Trajectory Similarity and Clustering

Teemu Pulkkinen, Petteri Nurmi

Questions

- What can we gain from clustering trajectories?
- How can we determine if two trajectories are similar?
- What are the benefits and problems of the different similarity measures?
- How can we use similarity measures for clustering?

Introduction

- Proliferation of GPS devices and indoor positioning brings about opportunities for collecting *trajectories*
- E.g., vehicles, but also animals and even phenomena like hurricanes
- It is safe to assume that objects that move in a similar way have similar guiding principles
- If we can detect this similarity, it is likely that we can discover interesting trends

Motivation

- Tracking animals and analyzing their trajectories, we can discover migratory behavior
- Understanding hurricane trajectories helps establish early warning systems
- Tracking customers in a supermarket carries several benefits for both researchers and retailers:
  - Find ”hot/coldspots” in the store; areas that suffer from congestion or are rarely visited

Motivation

- Detect common routes people take
  - Customers that move in similar patterns are likely looking for the same things, or are working under the same constraints
  - Plan displays accordingly
- Sometimes the motivation can even seem counter-intuitive:
  - Retailers **want** people to spend time in their stores, and expose them to shopping opportunities
  - A lot of shops reflect this in their layouts

Preliminaries

- Trajectories can be considered a special case of *time series*, in 3 dimensions
  - *Coordinates + time*
- *Time series analysis* is a well-defined, and well-researched, topic
- E.g., in marketing, this is used to analyze the development of stocks over time
- Most common approach is to analyze the shape of the actual time series object

Time series objects vs. trajectories

- Prior research provides many tools for comparing time series
- This translates nicely to the analysis of trajectories
- A popular approach is to consider the trajectory as a sequence of symbols
  - In other words, we can see it as a *string object*
- We can then compare two trajectories using approaches very similar to *edit distance* (or *Levenshtein distance*)

Similarity measures, features

- Several different approaches to comparing trajectories
- Most choose to focus on a specific set of problems
  - Pros and cons in all approaches
- **Metricity**
  - Distance measure that is metric is easier to use for indexing
    - Proper indexing speeds up clustering
  - Proofs of convergence, time complexity, etc. usually easier to justify

Metricity, in detail

- To be considered a metric, a distance measure has to satisfy 4 conditions:

- $D(x,y) \geq 0$, *non-negativity*
- $D(x,y) = 0 \text{ iff } x = y$, *identity of indiscernibles*
- $D(x,y) = D(y,x)$, *symmetry*
- $D(x,z) \leq D(x,y) + D(y,z)$, *triangle inequality*

Indexing

- Previous lecture mentioned storing trajectory data in location databases
- A metric distance measure allows for efficient indexing of trajectories
  - We can make assumptions about distances since we know they satisfy metricity conditions
  - Searching is efficient since we can ignore parts of the data that is justifiably irrelevant

Similarity measures, more features

- **Completeness**
  - Most distance measures force a comparison between all elements in both trajectories
    – A significant difference in only one section might overshadow the similarity in others
  - Comparing only sections that are similar means we avoid outliers
- **Efficiency**
  - Dynamic programming approach is usually demanding

Similarity measures, more features

- **Time dilation**
  - Similar trends might take place over different periods of time
  - Factor out speed as a variable; consider the overall shape
- **Robustness**
  - Systematic noise (e.g. from measurements) might have a cumulative effect
  - Outliers might have a significant impact on the total distance

Similarity measures: Euclidean

- The simplest possible measure between two time-series objects
- Compare the values in the objects at the same time instance, t_i

\+ Simple to implement, intuitive approach

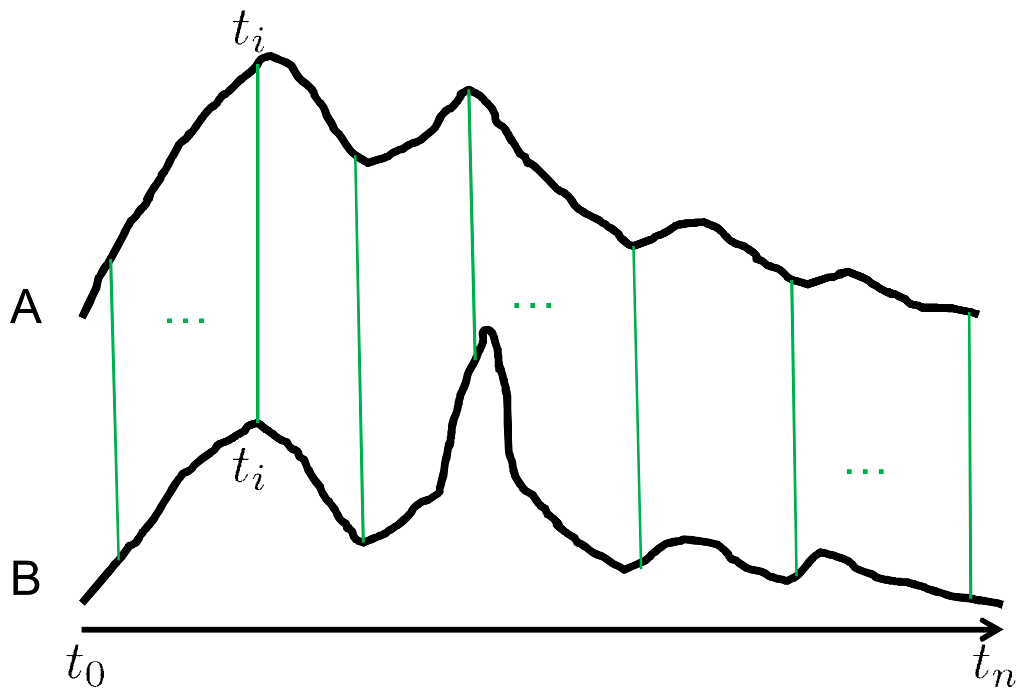
\+ Fastest approach

\- Naive approach means that no time dilation is taken into consideration

\- Any offset between objects has a cumulative effect

\- Noise is detrimental

Euclidean, visually



Dynamic programming

- Method for solving complex problems by breaking them into simpler sub-problems
- General technique applicable to any problem with *optimal substructure* and *overlapping sub-problems*
  - Optimal substructure: solution to given problem can be obtained as a solution to its sub-problems
  - Overlapping sub-problems: problem can be broken into sub-problems which can be reused
- Divide-and-conquer used when sub-problems are non-overlapping
- General form has $O(n^2)$ runtime

