Online Event Clustering in Temporal Dimension

Hoang Thanh Lam
IBM Research
Building 3, IBM Technology Campus
Damastown, Dublin 15, Ireland
t.l.hoang@ie.ibm.com

Eric Bouillet
IBM Research
Building 3, IBM Technology Campus
Damastown, Dublin 15, Ireland
bouillet@ie.ibm.com

ABSTRACT

This work is motivated by a real-life application that exploits sensor data available from traffic light control systems currently deployed in many cities around the world. Each sensor consists of an induction loop that generates a stream of events triggered whenever a metallic object e.g. car, bus, or a bicycle, is detected above the sensor. Because of the red phase of traffic lights objects are usually divided into groups that move together. Detecting these groups of objects as long as they pass through the sensor is useful for estimating the status of the toad networks such as car queue length or detecting traffic anomalies. In this work, given a data stream that contains observations of an event, e.g. detection of a moving object, together with the timestamps indicating when the events happen, we study the problem that clusters the events together in real-time based on the proximity of the event's occurrence time. We propose an efficient real-time algorithm that scales up to the large data streams extracted from thousands of sensors in the city of London. Moreover, our algorithm is better than the baseline algorithms in terms of clustering accuracy. We demonstrate motivations of the work by showing a real-life use-case in which clustering results are used for estimating the car queue lengths on the road and detecting traffic anomalies.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Data Stream, real-time monitoring, sensor network, clustering algorithms, transportation, SCOOT data, social good

1. INTRODUCTION AND MOTIVATIONS

1.1 Motivations

Our work is motivated by the SCOOT system[1] which is currently used for coordinating traffic lights in many cities. It is important to notice that although we use the SCOOT data as a motivation example the approach proposed in this paper can work well for any event stream data in general. The SCOOT system consists of thousands of SCOOT sensors. Figure 1 shows an example of a SCOOT sensor located a few hundreds metres from a traffic light. SCOOT sensors are designed for detecting moving objects in the nearby location. Whenever a moving object passes through the location of the sensor, an event is triggered and sent to a centralized computing system.

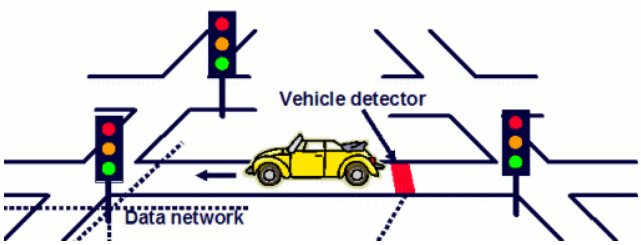

Figure 1: A SCOOT sensor (marked with a bold red line) is located hundreds of metres before a traffic light. An event is triggered when a car passes through the sensor.

Figure 2 shows a time series of binary bits received from a SCOOT sensor, where bit "1" indicates that a moving object is passing through the sensor and bit "0" means nothing is detected. An interesting pattern we can observe on this figure is that objects tend to move together in groups. An explanation for this pattern is due to the effects of the red phase of traffic lights which cuts down the long line of moving objects into smaller chunks.

Detection of groups of moving objects as long as they pass through the sensor is useful for monitoring the status of the road. For instance, the number of moving objects in each group can be used as an estimate of the car queue length on the road. Alternatively, when we have two sensors located at the downstream and the up stream location of a street, the clustering results can be used to simultaneously detect traffic anomalies in the corridor connecting two sensors.

1.2 The problem and our solutions

In general, finding groups of moving objects can be considered as an unsupervised classification problem in which the difference or the gap between observations of two consecutive moving objects can be classified either as a *separator*

[1] www.scoot-utc.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
Copyright 2014 ACM Copyright 2014 ACM 978-1-4503-3131-9/14/11 $15.000 http://dx.doi.org/10.1145/2666310.266639.

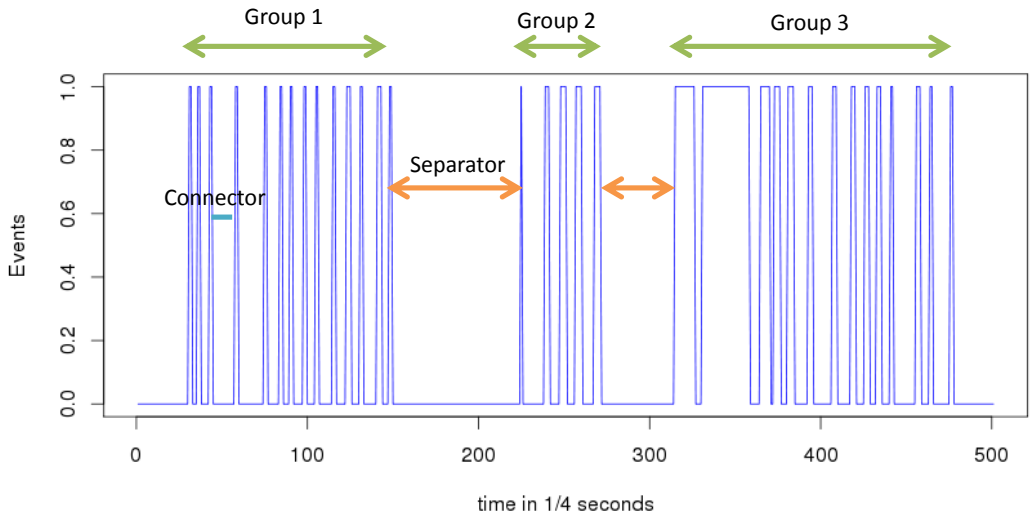

Figure 2: Data from a SCOOT sensor. Bit "1" indicates that a moving object is passing through the sensor and bit "0" means nothing is detected. Cars tend to move together in groups due to the effect of traffic lights. Gaps between groups are labelled as separators, while gaps between elements of a group are labelled as connectors. Separators seem to have larger value than connectors.

or a *connector*. Separators separate different groups while connectors connect objects in the same group.

