Mar. 2006, Vol.21, No.2, pp.284–296 *J. Comput. Sci. & Technol.*

Efficient Computation of k-Medians over Data Streams Under Memory Constraints

Zhi-Hong Chong[1,4,5] (崇志宏), Jeffrey Xu Yu[2] (于　旭), Zhen-Jie Zhang[3] (张振杰), Xue-Min Lin[4] (林学民), Wei Wang[4] (王　伟), and Ao-Ying Zhou[1] (周傲英)

[1] *Department of Computer Science and Engineering, Fudan University, Shanghai 200433, P.R. China*

[2] *Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin N.T. Hong Kong Special Administration Region, P.R. China*

[3] *School of Computing, National University of Singapore, 117574 Singapore*

[4] *School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia*

[5] *Department of Computer Science and Engineering, Southeast University, Nanjing 210096, P.R. China*

E-mail: {zhchong, ayzhou}@fudan.edu.cn; yu@se.cuhk.edu.hk; zhangzh2@comp.nus.edu.sg; {lxue,weiw}@cse.unsw.edu.au

Received October 8, 2005; revised December 13, 2005.

Abstract In this paper, we study the problem of efficiently computing k-medians over high-dimensional and high speed data streams. The focus of this paper is on the issue of minimizing CPU time to handle high speed data streams on top of the requirements of high accuracy and small memory. Our work is motivated by the following observation: the existing algorithms have similar approximation behaviors in practice, even though they make noticeably different worst case theoretical guarantees. The underlying reason is that in order to achieve high approximation level with the smallest possible memory, they need rather complex techniques to maintain a sketch, along time dimension, by using some existing off-line clustering algorithms. Those clustering algorithms cannot guarantee the optimal clustering result over data segments in a data stream but accumulate errors over segments, which makes most algorithms behave the same in terms of approximation level, in practice. We propose a new grid-based approach which divides the entire data set into cells (not along time dimension). We can achieve high approximation level based on a novel concept called $(1-\epsilon)$-dominant. We further extend the method to the data stream context, by leveraging a density-based heuristic and frequent item mining techniques over data streams. We only need to apply an existing clustering once to computing k-medians, on demand, which reduces CPU time significantly. We conducted extensive experimental studies, and show that our approaches outperform other well-known approaches.

Keywords data streams, k-medians, cluster, data mining

1 Introduction

Emerging data-intensive applications, such as sensor networks, networking flow analysis, e-business and stock market online analysis, need to handle various high speed data streams. As a major data mining technique, data clustering over high speed data streams has been studied. With limited memory space and the constraint of scanning-data only once, the data clustering over data streams is requested to achieve the same goal as to minimize the intra-cluster distances and maximize inter-cluster distances. Typically, a data cluster is characterized by canonical representatives, called cluster medians to minimize the sum of distance to them, cluster centers to minimize the maximum distance to them or cluster means to minimize the sum of squared distance to them. Among them, the k-medians problem over a data stream is to find k such medians to minimize the sum of distances from the data points to their nearest medians. The k medians as summary information over the data stream play an important role in online analysis of data streams. Examples include clustering massive transaction data to find similar groups of customers, grouping close points, such as IP packets described as vector points, generated by sensors or routers in a large network and so on. The common requirement is to locate a few, say k, representative points to summarize the whole data set with minimum error while it is infeasible to store the massive raw data usually generated in stream form.

In this paper, we study the k-medians problem over high dimensional and high speed data streams. Guha and Mishra[1] spearhead the k-medians problem over data streams, and propose a divide-and-conquer (denoted as $D\&C$) algorithm which can achieve $O(2^{1/\epsilon})$-approximation with $O(n^{\epsilon})$ memory. A promising randomized algorithm PLS[2] achieves $O(1)$-approximation with $O(k \operatorname{polylog} n)$ memory. Computational geometry[3,4] recently cast light on this problem, introducing algorithms based on an approximation data structure, called *coresets*.