Dynamic programming

- Solving dynamic programming tasks involves three steps:
  1. Define sub-problems
  2. Write recurrence that relates sub-problems
  3. Identify and solve base cases
- Dynamic programming algorithms typically specified by
  - Recurrence equation (sub-problem solution)
  - Initialization conditions (base case)
- Most trajectory (and time-series) distance measures calculated using dynamic programming

Edit Distance

- Dissimilarity measure for two strings, defined as the minimum number of operations needed to transform one string into the other
- Levenshtein distance: edit distance with three allowed string manipulation operations
  - Insert: adding a symbol
  - Delete: remove a symbol
  - Substitution: changing a symbol to another
- The common approach to solving this distance is to use dynamic programming

Edit distance: recursion

The value for each element in the EDIT matrix (i.e., the recursion equation) is defined as follows:

$$D_{EDIT}(A_i, B_j) = \begin{cases} EDIT(i-1, j-1) & \text{if } A_i = B_j \\ 1 + \min(EDIT(i-1, j-1), & \text{if } A_i \neq B_j \\ EDIT(i, j-1), & \\ EDIT(i-1, j)) & \end{cases}$$

Additionally the trivial case where either string is empty:

$$D_{EDIT}(A, B) = length(A) \text{ if } length(B) = 0, \text{ and vice versa}$$

Edit distance: pseudo-code

(adapted from matlab)

initialization

```
for i=0:m
    edit(i+1,1) = i;
end

for j=0:n
    edit(1,j+1) = j;
end
```

recursion

```
for i=2:m+1
    for j=2:n+1
        if A(i) ~= B(i)
            penalty = 1;
        else
            penalty = 0;
        end
        contenders = [edit(i-1,j)+1,edit(i,j-1)+1,edit(i-1,j-1)+penalty];
        edit(i,j) = min(contenders);
end

distance = edit(m+1,n+1);
```

Edit distance, example

$$D_{EDIT}(bored, snore) =$$

		s	n	o	r	e
	0	1	2	3	4	5
b	1	1	2	3	4	5
o	2	2	2	2	3	4
r	3	3	3	3	2	3
e	4	4	4	4	3	2
d	5	5	5	5	4	3

transform
insert/delete
same

$$= 3$$

Similarity measures: DTW

- *Dynamic Time Warping*
- A dynamic programming approach to time series analysis
- Behaves like edit distance most of the time, but carries a dynamic penalty:
  - Instead of adding ”1” as a cost, consider the actual distance between the elements
+ Time differences are taken into account
+ Provides a better match than Euclidean measure
- Mapping between all objects in both trajectories means every discrepancy is considered and added to the penalty
- Not a metric (not satisfy triangle inequality)
  - But there are works that have looked into deriving bounds on the distances which can be used for indexing

DTW, more formally

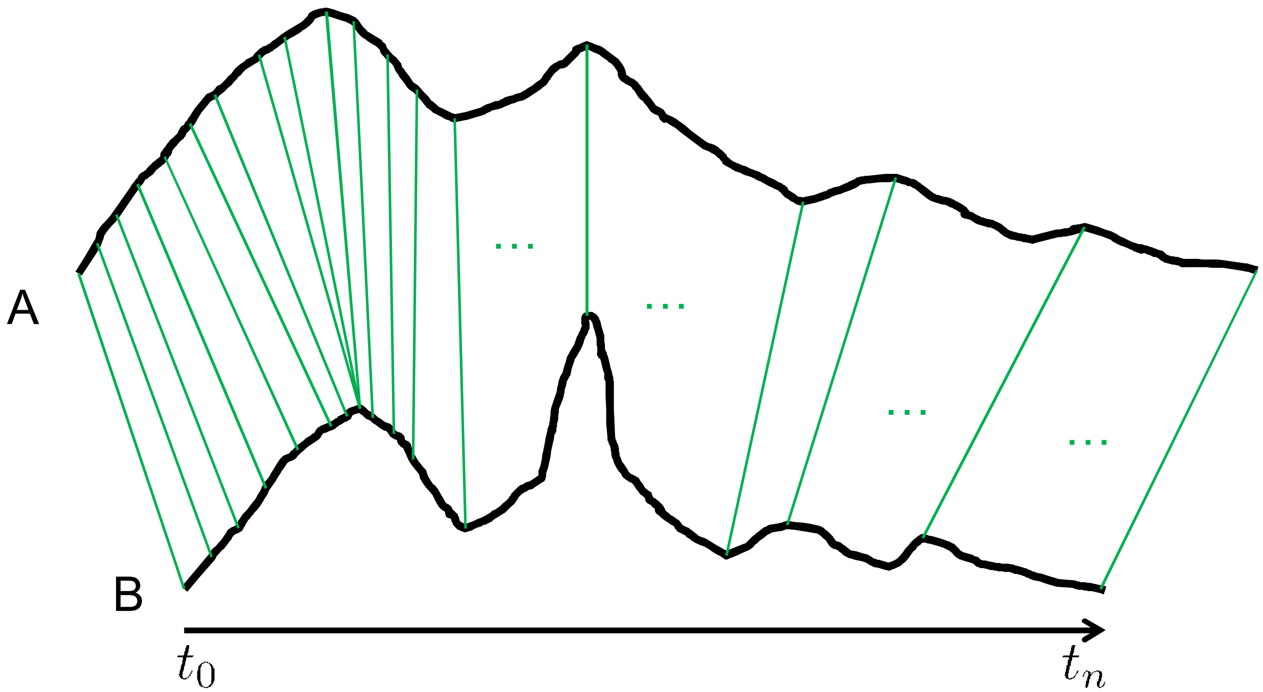
DTW calculation nearly identical to edit distance:

