

# Towards Parallel Detection of Movement Patterns in Large Spatio-temporal Datasets

## Oral Qualification

Andres Calderon

April 2, 2017

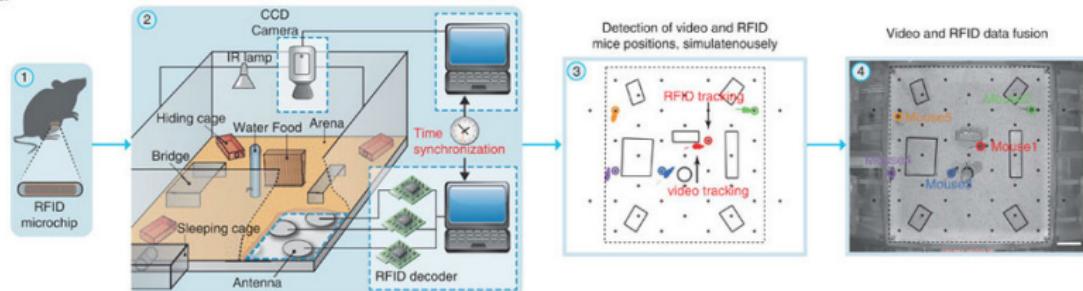
# Trajectory datasets

- ▶ Anything that could move (or could be moved), will be tracked...
  - ▶ Sensors, Sensors everywhere...
  - ▶ Smart phones, GPS, RFID, WiFi, Bluetooth, ZigBee, IoT, Satellites, Drones, Surveillance cameras...

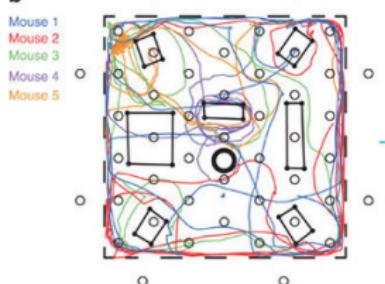
# Applications

## Tracking social behavior of animal colonies.

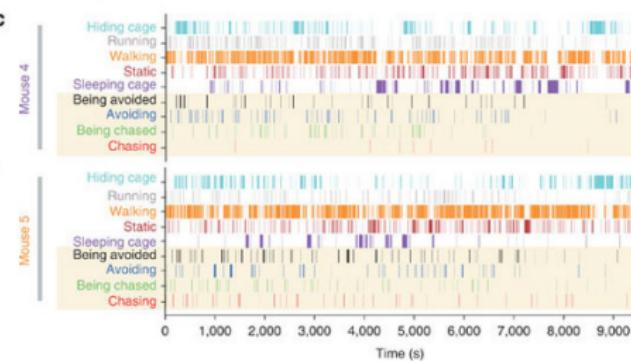
a



b

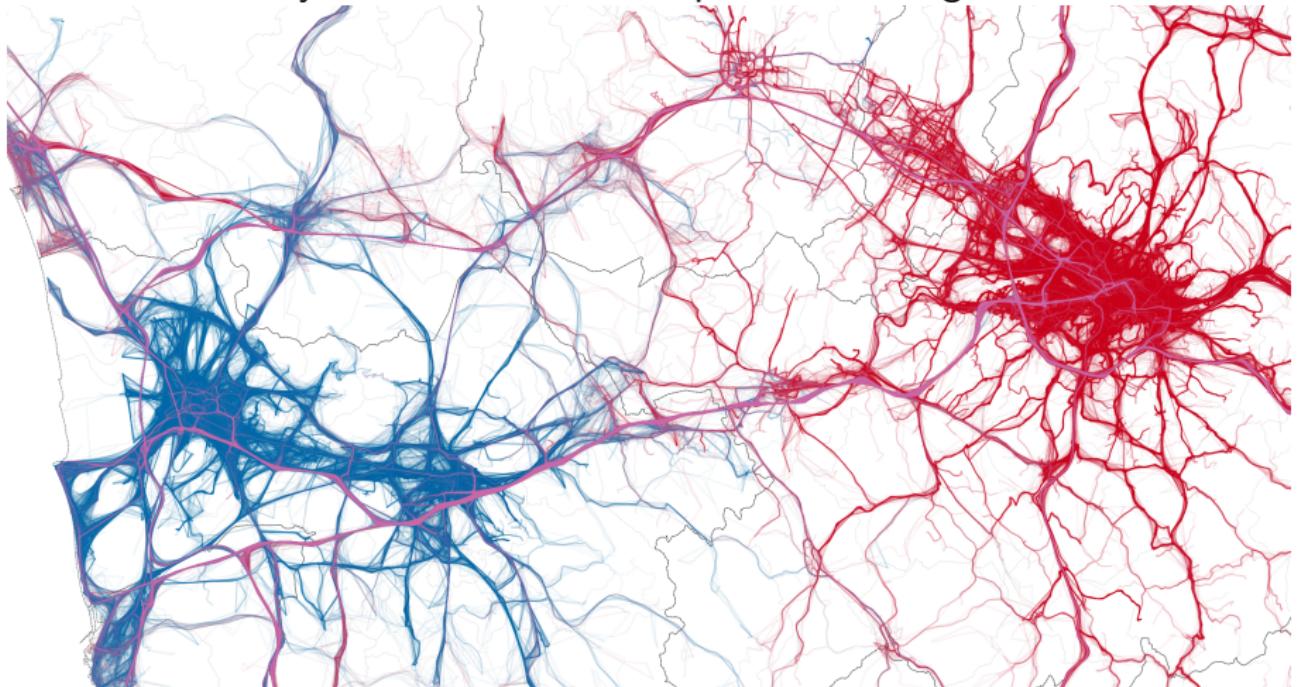


c

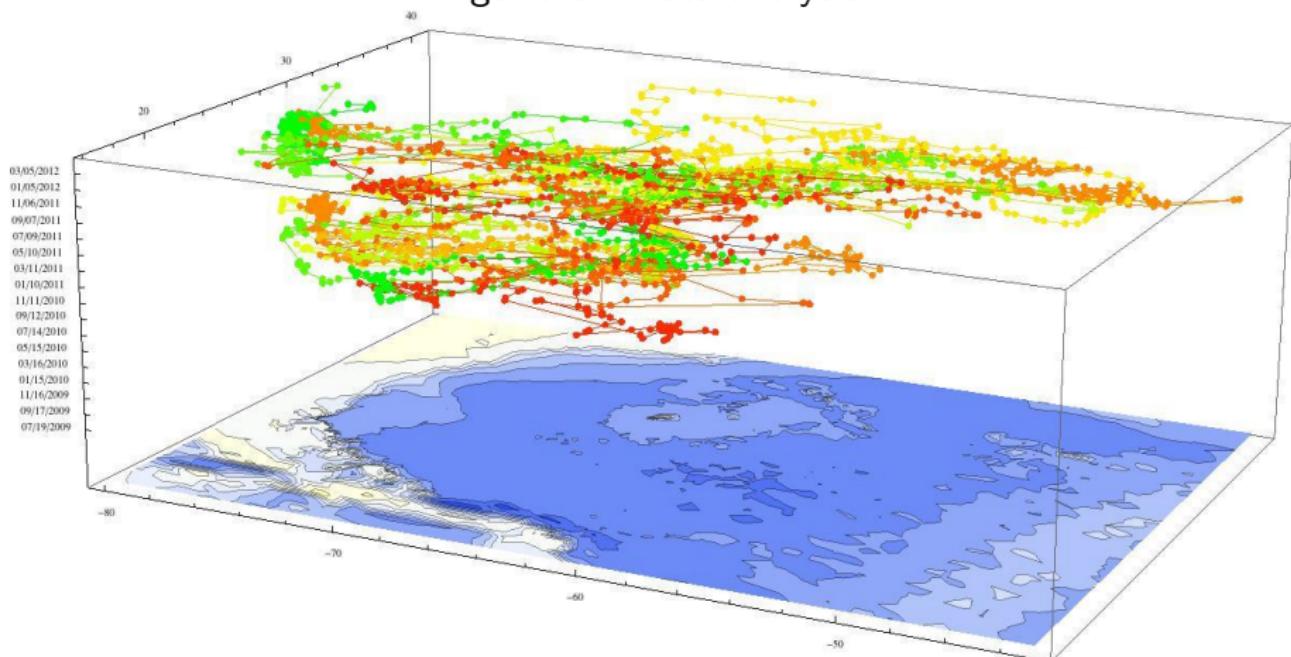


# Applications

Are you a returner or an explorer? Ask Big Data.

