Matching points separately can lead to inconsistent results
  - Mainly road-dense areas with position accuracy comparable to road separation
- Need a trajectory-level matching
  - Linked to route reconstruction



Route reconstruction

- Sometimes the space/time gap between consecutive points is significant

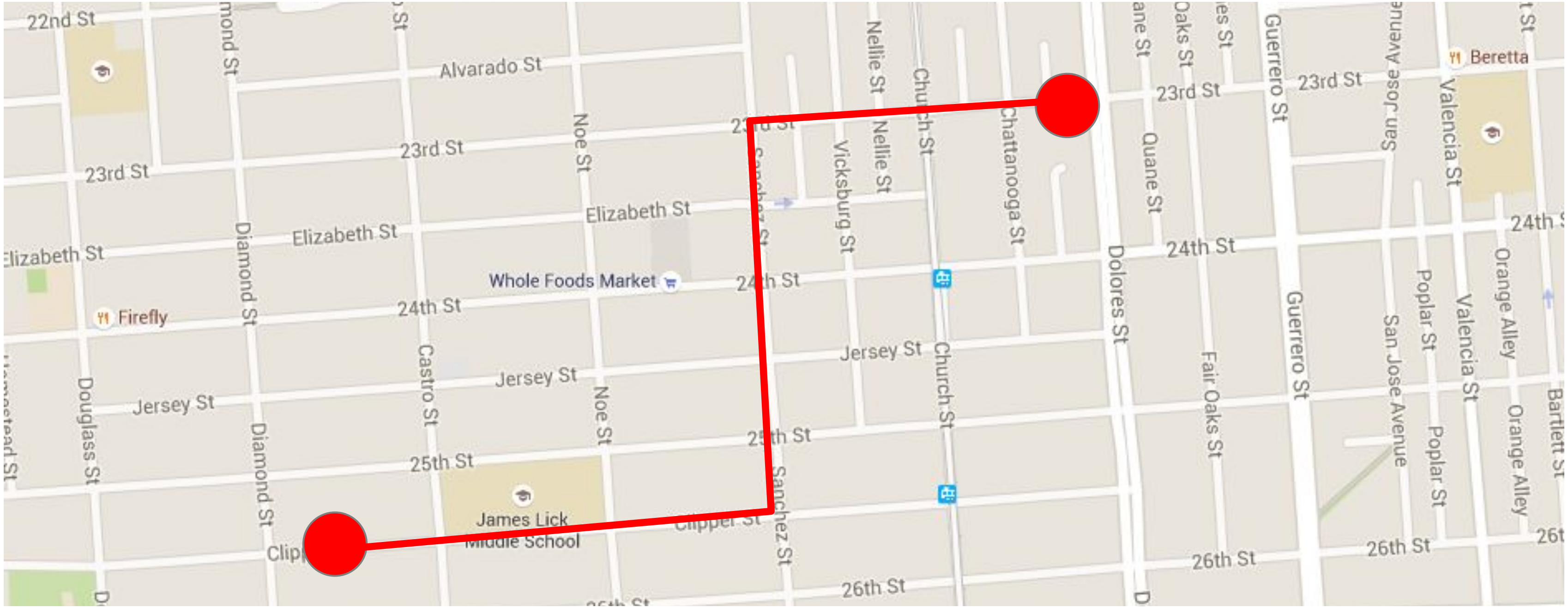


Route reconstruction

- Typical solutions:
  - Free movement => straight line, uniform speed

Route reconstruction

- Typical solutions:
  - Constrained movement => shortest path



Route reconstruction

Shortest paths can be replaced by alternative “optimal paths”

- Based on a notion of path cost
- Typical ones: path **length**, path **duration** (requires to know typical traversal times of roads)
- Alternative ones: fuel consumption, EV battery consumption, CO2 emissions, mixed costs

Algorithms applied are standard graph path optimization methods:

- Dijkstra's algorithm → efficient, requires that costs are non-negative
- Bellman-Ford algorithm → less efficient, can work with negative weights (but no cycles)

See *method* parameter
of *shortest_path*
function of NetworkX

Refresher: Dijkstra’s minimum cost algorithm

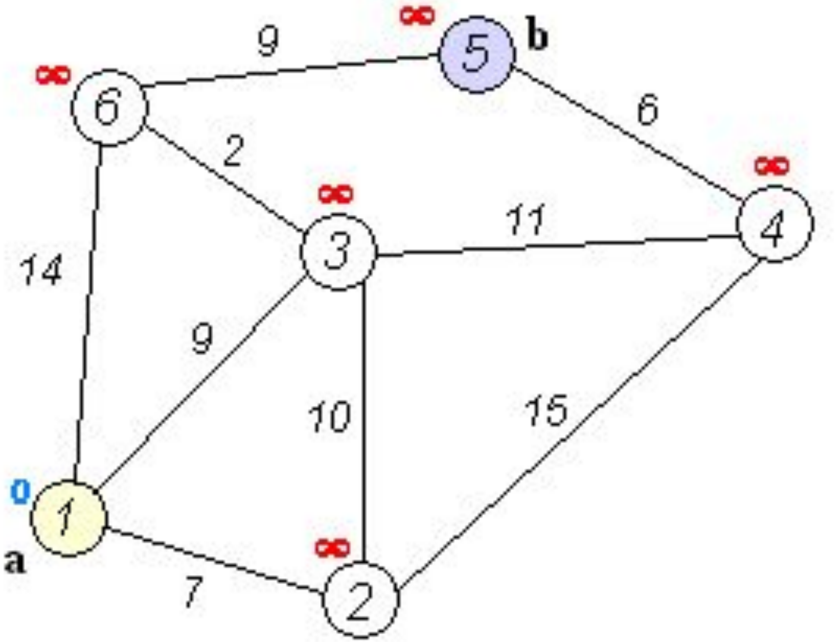
```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```



Refresher: Dijkstra's minimum cost algorithm

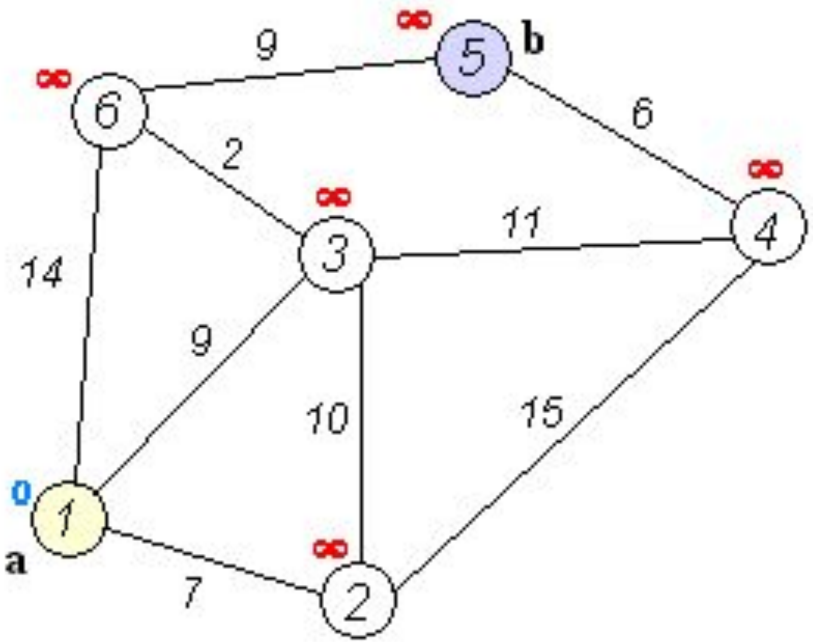
```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```