We observe that, through our extensive experimental studies, the existing algorithms achieve the similar approximation in practice, even though they make noticeably different worst case theoretical guarantees. The underlying reason is that in order to achieve high approx-

Regular Paper

This work is supported by the National High Technology Development 863 Program of China under Grant No. 2002AA413310 and ARC Discovery Grant, Australia under Grant Nos. DP0346004 and DP0345710.

imation level with the smallest possible memory, they need rather complex techniques to maintain a sketch. Such a sketch[1,3] needs to be maintained over an open-ended data stream. In order to maintain the sketch with high quality, the algorithms need to repeatedly apply some existing clustering algorithm over data segments of the data stream in an incremental fashion. However, because the existing clustering algorithms cannot guarantee the optimal clustering result over a data segment of a data stream, the accumulated errors over segments make most algorithms behave very similarly in terms of approximation level. In addition, such a technique of repeatedly applying clustering algorithms over a data segment makes those algorithms *inapplicable* to high speed data streams.

There are a few reported studies which do not need to repeatedly apply the existing clustering algorithm in order to maintain a sketch with high quality[3−5]. The work by Har-Peled and Mazumdar in [3] is very effective in a lower dimensional space but is not very desirable in a higher dimensional space due to an exponential space requirement regarding the dimensionality. In [4], Indyk gives a randomized algorithm to achieve a bicriterion $[1+\epsilon, O(\log\Delta(\log\Delta+\log(1/\epsilon)/\epsilon))]$-approximation, where Δ is the range width of a dimension; however, the bicriterion version of k-medians problem is different from the k-medians problem. In [5], Aggarwal *et al.* studied the problem of clustering evolving patterns over data streams. Their technique of maintaining a large number of micro-clusters, along the line of BIRCH[6], provides excellent opportunity for finding high-quality k-means over data streams. Their focus is to maintain clusters over time dimension and provide users with clustering patterns at any time slot with the aim to solve the k-means problem that is substantially different from the k-median problem. Therefore, they need to maintain clusters well scattered over the data space even a cluster may be too sparse to capture the unique feature of medians by using means.

The main contributions of this paper for the k-medians problem over high-dimensional and high speed data streams are listed below.

• We propose a new grid-based approach which divides the entire data space into cells. Our approach is different from [1, 2] because they divide data along time dimension and require to run some clustering algorithm repeatedly, to maintain the quality of a sketch at the cost of high CPU usage. The key issue of our grid-based approach is whether it is possible to use limited number of cells to maintain a sketch. We introduce a notion, called $(1-\epsilon)$-*dominant* cell, which indicates the fraction of the maximum number of data points in a cell to be assigned to the same nearest median. A 1-dominant cell means that all data points in a cell have the same nearest median and hence no error is introduced as long as we cluster the representative points of such cell correctly. With $(1-\epsilon)$-dominant, we can achieve a high approximation level *without* the necessity of increasing the number of cells by dividing the grid into even smaller cells (exponential explosion of memory usage). We present our theoretical approximation level for a data set.

• We provide two algorithms to maintain a sketch over a data stream with limited memory based on an heuristic to maintain dense cells which are most likely to be 1-dominant. Our technique thus maintains high density cells by treating such cells as frequent items over data streams and employing sophisticated cell management strategies. One feature is that we do not need to repeatedly apply any existing clustering algorithm to maintain a sketch. The CPU cost for maintaining sketch is thus minimized.

• We conduct extensive experiments to study different k-medians algorithms and show that the performance advantages of our grid-based approaches based on both synthetic and real data sets. Note: most algorithms for k-medians over data streams provide theoretical results. To the best of our knowledge, it is the *first* reported study of their effectiveness using a large number of different data streams.

The remainder of this paper is organized as follows. Section 2 gives the problem definition of k-medians. Section 3 gives a grid-based sketch. We discuss a grid-based approach with a notion called dominance, which can achieve a good approximation. Section 4 introduces two approaches to find high quality clusters by leveraging a cell density-based heuristic and streaming frequent item mining algorithms. Experimental results are presented in Section 5 followed by related work in Section 6. Finally, Section 7 concludes the paper.

2 Problem Definition

Consider a data stream as a sequence of $T = t_1, t_2, \ldots, t_i, t_{i+1}, \ldots$ where t_i is a data point in a ξ-dimensional metric space with distance function $d()$. The *continuous k-medians problem* of T is defined as computing k points, called *medians*, in ξ-dimensional space to minimize the sum of distances from the data points in T to their nearest medians. Note that these k medians do not necessarily belong to T in contrast to *discrete* k-medians problem. Below we formally define the problem.