$$D_{DTW}(A_i, B_j) = \begin{cases} DTW(i-1, j-1) & \text{if } A_i = B_j \\ \boxed{\text{cost}} + \min(DTW(i-1, j-1), & \text{if } A_i \neq B_j \\ DTW(i, j-1), & \\ DTW(i-1, j)) & \end{cases}$$

where

$$cost = d(A_i, B_j)$$

DTW, visually



Similarity measures: LCSS

- *Longest Common SubSequence*
- Inverts the problem: which parts of the trajectory **are** similar
- Similar dynamic programming approach
  - Increase similarity when elements match
  - ”Matching” can be defined by a distance threshold

+ Ignores parts that don’t match
+ Good with noise and outliers (because of the above)
+ Allows for time distortion

- Not metric

LCSS, formally

LCSS in its traditional form is actually a *similarity measure*:

$$S_{LCSS}(A_i, B_j) = \begin{cases} \emptyset & \text{if } \ i = 0 \text{ or } j = 0 \\ 1 + LCS(A_{i-1}, B_{j-1}) & \text{if } A_i = B_j \\ \max(LCS(A_i, B_{j-1}), LCS(A_{i-1}, B_j)) & \text{if } A_i \neq B_j \end{cases}$$

But we can easily express this as a *distance* (or *dissimilarity*) *measure:*

$$D_{LCSS}(A, B) = 1 - \frac{S_{LCSS}(A,B)}{min(n,m)},$$

where *n* and *m* are the lengths of A and B, respectively.

LCSS, example

$S_{LCSS}(bored, snore) =$

		s	n	o	r	e
	0	0	0	0	0	0
b	0	b!=s 0	b!=n 0	… 0	etc. 0	0
o	0	o!=s 0	… 0	o=o 1	o!=r 1	1
r	0	… 0	0	1	2	2
e	0	0	0	1	2	3
d	0	0	0	1	2	$\boxed{3}$

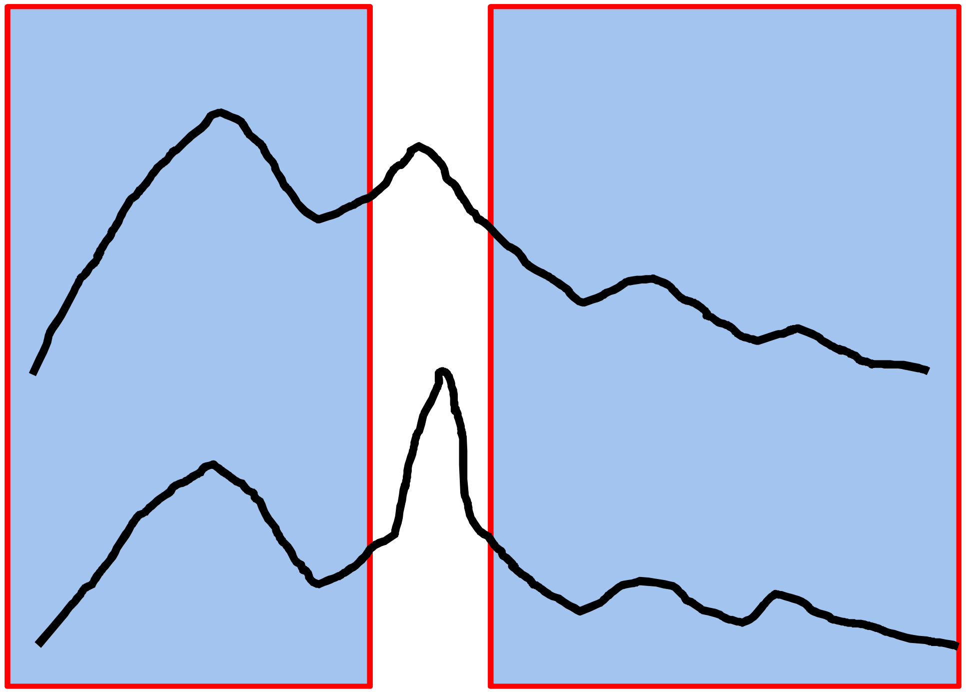
$= 3$

→ Different character: max([i-1,j],[i,j-1])