Refresher: Dijkstra’s minimum cost algorithm

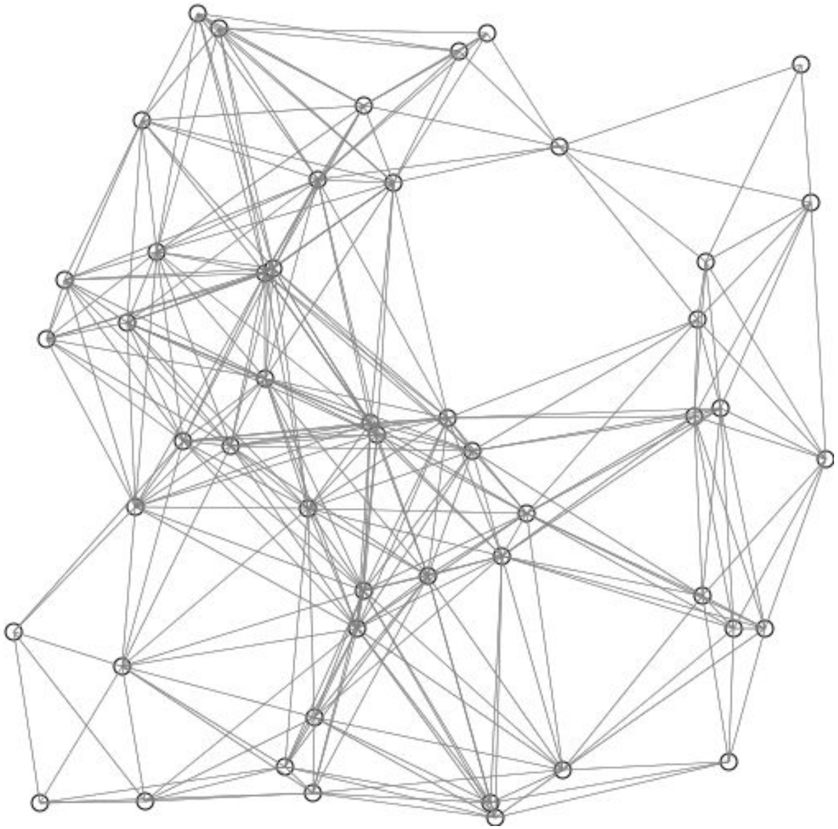
```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```



Trajectory Map Matching

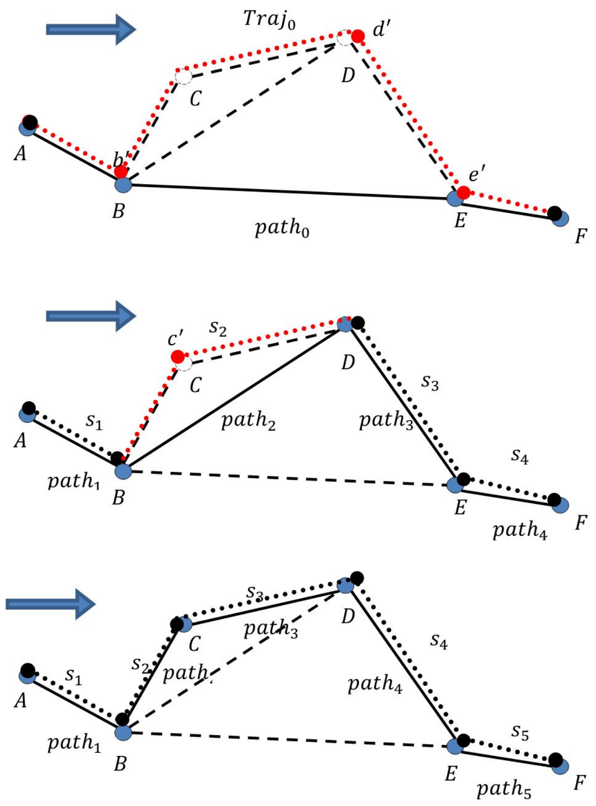
- Assigns points to road segments
- Reconstructs the movement between consecutive points
- Ensures coherence of the overall process

- Two sample approaches:
  - Based on shortest path
  - Based on probabilities

Shortest path-based Map Matching

Used by [MappyMatch]

- Similar ideas as trajectory simplification
  - Match first and last point
  - Compute shortest path on the network
  - Find farthest point from shortest path
  - If distance > threshold ⇒
    - split into two parts
    - run recursively the process on both

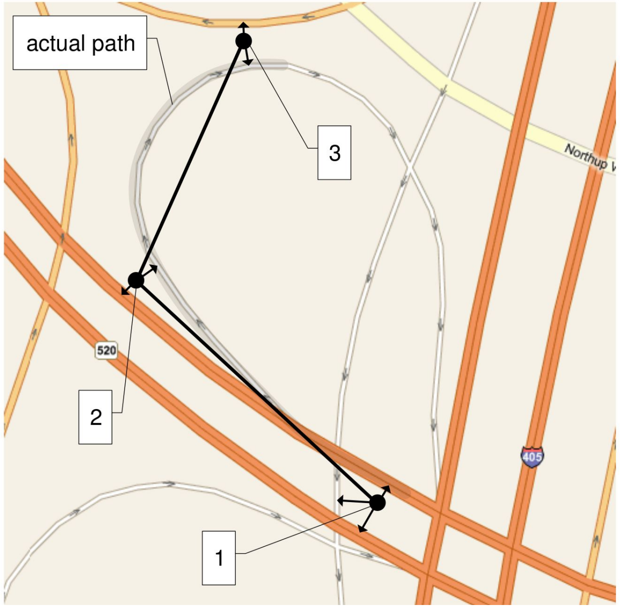


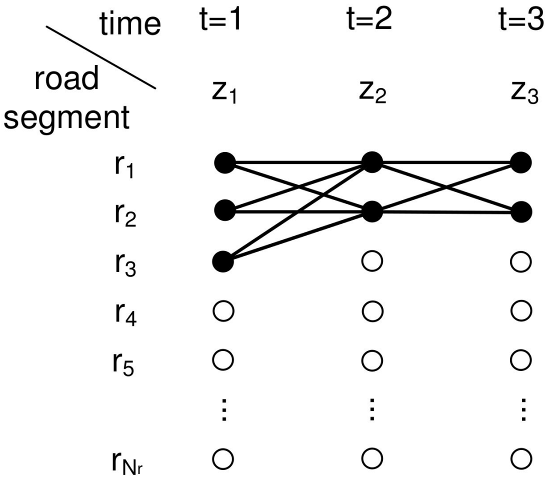
Reference: [Zhu, honda & Gonder. A Trajectory Segmentation Map Matching Approach for Large-Scale, High-Resolution GPS Data. TRB 2017.]

Probability-based Map Matching

Used by pyTrack

- Consider possible point-to-road assignments, with probabilities
- Compute most likely path that visits all points in the correct sequence

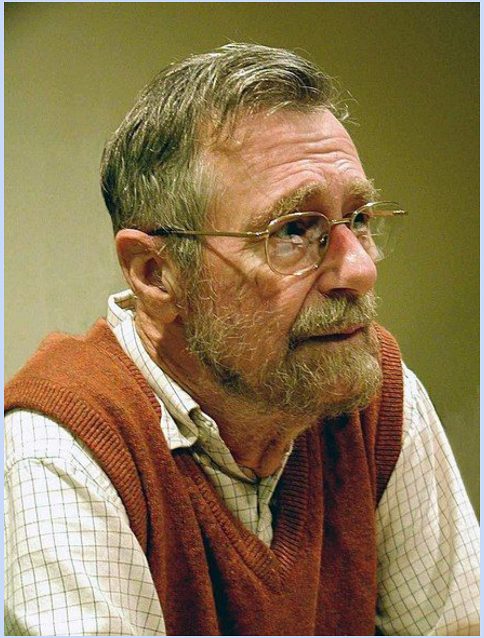




Reference: Newson & Krumm. Hidden Markov Map Matching Through Noise and Sparseness. ACM GIS'09.

INTERVALLO

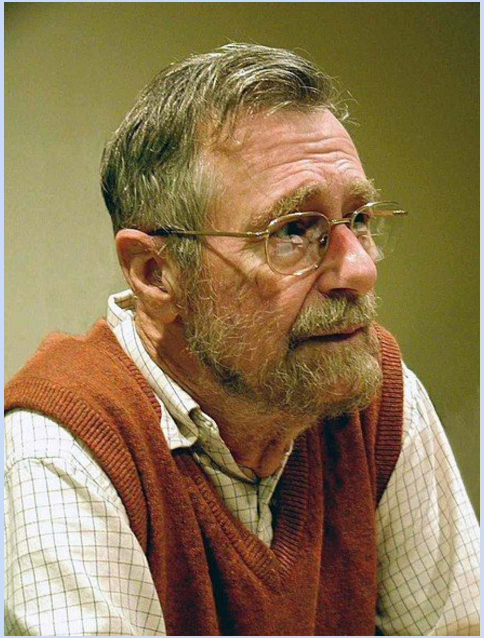
Who's Dijkstra



- 1930 - 2002
- Dutch computer scientist, programmer, software engineer, systems scientist, and science essayist
- 1972 Turing Award for “fundamental contributions to developing programming languages”

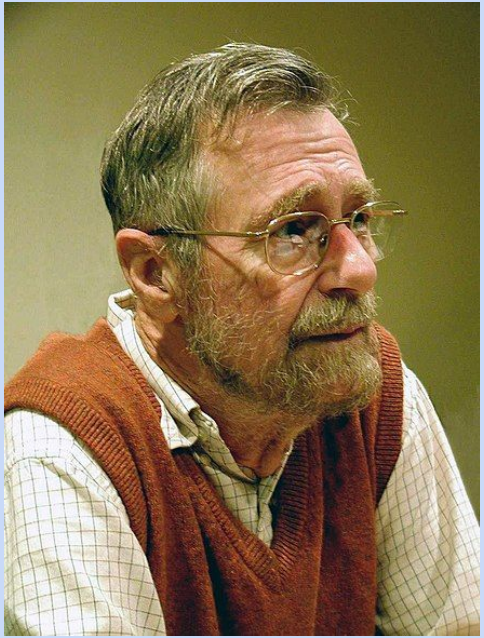
INTERVALLO

Dijkstra is famous for…

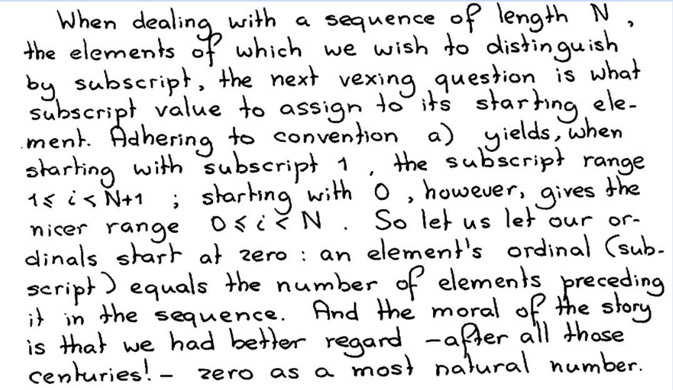


