Adaptive Grid-Based k-median Clustering of Streaming Data with Accuracy Guarantee

Jianneng Cao ⋆, Yongluan Zhou ‡, and Min Wu ⋆

⋆ Institute for Infocomm Research at Singapore, ‡ University of Southern Denmark
⋆ {caojn,wumin}@i2r.a-star.edu.sg, ‡ zhou@imada.sdu.dk

Abstract. Data stream clustering has wide applications, such as online financial transactions, telephone records, and network monitoring. Grid-based clustering partitions stream data into cells, derives statistical information of the cells, and then applies clustering on these much smaller statistical information without referring to the input data. Therefore, grid-based clustering is efficient and very suitable for high-throughput data streams, which are continuous, time-varying, and possibly unpredictable. Various grid-based clustering schemes have been proposed. However, to the best of our knowledge, none of them provides an accuracy guarantee for their clustering output. To fill this gap, in this paper we study grid-based k-median clustering. We first develop an accuracy guarantee on the cost difference between grid-based solution and the optimum. Based on the theoretical analysis, we then propose a general and adaptive solution, which partitions stream data into cells of dynamically determined granularity and runs k-median clustering on the statistical information of cells with an accuracy guarantee. An extensive experiment over three real datasets clearly shows that our algorithm provides high-quality clustering outputs in an efficient way.

1 Introduction

A data stream is a massive sequence of data objects, where each object can be described by a d-dimensional attribute vector. Nowadays, data streams are common in many applications, which include online financial transactions, telephone records, network monitoring, web applications, and sensor network. Analyzing data streams brings unique opportunities to optimize the operation of companies and organizations in a more responsive fashion. Typically, data streams are continuous, time-varying, unpredictable and possibly with a high throughput. Such unique characteristics have attracted great attention of the research community, and numerous techniques have been proposed to process data streams. Examples include query processing [27], sampling [32], clustering [12, 35], and classification [23].

In this work we study continuous k-median clustering on data streams. Informally, clustering is to divide data into groups, such that objects in a same group are more similar to each other than those from other groups. k-median clustering is known to be NP-hard, and it is thus time-consuming to compute, especially

for a large dataset. As such, grid-based clustering [17] was proposed. It partitions data space into (usually) uniform cells, and computes statistical information for each cell. Cell size, max/min/mean value, and standard deviation are the typical statistical information for a cell. Clustering is then run on the statistical information without a need to access the raw input data. Since statistical information is much smaller than the raw input dataset, grid-based clustering is much more efficient than clustering on the raw input data. In addition, grid-based clustering processes data streams on the fly. That is, it can process streaming tuples one by one without a need of considering their dependencies. Thus, it has been regarded as a good candidate solution for clustering continuous streaming data. Various grid-based solutions have been proposed thus far [17, 31, 21]. However, as far as we know none of them has given an accuracy bound, which clearly specifies the difference between their clustering output and the optimum.

Coreset-based clustering has also been proposed to continuously cluster data streams [4, 24, 20]. Given a clustering algorithm and a dataset, a coreset is a summary of the dataset, such that the output of the clustering algorithm running on the coreset can approximate that of the same algorithm running on the whole dataset. Since coreset is smaller than the input data, clustering algorithms based on coresets also have high efficiency. However, coresets cannot be computed on the fly as grid cells. To compute the coreset of a set of tuples, the whole set of data needs to be available. Batch-based approaches like [5, 34] split data streams into segments. Still, they require that a whole segment of streaming tuples is available before the coreset corresponding to the segment can be computed. Therefore, Coreset-based clustering has an output latency and a high storage consumption, especially when the size and time span of the segment is big.

In this paper, we propose a general and adaptive grid-based k-median clustering solution with an accuracy guarantee. Our solution adopts the sliding-window model, where only streaming tuples in a recent window contribute to the clustering output. Such a model is suitable for many real-world applications that care for more recent data instead of the whole unbounded data stream. Our solution leverages approximation algorithms [10, 16] to cluster data. An approximation k-median algorithm achieves a clustering cost at most α times of the optimum, where $\alpha > 1$ is a parameter given by the algorithm. Our solution is general, and can be built on any approximate algorithm. It adaptively partitions tuples in sliding windows into cells with a dynamically determined granularity, and applies approximate k-median algorithm on the statistical information of cells to output clustering results. In summary, the contributions of our work are as follows.

- We present a theoretical analysis of approximate clustering algorithm running on statistical information of cells, and provide accuracy guarantee for their clustering output.
- Based on the theoretical analysis, we provide a general and adaptive solution, which dynamically determines the granularity of cells, and runs k-median clustering on the cells with an accuracy guarantee.
- Our extensive experimental results over real datasets show that our solution has clustering cost close to that of approximate algorithms running on the

raw input data, but achieves an an efficiency improvement of more than 2 orders of magnitude.

We organize the remaining of our work as follows. We present the background knowledge in the next section and then perform a theoretical study of grid-based clustering in Section 3, based on which we propose our solution in Section 4. We carry out experimental evaluation in Section 5 and survey related work in Section 6. We conclude our work in Section 7.

2 Preliminaries

2.1 Data Stream

We model data stream $\mathcal{DS}$ as an infinite append-only array of tuples, and denote its tuple at timestamp t by $\mathcal{DS}$ $[t]$, where t is from 1 to ∞. We assume a data stream has d dimensions.