For example, in Figure 2 we can see that there are three groups of moving objects. The gaps between groups marked with the label separators seem to be larger than the gaps between objects in the same groups. Based on that observation, a simple approach is to use one of the change detection techniques such as the CUSUM chart [3] to label the set of gaps. For instance, we can keep track of the mean value of the gaps seen so far, if a new gap value is much larger than the mean value, a change is detected and the gap is labelled as a separator otherwise it is labelled as a connector. This approach typically requires pre-set threshold parameters, e.g. significant levels. However, under the context of the SCOOT data, depending on the current traffic on the road (busy or not), periods of day (night versus rush hours) etc, the distribution of the gaps dynamically changes. In experiments, we show that simple algorithms relying on pre-set threshold parameters do not work well.

In this work, we approach the problem as an online clustering problem in the sliding windows model. We keep the most recent gaps in a sliding window and do clustering them in real-time using the k-mean algorithm, where k is set to 2. The cluster which has larger mean value than the other is labelled as a separator while the other is labelled as a connector. In order to tackle data distribution change, the gaps are labelled using accumulated votes from a set of the most recent windows. In doing so, our algorithm becomes less sensitive to initial parameter settings while being able to quickly adapt to distribution change.

An important observation is that the problem we are trying to solve is equivalent to the 2-mean clustering problem in one dimensional space. Using the straightforward implementation of the k-mean algorithm is a naive solution. In fact, we show that a much better clustering algorithm can be proposed to solve the problem exactly while being much more efficient than the straightforward implementation of the k-mean algorithm.

We validated our approach with a synthetic and a real-world dataset with known ground-truths. Experimental results showed that our algorithm outperformed the baseline algorithms including an algorithm based on mean value change detection and the state of the art algorithms CluStream [1] and ClusTree [11] in terms of clustering accuracy. Moreover, our method scaled up to a large dataset at the city-scale with fast updates from thousands of sensors simultaneously. Finally, we showed the motivation of the work with a real-world use-case with the SCOOT data in which we used clustering results to detect interesting anomalies, or to estimate car queue length.

This paper is organized as follows. Related work is discussed in section 2. Section 3 shows the problem formulation. Different algorithms are proposed in section 4. Experiment results are discussed in section 5. Finally, in section 6, we discuss some conclusions and potential future work.

2. RELATED WORK

Clustering multi-dimensional data in a data stream is a well studied problem. Most work in the literature [7, 13, 4, 6] assumes that only a single pass through the data is allowed and the dimension of the data is high. Therefore, approximation algorithms are preferred in this case because of the efficiency issue. However, in the context of the online event clustering problem, the dimensionality of the data is not a problem because the data only concerns the temporal dimension. Besides, in this work we show that the problem we are trying to solve can be solved exactly with a very efficient method. Therefore, the approaches based on approximation are inappropriate because trading accuracy for efficiency is not necessary in our context.

Recently, there are many approaches proposed in the literature such as the BIRCH algorithm [14], the SWClustering

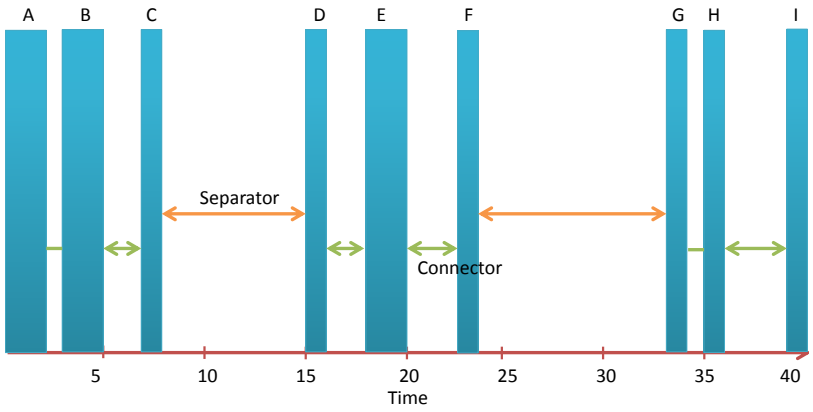

Figure 3: The gaps between 9 events A, B, C, D, E, F, G, H, I are $G = \{1, 2, 7, 2, 3, 9, 1, 3\}$. Using the 2-mean algorithm we can obtain two clusters $G_s = \{7, 9\}$ and $G_c = \{1, 2, 2, 3, 1, 3\}$ with the centroids at $\bar{g}_s = 8$ and $\bar{g}_c = 2$. G_s is labelled as separator while G_c is labelled as connector. As a result, the set of events can be divided into three groups (A, B, C), (D, E, F) and (G, H, I).

algorithm [15], the CluStream algorithm [1] or the ClusTree algorithm [11] that put more efforts on tackling data distribution changes and modelling the evolution of clusters. These approaches typically require two different phases. In the first phase, the data stream is summarized using a light-weight set of micro-clusters. Micro-clusters have the additive property so the summary of the stream can be incrementally updated or queried in the latter phase. In the second phase, macro-clustering techniques such as the k-mean algorithm can be used to cluster the micro-clusters together to produce the final clustering results. To deal with distribution change some techniques using decaying factors or keeping micro-clusters at the time horizons with preference on the most recent data were proposed. These approaches were shown to be very effective in clustering evolving data streams where change happened frequently.

However, since these algorithms were designed to deal with very high-dimensional data, they need to trade accuracy for efficiency. Moreover, in the context of the online event clustering problem, we need to determine the label of the event as long as it arrives in the data stream. How to label arrival events is not fully described in the implementation of the BIRCH, CluStream or ClusTree algorithms. Simply assigning the label based on the clustering results of the most recent window does not work well. On the other hand, our labelling method using votes from a set recent windows works well even under quick distribution changes.

