Distributed Spatio-Temporal Similarity Search

by

Demetris Zeinalipour

University of Cyprus &
Open University of Cyprus

Tuesday, July 4th, 2007, 15:00-16:00, Room #147 Building 12
European Thematic Network for Doctoral Education in Computing,
Summer School on Intelligent Systems
Nicosia, Cyprus, July 2-6, 2007

http://www.cs.ucy.ac.cy/~dzeina/

Disclaimer

Feel free to use any of the following slides for educational purposes, however kindly acknowledge the source.

We would also like to know how you have used these slides, so please send me emails with comments or suggestions.

This presentation is available at the URL:

http://www.cs.ucy.ac.cy/~dzeina/talks.html

* Thanks to Michalis Vlachos & Spiros Papadimitriou (IBM TJ Watson) and Eamonn Keogh (University of California – Riverside) for many of the illustrations presented in this talk.

Acknowledgements

This presentation is mainly based on the following paper:

``Distributed Spatio-Temporal Similarity Search''
D. Zeinalipour-Yazti, *S. Lin, D. Gunopulos,*
*ACM 15th Conference on Information and Knowledge Management, (**ACM CIKM 2006**), November 6-11, Arlington, VA, USA, pp.14-23, August 2006.*

Additional references can be found at the end!

Presentation Objectives

- **Objective 1: Spatio-Temporal Similarity Search problem.** I will provide the algorithmics and “visual” intuition behind techniques in centralized and distributed environments.

- **Objective 2: Distributed Top-K Query Processing problem.** I will provide an overview of algorithms which allow a query processor to derive the K highest-ranked answers quickly and efficiently.

- **Objective 3:** To provide the context that glues together the aforementioned problems.

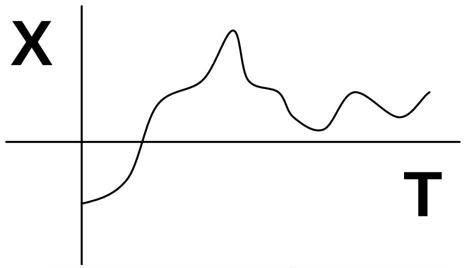
Spatio-Temporal Data (STD)

- **Spatio-Temporal Data** is characterized by:
  - A temporal (time) dimension.
  - At least one spatial (space) dimension.

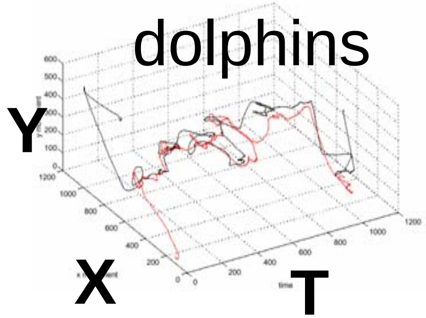
- **Example:** A car with a GPS navigator
  - Sun Jul 1st 2007 11:00:00 (time-dimension)
  - Longitude: 33° 23' East (X-dimension)
  - Latitude: 35° 11' North (Y-dimension)

Spatio-Temporal Data

- **1D (Dimensional) Data**
  - A car turning left/right at a static position with a moving floor
  - Tuples are of the form: (time, x)



- **2D (Dimensional) Data**
  - A car moving in the plane.
  - Tuples are of the form: (time, x, y)



- **3D (Dimensional) Data**
  - An Unmanned Air Vehicle
  - Tuples are of the form: (time, x, y, z)

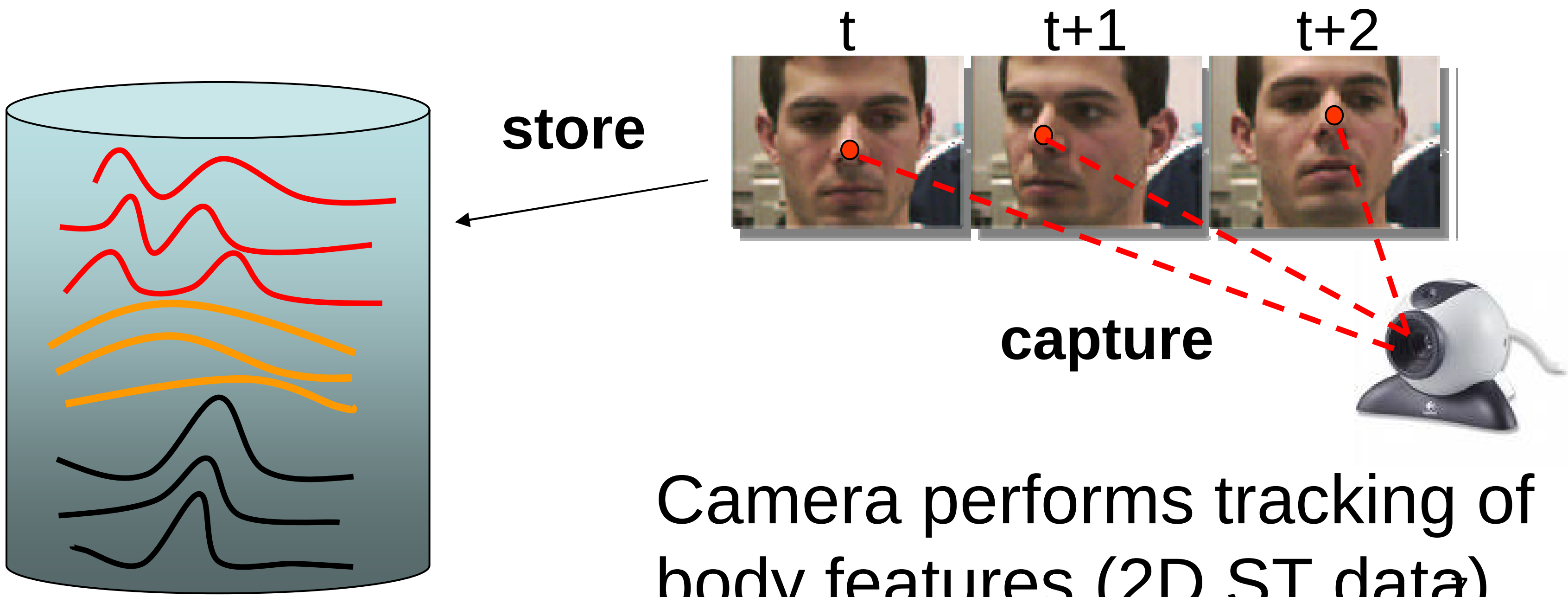
For simplicity, most examples we utilize in this presentation refer to 1D spatiotemporal data.

Centralized Spatio-Temporal Data

- **Centralized ST Data**

  When the trajectories are stored in a centralized database.

- ***Example:*** *Video-tracking / Surveillance*



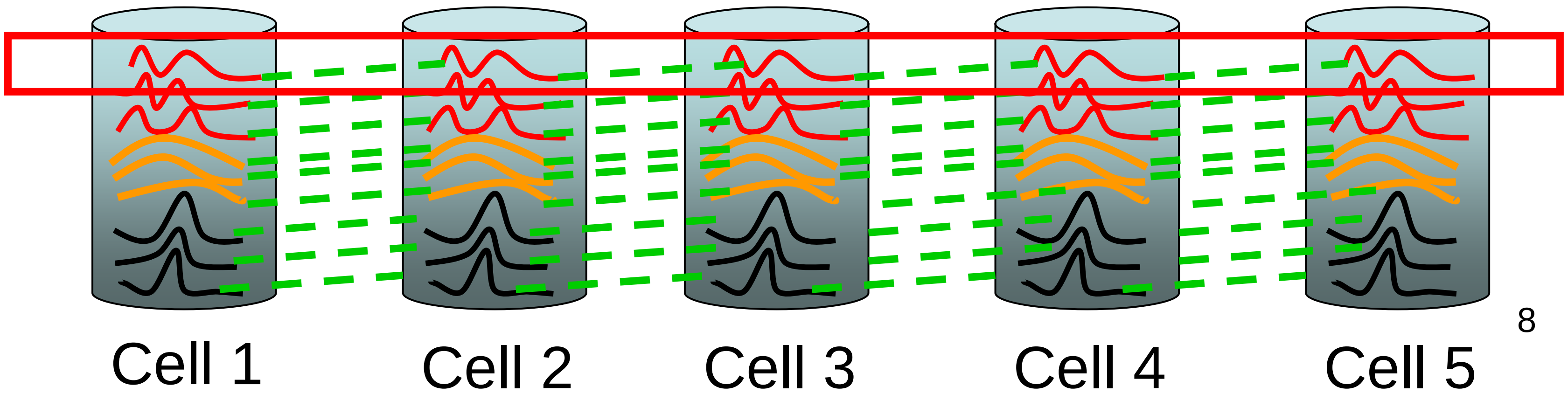
Camera performs tracking of body features (2D ST data)

Distributed Spatio-Temporal Data

Distributed Spatio-Temporal Data

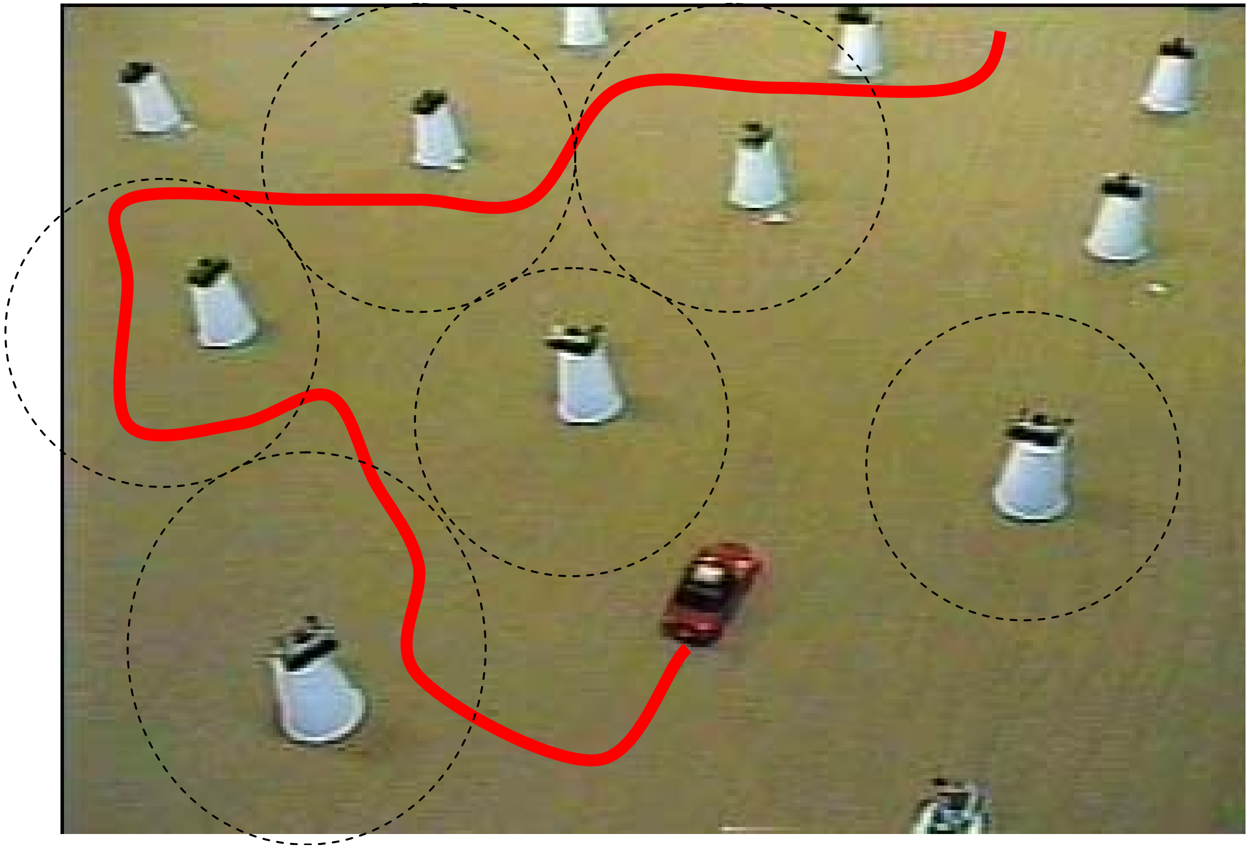
- *When the trajectories are vertically fragmented across a number of remote cells.*

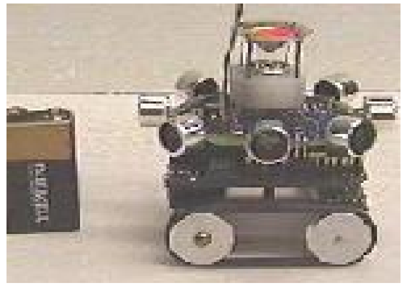
- *In order to have access to the complete trajectory we must collect the distributed subsequences at a centralized site.*



Distributed Spatio-Temporal Data

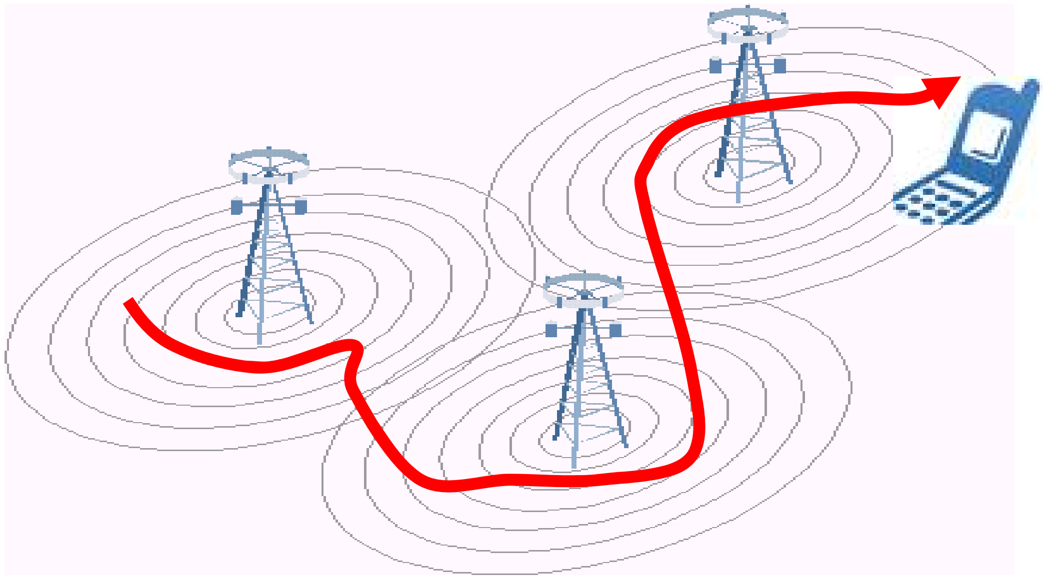
- **Example I (Environment Monitoring)**
  - A sensor network that records the motion of bypassing objects using sonar sensors.