→ Same character, add 1 to score

LCSS, visually

Outlier is ignored



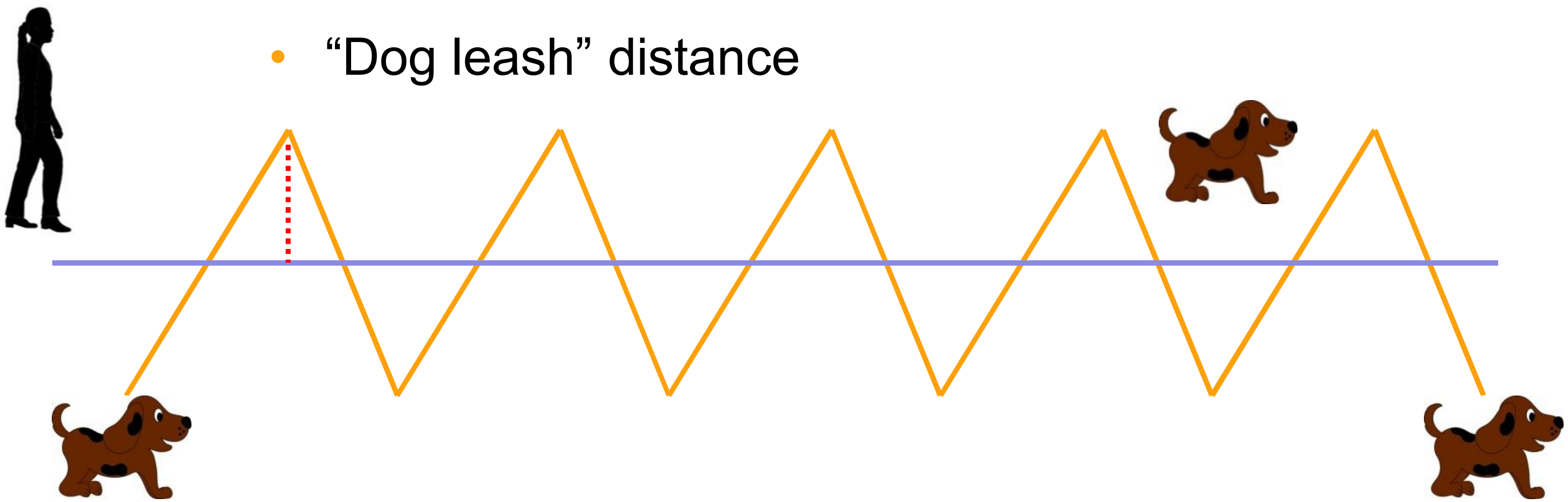
Similarity = total length of common subsections

Similarity measures: variants of DTW and Edit Distance

- EDR: *Edit Distance on Real sequence*
  - Relaxes equality requirement
  - \+ Good with noise
  - \+ More accurate than LCSS
  - \- Not metric
- ERP: *Edit distance with Real Penalty*
  - Considers distance between elements like DTW, but compares them to fixed value instead of each other
  - \+ Handles distortion
  - \+ Metric
  - \- Not as good with noise

Fréchet distance

- Shape-based similarity metric for curves
  - Minimum length of a leash required to connect a dog and its owner, constrained on two separate paths
  - Velocity of dog and owner can change, but backtracking is not allowed
  - “Dog leash” distance



Calculating Fréchet distance

- Can be approximated using *discrete Fréchet* distance
  - Considers only positions of the leash where endpoints are located at vertices of polygonal curves
  - Also called *coupling distance* as examines couplings of discrete points
  - Can be solved using dynamic programming

$$D_{Fréchet}(A_i, B_i) = \max \begin{cases} \min\left[D(i-1,j), D(i-1,j-1), D(i,j-1)\right] \\ d(A_i, B_i) \end{cases}$$

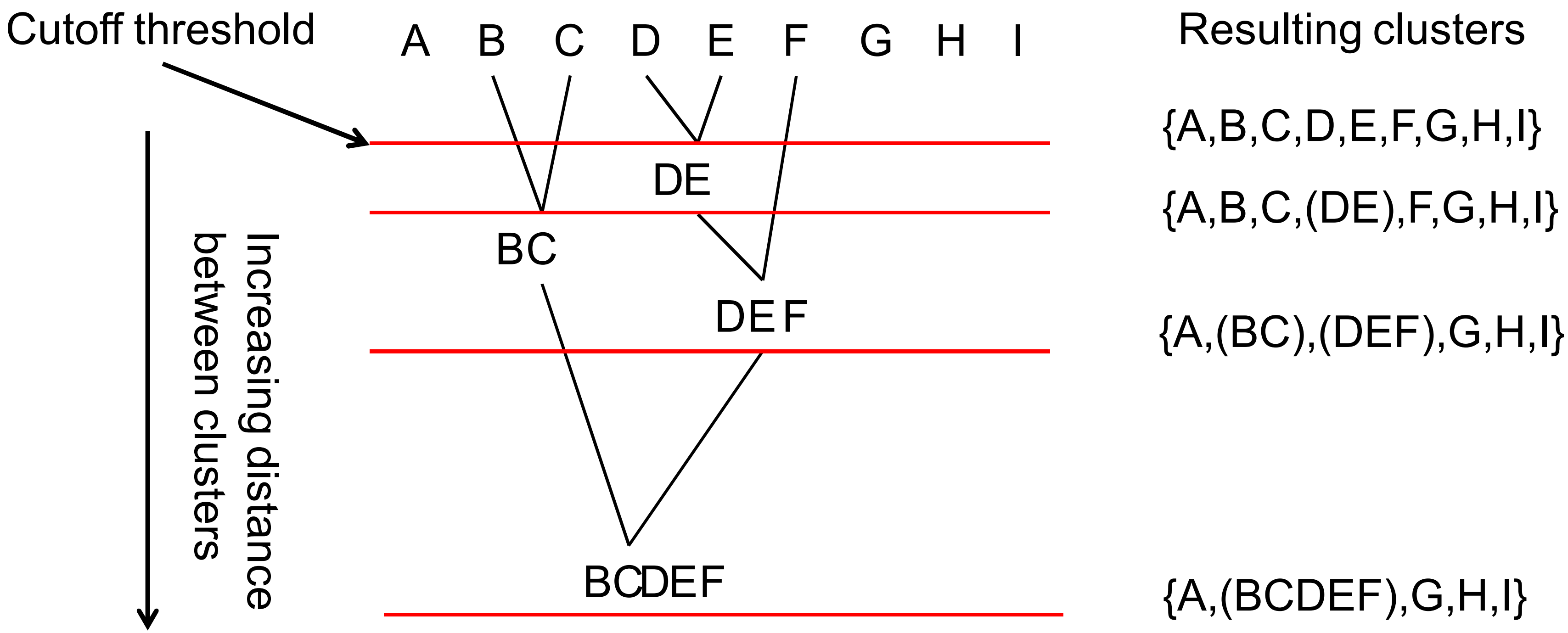
Clustering

- Once we have determined the similarity between trajectories, we can cluster them into meaningful groups
- The intuition is that trajectories in a cluster will exhibit similar characteristics
- Traditional clustering approaches often define a *centroid* around which objects are clustered
  - Not always clear what this means in terms of trajectories
- A common approach to clustering once the distance between all points is known is *agglomerative* (or *hierarchical*) *clustering.*