In the field of transportation research, many works used SCOOT data [2] together with the other kinds data such as cameras or GPS data [9, 8] or mobile sensor data [10] to estimate the status of the road networks. In practice, not all kind of data available for different companies. For instance, mobile data is only available for mobile network provider companies while GPS data is the property of GPS device providers.

Using SCOOT data alone to estimate the car queue length is a challenging problem because the SCOOT data provides very little information except the flow and the presence of the cars. Our work provides a solution to this problem using SCOOT data alone. The difference between our method and queue length estimate methods in the literature [12] is that our solution does not require information across different sensors, every sensor can estimate queue length based on its own data. Therefore, this property enables to accomplish this task with less communication between sensors, thus it cuts of expensive energy consumption due to communication between sensors.

3. PROBLEM FORMULATION

Let $S = \{e_1, e_2, \cdots, e_n\}$ be a stream of events. Every event e_i is associated with a pair of timestamps (s_i, f_i) indicating the time when the event starts and finishes. We only consider non-overlapping events, i.e. $f_i \leq s_{i+1}$ for $i = 1, 2, \cdots, n$.

Let $g_i = s_{i+1} - f_i$ denote the gap between two consecutive events e_i and e_{i+1}. Every stream of events S corresponds to a sequence of gaps denoted by $G = \{g_1, g_2, \cdots, g_{n-1}\}$. Because the sequence of gaps uniquely determines the corresponding stream of events, from now on we use the sequence of gaps as a representation of the stream of events.

A sliding window W with length w is the window containing w most recent elements of the stream, i.e. $W = \{g_{n-w+1}, g_{n-w+2}, \cdots, g_n\}$.

We are interested in the event clustering problem that groups events happening close to each other into the same group while events in different groups must happen furthest away from each other. Under the gap representation, the event clustering problem is equivalent to classification of the set of gaps with two class labels: *group separator* that separates two consecutive groups and *group connector* that connects two consecutive events in the same group.

EXAMPLE 1. *Figure 3 shows 9 events with start and finish times as follows: $A(1, 2)$, $B(3, 5)$, $C(7, 8)$, $D(15, 16)$, $E(18, 20)$, $F(23, 24)$, $G(33, 34)$, $H(35, 36)$, $I(39, 40)$. The gaps sequence is $1, 2, 7, 2, 3, 9, 1, 3$. The events can be grouped into three groups: (A, B, C), (D, E, F) and (G, H, I). Gaps between (C, D) and (F, G) are group separators while the other gaps are connectors. Separators have larger mean value than the mean value of connectors.*

Our intuition is that separators and connectors are generated by two different distributions such that the mean value

of separators is greater than the mean value of connectors. Therefore, we can find event groups by solving a 2-mean clustering problem. Let G_s and G_c be the subsets of the set of gaps G such that $G_s \cup G_c = G$ and $G_s \cap G_c = \emptyset$. Denote $\bar{g}_s$ and $\bar{g}_c$ as the mean value of gaps in G_s and G_c respectively, i.e. $\bar{g}_s = \frac{\sum_{g \in G_s} g}{|G_s|}$ and $\bar{g}_c = \frac{\sum_{g \in G_c} g}{|G_c|}$, where $|G_s|$ and $|G_c|$ stand for cardinalities of G_s and G_c. The event clustering can be formulated as follows:

DEFINITION 1 (EVENT CLUSTERING). *Find the partition of G into G_s and G_c such that $D(G_s, G_c) = \sum_{g \in G_s} (g - \bar{g}_s)^2 + \sum_{g \in G_c} (g - \bar{g}_c)^2$ is minimized.*

Assume that $\bar{g}_s > \bar{g}_c$, the set of gaps in the cluster G_s is classified as a separator while the gaps in the clusters G_c are considered as connectors. For instance, the sequence of gaps in Figure 3 are $G = \{1, 2, 7, 2, 3, 9, 1, 3\}$. Using the k-mean algorithm with $k = 2$ se can find two clusters $G_s = \{7, 9\}$ (separators) and $G_c = \{1, 2, 2, 3, 1, 3\}$ (connectors) where the centroids of each cluster are $\bar{g}_s = 8$ and $\bar{g}_c = 2$.

Definition 1 is for static data, under the streaming context, we define the online event clustering problem as follows:

DEFINITION 2 (ONLINE EVENT CLUSTERING). *Let W be a sliding window and G be a sequence of gaps extracted from the window. At any time point find a partition of G into G_s and G_c such that it minimizes $D(G_s, G_c) = \sum_{g \in G_s} (g - \bar{g}_s)^2 + \sum_{g \in G_c} (g - \bar{g}_c)^2$.*

Unlike the static case when the label of each gap is determined based on the results of the clustering algorithm, under the streaming context, since the clustering structure may change when the sliding window is updated inconsistency of gap labelling at different moments may happen. Choosing which label to assign to a new gap is discussed in the next section.

4. ALGORITHMS

Simple solution using the standard implementation of the k-mean algorithm is time demanding because the complexity of each update is $O(w.s)$, where w is the window size and s is the number of iterations needed for the k-mean algorithm to terminate. Moreover, the results of the k-mean algorithm are sensitive to the choice of the initial centroids which might lead to unpredicted results.

In subsection 4.1, we propose an algorithm called OTEC as the acronym for Online Temporal Event Clustering. OTEC solves problem 2 exactly while it requires only $O(w + \log w)$ operations per update. Subsequently, in subsection 4.2 we introduce labelling methods to tackle the inconsistency issue of gap labels when change happens. Finally, in subsection 4.3 we discuss how to make use of domain knowledge for improving the clustering algorithm.