- Dijkstra’s algorithm, of course
- Contributions to “self-stabilization of program computation”
  - Won him the “ACM PODC Influential Paper Award”, later renamed “Dijkstra Prize”
- Hundreds of papers on computational and science philosophy issues

Dijkstra is famous for...



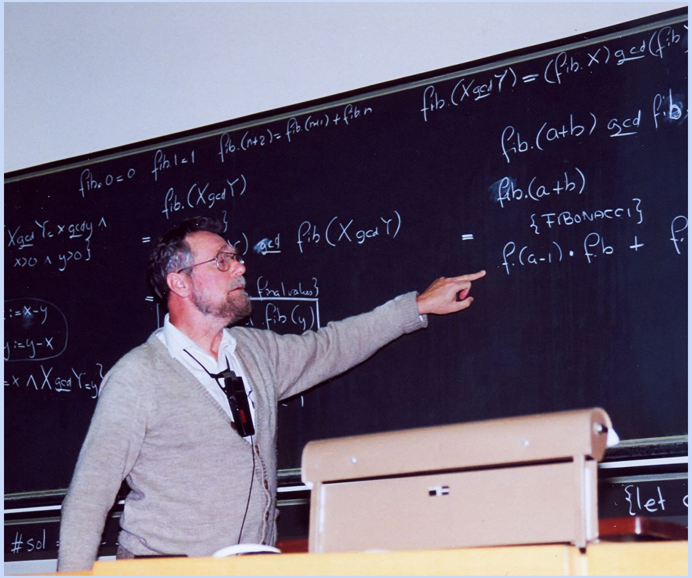
- His habit of writing everything with paper & fountain pen
- Hundreds of papers, many unpublished
  - E. W. Dijkstra Archive
- Counting should start from 0, not 1…



When dealing with a sequence of length N, the elements of which we wish to distinguish by subscript, the next vexing question is what subscript value to assign to its starting element. Adhering to convention a) yields, when starting with subscript 1, the subscript range $1 \leq i < N+1$; starting with 0, however, gives the nicer range $0 \leq i < N$. So let us let our ordinals start at zero: an element's ordinal (subscript) equals the number of elements preceding it in the sequence. And the moral of the story is that we had better regard —after all those centuries!— zero as a most natural number.

INTERVALLO

Dijkstra the teacher



- Chalk & blackboard, no projectors
- No textbooks
- Improvisation & long pauses
- No references in papers

  "For the absence of a bibliography I offer neither explanation nor apology."
- Long exams
  - Each student was examined in Dijkstra's office or home, and an exam lasted several hours

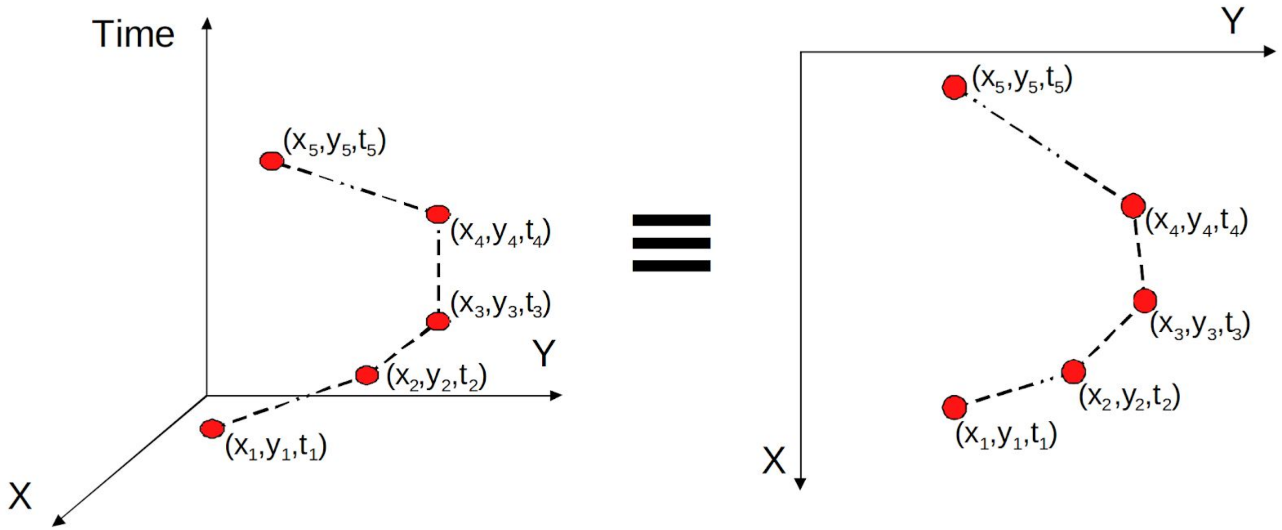
Trajectory compression / simplification

- Many algorithms for trajectories are expensive
  - Their complexity depends on the number of points
  - Sometimes trajectories have more points than needed

- Objective of compression / simplification
