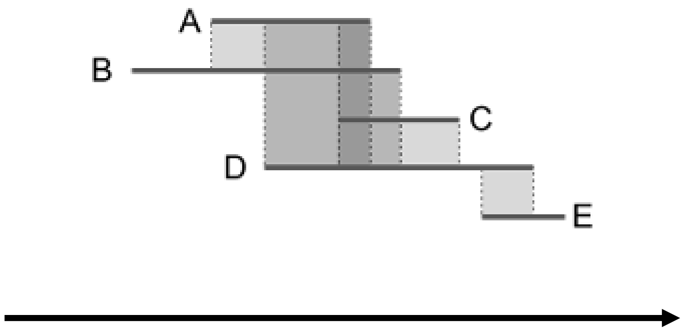
Mining and Exploration of Multiple Intersecting Axis-aligned Objects

Master's Thesis

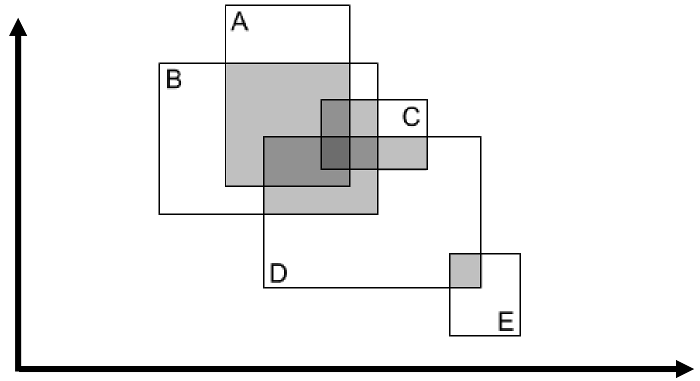
Tilemachos Pechlivanoglou

Supervisor:
Manos Papagelis

Axis-aligned objects

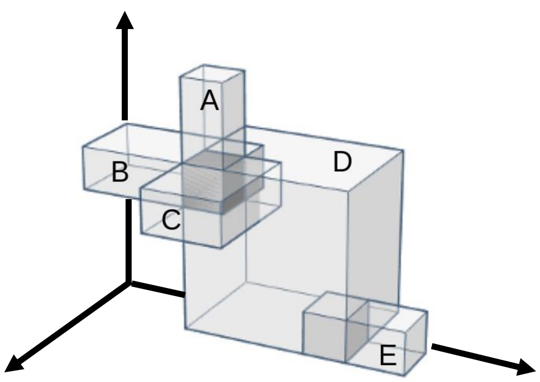


1-D line segments/intervals



2-D rectangles

Regions



3-D boxes/cuboids

Multidimensional

Object intersection problem

Object intersection problem

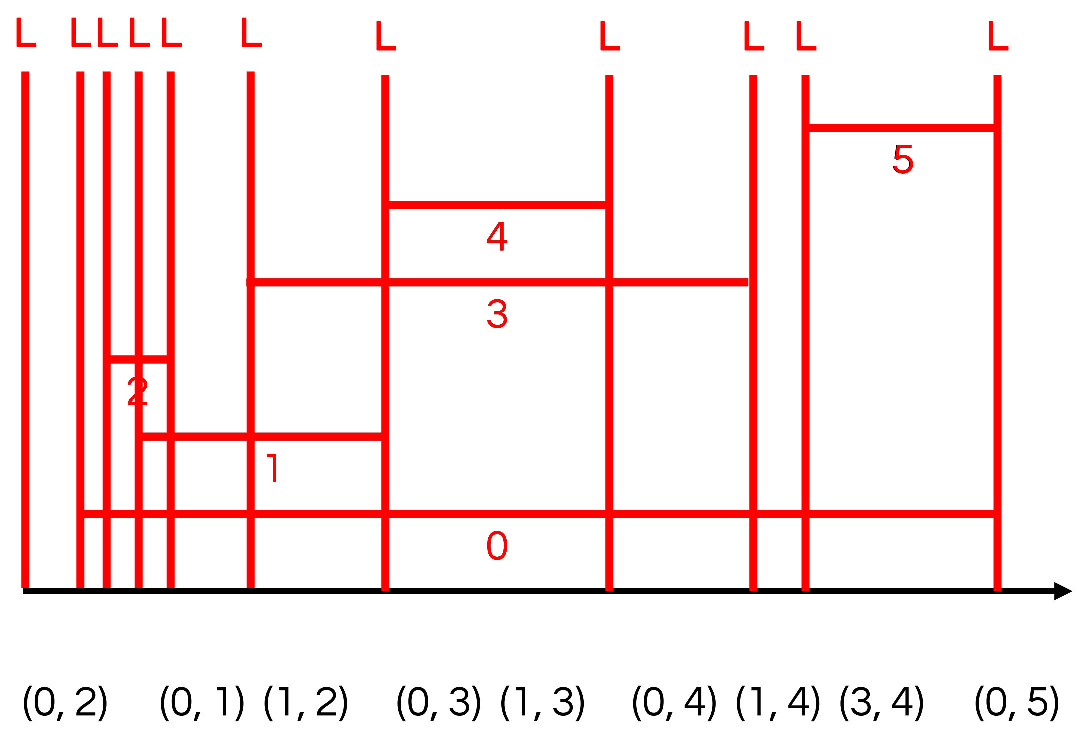
Input:

- a set of axis aligned geometric objects

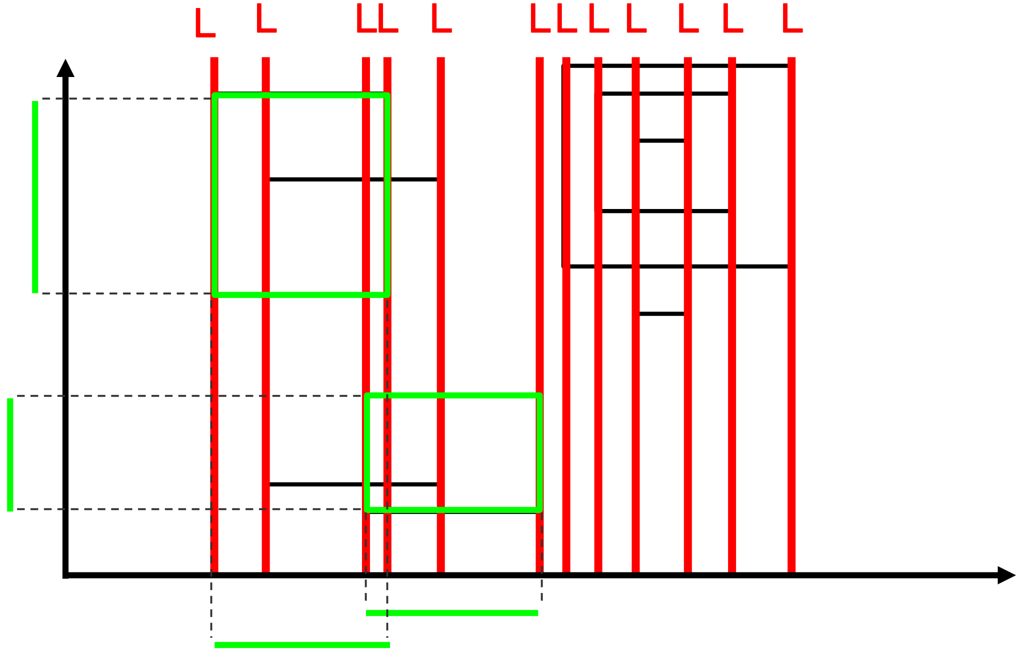
Output:

- which pairs of objects intersect
- how much

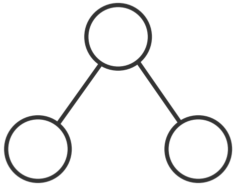
Sweep-line algorithm (1-D)



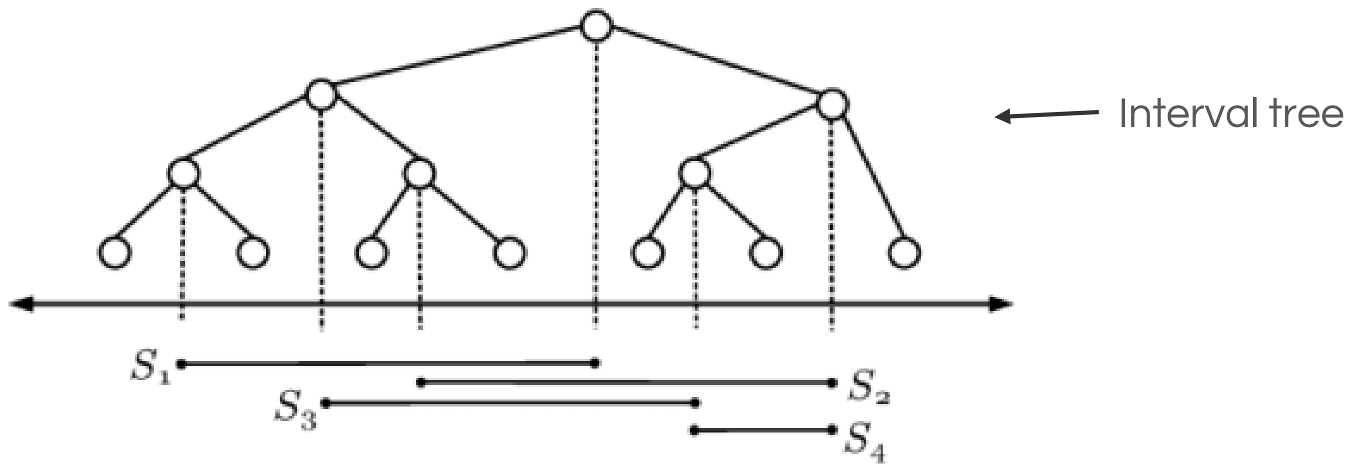
Sweep-line algorithm (2-D)



Interval tree:



Divide-and-conquer algorithm



Computationally **equivalent** to Sweep-Line

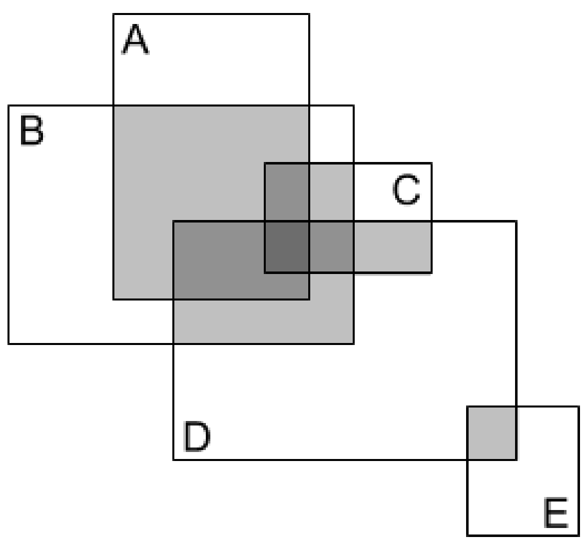
Multiple Intersecting Objects

Research questions

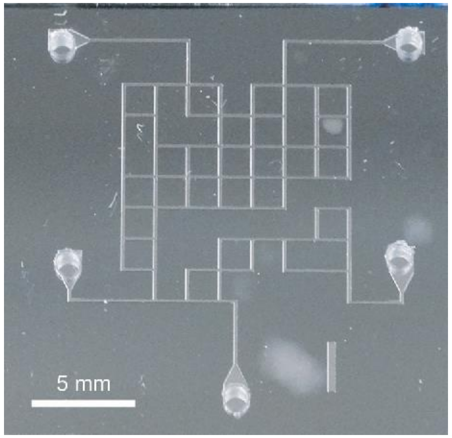
How to detect **multiple** intersecting objects?

What is the **size** of their overlap (**common region**)?

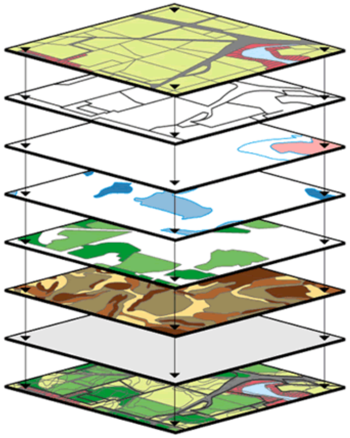
Where is that common region **located**?