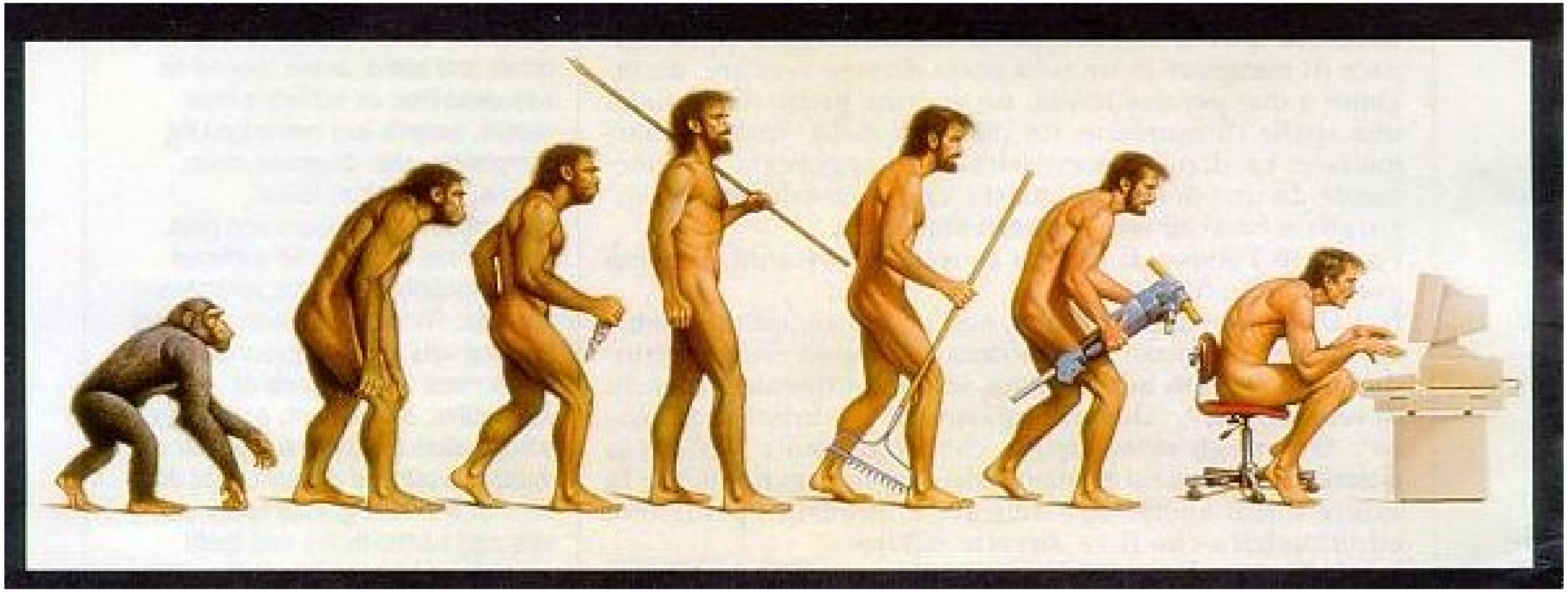
Distributed Spatio-Temporal Data

- **Example II (Enhanced 911):**
  – e911 automatically associates a physical address with every mobile user in the US.
  – Utilizes either GPS technologies or signal strength of the mobile user to derive this info.



Similarity

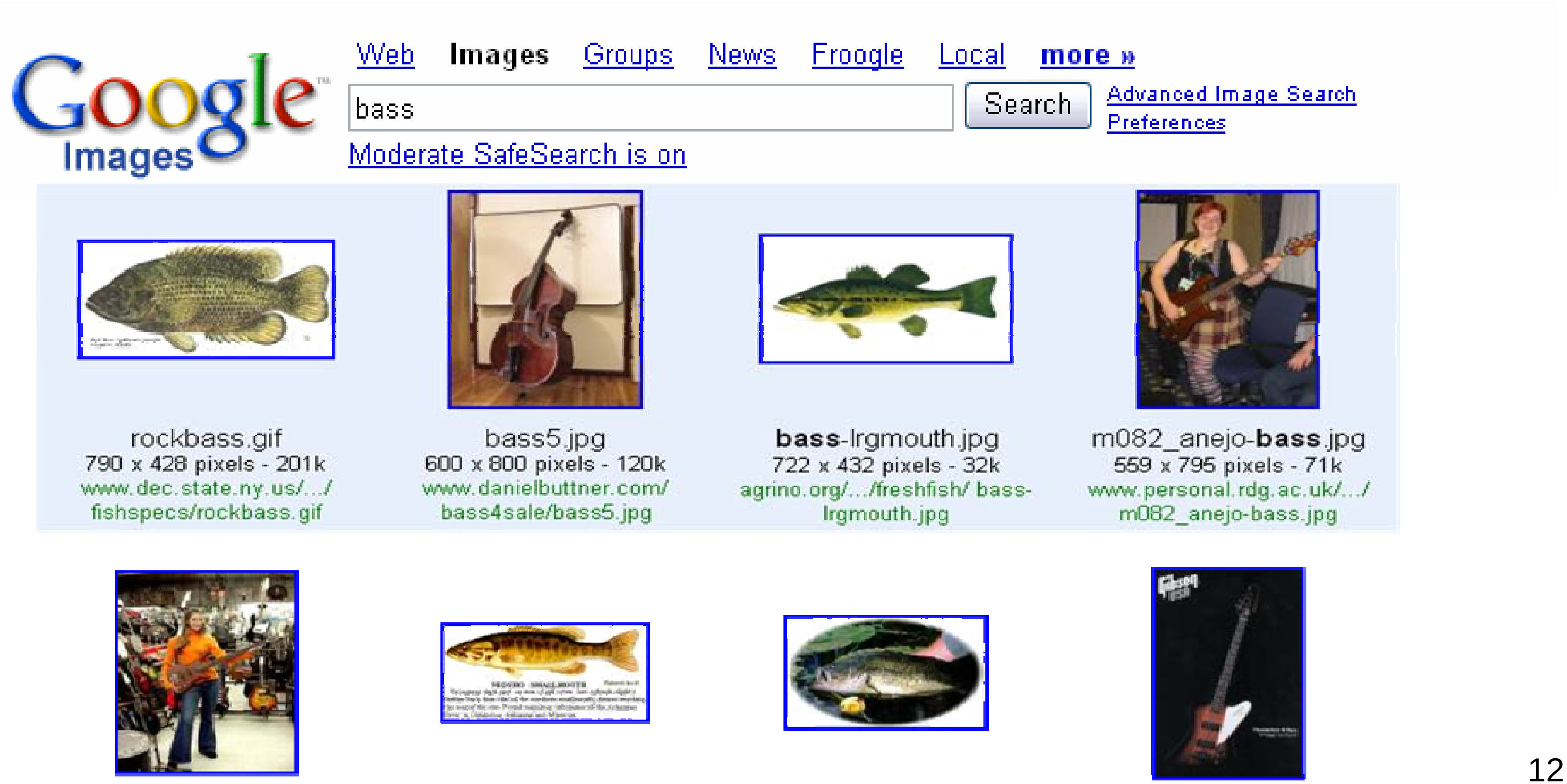
- A proper definition usually depends on the application.



- Similarity is always **subjective!**

Similarity

- *Similarity depends on the **features** we consider (i.e. how we will describe the sequences)*



Similarity and Distance Functions

- *Similarity between two objects A, B is usually associated with a* ***distance function***
- *The distance function measures the distance between A and B.*

Low Distance between two objects

==

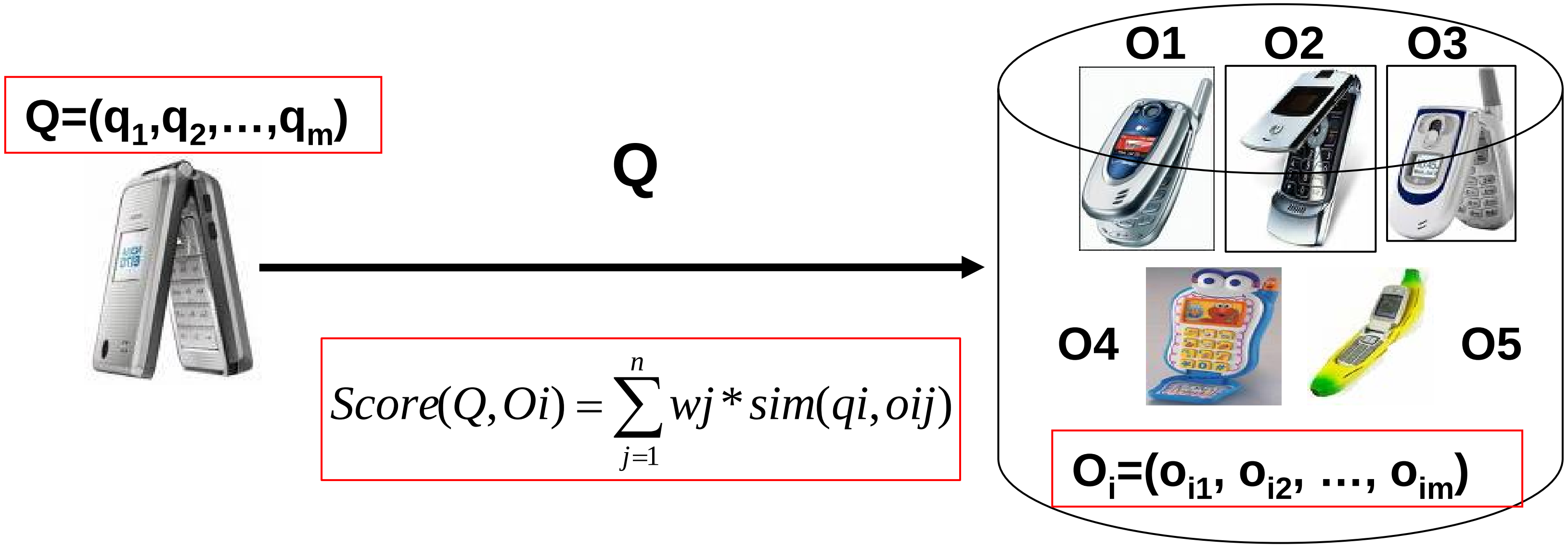
High similarity

- *Metric Distance Functions (e.g. Euclidean):*
  - *Identity: $d(x,x)=0$*
  - *Non-Negativity: $d(x,y)>=0$*
  - *Symmetry: $d(x,y) = d(y,x)$*
  - *Triangle Inequality: $d(x,z) <= d(x,y) + d(y,z)$*
- *Non-Metric (e.g., LCSS, DTW): Any of the above properties is not obeyed.*

Similarity Search

Example 1: Query-By-Example in Content Retrieval

- Let Q and m objects be expressed as **vectors of features** e.g. Q=(“color=#CCCCCC”, ”texture=110”, shape=“Λ”, .)
- **Objective: Find the K most similar pictures to Q**



- Answers are **fuzzy**, i.e., each answer is associated with a score (O3,0.95), (O1,0.80), (O2,0.60),….

