Road traffic congestion detection and tracking with Spark Streaming Analytics

Thorsteinn Thorri Sigurdsson (ttsi@kth.se)

Supervisors: Ahmad Al-Shishtawy & Zainab Abbas
Examiner: Vladimir Vlassov

Motivation and goal

- Traffic congestion poses several problems
  - Increased pollution and fuel consumption
  - Increased commuting times with higher transportation costs
  - Reduced traffic safety
- Goal
  - Real-time detection and tracking of congestion in road system
  - Using data from infrastructure traffic sensors

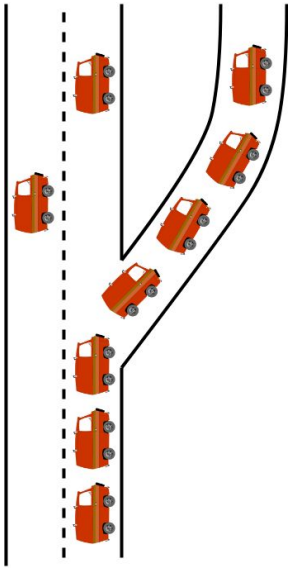
Benefits

Real time congestion detection use cases

- Send warnings to drivers approaching end-of-queue
- Congestion mitigation
  - Adaptive speed limits
  - Re-routing of traffic

Congestion tracking use cases

- Queue spill-back analysis
- Optimization of road network by traffic authorities



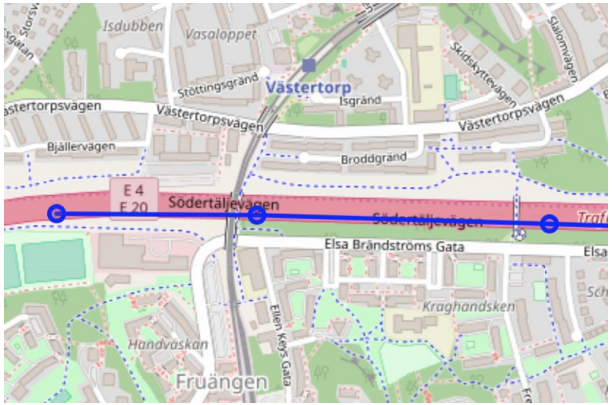
The data set

- 2059 radar sensors placed on main arteries in Stockholm (+ a single road in Göteborg)
- 150-400 meters apart
- Sensor measurement every minute
  - Flow (vehicles/minute)
  - Average speed (km/h)
  - -> Calculated density (vehicles/km)
- Sensors identified by
  - Road ID
  - Relative kilometer reference
  - Lane ID (sensor for each lane on the road)
  - + GPS coordinates for each sensor

E4N 55650. Lanes 1-4.
Image from Google Maps' Streetview.

Problem statement

- Instead of detecting congestion at individual sensors, extend detection into spatial dimension
- Detect regions of congestion
  - Connected congested sensors representing a queue
- Track evolution of queues through time
  - Build up, split, movement, dissipation
- Resolution down to the individual lane level



Problem statement

- Construct traffic sensor graph
- Frame the congestion detection problem as a graph processing problem
- Adapt existing graph processing algorithms for this task
  - Graph clustering
  - Community detection
- Use streaming graph processing algorithms for real time congestion detection and tracking

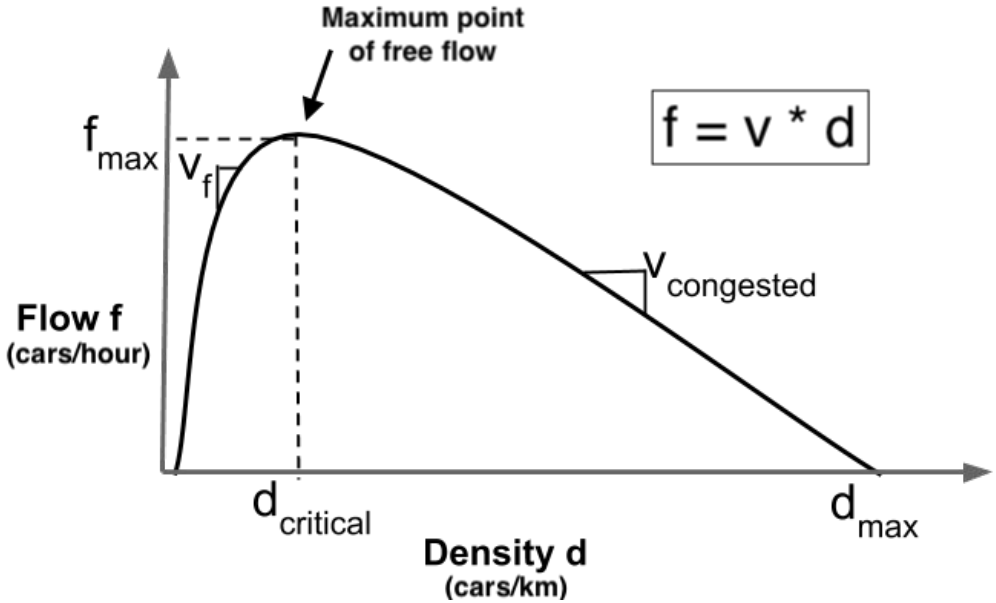
Background - Traffic theory

3 main metrics

- Flow (vehicles / hour)
- Speed (km / hour)
- Density (vehicles / km)

Traffic flow can be classified as:

- Free flow
  - Vehicle can do any maneuver the driver wishes
- Congestion
  - Complement of free flow



Fundamental diagram of traffic flow

Background - Traffic queues

- Can be defined as a row of vehicles waiting to be served
  - Moving at slow speeds or stopped
- Safety hazard for cars approaching the end-of-queue at higher speeds
- Can spread to disrupt traffic on other roads otherwise un-congested
  - Queue spillback
- We take a connected chain of congested traffic sensors to represent a queue

Background - shockwaves

A shockwave is a change or discontinuity in traffic conditions

- Propagation of change in speed or density
- Travels through the road system

Queue buildup at an intersection is a shockwave (queue extends upstream)

Can also form “out of thin air” when density exceeds critical threshold

Sugiyama et al.

Related work

- Traffic congestion can be detected with a number of different approaches
- Microscopic methods
  - Can view traffic flow at individual vehicle basis
    - Probe vehicles
    - Vehicle re-identification methods
    - Vehicle to vehicle communication systems
- Macroscopic methods
  - View traffic flow in aggregate
    - Cameras and image recognition
    - Acoustic sensors
    - Aerial and satellite imagery
    - Aggregated traffic statistics (average speed, flow, density)

Related work

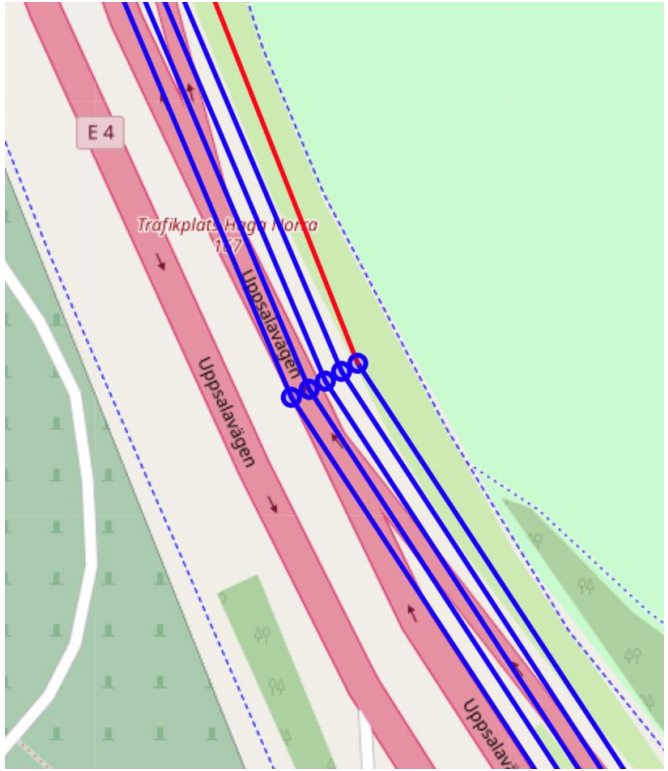
- Google Maps
  - Probe vehicles
  - Speed measurements from smartphones collected
  - Red, yellow, green roads in Google Maps application
- Coifman (2003)
  - Congestion detection with infrastructure sensors
    - Dual loop induction sensors
    - Give vehicle length measurements
  - Re-identification of vehicles at different locations, by vehicle length
  - Abnormally high link journey times -> congestion
  - Ground truth: Video recordings

Related work

- Li et al. (2007)
  - FlowScan algorithm
    - Density based clustering to find “hot routes”
    - i.e. not congestion detection
  - Probe vehicle trajectories
  - Directed graph
    - Vertices are intersections or landmarks
  - Uses number of shared trajectories and number of hops between edges for density based clustering
  - Ground truth
    - Traffic simulator data
  - Batch processing only

The traffic sensor graph

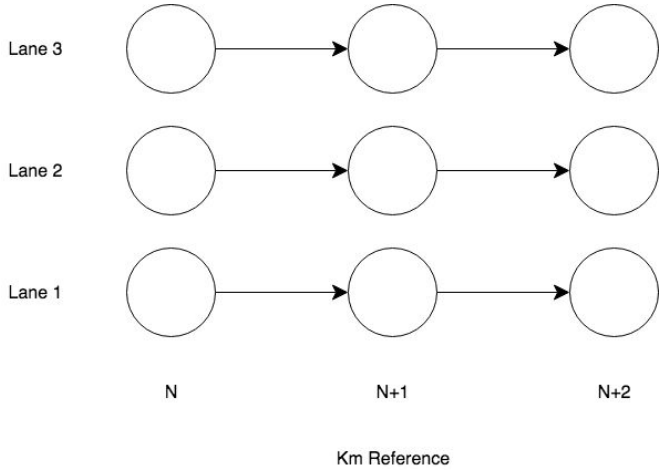
- Directed graph modelling spatial relationship between traffic sensors
- Sensors as vertices, road segments between sensors as edges
- A path in the graph is a possible path to drive in road system
- At most 2037 sensors
  - Graph valid from 2016-5-9 onwards

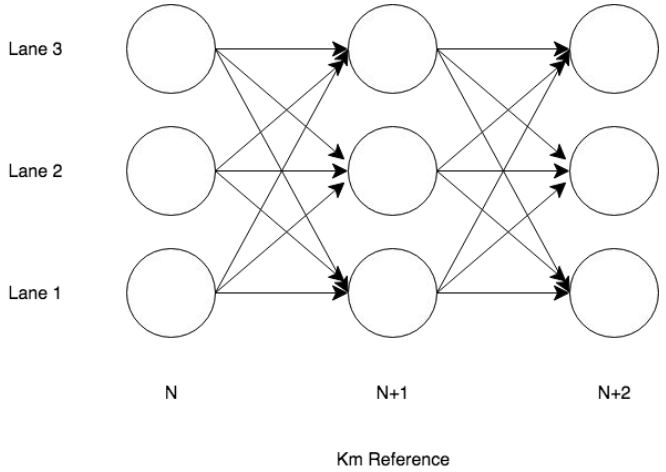


E4N next to Frösunda. Red edge is a connection to another road, E4A.

The traffic sensor graph

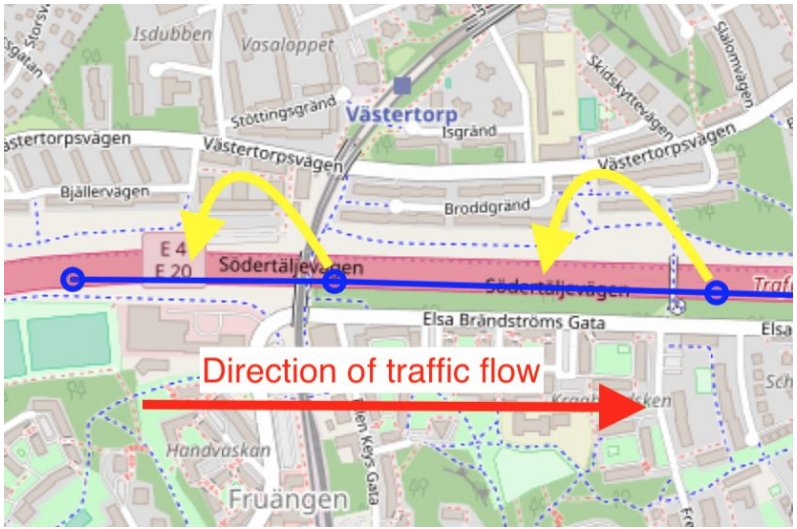
- Two types of graphs were constructed
- Base graph
  - No lane changes
  - Used in congestion detection and tracking, per lane
- Reachability graph
  - Allow lane changes
  - Can be used to send “congestion ahead” warnings to drivers