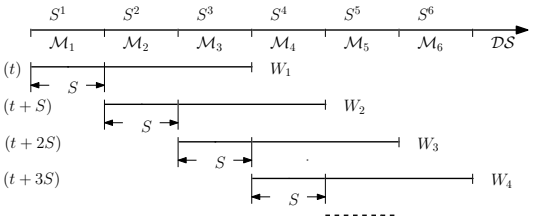


Fig. 1: An illustration of sliding windows

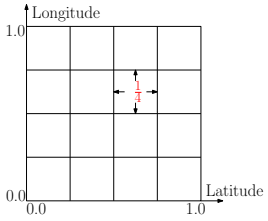


Fig. 2: A 4×4 uniform grid

Data stream has time-varying data distribution. To capture its evolving characteristics, recent data need to be assigned higher weight than the outdated when carrying out clustering analysis. In this paper we adopt sliding window model, which considers only the tuples in a recent window (i.e., from the current timestamp up to a certain timestamp of the past). A sliding window is characterized by two parameters: WS window size and S step size. The window slides forward at every time interval of S. As the sliding continues, a sequence of windows is generated. Figure 1 gives an example. The first window W_1 at timestamp t contains tuples in the time span $[t - (WS - 1), t]$. At the timestamp $t + S$, window W_1 slides forwards to window W_2, which contains tuples in the time span $[(t + S) - (WS - 1), t + S]$. At the timestamp $t + 2S$, window W_2 slides forwards to W_3, and so on. For information of other window-based models (e.g., damped window and landmark window), please refer to [19].

Sliding window moves forwards every time interval of S. For clear presentation of our solution, we mark the steps. In particular, we partition the data stream into segments, each having the length of S. Then, we take the first segment as the first step and label it by S^1, the second segment as the second step and label it by S^2, and so on. Figure 1 illustrates this.

2.2 k-medians Clustering and its Approximation

Given a set of tuples $X = \{x_1, x_2, \ldots, x_N\}$ in a d-dimensional space, k-medians clustering is to find k centers $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ in the same data space, such that the cost (i.e., the average distance between a tuple and its nearest center)

$$\mathsf{Cost}\,(X, \mathcal{O}) = \frac{1}{N} \sum_{\ell=1}^{N} \mathsf{Dist}\,(x_i, \mathsf{NN}\,(x_i, \mathcal{O})) \tag{1}$$

is minimized, where Dist is a distance function defined in a metric space satisfying triangular inequality, and $\mathsf{NN}\,(x_i, \mathcal{O})$ returns tuple x_i's nearest neighbor in $\mathcal{O}$ based on Dist. In this work we consider Euclidean distance.

k-medians clustering is NP-hard. The research community has proposed various approximation algorithms [29, 28, 15, 10]. An approximation algorithm is characterized by two parameters α and β, where $\alpha > 1$ and $\beta \geq 1$. Specifically, let $\mathcal{O}^\circ$ be a set of k cluster centers output by an optimal k-median clustering algorithm on dataset X, and $\mathcal{O}$ be a set of cluster centers output by an (α, β)-approximation clustering algorithm running on the same dataset. Then,

$$\frac{\mathsf{Cost}\,(X, \mathcal{O})}{\mathsf{Cost}\,(X, \mathcal{O}^\circ)} \leq \alpha \quad (2) \qquad \text{and} \qquad \frac{|\mathcal{O}|}{k} \leq \beta \quad (3)$$

The α value in Equality 2 shows the difference of clustering cost between the approximation algorithm and the optimal one. The smaller the α value is, the closer the approximation is to the optimum. The β value in Equality 3 bounds the maximum number of cluster centers needed in the approximation. Some approximation algorithms [29, 28] require $\beta > 1$, while others [15, 10] set $\beta = 1$. For an (α, β)-approximation algorithm with $\beta = 1$, we denote it by α-approximation.

We will study continuous k-medians clustering on sliding windows to learn the evolution of clusters under observation. We will dynamically partition streaming tuples into cells (i.e., d-dimensional rectangles), and run (α, β)-approximation clustering algorithms on the statistical information of cells. Let $\overline{\mathcal{O}}$ and $\mathcal{O}^\circ$ be the sets of cluster centers output by our solution and the optimal one, respectively. Our theoretical analysis in the next section proves that

$$\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) \leq \alpha \mathsf{Cost}\,(X, \mathcal{O}^\circ) + \delta. \tag{4}$$

where $\delta > 0$ is a function, showing the difference between (α, β)-approximation algorithm running on cells from its running on input data. The next section will give the details of theoretical analysis and specify δ.

3 Theoretical Analysis

This section investigates the accuracy guarantee of (α, β)-approximation clustering algorithms running on statistical information of cells.

3.1 Accuracy Analysis of Grid-based k-median Clustering

Let X be the input dataset in a d-dimensional data space. We uniformly partition each dimension into ω segments of equal length. Let $\overline{X}$ be the resultant cells from the partitioning. For each cell, we compute a representative, which is flexible – it can be a tuple in the cell, the cell mean, or a certain function. Given a cell c, we denote its representative by $\mathsf{R}(c)$. All the tuples $x \in c$ share the same representative, so we also use $\mathsf{R}(x)$ to denote the representative. For brevity, when the context is clear, we will also use $\overline{X}$ to denote the statistical information of the cells. Let $\overline{\mathcal{O}}$ be a set of cluster centers output by an (α, β)-approximation clustering algorithm running on $\overline{X}$. We define its clustering cost (i.e., the average distance between cell representatives and their nearest cluster centers in $\overline{\mathcal{O}}$) as

$$\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right) = \frac{1}{|X|} \sum_{c \in \overline{X}} |c| \cdot \mathsf{Dist}\left(\mathsf{R}(c), \mathsf{NN}\left(\mathsf{R}(c), \overline{\mathcal{O}}\right)\right), \tag{5}$$

where function $\mathsf{NN}\left(\mathsf{R}(c), \overline{\mathcal{O}}\right)$ returns the nearest neighbor of $\mathsf{R}(c)$ in $\overline{\mathcal{O}}$. The cost is a weighted average with the cell size as the weight.