Spatio-Temporal Similarity Search

Examples

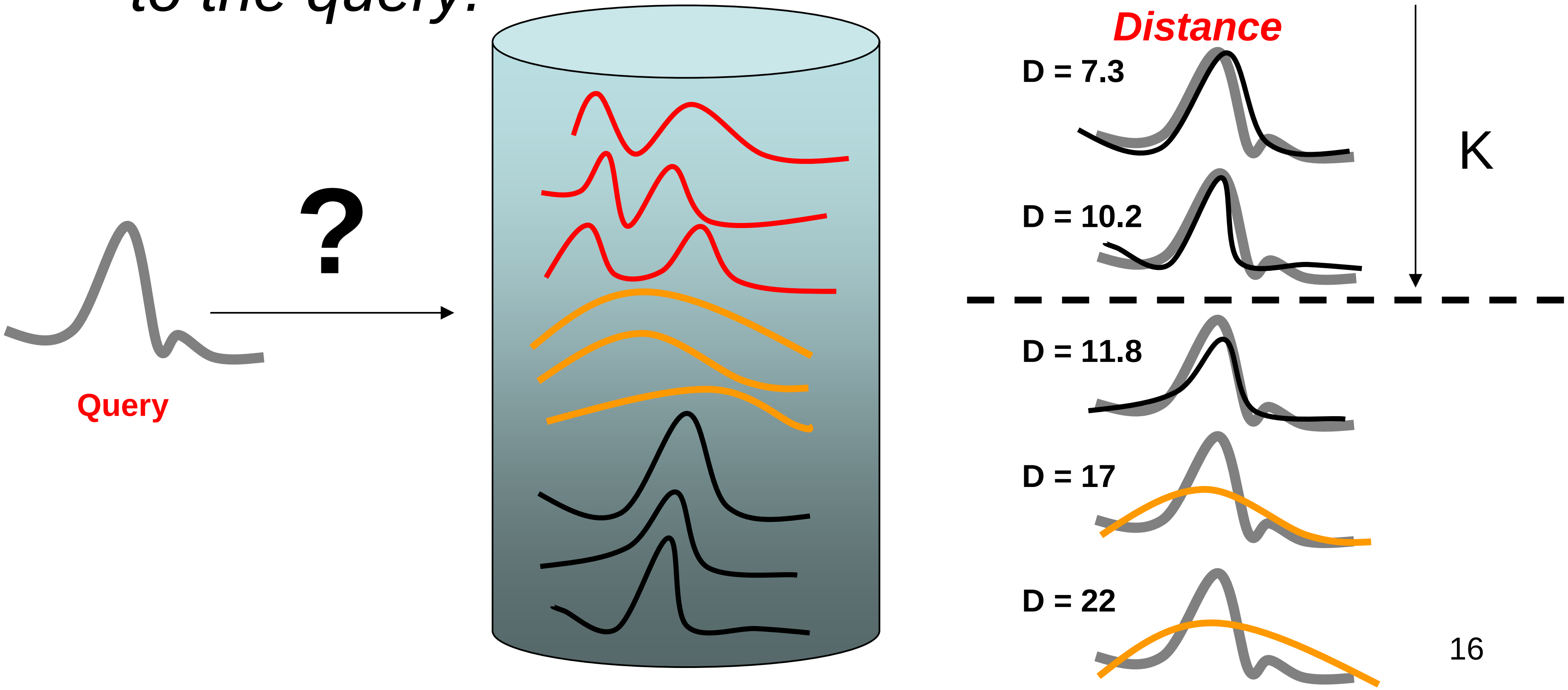
*- **Habitant Monitoring:** "Find which animals moved similarly to Zebras in the National Park for the last year". Allows scientists to u*nderstand animal migrations and interactions"

*- **Big Brother Query:** "Find which people moved similar to person A"*

Spatio-Temporal Similarity Search

- ***Implementation***

  Compare the query with all the sequences in the DB and return the k most similar sequences to the query.

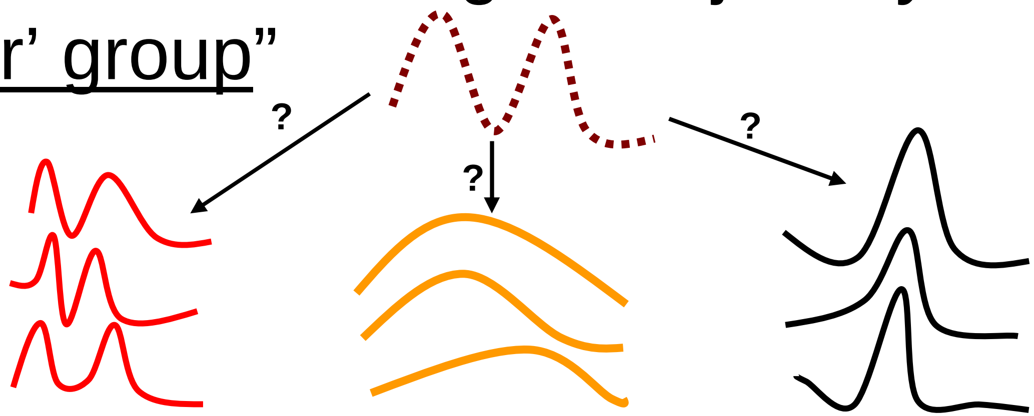


Spatio-Temporal Similarity Search

Having a notion of similarity allows us to perform:

- **Clustering**: “Place trajectories in ‘similar’ groups”

- **Classification**: “Assign a trajectory to the most <u>‘similar’ group</u>”



Presentation Outline

- Definitions and Context
- Overview of Trajectory Similarity Measures
  - Euclidean Matching
  - DTW Matching
  - LCSS Matching
  - Upper Bounding LCSS Matching
- Distributed Spatio-Temporal Similarity Search
  - The UB-K Algorithm
  - The UBLB-K Algorithm
  - Experimentation
- Distributed Top-K Algorithms
  - Definitions
  - The TJA Algorithm
- Conclusions

Trajectory Similarity Measures

A. Euclidean Matching

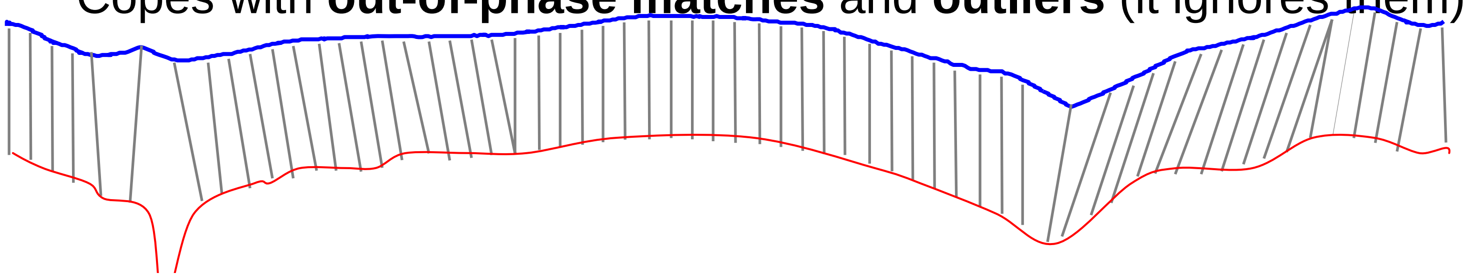
The trajectories are matched 1:1

B. Dynamic Time Warping Matching

Copes with **out-of-phase** matches (using a warping windows)

Longest Common SubSequence Matching

Copes with **out-of-phase matches** and **outliers** (it ignores them)



Euclidean Distance

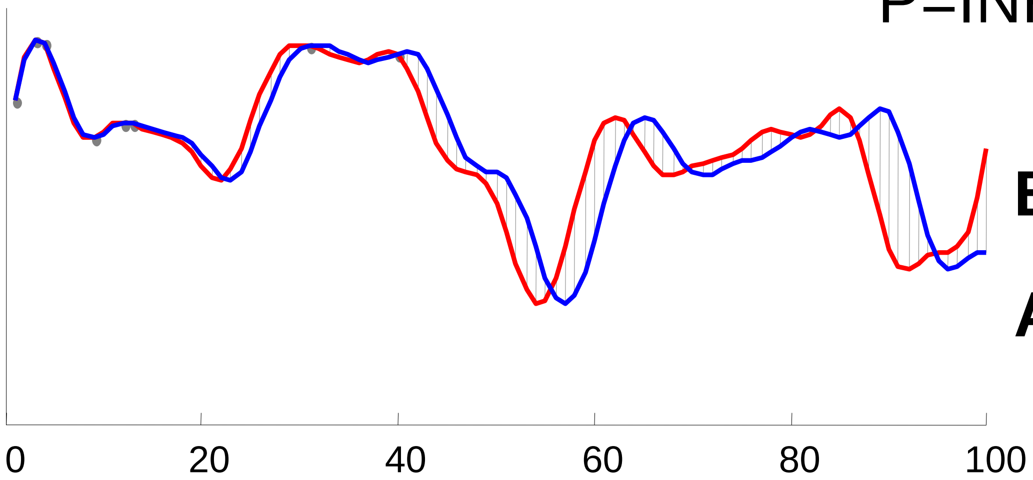
- *Most widely used distance measure*
- *Defines (dis-)similarity between sequences A and B as (1D case):*

$$L_p = \left(\sum_{i=1}^{n} |a[i] - b[i]|^p\right)^{1/p}$$

P=1 Manhattan Distance

P=2 Euclidean Distance

P=INF Chebyshev Distance



B={b1,b2,…,bn}

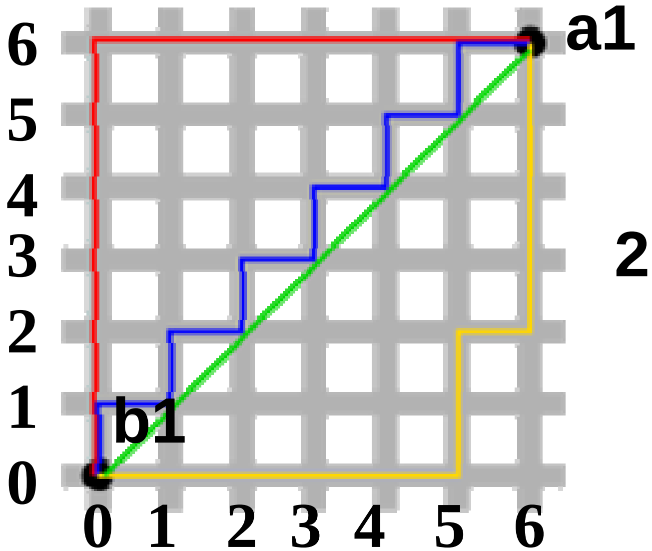
A={a1,a2,…,an}

Euclidean Distance

- ***Euclidean vs. Manhattan distance:***

  *- Euclidean Distance (using Pythagoras theorem) is 6 x √2 = **8.48 points**): Diagonal **Green** line*

  *- Manhattan (city-block) Distance (**12 points**): Red, Blue, and Yellow lines*



2-Dimensional Scenario

Disadvantages of Lp-norms

- ***Disadvantage 1:*** *Not flexible to* ***out-of-phase*** *matching (i.e., temporal distortions)*
  - *e.g., Compare the following 1-dim sequences:*

    A={1112234567}

    B={1112223456}

    Distance = 9
  - *Green Lines indicate successful matching, while red dots indicate an increase in distance.*
- ***Disadvantage 2:*** *Not flexible to* ***outliers*** *(spatial distortions).*

    A={1111191111}

    B={1111101111}

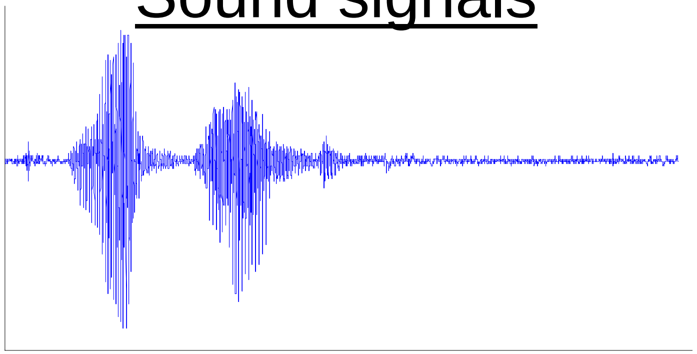
    Distance = 9

Many studies show that the Euclidean Distance Error rate might be as high as **~30%!**

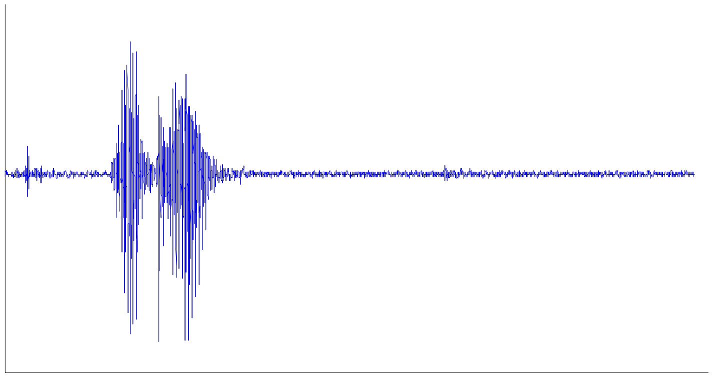
Dynamic Time-Warping

Flexible matching in time: *Used in speech recognition for matching words spoken at different speeds (in voice recognition systems)*

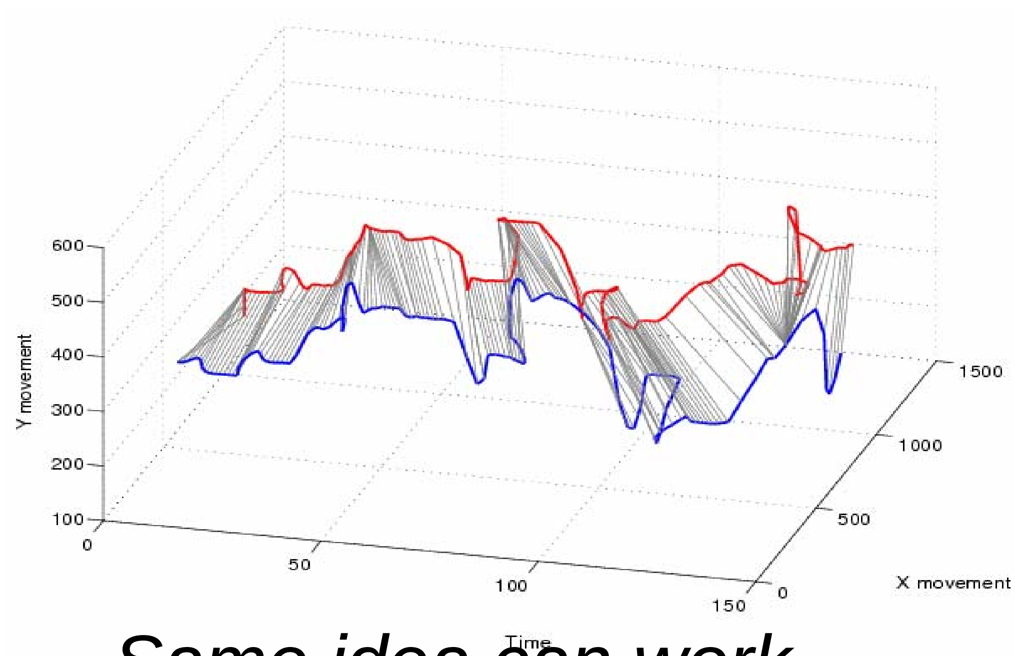
Sound signals



----Mat-lab--------------------------



---Maat--llaabb-------------------



Same idea can work equally well for generic spatio-temporal data…

Dynamic Time-Warping

How does it work?