Agglomerative clustering

- Some approaches described in earlier lectures
- Start with every object in its own cluster
- Merge clusters that are ”close enough” to each other (based on some criteria)
- Determine a cutoff threshold where clustering is considered complete
  - E.g., maximum/minimum/average distance between elements of each cluster (distance threshold)
  - Pre-defined limit for the number of clusters

Agglomerative clustering, example



Agglomerative clustering: linkage

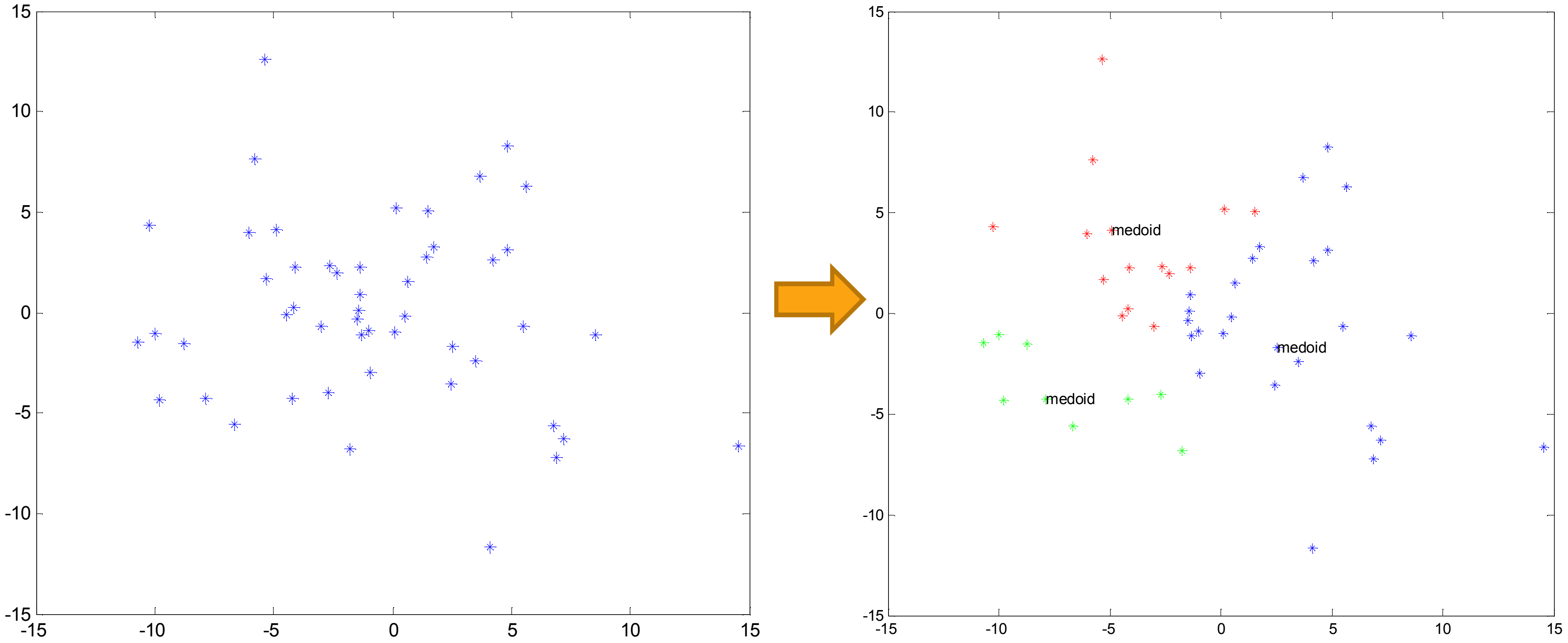
- Similarity measures covered thus far provide pairwise similarity
- To merge points, a similarity for sets of objects is required
  - This is called a *linkage*
- Examples of linkages:

Name	Formula
Complete-linkage	$\max \{ d(a,b) : a \in A, b \in B \}$
Single-linkage	$\min \{ d(a,b) : a \in A, b \in B \}$
Mean linkage	$(\lvert A \rvert \lvert B \rvert)^{-1} \sum\sum d(a,b)$
Centroid linkage	$\lvert\lvert c_A - c_B \rvert\rvert$

K-medoids

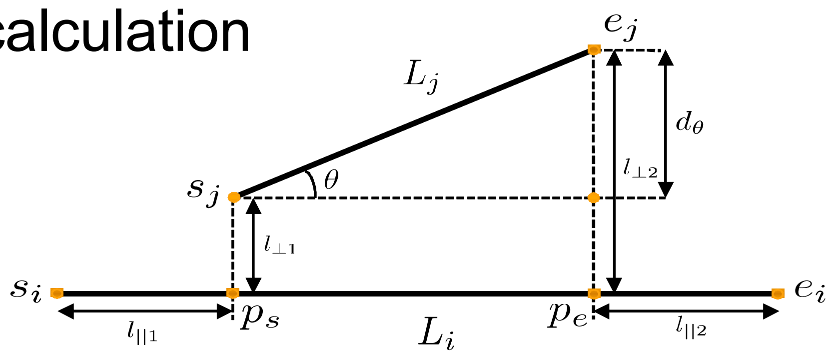
- Related to the K-means approach from earlier lecture
- Cluster points around a *medoid* instead of the center point of the elements
  - Usually the most centrally located object, i.e. the one with the smallest average distance to the rest of the cluster members
- More robust than K-means
  - Minimizes pairwise dissimilarities instead of sum of squared Euclidean distances
    - Outliers have lesser effect
- Has a well-defined ”representative” of the cluster that is actually an object (trajectory) itself