Lemma 1 *Let X and $\overline{X}$ be an input dataset and its cells, respectively. Let $x \in X$ be a tuple and $r = \mathsf{R}(x)$ be its representative in $\overline{X}$. Suppose that $\mathcal{O}$ is a set of cluster centers output by any k-median clustering scheme running on X. Then,* $|\mathsf{Dist}(r, \mathsf{NN}(r, \mathcal{O})) - \mathsf{Dist}(x, \mathsf{NN}(x, \mathcal{O}))| \leq \mathsf{Dist}(r, x)$.

Proof. Let $O' = \mathsf{NN}(r, \mathcal{O})$ and $O = \mathsf{NN}(x, \mathcal{O})$ be the cluster centers in $\mathcal{O}$ nearest to r and x, respectively. Then,

$$\mathsf{Dist}(x, O) \leq \mathsf{Dist}(x, O') \leq \mathsf{Dist}(x, r) + \mathsf{Dist}(r, O'). \tag{6}$$

$$\mathsf{Dist}(r, O') \leq \mathsf{Dist}(r, O) \leq \mathsf{Dist}(r, x) + \mathsf{Dist}(x, O). \tag{7}$$

Combing the two inequalities, we reach the conclusion of the lemma.

On the basis of Lemma 1, we develop the relationship between optimal clustering cost and that of approximation algorithm running on cells.

Theorem 1 *Let X and $\overline{X}$ be an input dataset and its cells, respectively. Suppose that $\overline{\mathcal{O}}$ is the cluster centers output by an (α, β)-approximation k-median algorithm on $\overline{X}$, and $\mathcal{O}^\circ$ is the cluster centers output by an optimal solution on X. Then*

$$\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right) \leq \alpha \cdot \left(\mathsf{Cost}(X, \mathcal{O}^\circ) + \frac{D}{|X|}\right) \tag{8}$$

where $D = \sum_{x \in X} \mathsf{Dist}(x, \mathsf{R}(x))$.

Proof. Let $X = \{x_1, x_2, \ldots, x_N\}$, and $r_i = \mathsf{R}(x_i)$ be the representative of x_i. Let $\overline{\mathcal{O}^\circ}$ be the cluster centers output by an optimal k-median algorithm on $\overline{X}$. Then,

$$\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}^\circ}\right) \leq \frac{1}{N} \sum_{i=1}^{N} \mathsf{Dist}(r_i, \mathsf{NN}(r_i, \mathcal{O}^\circ)). \tag{9}$$

Since $\overline{\mathcal{O}}$ is the output of an (α, β)-approximation k-medians algorithm on $\overline{X}$, it follows that

$$\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}^{\circ}}\right) \geq \frac{1}{\alpha} \cdot \mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right). \tag{10}$$

Combining Inequalities 9 and 10 gives

$$\begin{aligned}
\mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right) =& \frac{1}{N} \sum_{i=1}^{N} \mathsf{Dist}\left(x_i, \mathsf{NN}\left(x_i, \mathcal{O}^{\circ}\right)\right) \\
\geq& \frac{1}{N} \sum_{i=1}^{N} \left(\mathsf{Dist}\left(r_i, \mathsf{NN}\left(r_i, \mathcal{O}^{\circ}\right)\right) - \mathsf{Dist}\left(r_i, x_i\right)\right) \\
\geq& \frac{1}{\alpha} \cdot \mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right) - \frac{D}{N},
\end{aligned}$$

where the first inequality holds by Lemma 1. This concludes the proof.

In Theorem 1, the clustering cost of approximation algorithm is $\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right)$, which is relative to the cells. The next theorem will investigate the cost relative to the input dataset.

Theorem 2 *Let X and $\overline{X}$ be an input dataset and its cells, respectively. Suppose that $\overline{\mathcal{O}}$ is the cluster centers output by an (α, β)-approximation k-median algorithm on $\overline{X}$, and $\mathcal{O}^{\circ}$ is the cluster centers output by an optimal solution on X. Then*

$$\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) \leq \alpha \cdot \mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right) + (1+\alpha)\frac{D}{|X|}, \tag{11}$$

where $D = \sum_{x \in X} \mathsf{Dist}\left(x, \mathsf{R}\left(x\right)\right)$.

Proof. Let $X = \{x_1, x_2, \ldots, x_N\}$, and $r_i = \mathsf{R}\left(x_i\right)$ be the representative of x_i. Then,

$$\begin{aligned}
\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) =& \frac{1}{N} \sum_{i=1}^{N} \mathsf{Dist}\left(x_i, \mathsf{NN}\left(x_i, \overline{\mathcal{O}}\right)\right) \\
\leq& \frac{1}{N} \sum_{i=1}^{N} \mathsf{Dist}\left(x_i, \mathsf{NN}\left(r_i, \overline{\mathcal{O}}\right)\right) \\
\leq& \frac{1}{N} \sum_{i=1}^{N} \left[\mathsf{Dist}\left(x_i, r_i\right) + \mathsf{Dist}\left(r_i, \mathsf{NN}\left(r_i, \overline{\mathcal{O}}\right)\right)\right] \\
=& \frac{D}{N} + \frac{1}{N} \sum_{i=1}^{N} \mathsf{Dist}\left(r_i, \mathsf{NN}\left(r_i, \overline{\mathcal{O}}\right)\right) \\
=& \frac{D}{N} + \mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right).
\end{aligned} \tag{12}$$

Plugging Inequality 12 into Inequality 8, we reach Inequality 11.

