2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications

Evaluation of the Use of Streaming Graph Processing Algorithms for Road Congestion Detection

Zainab Abbas*, Thorsteinn Thorri Sigurdsson*, Ahmad Al-Shishtawy†, Vladimir Vlassov*
*KTH Royal Institute of Technology †RISE Research Institutes of Sweden
Stockholm, Sweden
*{zainabab, ttsi, vladv} @ kth.se †ahmad.al-shishtawy@ri.se

***Abstract*—Real-time road congestion detection allows improving traffic safety and route planning. In this work, we propose to use streaming graph processing algorithms for road congestion detection and evaluate their accuracy and performance. We represent road infrastructure sensors in the form of a directed weighted graph and adapt the Connected Components algorithm and some existing graph processing algorithms, originally used for community detection in social network graphs, for the task of road congestion detection. In our approach, we detect Connected Components or communities of sensors with similarly weighted edges that reflect different states in the traffic, e.g., free flow or congested state, in regions covered by detected sensor groups. We have adapted and implemented the Connected Components and community detection algorithms for detecting groups in the weighted sensor graphs in batch and streaming manner.**

We evaluate our approach by building and processing the road infrastructure sensor graph for Stockholm's highways using real-world data from the Motorway Control System operated by the Swedish traffic authority. Our results indicate that the Connected Components and DenGraph community detection algorithms can detect congestion with accuracy up to $\approx$ 94% for Connected Components and up to $\approx$ 88% for DenGraph. The Louvain Modularity algorithm for community detection fails to detect congestion regions for sparsely connected graphs, representing roads that we have considered in this study. The Hierarchical Clustering algorithm using speed and density readings is able to detect congestion without details, such as shockwaves.

***Index Terms*—Streaming, Graph Processing, Congestion, Community Detection, Connected Components**

I. Introduction

Congestion in road traffic networks poses several problems, such as increased pollution and fuel consumption [1] and reduced traffic safety [2]. Congestion mitigation strategies are therefore an important part of the operation of a traffic system. The focus of this research is on evaluating the use of streaming graph processing algorithms for real-time congestion detection and tracking to improve traffic safety and route planning. This enables mechanisms that communicate in real-time relevant information to drivers about the current traffic conditions in order to improve drivers' situational awareness.

In order to detect congestion in real-time, we propose to represent road infrastructure traffic sensors as a directed weighted graph and to apply streaming graph processing algorithms on it to detect congestion. We adapt the Connected Components algorithm [3] and some existing graph processing algorithms originally used for community detection [4] in social network graphs, for the task of road congestion detection.

In graph-based community detection algorithms, the communities are formed by grouping densely connected vertices. In our case sensors placed on the road network are represented as vertices with edges connecting neighbouring sensors. The density of edges does not reflect the communities, but rather the weights of edges (sensor readings) can be used for forming "communities" of sensors-vertices with similar edge weights (readings). Using these weights we want to see if neighbouring sensors behave in a similar way, i.e, showing similar readings, and can form communities representing different states of traffic, i.e, free flow or congested state. Thus, a detected sensor community reflects the traffic state in a period of time and a road region covered by the sensor community. Also, if sensors connected by edges with similar weights are located next to each other, then the Connected Components algorithm can be used to find components that are sub-graphs of connected sensors representing a specific traffic state, such as free flow or congested state in a corresponding region.

The main contributions of our work are as follows.

- We propose to represent the road infrastructure traffic sensors that measure the average flow and speed of vehicles per minute in the form of directed weighted graphs that allows detecting congestion regions in a timely manner.
- We propose to adapt and use streaming graph processing algorithms, namely Connected Components (CC) and Community Detection (CD) algorithms, for congestion detection by analysing the weighted sensor graph. Using CC and CD algorithms allows detecting sensor groups connected by edges with similar readings of speed or density. The detected sensor groups reflect different traffic states, e.g., free flow or congested state.
- We have adapted, implemented, and evaluated both community detection and Connected Components algorithms for congestion detection in batch and streaming modes.
- We provide an open source implementation[1] of graph processing algorithms using Apache Spark [5].

[1]https://github.com/thorsteinnth/road-congestion-detection

978-1-7281-1141-4/18/$31.00 ©2018 IEEE
DOI 10.1109/BDCloud.2018.00148

Main Findings: Results of our evaluation experiments indicate that Connected Components and community detection algorithms can be adapted to analyse traffic states on sensor graphs. Some algorithms show a rather high detection accuracy, while some are unable to detect congestion regions. In particular, our evaluation shows that the Connected Components algorithm and the DenGraph community detection algorithm can detect congestion with accuracy at best up to $\approx$ 94% for Connected Components and up to $\approx$ 88% for DenGraph. The Louvain Modularity algorithm for community detection fails to detect congestion regions for sparsely connected graphs, representing roads that we have considered in this study. The Hierarchical Clustering algorithm is able to detect congestion but without details, such as shockwaves.

Structure: The remainder of the paper is organized as follows. Section II contains necessary background, Section III explains the algorithms adapted and implemented in our work. Section IV describes implementation of our work, followed by Section V explaining the evaluation experiments and results. Finally, discussion and related work are presented in Section VI and conclusion and future work in Section VII.