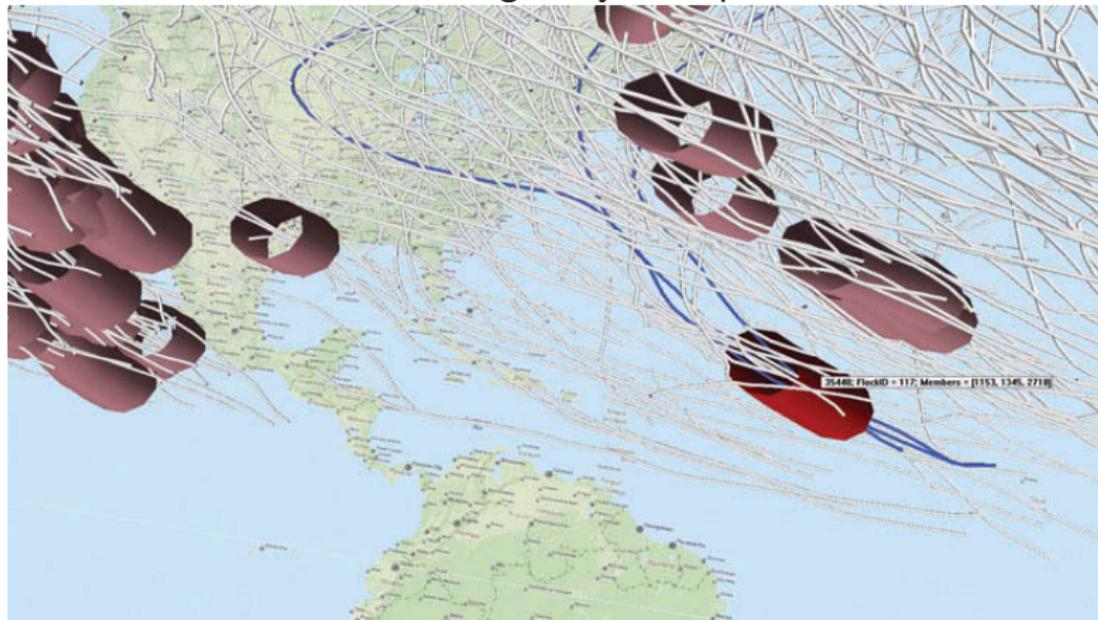


## Tiger shark data analysis.



# Applications

Visual mining of cyclone paths.



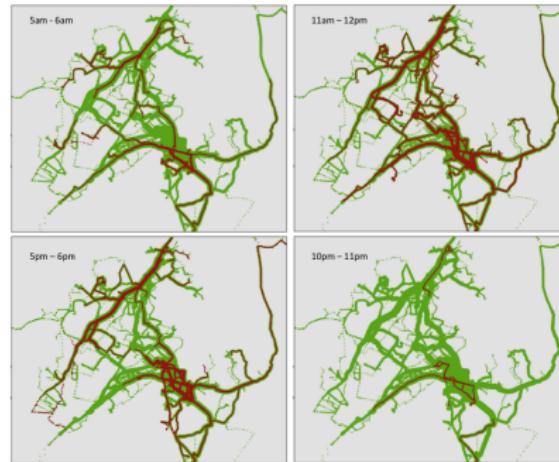
# Applications

Bird migration forecast.

# Complex movement patterns

Previous works focus on traditional queries:

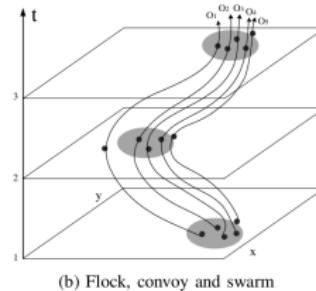
- ▶ Range
- ▶ Nearest Neighbors
- ▶ Similarity



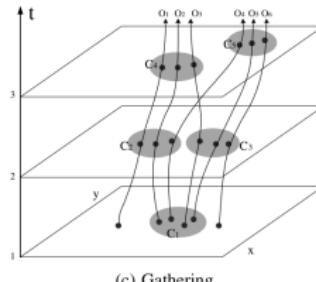
# Complex movement patterns

Recent works look for the aggregate behavior:

- ▶ Moving clusters
- ▶ Convoys
- ▶ Flocks
- ▶ Swarms
- ▶ Gatherings



(b) Flock, convoy and swarm



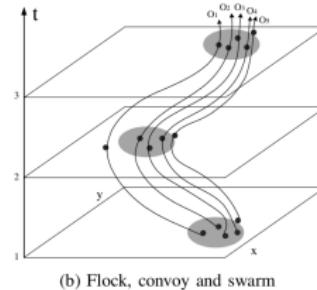
(c) Gathering

(Zheng et al, ICDE'13)

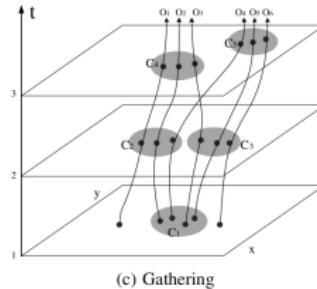
# Complex movement patterns

Recent works look for the aggregate behavior of trajectories as groups:

- ▶ Moving clusters
- ▶ Convoys
- ▶ **Flocks**
- ▶ Swarms
- ▶ Gatherings



(b) Flock, convoy and swarm



(c) Gathering

(Zheng et al, ICDE'13)

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

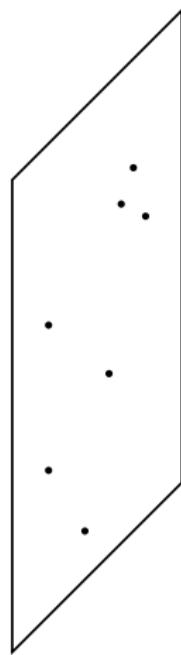
# What is a flock???

Definition  $((\mu, \epsilon, \delta) - \text{flock})$

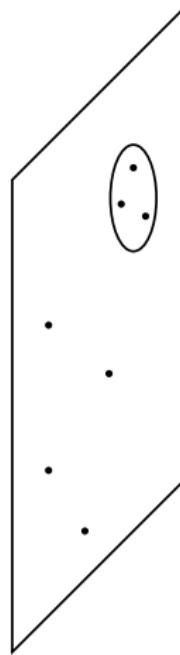
Sets of at least  $\mu$  objects moving close enough ( $\epsilon$ ) for at least  $\delta$  time intervals (Benkert et al, 2008).



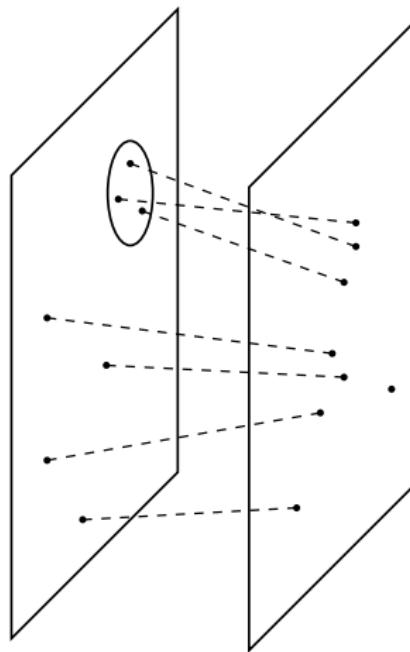
# BFE algorithm (Vieira et al, 2009)



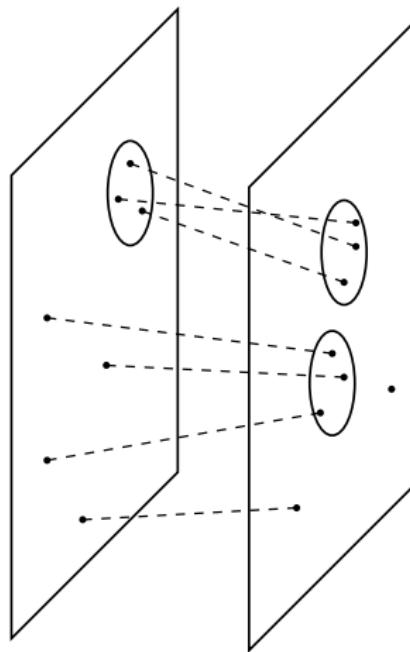
# BFE algorithm (Vieira et al, 2009)



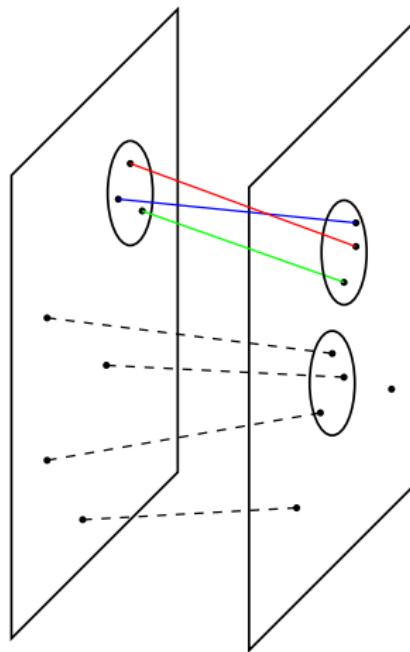
# BFE algorithm (Vieira et al, 2009)