Theorem 2 thus gives the accuracy guarantee. It has also computed the δ function in Inequality 4.

$$\delta = (1+\alpha) \cdot \frac{D}{|X|}. \tag{13}$$

3.2 Determining Grid Granularity

Our work adopts uniform grid. We normalize each dimension to [0.0, 1.0], so as to treat each dimension equally for its contribution to the clustering cost. Every cell in the grid is a d-dimensional rectangle with edge length equal to $\frac{1}{\omega}$. Thus,

$$\mathsf{Dist}\left(x, \mathsf{R}\left(x\right)\right) \leq \frac{\sqrt{d}}{\omega} \quad (14) \qquad \text{and} \qquad \frac{D}{|X|} \leq \frac{\sqrt{d}}{\omega} \quad (15)$$

Figure 2 illustrates an example, in which *Latitude* and *Longitude* are the 2 dimensions, $\omega = 4$, and the distance between a tuple and its representative is at most $\frac{\sqrt{2}}{4}$. Plugging Inequality 15 in Inequality 11 gives

$$\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) \leq \alpha \mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right) + (1+\alpha) \cdot \frac{\sqrt{d}}{\omega}. \quad (16)$$

The above Inequality implies a trade off between efficiency and accuracy. When the cells are of coarse granularity, the number of cells is small. Thus, the clustering is efficient, but $(1+\alpha) \cdot \frac{\sqrt{d}}{\omega}$ is big, leading to low clustering accuracy. As the cell granularity is finer, the number of cells increases. The efficiency decreases, but the accuracy improves. When each cell has a single tuple, our solution is the same as (α, β)-approximation algorithm running directly on input dataset.

We now study how to decide the ω value. Both $\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right)$ and $\mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right)$ are data dependent – their values change with data distribution. We thus also dynamically compute ω according to data distribution. Our computation starts from an analysis of relative distance between $\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right)$ and $\mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right)$.

Corollary 1 *Let X and $\overline{X}$ be an input dataset and its cells, respectively. Suppose that $\overline{\mathcal{O}}$ is the cluster centers output by an (α, β)-approximation k-median algorithm on $\overline{X}$, and $\mathcal{O}^{\circ}$ is the cluster centers output by an optimal solution on X. If the following inequality holds*

$$\frac{D}{|X|} \leq \frac{(\gamma - \alpha)\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right)}{\alpha(\gamma+1)}, \quad (17)$$

then $\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) \leq \gamma \mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right)$, where $\gamma > \alpha$ and $D = \sum_{x \in X} \mathsf{Dist}\left(x, \mathsf{R}\left(x\right)\right)$.

Proof. Combining Inequality 17 with Inequality 8, it follows that

$$\frac{\mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right)}{\mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right)} \leq \frac{\alpha(\gamma+1)}{\alpha+1},$$

which (if combined with Inequalities 17 and 11) gives $\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right) \leq \gamma \mathsf{Cost}\left(X, \mathcal{O}^{\circ}\right)$.

Now consider Inequalities 15 and 17 together. Clearly, Inequality 17 holds, if

$$\omega \geq \frac{\sqrt{d}}{\eta \cdot \mathsf{Cost}\left(\overline{X}, \overline{\mathcal{O}}\right)},$$

where $\eta = \frac{(\gamma-\alpha)}{\alpha(\gamma+1)}$. In our experiments, we empirically set $\eta = 0.5$, which gives good results. Our solutions runs k-median clustering on sliding windows continuously. Let W_i (for $i = 1$ to ∞) be the sequence of sliding windows, $\overline{X_i}$ be the statistical information of cells in sliding window W_i and $\overline{O_i}$ be the cluster centers derived from $\overline{X_i}$. Then, we set

$$\omega_{i+1} = \frac{\sqrt{d}}{\eta \cdot \mathsf{Cost}\left(\overline{X_i}, \overline{O_i}\right)}. \tag{18}$$

Note that our grid-based clustering solution only has the accuracy guarantee in Inequality 16. It does not guarantee that the relative difference between $\mathsf{Cost}\left(X, \overline{\mathcal{O}}\right)$ and $\mathsf{Cost}\left(X, \mathcal{O}^\circ\right)$ is at most γ, since the required ω value in Corollary 1 might be bigger than the one used to partition the data space. In brief, Equation 18 is a heuristic, which determines the ω value adaptively based on data distribution. Furthermore, as the dimensionality of the dataset increases, the data easily becomes sparse. As the ω value increases, the number of cells with small number of tuples increases fast. To address this, we set a threshold, which is the maximum allowed ω value for a data stream.

4 The Solution

We now develop our continuous k-median clustering solution for stream data based on the theoretical analysis in Section 3. For the clarity of presentation, we first present an algorithm, for which the sliding window size is equal to the step size (i.e., tumbling window). We will later extend it to the general case, where neighboring sliding windows may overlap.

Algorithm 1 is the solution for tumbling windows. Lines 1 and 2: an iterative search is applied on historical data to find an initial ω value, such that the clustering cost by the approximation algorithm on the cells (of historical data) determined by the ω value is 'similar' to the clustering cost by the same approximation algorithm on the input historical data. In the experiments, we use 10% tuples in a dataset as historical data, and the remaining ones as stream data. The streaming tuples are processed one by one on the fly (Lines 4 to 21). Given a tuple, the cell containing it is computed (Line 6). If the cell exists, then simply update the statistical information of the cell (Lines 7 to 8). Otherwise, a new cell is created (Lines 9 to 10). Our cell management strategy stores only non-empty cells, and thus ensures that the total number of cells is never larger than the total number of tuples. Once a new sliding window is generated, approximation algorithm runs on the statistical information of the cells (Lines 12 to 20), and ω is updated for the next sliding window (Line 19).