II. Background

A. Traffic Flow Theory

Traffic flow theory explains the movements of traffic streams on the road system in terms of three main variables [6]: flow f (vehicles/time unit), average speed v (distance/time unit) and density d (vehicles/unit distance). The relationship between these variables in given by the equation below:

$$f = d \times v \tag{1}$$

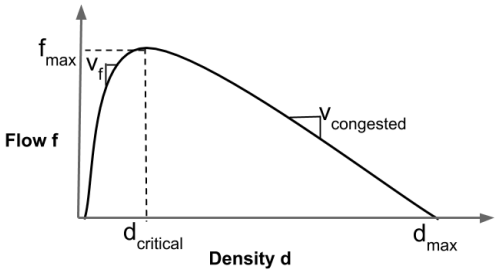


Fig. 1: The fundamental traffic flow curve

Fig.1 shows the traffic flow theory plot of flow f versus density d. The vehicles move with a free flow speed V_f in the low-density area shown on the positive slope of the curve, until the density reaches its limit $d_{critical}$ where the flow is maximum f_{max}. Beyond this point, congestion takes over, where vehicles' speed tends to decrease and the density keeps on increasing until it reaches its limit, i.e, d_{max}.

B. Traffic Congestion

Traffic flow can be either "free" or "congested" [7]. It is said that traffic is in "free flow" when it is possible for vehicles to drive, change lanes, overtake, and in general perform any maneuver the driver wishes [8]. Congested traffic flow can then be defined opposite to the free flow, i.e. when the conditions on the road do not allow for the free movement of vehicles.

C. Congestion Detection

A congestion threshold can be determined by identifying the empirical maximum point of free flow f_{max}, for a given traffic sensor from its historical data using the fundamental flow theory curve. The slope of a line drawn from the origin of the fundamental diagram to the empirical maximum flow point f_{max} gives the empirical minimum free-flow speed, according to equation 1. Sensor measurements on the left side of this line (speed higher than the minimum free flow speed) are then taken to belong to free-flow, while sensor measurements falling on the right side of the line are taken to belong to congestion.

To estimate the congestion magnitude, sensor measurements are classified into *congestion classes*, based on how far below the minimum free flow speed the measured average speed is in increments of 5 km/h. Fig.2 shows an empirical fundamental diagram, with a minimum free flow speed line and data points colored to indicate the magnitude of congestion.

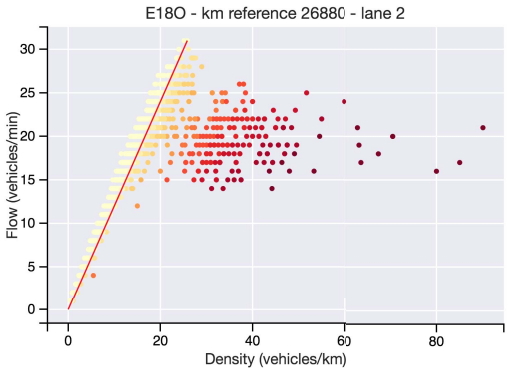


Fig. 2: Empirical fundamental diagram with minimum free flow speed line. Data point color represents congestion magnitude. Darker shades of red represent more severe congestion.

Density d can also be used as an indicator of congestion. While the critical density can be extracted from historical data on a per-sensor basis in the same way as described above, the critical density does not vary significantly between different locations in the road network. The critical density has almost the same value for different highways [9]. According to The Highway Capacity Manual from 2010, the maximum capacity for a freeway segment is 45 vehicles per mile per lane [10], translating to just under 28 vehicles per kilometer per lane.

Both average speed measurements as well, as density measurements, are used for congestion detection in our work.

D. Traffic Queues and Shockwaves

Congestion results in a buildup of traffic queues and shockwaves. A traffic queue can be defined as a row of vehicles waiting to be served, with the queue length usually being

defined as the number of vehicles waiting to be served [11]. In our work, a series of adjacent sensors experiencing congestion is taken to represent a traffic queue.

A shockwave is defined as a change or discontinuity in traffic conditions [10]. More precisely, shockwaves are a propagation of a change in flow and density [11], travelling through the road network at a certain speed. The build-up of congestion, where the end of a queue extends upstream through the road network with time, is an example of a shockwave.

E. Graph Based Analysis

Graphs are used to represent the relation between the data elements. We propose to represent the road infrastructure sensors in form of graphs because the readings recorded from sensors placed next to each other on the road network are correlated due to the spatiotemporal dependencies present in the traffic data. For example, if a traffic jam starts to build-up near a group of sensors then this traffic queue will propagate towards the sensors downstream. The sensor readings in the area of traffic jam will reflect the propagation of congestion state downstream.

Several graph processing algorithms are out there used for various purposes. Our work deals with detecting congestion which propagates through the traffic network affecting a group of sensors (in terms of the readings recorded by these sensors) located in the area where congestion takes place. These sensors will show high density and low-speed values. We use community detection and Connected Components algorithms to capture this trend in the infrastructure sensor graph.

III. Algorithms

We have implemented the Connected Components algorithm and community detection algorithms for congestion detection first in a batch fashion. After that, we picked the ones showing promising results to be executed in a streaming fashion. In general, for community detection algorithms, the communities are formed by grouping the vertices together if they are densely connected. In our case sensors are represented as vertices with edges connecting neighboring sensors. The density of edges does not reflect the communities, but rather the weights of edges (sensor readings) are used for forming "communities" of sensors with similar edge weights. The formed communities represent different traffic states, i.e, free flow state or congested state. Also, the Connected Components algorithm is used to find components that are sub-graphs of connected sensors representing a specific traffic state. The detail of these algorithms is as follows.