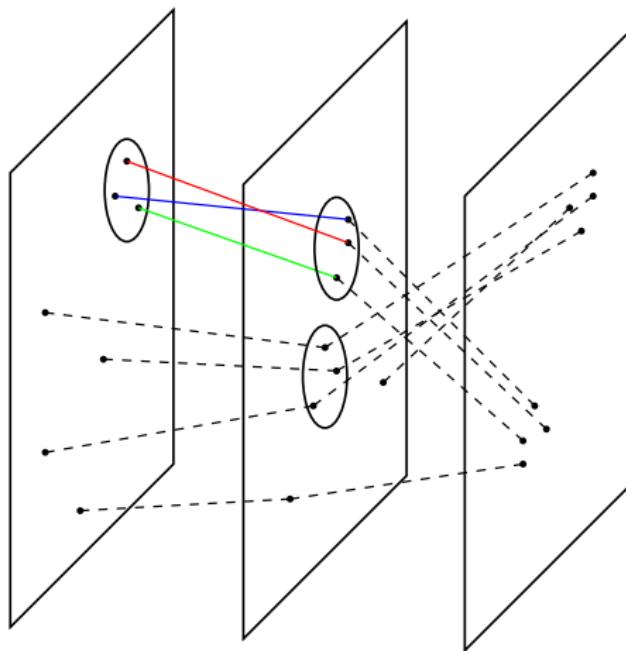
# BFE algorithm (Vieira et al, 2009)



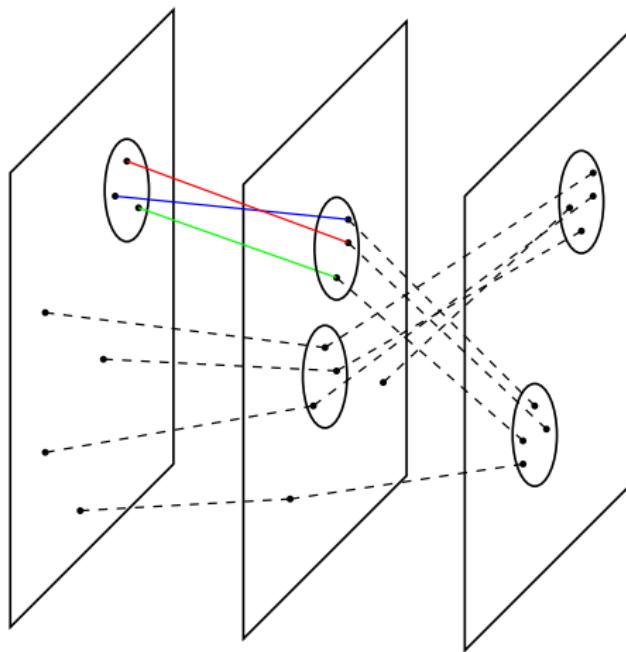
# BFE algorithm (Vieira et al, 2009)



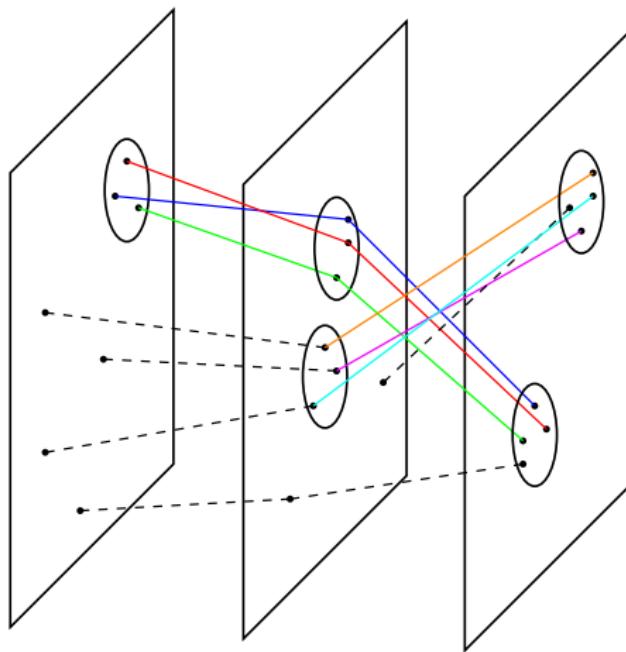
# BFE algorithm (Vieira et al, 2009)



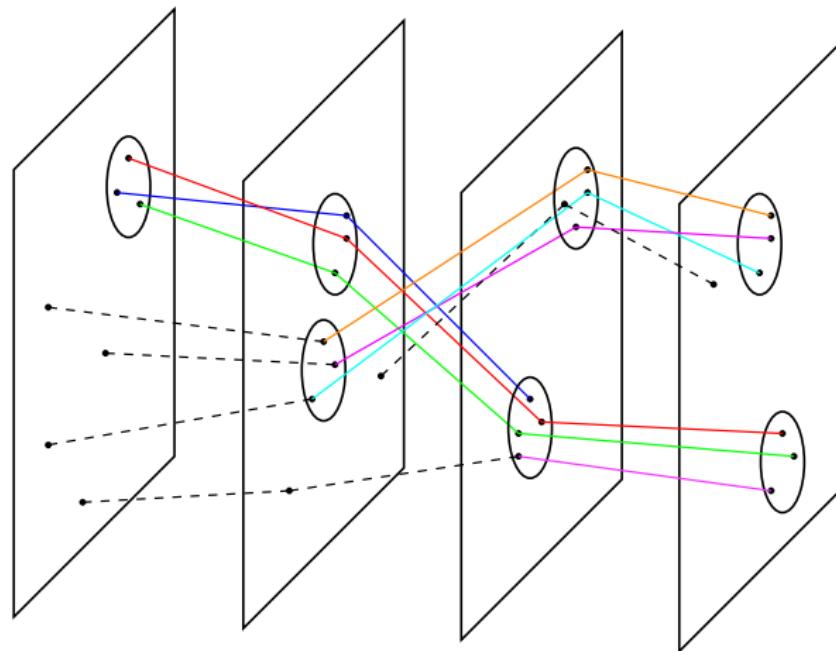
# BFE algorithm (Vieira et al, 2009)



# BFE algorithm (Vieira et al, 2009)



# BFE algorithm (Vieira et al, 2009)



# Motivation

Why is important to focus on flocks and finding disks???

- ▶ Why are moving flock patterns important?
  - ▶ They capture the collective behavior of trajectories as groups.
  - ▶ A general model for other movement patterns.
- ▶ Why is the finding of disks important?
  - ▶ It is no trivial, disks can be at any location.
  - ▶ It has a high complexity ( $\mathcal{O}(2n^2)$ ).

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

## BFE implementation

- ▶ On-Line Discovery of Flock Patterns in Spatio-Temporal Data (Vieira et al, SIGSPATIAL'09).
- ▶ Implementation available at  
<https://github.com/poldrosky/FPFlock>.

# PBFE implementation

- ▶ Written in Scala using Simba 1.6.3.
  - ▶ DISTANCE JOIN and CIRCLE RANGE operators were the key!!!
  - ▶ Tested by visual inspection and counting number of found disks.
- ▶ Code available at  
<https://github.com/aocalderon/PhD/tree/master/Y2Q1/SDB/Project/Code/Scripts/pbfe2>.

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

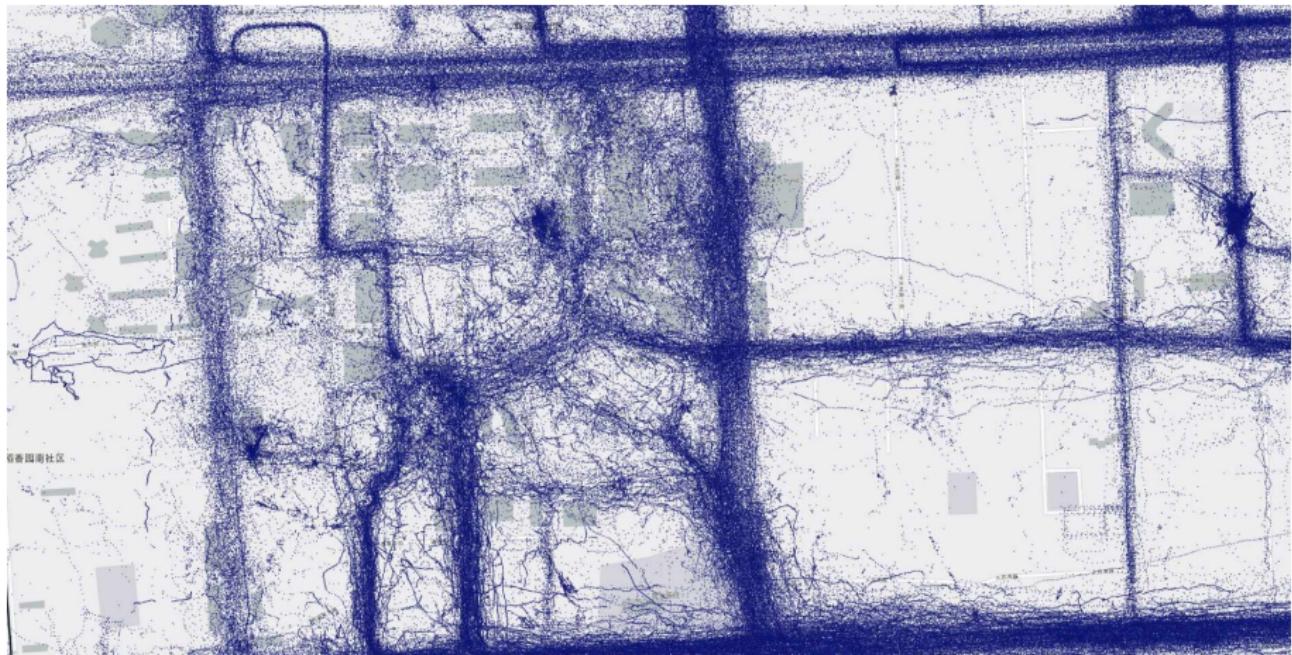
# Dataset

- ▶ **Beijing** from Geolife project<sup>1</sup>.
  - ▶ 182 users in a period of over three years (from April 2007 to August 2012).
  - ▶ 17,621 trajectories.
  - ▶ ≈18 million points (no duplicates).