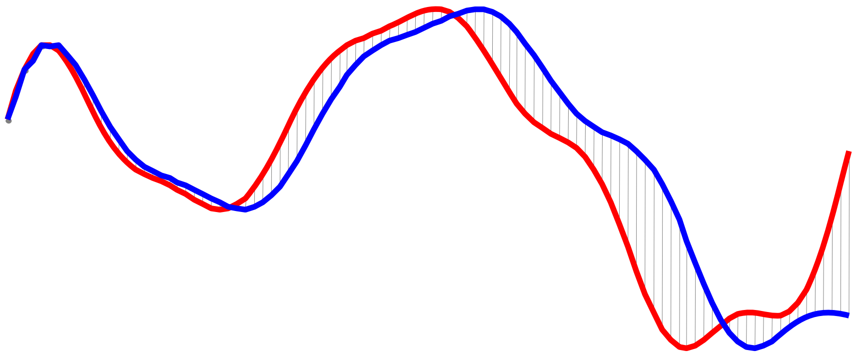
The intuition is that we span the matching of an element X by several positions after X.

Euclidean distance

A1 = [1, 1, 2, 2]

d = 1

A2 = [1, 2, 2, 2]



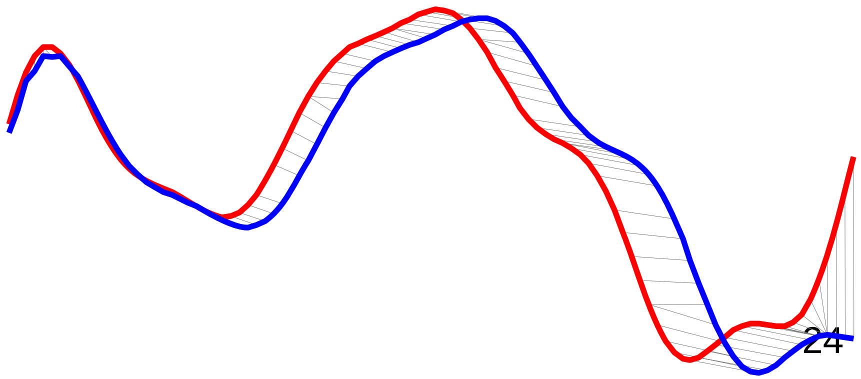
Euclidean: One-to-one alignment

DTW distance

A1 = [1, 1, 2, 2]

d = 0

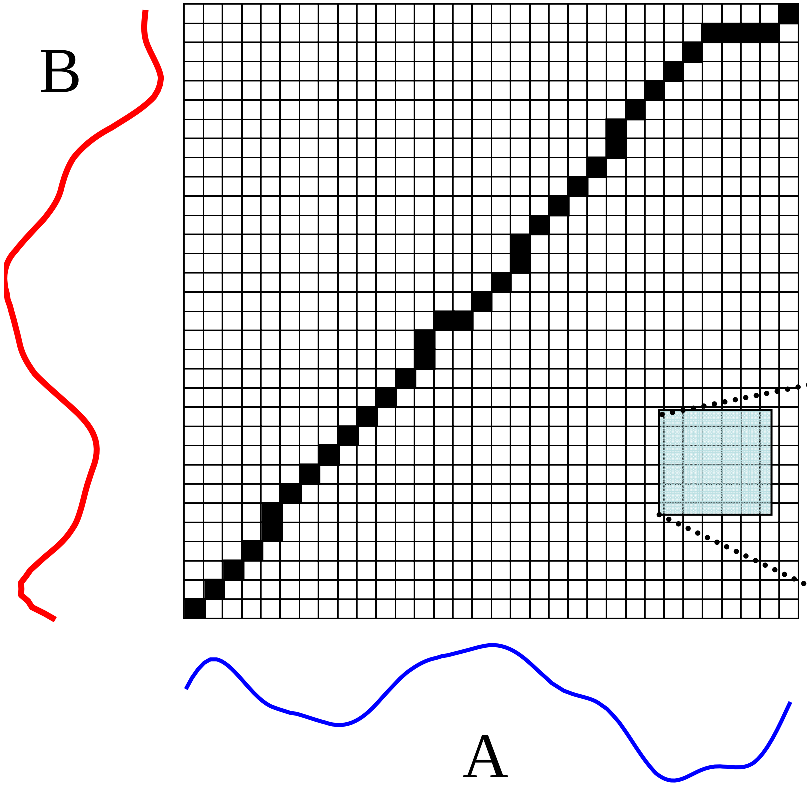
A2 = [1, 2, 2, 2]



DTW: One-to-many alignment

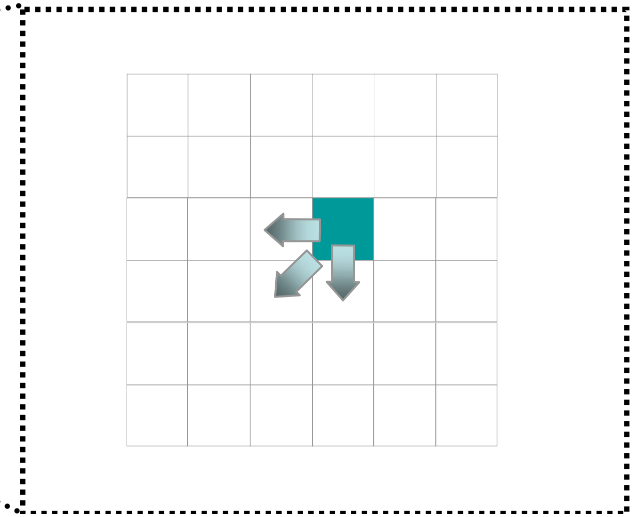
Dynamic Time-Warping

- *Implemented with dynamic programming* (i.e., we exploit overlapping sub-problems) *in **O(|A|*|B|).***
  - *Create an array that stores all solutions for all possible subsequences.*



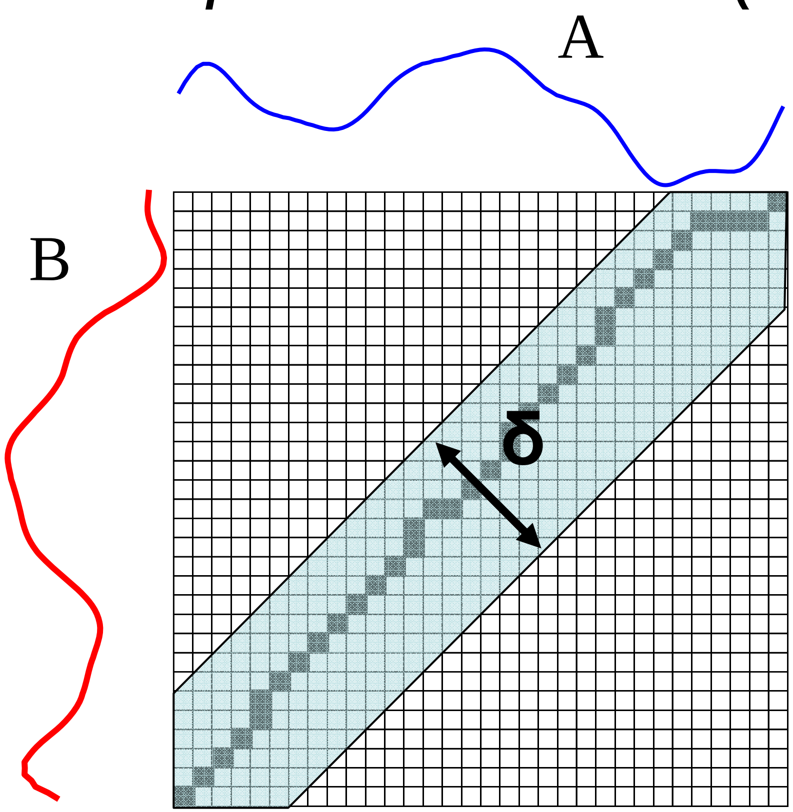
Recursive Definition

$$L[i,j] = LpNorm(A_i, B_j) + \min\{ L(i-1, j-1),\ L(i-1, j),\ L(i, j-1) \}$$

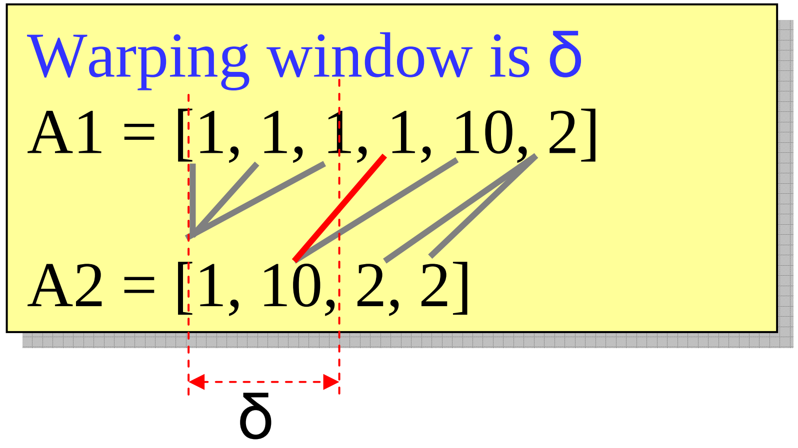


Dynamic Time-Warping

*The **O(|A|*|B|)** time complexity can be reduced to **O(δ*min(|A|,|B|))** by restricting the warping path to a temporal window **δ** (see LCSS for more details).*

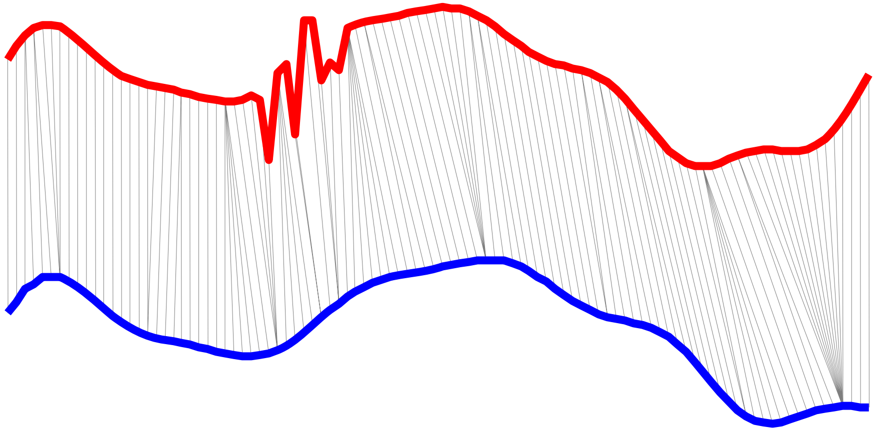


We will now only fill the highlighted portion of the Dynamic Programming matrix



Dynamic Time-Warping

- Studies have shown that **warping window δ=10%** is adequate to achieve high degrees of matching accuracy.
- The **Disadvantages** of DTW:
  - **All points are matched (including outliers)**
  - **Outliers** can distort distance



Longest Common Subsequence

- The Longest Common SubSequence (LCSS) is an algorithm that is extensively utilized in text similarity search, but is equivalently applicable in Spatio-Temporal Similarity Search!

- **Example:**
  - String: CGATAATTGAGA
  - **Substring (contiguous)**: CGA
  - **SubSequence (not necessarily contiguous):** AAGAA

- **Longest Common Subsequence:** Given two strings A and B, find the longest string S that is a subsequence of both A and B;

Longest Common Subsequence

- Find the LCSS of the following 1D-trajectory

A = 3, **2, 5**, 7, **4,** 8, 10, **7**

B = **2, 5, 4, 7,** 3, 10, 8, 6

LCSS = 2, 5, 4, 7

- **The value of LCSS is unbounded:** it depends on the length of the compared sequences.
- To normalize it in order to support sequences of variable length we can define the **LCSS distance**:
- LCSS Distance between two trajectories

dist(A, B) = 1 – LCSS(A,B)/min(|A|,|B|)

e.g. in our example dist (A,B) = 1 – 4/8 = 0.5

LCSS Implementation

- Implemented with a similar Dynamic Programming Algorithm (i.e., we exploit overlapping subproblems) as DTW but with a different recursive definition:

$$LCSS(A,B)=\begin{cases} 0 & \text{, If A or B is empty} \\ 1+LCSS(Tail(A),Tail(B)) & \text{, If Head[A]=Head[B]} \\ \max(LCSS(Tail(A),B),LCSS(A,Tail(B))) & \text{, otherwise} \end{cases}$$

TAIL Head

A = 3, **2, 5**, 7, **4,** 8, 10, **6**

B = **2, 5, 4, 7,** 3, 10, 8, 6

LCSS Implementation

Phase 1: Construct DP Table

int A[] = {3,2,5,7,4,8,10,7};

int B[] = {2,5,4,7,3,10,8,6};

int L[n+1][m+1]; // DP Table

```
// Initialize first column and row to assist the DP Table
for (i=0;i<n+1;i++)    L[i][0] = 0;
for (j=0;j<m+1;j++)    L[0][j] = 0;

for (i=1;i<n+1;i++) {
    for (j=1;j<m+1;j++) {
        if (A[i-1] == B[j-1]) {
                L[i][j] = L[i-1][j-1] + 1;
        } else {
                L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }
```

DP Table L[][]

m

A \ B		2	5	4	7	3	10	8	6
	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	1	1
2	0	1	1	1	1	1	1	1	1
5	0	1	2	2	2	2	2	2	2
7	0	1	2	2	3	3	3	3	3
4	0	1	2	3	3	3	3	3	3
8	0	1	2	3	3	3	3	4	4
10	0	1	2	3	3	3	4	4	4
7	0	1	2	3	4	4	4	4	4

n

Solution

LCSS(A,B) = 4

Running Time O(|A|*|B|)