A. Batch Based Algorithms

1) Connected Components: The Connected Components algorithm finds the sub-graphs in which all vertices are connected by paths [3]. We use this algorithm to identify Connected Components of *congested sensors*, i.e, the sensors with measurements indicating congestion on road, in the infrastructure sensor graph. In the implementation, edges of the base graph are first weighted with average speed measurements taken from the destination vertex sensor. Then the edges with weights less than the congestion threshold (Section II-C) are removed. After that, the Connected Components algorithm is run on the remaining sub-graphs, which results in assigning the same ids to the *congested sensors* present in the same sub-graph. These Connected Components of sensors represent the traffic queues. Since the traffic sensors give measurements every minute, a new graph is generated for every minute of data (starting with the full graph and then removing edges with weights less than the congestion threshold).

2) Louvain Modularity: The Louvain Modularity algorithm [12] is a community detection algorithm. We use this algorithm to identify communities of *congested sensors*. In the implementation, first, each edge in the base graph is weighted with the measured density value from the destination vertex sensor. Then the Louvain Modularity algorithm is run on the weighted graph. This algorithm results in generating groups or communities of sensors representing either the congested areas on the road or the free flow areas.

3) Hierarchical Clustering: Hierarchical Clustering [13] is one of the traditional methods of community detection. In the implementation, the sensors are first clustered based on their average speed and density measurements, irrespective of the base sensor graph. Then the base graph is used to find Connected Components in the clusters to find sensors that do not only have similar measurements but also are connected. The method cannot differentiate between congested or free flow sensor clusters because it is unsupervised.

4) DenGraph: The DenGraph algorithm [14] is a density-based community detection algorithm. In the implementation, first, the edges of the base graph are weighted with sensor readings, i.e, density values, at the destination vertex. In this implementation, the vertices are clustered together based on the distance between them. We used the following rules for assigning distance between two vertices, u and v in the graph:

$$dist(u,v) = \begin{cases} 0 & u = v \\ \frac{1}{W_{u,v}} & \exists (u,v) \in E \\ \textit{undefined} & \textit{otherwise}, \end{cases} \tag{2}$$

Where, $W_{u,v}$ is the weight of the edge between vertices u and v. Note that there is at most one edge between each pair of vertices in the base graph. The distance between pairs of vertices that are not connected in the graph is undefined.

As this is the non-incremental version of the algorithm, it is run on the entire set of weighted edges from the graph (weighted with the sensor measurements from a single minute). The algorithm essentially does a depth-first search from all core vertices, i.e, the vertices having η neighbours in a radius ϵ, in the graph, finding density-connected vertices and assigning them the same cluster ID as the core vertex at the start of the density-connected chain of vertices.

B. Streaming Based Algorithms

1) Connected Components: The streaming Connected Components algorithm [15] is a union-find algorithm that operates on a stream of edges. In the streaming implementation, first, the edges containing a value less than the congestion threshold are filtered out. Next, the single-pass Connected Components algorithm is run over the remaining edges. A data structure is maintained as "memory state" for recording Connected Components. For every input edge, it is checked if the end-vertices belong to the components present are present in the memory state or not. Based on that the merging and assignment of components' IDs are done. Algorithm 1 shows the pseudocode of streaming Connected Components, where M is the memory state containing component IDs and the corresponding vertices belonging to the component.

Input: Stream of filtered edges E
Output: Components IDs for each vertex in E
begin
  foreach *edge* (u, v) **do**
    if M *contains neither* u *nor* v **then**
      `/* Neither vertex has been seen before */`
      create new component with ID `min(u.ID, v.ID)` and assign u and v to it
    end
    else if M *contains both* u *and* v **then**
      `/* Both vertices have been seen before */`
      if u *and* v *are in different components* **then**
        merge the two components `c1, c2` and set the ID of the new component to `min(c1.ID, c2.ID)`
      end
    end
    else
      `/* Only one of the vertices has been seen before */`
      if u is in a component, add v to the same component, and vice versa
    end
  end
end

Algorithm 1: Streaming Connected Components

Since in our case, the sensor values are coming every minute, we reset the in-memory state M every minute for the incoming stream.

2) DenGraph: The incremental version of DenGraph works on a stream of edges. It maintains the memory state containing clustering result to incorporate each incoming edge. The effect an incoming edge can have on the clustering is either *positive* or *negative*. Positive changes refer to new edges being added, or the distance between two previously processed vertices being reduced so that they enter the ϵ-neighborhood of each other. A negative change occurs if the distance between two vertices that were previously within each other's neighborhoods increases so that they no longer fall in each other's neighborhoods, or the edge between them is removed entirely. The memory state is reset in the incremental DenGraph algorithm after processing each minute of sensor data. Since the state is reset, only positive changes to the clustering occur.