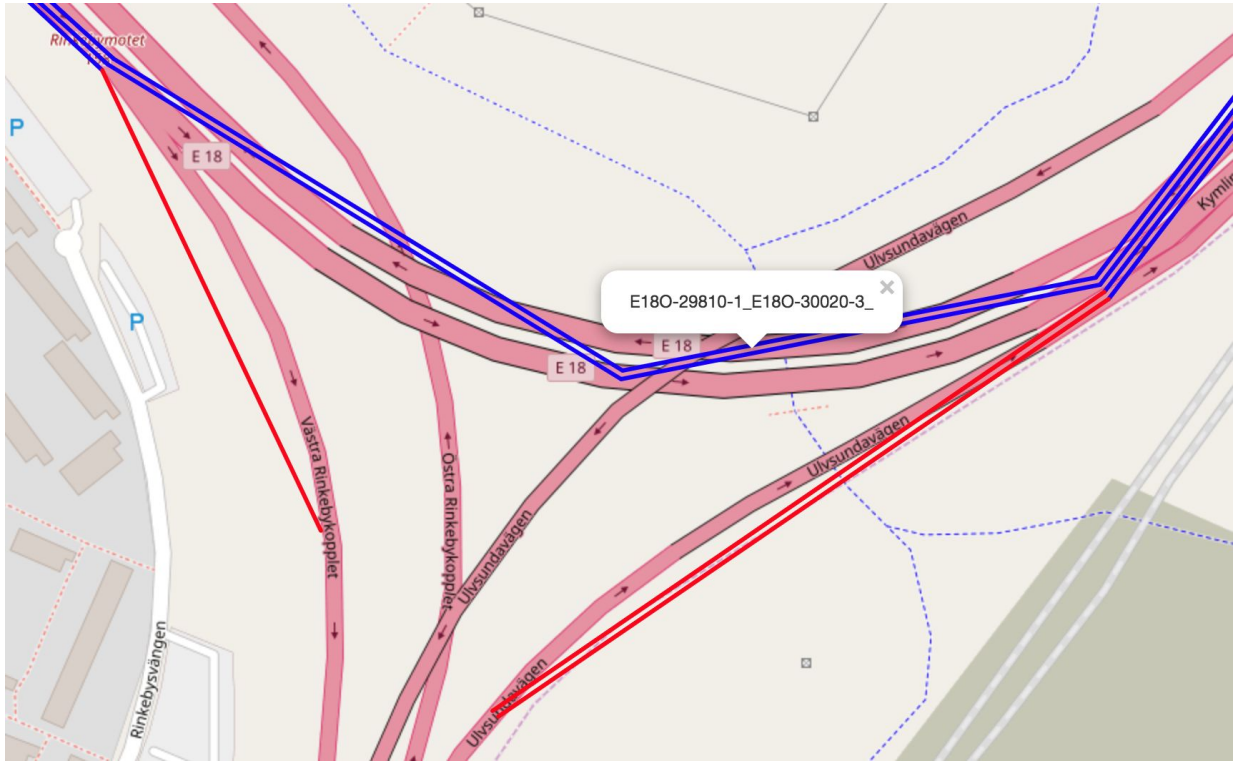
The traffic sensor graph

- Directed graph
  - Each edge has a source and destination vertex
- Edges weighted with sensor measurements from destination vertex
  - Assumed to represent the conditions on the road segment upstream of the destination vertex
- Weights updated every minute (new graph)



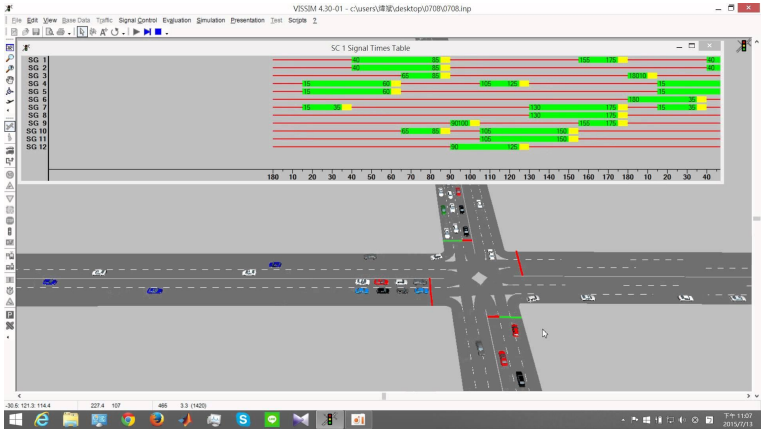
Edges weighted with sensor measurement from destination vertex

The traffic sensor graph



Ground truth

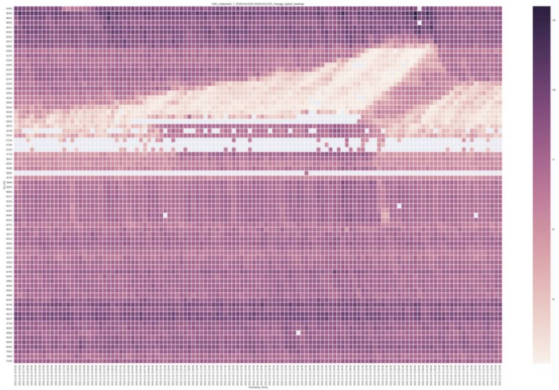
- All implemented methods unsupervised
- No ground truth available for our data set
  - What constitutes congestion?
- Related work has used
  - Video recordings of traffic
  - Domain experts to identify and label ground truth
  - Fusion with external data sets such as accidents
  - Traffic simulators
    - Allowing complete observation of synthetic traffic flow
    - Vehicle trajectories, number of vehicles in queue etc.



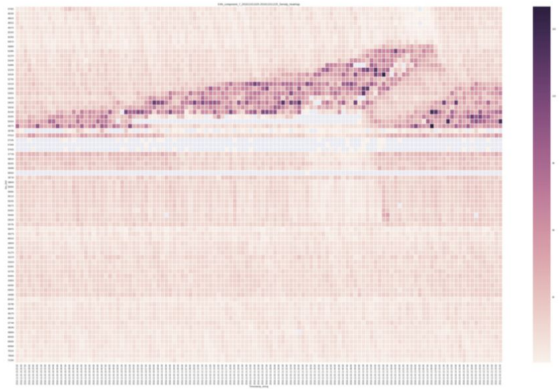
Screenshot of the VISSIM Traffic Simulator

Ground truth

- Use visual inspection of heat maps to compare results to underlying data
- Spatio-temporal patterns of measured traffic features
  - Time on x-axis, kilometer reference on y-axis (space)
- Congestion detection methods should find the same patterns
- Domain expert approach



Average speed



Density

Manually labelling data set

Find connected components in base graph

- Represent “actual lane” - path if driving without changing lanes
- Can’t rely on lane IDs of sensors

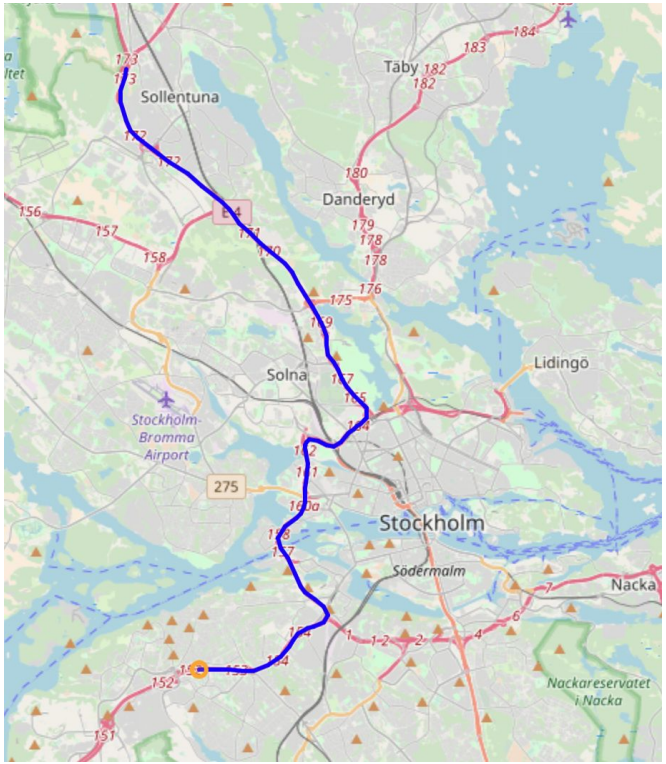
8 test congestion patterns identified on the “actual lanes”

- Queues and shockwaves labeled
- 16 queues, 15 shockwaves

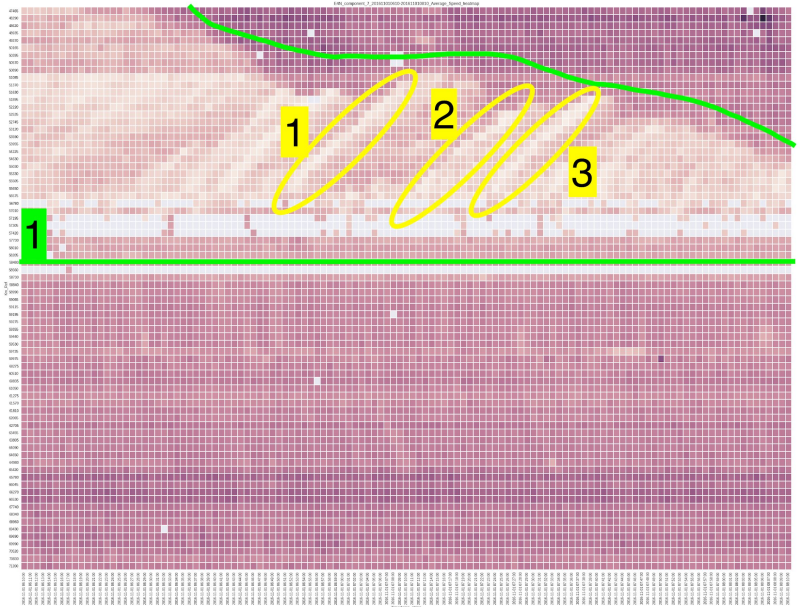
Criteria for selection

- Combined they should cover most of the road network
- Show interesting congestion patterns

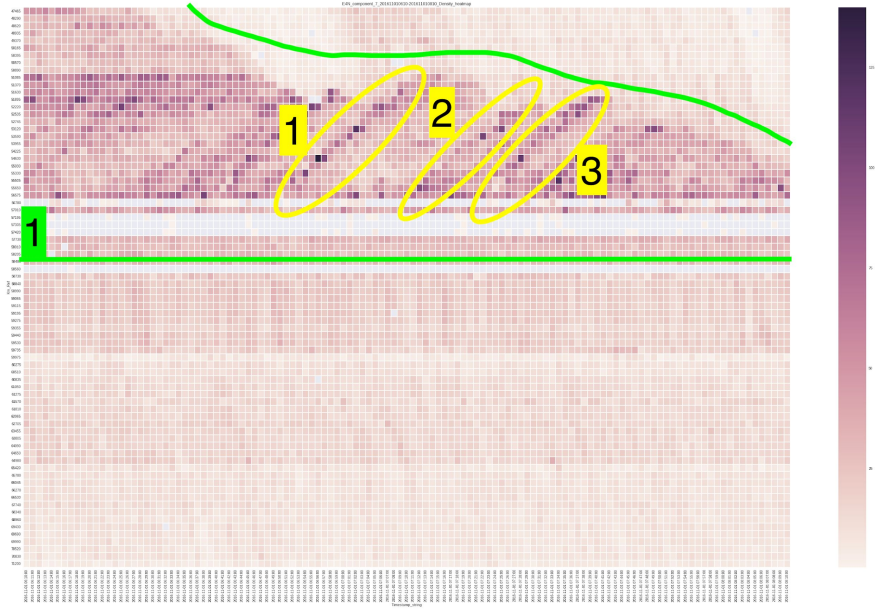
E4N 2016-11-1 (component 7)



E4N 2016-11-1 06:10 - 08:10 (component 7)