4.1 Online event clustering

Assume that the set of gaps $G = \{g_1, g_2, \cdots, g_n\}$ is sorted according to the ascending order of the gap value. A split of G at an index $1 \leq m < n$ is a partition of G into G_c^m and G_s^m such that $G_c^m = \{g_1, g_2, \cdots, g_m\}$ and $G_s = \{g_{m+1}, g_{m+2}, \cdots, g_n\}$. We prove the following important lemma showing that the online event clustering problem is equivalent to finding the best split in a sorted sequence of gaps:

Algorithm 1 OTEC(G, g)

```
Input: a sequence of gaps G sorted according to the ascending order and a new gap g
Output: two clusters G_s and G_c
Remove the expired element from G
Insert g to G such that it preserves the ascending order
Assume that G = {g_1, g_2, ..., g_w} is in the ascending order
x = 0
y = sum_{j=1}^{w} g_j
max = 0
for i=1 to w do
    x = x + g_i
    F = x^2/i + (y-x)^2/(w-i)
    if max < F then
        max=F
        G_c = {g_j | 0 <= j <= i}
        G_s = {g_j | i < j <= w}
    end if
end for
```

LEMMA 1. *The split (G_s^m, G_c^m) at the index m maximizing $\frac{\left(\sum_{i=1}^{m} g_i\right)^2}{m} + \frac{\left(\sum_{i=m+1}^{n} g_i\right)^2}{n-m}$ is the solution to the online event clustering problem.*

PROOF. First we rewrite the objective function of the event clustering problem as follows:

$$D(G_s, G_c) = \sum_{g \in G_s} (g - \bar{g}_s)^2 + \sum_{g \in G_c} (g - \bar{g}_c)^2 \quad (1)$$

$$= \sum_{g \in G} g^2 - \frac{\left(\sum_{g \in G_s} g\right)^2}{|G_s|} - \frac{\left(\sum_{g \in G_c} g\right)^2}{|G_c|} \quad (2)$$

Since $\sum_{g \in G} g^2$ is a constant, minimum value of $D(G_s, G_c)$ is achieved when $F(G_s, G_c) = \frac{\left(\sum_{g \in G_s} g\right)^2}{|G_s|} + \frac{\left(\sum_{g \in G_c} g\right)^2}{|G_c|}$ is maximized. Denote (G_s^*, G_c^*) as the partition of G that maximizes F.

We will prove that (G_s^*, G_c^*) corresponds to a split of G sorted according to the ascending order. We prove by contradiction, assume that (G_s^*, G_c^*) is not a split.

Let $X = \sum_{g \in G_s^*} g$ and $Y = \sum_{g \in G_c^*} g$, without loss of generality, assume that $\frac{Y}{|G_c^*|} \leq \frac{X}{|G_s^*|}$. Let $g_c = Max_{g \in G_c^*}\{g\}$ and $g_s = Min_{g \in G_s^*}\{g\}$.

Since (G_s^*, G_c^*) is not a split, we must have $g_c > g_s$. We create a new partition (G_s^+, G_c^+) of G such that G_s^+ is obtained from G_s^* by removing g_s and adding g_c and G_s^+ is obtained from G_c^* by removing g_c and adding g_s.

We will show that (G_s^+, G_c^+) will result in a larger value of the objective function F. In fact, we have:

$$\begin{aligned} F(G_s^+, G_c^+) - F(G_s^*, G_c^*) &= \frac{(X - g_s + g_c)^2}{|G_s^*|} + \frac{(Y - g_c + g_s)^2}{|G_c^*|} \\ &\quad - \frac{X^2}{|G_s^*|} - \frac{Y^2}{|G_c^*|} \\ &= (g_c - g_s) * \\ &\quad \left(\frac{2X - g_s + g_c}{|G_s^*|} - \frac{2Y - g_c + g_s}{|G_c^*|} \right) \\ &> 0 \end{aligned}$$

Therefore, $F(G_s^+, G_c^+) > F(G_s^*, G_c^*)$ which leads to contradiction. □

A direct consequence of lemma 1 is that problem 2 can be solved exactly by keeping the set of gaps in the ascending order and then finding the split that maximizes the value of the function F.

Algorithm 1 describes the main steps of the OTEC algorithm. It gets at the input a set of gaps G sorted in the ascending order and a new gap value g. First, it removes the expired element from G and inserts g to G (lines 3-4). If a balance binary tree is used as a data structure to store G the cost of this step is equal to $O(\log w)$, where w is the size of the sliding window. Subsequently, OTEC does a linear scan through G and incrementally calculates the value of the objective function F at each split (lines 9-17). The split that results in the maximum value of F is tracked to update the clusters G_s and G_c (lines 12-15). The complexity of this step is $O(w)$. In summary, the complexity of OTEC is $O(w + \log w)$ per update.

4.2 Gap labelling strategy

As long as we get the clusters structure from the window, gaps labelling is the next step. If the data does not change over time, labelling is simply done by assigning gap with the label, either as separator or connector depending on the cluster it belongs to. Nevertheless, under the streaming context, the labelling task is more complicated as the label of a given gap may change unpredictably from one to another moment. This section discusses different approaches for determining the right label of gaps in real-time.

4.2.1 A simple labelling strategy

The simplest labelling algorithm is to assign the new gap g with the label it gets from the from the window where it is the latest element. The following example shows how this approach work in every step of streaming updates.

EXAMPLE 2 (SIMPLE LABELLING). *Figure 4 shows an example with a sequence of gaps and a sliding window with size $w = 8$. The new gaps are assigned a label immediately as long as they are appended to the data stream:*

- *At time point $t = 19$, $G = \{1, 1, 1, 8, 1, 1, 1, 4\}$ is the gap sequence, the results of the OTEC algorithm are $G_c = \{1, 1, 1, 1, 1, 1, 4\}$ and $G_s = \{8\}$ and the labelled sequence is $CCCSCCCC$ where C and S are the acronyms for Connector and Separator. Therefore the final label assigned to the new gap $g = 4$ is C.*
- *At time point $t = 22$, $G = \{1, 1, 1, 4, 1, 1, 1, 1\}$ is the gap sequence, the clusters are $G_c = \{1, 1, 1, 1, 1, 1, 1\}$ and $G_s = \{4\}$ and the label sequence is $CCCSCCCC$. Therefore the label assigned to the new gap $g = 1$ is C.*

The simple labelling algorithm only works well when the distribution of the gaps does not change over time. When change happens it may assign a wrong label to the gap value. For instance, in Figure 4, we assume that at time point $t = 12$ the mean value of separators reduces from 8 to 4. We can see that the label for the gap $g = 4$ is determined at $t = 19$ as a connector , while it should have been labelled as a separator. If the label of $g = 4$ is assigned at time point $t = 22$, its label is correct. This situation incurs when change in the distribution of gaps happens and the sliding window is in the middle of the transition state. The next subsection discusses a solution for this issue.