In this paper, a set of medians of size k is denoted by $M = \{m_1, m_2, \ldots, m_k\}$ and the distance between a point t and a median set M is defined as follows.

Definition 1. *Given a point t and a set M of medians, $d(t, M) = \min\{d(t, m_i) | m_i \in M\}$.*

Therefore, the cost [①] $C_M(T)$ of M against T is defined below:

$$C_M(T) = \sum_{t \in T} d(t, M). \quad (1)$$

[①]For *k-means* problem, the cost is $C_M^2(T) = \sum_{t \in T} d^2(t, M)$ instead of $C_M(T)$.

The problem of continuous k-medians regarding T is to select a set M_T^* of k medians, such that the cost $C_{M_T^*}$ is minimized. That is $C_{M_T^*}(T) \leqslant C_M(T)$ for all M of k points. The quality of an approximate solution M is measured with the following ratio θ:

$$\theta = C_M(T)/C_{M_T^*}(T). \quad (2)$$

It is well known that finding an exact solution M_T^* is NP-hard even for a static data set T. The problem is even harder in data streams. The real-time computation over high speed data streams requires data to be stored in memory. However, data streams are usually massive in size and it is infeasible to maintain all data points in memory for such applications. Consequently, it needs to selectively maintain a sketch of T, denoted $\widetilde{T}$, in main memory such that $|\widetilde{T}| \ll |T|$. Here, each data point $\widetilde{t} \in \widetilde{T}$ in the sketch, is associated with a weight $\omega(\widetilde{t})$, which denotes how many data points in the original T $\widetilde{t}$ represents. Note that $\widetilde{T}$ does not have to be a subset of T.

In the following, we will discuss how to select such a $\widetilde{T}$ and obtain a set M of k-medians over $\widetilde{T}$ to approximate M_T^*. The cost function below regarding M and $\widetilde{T}$ is useful in our later analysis:

$$C_M(\widetilde{T}) = \textstyle\sum_{\widetilde{t}\in\widetilde{T}} \omega(\widetilde{t}) \cdot d(\widetilde{t}, M). \quad (3)$$

3 Grid-Based Sketch and Its Approximation

In this section, based on grid cells, we present a framework to construct a small sketch $\widetilde{T}$, for a data set T input in a data stream. The highlight is that we propose a notion of *dominance* for cells, followed by the discussions of approximation accuracy and the possibilities of significantly reducing the number of cells.

3.1 Grid-Based Sketch

Our algorithm to continuously maintain a sketch $\widetilde{T}$ over a data stream is based on an even partition of the whole data space by grids, or *cells*. Without loss of generality, we assume that the value domain on each dimension is the same as $R = [0, \mathcal{R}-1]$. For simplicity, we first discuss such a grid partition under the assumption that $\mathcal{R}$ is known. Then, we will show how to adjust the grid partition if new data points are beyond $[0, \mathcal{R}-1]$.

The value domain R on each dimension is divided evenly into intervals with length w ($w = \mathcal{R}/2^l$, where l is a non-negative integer). We call l *granularity level*. When $l = 0$, the entire data set is considered as a single cell. The granularity level determines the width w of a cell on each dimension.

A data point $t_i = (t_i^1, t_i^2, \ldots, t_i^\xi)$, where t_i^j represents the j-th dimension value of t_i, can be normalized using the following function $\phi_w(t_i)$:

$$\phi_w(t_i) = \left(\left\lfloor \frac{t_i^1}{w} \right\rfloor, \left\lfloor \frac{t_i^2}{w} \right\rfloor, \ldots, \left\lfloor \frac{t_i^\xi}{w} \right\rfloor\right). \quad (4)$$

Given a point t_i, we can identify the cell $\phi_w(t_i)$, called *grid code*, where t_i is populated. A cell x is a set of points satisfying $\phi_w(t_i) = \phi_w(t_j)$ for any pairs of points in this cell. For each cell x, we maintain a representative point x^r, such that x^r is chosen to minimize the total distance $\sum_{t_i\in x} d(t_i, x^r)$. As a result, $\widetilde{T}$ is a set of such representative x^r points for all non-empty cells. The weight of $x^r \in \widetilde{T}$, $\omega(x^r)$, is the number of data points in x. We will later show how to maintain x^r approximately.