  - Reduce the number of points…
  - … without affecting the quality of results

Trajectory data

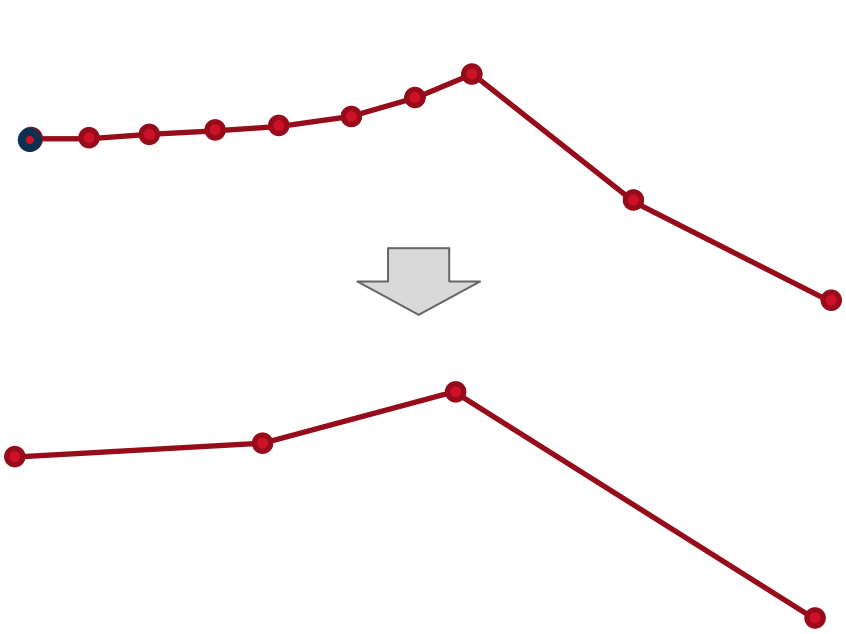
- A trajectory is a temporal sequence of time-stamped locations
- Most methods focus on the spatial component



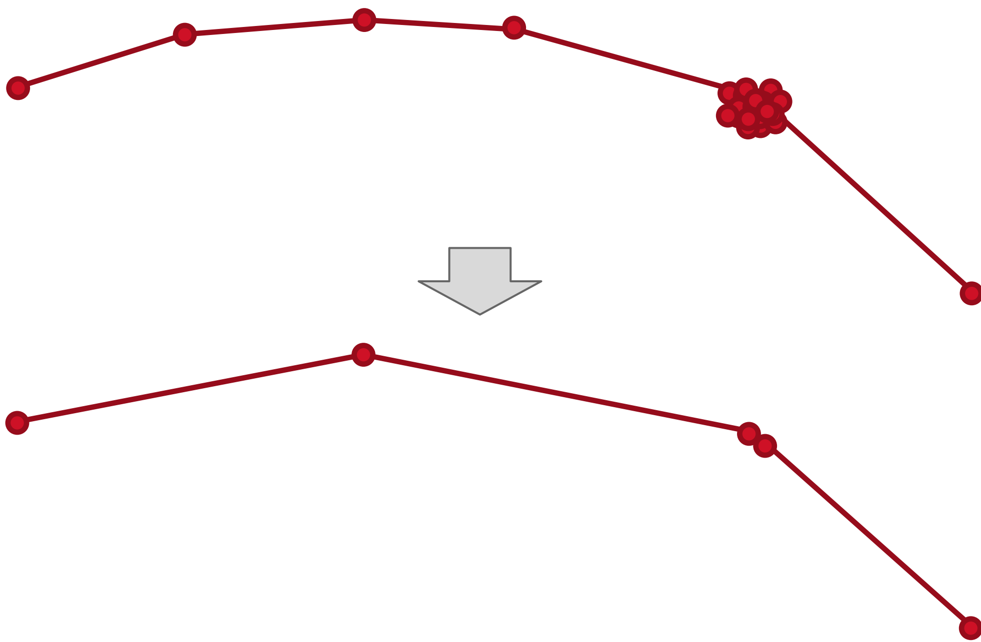
Trajectory compression / simplification

- Typical cases where points might be removed

Straight line movement



Negligible movement



Compression/simplification methods

Some standard methods for simplifying polygonal curves:

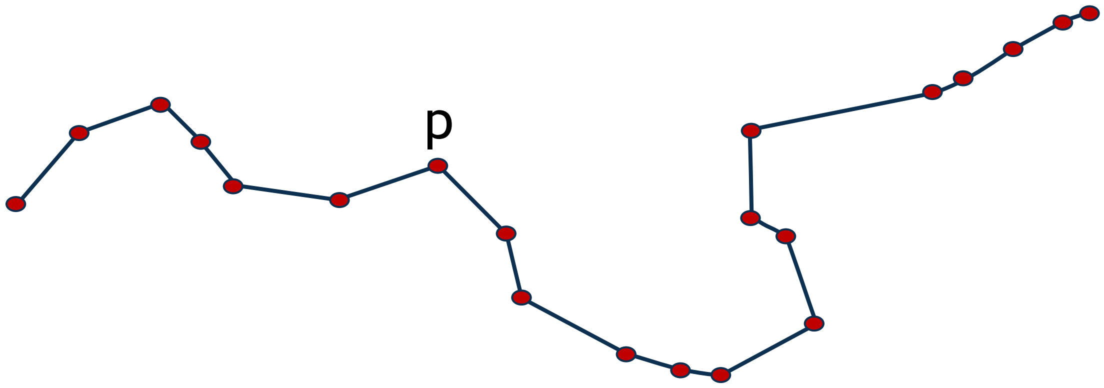
- Ramer–Douglas-Peucker, 1973
- Driemel–HarPeled–Wenk, 2010
- Imai–Iri, 1988

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

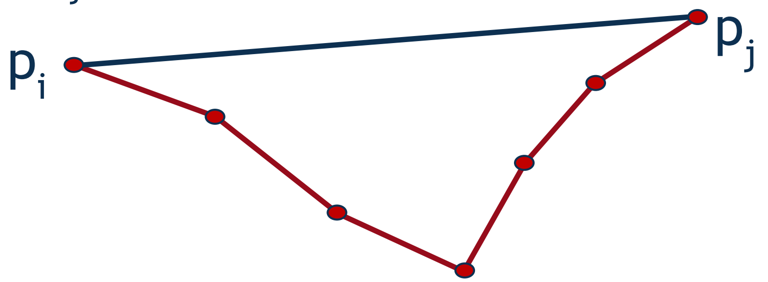
Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

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_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

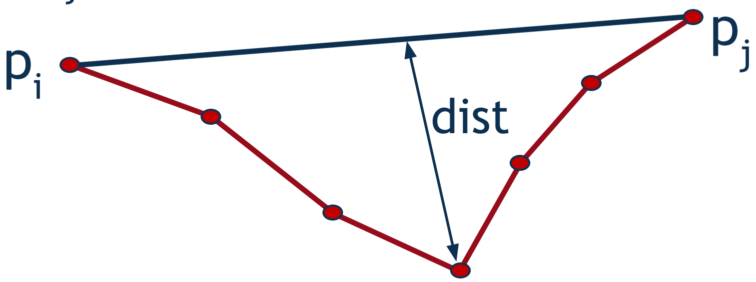
Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

Initially i=1 and j=n

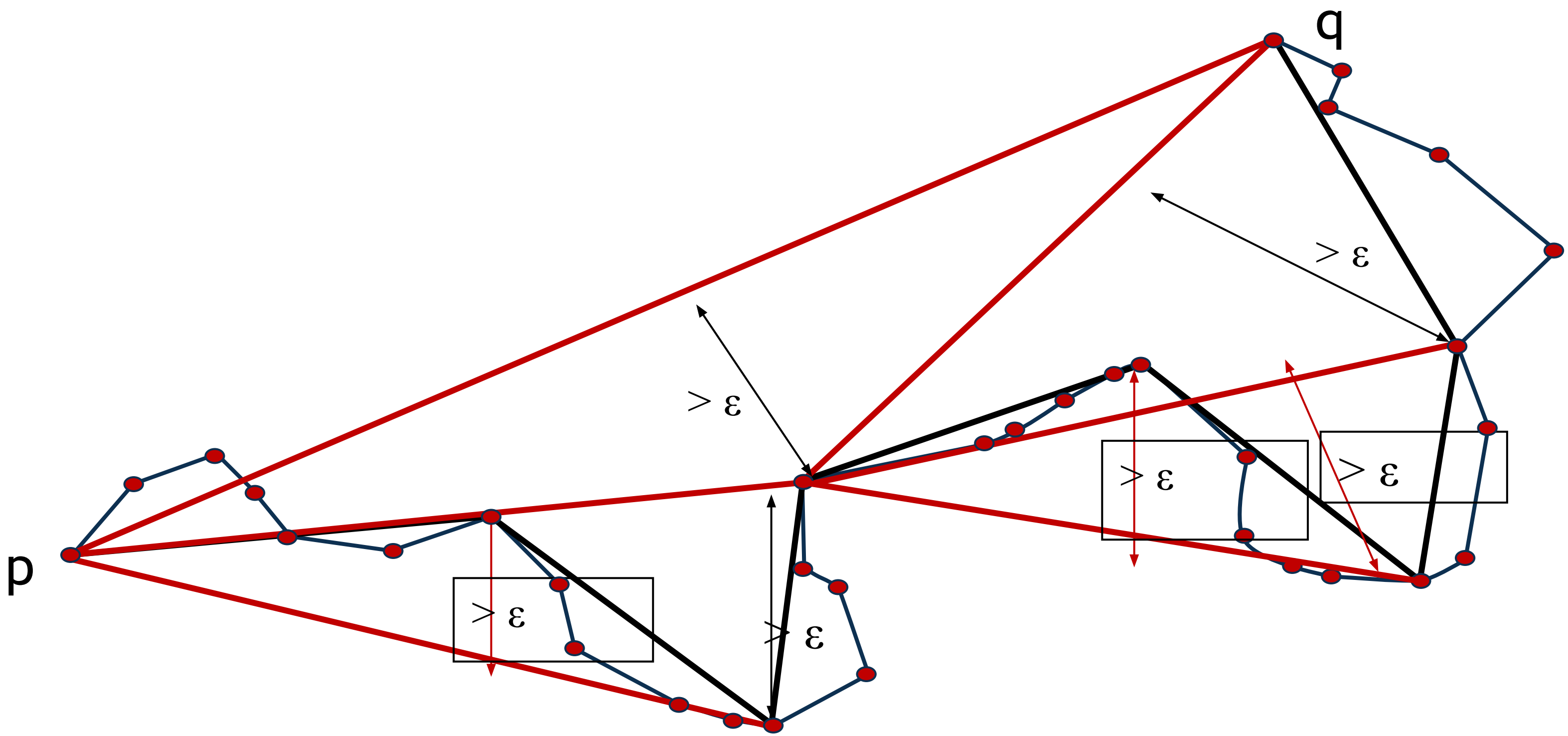
Algorithm DP(P,i,j)

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.

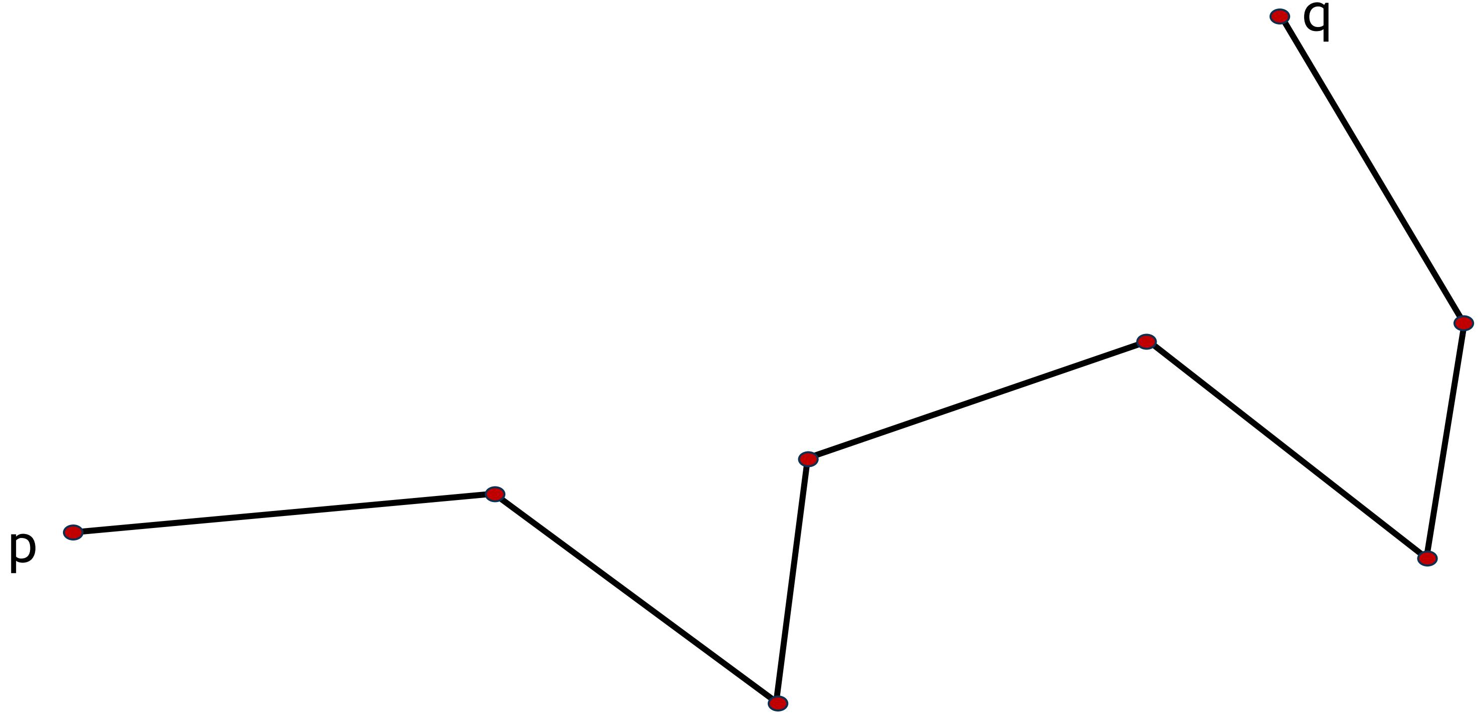
if dist > ε then
DP(P, v_i , v_f)
DP(P, v_f , v_j)
else
Output(v_iv_j)



Ramer-Douglas-Peucker



Ramer-Douglas-Peucker



Ramer-Douglas-Peucker

Time complexity?

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

Worst-case recursion?

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

Time complexity
$T(n) = O(n) + T(n-1) = O(n^2)$

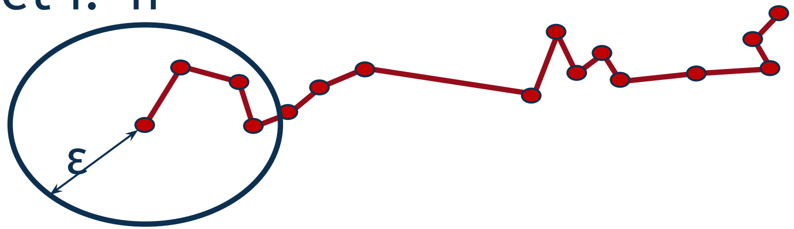
Algorithm DP(P,i,j)
    Find the vertex v_f farthest from p_ip_j.
    dist := the distance between v_f and p_ip_j.