4.2.2 Labelling by voting

Instead of getting the label by considering clustering results only from the current sliding window, the labelling by voting algorithm considers votes accumulated from a set of recent windows. In particular, let w be the window size. All the votes from all the windows containing $\frac{w}{2^k}$ most recent events for any $0 \leq k < \log_2 w$ are counted. The final label of the new gap is determined by the label got the most votes from the set of windows. In doing so, the labelling algorithm put more preference on the most recent windows which help it quickly adapt to change. Moreover, the amortized complexity of the voting algorithm remains cheap $O(w + \log w)$.

EXAMPLE 3 (LABELLING BY VOTING). *In Figure 4 at time point $t = 19$, the label of the new gap $g = 4$ is determined by counting the votes from three windows A, B and C with length $8, 4$ and 2 respectively. It is labelled as a separator twice, in the windows $C(1, 4)$ and $B(1, 1, 1, 4)$ and once as a connector in the window $A(1, 1, 1, 8, 1, 1, 1, 4)$. Therefore, the label separator gets most votes which is the final label for $g = 4$.*

4.3 Make use of domain knowledge

In the context of the event clustering problem domain knowledge can be used to improve the clustering results further. Domain knowledge can be given in many forms. In this work, we consider one of the forms of domain knowledge which is given as a constraint on the gap value. For example, in the SCOOT data, we know that if a gap value is greater than half of one minute long, it is definitely a separator. This type of knowledge can be easily integrated into the clustering algorithm by simply filtering out gaps matching the constraint. Using this type of knowledge can improve the clustering results in the case that the window size is initialized with a value which is not large enough to have both separators and connectors in a window. In the SCOOT data, this case happens at night when all the gaps in the window are separators.

5. EXPERIMENTS

5.1 Experiment set-up

We implemented our algorithm in Java and run the programs in a Linux 64 bits machine with four Intel Xeon 2.59 GHz cores and 16 GB of RAM. The datasets used for experiments are described as follows:

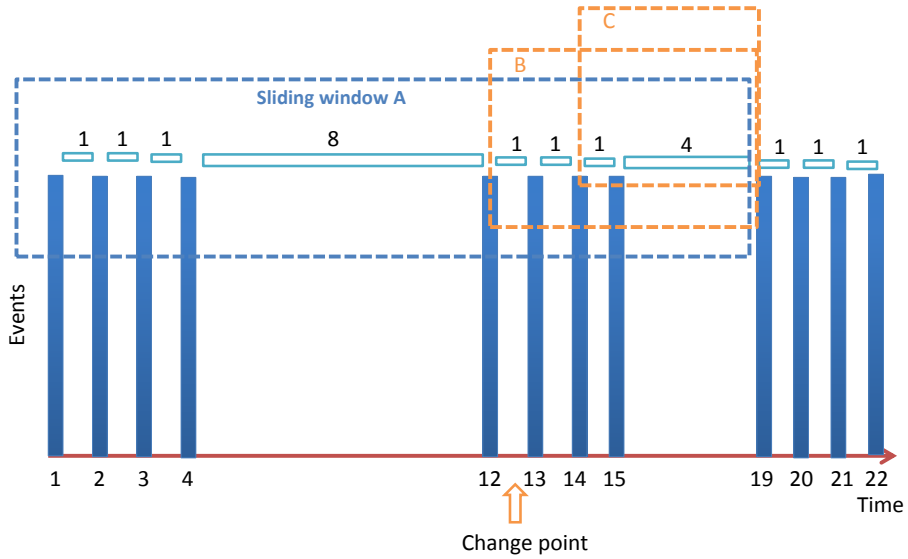

Figure 4: Change point happens at $t = 12$ when the mean value of separators is reduced from 8 to 4. The gap $g = 4$ at time point $t = 19$ is labelled as a connector in the sliding window A, and as a separator in the windows B and C. Counting votes from the windows A,B and C the gap $g = 4$ is considered as a separator.

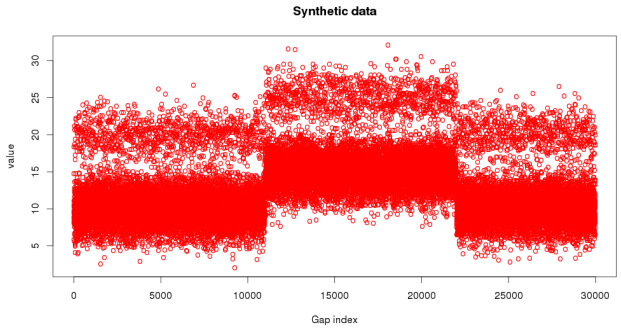

Figure 5: Synthetic data with 30000 gap values. In the first 10000 events and the last 10000 events, connectors are drawn from a normal distribution $N(\mu = 10, \sigma = 2)$ and separators are drawn from $N(\mu = 20, \sigma = 2)$. The connectors between 10000 and 20000 events are drawn from $N(\mu = 15, \sigma = 2)$, while the separators are drawn from $N(\mu = 25, \sigma = 2)$.