LCSS Implementation

Phase 2: Construct LCSS Path

Beginning at L[n-1][m-1] move backwards until you reach the left or top boundary

```
i = n;        j = m;
while (1) {
    // Boundary was reached - break
    if ((i == 0) || (j == 0))   break;

    // Match
    if (A[i-1] == B[j-1]) {
            printf("%d,", A[i-1]);
            // Move to L[i-1][j-1] in next round
            i--; j--;
    } else {
            // Move to max { L[i][j-1],L[i-1][j] } in next round
            if (L[i][j-1] >= L[i-1][j])   j--;
            else          i--;
    }
}
```

Running Time O(|A|+|B|)

DP Table L[][]

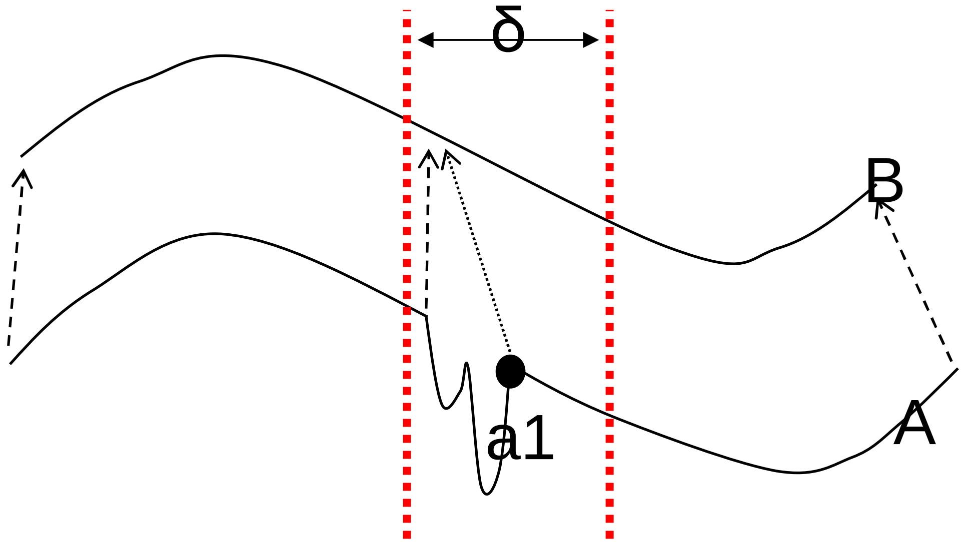
		2	5	4	7	3	10	8	6
	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	1	1	1
2	0	1	1	1	1	1	1	1	1
5	0	1	2	2	2	2	2	2	2
7	0	1	2	2	3	3	3	3	3
4	0	1	2	3	3	3	3	3	3
8	0	1	2	3	3	3	3	4	4
10	0	1	2	3	3	3	4	4	4
7	0	1	2	3	4	4	4	4	4

m,n

LCSS: 7,4,5,2

Speeding up LCSS Computation

- The DP algorithm requires **O(|*A*|*|*B*|)** time.
- However we can compute it in **O(*δ*(|*A*|+|*B*|))** time, similarly to DTW, if we limit the matching within a time window of **δ**.
- Example where δ=2 positions



		2	5	4	7	3	10	8	6
	0	0	0	0	0	0	0	0	0
3	0	0	0						
2	0	1	1	1					
5	0		2	2	2				
7	0			2	3	3			
4	0				3	3	3		
8	0					3	3	4	
10	0						4	4	4
7	0							4	4

LCSS: 10,7,5,2

δ=2

* Finding Similar Time Series, G. Das, D. Gunopulos, H. Mannila, In PKDD 1997.

LCSS 2D Computation

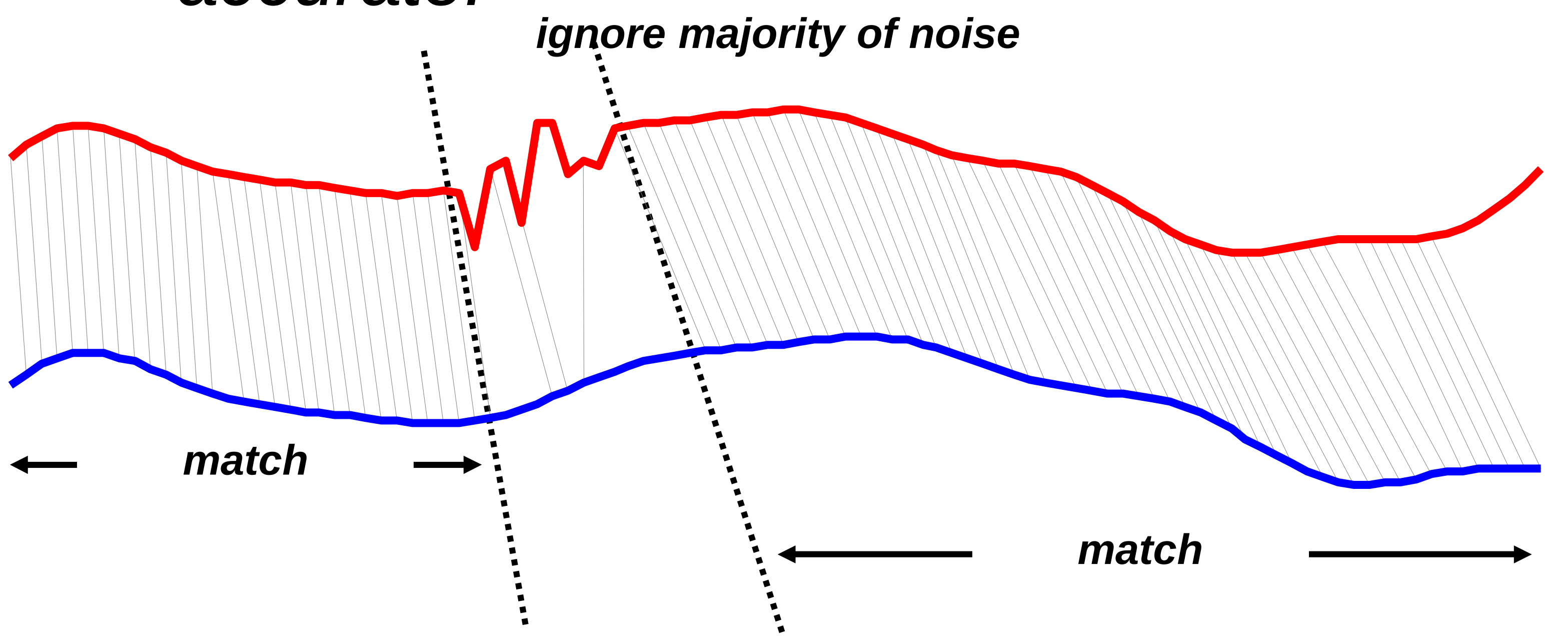
- The LCSS concept can easily be extended to support 2D (or higher dimensional) spatio-temporal data.
- The following is an adaptation to the 2D case, where the computation is limited in time **(by window δ)** and space **(by window ε)**

$$LCSS(A,B) = \begin{cases} 0, & \text{if } A \text{ or } B \text{ is empty} \\ 1 + LCSS(Tail(A), Tail(B)), & \\ & \text{if } \left| a_{i1} - b_{i2} \right| < \varepsilon \text{ and } \left| i_1 - i_2 \right| < \delta \\ max(LCSS(Tail(A), B), LCSS(A, Tail(B)), & \\ & \text{otherwise} \end{cases}$$

Longest Common Subsequence

Advantages of LCSS:

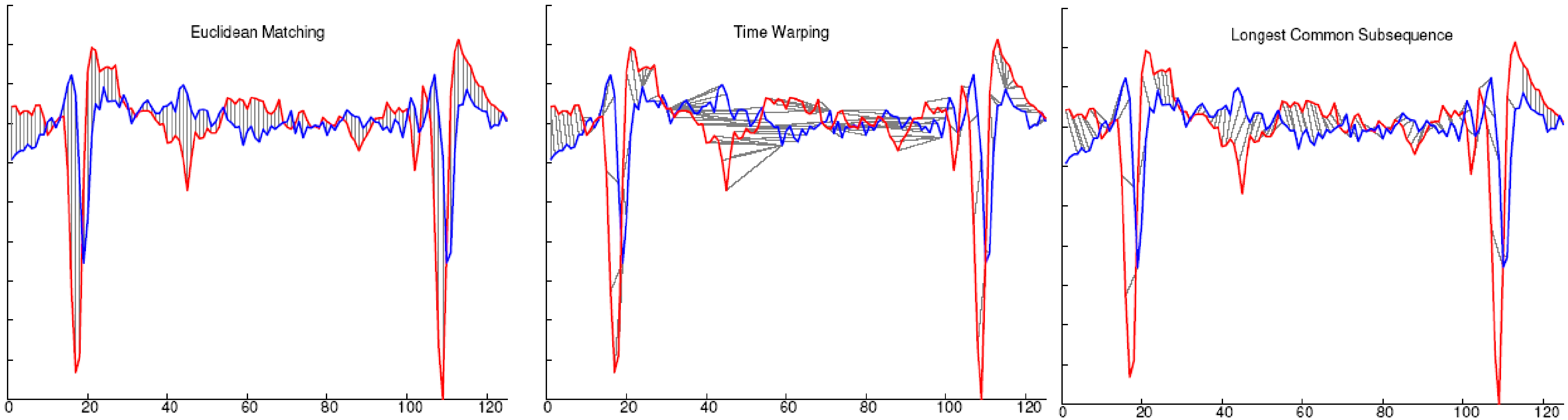
- *Flexible matching in time*
- *Flexible matching in space (ignores outliers)*
- *Thus, the Distance/Similarity is more accurate!*



Summary of Distance Measures

Method	*Complexity**	*Elastic Matching (out-of-phase)*	*1:1 Matching*	*Noise Robustness (outliers)*
Euclidean	*O(n)*	×	✓	×
DTW	*O(n*δ)*	✓	×	×
LCSS	*O(n*δ)*	✓	✓	✓

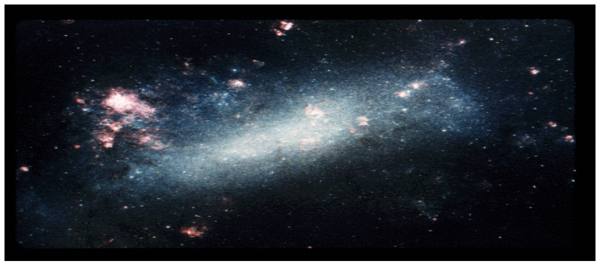
* Assuming that trajectories have the same length



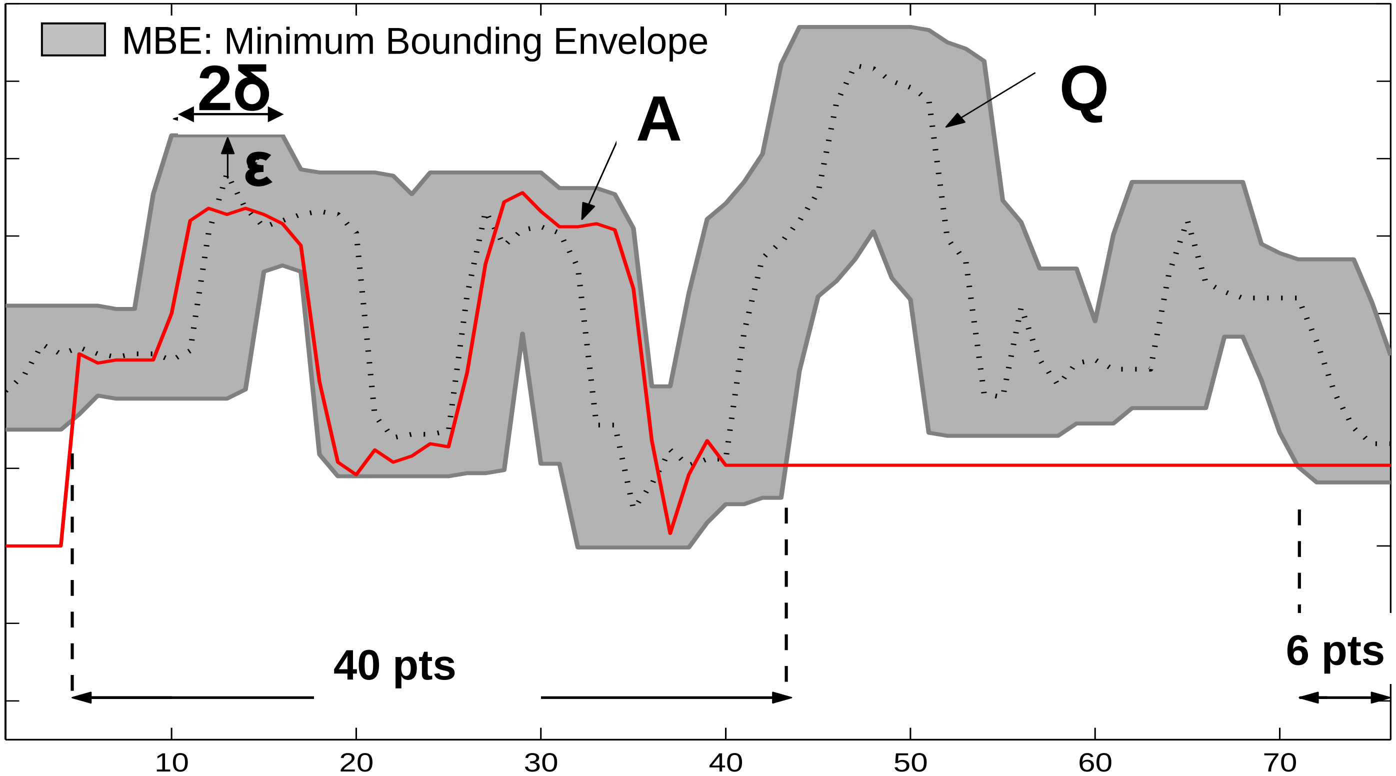
Any disadvantage with LCSS?