    if dist > ε then
        DP(P, v_i , v_f)
        DP(P, v_f , v_j)
    else
        Output(v_iv_j)

Driemel et al.

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

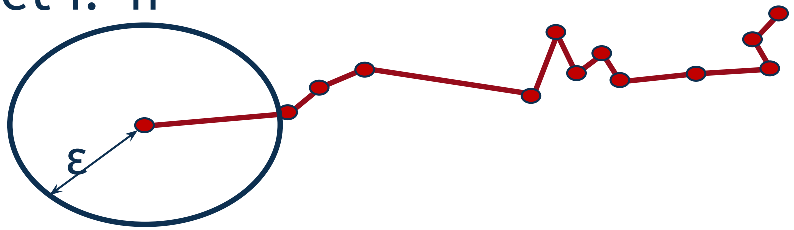
$P' := \langle p_1 \rangle$
$i := 1$
while $i < n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q - p_i| > \varepsilon$
    if no such vertex then set $i := n$
    add p_i to P'
end
return P'



Driemel et al.

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

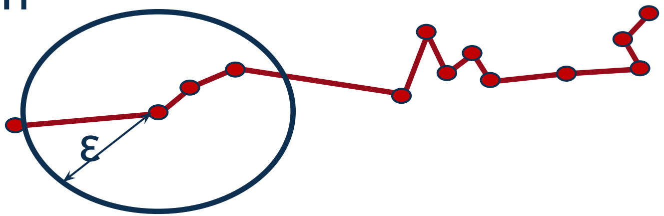
$P' := \langle p_1 \rangle$
$i := 1$
while $i<n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'



Driemel et al.

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

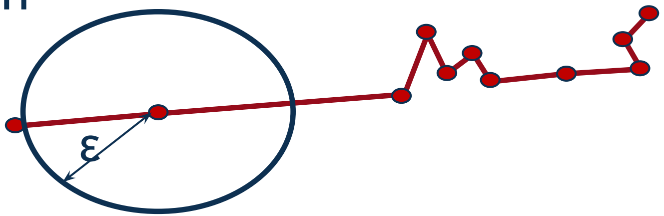
$P' := \langle p_1 \rangle$
$i := 1$
while $i < n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q - p_i| > \varepsilon$
    if no such vertex then set $i := n$
    add p_i to P'
end
return P'



Driemel et al.

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

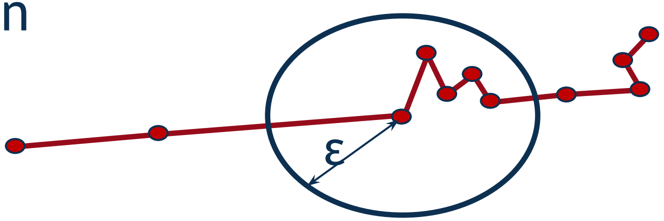
$P' := \langle p_1 \rangle$
$i:=1$
while $i<n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'



Driemel et al.

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

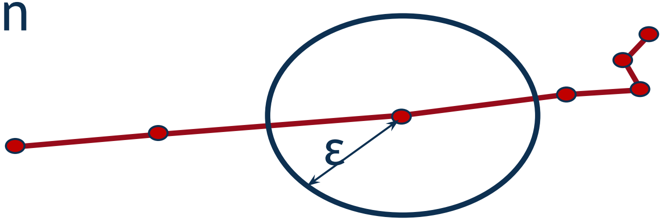
$P' := \langle p_1 \rangle$
$i := 1$
while $i<n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'



Driemel et al.

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

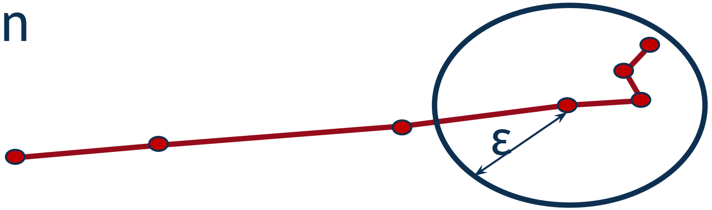
$P' := \langle p_1 \rangle$
$i := 1$
while $i<n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q-p_i| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'



Driemel et al.