In many data stream applications, it is unlikely that the domain on each dimension is known before hand. This can be dealt with by using an initial domain $[0, \mathcal{R}-1]$ for every dimension based on users domain knowledge. When a new arrival data point is beyond the domain in one dimension, we double $\mathcal{R}$. The width w will be changed from $w = \mathcal{R}/2^l$ to $2 \cdot \mathcal{R}/2^l$. An original cell, x, will then be a subcell of a new cell with a width twice that of x.

Example 1. Consider a 2-dimension space where the domain $\mathcal{R} = 4$. Let $w = \mathcal{R}/2^l = 4/2 = 2$ for $l = 1$. Suppose that there exists a data point, $t_i = (3, 3)$. $\phi_2(t_i) = (1, 1)$ identifies that t_i belongs to the cell x with the grid code $(1, 1)$. Suppose a new data point, $t_k = (5, 5)$ comes, which is out of the range $[0, 3]$. We double $\mathcal{R}$. Then the new domain is $[0, 7]$, and w is changed to 4 accordingly. The original cell x with the grid code $(1, 1)$ becomes a part of the cell with the grid code $(0, 0)$. The new data point t_k belongs to the cell with the grid code $(\lfloor 5/4 \rfloor, \lfloor 5/4 \rfloor) = (1, 1)$ in the new data space.

Clearly, in such a grid-based method the required space is $\Omega(2^{l\cdot\xi})$ if we keep each grid cell, as there are 2^l partitions on each dimension. Such an exponential space requirement is not desirable in data stream applications in a reasonably high dimensional space (i.e., ξ is reasonably large, say, 32) due to space limits.

Overview of Our Algorithms. To resolve the above exponential space requirement, in this paper we develop a novel cell-based algorithm. In our algorithm, we divide the whole data space in a ξ-dimensional space evenly into $2^{l\cdot\xi}$ grid cells as described above. However, to avoid an exponential space requirement we do not materialize each cell even when all of them have data points. Instead, we use the notion of *cell density* to effectively identify those "most significant" cells and then use them to absorb less "significant" cells to form large cells and therefore significantly reduce the space requirement. For each such cell x (an original grid cell or a large cell by merging grid cells together), we maintain a representative x^r that minimizes $\sum_{t_i\in x} d(t_i, x^r)$ as mentioned above. Final clustering results can be obtained by running a weighted k-medians algorithm on the sketch $\widetilde{T}$ instead of T.

3.2 Dominance

The approximate accuracy of our algorithms relies

on the notion of *dominance*. We define dominance regarding an exact solution of the continuous k-medians problem. Recall that $M_T^* = \{m_1, m_2, \ldots, m_k\}$ is an exact solution of the continuous k-medians problem for the data stream T. The dominance of a cell x is defined as the fraction of the maximum number of data points assigned to the same nearest median over the total number of data points in the cell. We denote it as $I(x, M_T^*)$. For example, suppose that 100 data points are in a cell x. 75 of them have m_i, 15 has m_j, and 10 has m_k as their medians. $I(x, M_T^*) = 0.75$. The dominance of the sketch $\widetilde{T}$ is defined as follows.

$$I(\widetilde{T}, M_T^*) = \frac{\sum_{x^r \in \widetilde{T}} I(x, M_T^*) \cdot \omega(x^r)}{|T|} \tag{5}$$

where $\omega(x)$ or $\omega(x^r)$ is the total number of points in the cell x or x^r represents. We say that the sketch $\widetilde{T}$ is $(1-\epsilon)$-*dominant* iff $I(\widetilde{T}, M_T^*) \geqslant 1-\epsilon$, where $\epsilon \in [0, \frac{k-1}{k}]$.

Recall that we obtain medians from sketch $\widetilde{T}$ for the original data stream T to approximate medians from T, the approximation factor is computed as follows, given a $(1-\epsilon)$-*dominant* sketch $\widetilde{T}$.

Charikar *et al.*[7] proposed an efficient polynomial time approximate algorithm to solve the discrete k-medians problem with a constant approximation factor λ. Consequently, applying the algorithm to the sketch $\widetilde{T}$ (a set of cell representative points) gives a λ-approximation to minimize (3) ②. For analysis, we use $\tau(t, M)$ to denote the nearest median in M for point t.