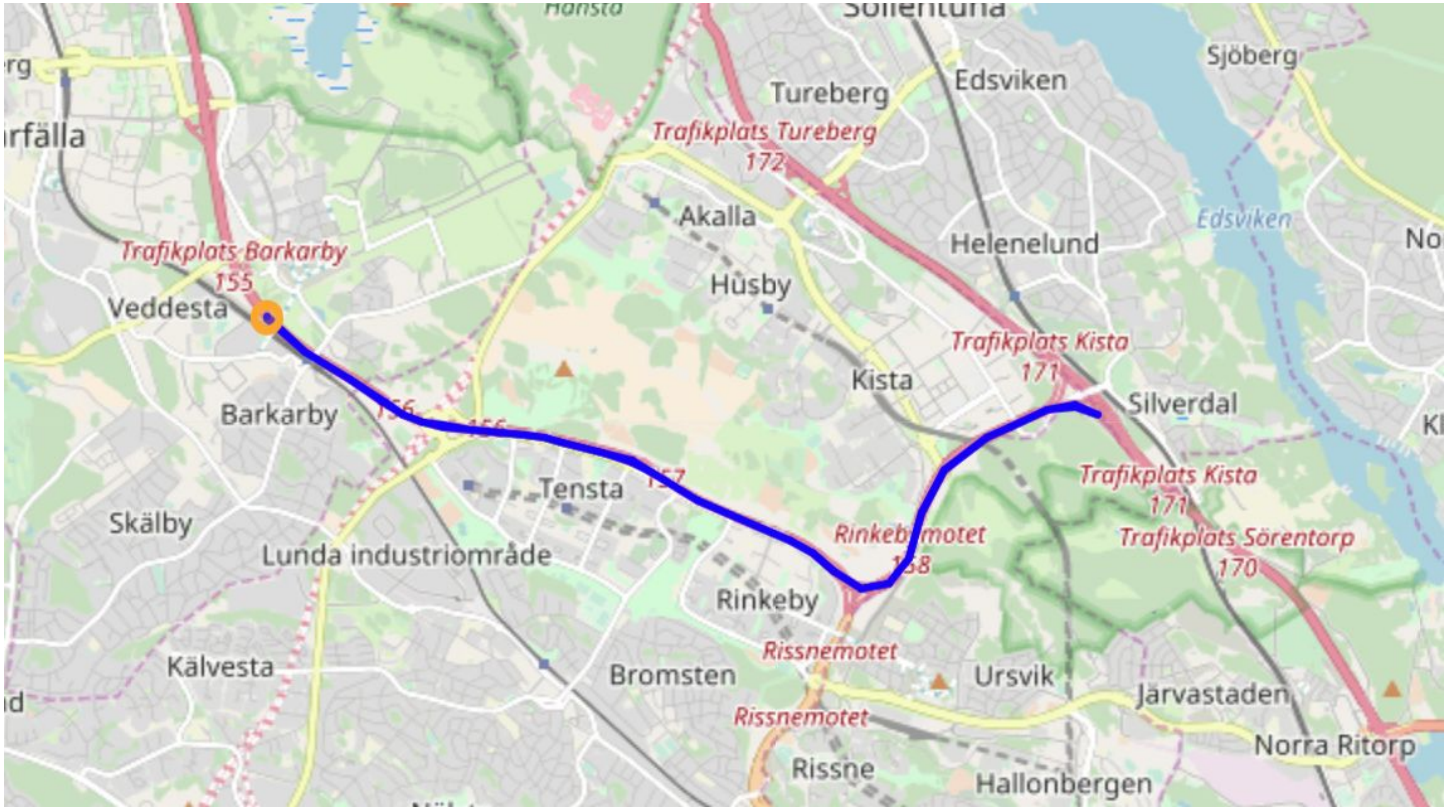


Average speed

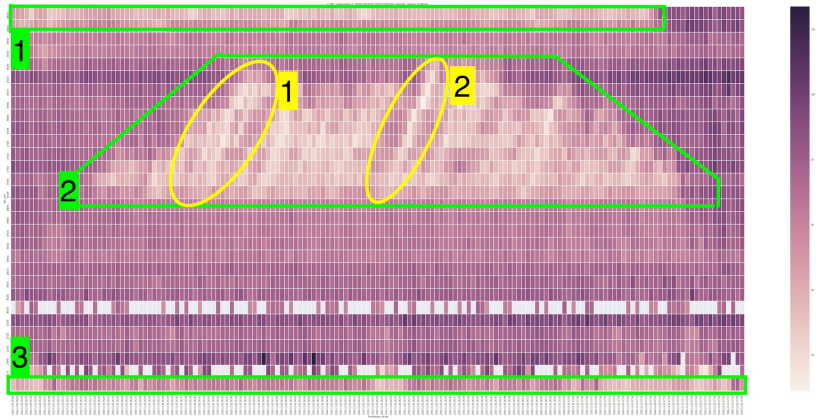


Density

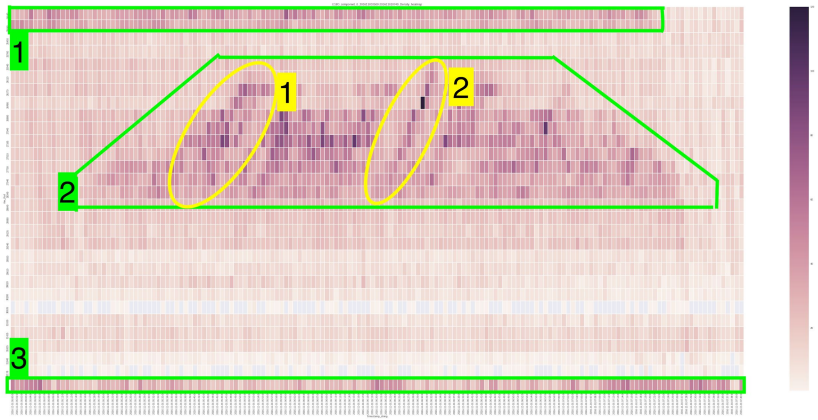
E18O 2016-11-1 05:00 - 07:40 (component 0)



E18O 2016-11-1 05:00 - 07:40 (component 0)



Average speed



Density

Congestion detection methods

4 congestion detection approaches implemented on batch data

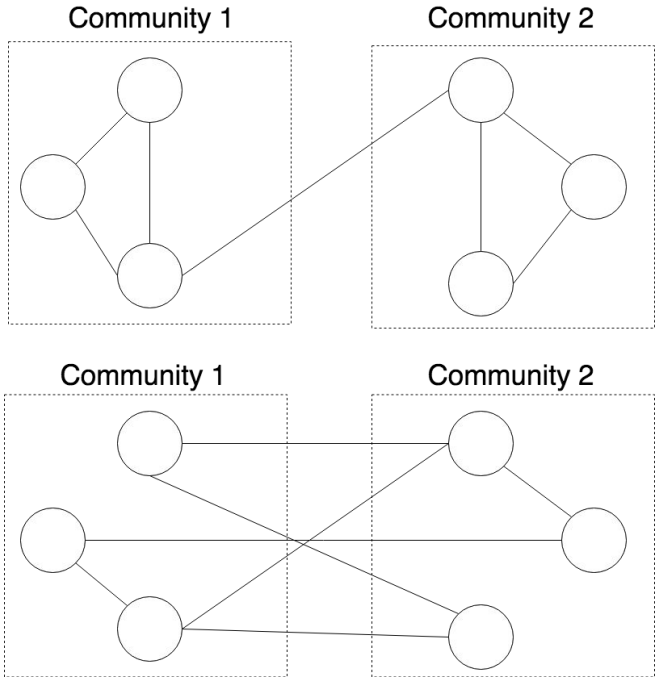
- Louvain modularity
- Hierarchical (data) clustering
- Congested components
- Dengraph

2 best performing implemented as streaming systems in Spark Structured Streaming

- Congested components
- Dengraph

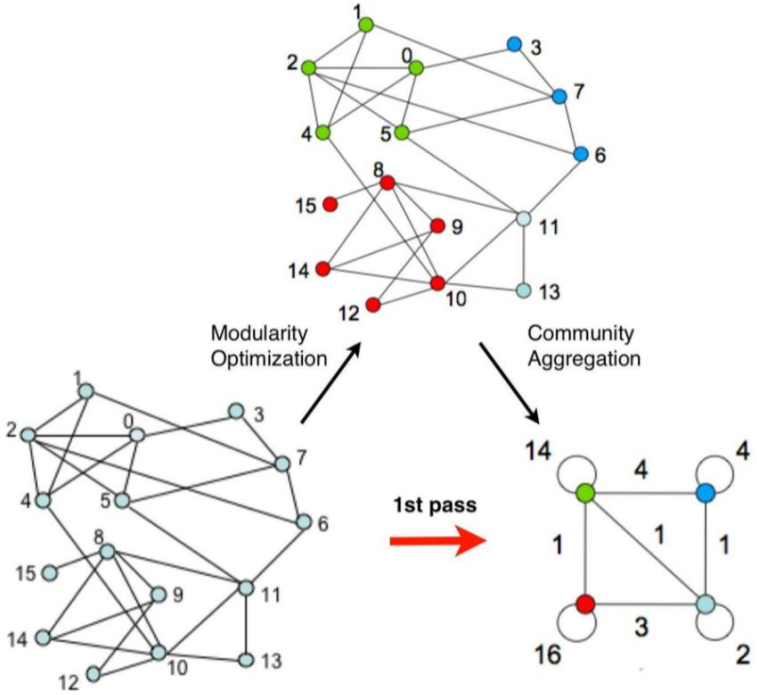
Louvain modularity

- Weighted modularity optimization community detection algorithm
- Modularity Q
  - Measure of quality of discovered communities, single number in range Q = [-1,1]
  - Fraction of edges in the graph that connect vertices within the same community, minus the expected fraction in a graph with the same community division and degree distribution, but edges connected randomly
  - How much better is the community division than would happen randomly
  - Q approaches 1 -> good community division
- Idea: Find communities of congested sensors



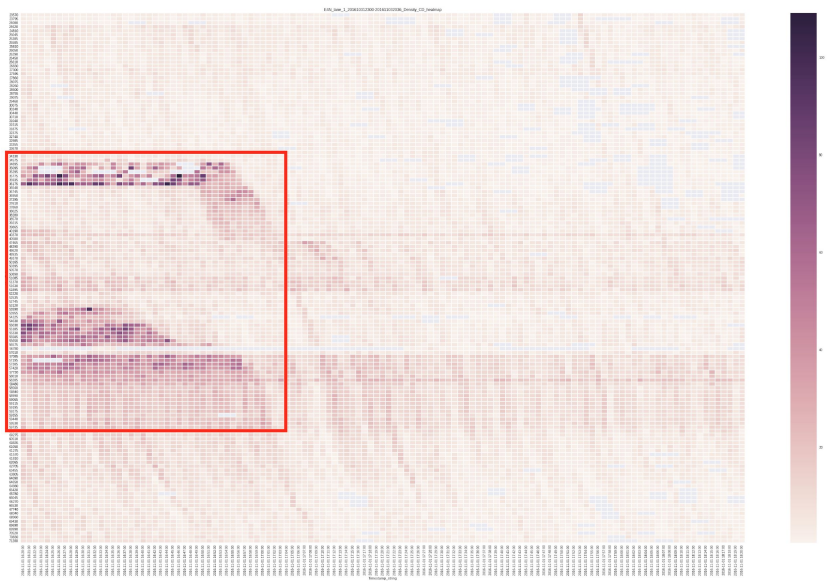
Louvain modularity

- Edges of graph weighted with density (cars/km)
- Run minute-by-minute on data from E4N, lane 1, 2016-11-1 16:20-18:20
- Average modularity value achieved (averaged over all minutes)
  - Q = 0.728 (high)
  - Finishes in one iteration
- Poor results
- Comparison to random graph not suitable for chain-like base graph
  - Assumes each node can connect to any other node

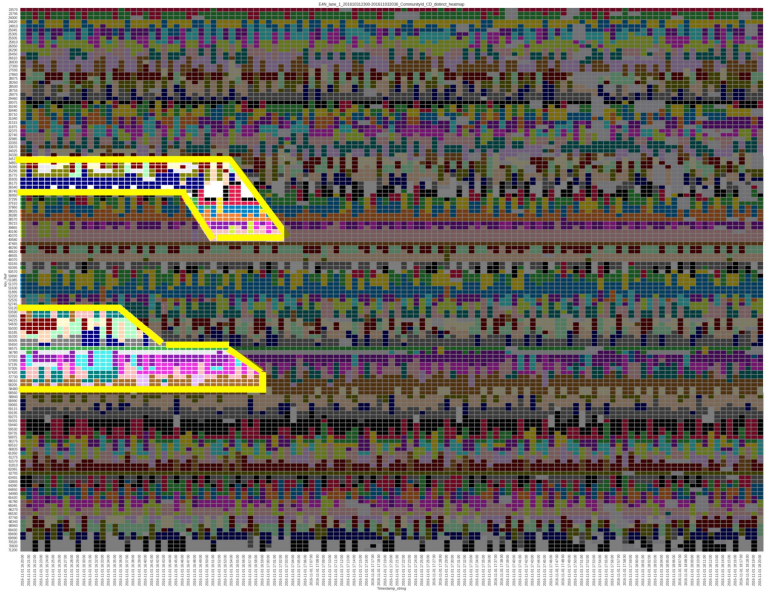


Blondel et al.

Louvain modularity results



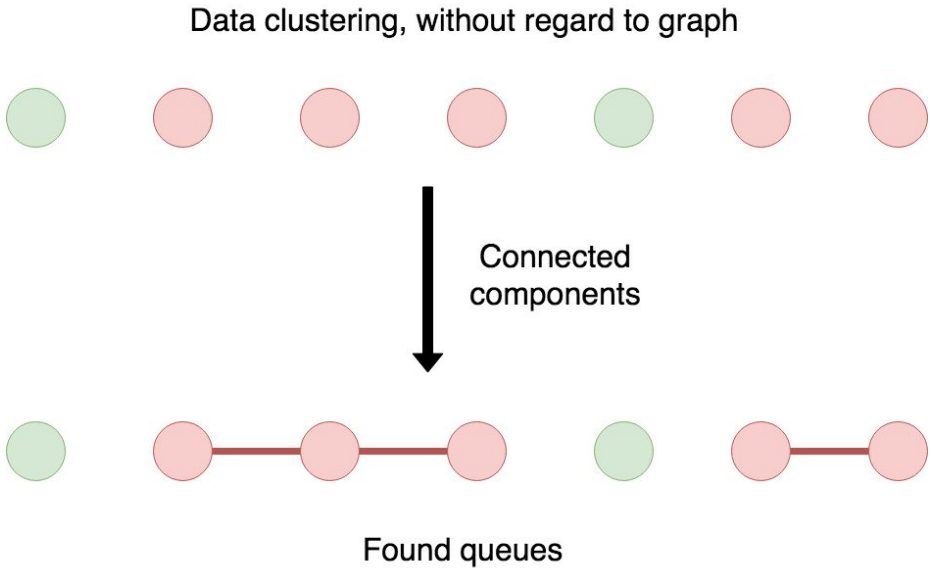
Observed congestion pattern (density)



