

Visually driven analysis of movement data by progressive clustering

*Salvatore Rinzivillo, Dino Pedreschi, Mirco Nanni,
Fosca Giannotti, Natalia Andrienko, Gennady Andrienko*

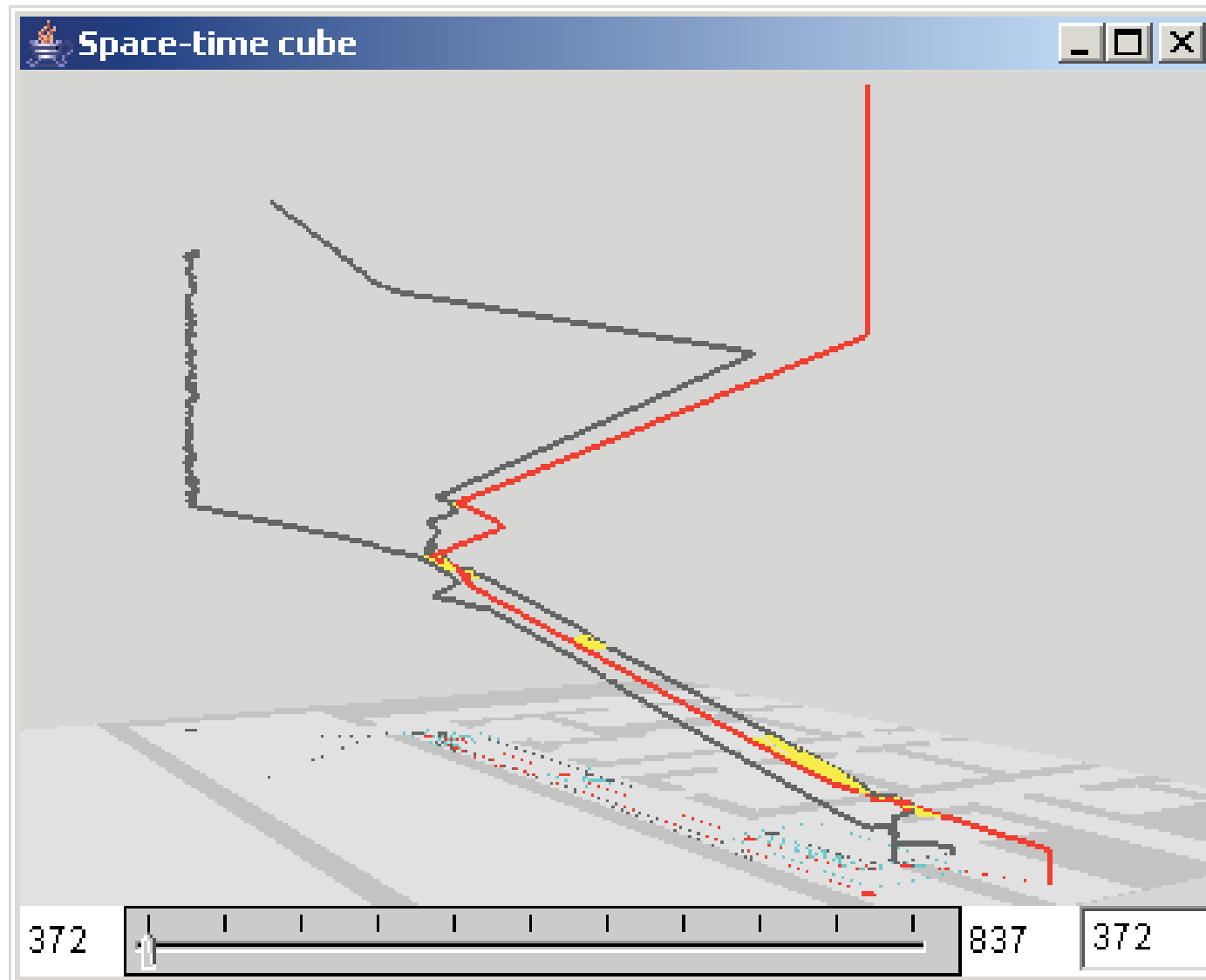
Information Visualization 2008

Eric Shook (eshook2@uiuc.edu)

Outline

- Introduction
- Density-based clustering method (OPTICS)
- Visualizing Cluster Trajectories
- Progressive Clustering Methods
- Case study – Car trajectories in Milan
- Summary
- Conclusions

Space-time cubes



Motivation

- Recent interest, because of availability of large amounts of movement or trajectory data
 - GPS
 - RFID
- Clustering is a good approach to explore and analyze large trajectory data sets
- Visual and interactive techniques play a key role in this analysis and exploration

Combine clustering and visual techniques

Quick review: Clustering methods

- Partition-based
 - K-means
 - K-mediod
- Self-Organizing Map (SOM)
- Dendrogram (Tree-like representation)
 - Top-down
 - Bottom-up
- Density-based
 - OPTICS