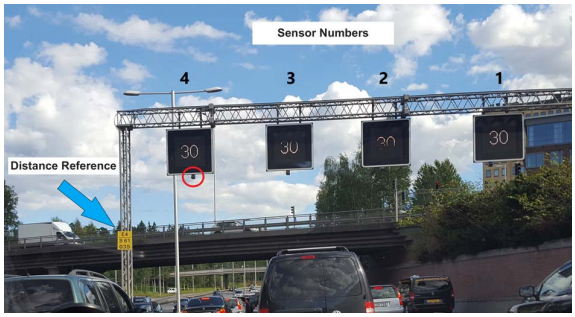


Fig. 3: Radar sensors on Stockholm highway.

In response to a positive change the following updates to the clustering may happen [14]: 1) A new cluster is created if vertices previously not part of a cluster become core vertices after the positive change, 2) Former noise vertices become part of an existing cluster, if they become density-reachable from a core vertex, and 3) A new neighborhood is formed which contains core vertices that belong to different clusters. These clusters are merged to form a new cluster.

Algorithm 2 shows the pseudocode for DenGraph, where M is the state maintained in memory indicating the vertices and their cluster information and `positiveUpdate()` corresponding to a positive change. We modified the DenGraph algorithm to not give overlapping communities.

IV. Implementation

1) DataSet: The traffic data set we use in our work is provided by the Swedish Transport Administration (Trafikverket). The data is taken from radar sensors placed on Stockholm highways, as shown in Fig. 3. These sensors are placed per lane every 150-400 meters on the highways recording the average flow f and speed v of vehicles passing the sensors per minute. In addition to the measurements, the data set contains 1) information about the road each sensor is placed on, 2) a kilometer reference giving the location of the sensor relative to the start of the road, 3) the lane each detector is monitoring, and 4) the GPS coordinates of each sensor.

2) Graph Construction: Sensor graph is created using the data set mentioned in Section IV-1. The constructed sensor graph is a directed graph where vertices represent the sensors, and the edges represent the road segments between the sensors. A path in the graph is a possible road path for vehicles on the highway. Two types of graphs were constructed, a *base graph* and a *reachability graph*. In the base graph, sensors present on the same lane are connected with edges, as in Fig. 4a. Paths

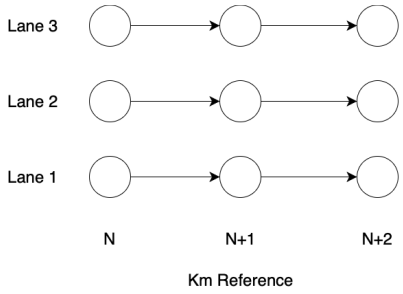
Function *PositiveUpdate(Vertex v, ϵ, η)*
  if $N_\epsilon(v) \geq \eta$ **then**
    /* This is a core vertex */
    Collect all distinct cluster IDs of core vertices in $N_\epsilon(v)$, along with $v.clusterId$ if v is a core vertex.
    if *setOfDistinctClusterIds.size == 0* **then**
      /* Create new cluster */
      Create new cluster. Assign v and all vertices $u \in N_\epsilon(v)$ to cluster. Mark all vertices $u \in N_\epsilon(v)$ as border vertices.
    end
    if *setOfDistinctClusterIds.size == 1 and v.state != core* **then**
      /* v and its neighborhood is absorbed to an existing cluster */
      Vertex v and all non-core vertices $u \in N_\epsilon(v)$ assigned to existing cluster. Mark all non-core vertices $u \in N_\epsilon(v)$ as border vertices.
    end
    if *setOfDistinctClusterIds.size > 1* **then**
      /* Merging of clusters */
      Create new cluster.
      Assign v and all vertices $\{u \in M \mid u.clusterId \in setOfDistinctClusterIds\}$ to new cluster. Assign all non-core vertices $u \in N_\epsilon(v)$ to new cluster and mark them as border vertices.
    end
    v.state = core
  end
end

Algorithm 2: Incremental DenGraph's PositiveUpdate method. Adapted from [16].

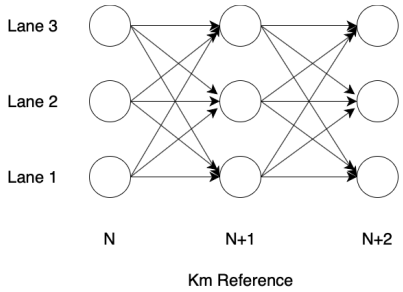
in the base graph depict paths that vehicles take if they drive without changing lanes. Whereas, in the reachability graph, the sensors on a given kilometer reference N are connected to all sensors at the next kilometer reference $N + 1$, as in Fig. 4b. Paths in the reachability graph depict the paths a vehicle might take if it changes lanes.

The graphs that we constructed were based on data taken from the year 2016 onwards. These graphs consisted of 2037 sensors, i.e, vertices, and 2077 edges.

3) Graph Usage: We have used the base graph is for congestion detection methods. It allows tracking the congestion in the lanes. The reachability graph can then be used to send warnings to the vehicles moving upstream towards the congested area. The graph is weighted and directed, where each edge contains a source and destination vertex. Edges are weighted based on the measurements of the destination vertex sensor. These weights are updated every minute during



(a) Base graph: Sensors on the same lane are connected with edges.



(b) Reachability graph: Each sensor is connected to sensors on the next Km reference.