In the algorithm we maintain a map $\mathcal{M}$ to efficiently locate the cells containing incoming streaming tuples. We use the cell's boundary as the key of the map. Each cell has simple statistical information, which is the summation of tuple values in the cell, the cell size, and the cell mean. We use the mean to represent the cell. Other statistical information can be easily added if needed,

Algorithm 1 adapGridKM

Input: data stream $\mathcal{DS}$, window size WS, and step size S
Output: the continuous update of βk cluster centers

1: Let HD be a set of history data of $\mathcal{DS}$
2: $\omega \leftarrow$ findOmega (HD)
3: Initialize an empty map $\mathcal{M}$
4: **for** $i \leftarrow 1$ **to** $|\mathcal{DS}|$ **do**
5: Let $\mathcal{DS}$ [i] be the i-th tuple in $\mathcal{DS}$ ▷ *Lines 5-10: Insert a tuple to a cell.*
6: $c \leftarrow$ Compute$(\mathcal{DS}[i], \omega)$
7: **if** $c \in \mathcal{M}$ **then**
8: $c.sum = c.sum + \mathcal{DS}[i], \quad c.size = c.size + 1$
9: **else**
10: $c.sum = \mathcal{DS}[i], \quad c.size = 1, \quad$ Insert c into $\mathcal{M}$
11: **end if**
12: **if** $(i \geq WS) \wedge (i \mod S == 0)$ **then**
13: **for** each cell $c \in \mathcal{M}$ **do** ▷ *Lines 13-19: Cell clustering and ω update.*
14: $c.mean = \frac{c.sum}{c.size}$
15: **end for**
16: Run (α, β)-approximation clustering on the means of the cells
17: Update the cluster centers
18: Let $\mathcal{C}$ be the cost of clustering on the cells
19: $\omega \leftarrow \frac{\sqrt{d}}{\eta \cdot C}, \quad$ Empty $\mathcal{M}$
20: **end if**
21: **end for**

and the cell representative can also be changed to cell center or a streaming tuple in the cell or some function as required.

We now extend Algorithm 1 to the general case, where neighboring sliding windows may overlap. The extension is simple. In Algorithm 1, each sliding window has an ω value, and all the tuples in the window are partitioned into cells only once based on the ω value. In the general case, overlapping windows share common tuples, which need to be partitioned into cells for multiple times, each for one sliding window. As such, we maintain an ω value for each step, and partition all the tuples in a window based on the ω value of the first step in the window. We use the following example to better illustrate our ideas.

Example 1 *In Figure 1, we assume that the sliding window is 3 times as big as the step. Suppose that at timestamp t we have the first sliding window W_1, which includes steps S^1, S^2, and S^3. Furthermore, suppose that the ω values of these three steps are ω_1, ω_2, and ω_3, respectively. Then, the tuples in windows W_1, W_2, and W_3 are inserted into cells determined by ω_1, ω_2, and ω_3, respectively. We run clustering on the cells of W_1. An ω value is then computed by Equation 18. Denote the value by ω_4, and assign it to step S^4. All the tuples in window W_4 will then be inserted into cells determined by ω_4. At timestamp $t+S$, window W_1 slides forward to window W_2. Clustering is run on the cells of W_2, and another*

ω value (denoted by ω_5) is computed and assigned to step S^5. All the tuples in window W_5 (not shown in Figure 1) are inserted into cells determined by ω_5. At timestamp $t+2S$, window W_2 slides forwards to window W_3. The tuple insertion into cells, clustering, and window sliding then continue iteratively like the above.

Storage Complexity. Consider a single cell. In the d-dimensional data space, the storage cost of its boundary is $2d$. Its statistical information (i.e., summation, size, and mean) is $2d+1$. Thus, the storage cost of a cell is $4d+1$. Let n be the total number of cells in a sliding window. Then, the storage cost is $\Theta(nd)$. In our work, we only store non-empty cells. Thus, $n \leq WS$.

Time Complexity. The time cost consists of inserting tuples into cells and k-median clustering on cells. Given a tuple, locating the cell containing it takes $2d$. Updating the cell size and the summation takes $d+1$. Therefore, the time complexity of inserting all the tuples of a sliding window into cells is $\Theta(WS \cdot d)$. The time cost of clustering varies from one approximation algorithm to another. For the simplicity we consider one of the core operations – computing the clustering cost (i.e., distances between tuples and their nearest centers as defined in Equation 1). For this operation, the cost of an approximation algorithm on input data is $\Theta(WS \cdot k \cdot C_d)$, where C_d is the cost of computing the distance between two points in the d-dimensional space, $k \cdot C_d$ is the cost of finding the nearest center out of the given k centers for a given tuple, and WS is the window size. For this operation, the cost of the same approximation algorithm running on the cells is $\Theta(n \cdot k \cdot C_d)$, where n is the number of cells in the sliding window. Since $n \leq WS$, the approximation algorithm running on cells is faster than that on input data. The experimental results in the next section conform this.

5 Experimental Evaluation

5.1 Experimental Setup

We adopt the $(5+\epsilon)$-approximation k-median algorithm [10], which is based on local search. It first randomly selects k cluster centers as the initial solution. It then iteratively improves the centers, until a better solution with an improvement factor of at least $\frac{\epsilon}{5k}$ cannot be found. We denote this algorithm applied on input stream tuples by locSearch. We build our grid-based k-median clustering algorithm on top of [10] – we use it to cluster cells generated from streaming tuples. We denote our algorithm by adapGridKM, representing adaptive grid-based k-median clustering. Furthermore, we include a benchmark fixedGridKM, whose only difference from adapGridKM is fixing the ω value.