For a set M of k points in a ξ-dimensional space, based on triangle inequalities we have:

$$\begin{aligned} C_M(T) &= \sum_{t_i \in T} d(t_i, M) \leqslant \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, \tau(x^r, M)) \\ &\leqslant \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} (d(x^r, \tau(x^r, M)) + d(t_i, x^r)) \\ &= \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(x^r, \tau(x^r, M)) + \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r) \\ &= C_M(\widetilde{T}) + \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r). \end{aligned} \tag{6}$$

Here, each x is a partition cell of T for the construction of $\widetilde{T}$. It can also be immediately verified that:

$$C_M(\widetilde{T}) \leqslant C_M(T) + \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r). \tag{7}$$

Now, let $C(\widetilde{T}, T) = \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r)$. We have the following theorem.

Theorem 1. *A 1-dominant sketch $\widetilde{T}$ obtained against a data stream T leads to a c-approximation solution to the problem of the continuous k-medians problem, where c is a constant and $|\widetilde{T}| \geqslant k$.*

Proof Sketch. Let M' be a set of k-medians with λ-approximation to minimize (3) and M_T^* is an exact solution of the continuous k-medians problem for the data stream T. Then,

$$C_{M'}(\widetilde{T}) \leqslant \lambda \cdot C_{M^*_{\widetilde{T}}}(\widetilde{T}) \leqslant \lambda \cdot C_{M_T^*}(\widetilde{T}) \tag{8}$$

where $M^*_{\widetilde{T}}$ is the optimum k-medians set to minimize (3).

After replacing M in (6) by M' and using (8), we have:

$$C_{M'}(T) \leqslant \lambda \cdot C_{M_T^*}(\widetilde{T}) + c(\widetilde{T}, T). \tag{9}$$

Applying (7) by using M_T^* to (9), we obtain:

$$C_{M'}(T) \leqslant \lambda \cdot C_{M_T^*}(T) + (1+\lambda) \cdot C(\widetilde{T}, T). \tag{10}$$

Now, we show that $C(\widetilde{T}, T) = \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r)$ is less than or equal to $C_{M_T^*}(T)$ when the sketch $\widetilde{T}$ is 1-dominant (i.e., $\epsilon = 0$) regarding M_T^*. The following inequality holds because each representative data point x^r minimizes the sum of distance from x^r to the data points in x.

$$\sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, x^r) \leqslant \sum_{x^r \in \widetilde{T}} \sum_{t_i \in x} d(t_i, M_T^*) = C_{M_T^*}(T).$$

This together with (10) immediately implies the theorem where $c = 1 + 2\lambda$. □

Note that Theorem 1 holds regardless the number of cells and the scales of cells. This gives a great chance to obtain a good approximation without having a massive number of cells. Next, we discuss the approximation when data points in a cell do not have the same nearest median to be assigned to.

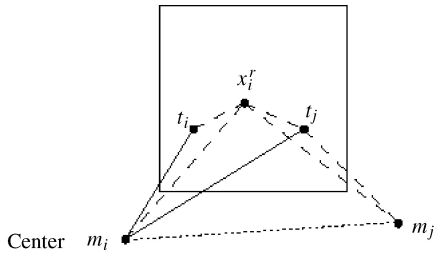


Fig.1. Example for $(1-\epsilon)$-dominant.

Fig.1 shows an example where a grid x_i contains points that have different nearest medians. Here, t_i has the median m_i and t_j to median m_j. When we assign the representative x_i^r (or all data points in x_i) to a median m_j, for point t_j, it incurs additionally cost at most $d(m_i, m_j)$ by triangle inequality. If $\widetilde{T}$ is $(1-\epsilon)$-dominant, then there are at most $\epsilon \cdot |T|$ points which give additional costs at most $\epsilon \cdot \Delta \cdot |T|$, where Δ is the maximum distance between any two medians. Therefore,

$$\begin{aligned} C_{M'}(T) &\leqslant C_{M'}(\widetilde{T}) + C(\widetilde{T}, T) \quad \text{(recall (6))} \\ &\leqslant (2\lambda + 1) \cdot C_{M_T^*}(T) + \epsilon \cdot \Delta \cdot |T|. \end{aligned}$$

② To apply the algorithm, we can view $\omega(x^r)$ as the number of the same points x^r for each x^r.