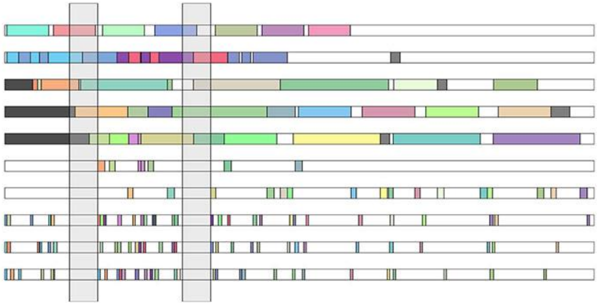
Applications



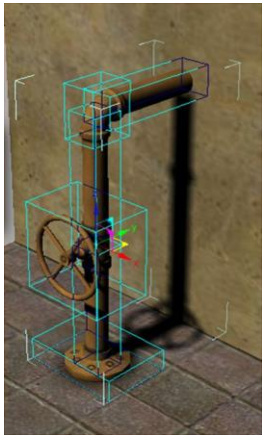
Circuit design



Spatial databases



Task scheduling



Simulations

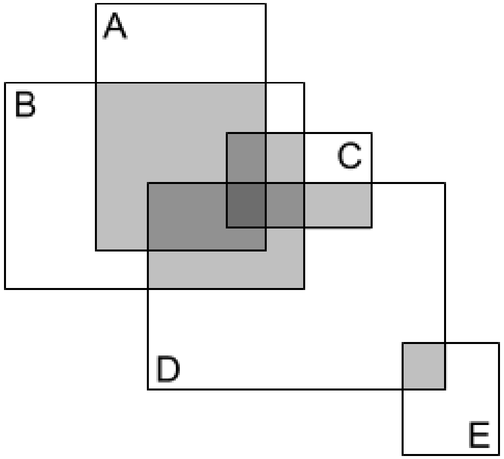
The problem

Input:

- a set of regions in R^d:

Output:

- enumeration of all intersecting sets of regions
- size and position of each common region



Sets:

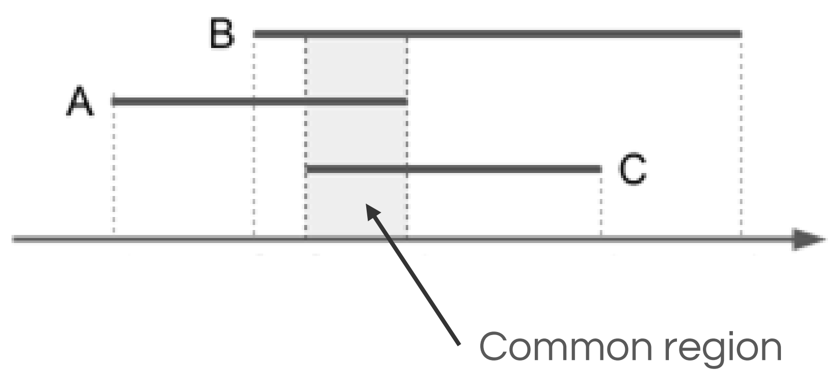
A,B	A,B,C
A,C	A,B,D
A,D	B,C,D
B,C	
B,D	A,B,C,D
C,D	
D,E	

Multiple Intersection Calculation

Common region

A set of 3 or more objects, **all intersecting pair-wise** with each other have a non-empty **common region**.

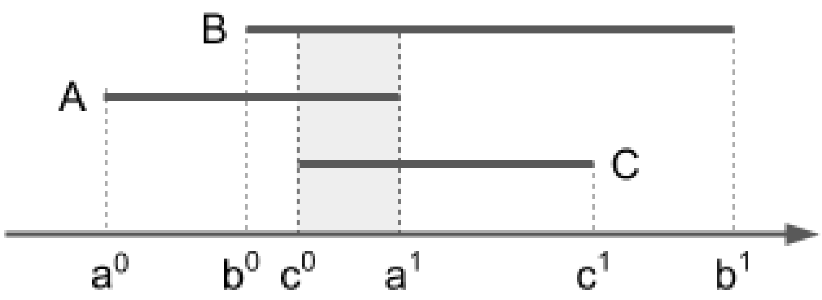
(Helly's theorem, convex sets)



Common region size: 1-D

For a fully intersecting set I, the **common region length** |Z| is:

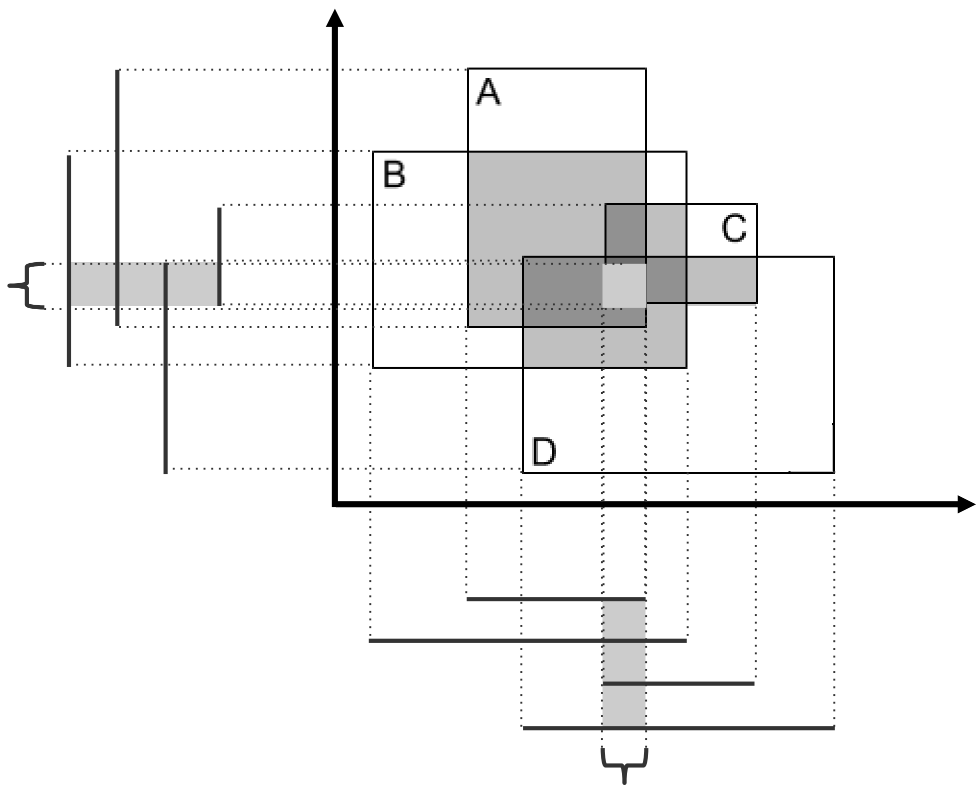
$|Z_I|$ = max(start points) - min(end points)



$|Z_{ABC}|$ = max(a^0, b^0, c^0) - min(a^1, b^1, c^1) = a^1 - c^0

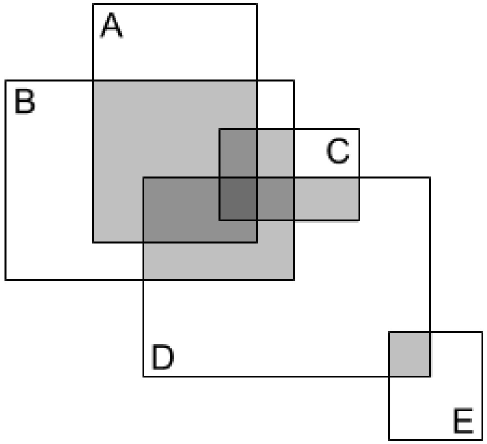
Common region size: 2-D, 3-D …

For more dimensions, $|Z|$ is the **product** of the common region **lengths** in **each dimension** $|Z_d|$



Intersection cardinality (k)

The **number** of simultaneously overlapping objects in a set



$k_{ABCD} = 4$
$k_{DE} = 2$

$k_{AE} = 0$
$k_{ABCDE} = 4$

Sensible baseline algorithms

Naive approach

1. Compare **each object** with **every other**
2. If any 2 intersect, compare the **pair's** common region with **every other object**
3. If any 3 intersect, compare the **triplet's** common region with **every other object**
4. **Repeat** until no intersections found or no objects left

- many nested loops
- very high computational cost

Modified sweep-line approach

1. Execute **sweep-line** algorithm to find intersecting pairs
2. Get the common regions of all resulting **pairs**
3. Execute **sweep-line** on them to find **triplets**, **quadruplets**
4. **Repeat** until no intersections found

- better performance than naive

Limitations

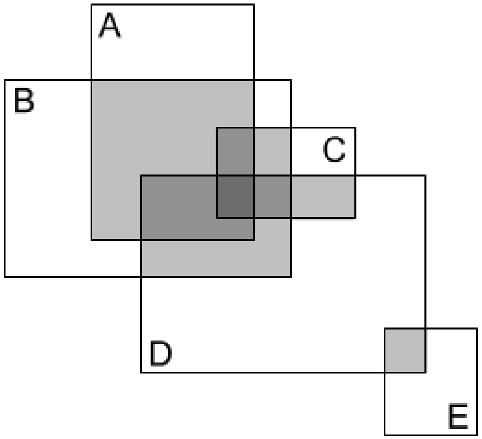
- High computational cost
- Difficult to implement
- Lack of versatility
  - different implementations needed for different problems
  - hard to process/explore specific part of dataset

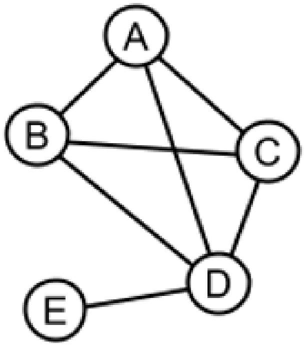
Our approach (SLIG)

Region intersection graph

A **graph** data structure where:

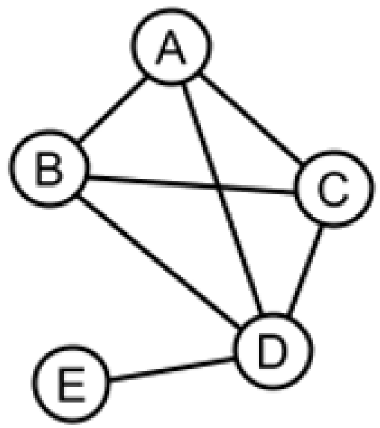
- Each **vertex** corresponds to an **object**
- An **edge** exists between two vertices if the corresponding objects **intersect**





Clique

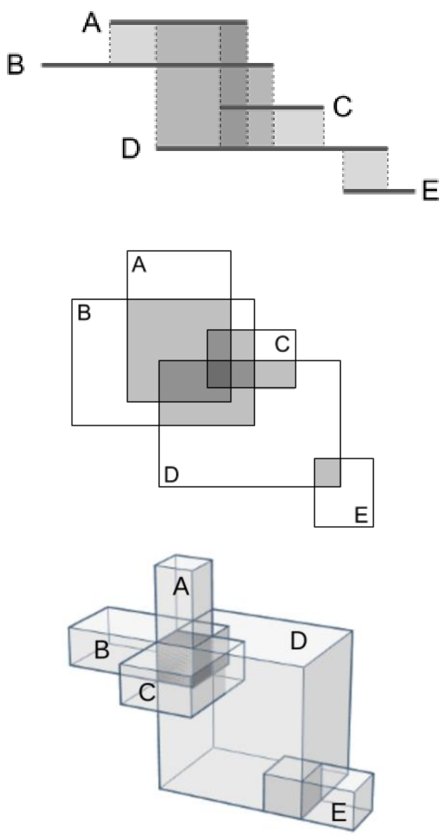
Subset of vertices where every two are connected (i.e. a fully connected subgraph)

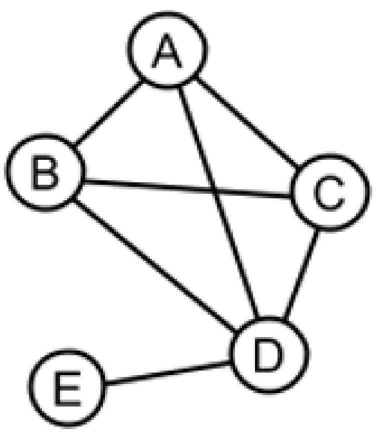


size-3 cliques: ABC, ABD, ACD, BCA
size-4 clique: ABCD (**maximal clique**)