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

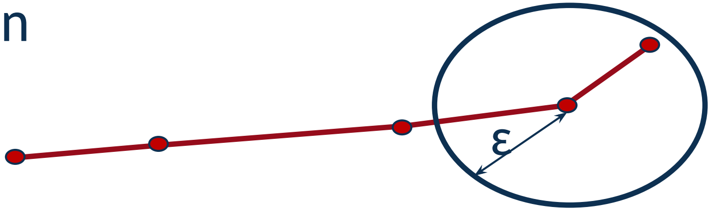
$P' := \langle p_1 \rangle$
$i := 1$
while $i < n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n \rangle$ s.t. $|q - p_i| > \varepsilon$
    if no such vertex then set $i := n$
    add p_i to P'
end
return P'



Driemel et al.

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

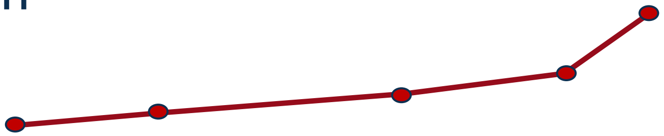
$P' := \langle p_1 \rangle$
$i:=1$
while $i<n$ do
    $q := p_i$
    $p_i :=$ first vertex p_i in $\langle q,\ldots,p_n\rangle$ s.t. $|q-p_i| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'



Driemel et al.

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



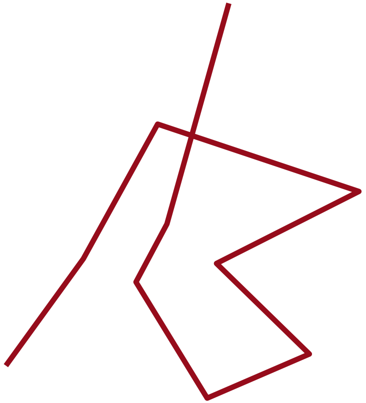
Summary: Driemel et al.

Simple simplification: can be computed in $O(n)$ time

Property 1:
All edges (except the last one) have length at least ε.

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

(δ_F = Fréchet distance. We will discuss it later…)



Imai-Iri

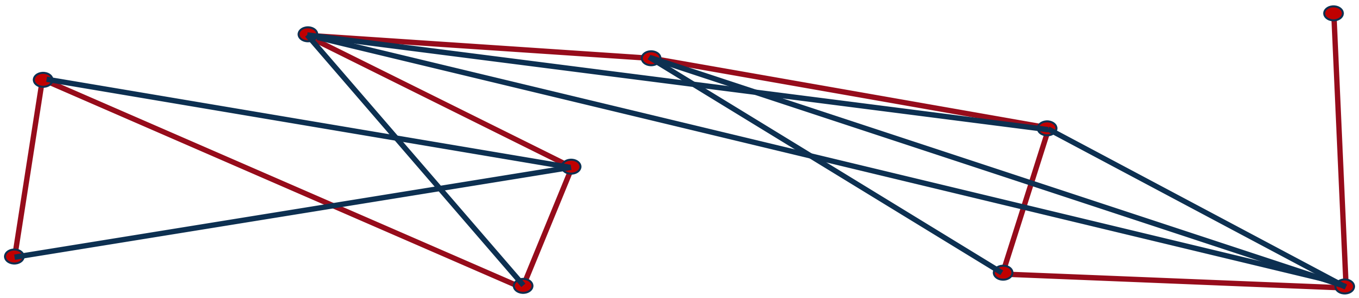
Both previous algorithms are simple and fast but do not give a bound on the complexity of the simplification!

Imai-Iri 1988 gave an algorithm that produces a ε-simplification with the minimum number of links.

Imai-Iri

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

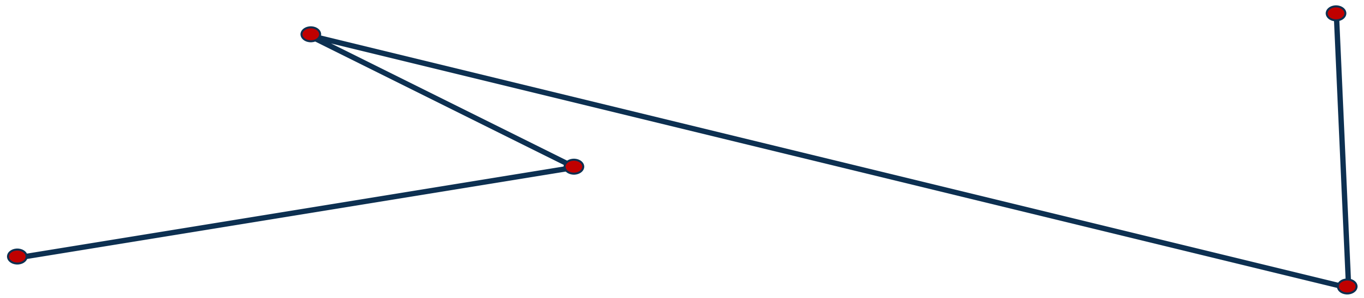
1. Build a graph G containing all valid shortcuts.
2. Find a minimum link path from p_1 to p_n in G



Imai-Iri

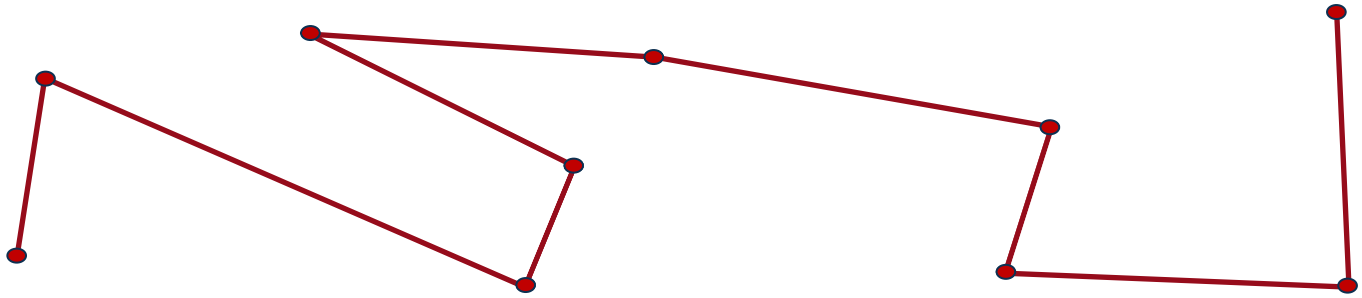
Input polygonal path $P = \langle p_1,\ldots,p_n \rangle$ and threshold ε

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



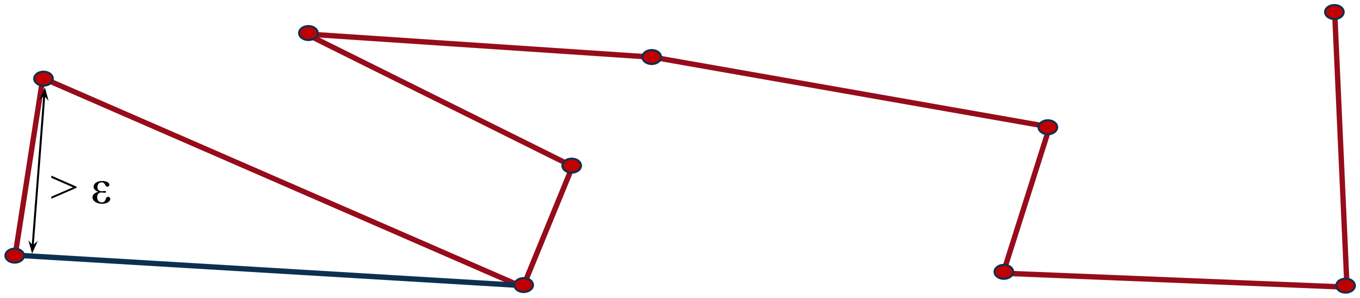