K-medoids, example



TRACLUS

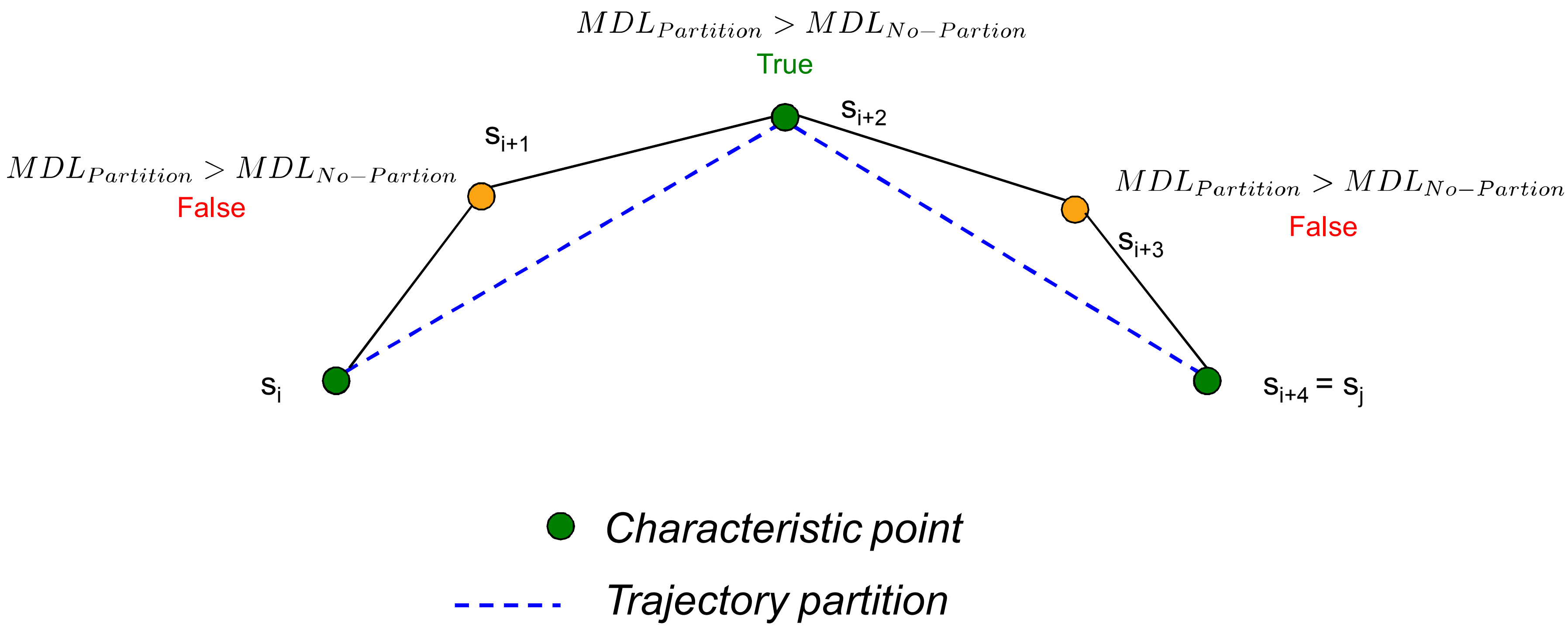
- *TRAjectory CLUStering*
- A trajectory clustering approach that considers *sub-trajectories*
  - Parts of trajectories might match even when the trajectory as a whole does not
- Distance is measured between similar *segments*
- Trigonometric measures used for distance calculation



TRACLUS

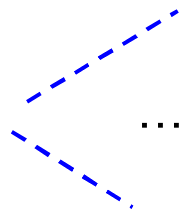
- The MDL principle is used to define *characteristic points* in the trajectory:
  - Balance between ***preciseness***[1] and ***conciseness***[2]
  - [1] Difference between original trajectory and model should not be too large
  - [2] To be beneficial, the model should be smaller than the trajectory it is modelling
- Segments between characteristic points then become the target for clustering
- Clustering performed using DBSCAN
  - Lines instead of points, but Epsilon neighborhoods etc. remain

TRACLUS, example



TRACLUS, example

1. Partition trajectories

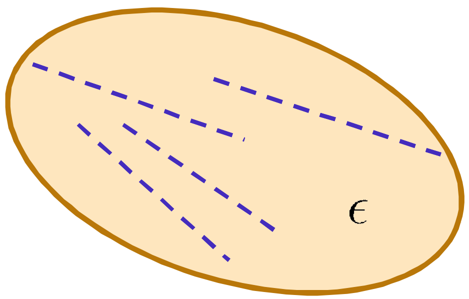


2. Calculate weighted sum of distances between them

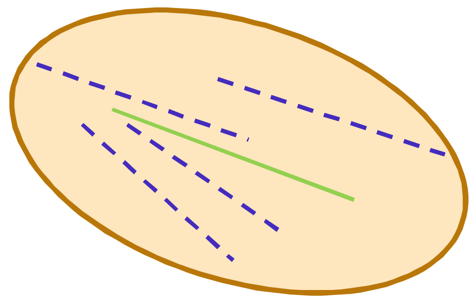
$$d_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}} \qquad d_{||} = Min(l_{||1}, l_{||2})$$