Observation

On an intersection graph, a **clique** corresponds to a **fully intersecting set** with a **common region**





Sweep-Line with Intersection Graph (SLIG)

1. Execute **sweep-line** algorithm to find intersecting pairs
2. Use **pairs** to construct the intersection graph
3. Execute a **clique enumeration** algorithm on graph

- best performance
- using established, efficient clique enumeration methods
- much easier to implement

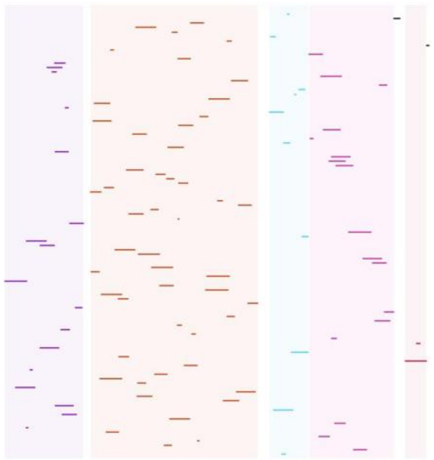
Extensions: Querying capability

The intersection graph provides **additional mining options**, such as exploration using **queries**:

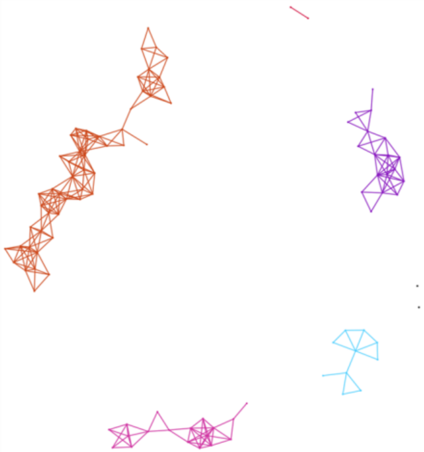
- **Single Region Query**: given an object find all other objects intersecting with it
- **Multiple Region Query**: given a set of objects, find all intersections occuring in the set