Detected community division

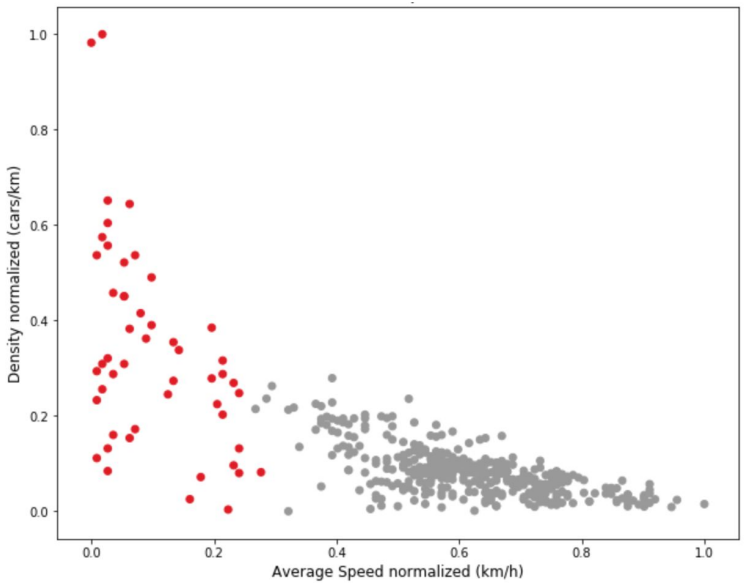
Hierarchical data clustering

- Base graph is very simple (chain of vertices)
- Idea:
  - Data clustering to find congested sensors (i.e. not graph)
  - Run connected components on graph to find connected component of congested sensors (queue)
- Hierarchical clustering chosen to explore clustering result at different hierarchies
  - Congestion hierarchy



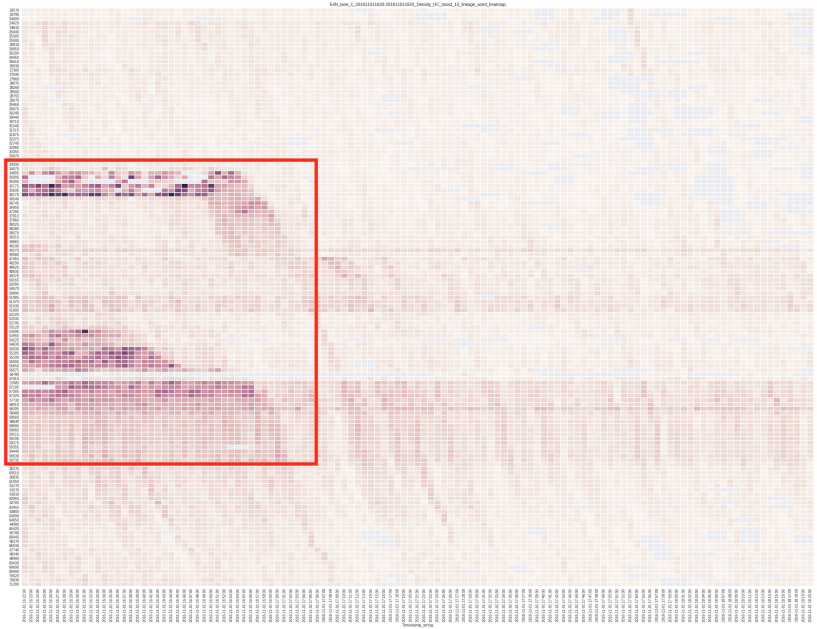
Hierarchical data clustering

- Features:
  - Average speed and density
  - Normalized for each minute
- Run minute-by-minute on data from E4N, lane 1, 2016-11-1 16:20-18:20
- Dendrogram cut to generate two clusters
  - Free flow and congested
- Ward's variance minimization linkage criteria gave best results

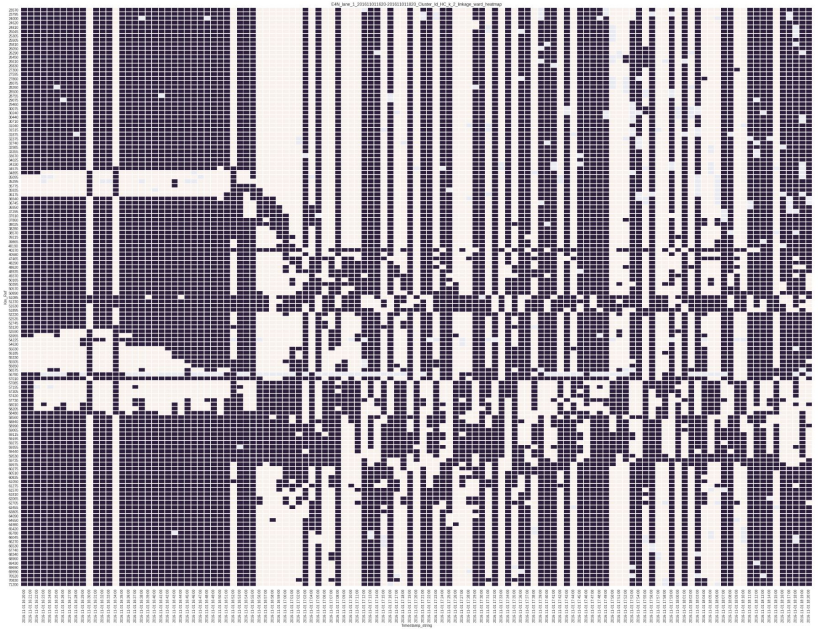


Clustering result for 2016-11-01 16:20. Ward linkage. 2 clusters.

Hierarchical data clustering - result

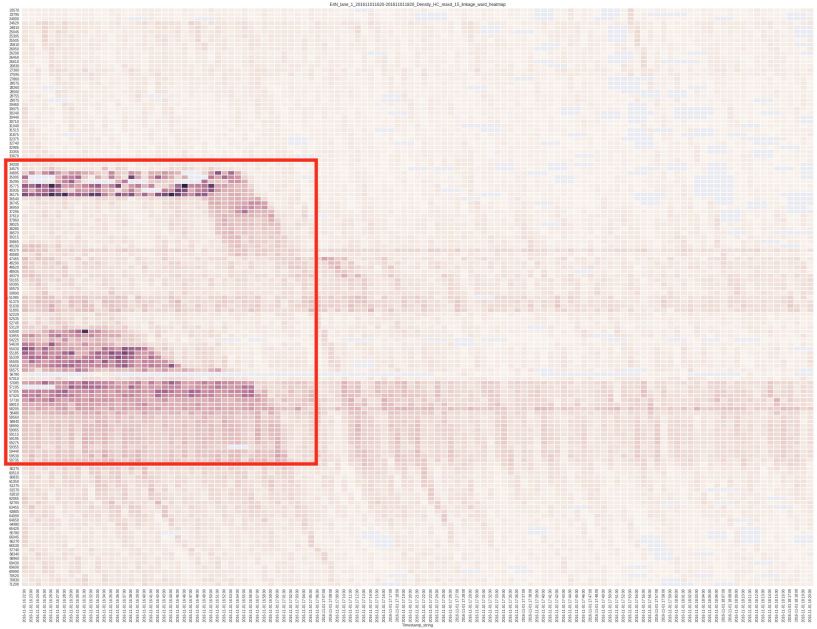


Observed congestion pattern (density)

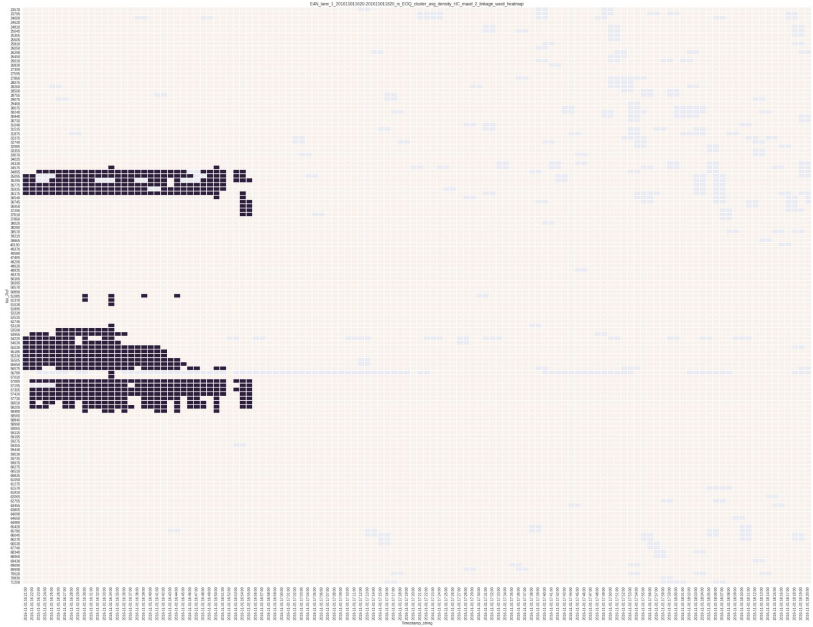


Cluster division (before split)

Hierarchical data clustering - results



Observed congestion pattern (density)

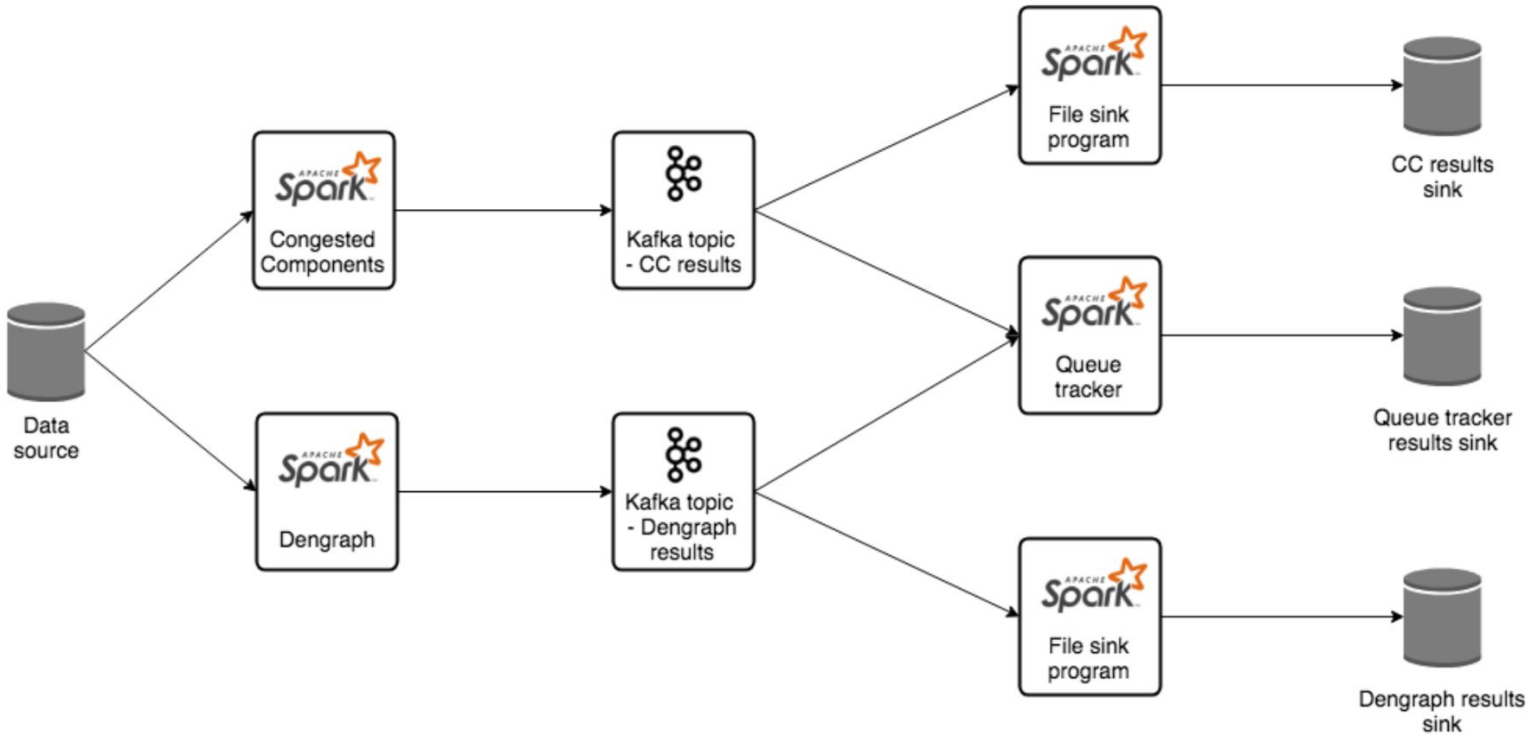


Queue clusters - after split
Queue detection heuristic: avg density within cluster > 30