The term $\epsilon \cdot \Delta \cdot |T|$ is the additional cost when $\epsilon \cdot |T|$ data points are assigned to a wrong median. From $\theta = \frac{C_{M'}(T)}{C_{M_T^*}(T)}$, it follows that

$$\theta = (2\lambda + 1) + \epsilon\Delta / \overline{C_{M_T^*}(T)}. \quad (11)$$

Here, $\overline{C_{M_T^*}(T)} = C_{M_T^*}(T)/|T|$.

Theorem 2. *A $(1-\epsilon)$-dominant sketch $\widetilde{T}$ can achieve $(2\lambda + 1 + \frac{\epsilon \cdot \Delta}{\overline{C_{M_T^*}(T)}})$-approximation for the continuous k-medians problem.*

4 Computing k-Medians over Data Streams

While the basic sketch introduced in the last section has moderate memory usage and has upper bound on the approximation quality, it cannot be directly applied to streaming data. There are a few new challenges therein:

• *Hard Memory Bound.* Data stream algorithms usually have a hard limit on the memory usage. A clustering algorithm for streaming data must be able to work with a space budget smaller than the number of distinct cells while trying to minimize the additional cost $\epsilon \cdot \Delta \cdot |T|$.

• *Cell Management Strategies.* Due to the above reason, we first need to define which cells to be maintained in the sketch and then the strategy to process cells that will be included in the sketch as well as those that will *not* be included.

We briefly outline our approaches to the k-medians clustering over data streams which address the above questions. We propose a heuristic to selectively maintain only *dense* cells using a streaming frequent item mining algorithm; we still take into consideration the effects of sparse cells by merging a sparse cell with the surrounding dense cells.

4.1 Density-Based Heuristic

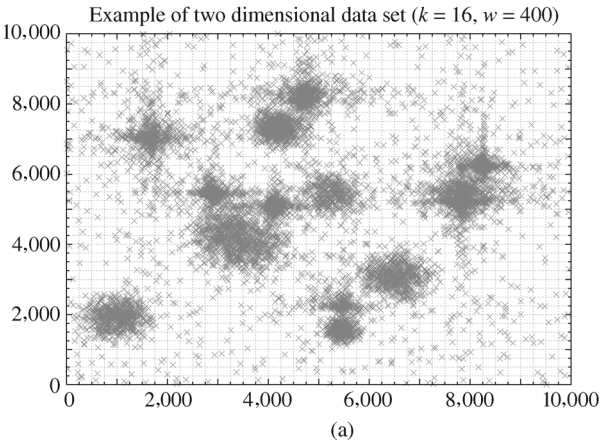
Our heuristic for constructing the sketch $\widetilde{T}$ for the whole data set T is to maintain cells whose densities are larger than a certain density threshold h_0. The major benefit of such heuristic is to control the maximum size of the memory needed while achieving a high quality summary of the original data distribution. Our density-based heuristic is based on the following observations:

• Dense cells are more likely to be pivotal to the final clustering surrounded by points in sparse regions. It is also more likely that a dense cell has nearly 1.0 dominance values.

• Although some *un-maintained* sparse cells might have low dominance values, their effects to the final clustering results are small as either the number of points within such cells are small or errors they introduced are small if they are located far away from any cluster centers.

We will illustrate each of the above observations in the following.

It is a common technique to monitor *dense* regions in the data space as an effective summarization and approximation for clustering applications[6,8−11]. As our approach is based on a regular grid over the data space, we naturally adopt the definition of *cell density*, defined as $h(x_i) = \omega(x_i)/|T|$, where $|T|$ is the total number of points in the data set T.



(a)

Density range (%)	Percentage (%)
0.00 – 0.10	65.00
0.00 – 0.20	85.00
0.00 – 0.30	90.00
0.00 – 0.40	97.50
0.00 – 0.50	97.50
0.00 – 0.60	97.50
0.00 – 0.70	100.00
0.00 – 0.80	100.00
0.00 – 0.90	100.00
0.00 – 1.00	100.00

(b)

Fig.2. Correlation of density and dominance on an example data set. (a) Data set. (b) Distribution of impure cells with respect to density. (Among all the cells, 7.13% are impure. $I(\widetilde{T}, M) = 99.20\%$.)