---

<sup>1</sup><http://tinyurl.com/j7t2cao>

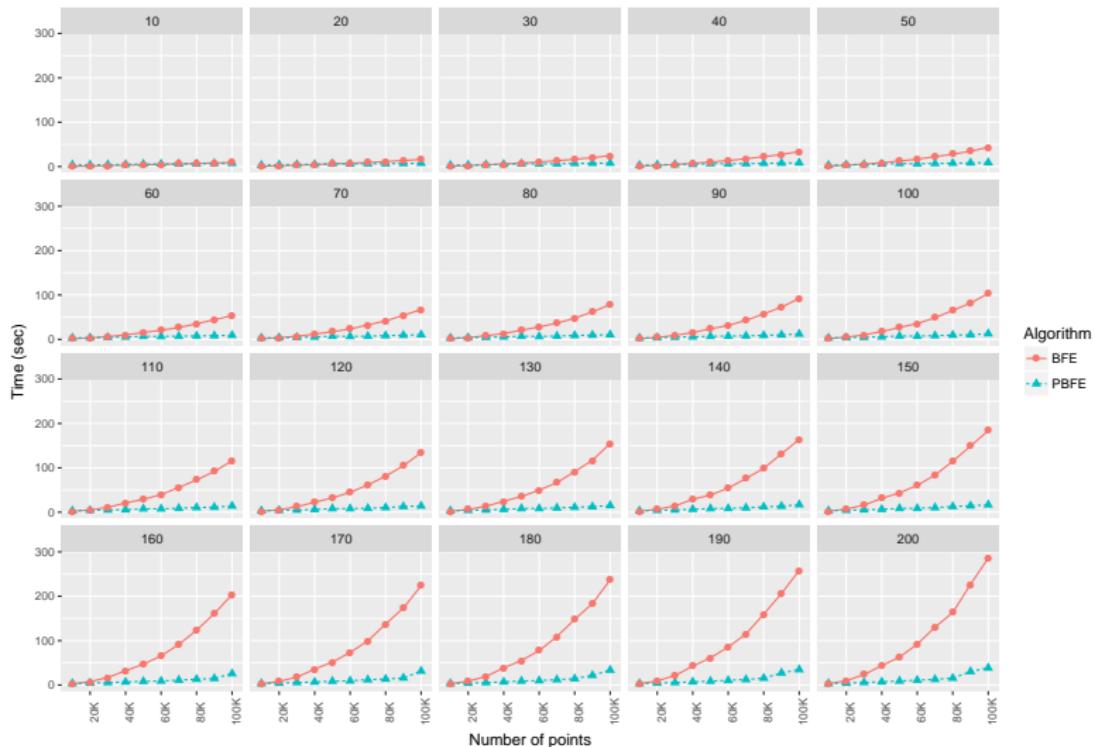
# Beijing dataset



# Setup

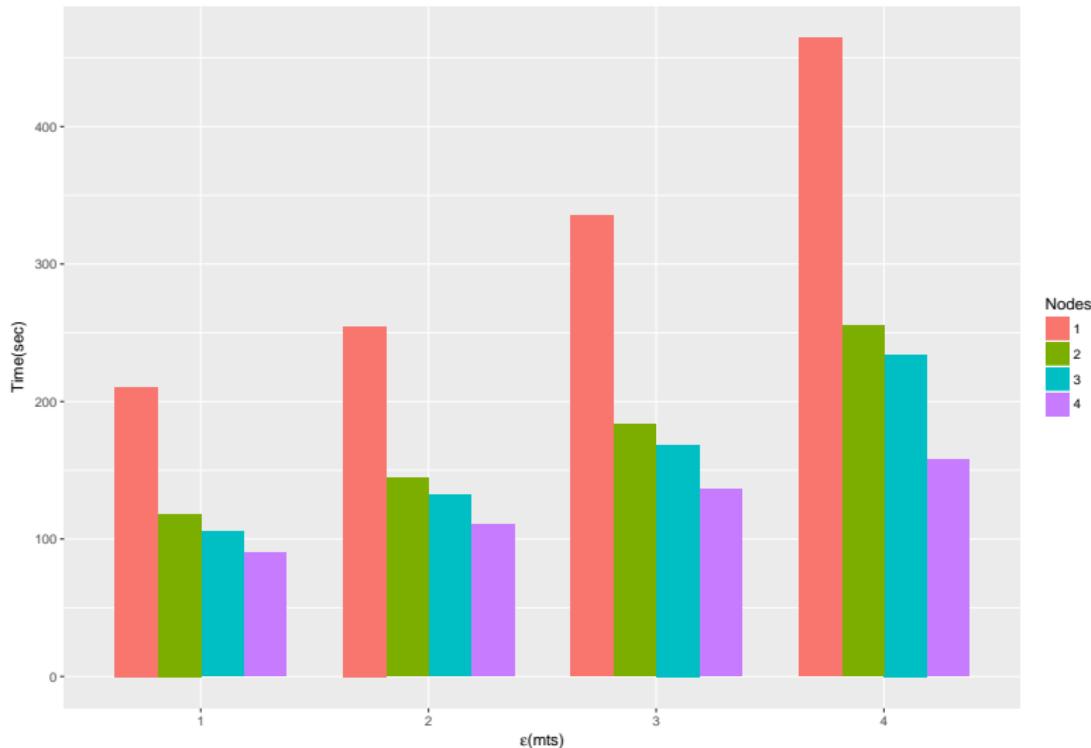
- ▶ Single-node.
- ▶ Processor: 4-core Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz
- ▶ RAM: 8 GB.
- ▶ Ubuntu 16.04 LTS, Simba/Spark 1.6.3.

# Execution time

Execution time by  $\varepsilon$  (radius of disk in mts) in Beijing dataset.

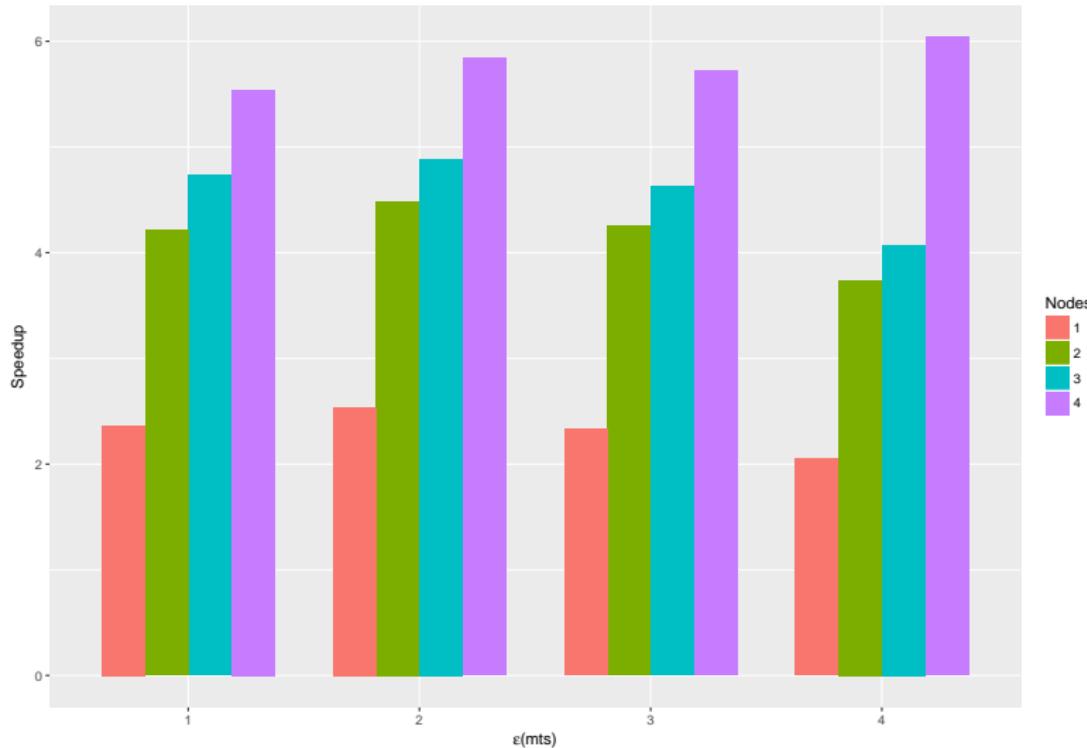
# Scaleup

Scaleup in Beijing dataset.