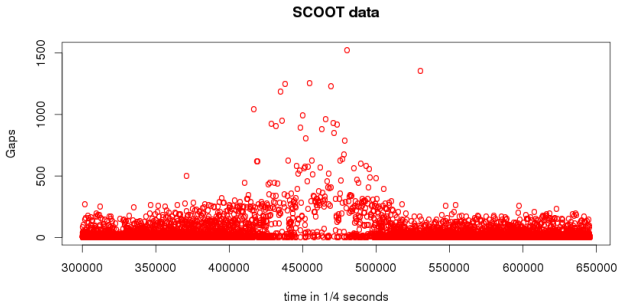

Figure 6: Gap distribution in the one day ground-truth set extracted from the SCOOT data. Distribution of gaps changes very quickly during different periods of the day and abruptly changes at night.

- **Synthetic data**: consists of 30000 events. Events are organized in groups consisting of 10 events. In the first 10000 events and the last 10000 events, connectors are drawn from a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 2$ and separators are drawn from a normal distribution with mean $\mu = 20$ and standard deviation $\sigma = 2$. The events from the number 20000 to 30000 are generated such that the connectors are drawn from a normal distribution with mean $\mu = 15$ and standard deviation $\sigma = 2$ and separators are drawn from a normal distribution with mean $\mu = 25$ and standard deviation $\sigma = 2$. Figure shows a plot of the gaps in the synthetic data. We can see too change points happening at the events numbered 10000 and 20000. This dataset is created to see how the algorithms work under mean value changes.

- **SCOOT data**: contains about three weeks of the SCOOT data from the city of London. The dataset consists of 35 billions reads from 5049 SCOOT sensors. We use the whole dataset to measure the scalability of the proposed algorithm and pick one sensor from the set of sensors for validating clustering accuracy. In order to get the ground-truths, we label one day of data from that sensor. Totally 12305 gaps are labelled manually. The manual labelling task was performed by first visualizing every window of length 250 seconds and manually picking the separators based on our observation. The gaps which were not picked in the labelling process were considered as connectors. The ground-truth set contains 2076 separators and 10229

connectors. Figure 6 shows the distribution of the gaps in the ground-truth set with an abrupt change happening at night.

We compare our approach (OTEC with labelling by voting) to the following baseline algorithms:

- **CluStream [1]:** is considered as a benchmark algorithm for evolving data stream clustering. We used the MOA implementation of CluStream [5] which is currently available in the R stream package[2]. There are three parameters including the horizon window size (h), the number of micro-clusters (k) and the kernel radius factor (t) in the implementation of $CluStream$. We vary the window size to see how the algorithm perform with different horizons while keeping the other parameters unchanged with their default values ($k = 100$ and $t = 2$). CluStream is used to summarized the stream with a set of micro-clusters. For each data stream update, an offline macro-clustering algorithm based on the standard 2-mean algorithm is used to re-cluster the micro-clusters. The label of the recently arrival gap is assigned based on results of the macro-clustering algorithm. If the gap is assigned to the cluster with larger mean value, its label is determined as a separator, otherwise, its label is chosen as a connector.

- **ClusTree [11]:** is proposed recently and considered as the state of the art algorithm for evolving data stream clustering. The algorithm is very efficient and was shown to be very effective in handling change. We used the implementation of ClusTree available in the R stream package. There are three parameters including the horizon window size (h), the maximum height of the tree (k) and the decaying factor (t) in the implementation of ClusTree. We vary the window size to see how algorithm performs with different horizons while keeping the other parameters unchanged with their default values ($k = 8$ and t is automatically defined by the algorithm). ClusTree is used to summarized the stream with a set of micro-clusters. For each data stream update, a macro-clustering algorithm based on the standard 2-mean algorithm is used to re-cluster the micro-clusters. The label of the recently arrival gap is assigned based on results of the macro-clustering algorithm. If the gap is assigned to the cluster with larger mean value, its label is determined as a separator, otherwise, its label is chosen as a connector.

- **Mean value change detection (MVCD):** the algorithm keeps track of the mean value in the sliding window. It compares the gap to the mean value, and calculates the empirical p-value of observing the gap in the positive direction. If the p-value is less than a given threshold α (a pre-set parameter) the gap is considered as a separator, otherwise it is considered as a connector.

5.2 Accuracy

We performed an experiment with the synthetic and the SCOOT dataset to evaluate the clustering accuracy of different algorithms. Clustering accuracy was calculated as the fraction between the number of gaps that were labelled correctly and the total number of gaps in the ground-truth sets.For the synthetic dataset, no domain knowledge is given, while for the SCOOT dataset, gaps having value greater than 30 seconds long are considered as separators. These gaps are filtered out during the clustering process.

The baseline algorithms we considered in the comparison are the CluStream algorithm, the ClusTree algorithm and the mean value change detection algorithm (MVCD). The significance parameter α in the MVCD algorithm was set to 0.1, 0.01 and 0.001 to see the behaviour of the MVCD algorithm with different parameter settings.

In Figure 7, we plot the accuracy of the algorithms in the synthetic and the SCOOT dataset when the window length is varied. The first impression is that the performance of the MVCD algorithm is very sensitive to the value of the parameter α. Moreover, the accuracy of the MVCD algorithm is very low. It is even less accurate than the guess that always assign all the gaps with the connector label which results in the accuracy of 0.9 and 0.83 in the synthetic and SCOOT datasets. This empirical result shows that the methods relying on parameter pre-settings do not work well because of the sensitivity of the accuracy when change happens.

In contrast to the MVCD algorithm, the OTEC, the CluStream and the ClusTree algorithm worked quite well. However, the OTEC algorithm outperformed the CluStream and the ClusTree algorithms. On the other hand, the accuracy of the OTEC algorithm results remain stable while the performance of the CluStream and the ClusTree algorithms vibrated when the window length is varied . A more careful analysis on the clustering results showed that CluStream and ClusTree started dropping the accuracy when the length of the gaps changes abruptly at night as shown in Figure 6. In contrast to these algorithms, the OTEC algorithm was able to quickly adapt to the change by getting votes from the recent windows.