We use three real datasets in the experiments. The first is the 2006 Topologically Integrated Geographic Encoding and Referencing (TIGER) dataset [1]. We extract the GPS coordinates of road intersections in the states of Washington and New Mexico. We randomly sample 1 million tuples, and set $k = 2$. The second is the Gowalla dataset [2], which records the check-in information of users of a social network. The information includes user ID, check-in time, check-in

location (by latitude and longitude), and location ID. We keep check-in time, latitude, and longitude, and randomly sample 2 million tuples. For this dataset, we set $k = 5$. The last is Power dataset [3], which measures the electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. After removing tuples with missing values, 2,049,280 tuples are left. We keep all its 9 attributes, and set $k = 4$.

We treat each dimension equally for its contribution to the clustering cost (Equation 1) by normalizing its values to [0.0, 1.0]. Our reported experimental results are window-based. We compute various measures (e.g., clustering cost and elapsed time) for each sliding window, and report the average for all the sliding windows. All the experiments were conducted on an Intel dual core i7-3770 CPU machine with 8G RAM running windows 7.

5.2 Effectiveness and Efficiency

We first investigate the experimental results of the TIGER dataset. In the first row of Figure 3, we fix the step size to 30,000, and vary the window size from 30,000 to 210,000. Figure 3(a) compares the three approaches with respect to effectiveness (i.e., clustering cost by Equation 1). The tuning on history data suggests $\omega = 128$. We thus set the initial ω value to 128 for adapGridKM, and fix the ω value of fixedGridKM to 128. Figure 3(a) shows that the tow grid-based approaches, adapGridKM and fixedGridKM, are as effective as locSearch, i.e., their clustering cost is very close to that of locSearch.

Figure 3(b) compares the approaches in terms of efficiency (i.e., average clustering time for a window). The results show that adapGridKM is consistently 4 orders of magnitude faster than locSearch. The fixedGridKM approach is also more efficient than locSearch, but less efficient than adapGridKM. A careful study of the experimental output shows – the ω values computed by adapGridKM are mostly 32 and 64, and only a few 128. Thus, on average the number of cells in a sliding window of adapGridKM is smaller than that of fixedGridKM. This explains why adapGridKM is more efficient. As the window size increases, the number of tuples (cells) grows. Thus, the time cost for all the approaches grows. Considering Figures 3(a) and 3(b) together, we can see that the two grid-based approaches have similar clustering cost [1], but adapGridKM is much faster than fixedGridKM. The reason behind is that fixedGridKM overestimates the ω value based solely on historical data. This result highlights the importance of adaptively determining the ω value by stream data distribution.

Next, for the TIGER dataset, we fix window size to 180,000, and vary the step size from 10,000 to 180,000 (the second row of Figure 3). Again, in Figure 3(c) the clustering cost of adapGridKM is very close to that of locSearch. For locSearch, the number of tuples for clustering is equal to the fixed window size. Hence, as the step size increases, the average time of clustering tuples does not

[1] When window size is 180,000, adapGridKM has even lower cost than fixedGridKM. The possible reason is that a finer grid granularity does not necessarily give lower clustering cost, since clustering output also depends on data distribution.

change obviously. For adapGridKM (locSearch), on average the number of cells generated from each sliding window is almost the same when varying the step size. Thus, the clustering time does not vary obviously either.

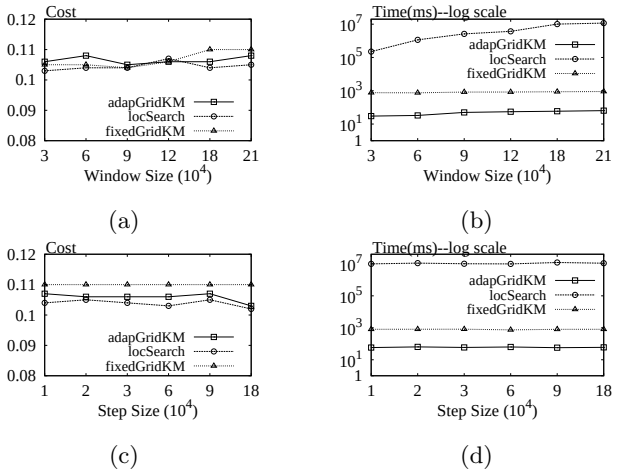


Fig. 3: The evaluation on TIGER dataset

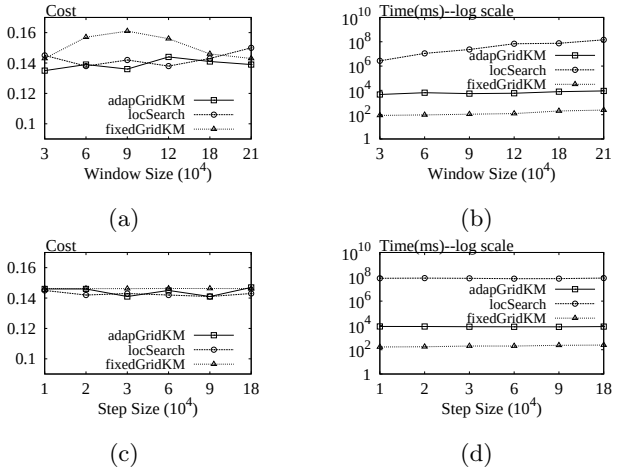


Fig. 4: The evaluation on Gowalla dataset

We now investigate the experimental results on the Gowalla dataset. The tuning on the history data suggests $\omega = 8$. We thus set the initial ω value of adapGridKM to 8, and fix that of fixedGridKM to 8. Figure 4(a) reports the clustering cost. Clearly, adapGridKM outperforms fixedGridKM. Through a careful study of experimental output, we find that most ω values generated by adapGridKM are 32. This shows that fixedGridKM underestimates the grid granularity. The experimental results thus once again proves the importance of adaptively adjusting grid granularity based on stream data distribution. Since fixedGridKM has a coarser granularity, its is faster than adapGridKM in Figure 4(b).