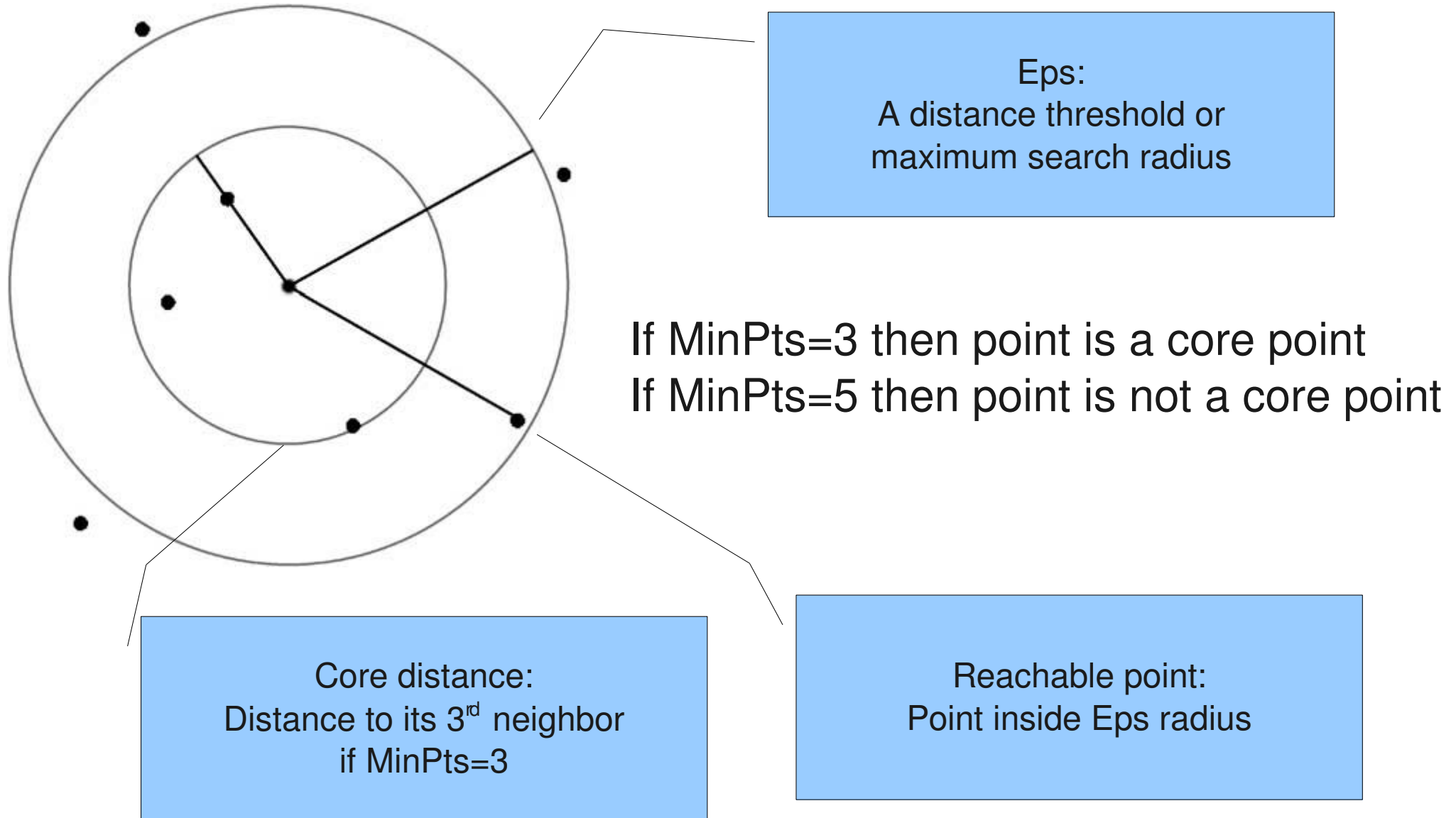
OPTICS algorithm

- Density-based clustering algorithm
 - Robust to noise and outliers – desirable trait when analyzing trajectory data
- DBSCAN family
 - Covered by Dr. Yu in previous lectures
- Has core, reachable, and noise objects

High-level Definitions

- Core object
 - If the number of objects, N , within search radius (Eps) is greater than MinNbs from a given point then that point is a core point
- Reachable object
 - A point that is within search radius (Eps) distance to a core point, but not a core point
- Noise object
 - A point that is not a core or reachable point

Example



OPTICS algorithm pseudo-code

OPTICS (SetOfTrajectories, Eps, MinNbs, Distance_function)

```
for each t in SetOfTrajectories do
  if (not t.isProcessed()) then
    neighbors := Distance.neighbors(t, Eps)
    t.setProcessed(true);
    t.setReachabilityDistance(Infinity);
    t.setCoreDistance(neighbors, Eps, MinNbs);
    emit(t);
    if (t.getCoreDistance() < Infinity) then           // This is a core point ⇒ Expand the cluster
      PriorityQueue.updateOrAddAll(Neighbors);
      while (not PriorityQueue.isEmpty())
        t1 = PriorityQueue.pop();
        t1_neighbors = Distance.neighbors(t1, Eps)
        t1.setProcessed(true)
        t1.setCoreDistance(t1_neighbors, Eps, MinNbs)
        emit(t1);
        if (t1.getCoreDistance() < Infinity) then     // This is a core point
          PriorityQueue.updateOrAddAll(t1_neighbors);
        end while
      end if
    end if
  end if
end for
```

OPTICS algorithm pseudo-code

For each unprocessed point

get neighboring points (closer than Eps)

mark point as processed

not currently a reachable point

see if a core point (coreDistance)

...

end if

end for

OPTICS algorithm pseudo-code

If a core point

Add all neighbors to priority queue

While priority queue has points

Get a point

Get the points neighbors

Mark point as processed

See if point is a core point (coreDistance)

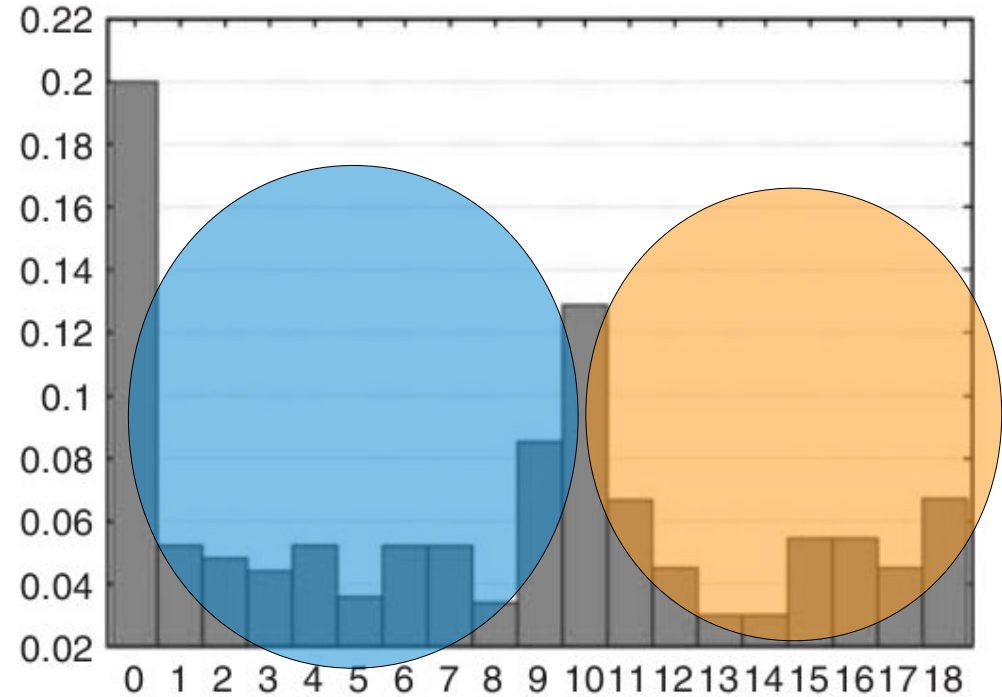
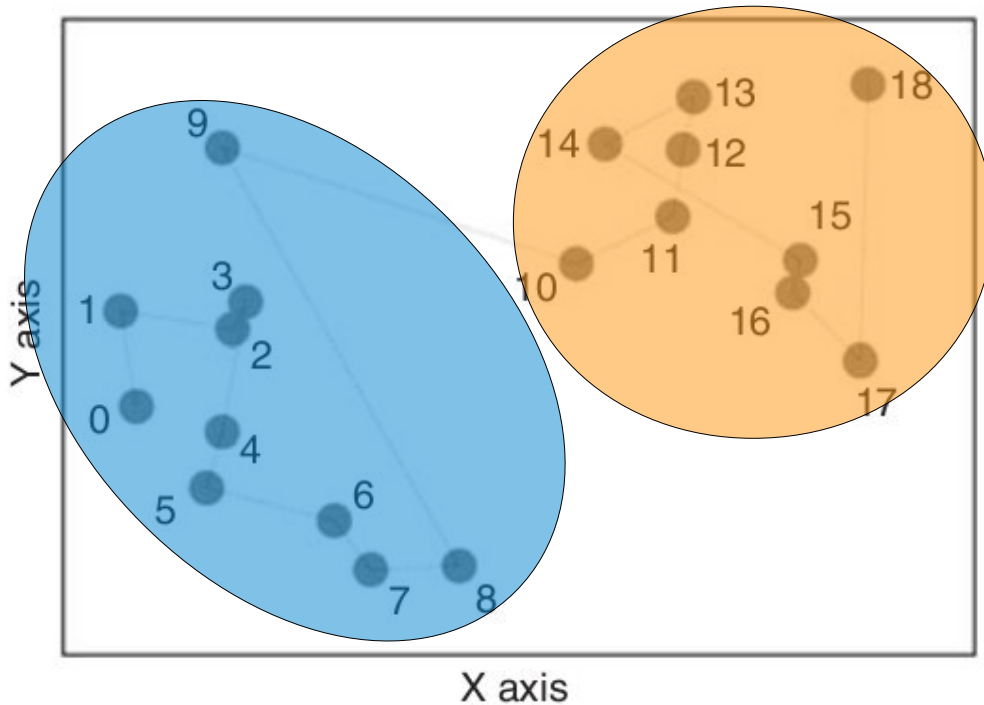
If point is a core point

Add all point's neighbors to priority queue

end while

end if

Reachability plot in relation to clusters



Clusters represented as valleys

OPTICS algorithm
generates a distance for each point
that can be plotted in a reachability plot

Case study

Car trajectory data

Data for analysis

- Using a subset of a larger dataset consisting of:
 - GPS-tracked cars: 17,241
 - Trajectories: 6187
 - Date: Wednesday, 4/4/2007
 - Time: 6:00 am to 10:00 am
 - Location: Milan, Italy
- Complete data set contains 176,000 trajectories

Trajectories data

- Spatio-temporal data
 - $x, y + \text{time}$
- $\{\text{id}, (x_1, y_1, t_1), (x_2, y_2, t_2), \dots\}$
- Most clustering methods experience difficulty handling poly-lines (trajectories)
- Convert trajectory $\rightarrow ?$
to be analyzed by clustering methods
- Two primary approaches for clustering trajectories
 - Feature-based
 - *Direct geometric models* *

Clustering trajectory data?