Multiple Intersections Evaluation

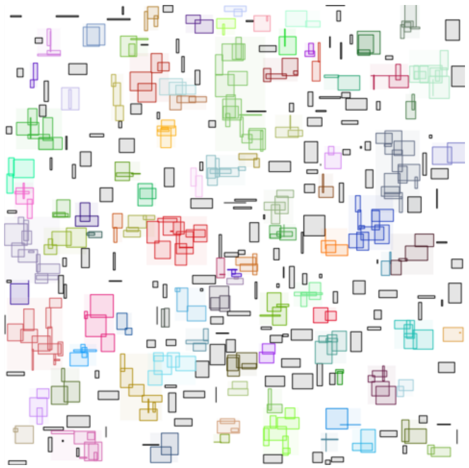
Randomly generated objects



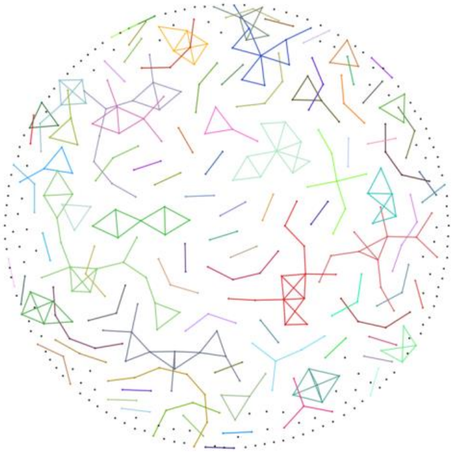
1-D intervals



1-D intersection graph

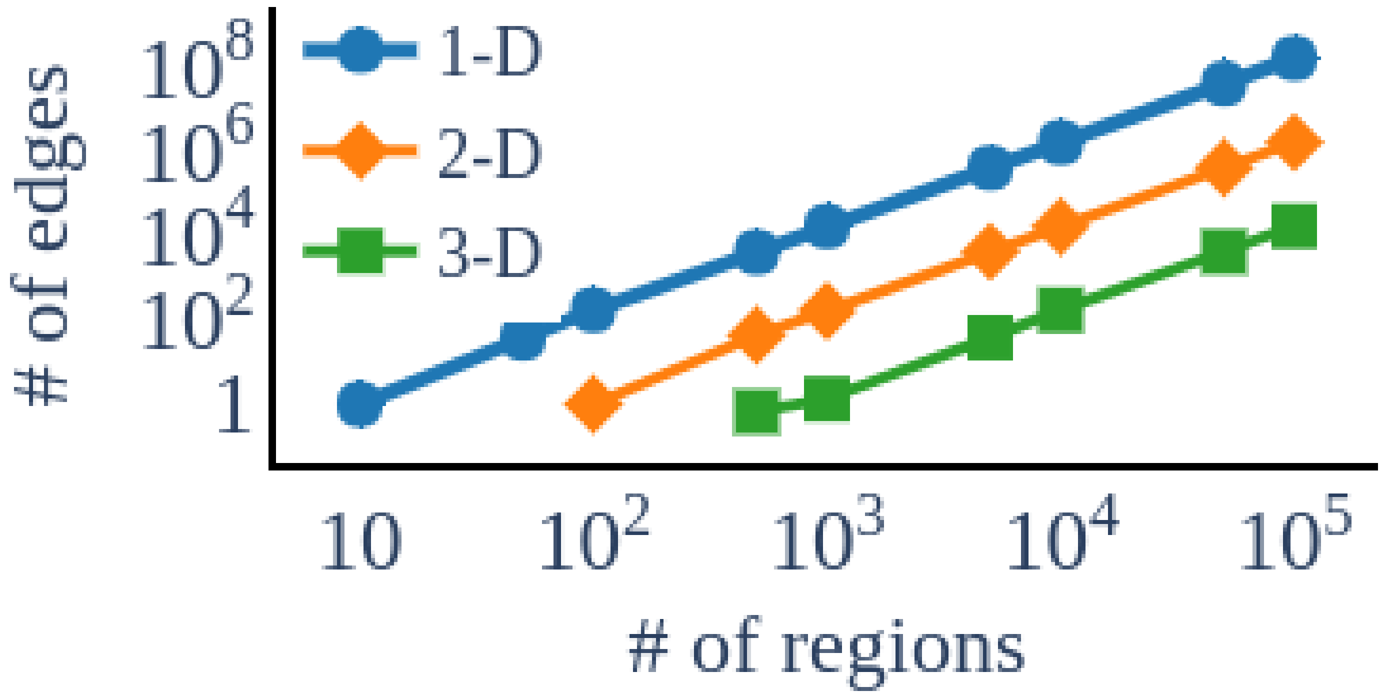


2-D rectangles

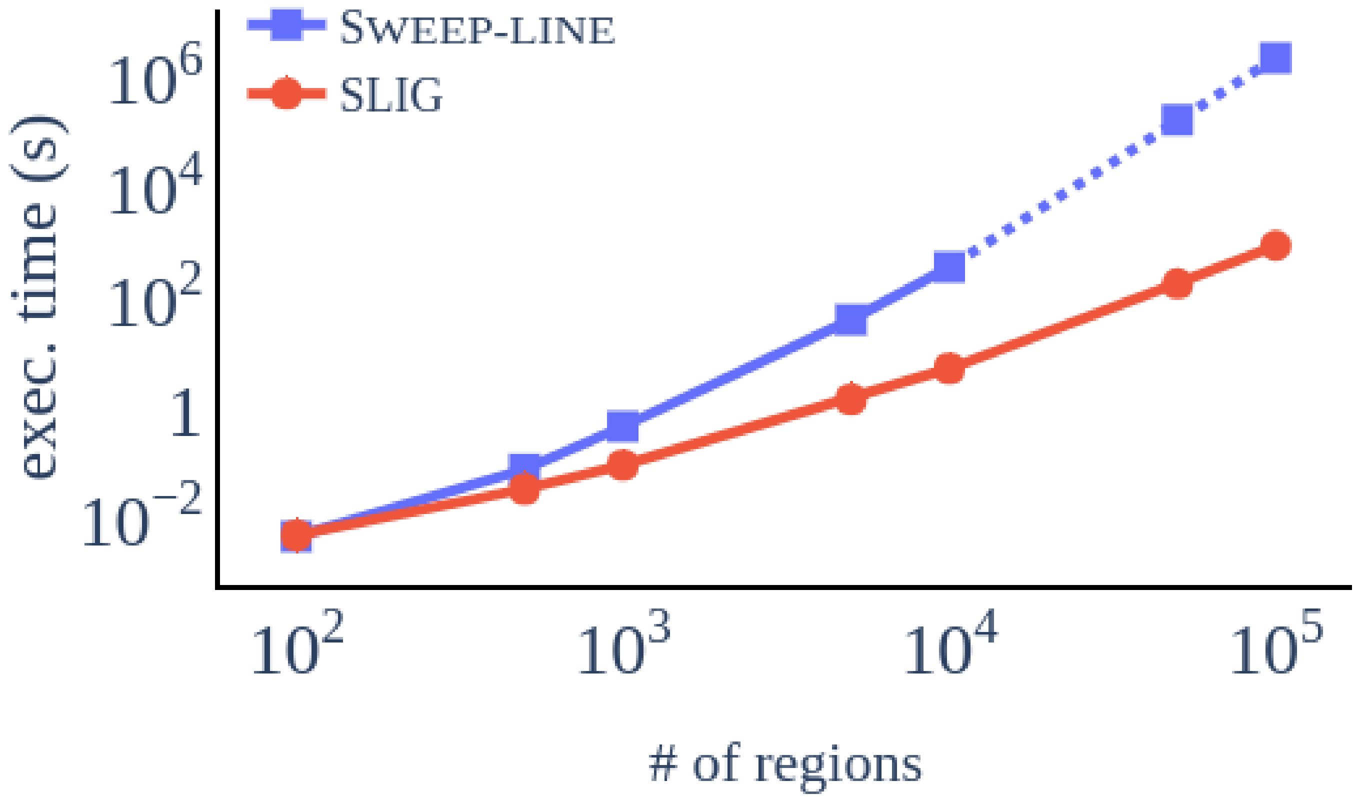


2-D intersection graph

Intersection graph size

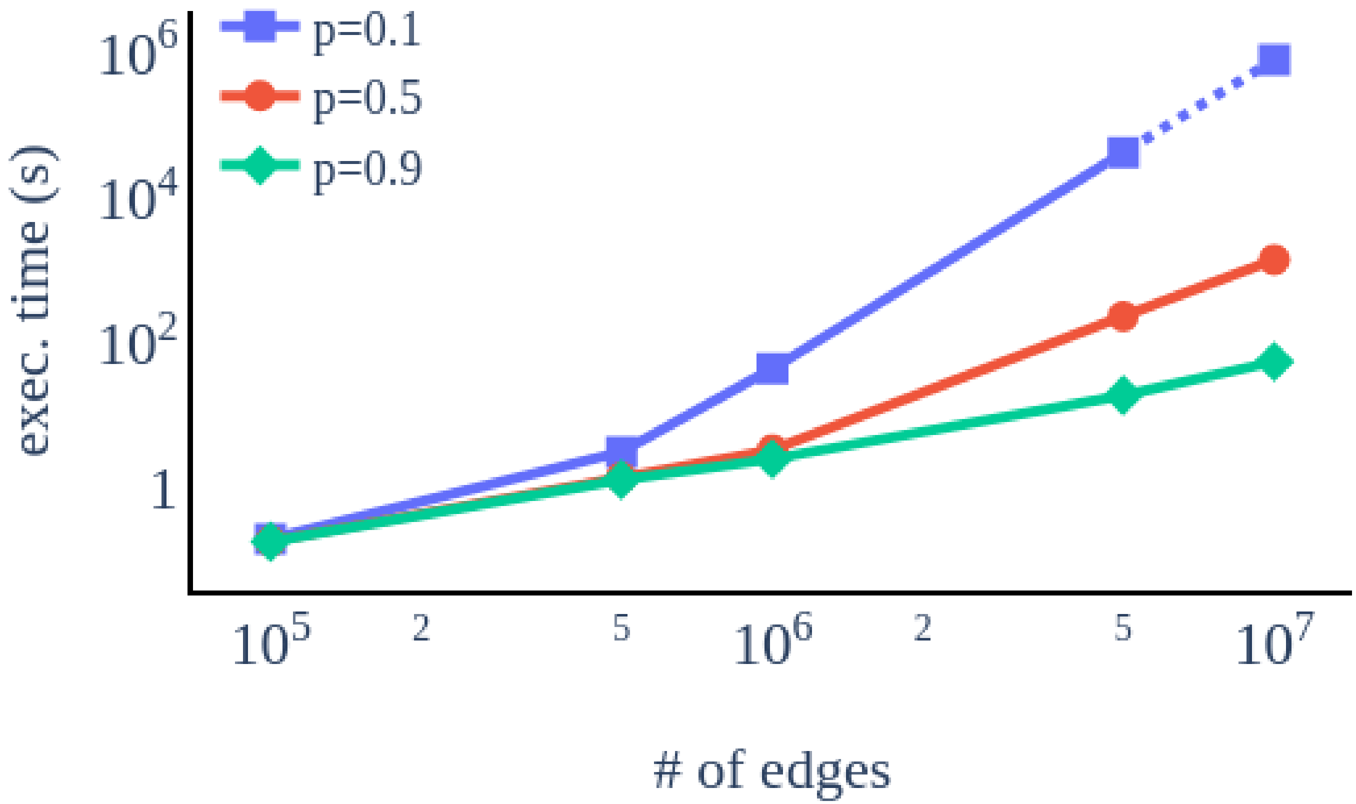


Performance of SLIG



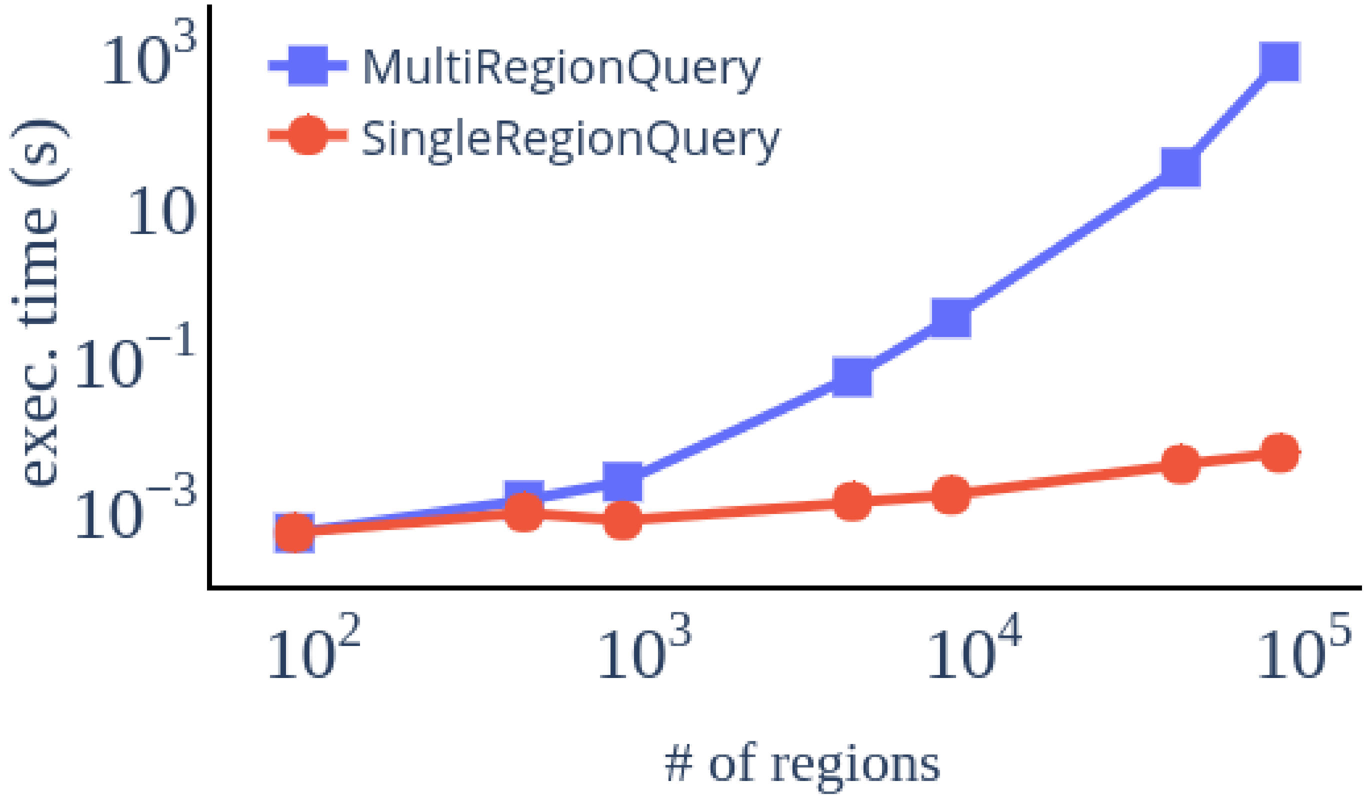
SLIG scales much better than baseline

Effect of graph topology

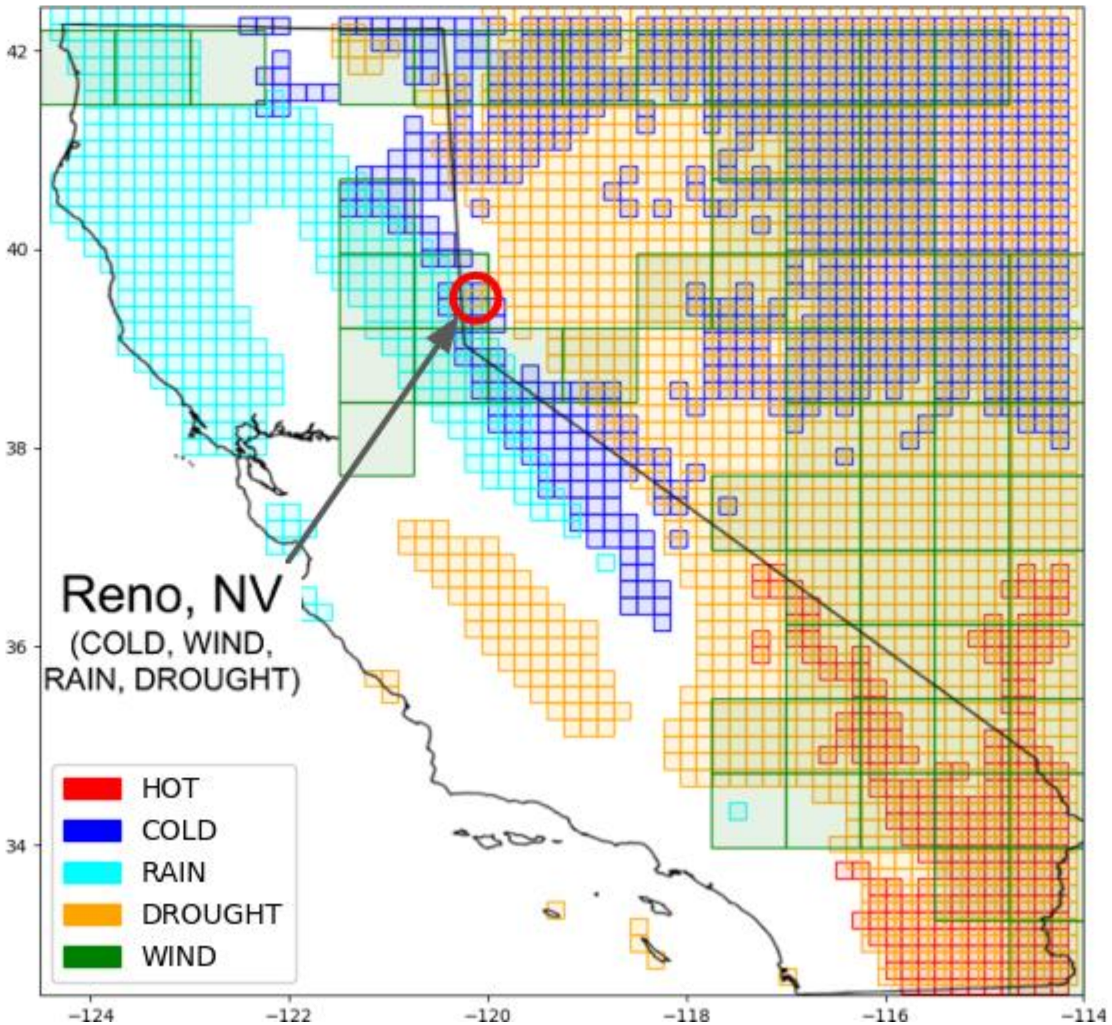


smaller/sparser objects -> sparser graphs -> faster execution

SLIG query performance



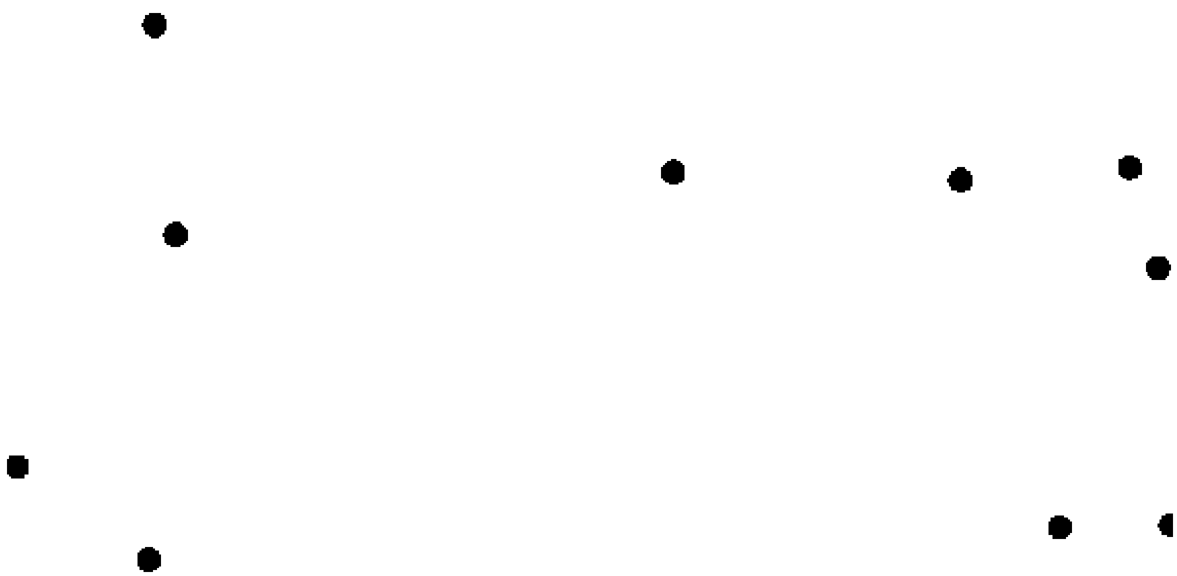
Real-world data



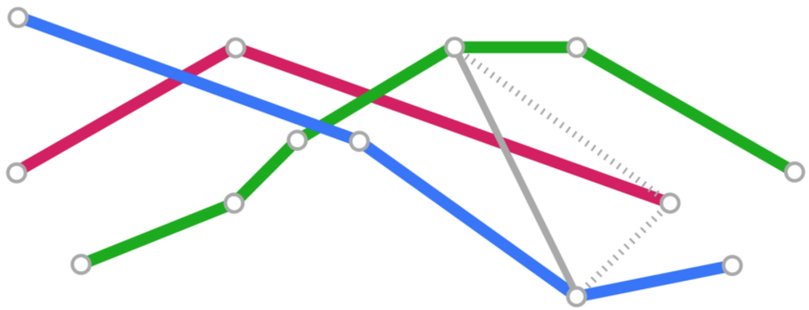
Overlapping areas of extreme weather in CA & NV, USA