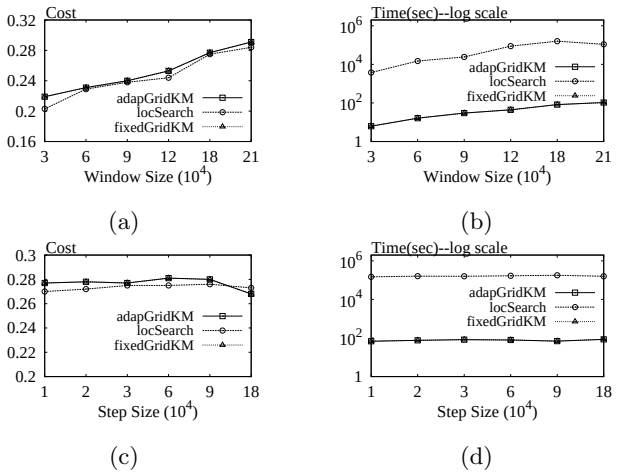


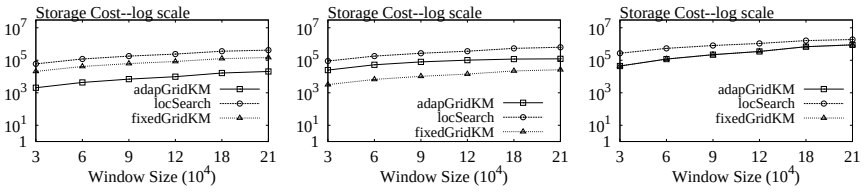
Fig. 5: The evaluation on Power Consumption dataset

Figure 5 reports the results on the Power dataset. This dataset has 9 dimensions; its data is much sparser than the other two lower-dimensional datasets. Thus, an increase of ω may increase the number of non-empty cells dramatically. To address this issue, we set an upper bound (i.e., 8) for the ω value. Such a setting converges the two grid-based approaches – they both use $\omega = 8$ to partition data into cells. Again, grid-based approaches are comparable to locSearch in terms of clustering cost, but are 2 orders of magnitude more efficient.

5.3 Storage and Tuple-processing Cost

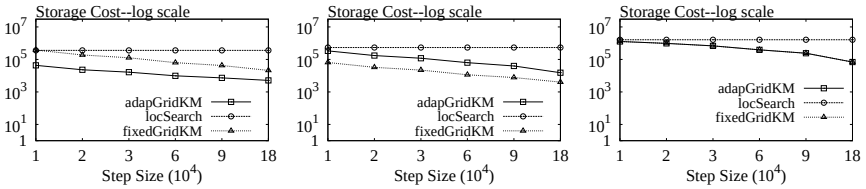
We now compare the storage cost. For a d-dimensional tuple, we take its storage cost as d, i.e., 1 unit for 1 coordinate. The storage cost of a cell as analyzed in Section 4 is $4d + 1$. Figure 6 shows the results as we vary the window size. Clearly, grid-based approaches need less storage. Take the TIGER dataset as an example (Figure 6(a)). When step size is 180,000, the storage cost of locSearch is 360,000, while those of adapGridKM and fixedGridKM are less than 17,000 and 130,000, respectively. The storage cost of grid-based approaches is up to the number of cells. For the TIGER dataset, the grid granularity of adapGridKM is coarser than that of fixedGridKM. For the Gowalla dataset, it is the opposite. Therefore, in Figure 6(a) adapGridKM has lower storage cost, while having higher cost in Figure 6(b). When window size increases, the number of tuples (and also the cells generated from them) grows. Therefore, the storage cost of all the approaches increases. In Figure 7 we fix the window size to 180,000, and vary the step size. When the step size increases, the number of overlapping windows we need to maintain is smaller. Therefore, the storage cost of the two grid-based approaches decreases.

The tuple-processing time of our approach adapGridKM consists of: a) *Insertion* – the time cost of inserting tuples into cells, and b) *Clustering* – k-median



(a) TIGER (b) Gowalla (c) Power

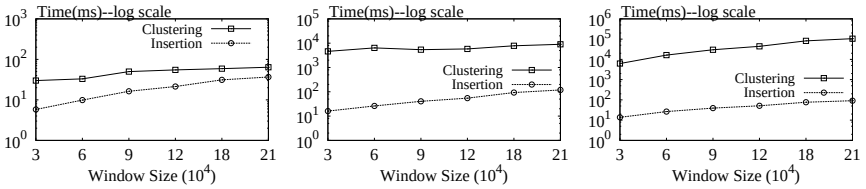
Fig. 6: The storage cost when varying window size



(a) TIGER (b) Gowalla (c) Power

Fig. 7: The storage cost when varying step size

clustering on the cells. Figure 8 and Figure 9 report the results, when varying window size and step size, respectively. In both figures, the elapsed time of *Insertion* is much smaller than that of *Clustering*. For example, in the Gowalla dataset, when the window size and the step size are 60,000 and 30,000, respectively, the elapsed time for *Insertion* and *Clustering* are approximately 26 ms and 6,400 ms, respectively.