Fig. 4: The two road network graphs over three lanes.

processing because the sensor readings are taken every minute. There were some sensors having no incoming edges to represent their readings. In order to record readings of these sensors having no incoming edges, dummy sensors were placed with outgoing edges towards them. The weight of these edges was based on the readings of these sensors.

4) System Architecture: Our work is implemented using Apache Spark [5]. It is a distributed data processing engine that provides support for both batch and stream processing over big data. We have used Spark GraphFrames for creating our sensor graphs and Spark Structured Streaming API for implementing the congestion detection algorithms.

V. Evaluation

A. Ground Truth

One of the challenges we faced in our work was finding the ground truth about congested queues because it is not directly measured in the data. Other works mentioned in Section VI use ground truth based on video recording of road traffic [17], traffic simulators to simulate the behaviour of real traffic for having access to ground truth [18], identifying and labeling the ground truth using domain experts or using data fusion techniques with external datasets such as accident reports [19].

In our work, we generate heat maps of measured values and use them as ground truth for comparing them with the congestion detection algorithm results. These maps are useful for showing the spatiotemporal pattern of measured traffic readings. We have selected various congested traffic areas on

the road and generated their heat maps for the evaluation of the congestion detection algorithms.

B. Experimental Setup

We performed our experiments using a local machine comprising of 2.5 GHz quad-core processor and 16 GB of RAM. We used Apache Spark v2.3.1 with 8 threads (using the local configuration) and with 1 GB memory allocated for the driver, along with Kafka v1.1. The congestion detection algorithms were written using with Java v1.8. Data preparation and analysis of results was performed with PySpark, Spark's Python API, using Python v3.6.3.

C. Batch Approaches

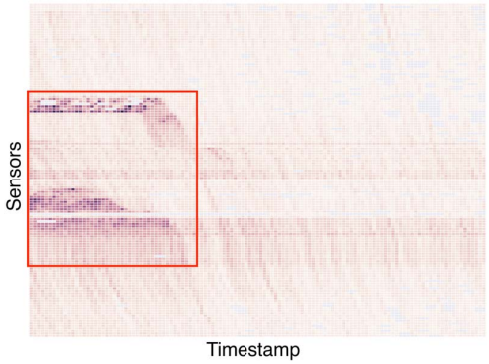
We show only results for Louvain Modularity and Hierarchical Clustering from batch approaches. The results for Connected Components and DenGraph are shown next in the streaming Section V-D since they are identical to the results generated when running in a batch fashion. The results of the batch algorithms were compared to the congestion pattern observed in the heat map of density values given below in Fig.5a. These measurements are taken from the road E4N, lane 1. The x-axis here represents the time and the y-axis represents the sensors ordered by km reference. The dark shades in the heat map depict high density, i.e, the congestion pattern flowing from left to right. Next, we explain the results from Louvain Modularity and Hierarchical Clustering on the same observed congestion pattern.

1) Louvain Modularity: The Louvain Modularity algorithm was run, minute-by-minute with one iteration per minute. Fig.5b shows the communities detected by the algorithm. We expect the sensors with readings showing congestion measurements to be in same community. Adjacent cells within a column, belonging to the same community, are shown with the same color. We can observe that the results from the Louvain Modularity algorithm do not reflect the observed congestion pattern in Fig.5a. These results make the modularity approach for community detection unsuitable for sparse graphs.

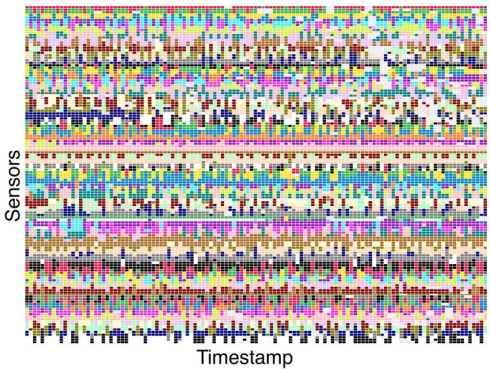
2) Hierarchical Clustering: Hierarchical Clustering was also run, minute-by-minute. The results for this algorithm, with the resulting hierarchical tree of clusters cut to generate two clusters (free flow and congested), are shown in Fig.5c. Adjacent cells within a column, belonging to the same cluster, are shown with the same color. The algorithm requires that the difference between the free flow and congested state, in terms of both density and average speed, to be large in order to precisely group the sensors. Finer details in the congestion pattern, such as the dissipation of the topmost queue in Fig.5a, are not detected without introducing considerable noise. Furthermore, the method requires two steps; a clustering step followed by the computation of Connected Components. Therefore, the Connected Components and DenGraph congestion detection methods are more suitable.

D. Streaming Approaches

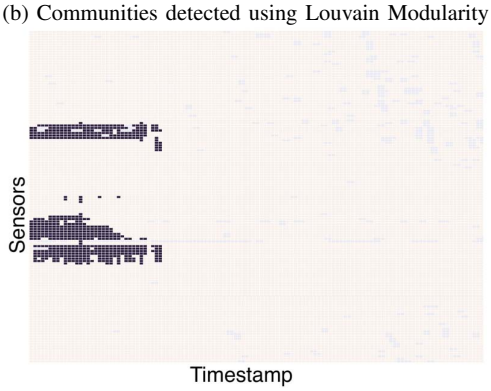
For evaluating the streaming algorithms, we chose eight congestion patterns from different times of the day on 2016-11-1. Since the Connected Components algorithm makes use of average speed values taken from the sensors, we compare the components generated by Connected Components with the average speed measurements heat map of a selected congested area, as shown in Fig.6a. The light shade here indicates the group of sensors showing low speed, i.e, high density and congestion. Whereas, results from DenGraph are compared to the density based heat map, shown in Fig.6c. The dark shade



(a) Observed congestion pattern from sensor measurement



(b) Communities detected using Louvain Modularity



(c) Communities detected using Hierarchical Clustering with average density within community $>$ 30 vehicles/km.