Node Importance in Trajectory Networks

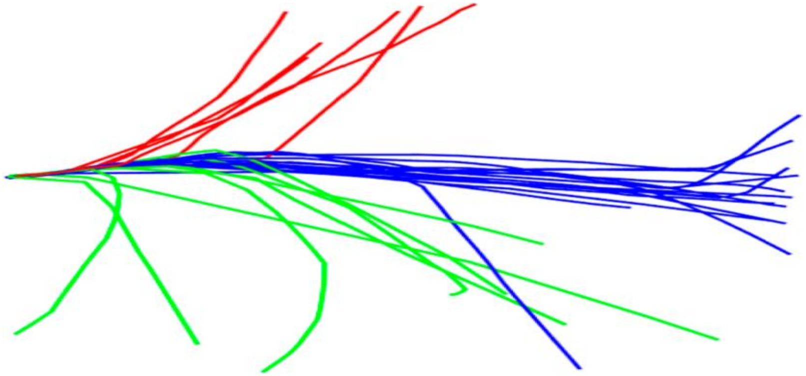
Trajectories of moving objects



Trajectory Mining



Trajectory similarity

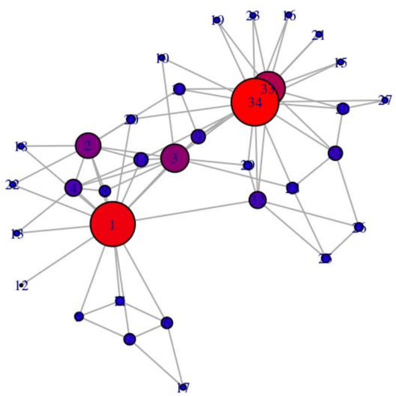


Trajectory clustering

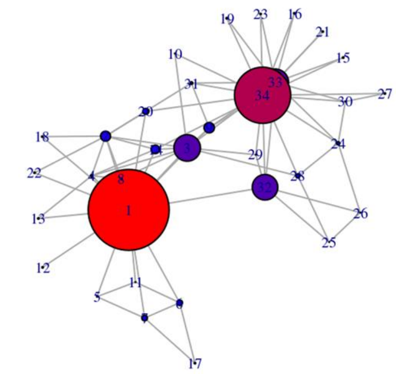
Trajectory anomaly detection
Trajectory pattern mining
Trajectory classification
...more

Node Importance

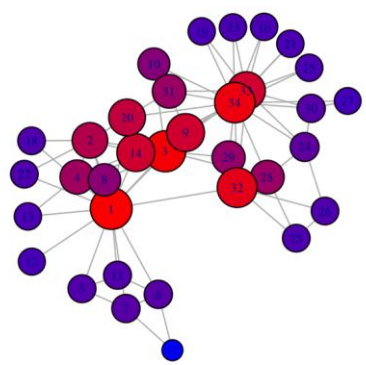
Node importance (or centrality)



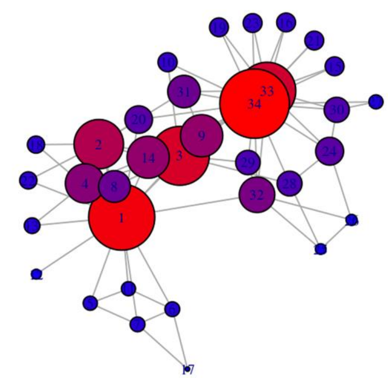
Degree centrality



Betweenness centrality

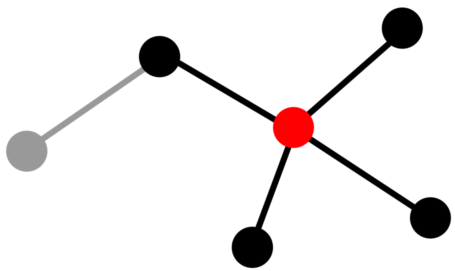


Closeness centrality

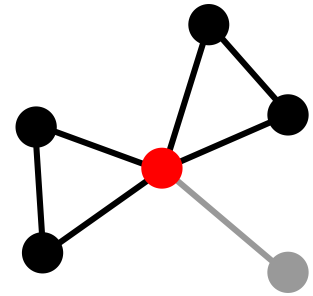


Eigenvector centrality

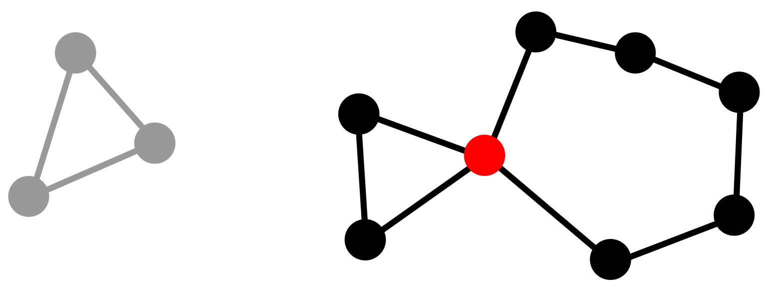
Over time



Node degree **over time**

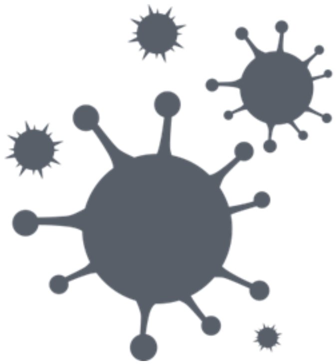


Triangles **over time**



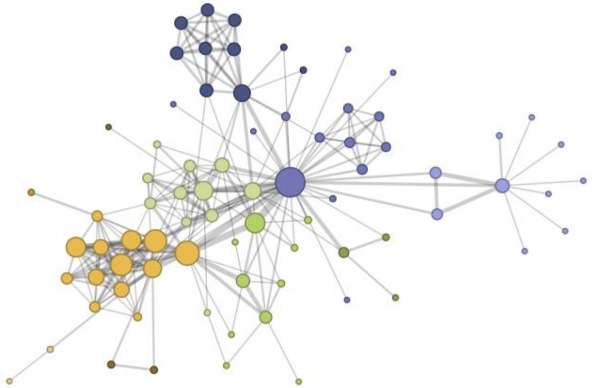
Connected components **over time**
(connectedness)