- OPTICS algorithm takes a distance function to compute a relative distance between two objects (or trajectories)
- We discuss several distance functions

Simple distance functions

- **Common source**

- Distance between start points of trajectory

- **Common destination**

- Distance between end points of trajectory

- **Common source and destination**

- Average distance between source and destinations

Distance functions

- **k-points**

- Defines the number of intermediate points to check when comparing two paths

- **Time steps**

- Defines the desired temporal distance between check points

Generally these only work for perfect data

- Equal time intervals
- Minor positioning error

Distance functions

- **Route similarity**

- Tolerate incomplete trajectories
- Repeatedly search for the closest pair of positions in the two trajectories
 - Compute mean distance between these positions
 - Subtract “penalty” distance for unmatched positions
- Computationally expensive

- **Route similarity + dynamics**

- Account for relative times of arriving in corresponding positions
- Temporal distance to some reference time moment

Visualizing clustered trajectories

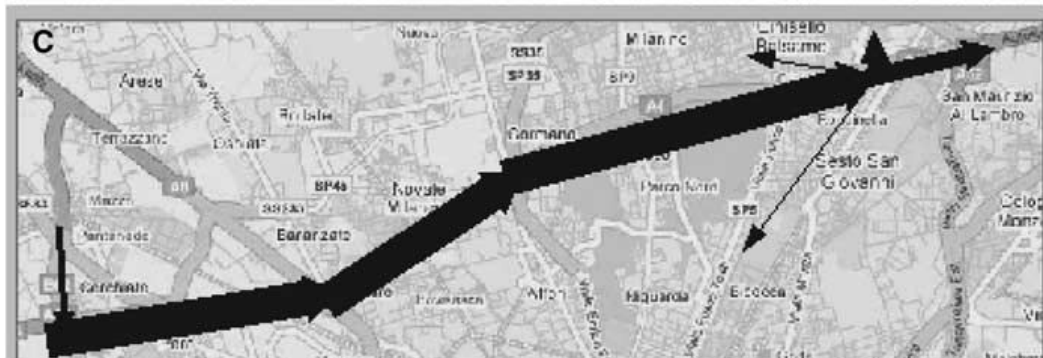
Visualizing a cluster of trajectories



Single trajectory

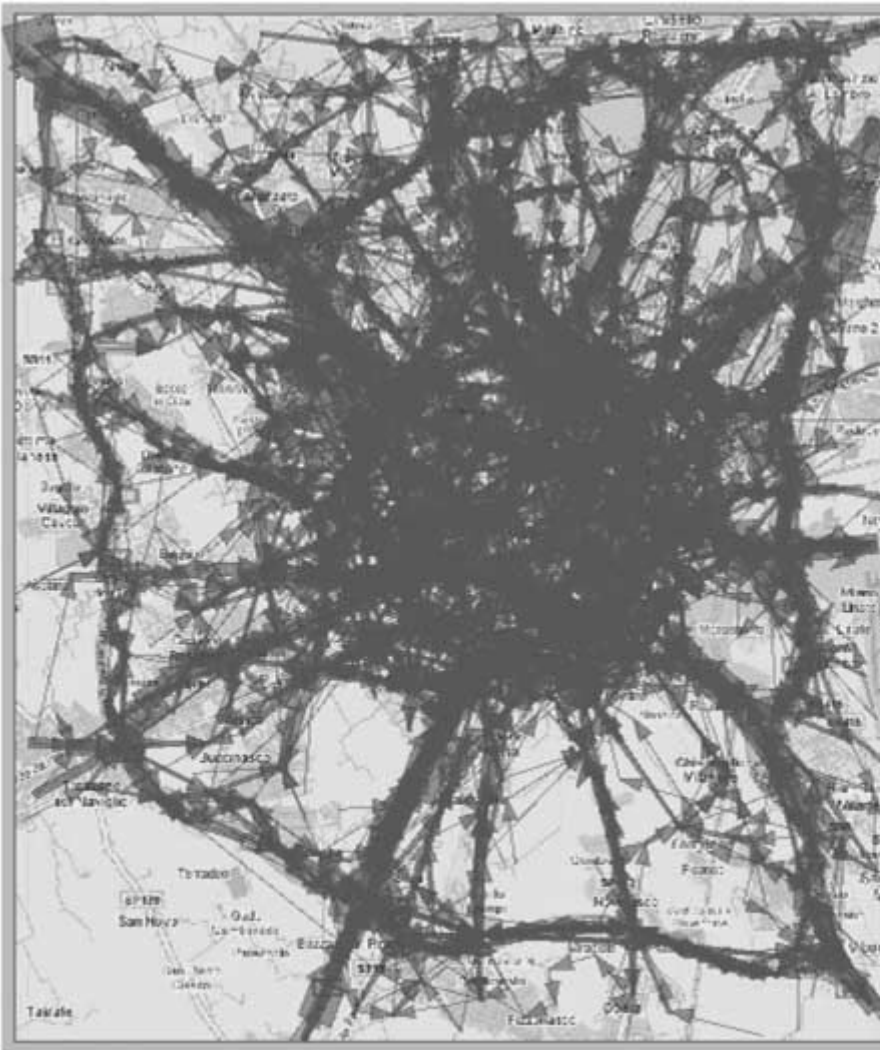


Cluster of trajectories
(each trajectory at 30% opacity)



Summarized representation of the cluster
(thicker lines represent more shared trajectories)

Shared trajectories



- Summarized representation does not work for arbitrary movement data
- Hard to make sense of the data