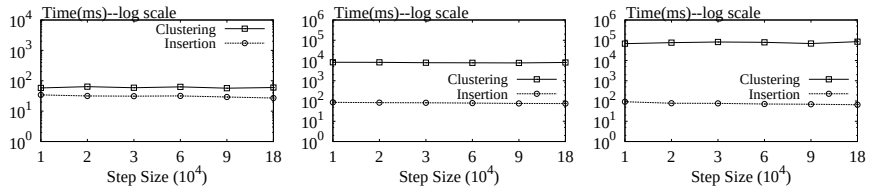


(a) TIGER (b) Gowalla (c) Power

Fig. 8: Clustering and tuple-insertion time when varying window size

6 Related Work

Grid-based clustering is closely related to our work. It was first proposed for static dataset. Representative solutions include STING [33] and CLIQUE [7]. Grid-based clustering has also been applied to the context of data stream to



(a) TIGER (b) Gowalla (c) Power

Fig. 9: Clustering and tuple-insertion time when varying step size

efficiently cluster continuous stream data. The proposed solutions include but not limited to D-Stream [17], cell-tree [30, 31], and top-m most frequent cells [21]. However, all the above solutions do not give accuracy guarantee of their clustering output. Although our solution is also grid-based, it has clear accuracy bound, thus distinguishing it from existing ones.

Coreset-based clustering [25, 5, 34, 4, 24, 20] is also related to our work. However, as stated in the introduction, such solutions cannot process streaming tuples on the fly. To compute the coreset of a set of tuples, the whole set needs to be available. Grid-based clustering solutions [17, 31] and ours instead can process stream tuples on the fly, i.e., processing tuples independently one by one. Besides the grid-based and coreset-based approaches, data stream clustering has also been studied in [6, 12, 18, 35, 22, 8]. Refer to [19] for more related work.

k-median clustering is NP-hard. Plenty of (α, β)-approximation algorithms have been proposed [29, 28, 9, 11, 13, 15, 26, 14, 10]; they ensure that the difference between the output and that of the optimal solution is within a given bound. Of the algorithms, [29, 28, 11, 13] have approximation factor α, which is dependent on the dataset size and/or the k parameter. Constant-factor approximation algorithms [15, 26, 14, 10] instead have a constant α. Charikar et al. [15, 16] proposed a $6\frac{2}{3}$-approximation algorithm. Jain and Vazirani [26] applied the primal-dual schema to k-median problem, and developed a 6-approximation algorithm. Charikar and Guha [14] refined algorithm [26] and developed a 4-approximation solution. Arya et al. [10] proposed a $(5+\epsilon)$-approximation k-median solution using local search heuristics, where ϵ is a tunable parameter controlling the convergence rate of the solution. Note that our solution is general. It can leverage any constant-factor approximation k-median clustering algorithm discussed above to build a grid-based clustering solution for data stream.

7 Conclusion

In this paper we have proposed a general and adaptive sliding-window-based k-median clustering solution. Our solution dynamically determines the granularity of cells, and runs clustering efficiently on the statistical information of cells. It has a theoretical accuracy bound between its clustering cost and the optimum. The extensive experimental results show that the proposed solution is efficient and effective.

References

1. https://www.census.gov/geo/maps-data/data/tiger.html.
2. https://snap.stanford.edu/data/loc-gowalla.html.
3. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.
4. M. R. Ackermann and J. Blömer. Coresets and approximate clustering for bregman divergences. In *SODA*.
5. M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. Streamkm++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithmics*, 17(1), 2012.
6. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.
7. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.
8. N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *NIPS*, pages 10–18, 2009.
9. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for euclidean k-medians and related problems. In *STOC*, pages 106–113, 1998.
10. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *STOC*, pages 21–29, 2001.
11. Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, pages 184–193, 1996.
12. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, pages 328–339, 2006.
13. M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median. In *STOC*, pages 114–123, 1998.
14. M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS*, pages 378–388, 1999.
15. M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *STOC*, pages 1–10, 1999.
16. M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.
17. Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *KDD*, pages 133–142, 2007.
18. G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. In *ICDE*, pages 1036–1045, 2007.
19. 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.
20. D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *SODA*.
21. J. Gama, P. P. Rodrigues, and L. M. B. Lopes. Clustering distributed sensor data streams using local processing and reduced communication. *Intell. Data Anal.*, 15(1):3–28, 2011.

22. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. Knowl. Data Eng.*, 15(3):515–528, 2003.
23. T. Guo, X. Zhu, J. Pei, and C. Zhang. Snoc: Streaming network node classification. In *ICDM*, 2014.
24. S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*.
25. S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, pages 291–300, 2004.
26. K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *FOCS*, pages 2–13, 1999.
27. N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB*, pages 804–815, 2004.
28. J. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Inf. Process. Lett.*, 44(5):245–249, 1992.
29. J. Lin and J. S. Vitter. epsilon-approximations with minimum packing constraint violation (extended abstract). In *STOC*, pages 771–782, 1992.
30. N. H. Park and W. S. Lee. Statistical grid-based clustering over data streams. *SIGMOD Record*, 33(1):32–37, 2004.
31. N. H. Park and W. S. Lee. Cell trees: An adaptive synopsis structure for clustering multi-dimensional on-line data streams. *Data Knowl. Eng.*, 63(2):528–549, 2007.
32. Y. Tao, X. Lian, D. Papadias, and M. Hadjieleftheriou. Random sampling for continuous streams with arbitrary updates. *IEEE Trans. Knowl. Data Eng.*, 19(1):96–110, 2007.
33. W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, pages 186–195, 1997.
34. Q. Zhang, J. Liu, and W. Wang. Approximate clustering on distributed data streams. In *ICDE*, pages 1131–1139, 2008.
35. Z. Zhang, H. Shu, Z. Chong, H. Lu, and Y. Yang. C-cube: Elastic continuous clustering in the cloud. In *ICDE*, pages 577–588, 2013.