Applications



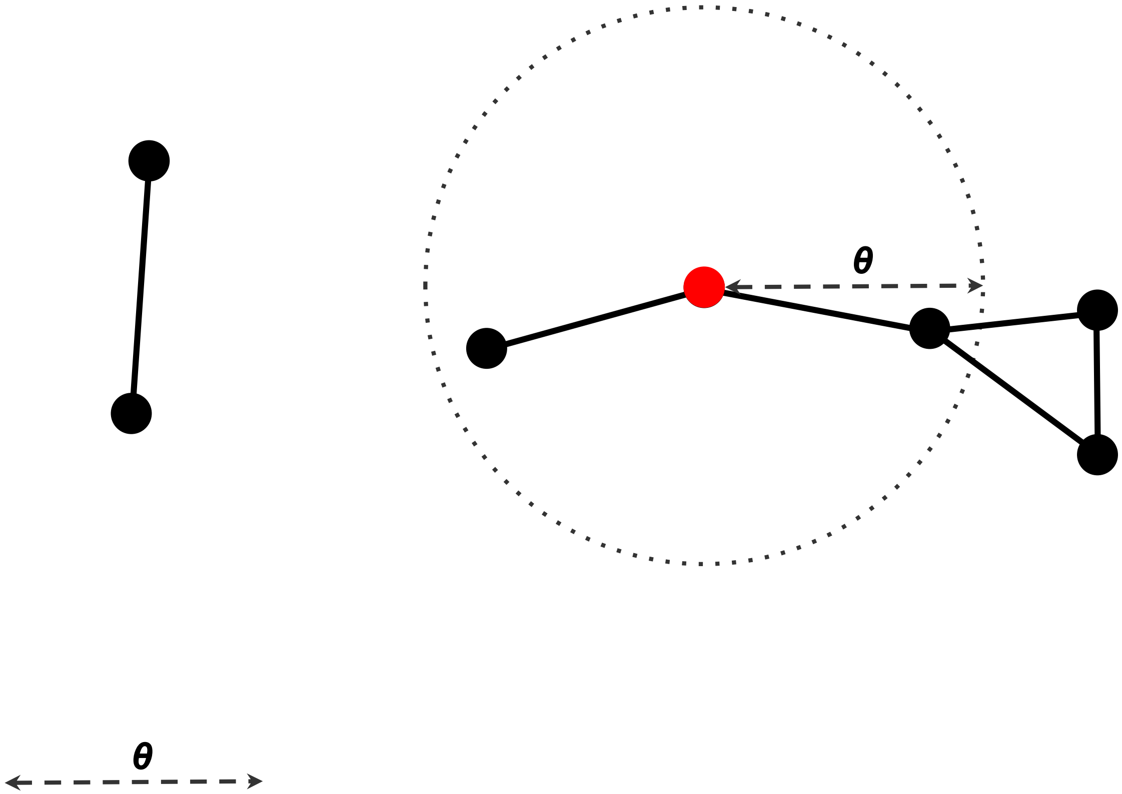
Infection spreading

Wireless signal security

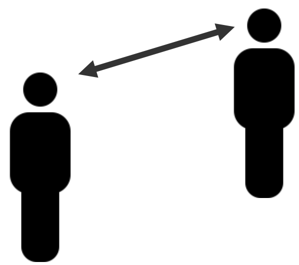


Rich dynamic network analytics

Proximity networks



Distance can represent

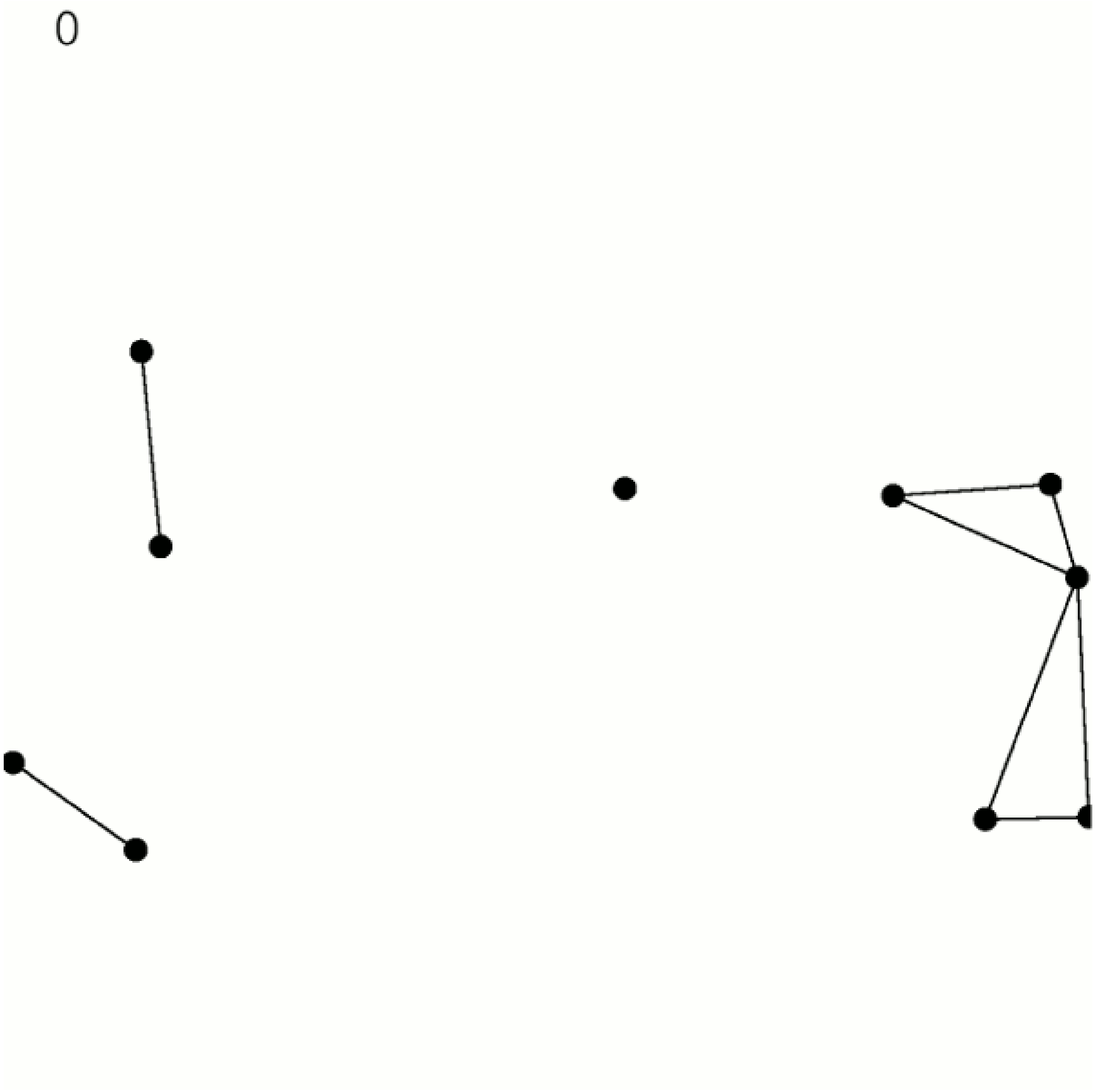


line of sight

Wifi signal range

travel distance in a day

Trajectory networks



Problem difficulty

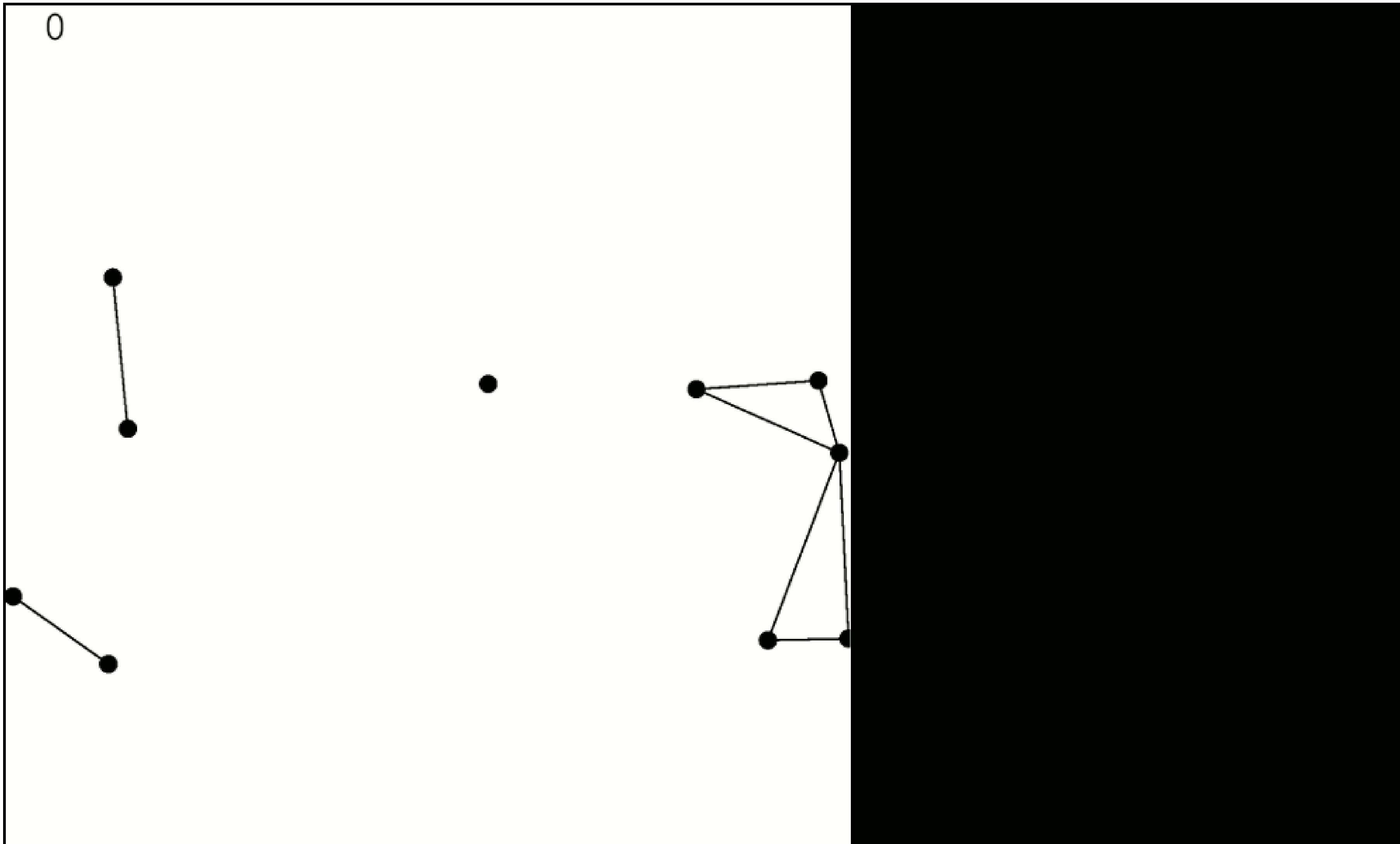
Node importance algorithms for **static networks**

Sequence of static networks (**snapshots**)

One **large** network **per** discrete **time unit**!

Node Importance in Trajectory Networks

Naive approach



Naive approach

For every discrete time unit:

1. get static snapshot of network
2. run static node importance algorithms on snapshot

Aggregate results at the end

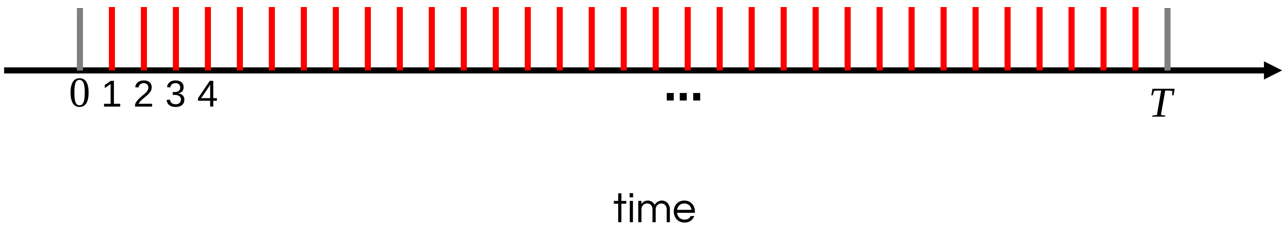
Streaming approach

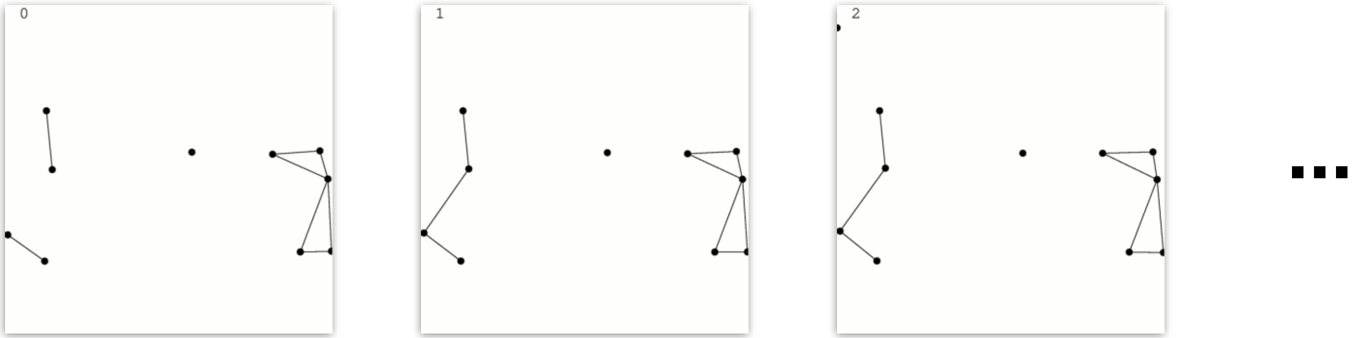
Similar to naive, but:

- no final aggregation
- results calculated iteratively at every step

Still every time unit

Every discrete time unit





Sweep Line Over Trajectories (SLOT)

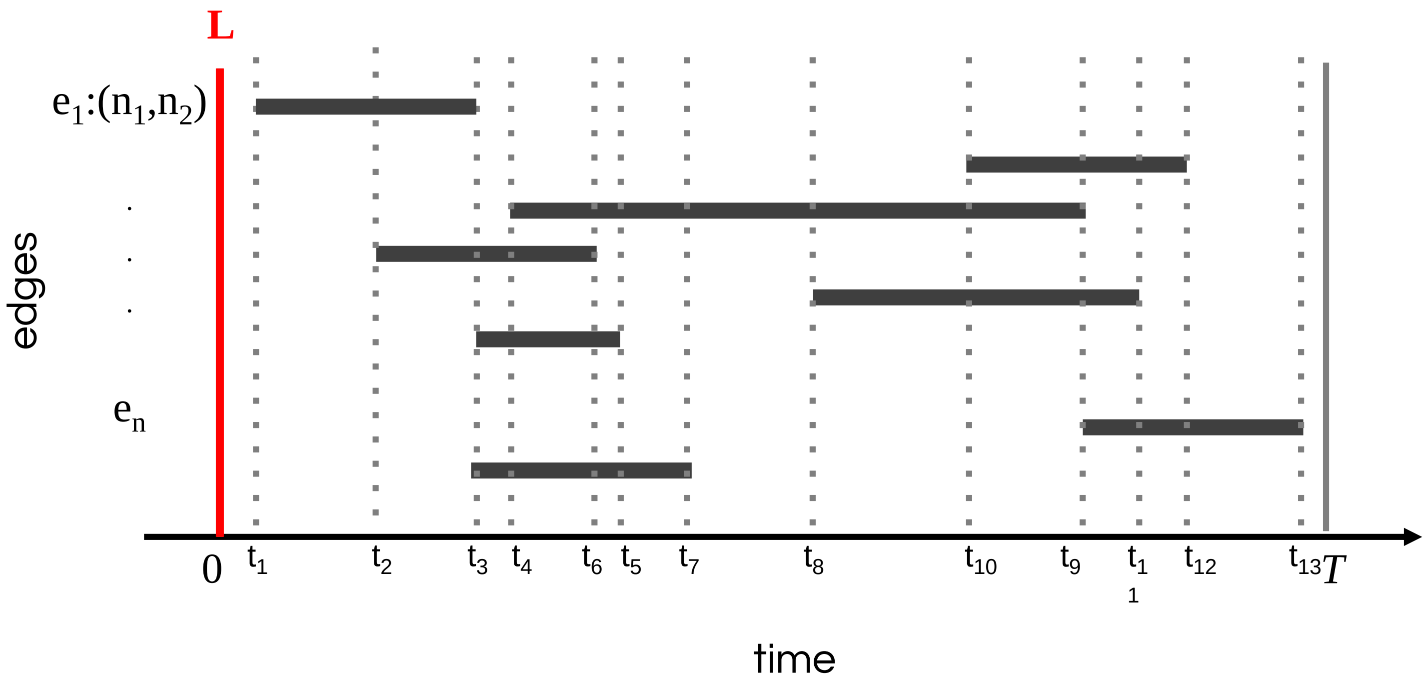
(**algorithm sketch**)

represent TN **edges** as **time intervals**

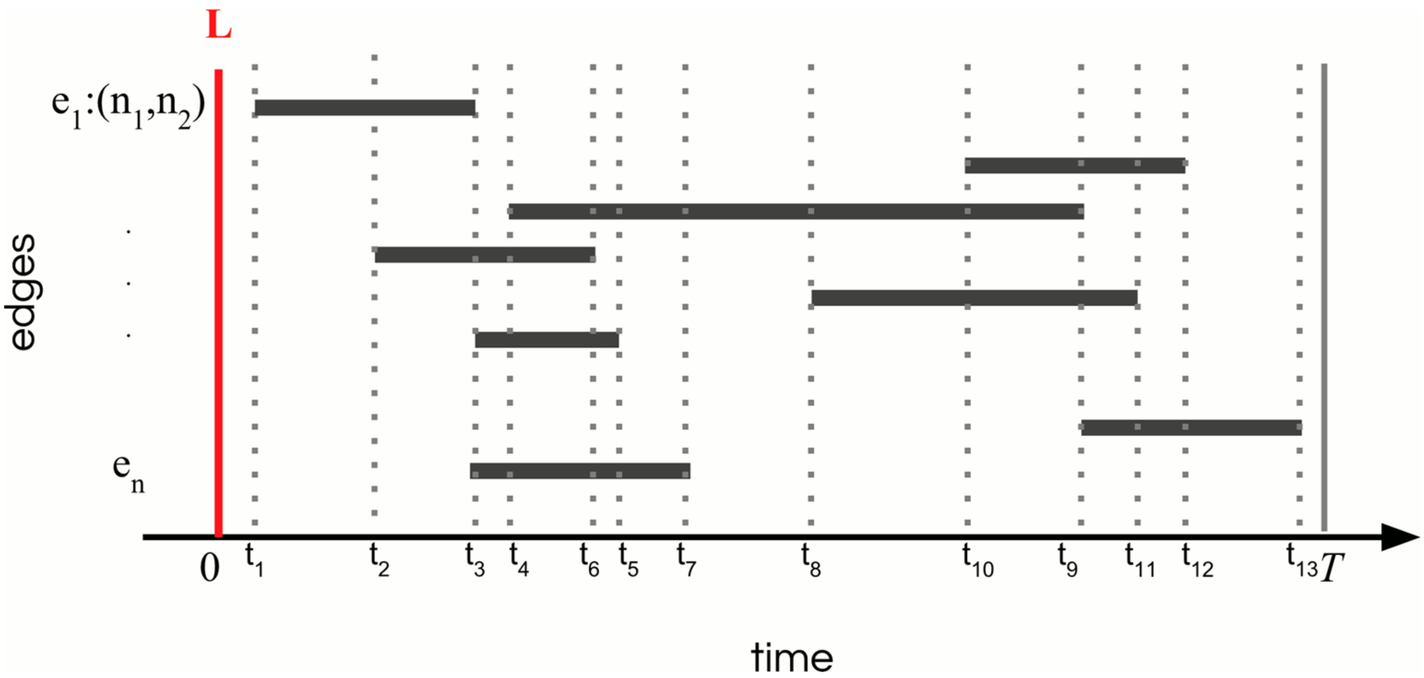
apply variation of **sweep line** algorithm

simultaneously compute *node degree, triangle membership, connected components* in **one pass**