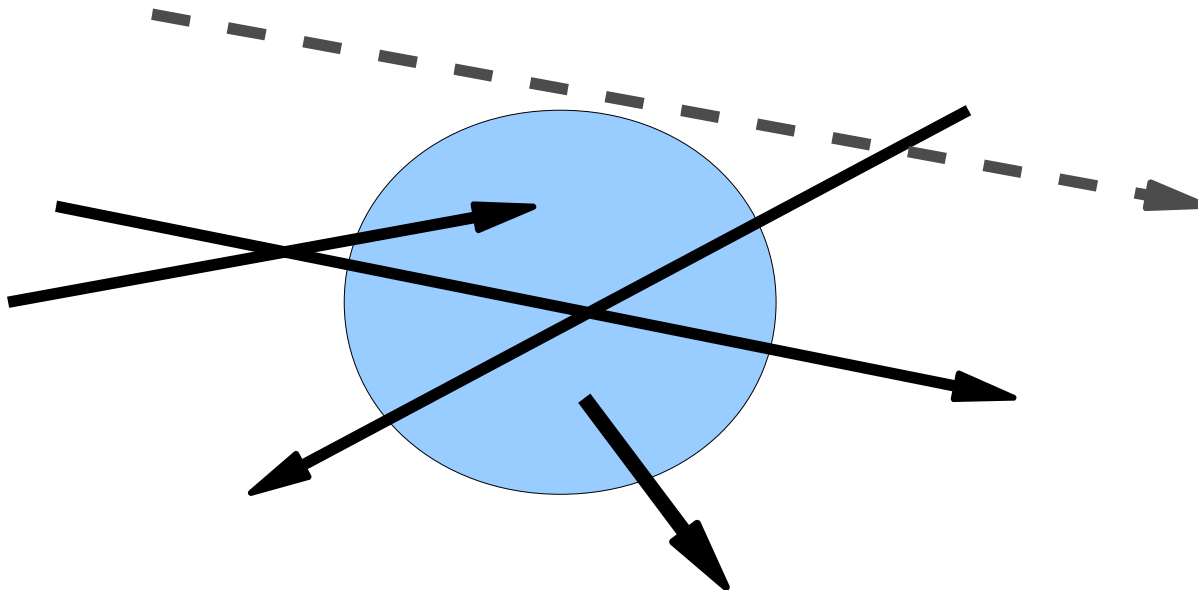
Dynamic aggregator

- Compute aggregate attributes for all members in the aggregator
 - Member count, minimum, maximum, average, median, etc.
- Responds to interactive filtering mechanisms
- Two types of aggregators
 - Summation places
 - Aggregate moves

Aggregator

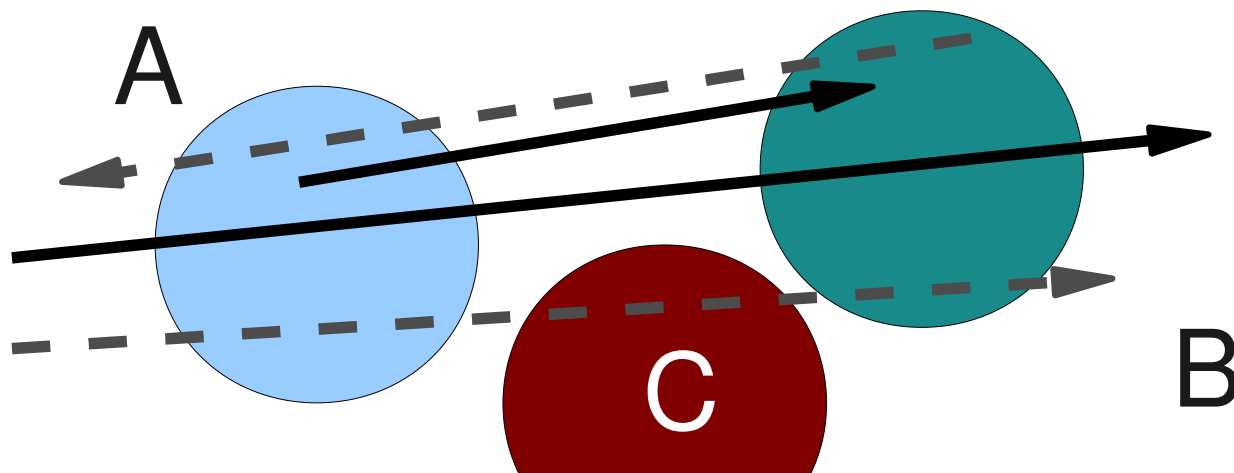
- **Summation places**

- An area in space that is 'aware' of all trajectories that pass through it
- Record positions and times entering and exiting the area for each trajectory



Aggregator

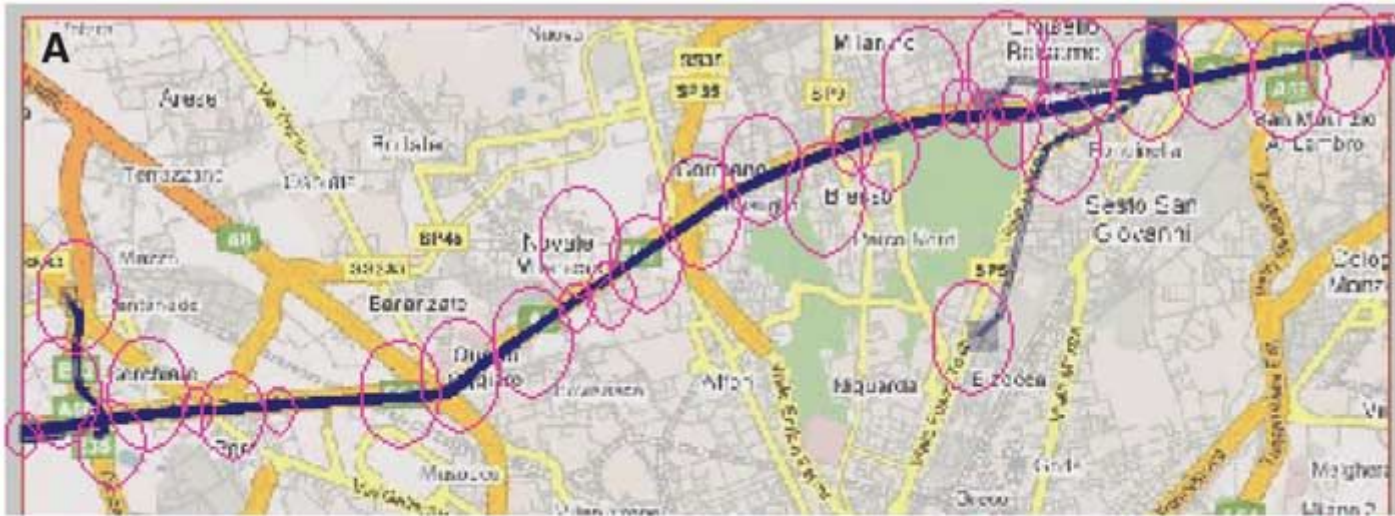
- **Aggregate moves**
 - $A \rightarrow B$
 - Two areas (A & B) that are 'aware' of all trajectories that go directly from A to B without passing another area
 - Record member count, statistical summaries, lengths of fragments, durations, and speeds



Finding areas for aggregators

- Important to find interesting areas for aggregators that contain many trajectories
- Method for finding some interesting areas
 - Extract start, end, stop, and turn **points**
 - Parameter: Minimum duration of a stop
 - Parameter: Minimum angle of turn
 - Build **circles** around concentration of points
 - Parameter: Minimum circle area
 - Parameter: Maximum circle area