Streaming approaches

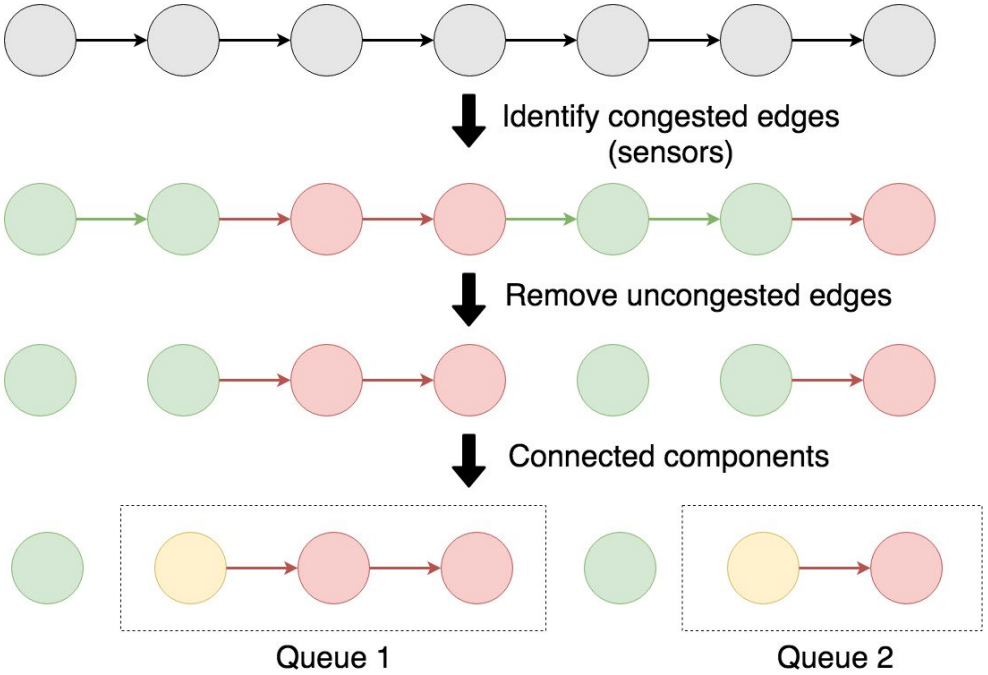
- Congestion detection
  - Congested components
  - Dengraph
- Queue tracker

System architecture



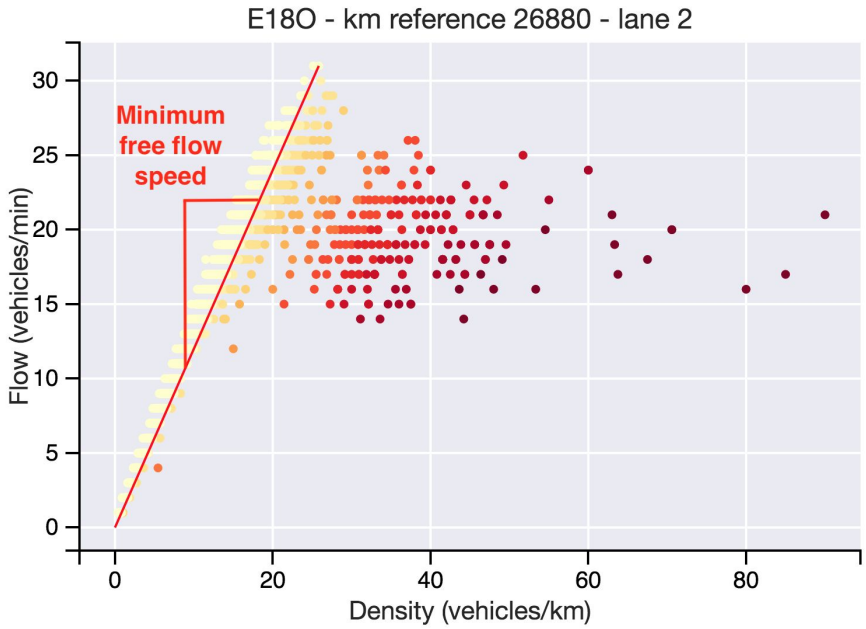
Congested components

- Idea: Find connected components of congested sensors
- They represent queues



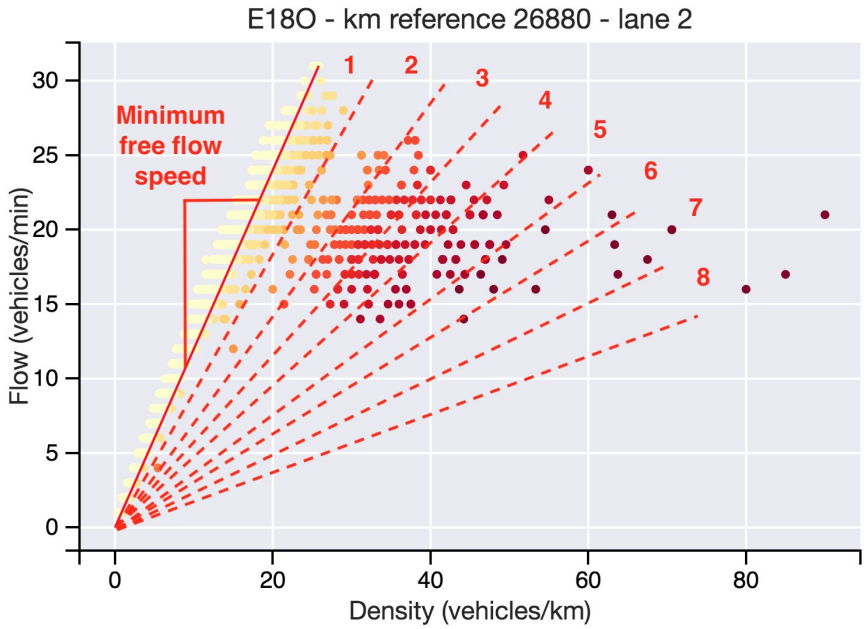
Congested components

- Detect congestion based on speed
- Congestion threshold
  - Minimum free flow speed
  - Slope of line from origin to maximum free flow point
  - Determined for each individual sensor from historical data
  - Variable between sensors
- Congestion classes
  - Measure of severity of congestion
  - Classify sensor measurements based on how far below minimum free flow speed
    - (increments of 5 km/h)

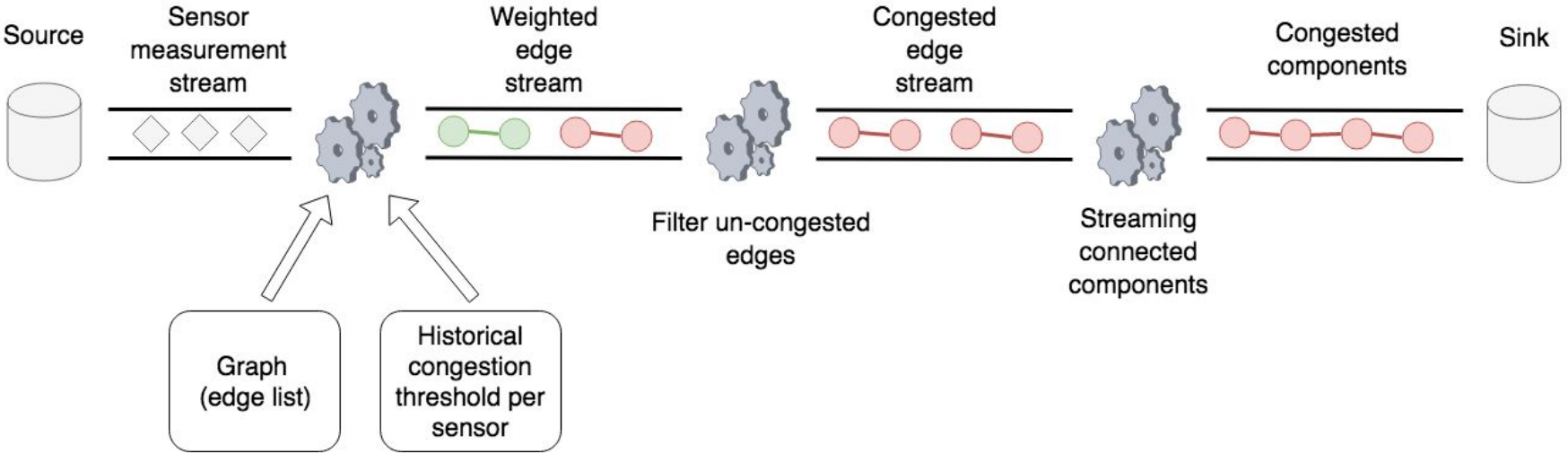


Congested components

- Detect congestion based on speed
- Congestion threshold
  - Minimum free flow speed
  - Slope of line from origin to maximum free flow point
  - Determined for each individual sensor from historical data
  - Variable between sensors
- Congestion classes
  - Measure of severity of congestion
  - Classify sensor measurements based on how far below minimum free flow speed
    - (increments of 5 km/h)



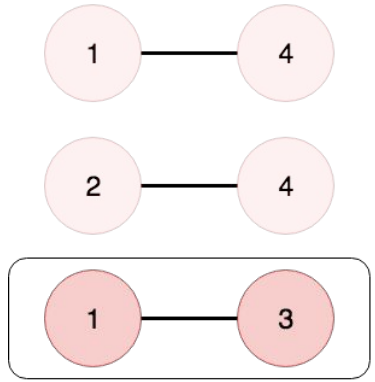
Congested components



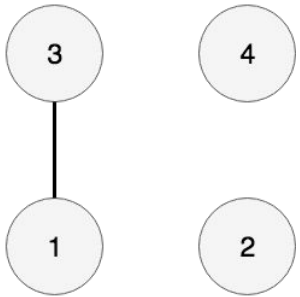
Streaming connected components

- Single-pass algorithm
- For each edge (u,v)
  - If neither u nor v in component, create component with ID min(u.ID,v.ID)
  - If u,v in different components C1 and C2, merge components and update component ID to min(C1.ID, C2.ID)
  - If only u or v in component, add the other one to the component
- State:

  Component ID to vertex IDs map
- State built up for each minute
  - One minute tumbling window



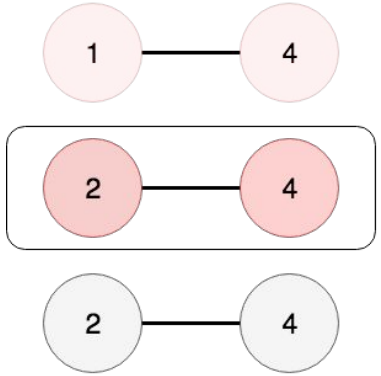
Component	Vertices
1	1,3



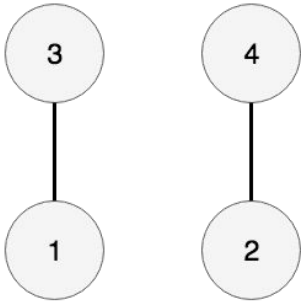
Streaming connected components

- Single-pass algorithm
- For each edge (u,v)
  - If neither u nor v in component, create component with ID min(u.ID,v.ID)
  - If u,v in different components C1 and C2, merge components and update component ID to min(C1.ID, C2.ID)
  - If only u or v in component, add the other one to the component
- State:

  Component ID to vertex IDs map
- State built up for each minute
  - One minute tumbling window



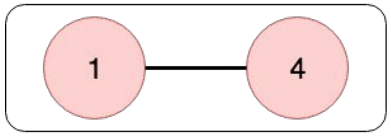
Component	Vertices
1	1,3
2	2,4

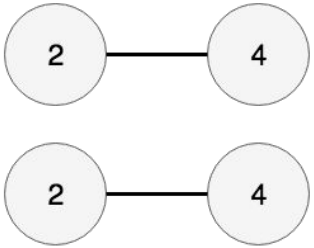


Streaming connected components

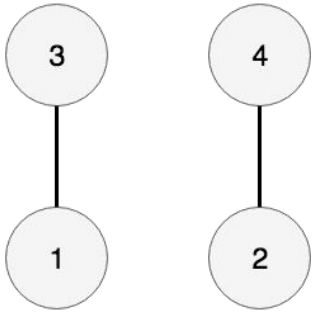
- Single-pass algorithm
- For each edge (u,v)
  - If neither u nor v in component, create component with ID min(u.ID,v.ID)
  - If u,v in different components C1 and C2, merge components and update component ID to min(C1.ID, C2.ID)
  - If only u or v in component, add the other one to the component
- State:

  Component ID to vertex IDs map
- State built up for each minute
  - One minute tumbling window





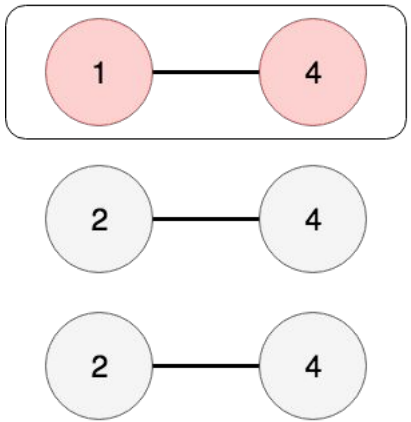
Component	Vertices
1	1,3
2	2,4



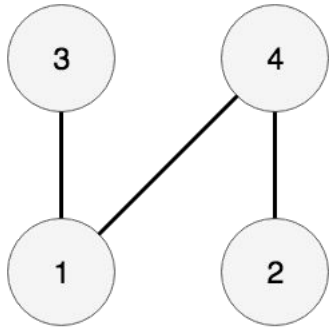
Streaming connected components

- Single-pass algorithm
- For each edge (u,v)
  - If neither u nor v in component, create component with ID min(u.ID,v.ID)
  - If u,v in different components C1 and C2, merge components and update component ID to min(C1.ID, C2.ID)
  - If only u or v in component, add the other one to the component
- State:

  Component ID to vertex IDs map
- State built up for each minute
  - One minute tumbling window