Speeding Up LCSS

- **O(δ*n)** is not always very efficient!
- Consider a **space observation system** that records the trajectories for **millions** of stars**.**
- To compare 1 trajectory against the trajectories of all stars it takes **O(δ*n*trajectories)** time .
- **Solution:** Upper bound the LCSS matching using a Minimum Bounding Envelope
  - Allows the computation of similarity between trajectories in O(n*trajectories) time!



Upper Bounding LCSS*



Theorem: $LCSS_{\delta,\varepsilon}(Q, A) \leq LCSS_{\delta,\varepsilon}(MBE(Q), A)$

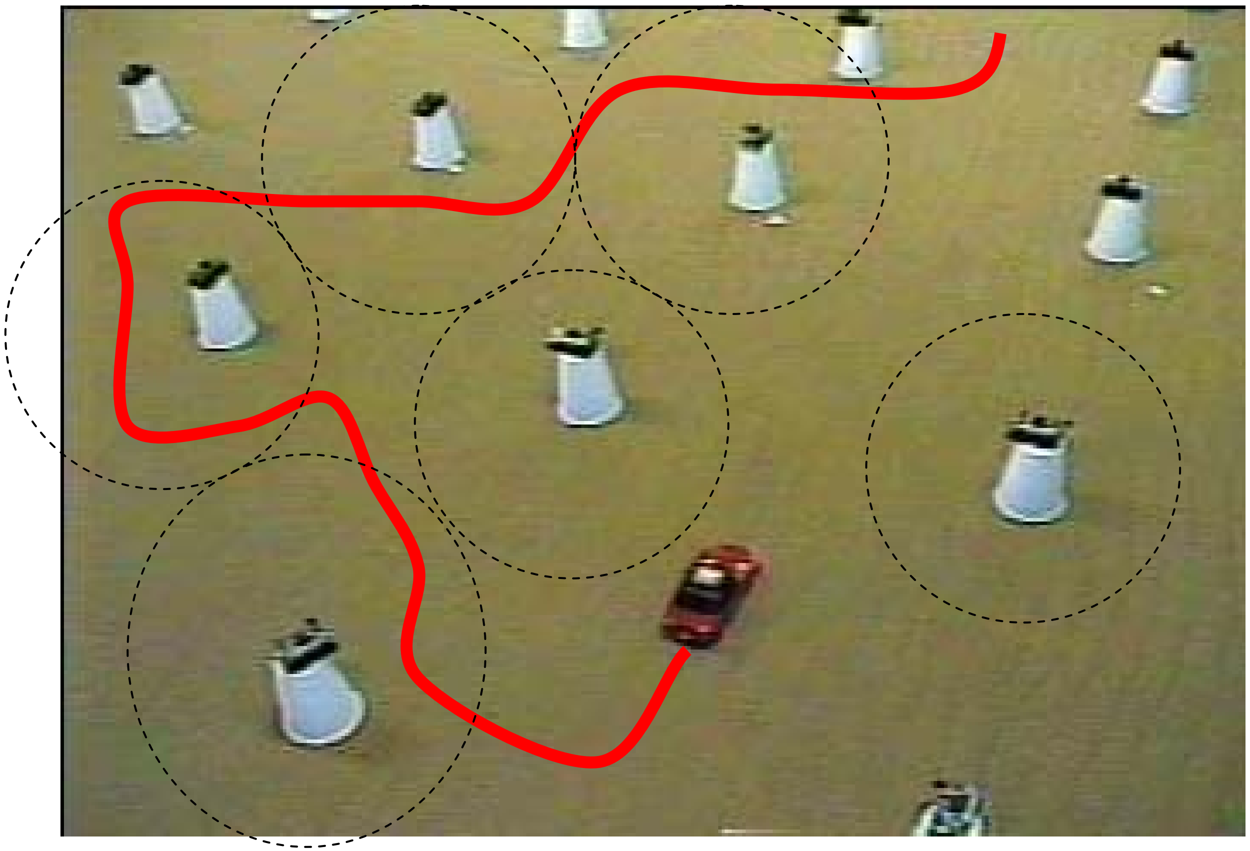
* *Indexing multi-dimensional time-series with support for multiple distance measures*, M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, E. Keogh, In KDD 2003.

Presentation Outline

- Definitions and Context
- Overview of Trajectory Similarity Measures
  - Euclidean Matching
  - DTW Matching
  - LCSS Matching
  - Upper Bounding LCSS Matching
- **Distributed Spatio-Temporal Similarity Search**
  - **Definitions**
  - **The UB-K and UBLB-K Algorithms**
  - **Experimentation**
- Distributed Top-K Algorithms
  - Definitions
  - The TJA Algorithm
- Conclusions

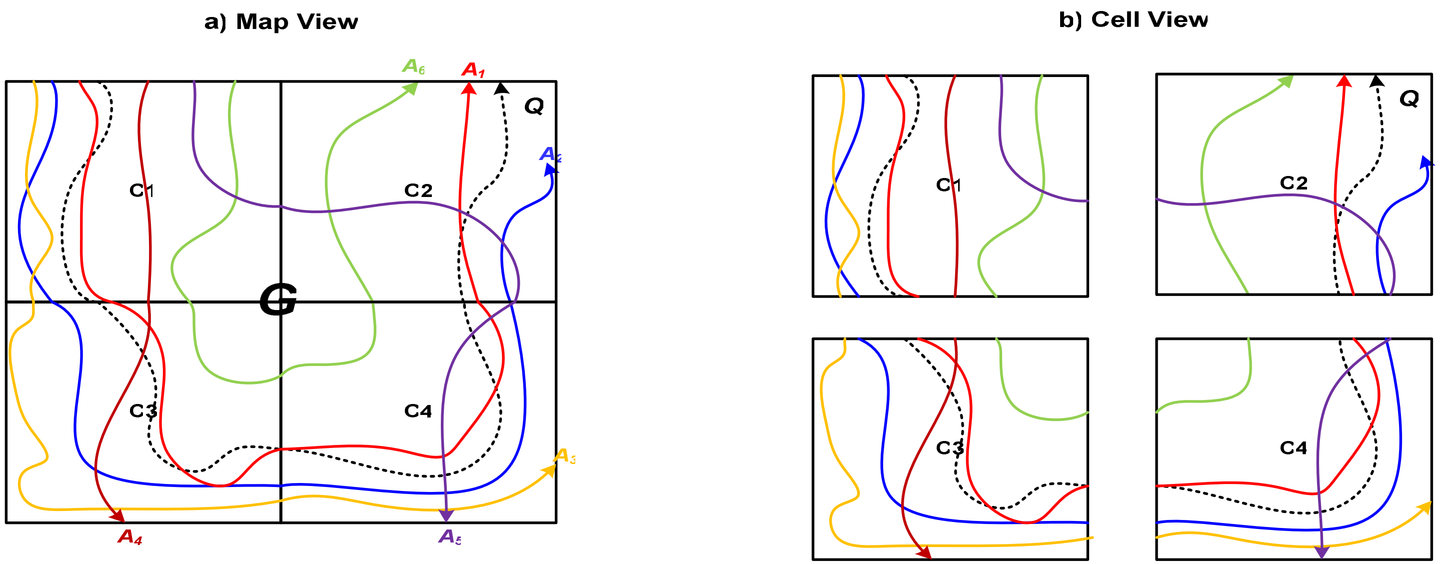
Distributed Spatio-Temporal Data

- Recall that trajectories are segmented across n distributed cells.



System Model

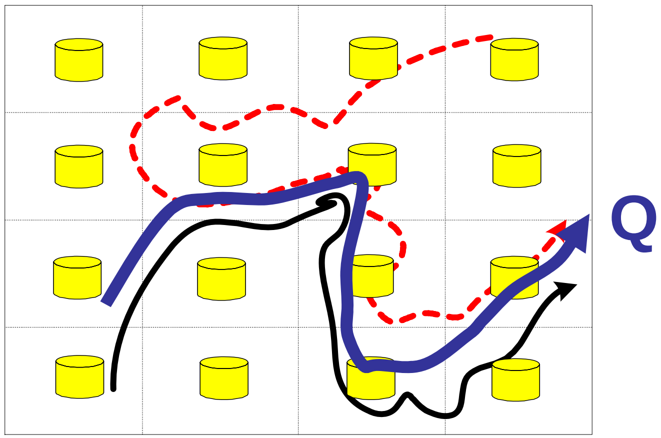
- Assume a geographic region **G** segmented into **n** cells {C1,C2,C3,C4}
- Also assume **m** objects moving in G.
- Each cell has a device that records the spatial coordinated of each passing object.
- The coordinates remain locally at each cell



Problem Definition

*Given a distributed repository of **trajectories** coined **DATA**, retrieve the K most similar trajectories to a query trajectory **Q**.*

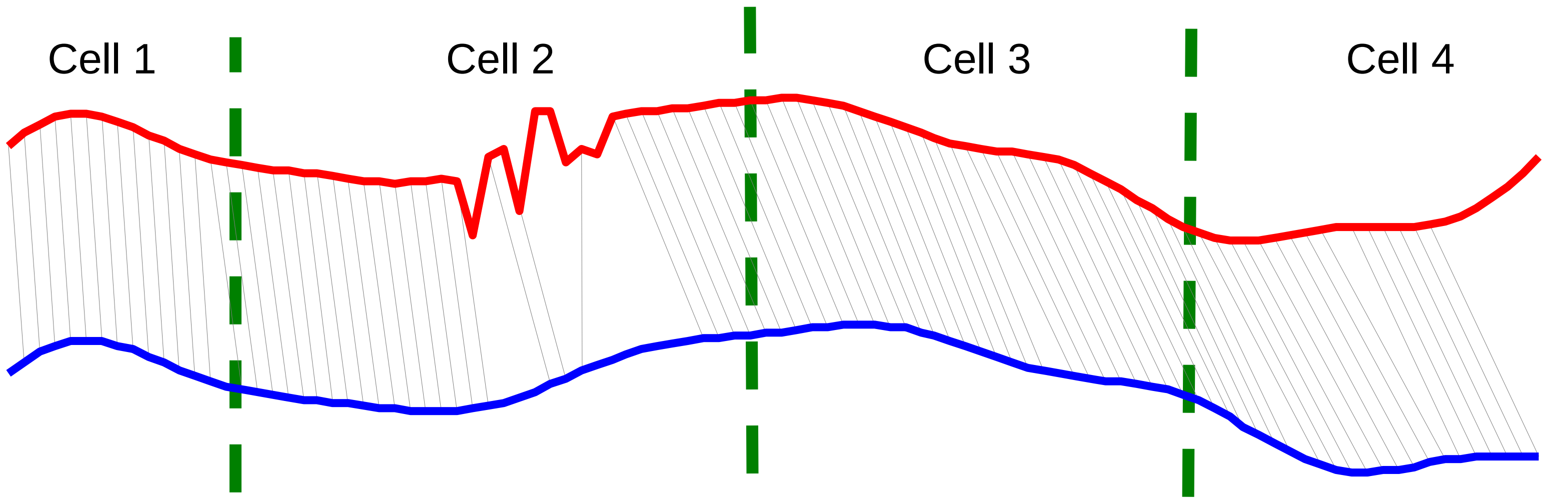
DATA:



- **Challenge:** The collection of all trajectories to a centralized point for storage and analysis is expensive!

Distributed LCSS

- Since trajectories are segmented over n cells the computation of LCSS now becomes difficult!
  - The matching might happen at the **boundary** of neighboring cells.
  - In LCSS matching occurs **sequentially**.