Automated method



Automated method
used to find
summation places

Comparing methods



Semantically defined significant places



Automated method for finding interesting places

Improving the understanding of clusters of trajectories

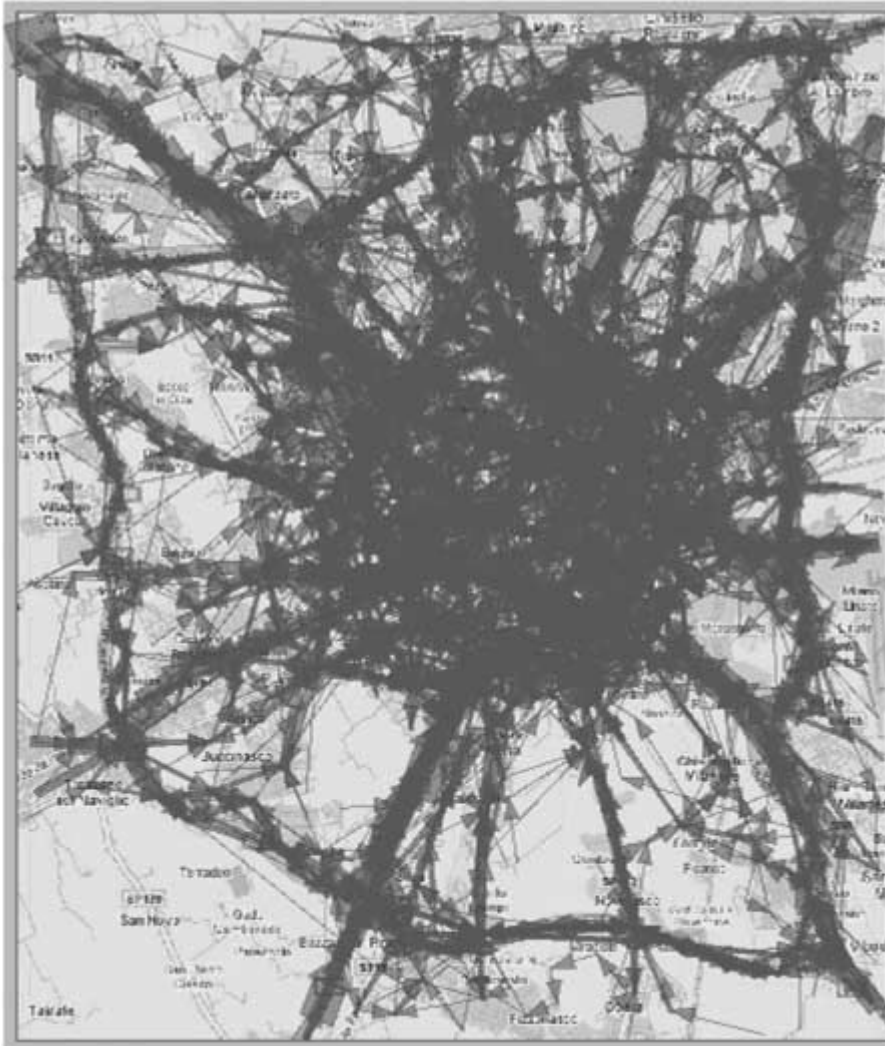
- Further work is needed to automatically find interesting areas for these methods to work sufficiently well
- Therefore other methods are also used
 - Interactive filtering
 - Visualize aggregate attributes
 - Progressive clustering

Interactive filtering

Interactive filters

- Temporal filter
 - Select a time interval
- Spatial filter
 - Select a 'window' of space
- Attribute filter
 - Select by attributes such as length and speed
- Cluster filter
 - Selection of clusters