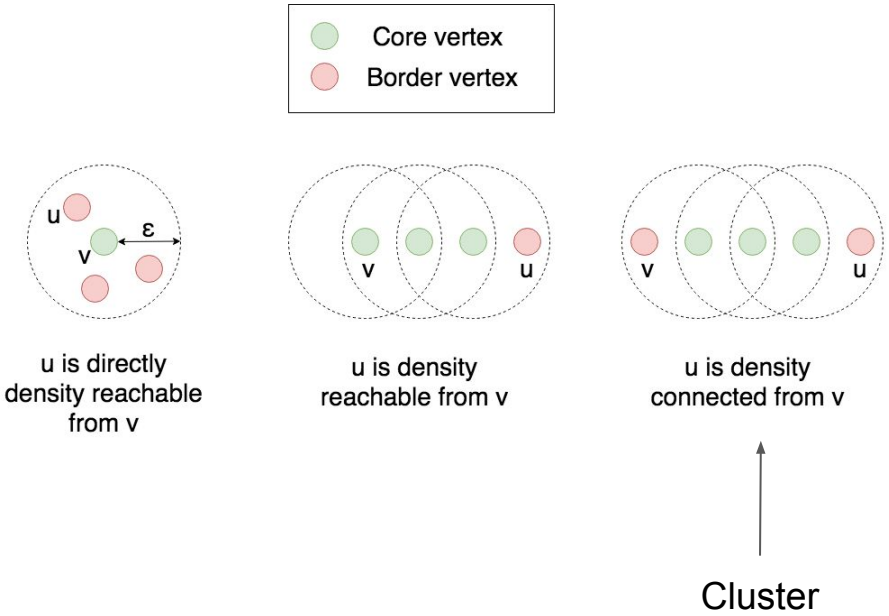


Component	Vertices
1	1,2,3,4



Dengraph

- Density based graph clustering
- Parameters
  - ϵ - neighborhood radius (distance)
  - η - min number of vertices within neighborhood for a vertex to be a core vertex
- Classify vertices
  - Core, border, noise
- Cluster:
  - Maximal set of vertices that are density connected w.r.t. ϵ, η
  - Nodes not in cluster are classified as noise



Dengraph distance function

- Distance function

$$dist(p,q) = \begin{cases} 0 & p = q \\ \frac{1}{W_{p,q}} & \exists (p,q) \in E \\ undefined & otherwise, \end{cases}$$

- Edges weighted with density
- Essentially thresholds on density
- Critical density about the same for all roads [Highway Capacity Manual 2010]
  - ≈ 28 vehicles/km/lane

Neighborhood radius ϵ	**Corresponding density (vehicles/km)**
0,025	40
0,03	33,3
0,035	28,6
0,04	25
0,045	22,2
0,05	20

Dengraph

- Incremental clustering algorithm
  - Weighted edge stream
- Each edge can have a *positive* or *negative* effect on clustering
  - Positive if edge not seen before or it brings two vertices close enough to fall in each others neighborhoods
    - Cluster creation, absorption or merge
  - Negative if edge removed or if distance between two vertices grows so they are no longer in neighborhood
    - Cluster removed, shrunk or split
- State not carried between minutes
  - Only positive updates (new graph every minute)
  - One minute tumbling window
- State: All edges seen so far, vertex state for all vertices (cluster ID, core /border/noise)

Dengraph noise resistance

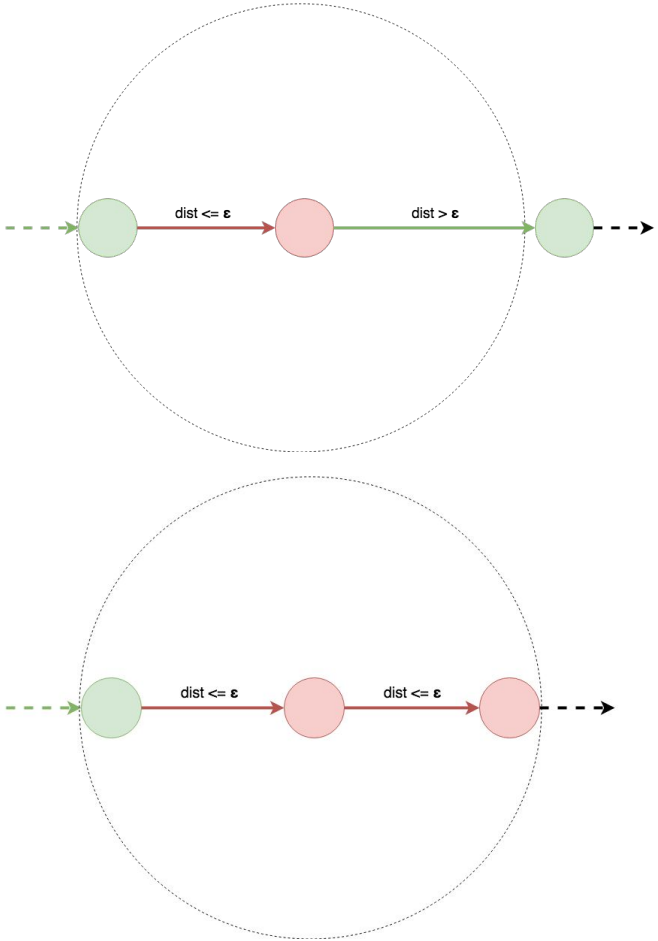
Resistant to noise through η parameter

- Minimum number of nodes in neighborhood to be a core vertex

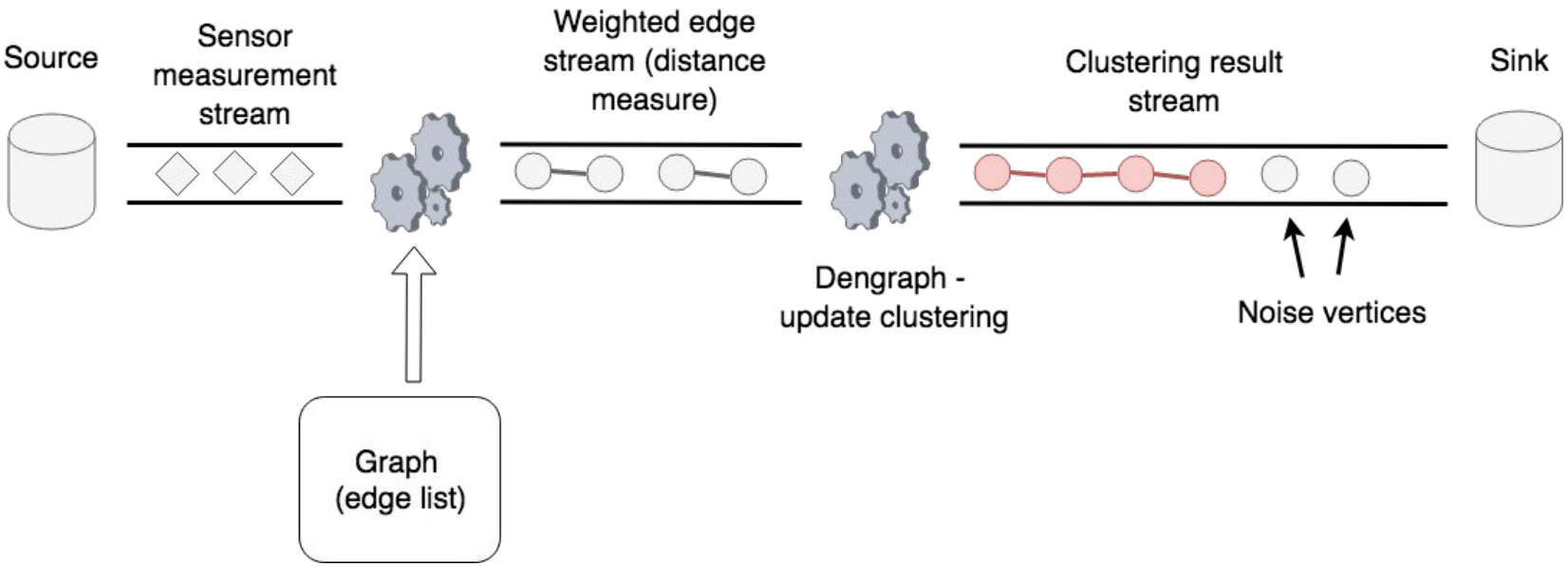
Used η = 2

- Need two vertices in neighborhood to form a cluster
  - At least two congested sensors to form a queue (+ a 3rd vertex)

Noise gets classified as noise vertices

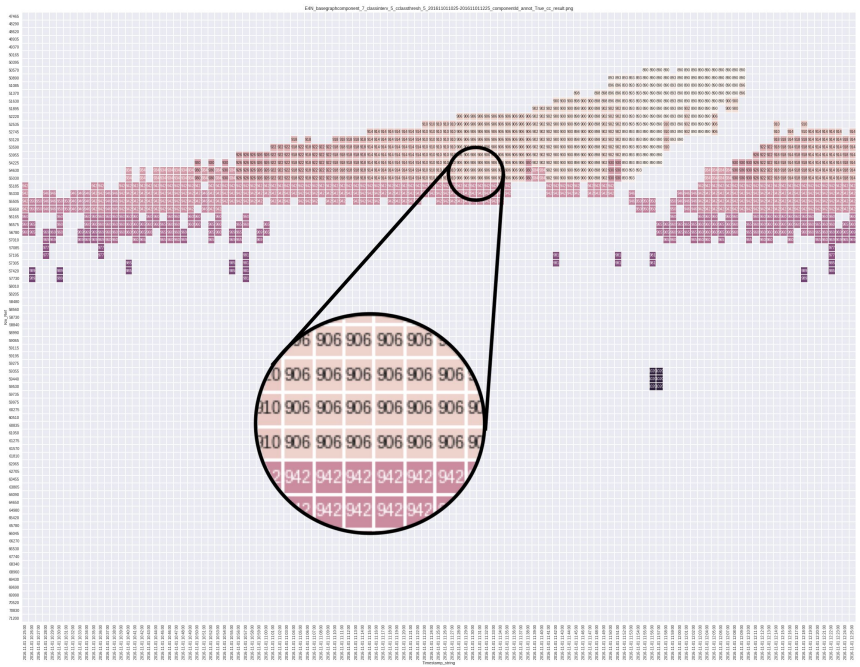


Dengraph


Source
Sensor measurement stream
Weighted edge stream (distance measure)
Clustering result stream
Sink
Dengraph - update clustering
Noise vertices
Graph (edge list)

Detecting individual queues

Both algorithms find individual queues, with unique queue IDs (within minute)



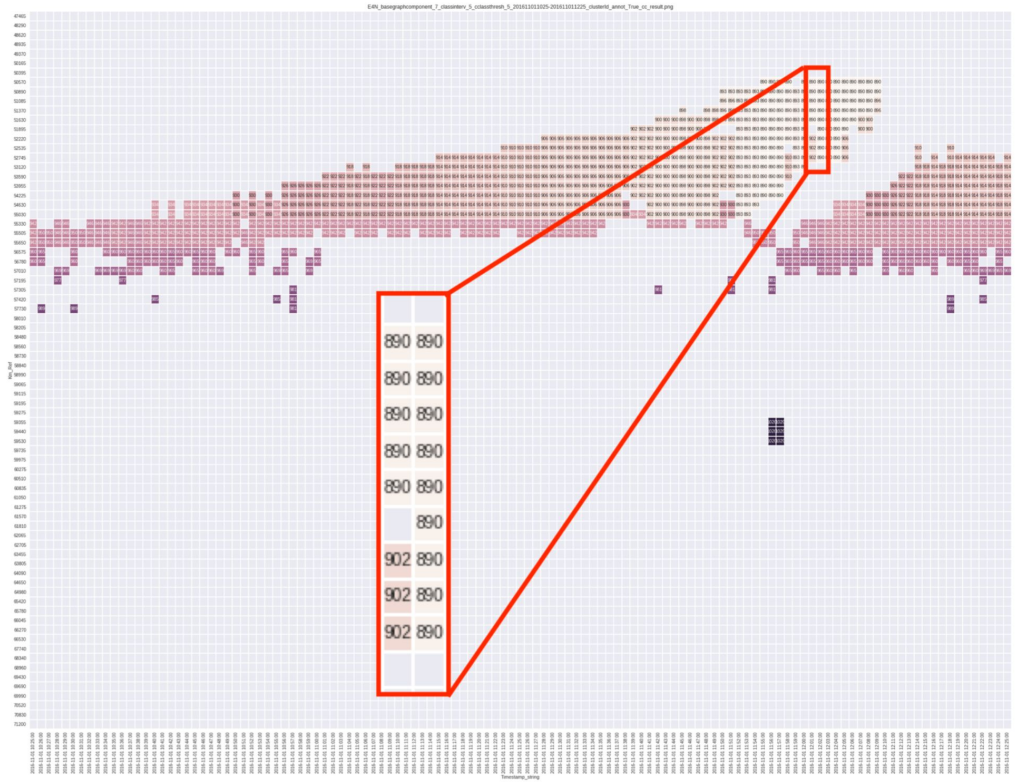
Queue tracker (through time)

- Implemented as separate streaming system that reads CC/Dengraph results from Kafka
- Calculate Jaccard distance between clusters at time t and clusters at time t-1, pairwise
- Distance threshold 1
  - If Jaccard distance < 1 then we found a parent queue
  - Only one sensor needed in overlap
- Naive, scalability concerns (pairwise comparison - O(n^2))
  - Worst case - each sensor in its own cluster (won't happen)
  - 2037 sensors at most in the graph (2016-5-9 onwards)
  - N(N-1)/2 = 2.073.666 cluster comparisons
  - CC worst case 518.671 (2 vertices per cluster)
  - Dengraph worst case 230.181 (3 vertices per cluster)
- Run only on *final* results for each pair of minutes

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$$d_J(A,B) = 1 - J(A,B)$$

Queue tracker (through time)