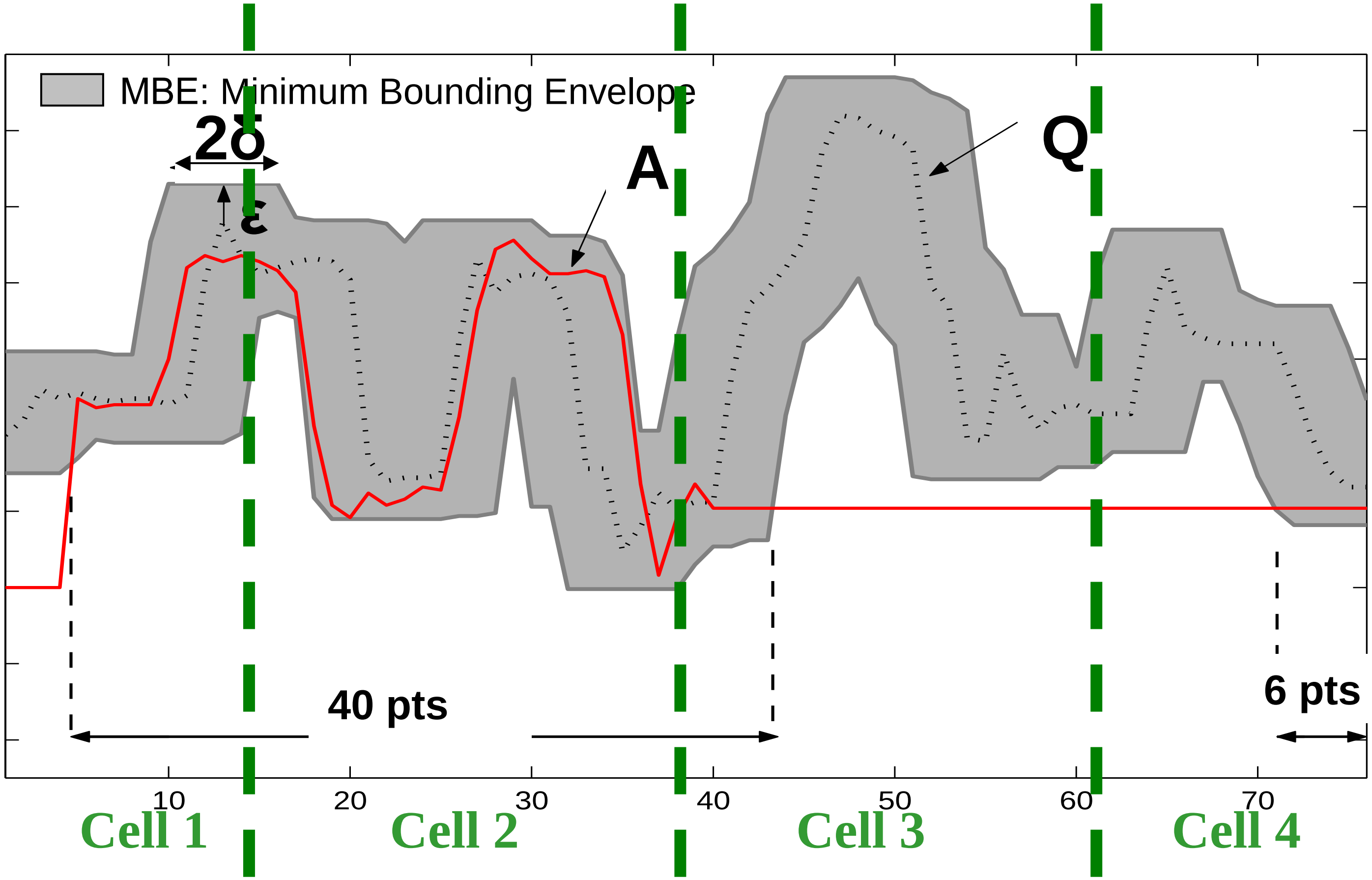
Distributed LCSS

- Instead of computing the LCSS directly, we measure partial lower bounds **(DLB_LCSS)** and partial upper bound (**DUB_LCSS)**
  - i.e., instead of **LCSS(A0,Q)=20** we compute **LCSS(A0,Q)=[15..25]**

- We then **process** these scores using some novel algorithms we will present next and **derive** the **K** most similar trajectories to **Q**.

- Lets first see how to construct these scores…

Distributed Upper Bound on LCSS



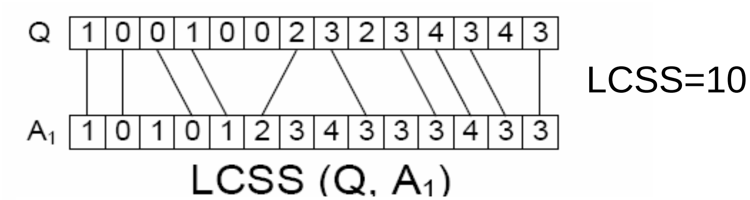
DUB_LCSS:

$$\boxed{\sum_{j=1}^{n} LCSS_{\delta,\varepsilon}(MBE(Q), A_{ij}) \geq LCSS_{\delta,\varepsilon}(Q, A_i)}$$

Distributed Lower Bound on LCSS

- We execute LCSS(Q, Ai) locally at each cell without extending the matching beyond:
  - The **Spatial boundary of the cell**
  - The **Temporal boundary of the local A_{ix}.**

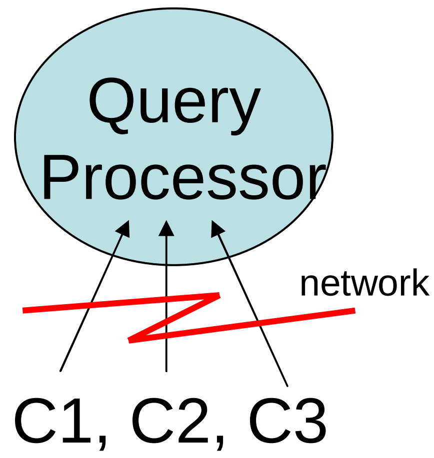
- At the end we add the partial lower bounds and construct DLB_LCSS:



$$\boxed{\sum_{j=1}^{n} LCSS_{\delta,\varepsilon}(Q, Aij) \le LCSS_{\delta,\varepsilon}(Q, Ai)}$$

The METADATA table

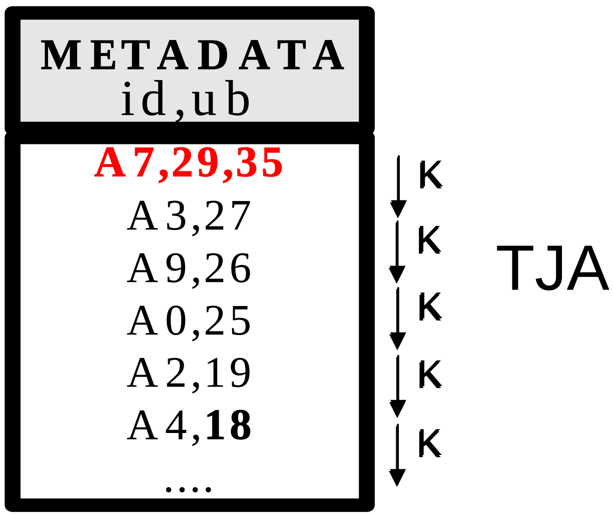
- **METADATA** Table: A vector that contains bounds on the similarity between Q and trajectories Ai
- **Problem:** Bounds have to be transferred over an expensive network



c1 id,lb,ub		c2 id,lb,ub		c3 id,lb,ub		METADATA id,ub
A2,4,6		A4,3,5		A4,1,3		**A7,29,35**
A0,6,8		A2,4,6		A0,8,10		A3,27
A4,8,10		A0,5,7		A2,5,7	=	A9,26
A7,7,9	+	A3,4,6		A9,5,7		A0,25
A3,9,11		A9,8,10	+	A3,8,10		A2,19
A9,7,9		**A7,11,13**		**A7,11,13**		A4,**18**
....		....		....		....

The METADATA table

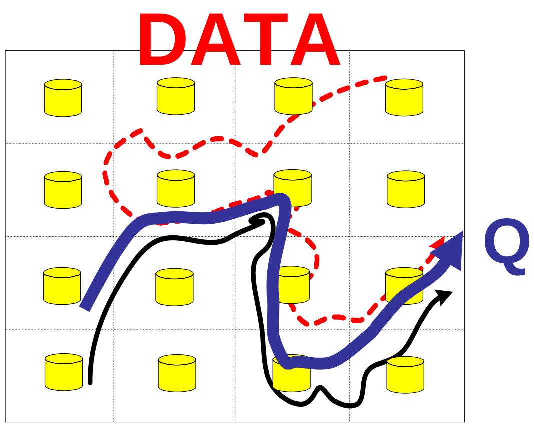
- **Option A:** Transfer all bounds towards QP and then join the columns.
  - Too expensive (e.g., Millions of trajectories)
- **Option B:** Construct the METADATA table incrementally using a distributed top-k algorithm
  - Much Cheaper! - TJA and TPUT algorithms will be described at the end!



The UB-K Algorithm

- An iterative algorithm we developed to find the K most similar trajectories to Q.

- Main Idea: It utilizes the upper bounds in the **METADATA** table to minimize the transfer of **DATA.**



METADATA id,ub
A7,29,35
A3,27
A9,26
A0,25
A2,19
A4,**18**
....

UB-K Execution

Query: Find the K=2 most similar trajectories to Q

Retrieve the sequences A4, A2

Stop if TJA Kth LCSS >= TJA Λth UB

Λ=3

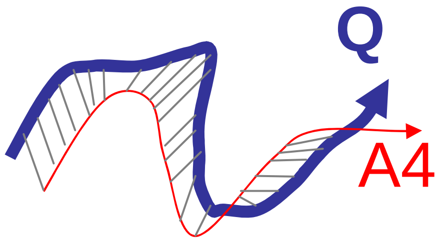
Λ=5

METADATA id,ub
A4,30
A2,27
A0,25
A3,20
A9,18
A7,12
....

LB

DATA
A4,23
A2,22
A0,16
A3,18
A9,15
A7,10
....

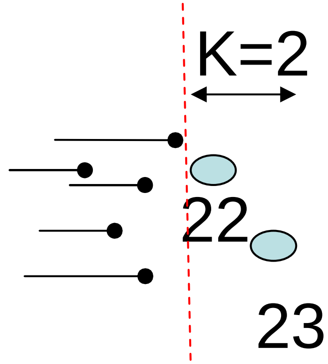
EXACT



LCSS(Q,A4)=23

=>Kth LCSS

≥?



The UBLB-K Algorithm

- Also an iterative algorithm with the same objectives as UB-K
- **Differences:**
  - Utilizes the distributed LCSS **upper-bound (DUB_LCSS)** and **lower-bound (DLB_LCSS)**
  - Transfers the **DATA** in a final **bulk** step rather than incrementally (by utilizing the **LBs**)

UBLB-K Execution

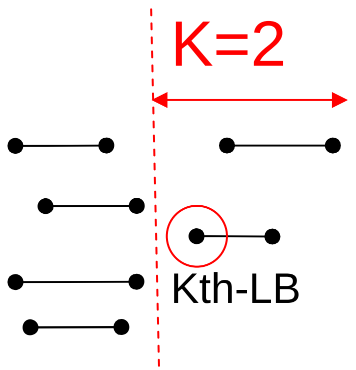
Query: Find the K=2 most similar trajectories to Q

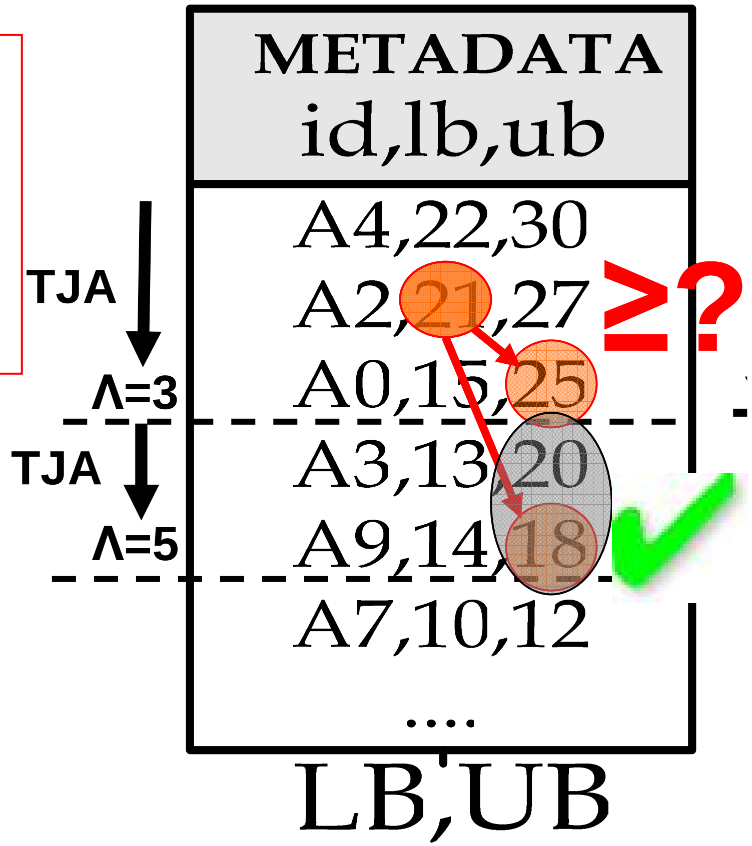
Stop if

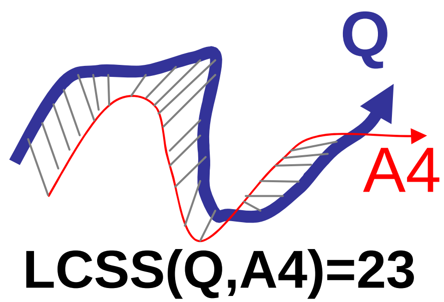
Kth LB

>=

Λth UB



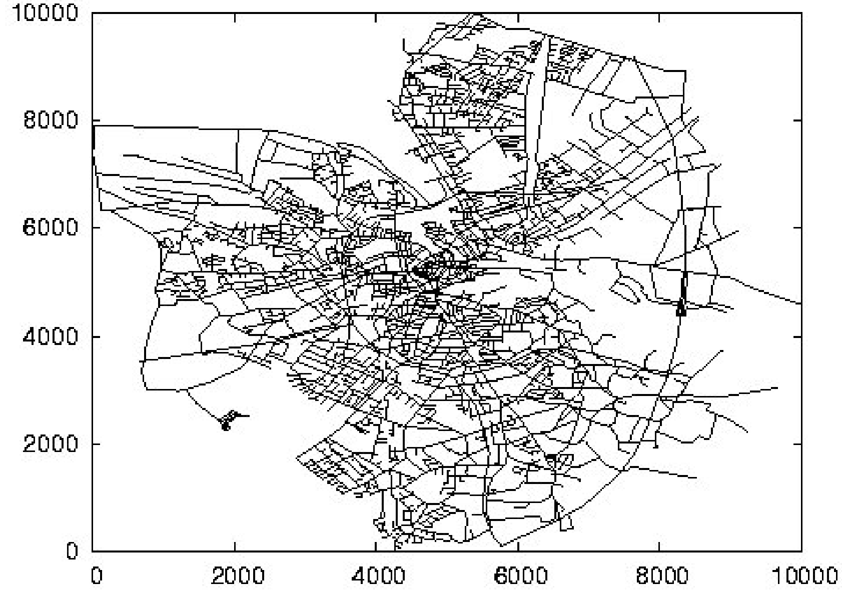




Note: Since the Kth LB 21 >= 20, anything below this UB is not retrieved in the final phase!