Results of interactive filters



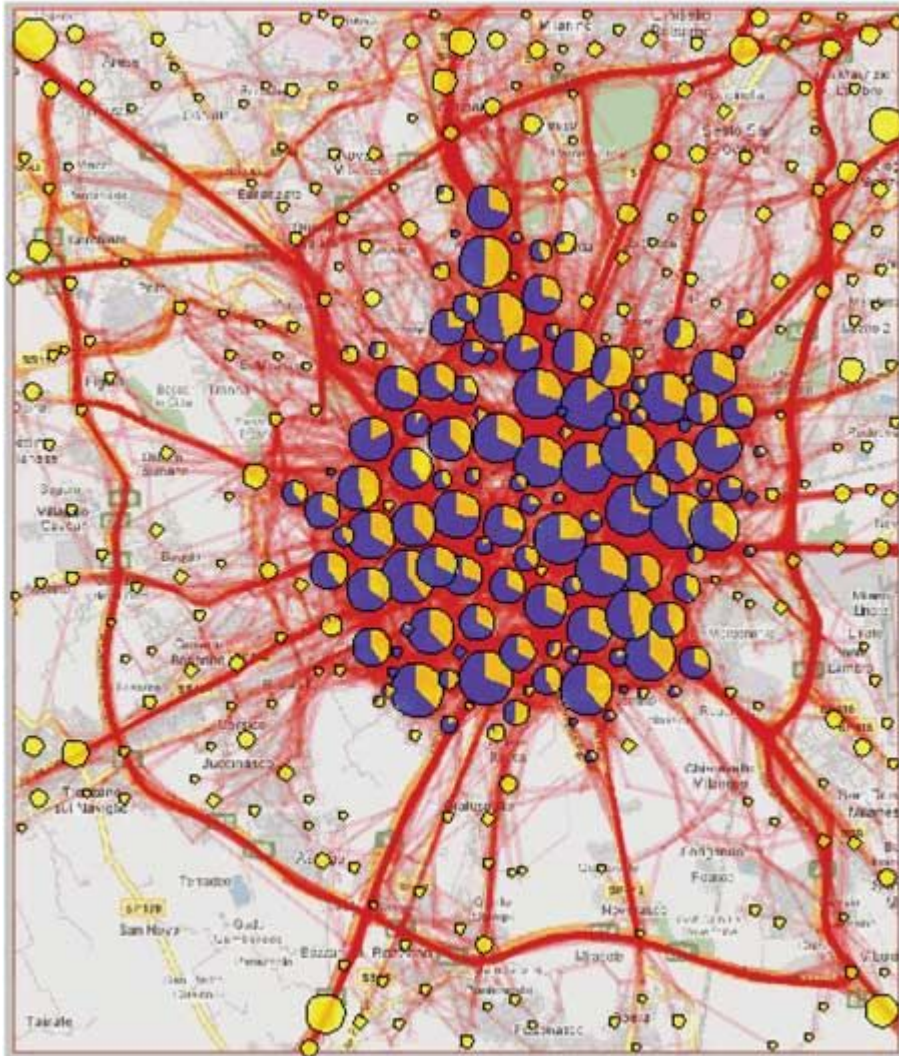
Aggregate moves



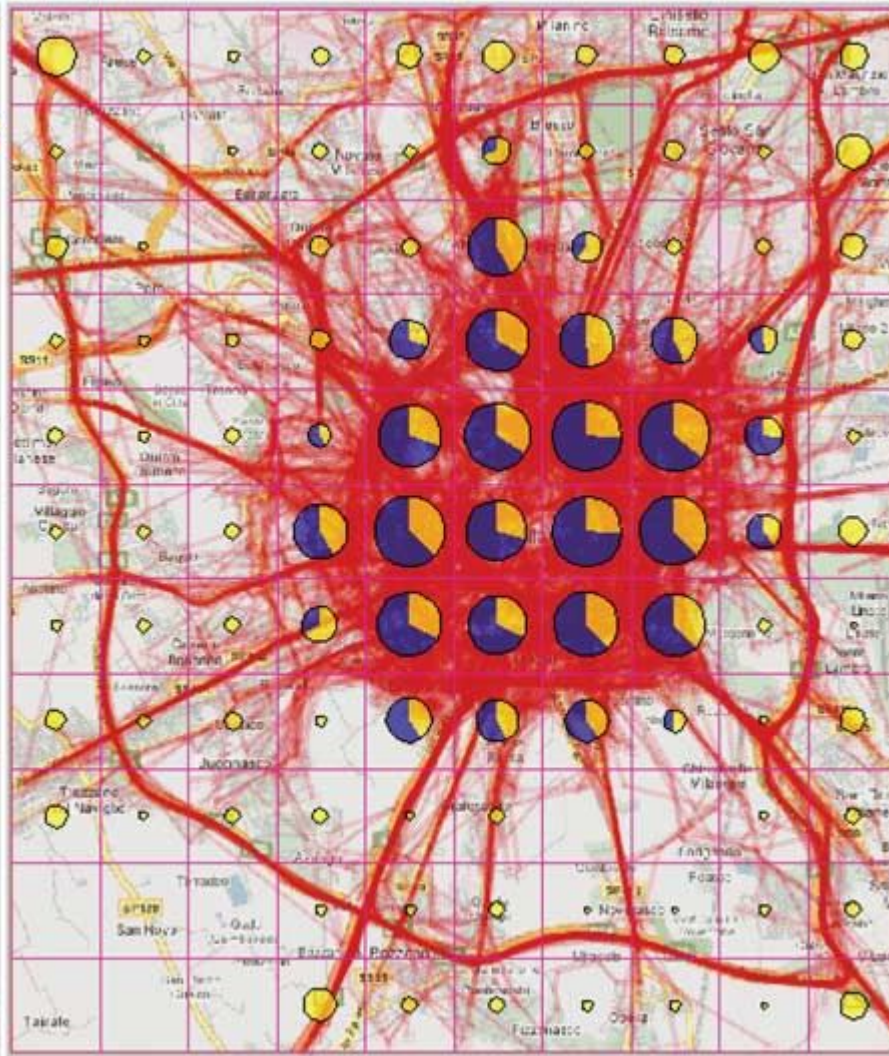
Only clusters with ≥ 15 members

Visualize non-trajectory attributes

Visualize aggregate attributes



Places from automated method



Grid cells

Yellow
Start
Blue
End

Progressive clustering

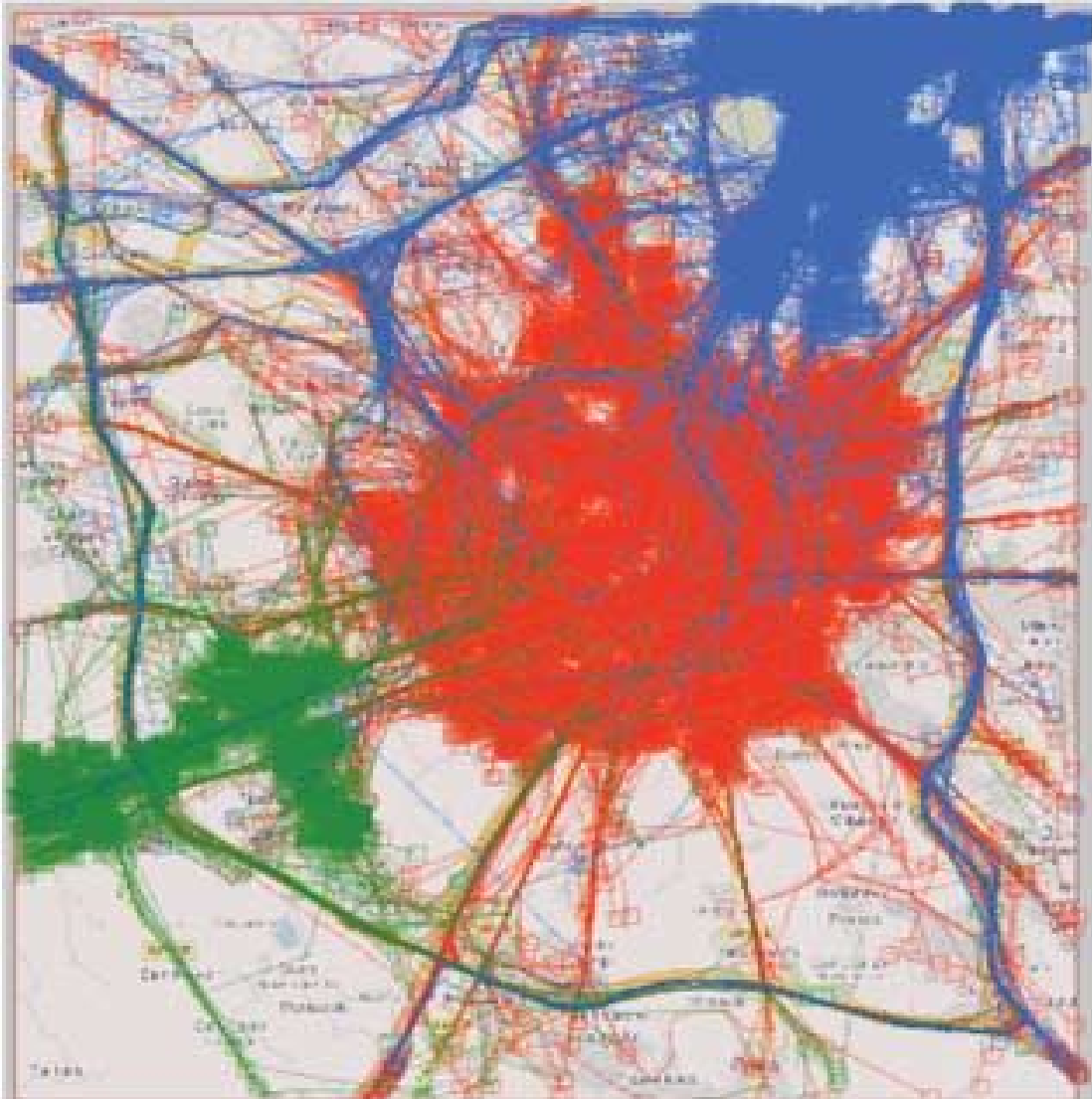
Progressive clustering example

- The data set contains the trajectories for people primarily driving to work.
- **Goal:** Find the major areas where people drive to work.
- **Problem:** Density of data set is not uniform causing problems for OPTICS density-based algorithm
- **Solution:** Progressive clustering is a practical approach to analyzing unevenly dense data

Progressive clustering steps

- Pick a large minimum number of neighbors (MinNbs)
 - Play with parameter values until a few easy to distinguish clusters appear
 - Examine, describe, and explain clusters
 - Remove the objects in these clusters from analysis
- Continue analysis with a smaller number of minimum neighbors (MinNbs)
- Repeat until analysis is sufficiently complete