We observe that, in many cases, dense cells are likely to be 1-dominant. The intuition is that many natural clusters tend to have dense areas around their centers (see Fig.2(a)). This feature also underlies existing statistical clustering algorithms which essentially learn a mixture model of Gaussian distributions[12]. We have conducted extensive testing to investigate the relationships between dominant cells and dense cells, and found these two measures are highly correlated. We illustrate this using an example data set generated by our synthetic data generator also used in the experiment. The example two-dimensional data set, T, is shown in Fig.2(a). The data set T consists of 10,000 data points in $L = 16$ clusters where 8 clusters follow the Gauss distribution and 8 clusters follow the Pareto distribution. The width of a cell in the grid is 400×400, while the domain of each dimension is fixed at 10,000.

The correlation between cell density and cell domi-

nance is shown in Fig.2(b). In this example data set, there are only 7.13% of the cells whose dominance is strictly smaller than 1.0 (i.e., cells that will introduce error), regardless of the cell density. We also call those cells "impure" cells. We then partition impure cells into different density ranges, and compute the fraction of impure cells within that density range against all such cells. For example, the fourth row in table means 97.50% of the impure cells are cells with density below 0.40%. In other words, if we keep track of cells whose density is above 0.40%, we then have captured most of the 1-dominant cells. On the other hand, another benefit of maintaining the dense cells is that, among the cells with low dominance values, we essentially choose to maintain those who contain a large number of points in a greedy manner. Intuitively, this heuristic should work better than the one that maintains sparse impure cells with the same memory budget. In fact, the effectiveness of our density-based heuristic has been confirmed in our experiment.

Furthermore, although many of the sparse cells straddling boundaries of clusters might have a low dominance value, we argue that they might not introduce much error in the clustering results even if they are assigned to the wrong cluster. The reason is two-fold: on one hand, sparse cells have fewer number of points inside, and (in some sense, we have already picked dense cells to maintain); on the other hand, the errors introduced by $(1-\epsilon)$-dominant cells that are far away from any cluster centers are bounded. We show this effect in the following simplified example, as shown in Fig.3. We consider a cell whose center coincide with the perpendicular bisector of two cluster centers of $2d$ distance away (C_1 and C_2). The width of the cell is w. The lower edge of the cell is L distance from the middle point of C_1 and C_2. This example illustrates the "worst" case scenario where the dominance of the cell can be as bad as 0.5, when an adversary fill the cell with half of the points in the left half and the other half of the points in the right half. Now we consider any point q that falls within the cell with coordinates (x, y), with the constraints $x \in [-\frac{w}{2}, \frac{w}{2}]$ and $y \in [L, L+w]$. The maximum error, $err_{\max}$, introduced by wrongly assign a point to other cluster can be quantified by $|d(q, C_1) - d(q, C_2)|$. By solving the partial differential equations, we can show that the error is maximized when q is located at the left (or right) bottom corner of the cell, i.e., $err_{\max} = \sqrt{(d+\frac{w}{2})^2 + L^2} - \sqrt{(d-\frac{w}{2})^2 + L^2}$. When $\frac{d}{w} \ll 1$, $err_{\max} \approx w \cdot \frac{d}{\sqrt{d^2+L^2}}$. This means, with large enough L (i.e., when the cell is far away from any cluster center), the maximum error introduced by low dominance value cells can be negligible, regardless of which median the point is assigned to. Meanwhile, it can be shown that the number of cells "near" cluster centers (i.e., whose distance is smaller than L) is only a small fraction of the total number of cells, under the uniform assumption of data points.

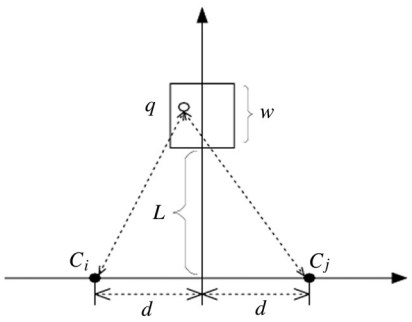


Fig.3. Upper bound of the errors introduced by $(1-\epsilon)$-dominant cells.

4.2 Generic Algorithm

Based on the density-based cell selection heuristic, the main task of our k-medians clustering algorithm becomes maintaining dense cells, i.e., cells whose density (so far) exceeds a certain threshold h_0, over the data stream. Our approach is to convert this problem into a frequent item mining problem, where an item is a cell, in a data stream, and the minimum support is set to be h_0.