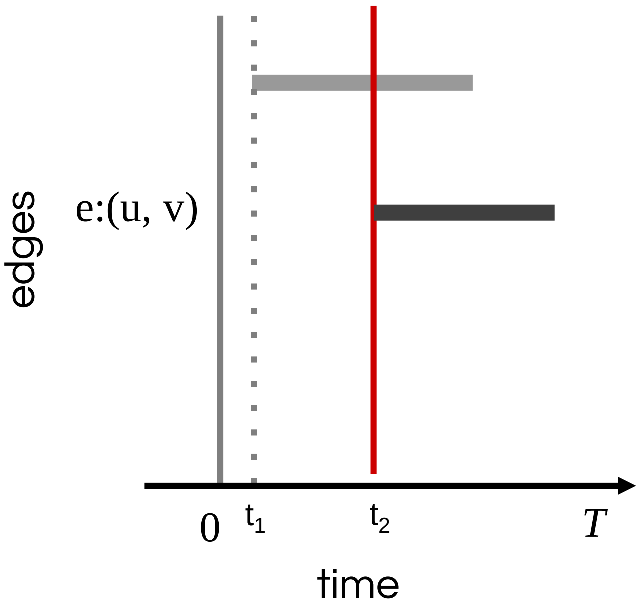
Edges as time intervals…

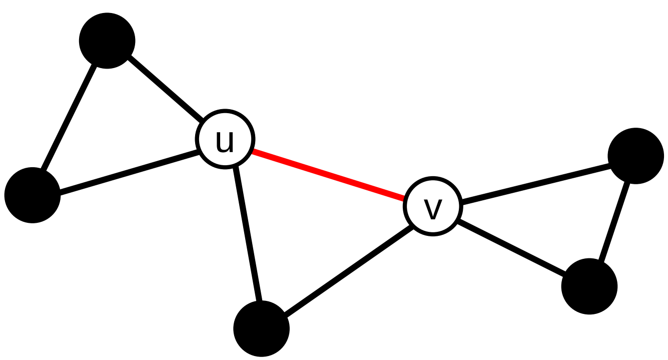


Sweep Line Over Trajectories (SLOT)



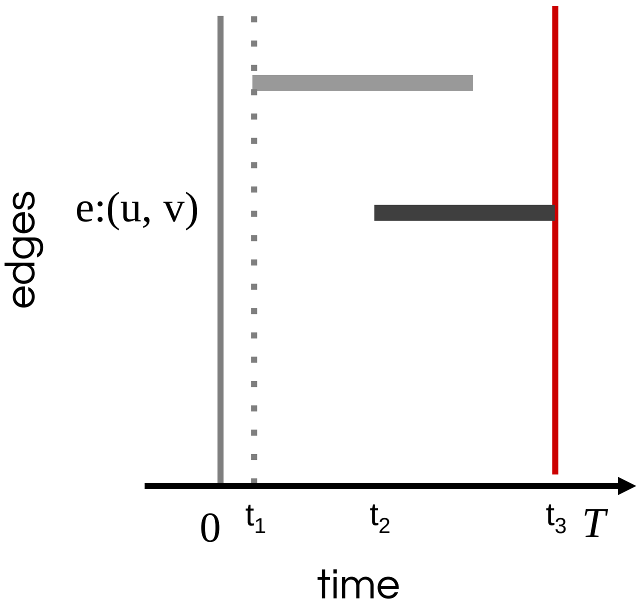
At every edge start

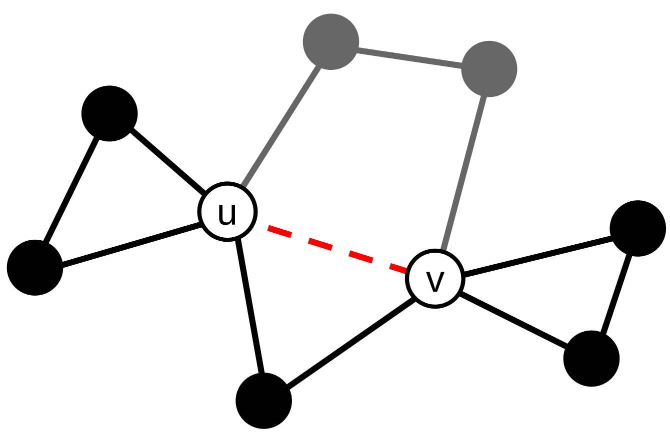




- **Degree**
  - nodes u, v now connected
  - increment u, v degree
- **Triangles**
  - did a triangle just form?
  - look for u, v common neighbors
  - increment triangle (u,v,common)
- **Components**
  - did two previously unconnected components connect?
  - compare old components of u, v
  - if not same, merge them

At every edge stop





- **Degree**
  - nodes u, v now disconnected
  - decrement u, v degree
- **Triangles**
  - did a triangle just break?
  - look for u, v common neighbors
  - decrement triangle (u,v,common)
- **Components**
  - did a component separate?
  - BFS to see if u, v still connected
  - if not, split component to two

SLOT: At the end of the algorithm…

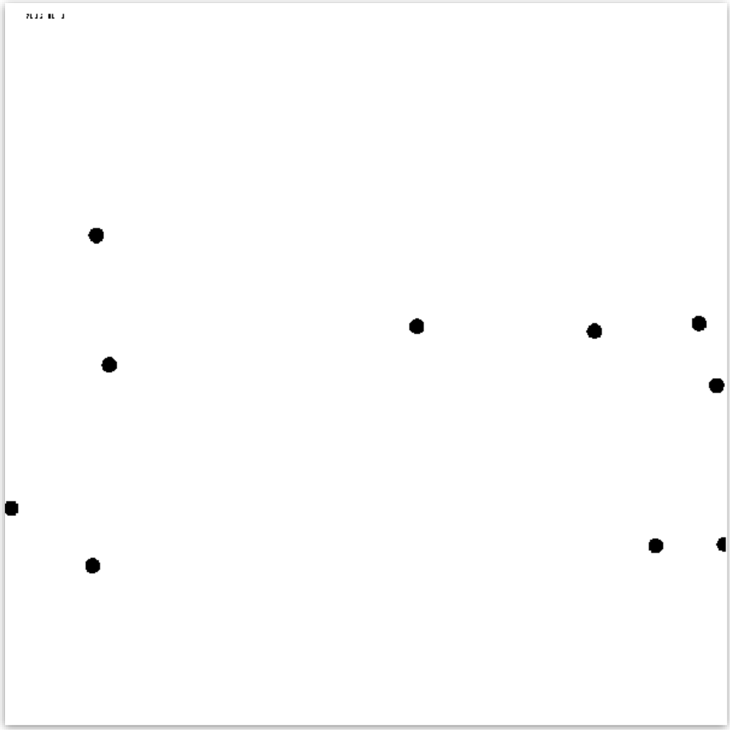
Rich analytics

- node degrees: start/end time, duration
- triangles: start/end time, duration
- connected components: start/end time, duration

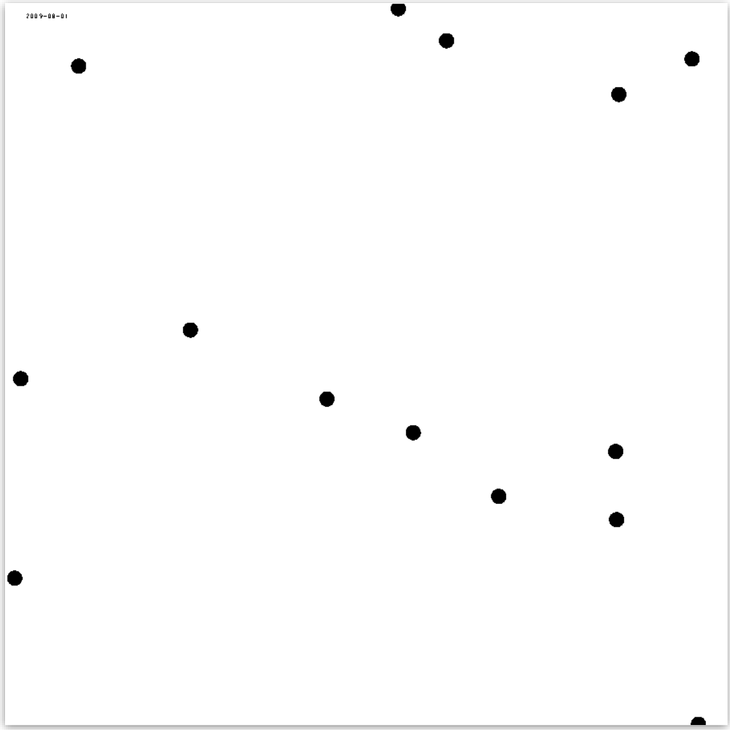
Exact results (not approximations)