# Speedup

Speedup in Beijing dataset.



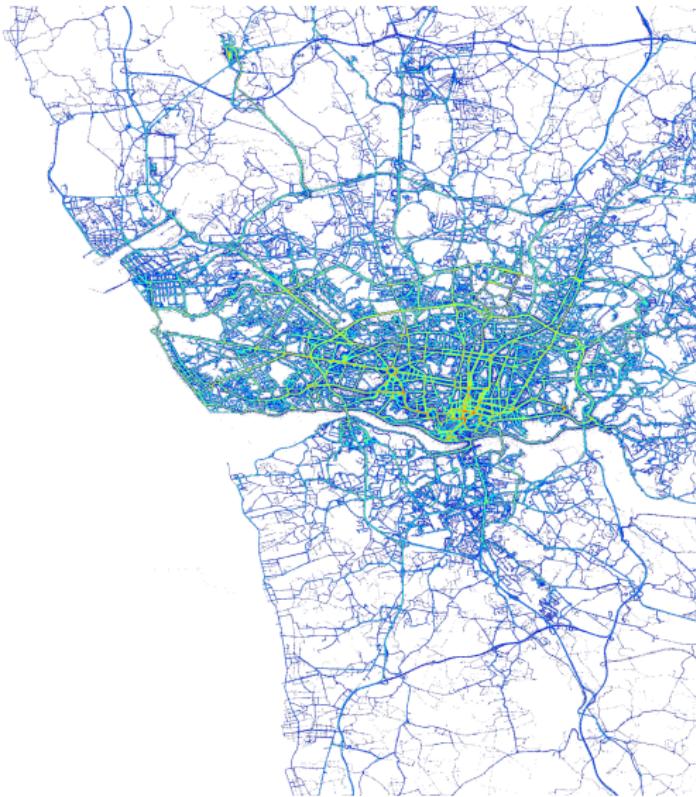
# Dataset

- ▶ **Porto** from ECML/PKDD'15 Taxi Trajectory Prediction Challenge<sup>2</sup>.
  - ▶ A complete year (from 01/07/2013 to 30/06/2014).
  - ▶ Trajectories for all the 442 taxis running in the city of Porto, in Portugal.
  - ▶ ≈17.7 million points (no duplicates).

---

<sup>2</sup><http://tinyurl.com/zzbtlt9>

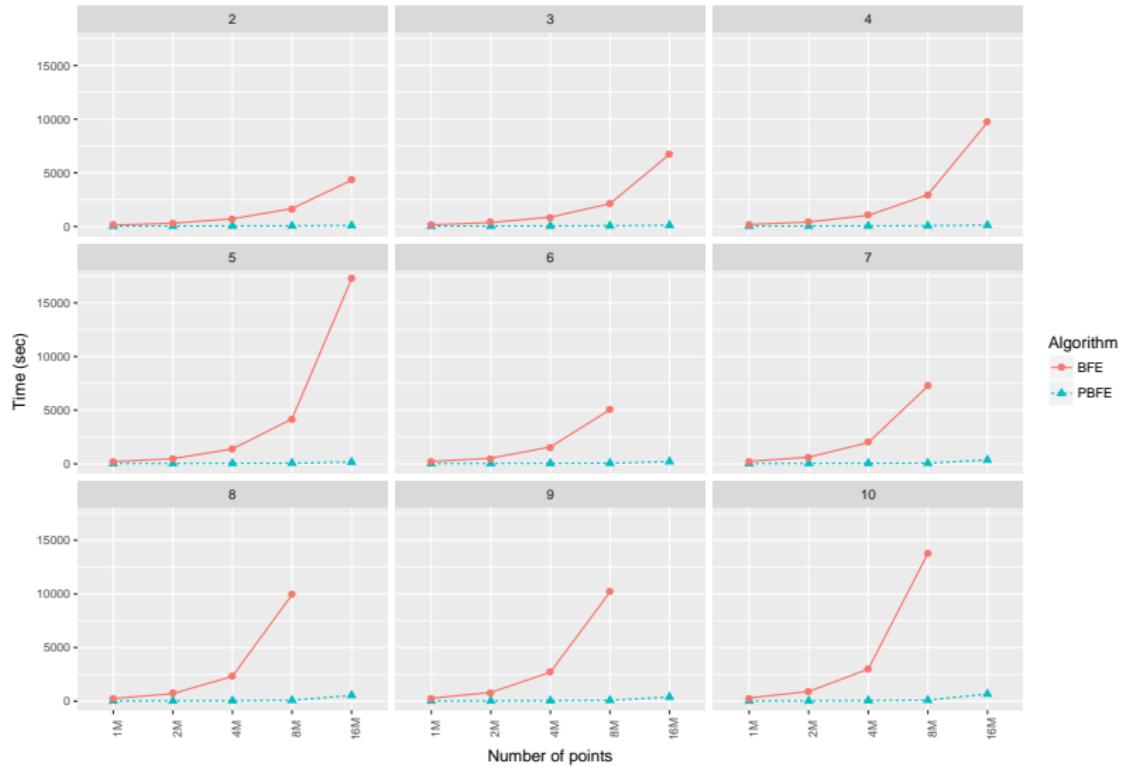
# Porto dataset



# Setup

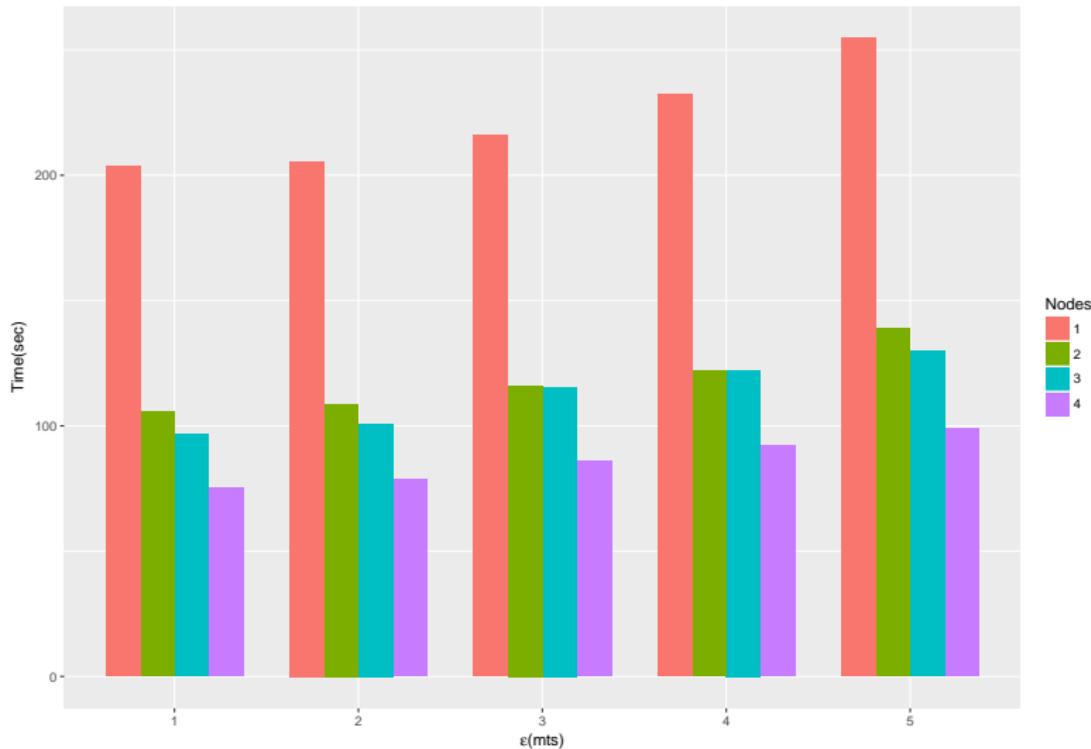
- ▶ 4-node cluster at DBLab.
- ▶ Processors: 8-core Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
- ▶ RAM: 15.5 GB.
- ▶ Centos 6.8, Simba/Spark 1.6.3.

# Execution time

Execution time by  $\epsilon$  (radius of disk in mts) in Porto dataset.