Accuracy evaluation

- Congestion detection algorithms evaluated w.r.t. percentage of labeled queues and shockwaves they detect (recall)
  - As determined by visual inspection
  - Domain expert approach
- Criteria
  - Detected shape comforms to shape in underlying data
    - Emphasis on upstream congestion boundary (EOQ)
  - Boundaries are clear
  - Noise at boundary minimal

	Found queues	Queue recall	Found s.waves	S.wave recall	WSM
CC (c. class 1)	1	6.3%	0	0%	0.05
CC (c. class 3)	5	31.3%	1	6.7%	0.25
CC (c. class 5)	**15**	**93.8%**	10	66.7%	**0.87**
CC (c. class 7)	13	81.3%	12	80%	0.81
CC (c. class 9)	9	56.3%	**14**	**93.3%**	0.66
DG ($\epsilon = 0,025$)	3	18.8%	10	66.7%	0.31
DG ($\epsilon = 0,03$)	4	25%	11	73.3%	0.37
DG ($\epsilon = 0,035$)	13	81.3%	11	73.3%	0.79
DG ($\epsilon = 0,04$)	14	87.5%	6	40%	0.76
DG ($\epsilon = 0,045$)	12	75%	2	13.3%	0.60
DG ($\epsilon = 0,05$)	5	31.3%	0	0%	0.23

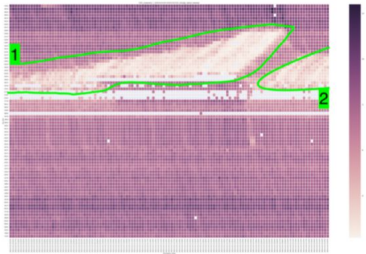
Accuracy evaluation.
16 queues, 15 shockwaves in labeled data set.
WSM 75% weight on queues, 25% weight on shockwaves

Congested components results

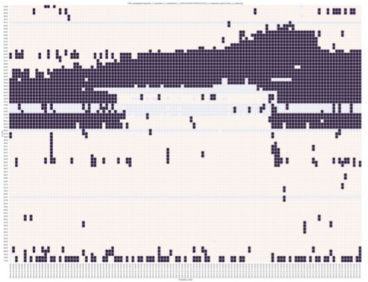
Run with different congestion class thresholds (1-9)

Congestion class threshold:

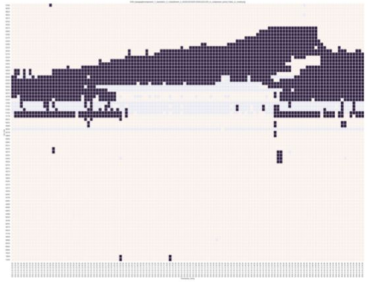
- 1 - No queue visible, noise
- 3 - Queue 1 detected
- 5 - Queue 1 and 2 detected



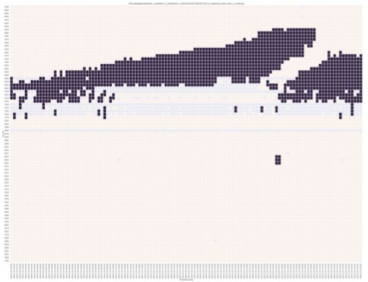
(a) Measured average speed values. Two queues delimited by green borders.



(b) Congestion class threshold 1



(c) Congestion class threshold 3

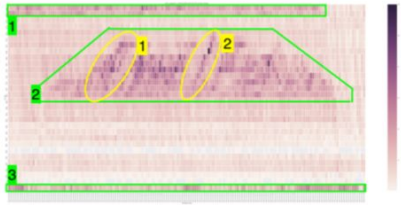


(d) Congestion class threshold 5

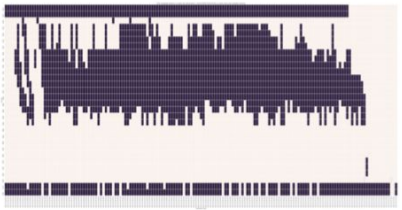
Test congestion pattern 3
E4N, component 7, 2016-11-01 10:25-12:25

Dengraph results

- Run with different ϵ values
  - Neighborhood radius (distance)
- η = 2 constant
  - min number of vertices in neighborhood
- ϵ = 0,05
  - Queues 1 and 3 detected, no shockwaves
- ϵ = 0,04
  - All queues detected, no shockwaves
- ϵ = 0,03
  - Queue 2 detected, both shockwaves detected

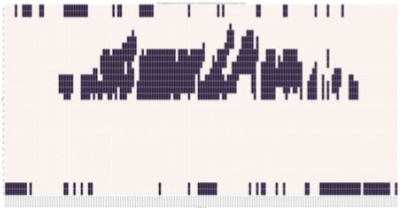


(a) Measured density values. 3 queues delimited in green, 2 shockwaves in yellow.



(b) $\epsilon = 0,05$

(c) $\epsilon = 0,04$

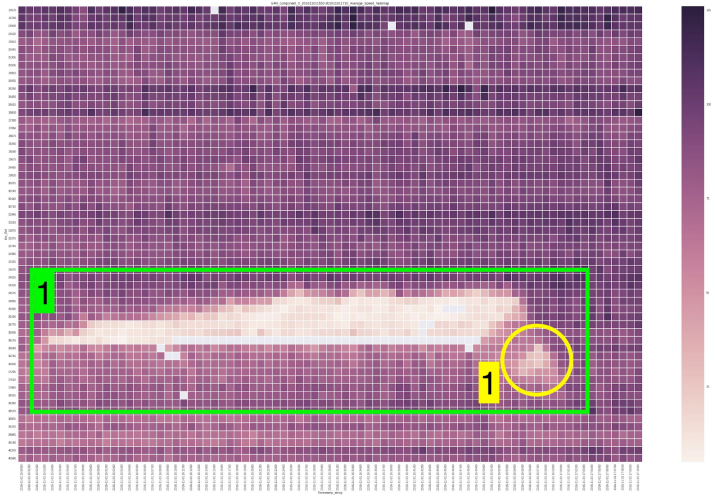


(d) $\epsilon = 0,03$

Test congestion pattern 6
E18O, component 0, 2016-11-01 05:00-07:40

Performance evaluation

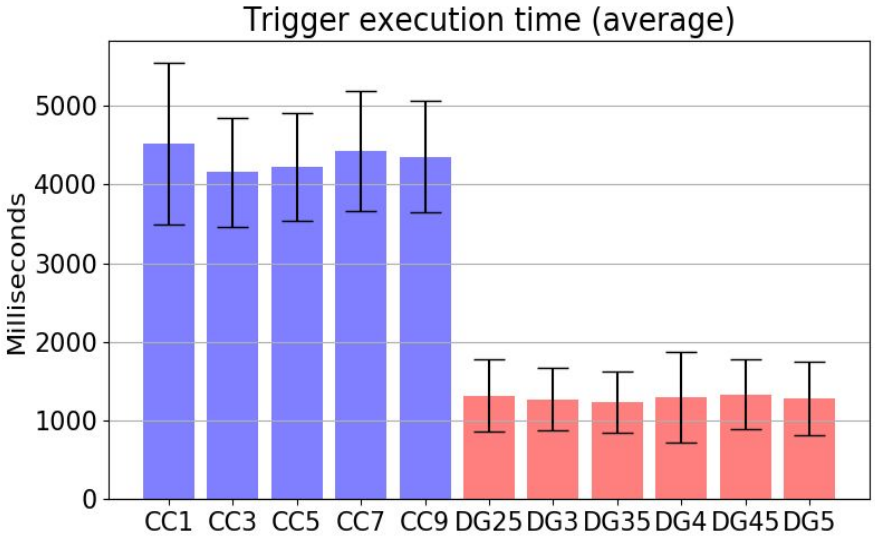
- Algorithms run on data from 2016-11-01 15:50-17:10
- Evaluation w.r.t.
  - Time to process each minute
  - Throughput (rows/sec)
  - Memory footprint (for each minute)
- Sensor measurements from **entire graph** is streamed
  - Average 1837 measurements per minute (after filtering errors)
- Evaluation on local machine
  - 2.5 GHz quad core, 16GB RAM, Spark v2.3.1 (8 threads), Kafka v1.1



Test congestion pattern 1.
E4N component 0. 2016-11-01 15:50-17:10.

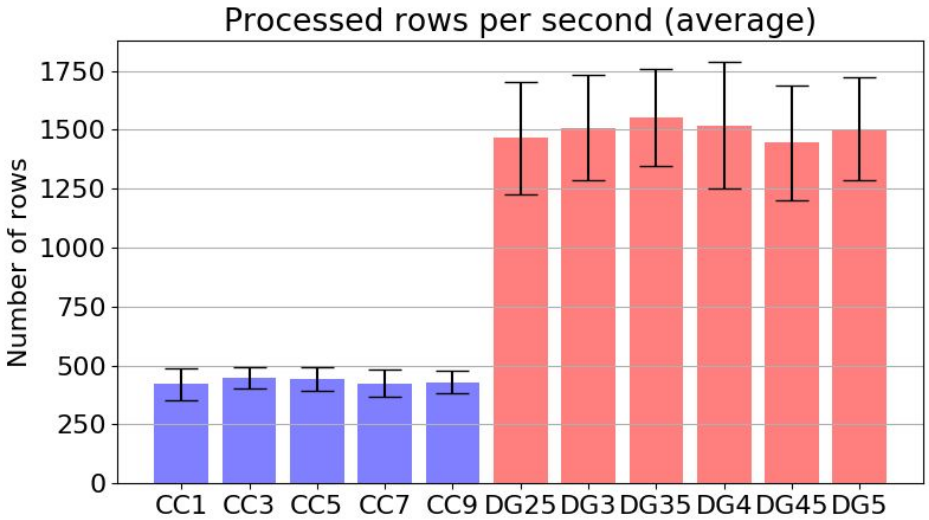
Trigger execution time

- Stream source set up so that entire minute's worth of data processed in a trigger
- Average values over the 80 minutes of the time span
- Time measures entire pipeline, not just core algorithm
- Dengraph twice as fast



Processed rows per second

Dengraph more than twice as fast



Memory usage per trigger

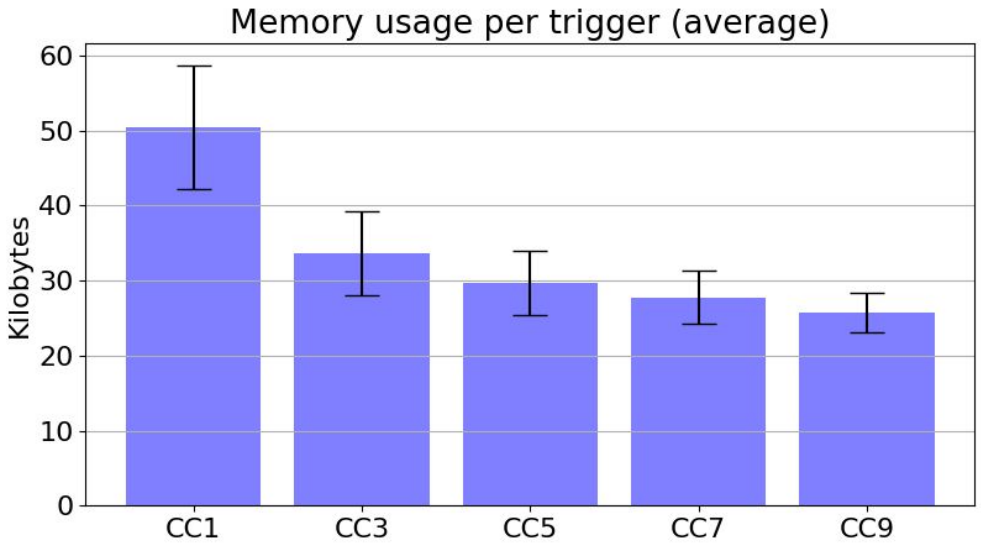
State from two adjacent minutes in memory at any given point