Experimental Evaluation

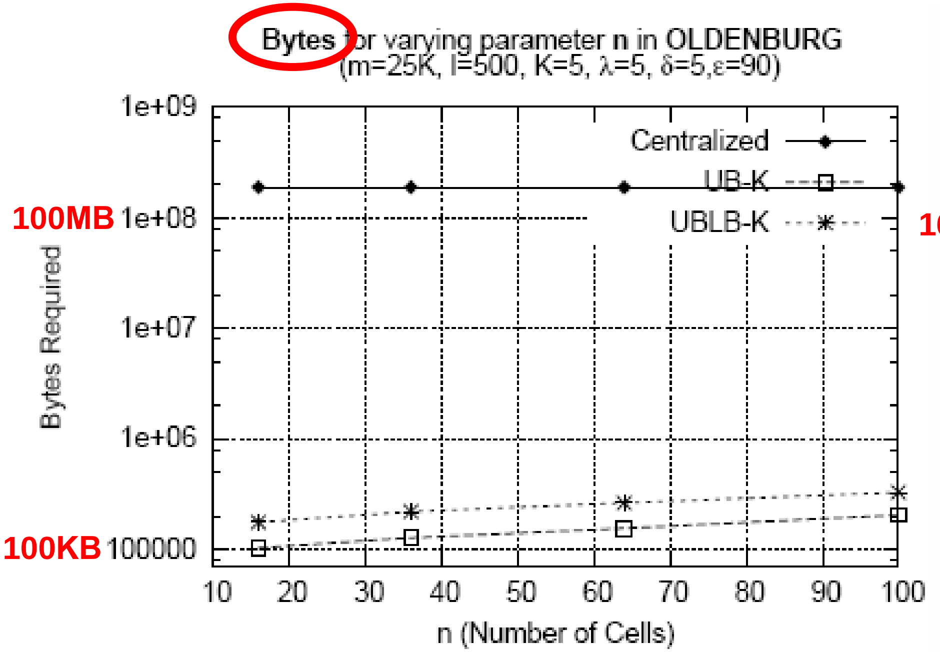
- **Comparison System**
  - Centralized
  - UB-K
  - UBLB-K
- **Evaluation Metrics**
  - Bytes
  - Response Time

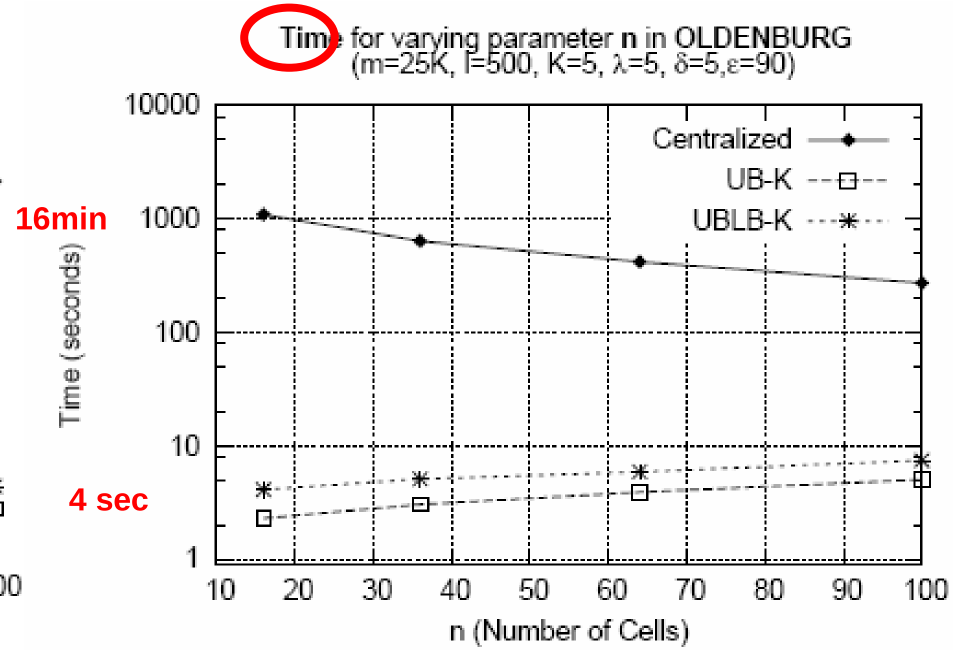


- **Data**
  - 25,000 trajectories generated over the road network of the **Oldenburg** city using the *Network Based Generator of Moving Objects*.*

* Brinkhoff T., "A Framework for Generating Network-Based Moving Objects". In GeoInformatica,6(2), 2002.

Performance Evaluation





- Remarks
  - **Bytes:** UBK/UBLBK transfers 2-3 orders of magnitudes fewer bytes than Centralized.
    - Also, UBK completes in 1-3 iterations while UBLBK requires 2-6 iterations (this is due to the LBs, UBs).
  - **Time:** UBK/UBLBK 2 orders of magnitude less time.

Presentation Outline

- Definitions and Context
- Overview of Trajectory Similarity Measures
  - Euclidean Matching
  - DTW Matching
  - LCSS Matching
  - Upper Bounding LCSS Matching
- Distributed Spatio-Temporal Similarity Search
  - Definitions
  - The UB-K and UBLB-K Algorithms
  - Experimentation
- **Distributed Top-K Algorithms**
  - **Definitions**
  - **The TJA Algorithm**
- Conclusions

Definitions

Top-K Query (Q)

*Given a **database D** of **n** objects, a **scoring function** (according to which we rank the objects in D) and the number of **expected answers K**, a Top-K query Q returns the K objects with the highest score (rank) in D.*

Objective:

Trade # of answers with the query execution cost, i.e.,

- Return less results (K<<n objects)
- …but minimize the **cost that is** associated with the retrieval of the answer set (i.e., disk I/Os, network I/Os, CPU etc)

Definitions

The Scoring Table

An m-by-n matrix of scores expressing the similarity of Q to all objects in D (for all attributes).

In order to find the K highest-ranked answers we have to compute **Score(o_i)** *for all objects (requires O(m*n) time).*

$$Score(o_i) = \sum_{j=1}^{n} sim(q_j, o_{ij})$$

Score

trajectoryID

m trajectories

c1	c2	c3	c4	c5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35

n cells

Threshold Join Algorithm (TJA)

- TJA is our **3-phase algorithm** that optimizes top-k query execution in distributed (hierarchical) environments.

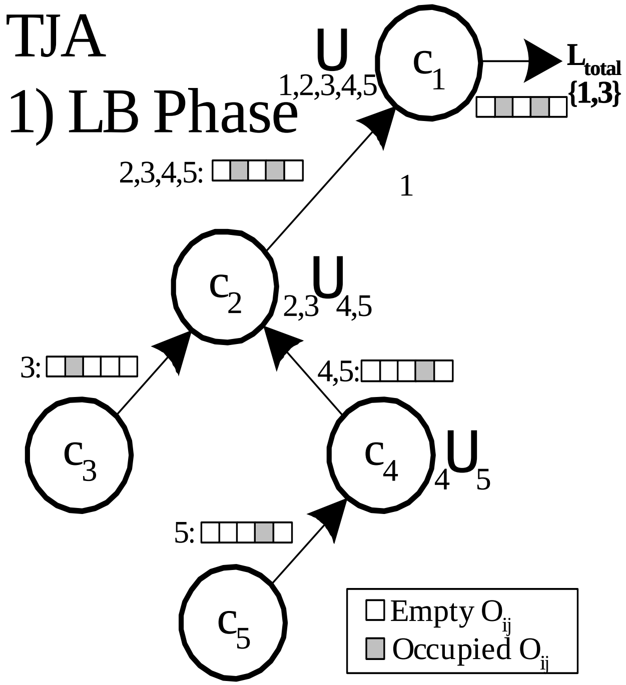
- **Advantage:**
  - **It usually** completes in **2 phases.**
  - **It never completes** in more than **3 phases** (LB Phase, HJ Phase and CL Phase)
  - It is therefore highly appropriate for distributed environments

"The Threshold Join Algorithm for Top-k Queries in Distributed Sensor Networks", D. Zeinalipour-Yazti et. al, Proceedings of the 2nd international workshop on Data management for sensor networks DMSN (VLDB'2005), Trondheim, Norway, ACM Press; Vol. 96, 2005.

Step 1 - LB (Lower Bound) Phase

- Each node sends its K highest objectIDs

- Each intermediate node performs a **union** of the received results (defined as **τ**):

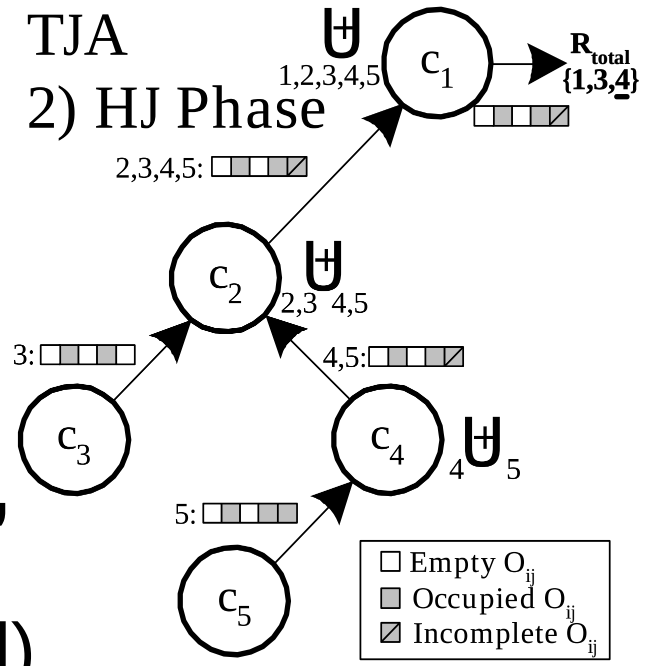


c1	c2	c3	c4	c5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35

LB
T={o3, o1}   **Query: TOP-1**

Step 2 – HJ (Hierarchical Join) Phase

- Disseminate **τ** to all nodes
- Each node sends back everything with score above all objectIDs in **τ**.
- Before sending the objects, each node tags as **incomplete,** scores that could not be computed exactly (upper bound)



c1	c2	c3	c4	c5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54
o4, 44	o2, 01	o0, 01	o4,19	o0, 35

HJ	
o3, 405	Complete
o1, 363	Complete
o4',354	Incomplete

Step 3 – CL (Cleanup) Phase

Have we found K objects with a complete score?

Yes: The answer has been found!

No: Find the *complete score* for each incomplete object (all in a single batch phase)

- **CL ensures correctness!**
- **This phase is rarely required in practice.**

c1	c2	c3	c4	c5
o3, 99	o1, 91	o1, 92	o3, 74	o3, 67
o1, 66	o3, 90	o3, 75	o1, 56	o4, 67
o0, 63	o0, 61	o4, 70	o2, 56	o1, 58
o2, 48	o4, 07	o2, 16	o0, 28	o2, 54
o4, 44	o2, 01	o0, 01	o4, 19	o0, 35

TOP-5
o3,405
o1, 363
o4, 207
o0, 188
o2, 175

Conclusions

- I have presented the **Spatio-Temporal Similarity Search problem**: find the most similar trajectories to a query Q when the target trajectories are vertically fragmented.

- I have also presented **Distributed Top-K Query Processing algorithms**: find the K highest-ranked answers quickly and efficiently.

- **These algorithms are generic and could be utilized in a variety of contexts!**

Bibliography

- (PAPER) **``Distributed Spatio-Temporal Similarity Search'',** ***D. Zeinalipour-Yazti****, S. Lin, D. Gunopulos, ACM 15th Conference on Information and Knowledge Management, (****ACM CIKM 2006****), November 6-11, Arlington, VA, USA, pp.14-23, August 2006.*
- (PAPER) "**The Threshold Join Algorithm for Top-k Queries in Distributed Sensor Networks",** D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, D. Srivastava , In **DMSN (VLDB'05),** Trondheim, Norway, ACM Series; Vol. 96, Pages: 61-66, 2005.
- (PAPER) **"Efficient top-K query calculation in distributed networks"**, P. Cao, Z. Wang, In PODC, St. John's, Newfoundland, Canada, pp. 206 – 215, 2004.
- (PAPER) ***"Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures",*** *Vlachos, M., Hadjieleftheriou, M., Gunopulos, D. & Keogh. E. (2003). In the 9th* ***ACM SIGKDD*** *International Conference on Knowledge Discovery and Data Mining. August, 2003. Washington, DC, USA. pp 216-225.*
- (PAPER) **Using Dynamic Time Warping to Find Patterns in Time Series.** Donald J. Berndt, James Clifford, In KDD Workshop 1994.
- (PAPER) **Finding Similar Time Series.** G. Das, D. Gunopulos and H. Mannila. In Principles of Data Mining and Knowledge Discovery in Databases (PKDD) 97, Trondheim, Norway.

Bibliography

- ***(TUTORIAL)* "Hands-On Time Series Analysis with Matlab",** *Michalis Vlachos and Spiros Papadimitriou, International Conference of Data-Mining (ICDM), Hong-Kong, 2006*
- **(TUTORIAL)** "Time Series Similarity Measures", D. Gunopulos, G. Das, Tutorial in SIGMOD 2001.
- Other Tutorials by Eamonn Keogh http://www.cs.ucr.edu/~eamonn/tutorials.html

- **(BOOKS)** Jiawei Han and Micheline Kamber

  Data Mining: Concepts and Techniques, 2nd ed.

  The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor Morgan Kaufmann Publishers, March 2006. ISBN 1-55860-901-6

Distributed Spatio-Temporal Similarity Search

Thanks!

Questions?

This presentation is available at the following URL:
http://www.cs.ucy.ac.cy/~dzeina/talks.html

Related Publications available at:
http://www.cs.ucy.ac.cy/~dzeina/publications.html