Imai-Iri

Find all possible valid shortcuts



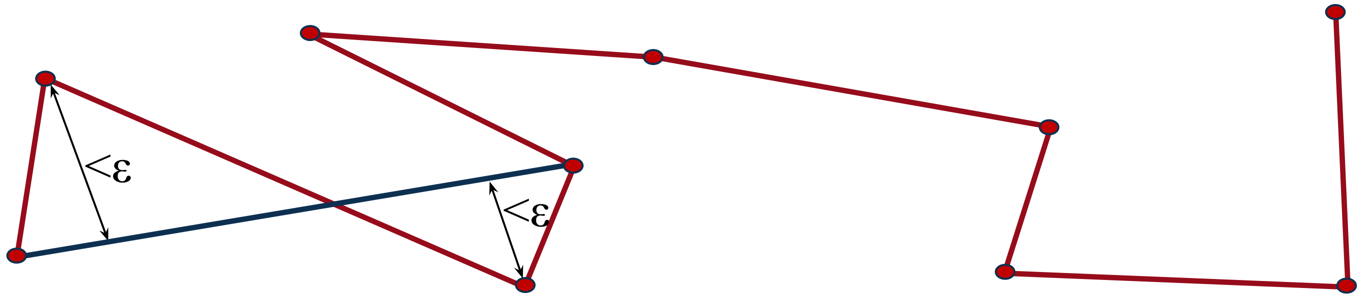
Imai-Iri

Find all possible valid shortcuts



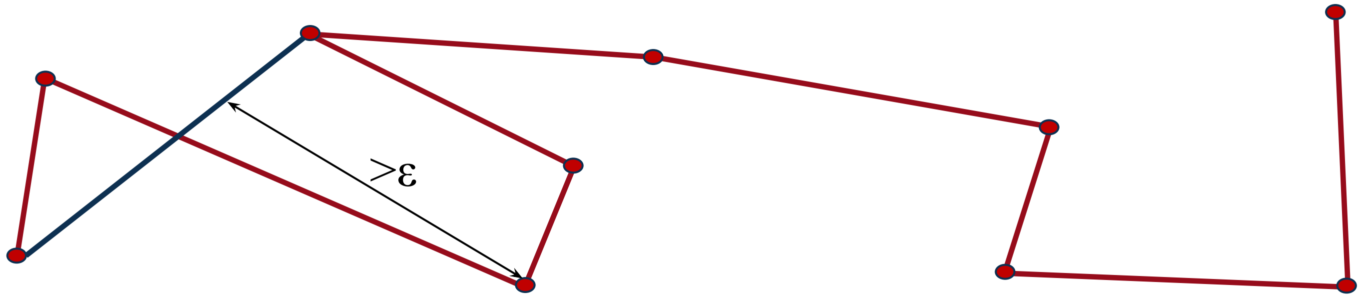
Imai-Iri

Find all possible valid shortcuts



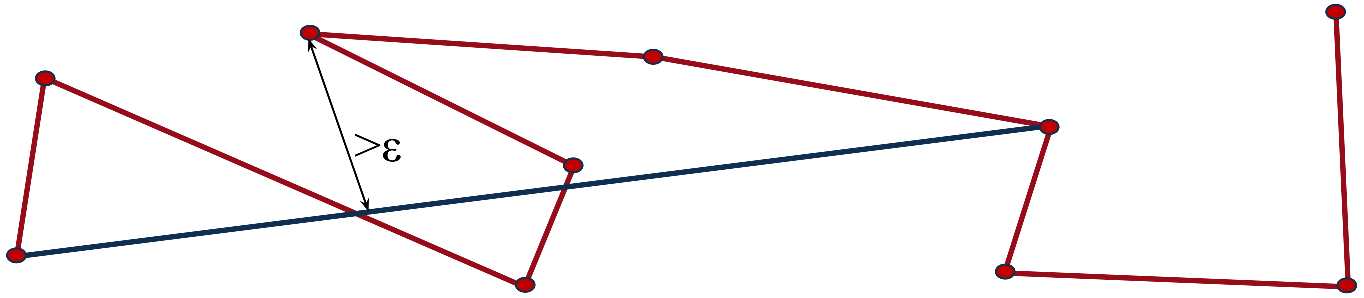
Imai-Iri

Find all possible valid shortcuts



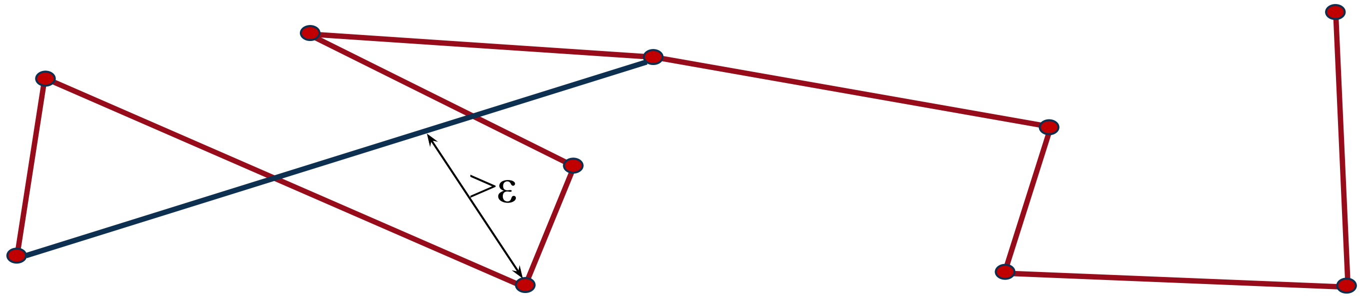
Imai-Iri

Find all possible valid shortcuts



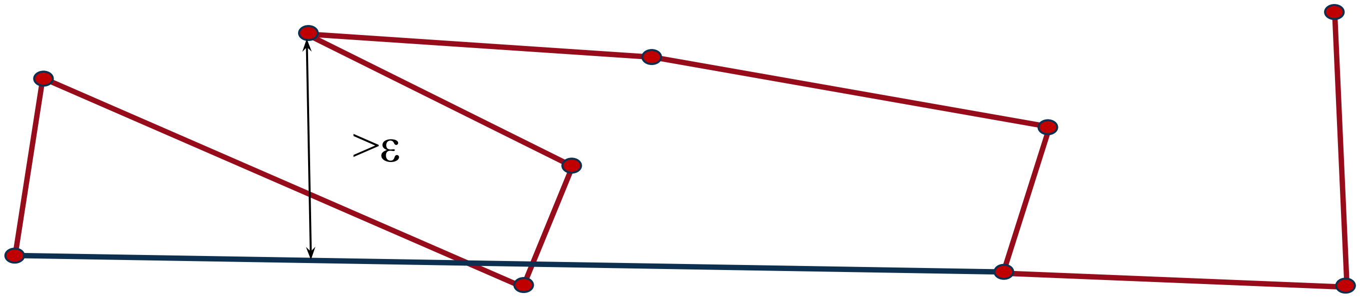
Imai-Iri

Find all possible valid shortcuts



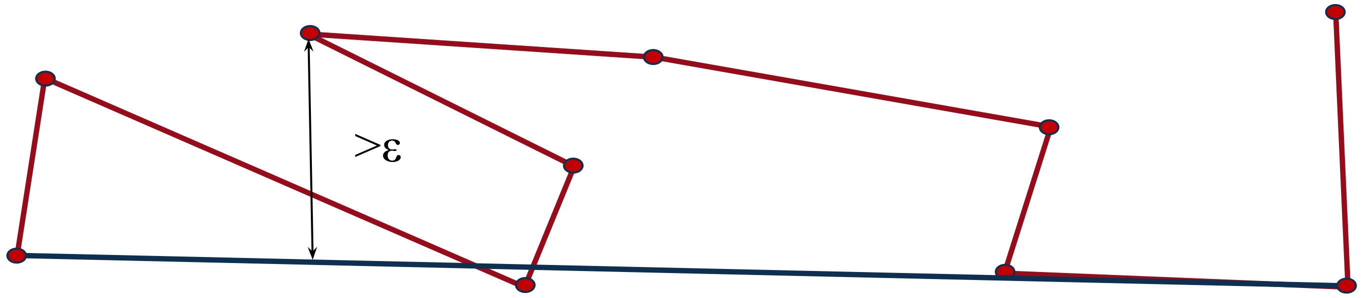
Imai-Iri

Find all possible valid shortcuts



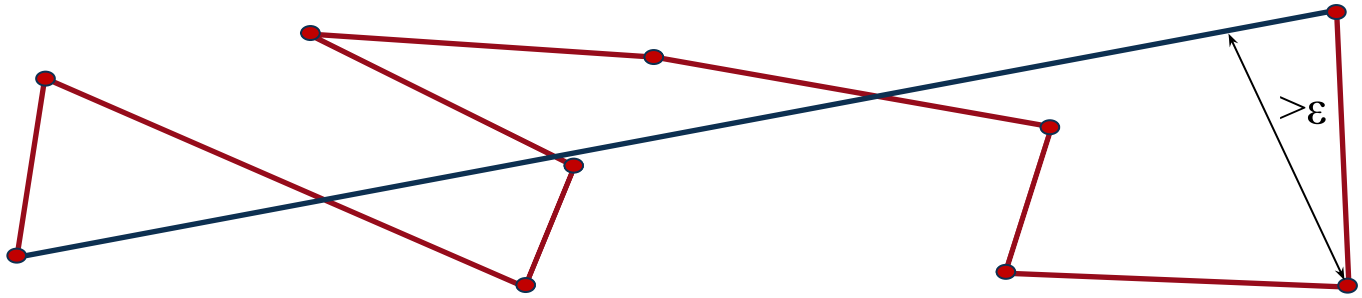
Imai-Iri

Find all possible valid shortcuts



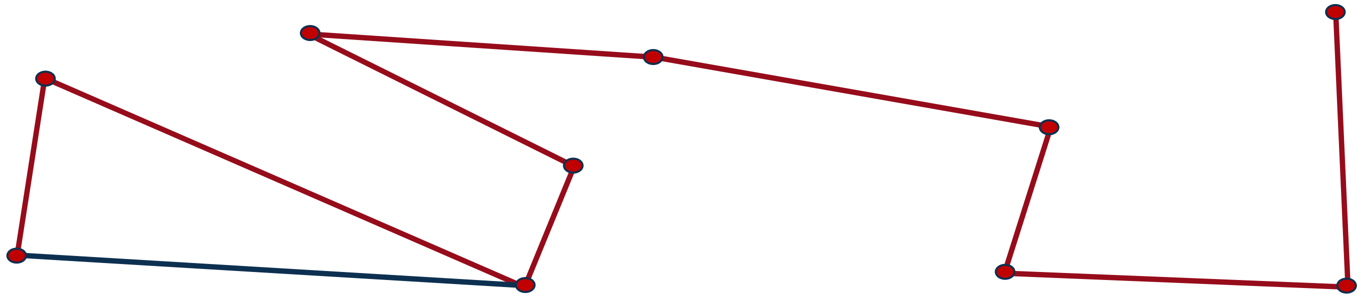
Imai-Iri

Find all possible valid shortcuts



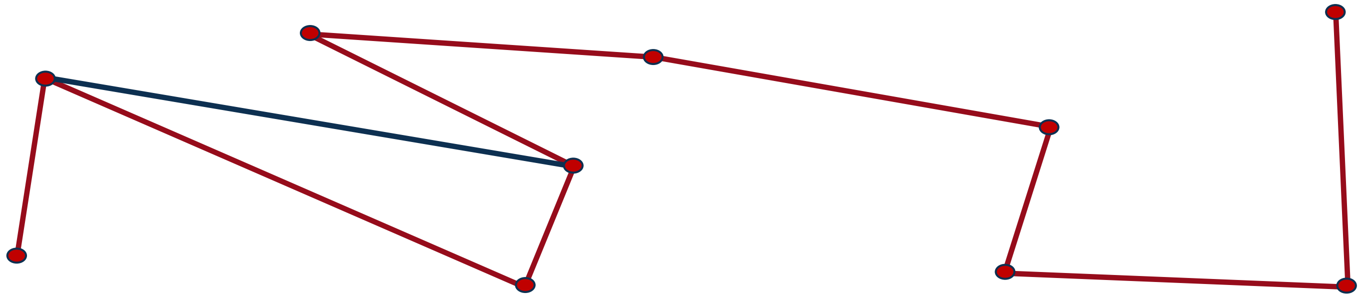
Imai-Iri

Find all possible valid shortcuts



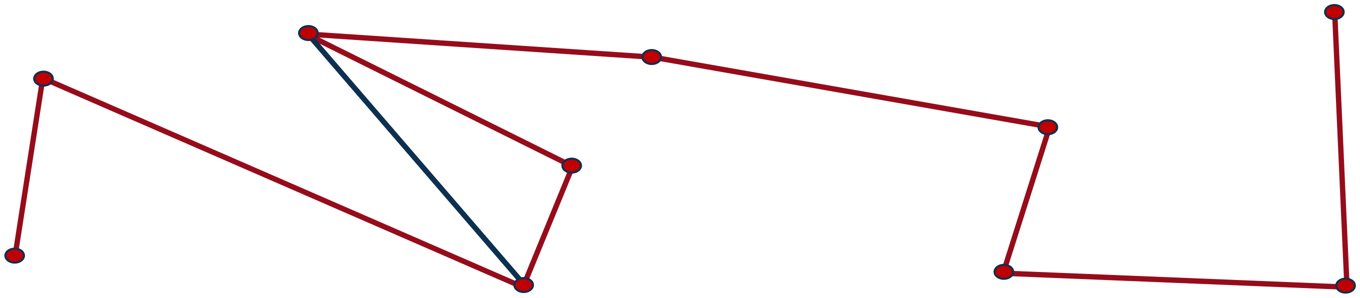
Imai-Iri

Find all possible valid shortcuts



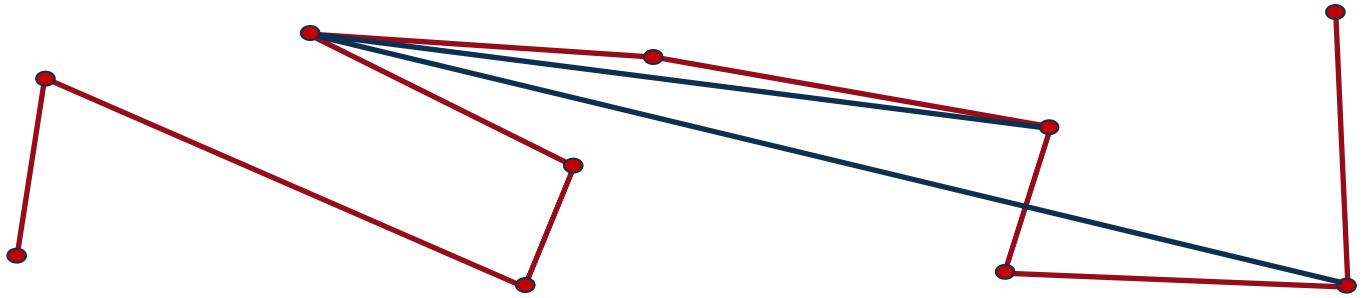
Imai-Iri

Find all possible valid shortcuts



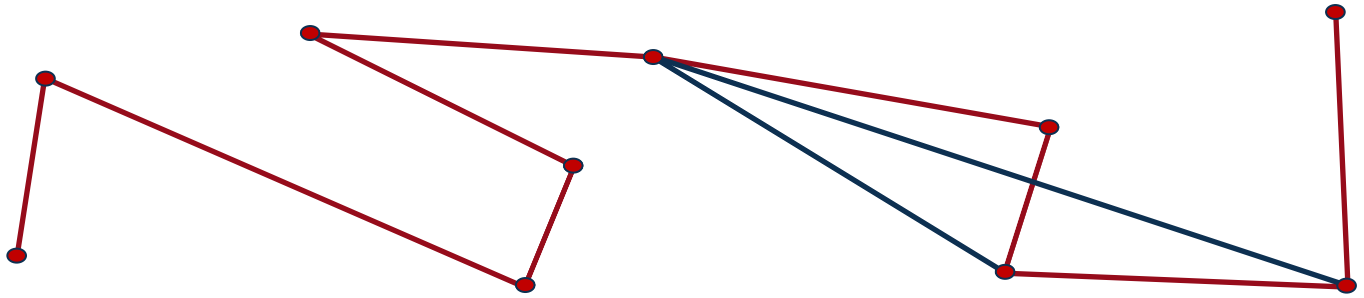
Imai-Iri

Find all possible valid shortcuts



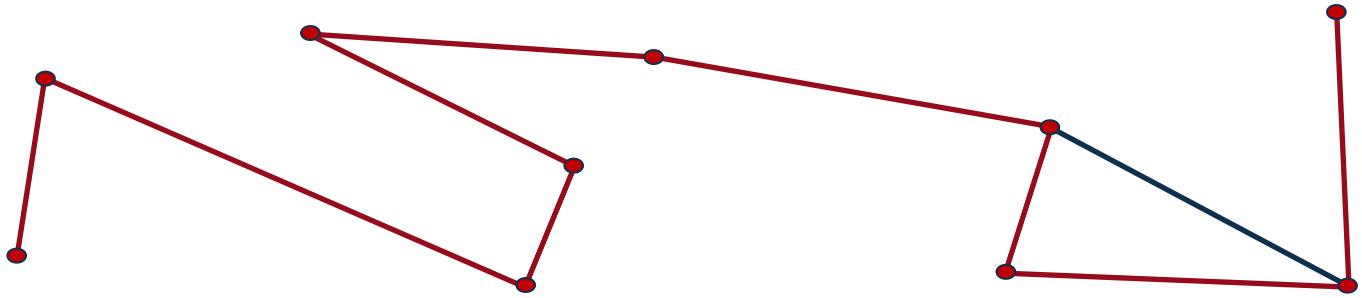
Imai-Iri

Find all possible valid shortcuts



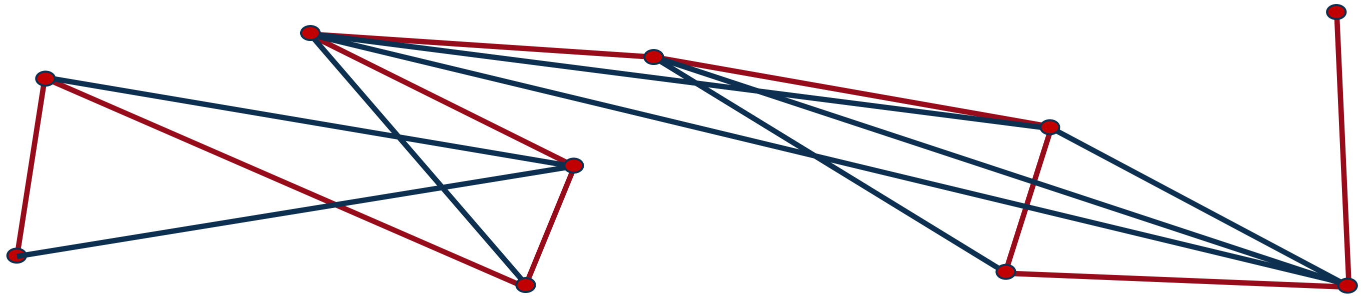
Imai-Iri

Find all possible valid shortcuts



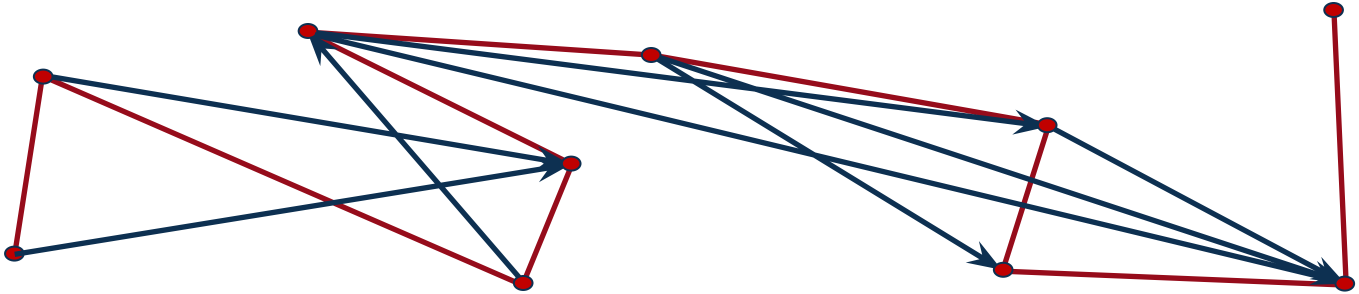
Imai-Iri

All possible shortcuts!



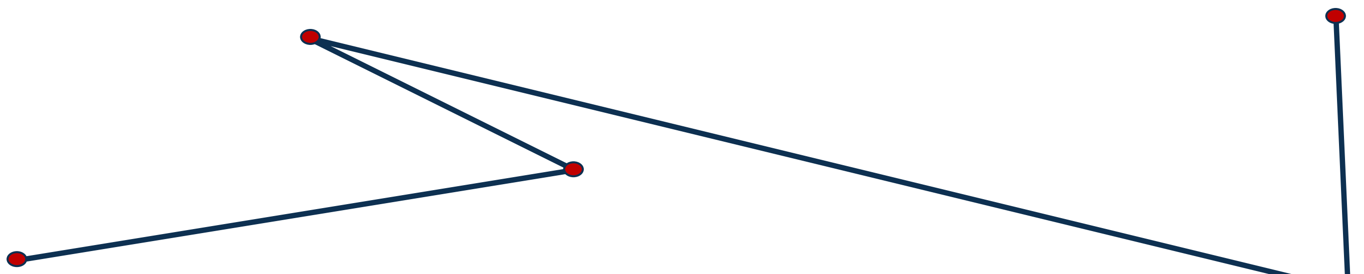
Imai-Iri

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 ?

Summary: Imai-Iri

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

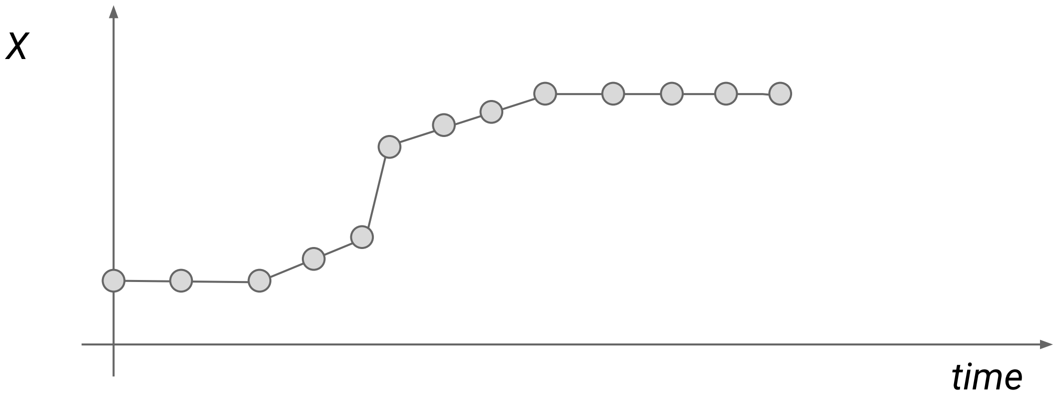
Output: A path with minimum number of edges

Improvements:

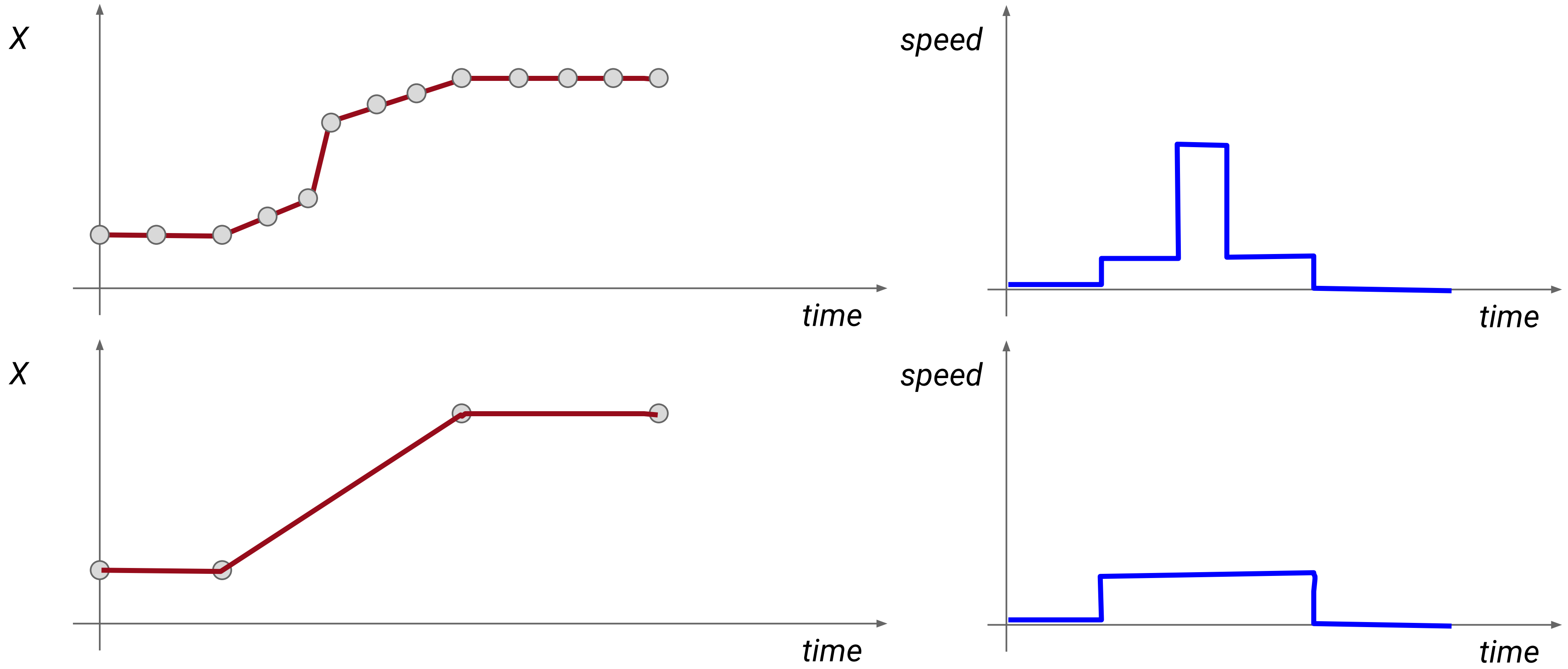