Evaluation of SLOT

Simulating trajectories

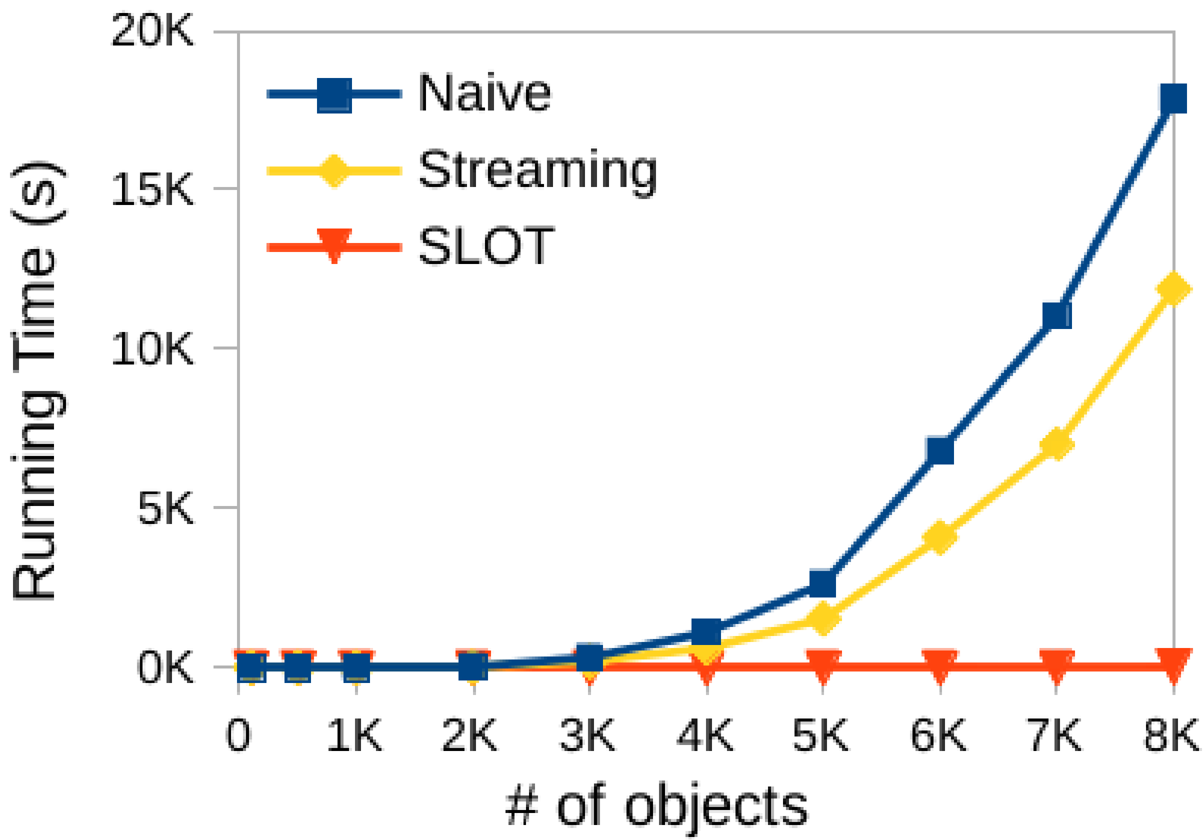


constant velocity

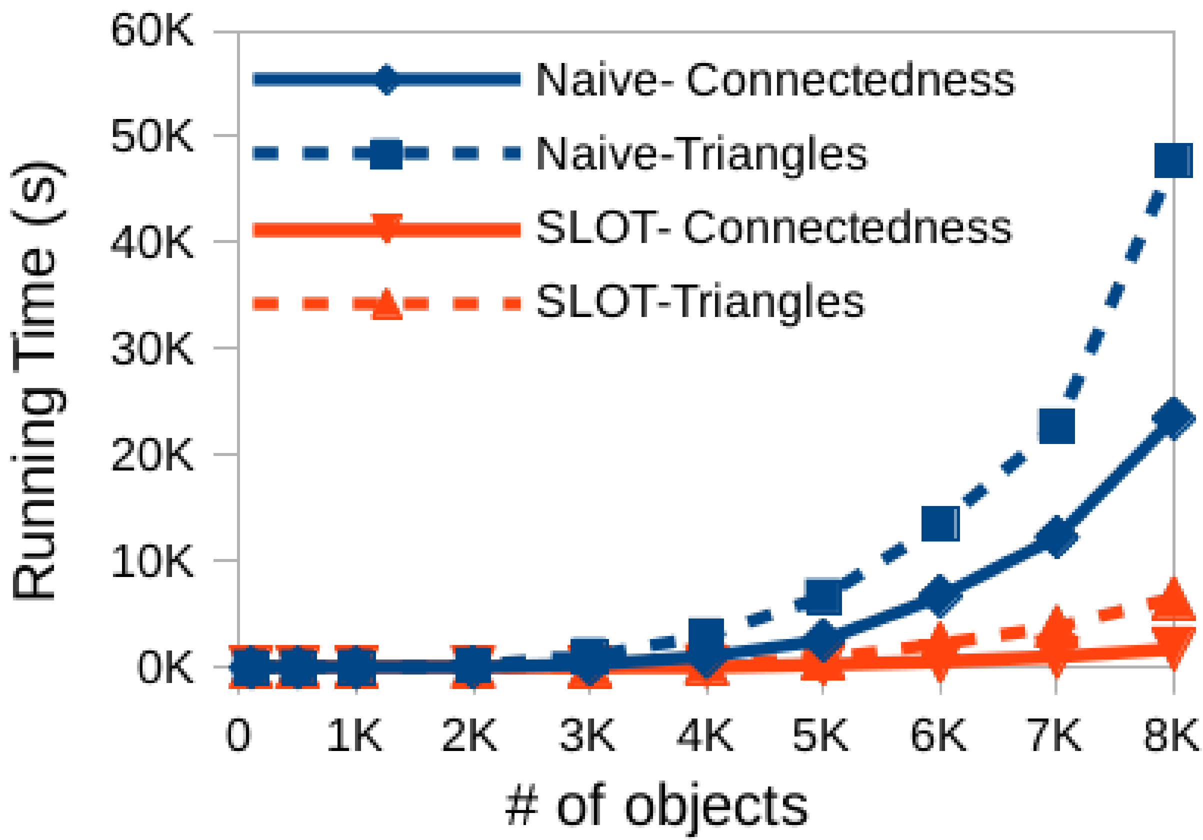


random velocity

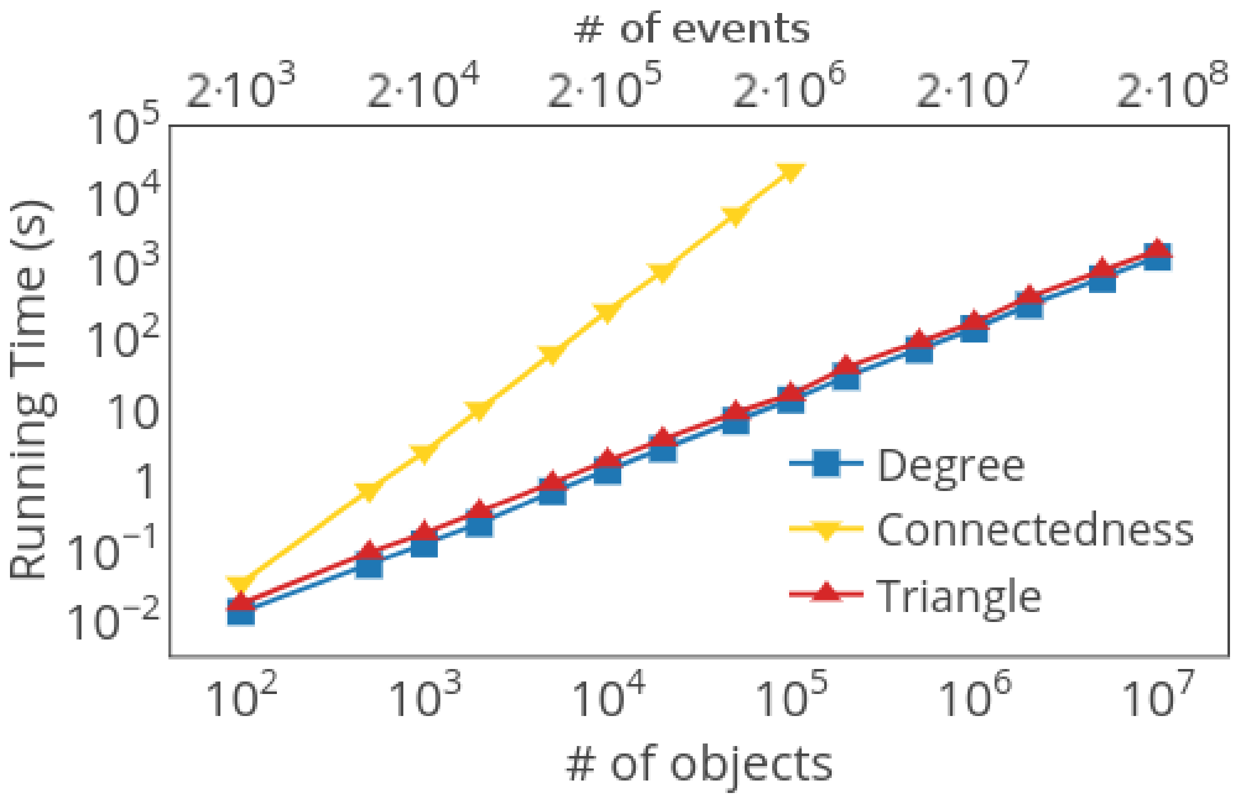
Degree


Naive
Streaming
SLOT
20K
15K
10K
5K
0K
Running Time (s)
0
1K
2K
3K
4K
5K
6K
7K
8K
# of objects

SLOT performance (triangles, connectedness)



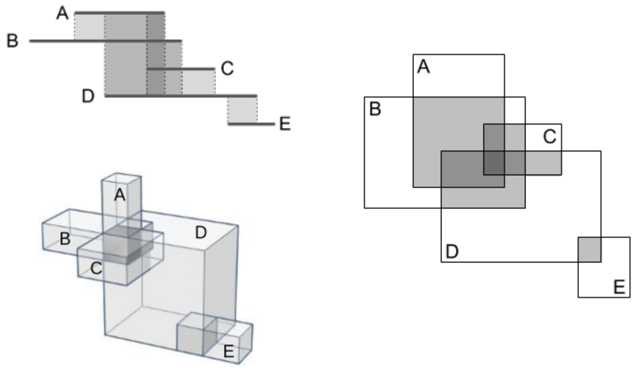
with max=0.15, min=0



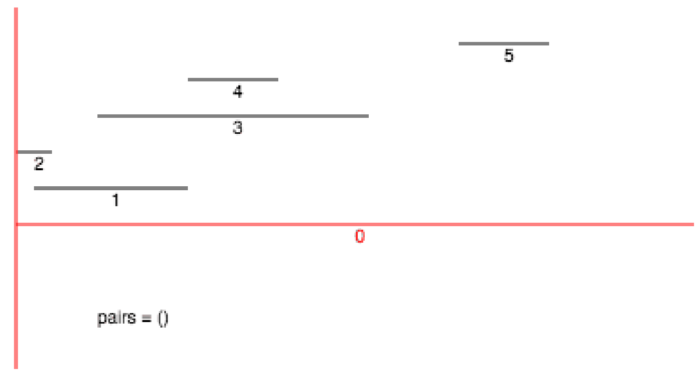
Seagull migration trajectories

Summary

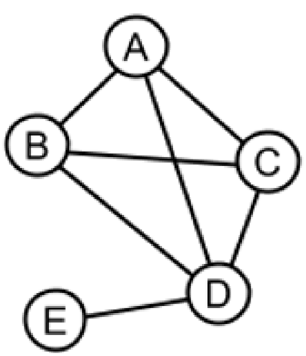
Multiple Intersections



Axis-aligned object intersections



Sweep-line algorithm

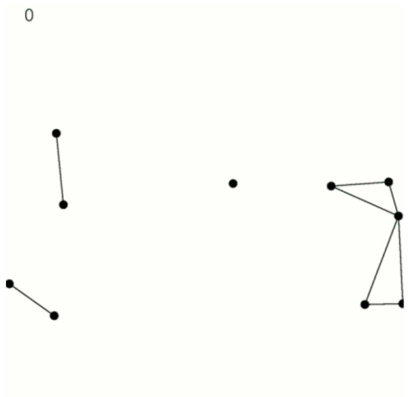


Region intersection graph

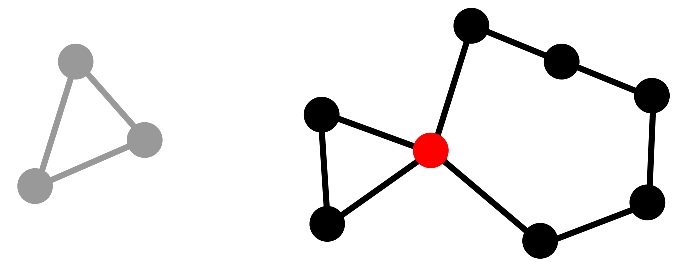
SLIG properties:

- Fast & efficient
- Exact
- Query capabilities

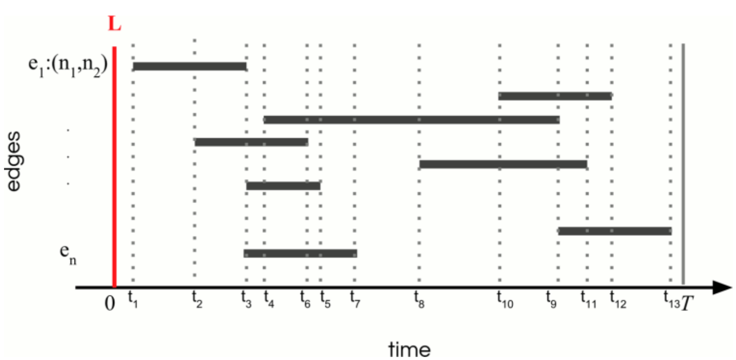
Node importance in TNs



Trajectory networks



Network Importance **over time**



SLOT algorithm

SLOT properties:

- Fast
- Exact
- Scalable

Contributions

- Fast and Accurate Mining of Node Importance in Trajectory Networks
  - IEEE International Conference on Big Data, 2018
- Efficient Mining and Exploration of Multiple Axis-aligned Intersecting Objects
  - Pending review in IEEE International Conference on Data Mining, 2019
- Working on extensions/applications of shown concepts
  - Data visualization
  - Location-aware computation offloading
  - Distributed versions of algorithms
- Industry collaboration project with Fortran Traffic

Thank you!