First clustering - Route Destination



Large clusters

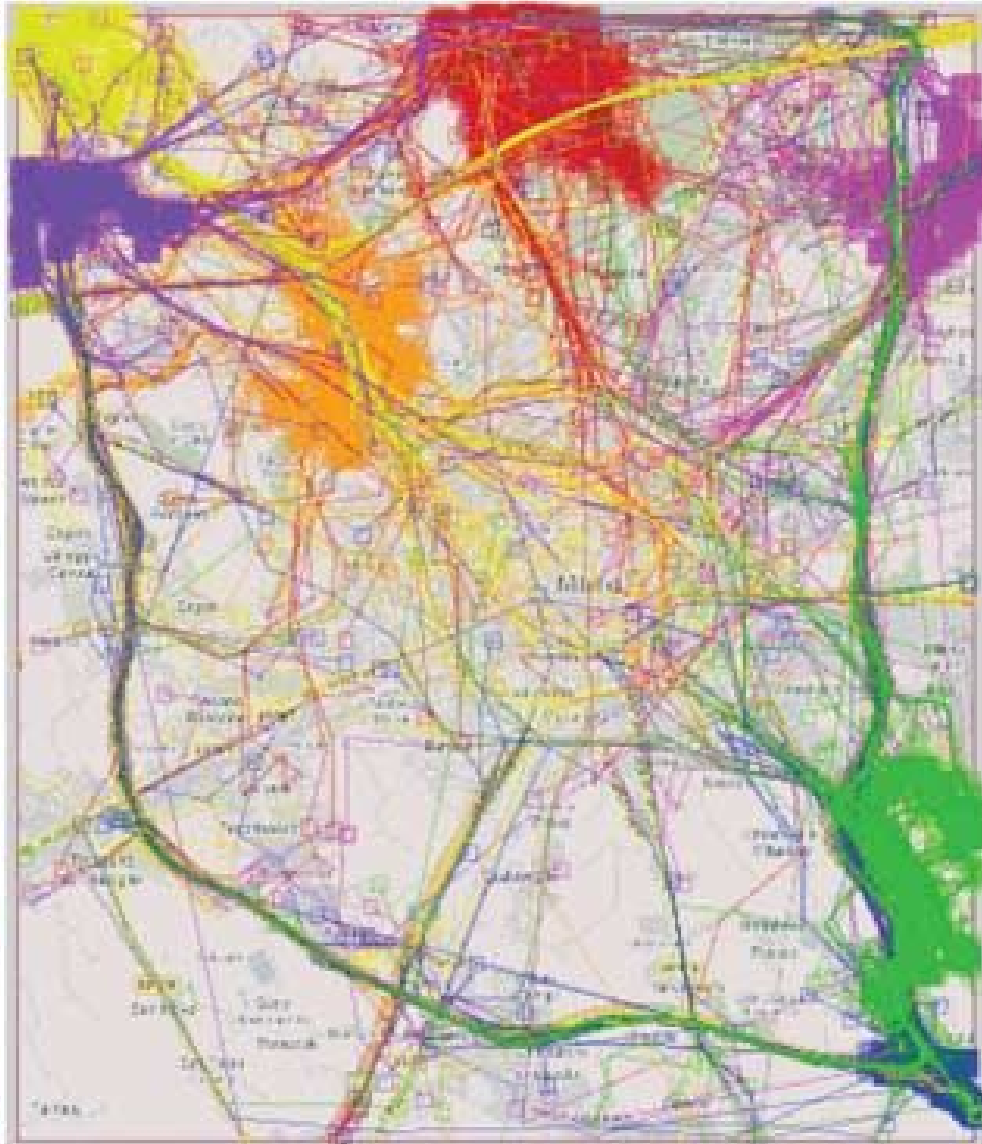
Destination function

Eps=500m

MinNbs=15

1781, 893, 354 trajectories

Second clustering



Smaller clusters
(large clusters removed)

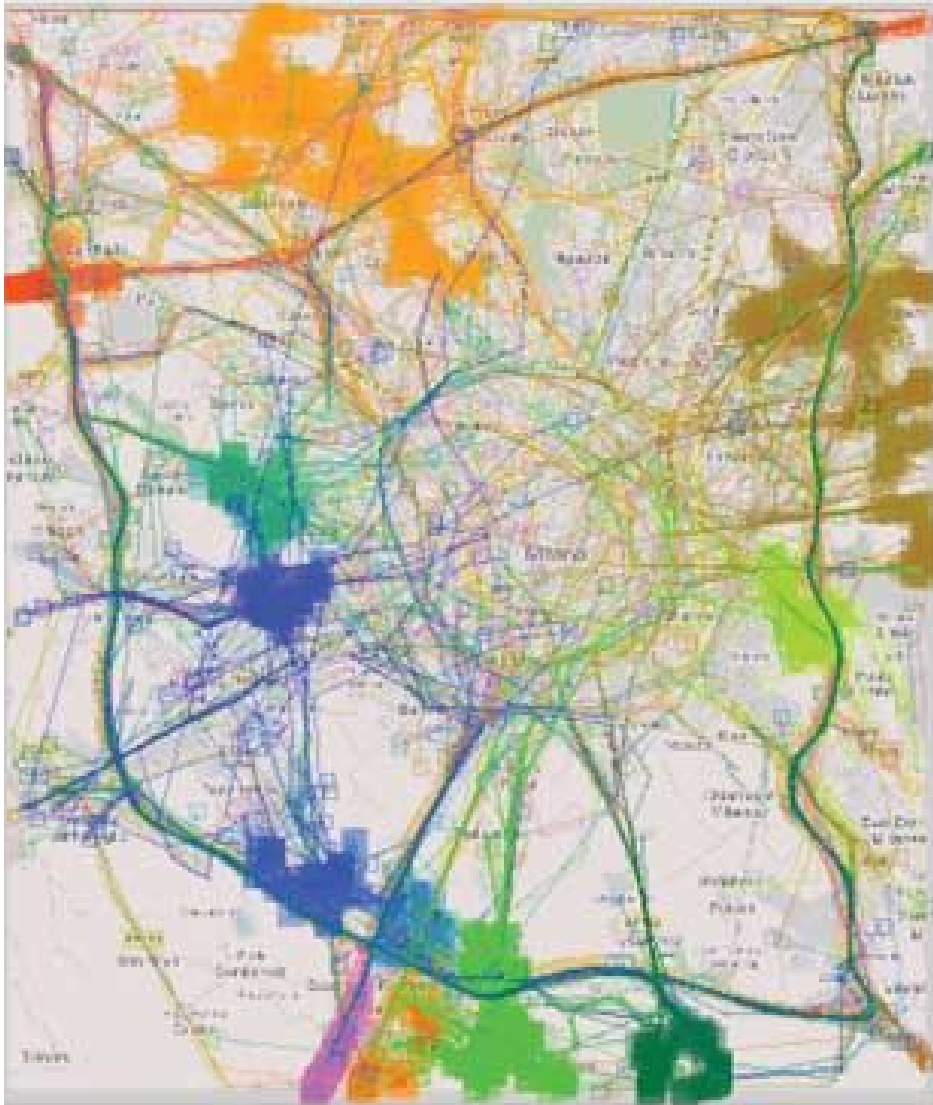
Destination function

Eps=500m

MinNbs=10

299-127 trajectories

Third clustering



Smallest clusters
(larger clusters removed)

Destination function

Eps=500m

MinNbs=5

219-45 trajectories

Results of progressive clustering on destination locations

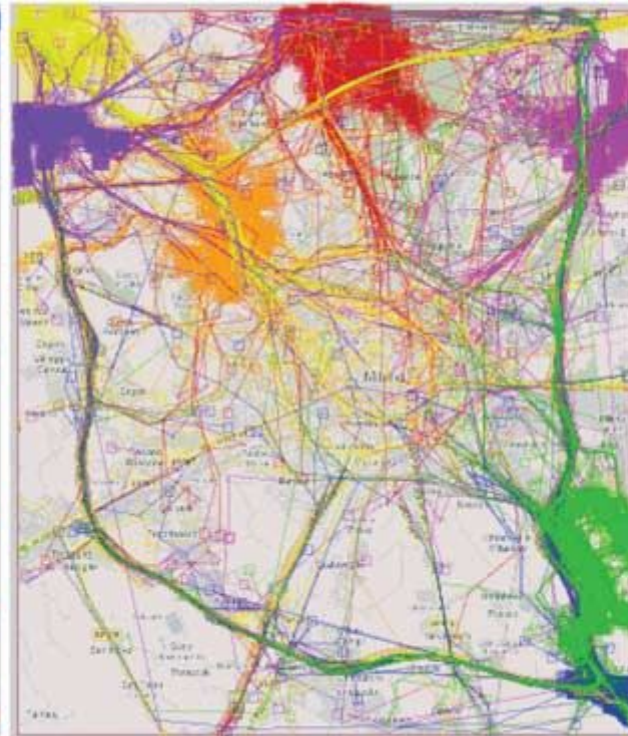


Large clusters

Eps=500m

MinNbs=15

1781, 893, 354 trajectories



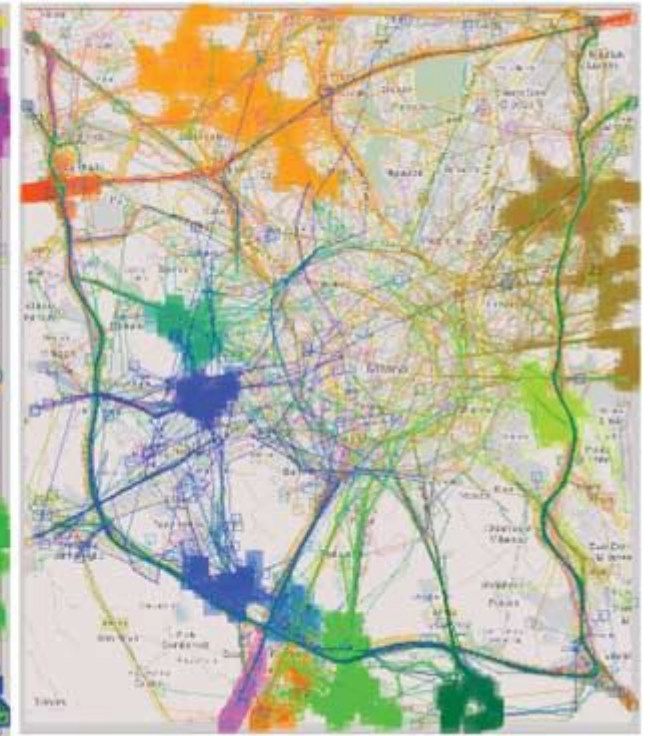
Smaller clusters

(large clusters removed)