$$d_\theta = ||L_j|| \times \sin\theta$$

3. Apply DBSCAN



4. Define *representative* trajectory



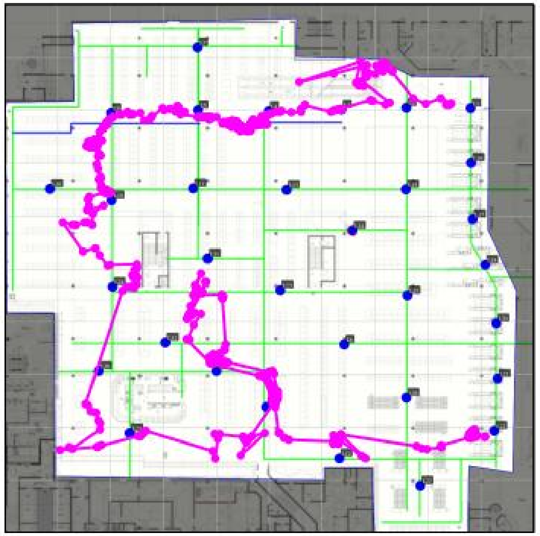
Case Study: Retail Analytics

- Task: identify different shopping styles from customer pathways
  - Museum visitors have been shown to have 4 different styles, how many styles have customers of supermarkets?
- Measurements: indoor positioning measurements collected from tags installed on handlebars of shopping carts
- Preprocessing:
  - Path extraction / segmentation: identifying when a shopping visit starts or ends
    - Based on various temporal and spatial heuristics
    - Considering opening times, entrance and cashier areas
  - Data cleaning:
    - Removing erroneous measurements and paths containing insufficient amount of measurements

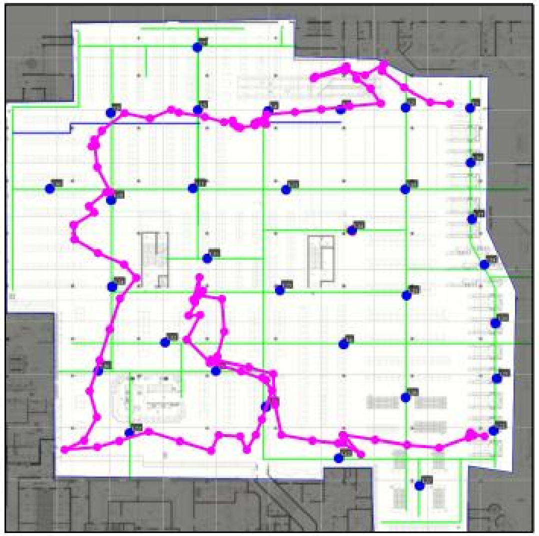
Case Study: Retail Analytics

- Feature extraction: original path converted into percentile paths
  - Length percentile: calculate the length of a 1% of the overall path and replace measurements with points that are closest to each i%
    - – First point entry to shop, last cashier
    - – Second point measurement that closest to 1% length of the shopping route
  - Time percentile: similar to the length percentile but uses time instead
    - – If a path lasts 100minutes, the 2nd point is the measurement that is 1 minute along the path

Case Study: Retail Analytics



Original



Length percentile

Time percentile

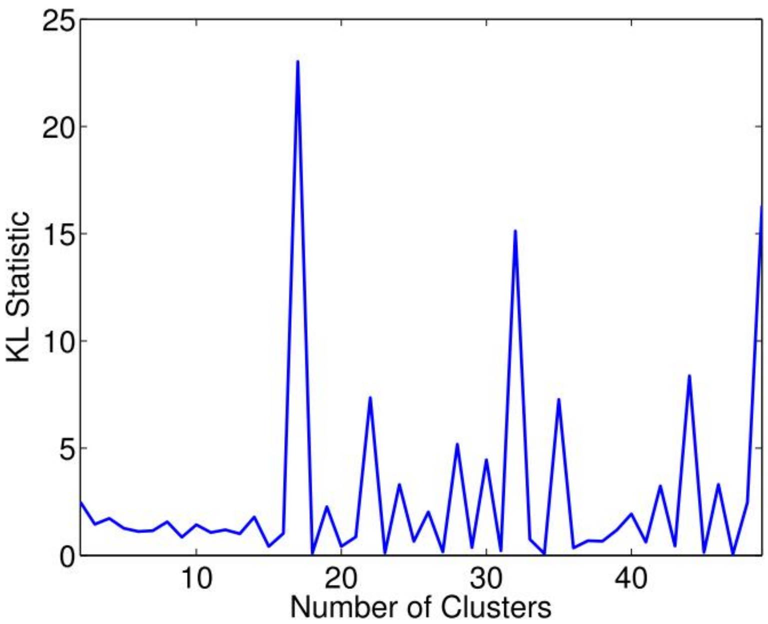
Case Study: Retail Analytics

- Shopping styles determined by performing trajectory clustering on the resulting percentile pathways
  - K-medoids using Euclidean distance
    - Euclidean distance can be used thanks to the percentile conversion before clustering
    - The number of shopping styles corresponds to the “optimal” value of k
  - Cluster count determined using the KL criterion

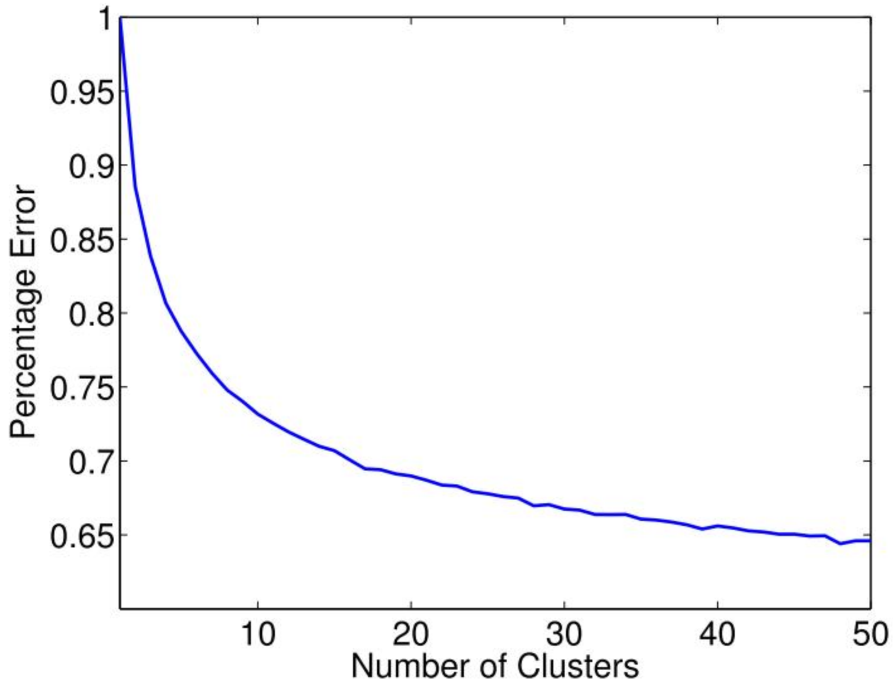
$$\mathrm{DIFF}(k) = (k-1)^{2/p}\, cost_{k-1} - k^{2/p}\, cost_k$$