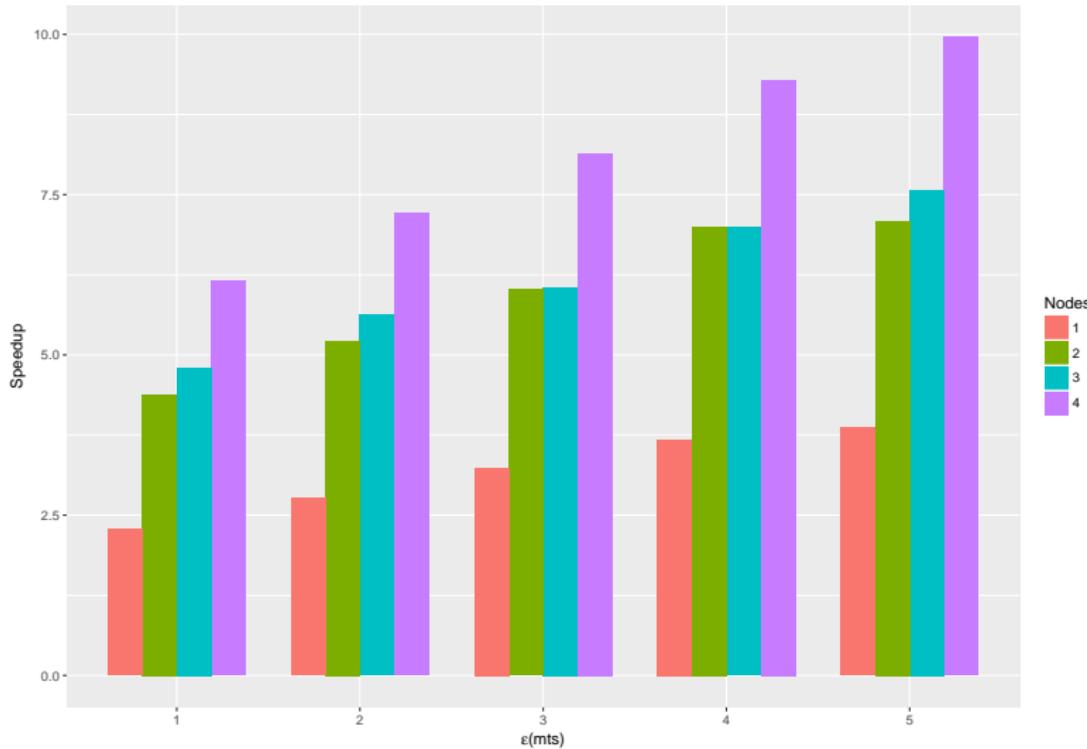
# Scaleup

Scaleup in Porto dataset.



# Speedup

Speedup in Porto dataset.



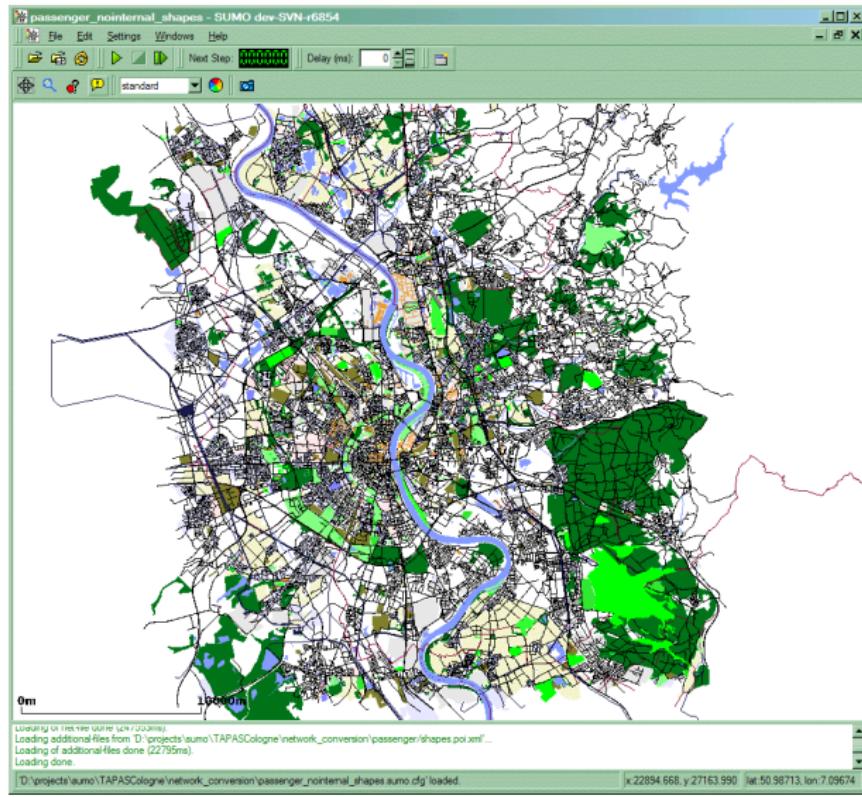
# Dataset

- ▶ **Cologne** from the SUMO (Simulation of Urban MObility) project<sup>3</sup>.
  - ▶ Simulation scenario describing a whole-day traffic in the city of Cologne (Germany).
  - ▶ Data demand based on traveling habits and infrastructure (TAPAS system).
  - ▶ ≈70 million points (no duplicates).

---

<sup>3</sup> <https://tinyurl.com/l9lbw6b>

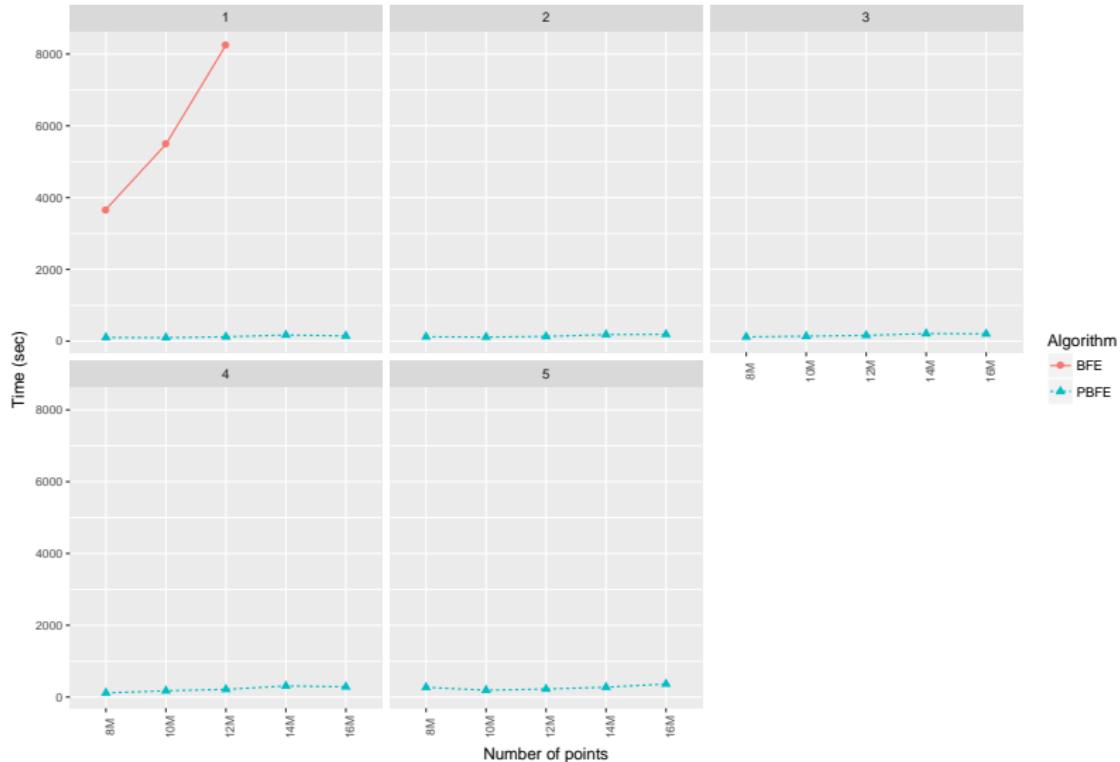
# Cologne dataset



# Setup

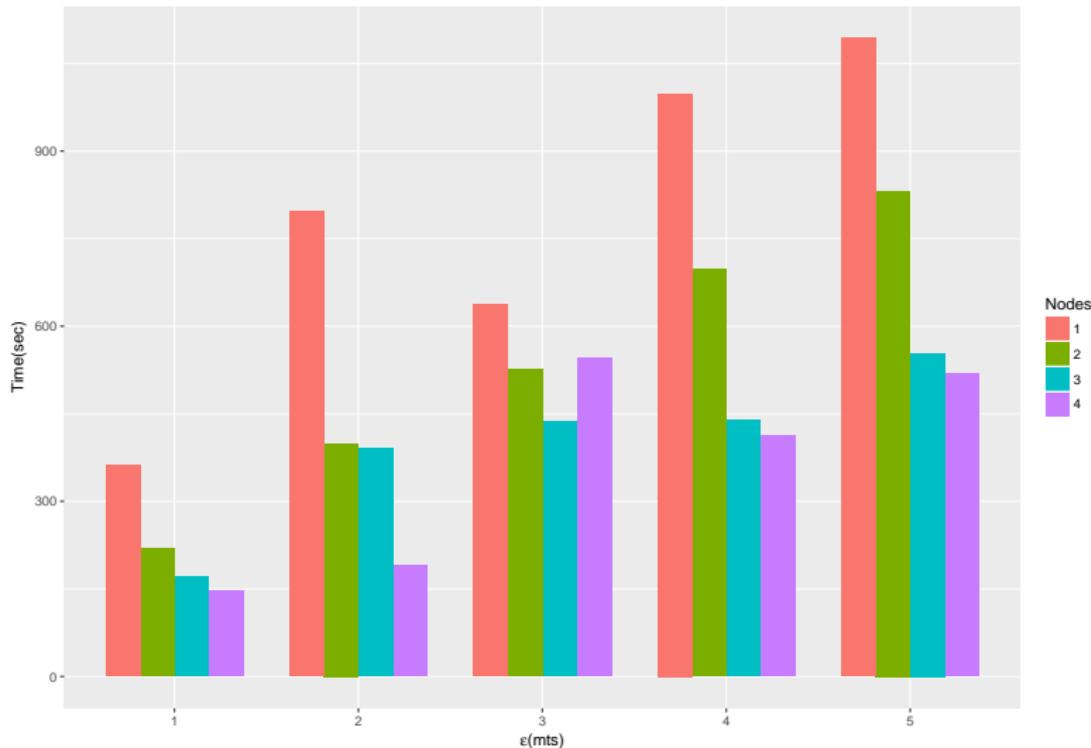
- ▶ 4-node cluster at DBLab.
- ▶ Processors: 8-core Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
- ▶ RAM: 15.5 GB.
- ▶ Centos 6.8, Simba/Spark 1.6.3.