Chan and Chin'92: $O(n^2)$

Limits of the previous approaches

- What about time and speeds?
  - Time-stamps were never considered in the algorithms
  - They considered on impact on space / geometry of trajectories
  - What impact on time-related aspects, e.g. speed?

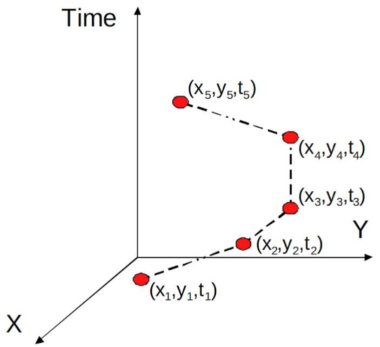


Impact on speed



Time-aware simplification methods

- Must consider the 3D (space + time) nature of point
- Simplest approach: modified Driemel at al.



Simple simplification **with speeds** ($P = \langle p_1,\ldots,p_n\rangle$, 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. $|q-p_i| > \varepsilon$ **or** $|AS(q,p_i) - AS(p_{i-1},p_i)| > \varepsilon$
    if no such vertex then set $i:=n$
    add p_i to P'
end
return P'

AS(a,b) = average speed between a and b
= dist(a,b) / [time(b) - time(a)]

INTERVALLO

How fast is a cow?

INTERVALLO

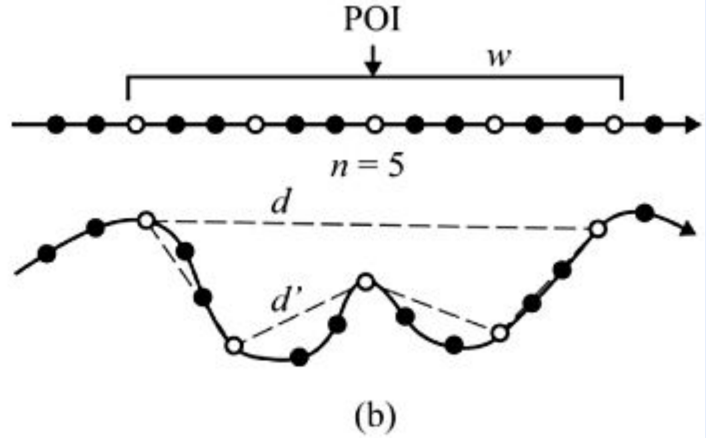
How fast is a cow?

- Trajectory compression / simplification changes the scale of the analysis
  - Simplified data → macroscopic analysis
  - Detailed data → microscopic analysis
- Several movement characteristics can be affected

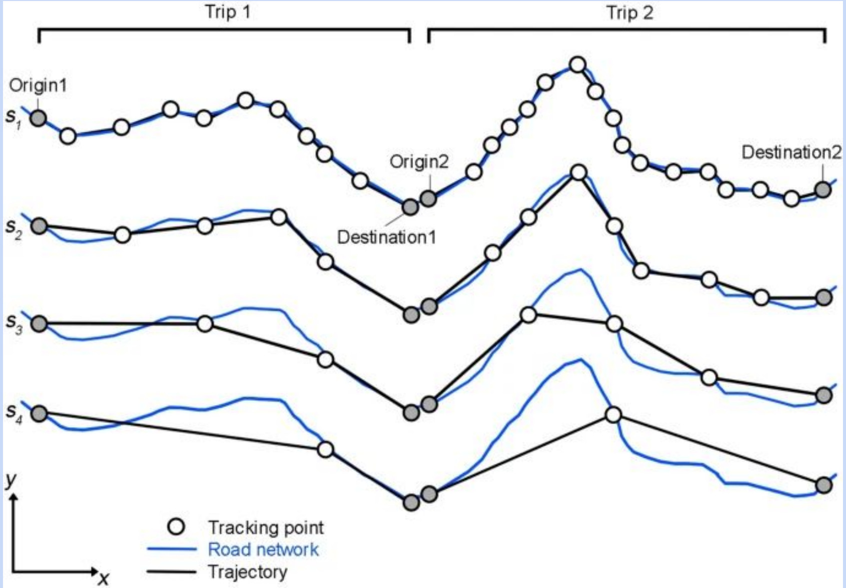
INTERVALLO

How fast is a cow?

How fast is a cow? Cross-Scale Analysis of Movement Data
Laube P, Purves RS (2011)



Understanding the impact of temporal scale on human movement analytics
Su, R., Dodge, S. & Goulias, K.G (2022)

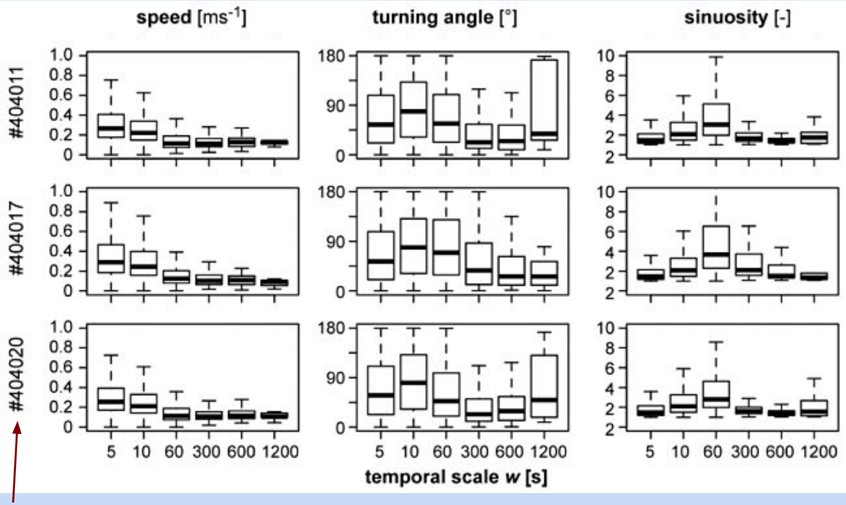


INTERVALLO

How fast is a cow?

How fast is a cow? Cross-Scale Analysis of Movement Data
Laube P, Purves RS (2011)

Understanding the impact of temporal scale on human movement analytics
Su, R., Dodge, S. & Goulias, K.G (2022)



Cow ID