Fig. 5: Heat maps of measured (a) density values; Congestion detection results using (b) Louvain Modularity and (c) Hierarchical Clustering algorithms. For the highway E4N lane 1, 2016-11-01 16:20-18:20.

here indicates the group of sensors showing high density and congestion. In these maps, the x-axis represents the time and the y-axis represents the sensors ordered by km reference. The green boundary represents one queue and the yellow boundaries indicate three shockwaves.

1) Connected Components: The heat map generated using the Connected Components (CC) algorithm is shown in Fig.6b. The heat map shows similar congestion pattern as the one in the measured speed values in Fig.6a. Queue and shockwaves are visible in the heat map generated using CC.

2) DenGraph: The heat map generated using DenGraph (Fig.6d) shows similar congestion pattern as the one generated using the measured density values in Fig.6c. Queue and shockwaves are visible in the heat map generated using DenGraph.

E. Comparison of Streaming Approaches

According to our results, since the streaming Connected Components (CC) and DenGraph (DG) were able to detect congestion patterns in road traffic, now we want to compare their accuracy and performance. For the accuracy, we chose eight different congestion patterns containing 16 queues and 15 shockwaves and measured the number of queues and shockwaves correctly detected using these algorithms. For performance, we measure the trigger time of these algorithms.

1) Accuracy: Table I shows the accuracy evaluation for CC and DG using various input parameters for these algorithms. The recall indicates the actual queues found by the algorithm. WSM is the weighted sum computed using 75% recall fraction of queues and 25% recall of shockwaves. It is used to get an overall score of detecting queues and shockwaves.

According to our accuracy results, CC with class threshold 5 detects the highest number of queues with a greater recall. Also, CC with class threshold 9 is able to determine the maximum number of shockwaves. Overall the weighted score WSM for CC with class threshold 5 is higher than the others, indicating it to be better than the others in determining queues and shockwaves. However, there is a trade-off between queue and shockwave detection using various CC parameters. As we increase the class threshold beyond 5, the number of detected queues decrease and shockwaves increase.

2) Performance: To evaluate the performance of CC and DG we consider the following metrics: 1) The trigger execution time, i.e, the time taken to process per minute sensor data, 2) the throughput, i.e, the number of rows processed per second and, 3) the average memory used per trigger.

Fig.7a shows the trigger time of CC and DG in milliseconds. CC with class threshold 1 has the highest trigger execution time. Whereas, DG with $\epsilon = 0,035$ has the lowest trigger execution time. Overall, DG has low trigger execution time, which makes it faster than CC. For the throughout results plotted in Fig.7b, CC with class threshold 1 and 7 processes the lowest number of records. DG, on the other hand, has overall higher throughout with $\epsilon = 0,035$ giving the best throughput. Lastly, in terms of memory DG uses a constant amount of memory on average, i.e, 965.71 KB. The memory used by CC is plotted in Fig.7c indicating less usage of memory by CC compared to DG.

	Found queues	**Queue recall**	**Found shock-waves**	**Shock-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

TABLE I: Accuracy evaluation for Connected Components CC and DenGraph DG. The minimum number of nodes required in a neighborhood for it to be considered a cluster in Dengraph was kept constant at $\eta = 2$.

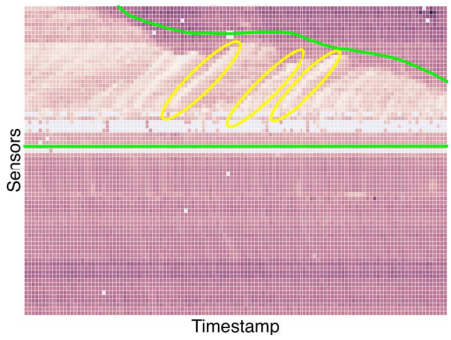
Findings: Our results indicate that 1) Louvain Modularity is unable to detect congestion patterns in the data, and Hierarchical Clustering can detect congestion patterns with a low level of detail 2) Connected Components and DenGraph can detect queues and shockwaves, with Connected Components showing better accuracy and being memory efficient, and DenGraph being faster with better throughput.