# Execution time

Execution time by  $\epsilon$  (radius of disk in mts) in Cologne dataset.

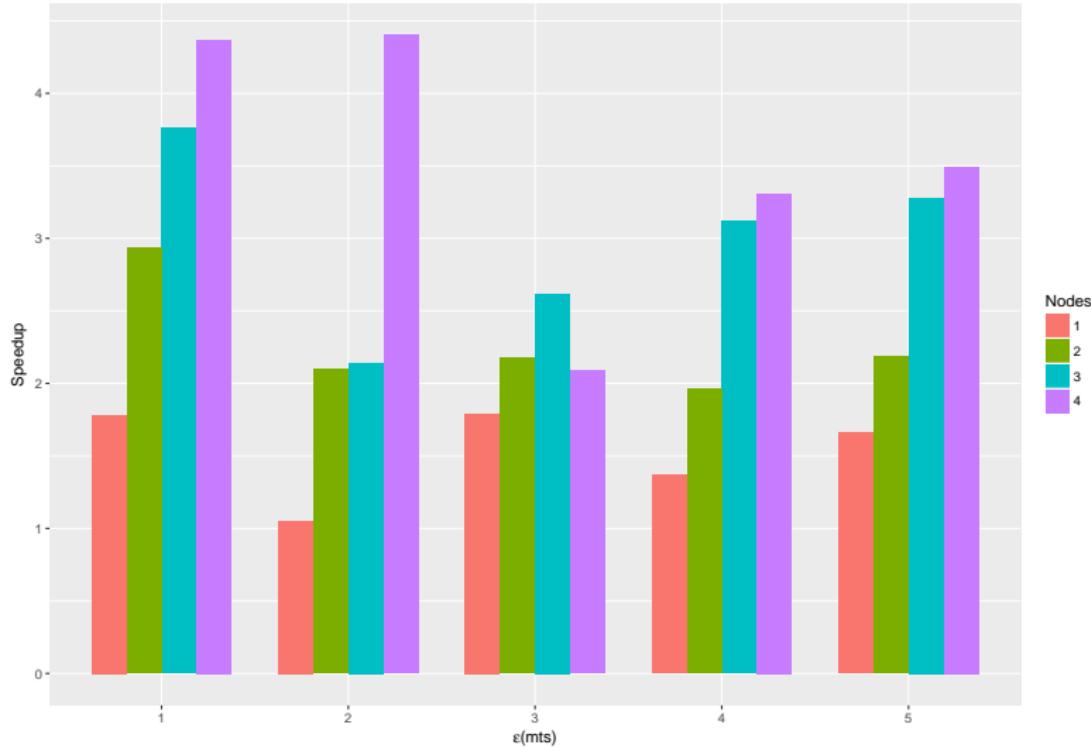
# Scaleup

Scaleup in Cologne dataset.



# Speedup

Speedup in Cologne dataset.



# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

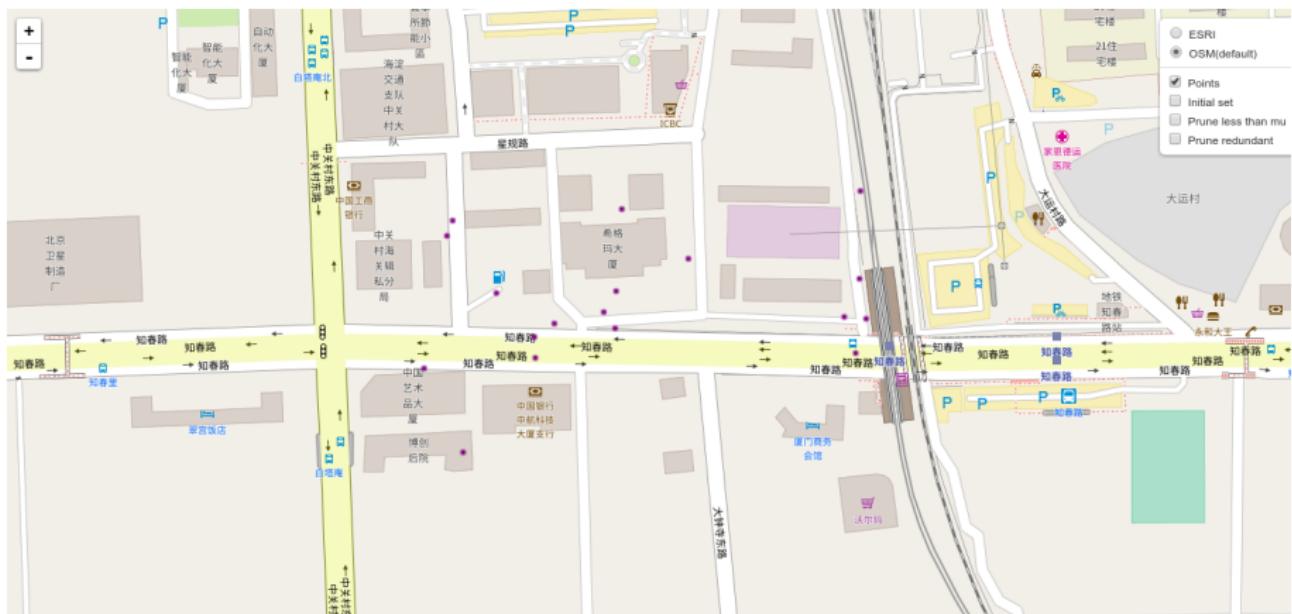
Implementation

Conclusions and future work

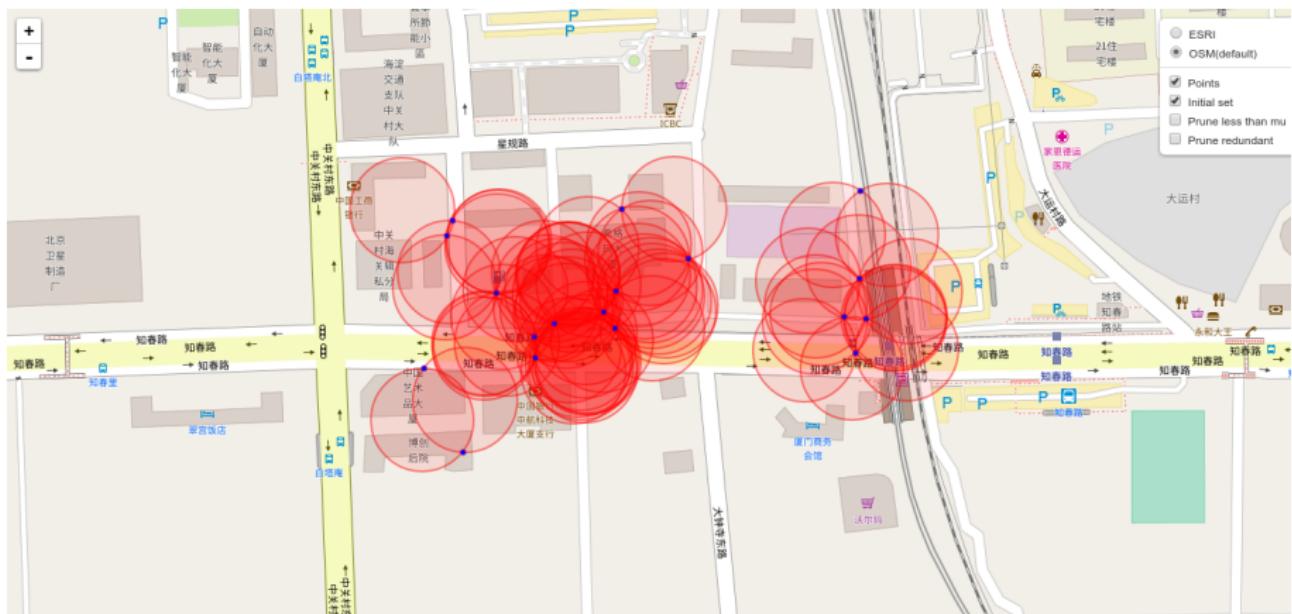
# Finding maximal disks

- ▶ The first stage of BFE finds all possible disk locations for each pair of “close enough” points...
- ▶ This candidate set still have redundant and under-the-threshold ( $\mu$ ) disks...
- ▶ BFE uses a simple approach to filter the disks but it can be costly ( $O(n^2)$ )