$$\mathrm{KL}(k) = \left| \frac{\mathrm{DIFF}(k)}{\mathrm{DIFF}(k+1)} \right|$$

Case Study: Retail Analytics



KL

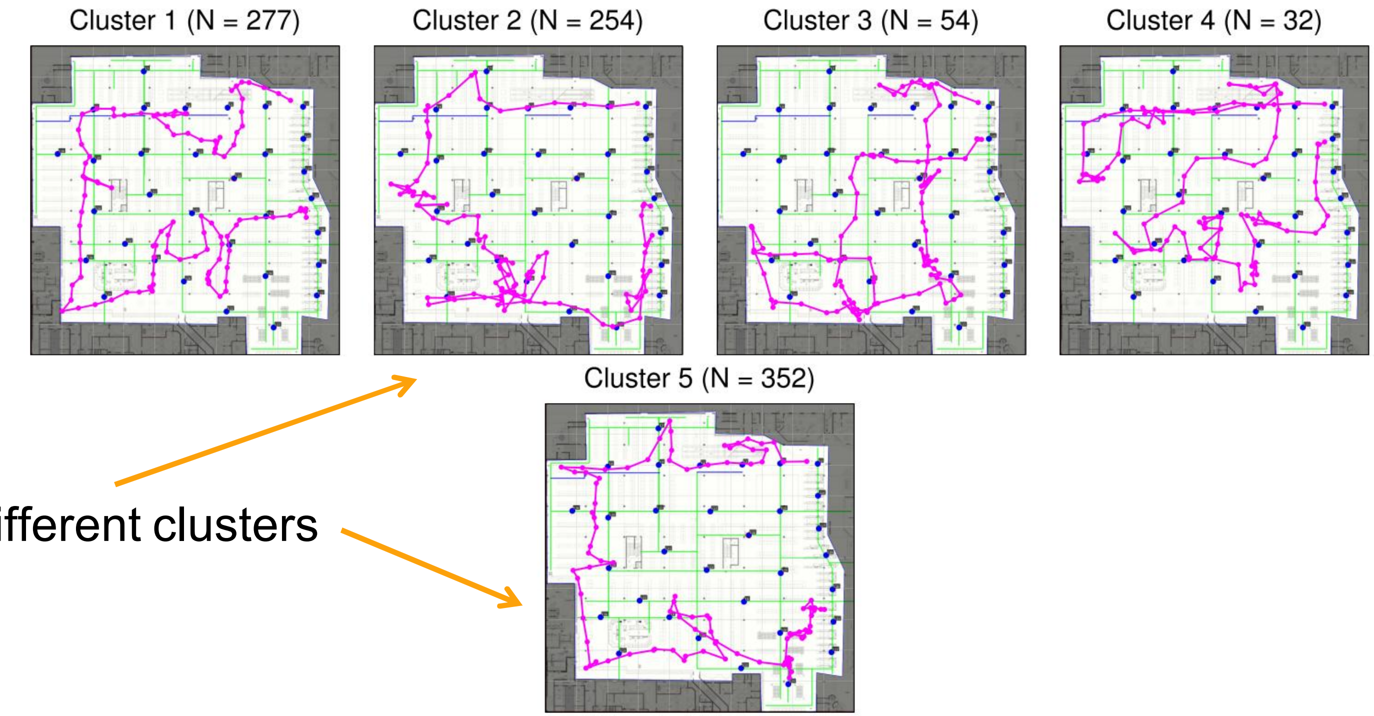


Scree

Case Study: Retail Analytics

- Best results obtained using a hierarchical clustering scheme
  - Paths manually split into short (14), medium (5), and long (12) duration paths
    - Based on predefined temporal thresholds
    - Selected from an earlier study
    - 17 clusters when all data considered together

Case Study: Retail Analytics



Centroids of different clusters

Summary

- Trajectories can exhibit trends when clustered together
  - Useful for analysis
- The distance between trajectories is usually calculated with a dynamic programming approach
  - Main difference between approaches is how they calculate the matrix
  - DTW: consider the distance between elements
  - LCSS: compare sections that are similar

Summary

- Hierarchical clustering can use a distance matrix to define clusters based on the distances between the elements in them
- Some approaches consider *partitions* of trajectories
  - TRACLUS combines MDL simplification with a modified version of DBSCAN

Literature

- ERP:
  - Chen, L. & Ng, R.: *On the marriage of Lp-norms and edit distance* Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB Endowment, **2004**, 792-803
- EDR:
  - Chen, L.; Özsu, M. T. & Oria, V.:
    Robust and Fast Similarity Search for Moving Object Trajectories
    Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, **2005**, 491-502
- TRACLUS:
  - Lee, J.-G.; Han, J. & Whang, K.-Y.:
    Trajectory clustering: a partition-and-group framework
    Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD), ACM, **2007**, 593-604