Eps=500m

MinNbs=10

299-127 trajectories



Smallest clusters

(larger clusters removed)

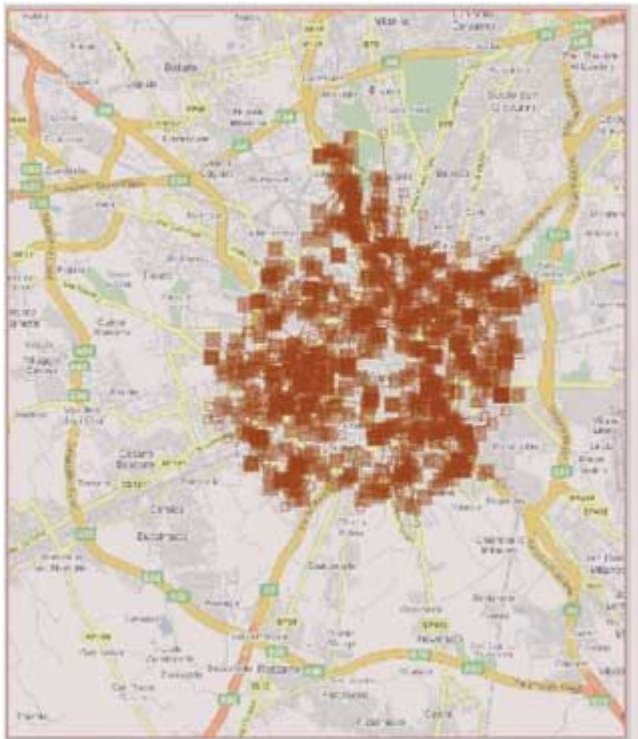
Eps=500m

MinNbs=5

219-45 trajectories

Analyzing the largest (red) cluster

- Select largest cluster out of the previous analysis step and apply further analysis
- Cluster routes based on 'route similarity'



Most routes are short and in the center

517 short routes (29% of cluster)

Eps=1000m

MinNbs=5

Analyzing the largest (red) cluster

- Without short routes
- Cluster routes based on 'route similarity'



Very few clusters

Cluster sizes range from 11 to 28

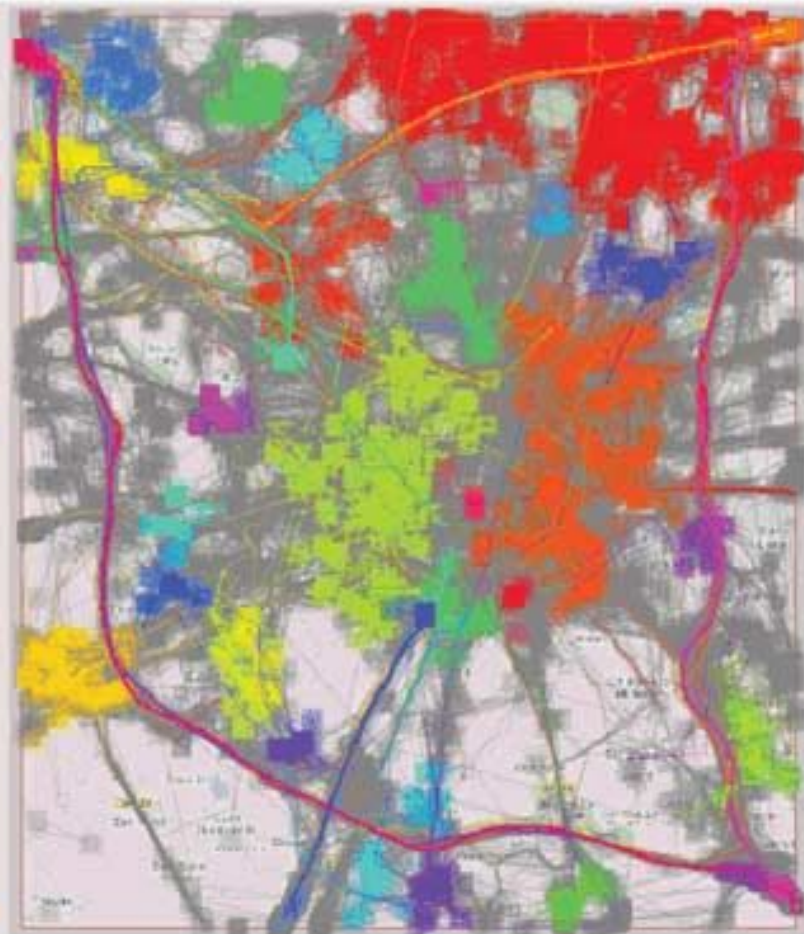
Few routes are distance routes

Remaining 63% trajectories are noise

* Other 2 large clusters generate similar results

Second example: source and destination

- Apply source + destination function to cluster trajectories in entire dataset



2% (127 trajectories)
significantly different start/end

23% (1439 trajectories)
starts close to ends

75% (4621 trajectories)
noise
(gray)

Includes roundtrips

Eps=800m
MinNbs=8

Long distance routes

- Remove short and roundtrip routes
- Apply 'route similarity' to remaining routes



Peripheral routes

Inward routes

Outward routes

Long distance routes

- 41 clusters
- 66-5 trajectories per cluster
- Inward routes are larger and more numerous than outward routes
- Authors note that different cluster functions bring similar results

Summary of findings

- No destinations are significantly more popular than other destinations
- Most trips are short local trips
- Most trajectories follow unique routes
- Many peripheral routes that do not enter city and follow belt around the city
- Inward trips are more frequent than outward trips, since the morning traffic

Progressive clustering + simple distance functions

- Progressive clustering with simple distance functions
 - Only apply computationally intensive functions to small areas of interest
 - User understands the steps created for each result
- Its possible to combine distance functions into a comprehensive function to address heterogeneous attributes
 - Computational limitations may make this approach infeasible
 - User will have a difficult time making sense of the clusters and the results of the analysis

Future work

- Re-clustering
 - Automatically search a range of parameters $500 \leq \text{Eps} \leq 1000$ and $5 \leq \text{MinNbs} \leq 10$
 - Return the set of results for the user to review
- Automatic quality assessment
 - Automate a process to evaluate a large number of possibilities
 - Use quality measure such as average density or cohesion
 - Applying to trajectory data is an open problem

Conclusions

- Trajectories are complex spatiotemporal constructs and are difficult to analyze
- Progressive clustering with visualization and interaction techniques are a suitable approach to explore and analyze the data
- Combine several functions iteratively helps users better understand the data
- Allows the user to control the amount of computation to apply to each step

Thank you

Questions?