VI. Discussion and Related Work

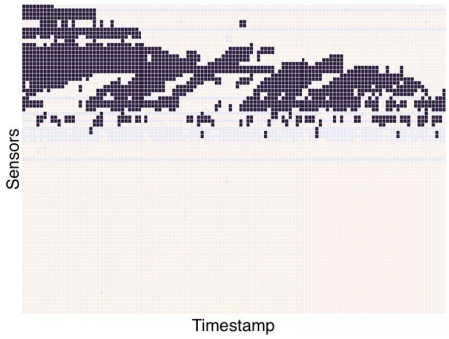
A number of interesting works exist for congestion detection. The most widely used congestion detection system is Google Maps; it uses GPS coordinates from users phones to track the speed for congestion detection. Besides this, Coifman [17] uses dual loop detectors to track the vehicles. Another work by Li et al. [18] uses a density-based algorithm to identify hot routes in the road network. This algorithm is also executed on a graph of the road network, with road segments as edges and intersection as vertices, and the algorithm is similar to the DenGraph algorithm used in our work. However, the graph in our work is different because we have a sensor graph, also we are exploring streaming algorithms for real-time queue detection. Table II presents a summary of congestion detection methods proposed in related and our work. We have not compared our work with related work by evaluation experiments because we have not got access to implementation and data sets of related work. Congestion detection techniques can also be used for predicting future congestion by first predicting traffic using different time-series prediction techniques [20], [21], and then detecting congestion in the predicted traffic.

VII. Conclusion and Future Work

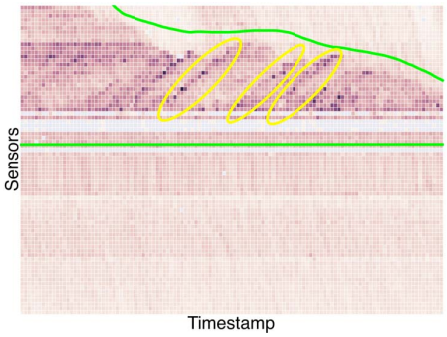
In this work, we have studied the use of stream processing algorithms for Connected Components and community detection to identify groups in weighted graphs representing road infrastructure traffic sensors. The detected sensor groups reflect traffic states, e.g, free flow and congestion state. We have adapted, implemented and evaluated four algorithms,



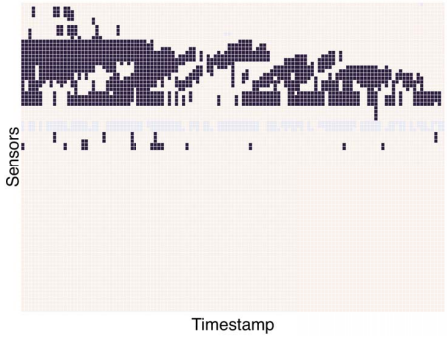
(a) Heat map of the measured speed values. A queue delimited by green border and three shockwaves in yellow border



(b) Congestion pattern from Connected Components algorithm using class threshold 7

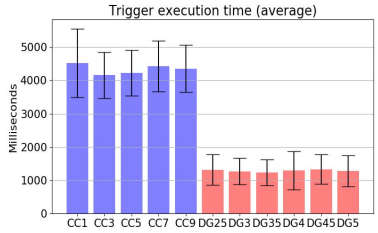


(c) Heat map of the measured density values. A queue delimited by green border and three shockwaves in yellow border

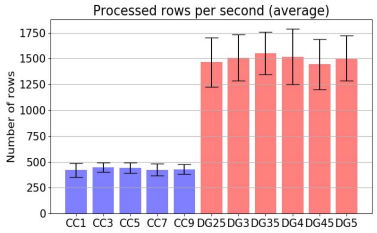


(d) Congestion pattern from DenGraph community detection algorithm using $\epsilon = 0,03$

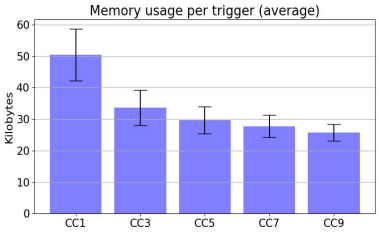
Fig. 6: Heat maps of measured (a) average speed and (c) density values; Congestion detection results using (b) CC and (d) DenGraph algorithms. For the highway E4N, 2016-11-01 06:10-08:10.



(a) Trigger execution time



(b) Number of rows processed per second



(c) Average memory used per trigger

Fig. 7: Performance comparison of Connected Components (CC) and DenGraph (DG)

namely, Louvain Modularity, DenGraph community detection, Connected Components, and Hierarchical Clustering, for congestion detection in road traffic. These algorithms were applied on directed weighted graphs created using sensors and their measured values obtained from the Motorway Control System operated by the Swedish traffic authority. Connected Components (CC) and DenGraph (DG) algorithms have shown good results in terms of detecting the congestion queues and shockwaves with CC using less memory and being more accurate, while DG being faster. CC takes longer to process the data, with about 4.5 seconds for processing one-minute records, while DG is more than twice faster taking about 1.5 seconds. In terms of accuracy, CC performs slightly better with 94% accuracy compared to DG with 88%.

Our evaluation shows that Louvain Modularity is unable to detect congestion regions and Hierarchical Clustering is able to detect congestion without details that include shockwaves. In particular, Louvain Modularity algorithm shows poor performance on sparse graphs. Using the Hierarchical Clustering algorithm requires that the difference between the free flow and congested state, in terms of both density and average speed, to be large in order to precisely group the sensors.