5.3 Scalability

We run our algorithm on the whole SCOOT data to show the scalability of our algorithm when the window size is varied. The average time per one sensor update is reported in Figure 8. We can see that the OTEC algorithm scaled linearly with the length of the sliding windows. Moreover, it is very efficient. For example, when the window size is set to 10000, the OTEC algorithm can handle at least 20000 updates per second. While one sensor in the SCOOT system generates 4 updates per second, by using a commodity computer, our algorithm can work well with a SCOOT system at the city-scale containing at least 5000 sensors.

5.4 A use-case

In this subsection we discuss an application of our algorithm for estimating the car queue length and anomaly detection. We pick two sensors located at the upstream and downstream locations of a street as shown in Figure 9. The direction of car movement is from sensor A to B. Between A and B there is junction with a traffic light so not all the cars passing through A will move to B and not all the cars passing B move from A. However, since the junction between A and B is not crowded with one-way flow toward A, most of the flow from A will go to B.

5.4.1 Car queue length estimates

Estimating the length of a queue of cars waiting at a traffic

[2]http://cran.r-project.org/web/packages/stream/index.html

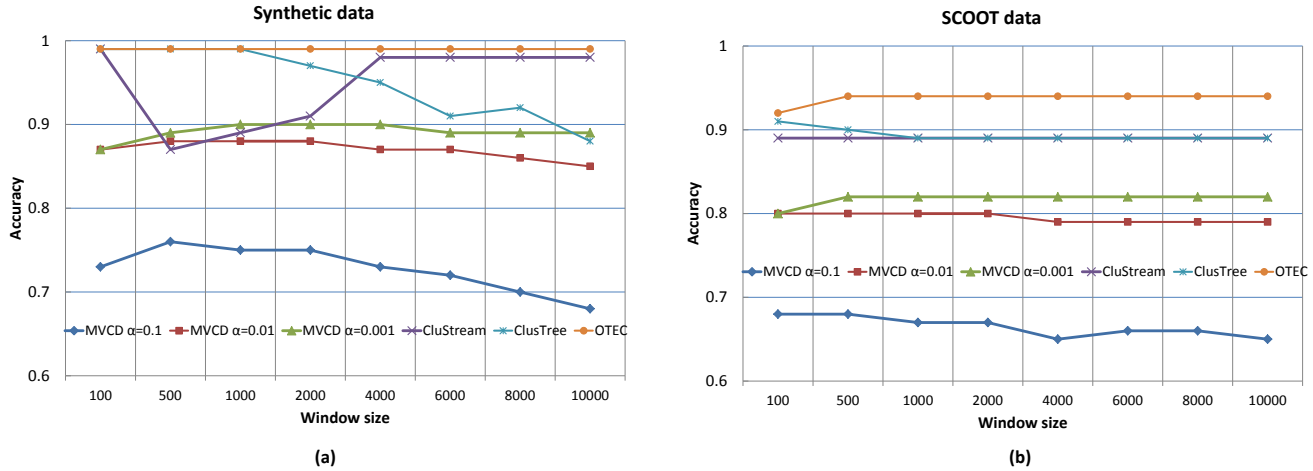

Figure 7: Accuracy of the algorithms in the synthetic and SCOOT dataset. Our algorithm (OTEC with votes) is significantly better than the baseline algorithms. The performance of the algorithm OTEC with votes is very stable even when the window length is varied.

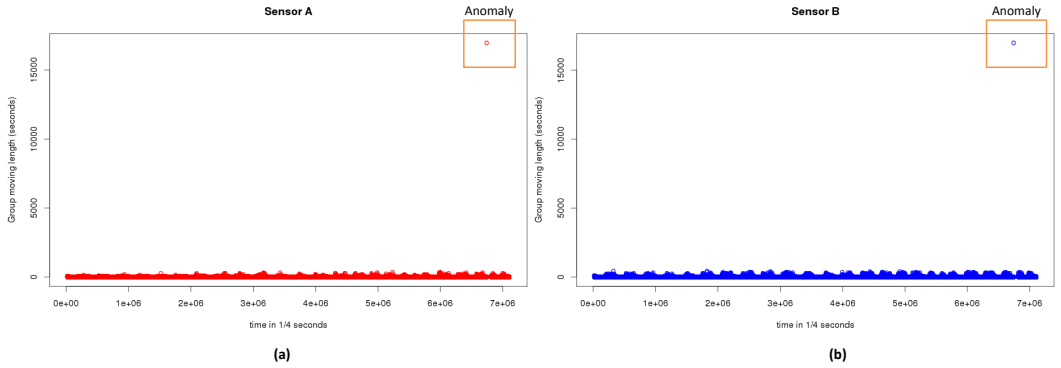

Figure 11: Group moving time derived from two sensors A and B. An anomaly was detected simultaneously in both sensors when a small moving group of cars need more than 5 hours to pass through each sensor.

light is an important problem for monitoring the road network status. In this subsection, we discuss how to estimate this measure using the clustering results. Our assumption is that groups of cars usually move and stop at the traffic light together with a small variation in the number of cars in a group. With that assumption in mind we can use the number of cars in a group as an estimate for the car queue length.

This estimate is not a hundred percent accurate but it reasonably reflects the status of the road network. For instance, in Figure 10, we plot the number of cars in clusters and its variation during different period of the day. The plot shows data of one week with mean value equal to 7 cars per queue. We can see an interesting pattern that repeats everyday. During the week day (from Monday to Friday), there are always two peak times at the beginning and the end of the day. During weekend, the peak time happens only once in the morning. During peak times the number of cars in a queue can reach up to 150 cars and it lasts for 10 minutes long for the whole queue to pass through the sensor completely. This may happen when there are traffic jams at the location.