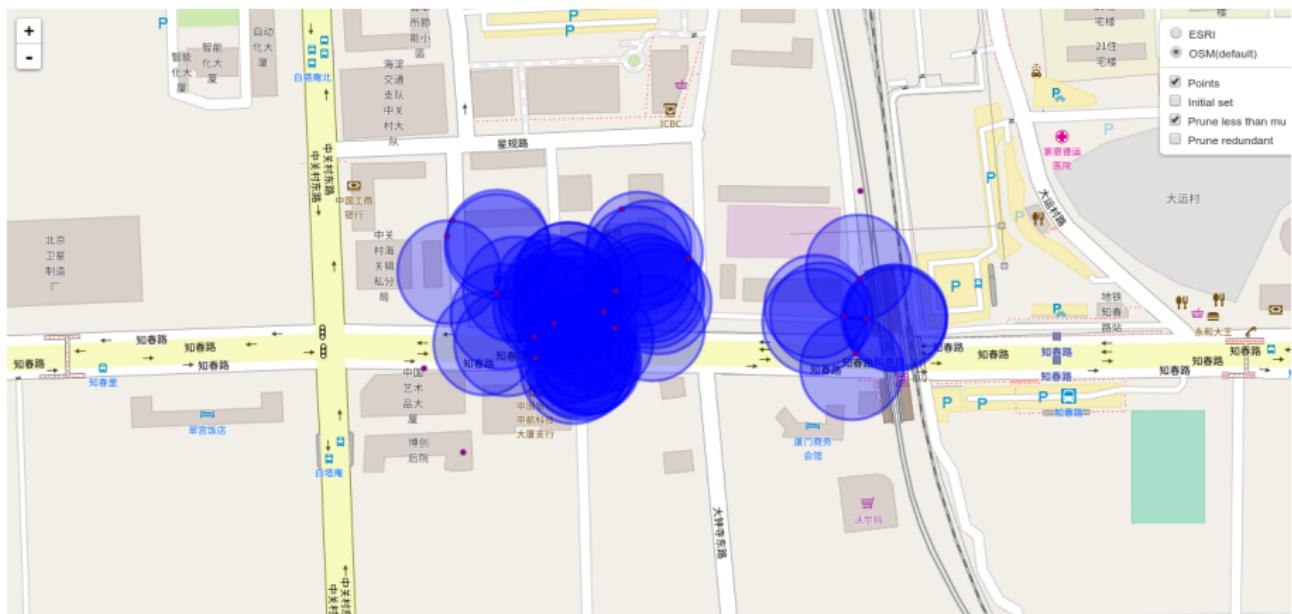
# Finding maximal disks



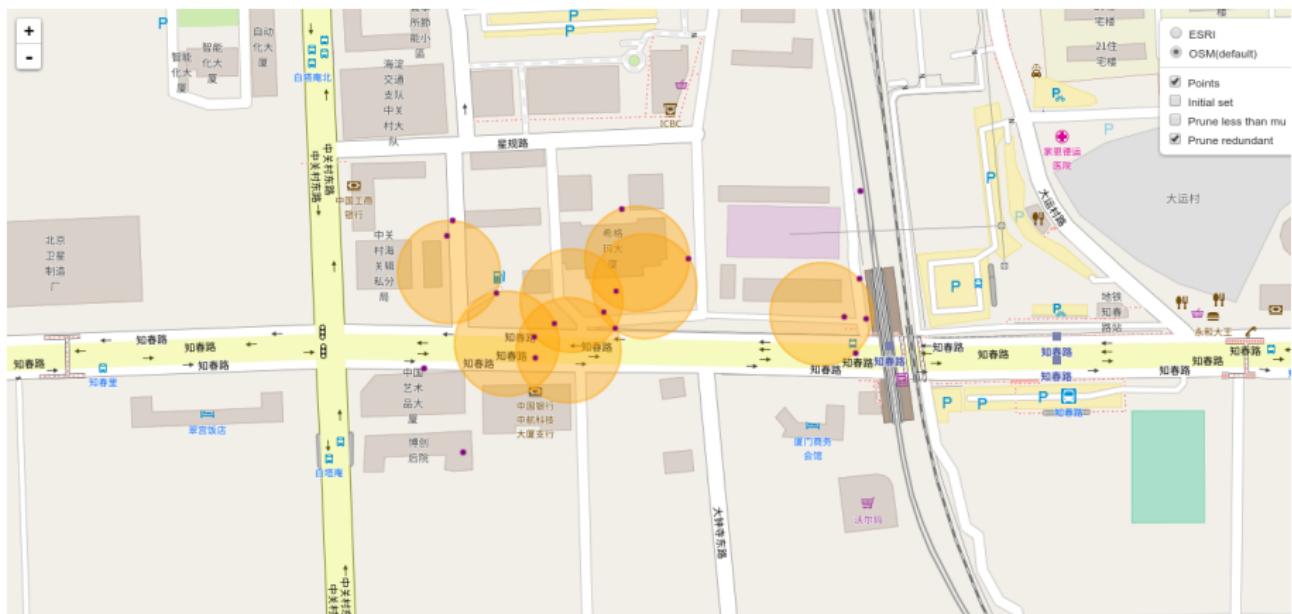
# Finding maximal disks



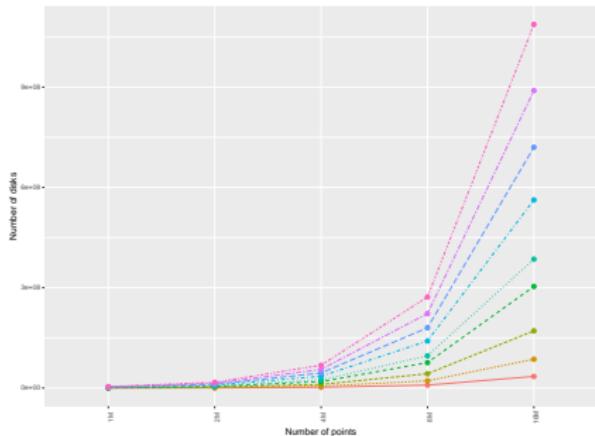
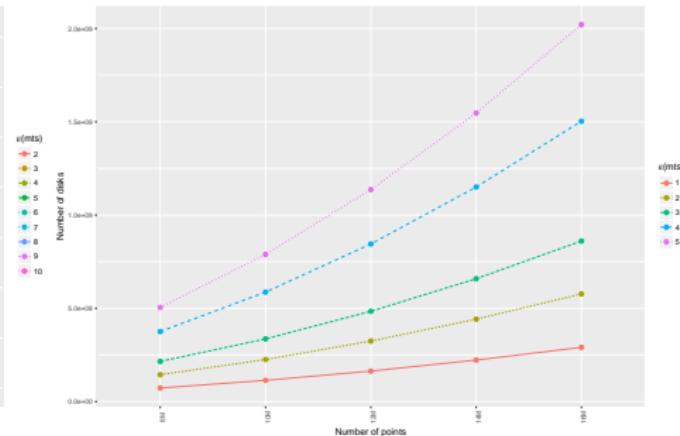
# Finding maximal disks



## Finding maximal disks



# Number of disks can be huge...

Number of disks by  $\epsilon$  (radius of disk in mts) in Porto dataset.Number of disks by  $\epsilon$  (radius of disk in mts) in Cologne dataset.

# Finding maximal disks using Maximal Pattern algorithms

- ▶ Points enclosed by disks become transactions...
- ▶ Apply well-known algorithms (LCM in this case)...
- ▶ Filter patterns with length less than  $\mu$ .

# A Frequent/Closed/Maximal pattern example...

Suppose a database  $D$  contains 4 transactions:

$$D = \{\langle a_1, a_2, \dots, a_{100} \rangle; \langle a_1, a_2, \dots, a_{100} \rangle; \langle a_{20}, a_{21}, \dots, a_{80} \rangle; \langle a_{40}, a_{41}, \dots, a_{60} \rangle\}$$

With  $\text{min\_sup} = 2$ :

- ▶  $F = \approx 1.27 * 10^{30}$
- ▶  $C = \{\{a_1, a_2, \dots, a_{100} : 2\}; \{a_{20}, a_{21}, \dots, a_{80} : 3\}; \{a_{40}, a_{41}, \dots, a_{60} : 4\}\}$
- ▶  $M = \{\{a_1, a_2, \dots, a_{100} : 2\}\}$

# Outline

Moving flock patterns

Finding candidate disks

Implementation

Experiments

Finding maximal disks

Implementation

Conclusions and future work

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Conclusions

- ▶ An implementation of a parallel method to detect a maximal set of disks for the BFE algorithm has been presented.
- ▶ The method proves to be scalable and reliable.
- ▶ Execution time improves up to 3 orders of magnitude compared to the sequential code.
- ▶ Good behavior of Scaleup and Speedup metrics.

# Future work

- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.

## Future work

- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.

## Future work

- ▶ Work on a parallel strategy to join the sets of valid disks between time intervals.

# Thank you!!!

Do you have any question?