For future work, we suggest different dimensions that can be explored related to this work. Scalability is one major concern

	Data Source	Measured Parameters	Purpose	Method	Limitations	Evaluation
Google maps [22]	Mobile phones and probe vehicles	Speed values	Congestion detection	Not disclosed	Privacy concern	Not applicable
Coifman [17]	Loop detectors	Vehicle trajectory	Congestion detection	Vehicles measurements are matched between stations	Cannot detect small vehicles accurately	Up-to 50% for long vehicles
Anbaroglu [19]	Cameras	Link Journey Times	Non-recurrent congestion events	Measures Link Journey Times on a graph where cameras are vertices, road between them are edges	Cannot detect usual traffic jams	Link Journey Times higher than 40% of their expected values indicate a Non-Recurrent Congestion event
Li [18]	Probe Vehicles, traffic simulator	Vehicle trajectories and density values	Hot routes	Density based clustering on a graph where vertices are intersection or landmarks and edges are road segments between them	Synthetic dataset	Efficiently discovers hot routes
Our work	Radar sensors	Speed, flow and density values of traffic	Congestion detection	Uses Connected Components and community detection methods on a graph of road sensors	Cannot work with probe vehicles	Connected Components can detect congestion with accuracy at best up to 94% and DenGraph up to 88%

TABLE II: Congestion detection methods

especially because the number of sensors is increasing every year. Another subject for future work is to do end-of-queue detection for creating an efficient warning system for vehicles approaching the traffic jam queues. Lastly, evolutionary clustering can also be explored for keeping track of the queues in the road traffic. We also plan to explore applying this approach to other application domains where spatiotemporal data can be modelled in the form of weighted graphs.

ACKNOWLEDGMENT

This work was supported by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the Education, Audiovisual and Culture Executive Agency (EACEA) of the European Commission under FPA 2012-0030, by the project BADA: Big Automotive Data Analytics in the funding program FFI: Strategic Vehicle Research and Innovation (grant 2015-00677) administrated by VINNOVA the Swedish government agency for innovation systems, and by the project BIDAF: Big Data Analytics Framework for a Smart Society (grant 20140221) funded by KKS the Swedish Knowledge Foundation.

REFERENCES

[1] M. Barth and K. Boriboonsomsin, "Real-World CO2 Impacts of Traffic Congestion," p. 24.

[2] U. States., *Vehicle- and infrastructure-based technology for the prevention of rear-end collisions [electronic resource]*. National Transportation Safety Board Washington, D.C, 2001.

[3] J. Hopcroft and R. Tarjan, "Algorithm 447: efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, 1973.

[4] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[5] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[6] M. J. Lighthill and G. B. Whitham, "On kinematic waves ii. a theory of traffic flow on long crowded roads," *Proc. R. Soc. Lond. A*, vol. 229, no. 1178, pp. 317–345, 1955.

[7] B. S. Kerner, *The physics of traffic: empirical freeway pattern features, engineering applications, and theory*. Berlin: Springer, 2010.

[8] H. Rehborn and J. Palmer, "ASDA/FOTO based on Kerner's three-phase traffic theory in North Rhine-Westphalia and its integration into vehicles." IEEE, Jun. 2008, pp. 186–191.

[9] Y. Sugiyama, M. Fukui, M. Kikuchi, K. Hasebe, A. Nakayama, K. Nishinari, S.-i. Tadaki, and S. Yukawa, "Traffic jams without bottlenecksexperimental evidence for the physical mechanism of the formation of a jam," *New Journal of Physics*, vol. 10, no. 3, p. 033001, Mar. 2008.

[10] *Highway Capacity Manual: volume 1: concepts.* Washington: Transportation research board, 2010.

[11] L. Elefteriadou, *An introduction to traffic flow theory*, ser. Springer optimization and its applications. New York: Springer, 2014, vol. 84.

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[13] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[14] T. Falkowski, A. Barth, and M. Spiliopoulou, "Studying Community Dynamics with an Incremental Graph Mining Algorithm," p. 12, 2008.

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "Graph distances in the data-stream model," *SIAM Journal on Computing*, vol. 38, no. 5, pp. 1709–1727, 2008.

[16] T. Falkowski, "Community analysis in dynamic social networks," Ph.D. dissertation, Ph. D. dissertation, angenommen durch die Fakultat fur Informatik der Otto-von-Guericke-Universität, 2009.

[17] B. Coifman, "Identifying the onset of congestion rapidly with existing traffic detectors," *Transportation Research Part A: Policy and Practice*, vol. 37, no. 3, pp. 277–291, Mar. 2003.

[18] X. Li, J. Han, J.-G. Lee, and H. Gonzalez, "Traffic Density-Based Discovery of Hot Routes in Road Networks," in *Advances in Spatial and Temporal Databases*, D. Papadias, D. Zhang, and G. Kollios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 4605, pp. 441–459.

[19] B. Anbaroglu, B. Heydecker, and T. Cheng, "Spatio-temporal clustering for non-recurrent traffic congestion detection on urban road networks," *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 47–65, Nov. 2014.

[20] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results," *Journal of transportation engineering*, vol. 129, no. 6, pp. 664–672, 2003.

[21] Z. Abbas, A. Al-Shishtawy, S. Girdzijauskas, and V. Vlassov, "Short-term traffic prediction using long short-term memory neural networks," in *2018 IEEE International Congress on Big Data (BigData Congress)*, July 2018, pp. 57–65.

[22] "The bright side of sitting in traffic: Crowdsourcing road congestion data." [Online]. Available: https://googleblog.blogspot.com/2009/08/bright-side-of-sitting-in-traffic.html