Congested components

- Clear trend
- Less memory with higher congestion threshold

Dengraph

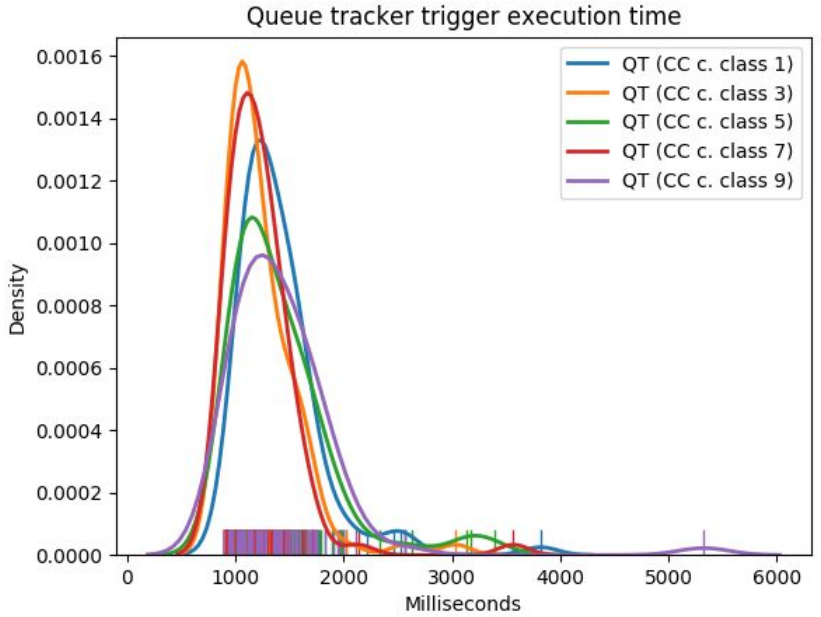
- Constant at 965.71 KB on average



Queue tracker - Trigger execution time

Uneven work performed in each trigger

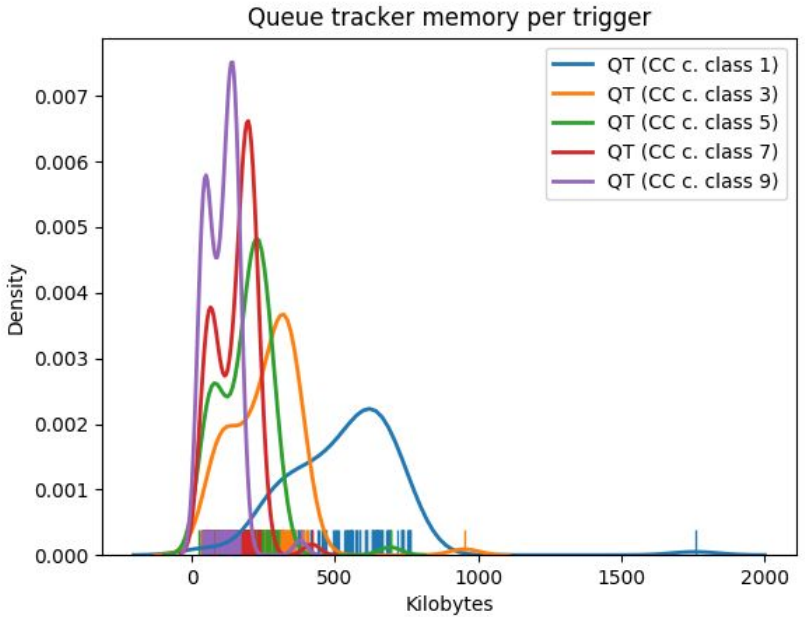
- Only computes parent clusters when all final rows from pair of minutes to be received
- Otherwise, just accumulates final rows in state
- May compute parent clusters for a number of minute-pairs in same trigger



Queue tracker - Memory per trigger

Clear trend

- Less memory with higher congestion threshold



Conclusions

- Congested components better accuracy overall
  - Able to detect queues with accuracy at best up to ≈94% and shockwaves ≈93%
- Dengraph detects at best ≈87.5% of queues and ≈73.3% of shockwaves
- Dengraph however more than twice as fast
  - ≈1.5 seconds to process each minute while taking up more space
- Tradeoff between detection of queues (congestion areas) and shockwaves (detail in congestion areas)
- Louvain modularity not suitable for simple road network graph
- Hierarchical data clustering approach feasible
  - Two phases needed
    - Clustering (incremental)
    - Connected components

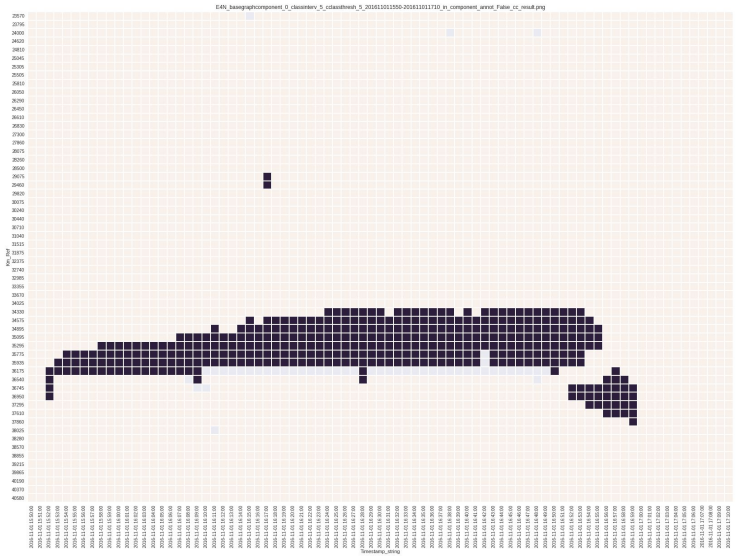
Future work

- Scalability considerations
  - Evaluation and graph partitioning
- Ground truth and accuracy evaluation strategy
  - Traffic simulators
  - Per sensor congestion labelling by human expert
- End-of-queue warning system
  - Use reachability graph to send warnings to sensors/cars approaching end-of-queue
- Dedicated end-of-queue detection system
  - Detect high risk jumps in density or drops in average speed
- Integrate with streaming predictions
  - Predict congestion, existence and movement
- Evoluationary clustering
- Continuously update minimum free flow speed per sensor

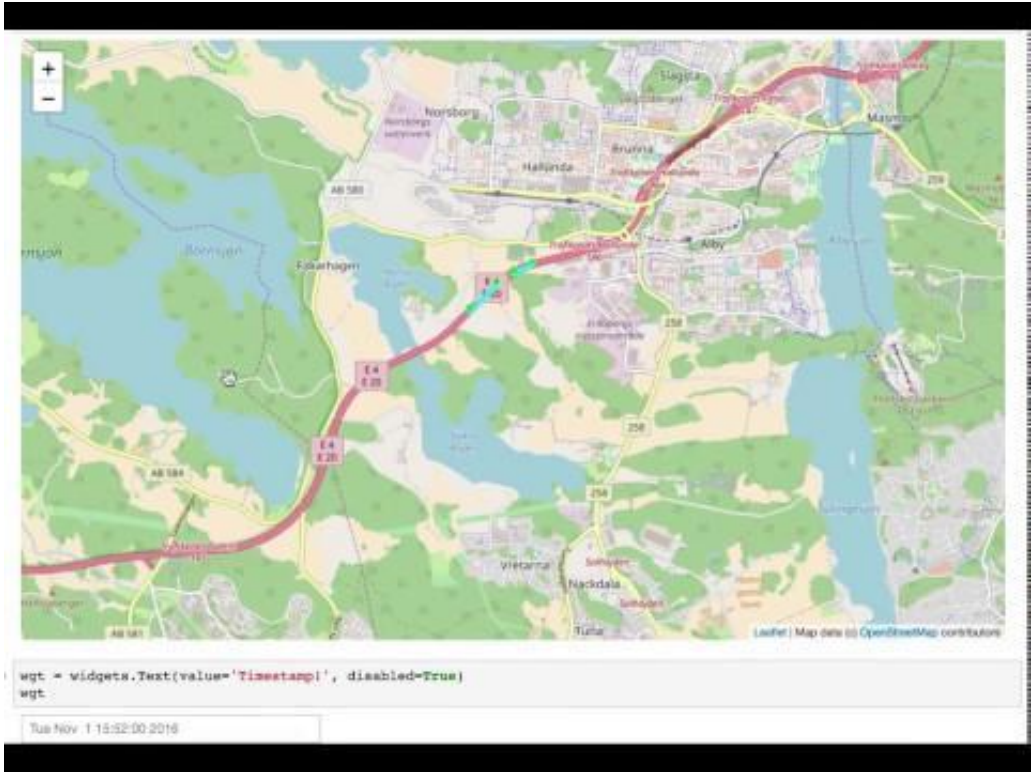
Thank you!

Appendix

Video visualization



E4N, component, 0, 2016-1101 15:50-17:10
Congested components, cclass 5



Louvain modularity - procedure

2 phases

Phase 1

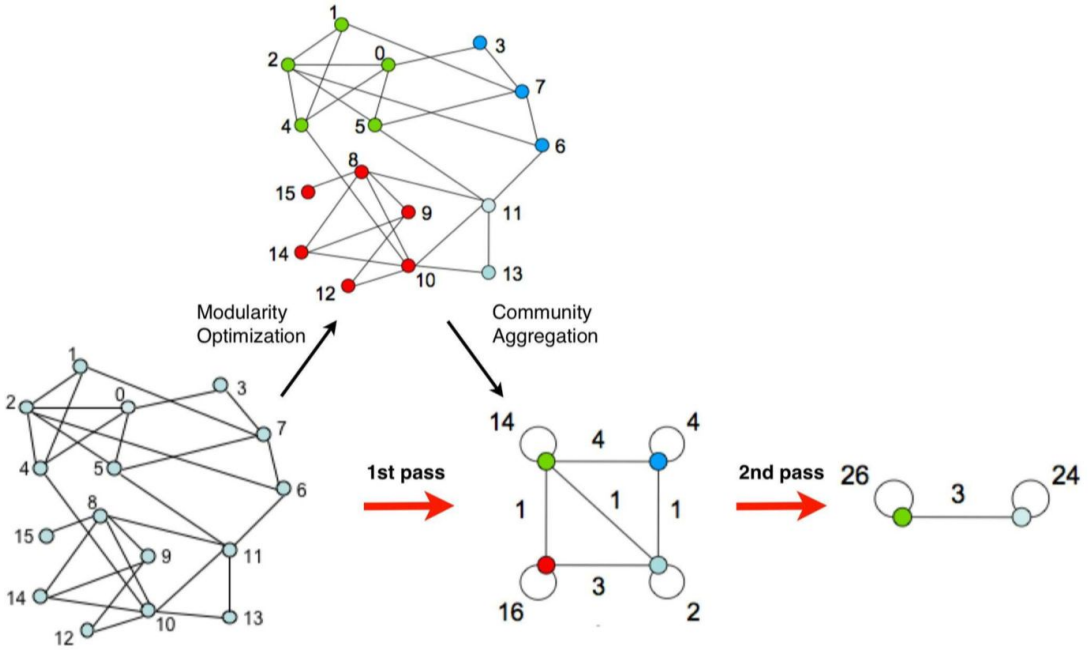
- For each node - move into community of neighbor which gives the greatest gain in modularity
- Repeat until no further gain in modularity can be achieved

Phase 2

- New graph constructed
- Communities found in phase 1 are the vertices
- The weights of the edges between the community-vertices in this new graph are the sum of the weights of edges between vertices in the two communities that the community-vertices we are linking represent

Repeat until no changes made in phase 1 and modularity has been maximized

Louvain modularity



Blondel et al.

Weighted modularity

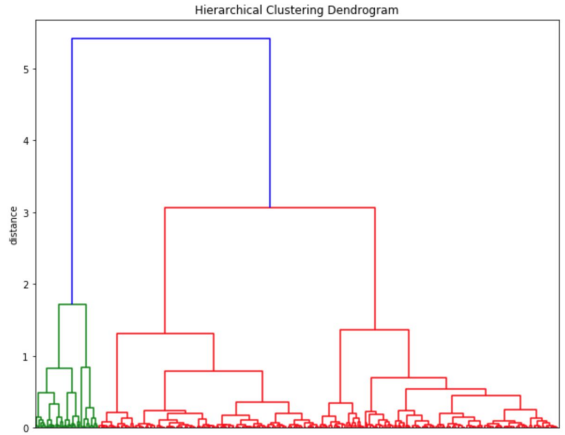
$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta\Big(c_i, c_j\Big),$$

A_{i,j} := weight of edge connecting i,j
k_i = sum of all weights connecting to i
C_i = community of i
delta = 1 if both same, 0 otherwise
m = ½ * sum_{i,j} A_{i,j} i.e. sum of all weights / half

(Sum of all weights within communities - (sum of all weights to i * sum of all weights to j)/sum of all weights)
/ sum of all weights

Hierarchical clustering Ward's variance minimization

Hierarchical clustering
- Start with all in separate clusters
- Merge clusters closes to each other depending on distance metric - stepwise
- Builts hierarchy (dendrogram)



Ward's variance minimization distance function:

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2}$$

where u is the newly joined cluster consisting of clusters s and t, v is an unused cluster in the forest, $T = |v| + |s| + |t|$, and $|*|$ is the cardinality of its argument. This is also known as the incremental algorithm.

Dengraph positive updates

Positive updates can lead to

- Cluster creation
  - New core vertex
  - Create new cluster
- Cluster absorption
  - Noise vertices now in core vertex neighborhood
  - Noise vertices absorbed to core vertex’s cluster
- Cluster merge
  - Core vertices from different clusters now in each others neighborhoods
  - Merge clusters

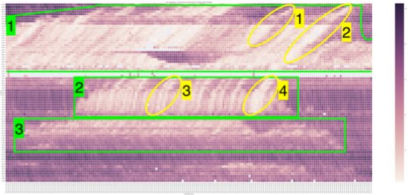
Congested components results

Run with different congestion class thresholds (1-9)

Accuracy evaluation:

Congestion class threshold

- 5 - All 3 queues, shockwaves 2,3,4
- 7 - Queue 1,2 detected, all shockwaves detected
- 9 - Queue 1 detected, shockwaves 1,2,4 clear



(a) Measured average speed values. 3 queues delimited in green, 4 shockwaves in yellow.

(b) Congestion class threshold 5

(c) Congestion class threshold 7

(d) Congestion class threshold 9

Test congestion pattern 4
E4N, component 7, 2016-11-01 13:15-16:15

The traffic sensor graph

Time-span graphs

- Sensors added and removed throughout the years
- Time-span graphs to accurately represent the road system at any point
- Latest graph the largest, 2037 sensors and 2077 edges

Graph no.	Valid from	Valid to	No. of sensors
1	2001-1-1	2015-2-25	1843
2	2015-2-26	2015-11-10	1845
3	2015-11-11	2016-3-26	1917
4	2016-3-27	2016-3-28	1994
5	2016-3-29	2016-5-8	1976
6	2016-5-9	onwards	2037