5.4.2 *Anomalies detection*

Besides the car queue length, from each cluster we can easily derive information about the total time a group of cars needs to pass through the sensor completely. We call this measure as group moving time. By monitoring group moving time we can detect if a group of cars get too much delay when they are passing through a location where a sensor is located. In Figure 11 we plot group moving time derived from two sensors at the location A and B as shown in figure 9. We can see that there is an abnormal point (marked with big rectangles) detected in both sensors simultaneously. Further investigation shows that there is a small moving group with only 3 and 4 cars detected by each sensor at the same time. The total moving time is about 5 hours. This result suggests that either the sensors were malfunctioning at that moment or a serious congestion was happened during that period along the corridor connecting two sensors.

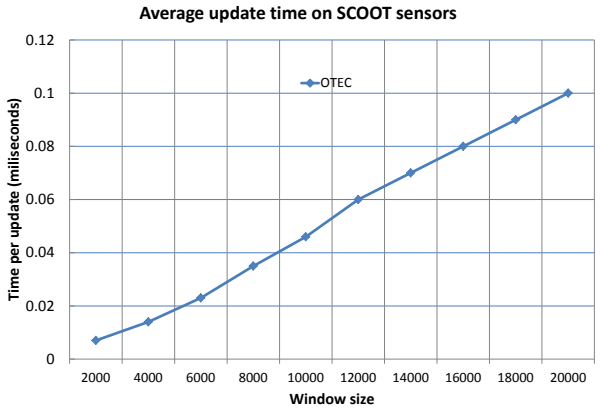

Figure 8: Average time per update in SCOOT data. Our algorithms are highly efficient and scaled linearly with the window size. When the window size is set to 10000, the algorithms can handle at least 20000 updates per second. One sensor in the SCOOT system generates 4 updates per second. Therefore, with a commodity computer our algorithm works well for a SCOOT system containing at least 5000 sensors.

6. CONCLUSIONS AND FUTURE WORK

In this work, we formulated the online event clustering problem in an event stream. This problem can be solved by using a k-mean algorithm on one dimensional space, where $k = 2$. However, straightforward adoption of the k-mean algorithm produces inefficient and inaccurate results due to data distribution change. We showed that the problem can be solved exactly using a linear algorithm in the length of the sliding window.

Besides, to tackle data distribution change, we proposed a ensemble based approach that labels events using votes from a set of recent windows. Our approach was shown to be more accurate than the state-of-the-art algorithms in terms of clustering accuracy in an experiment with one synthetic

Figure 9: Locations of the upstream sensor A and the downstream sensor B. Cars move from A to B. Between A and B there is a junction. Most of the flow from A moves further to B.

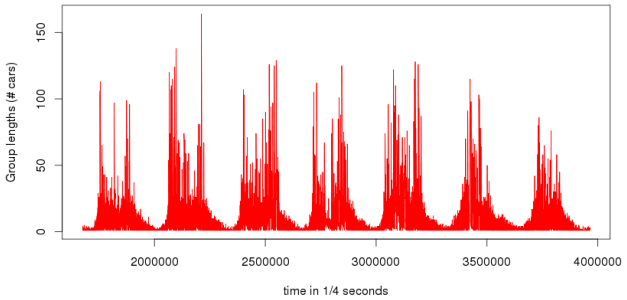

Figure 10: One week data shows the queue length during different periods of the day. There are two peak hours every week day (from Monday till Friday) and only one peak time during weekend. In average the queue length is equal to 7 cars per queue. During peak time the number of cars can reach up to 150 cars per queue. This may happen when there is traffic jam along the corridor.

and a real-life dataset.

Moreover, our approach scaled up to the size of a large dataset extracted from the SCOOT system in the city of London. We motivate the problem by showing an application of the algorithm in estimating the car queue length and detecting anomalies on the roads. As a future work, we plan to use the algorithm to estimate different measures of the road networks such as the journey time between two sensors located at the downstream and the upstream locations when a group of cars is tracked across the sensor data. These measures can be further used as predictors in a flow prediction model and or can be used for monitoring the status of the road networks.

7. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 81–92. VLDB Endowment, 2003.

[2] J. Bacon, A. I. Bejan, A. R. Beresford, D. Evans, R. J. Gibbens, and K. Moody. Using real-time road traffic data to evaluate congestion. In C. B. Jones and J. L. Lloyd, editors, *Dependable and Historic Computing*, volume 6875 of *Lecture Notes in Computer Science*, pages 93–117. Springer, 2011.

[3] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[4] J. Beringer and E. Hüllermeier. Online clustering of parallel data streams. *Data Knowl. Eng.*, 58(2):180–204, 2006.

[5] A. Bifet, G. Holmes, B. Pfahringer, J. Read, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. Moa: A real-time analytics open source framework. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *ECML/PKDD (3)*, volume 6913 of *Lecture Notes in Computer Science*, pages 617–620. Springer, 2011.

[6] J. de Andrade Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho, and J. Gama. Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1):13, 2013.

[7] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, Mar. 2003.

[8] R. Herring, P. Abbeel, A. Hofleitner, and A. Bayen. Estimating arterial traffic conditions using sparse probe data, 2010.

[9] R. Herring, A. Hofleitner, P. Abbeel, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 929–936, Sept 2010.

[10] X. Jeff Ban, P. Hao, and Z. Sun. Real time queue length estimation for signalized intersections using travel times from mobile sensors. *Transportation Research Part C: Emerging Technologies*, Feb. 2011.

[11] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, 29(2):249–272, 2011.

[12] H. X. Liu, X. Wu, W. Ma, and H. Hu. Real-time queue length estimation for congested signalized intersections. *Transportation Research Part C: Emerging Technologies*, 17(4):412–427, Aug. 2009.

[13] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of IEEE International Conference on Data Engineering*, page 685, 2001.

[14] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.

[15] A. Zhou, F. Cao, W. Qian, and C. Jin. Tracking clusters in evolving data streams over sliding windows. *Knowl. Inf. Syst.*, 15(2):181–214, May 2008.