We propose to adopt two existing algorithms as underlying routines in our solution and obtain two algorithms, namely, *P-Approx* and *N-Approx*. Algorithm *P-Approx* and *N-Approx* are based on the Lossy-Counting algorithm[13] and the FDPM-1 algorithm[14], respectively, which are designed to count frequent items in a data stream environment. The initial P/N means false-Positive/false-Negative, because Lossy-Counting is a false-positive approach, which allows some infrequent items with an error bound to be included in the result of frequent items, whereas FDPM-1 is a false-negative approach, which may miss some frequent items with probability control.

We show the generic version of our k-medians clustering algorithm in Fig.4. Both *P-Approx* and *N-Approx* follow the same flow but use different data structures and implement the subroutines createCell and mergeSparseCell using different cell management strategies (we will cover that shortly). The basic idea of the algorithm is to maintain a sketch $\widetilde{T}$ over the data stream and using a state-of-the-art k-medians algorithm to obtain the solution on demand from the sketch instead of the original data.

The algorithm takes three parameters, h_0, h_ϵ and η. h_0 is the density threshold, h_ϵ $(0 < h_\epsilon < h_0)$ is the error bound for finding frequent items, and η is a parameter used to control the merge process of a sparse cell. Initially, the number n of received data points is set with zero, and the sketch $\widetilde{T}$ is empty (Line 1). Consider a new data point t_i arrives. If there exists a cell x_j that t_i can be assigned to, it calls the function increaseCell which increases the count of a cell ν by 1 and recomputes its representative $\boldsymbol{x}_j^r = \frac{(\nu-1)\cdot \boldsymbol{x}_j^r + t_i}{\nu}$ (Line 3). We choose to

approximate the 1-median of points within a cell by its mean, as this significantly saves the computation cost without much sacrifice on the cost. Otherwise, we need to create a new cell for the incoming point (Line 4). Before that, we need to judge if we have enough space in memory by comparing the size of $\widetilde{T}$ with the given space bound denoted as β. If there is not enough space, we need to free the space used by sparse cells via the mergeSparseCell function (Line 4). After that, the call of createCell will create a new cell for the incoming point. Finally, in Line 6, it computes k-medians based on the sketch $\widetilde{T}$ by treating the representative point of each remaining cells as a data point with weight $\nu + \varrho$ and feeding them into an offline state-of-the-art k-medians algorithm.

Algorithm 1. Generic (h_0, h_ϵ, η)

1: $n \leftarrow 0$, $\tilde{T} \leftarrow \emptyset$;
2: **while** a new data point t_i arrives **do**
3: **IF** there is a cell x_j in $\widetilde{T}$ such that $\phi(t_i) \in x_j$ **THEN** incCell(x_j); $n \leftarrow n+1$ and break; //Not create a cell;
4: **ELSE IF** $|\widetilde{T}| \geqslant \beta$ **THEN** mergeSparseCell(η); createCell(x_j) and $n \leftarrow n+1$; //Create a cell;
5: **end while**
6: One demand, obtain k-median, M, over the sketch $\widetilde{T}$;

Fig.4. Algorithm for computing k-medians over data streams.

4.3 Cell Management Strategies

We describe the details of both *P-Approx* and *N-Approx* from the perspective of their cell management strategy.

One difference between our generic algorithm and the standard frequent item mining algorithm is that while the latter can simply discard an item (and its counts), we need to carefully preserve certain information associated with the item (i.e., cell) before it is deleted. This happens when the memory is not enough for a new item (cell). We note that different (offline or streaming) clustering algorithms take different approaches on this issue. In both of our algorithms, we choose to free up space by merging sparse cells into nearby cells.

Cell Merge Strategies in P-Approx. The cell management in *P-Approx* is described as follows.

• Each in-memory cell is represented as an entry of $(\nu, \varrho, \delta, \boldsymbol{x}_j^r)$, where ν is the number of data points in cell x_j since the creation, δ is used to estimate the maximum number of data points merged to other cells before the last creation, ϱ is the number of data points merged to itself from other cells and vector $\boldsymbol{x}_j^r$ is the representative point for $\nu + \varrho$ data points in the cell since the creation.

• We classify all the cells in $\widetilde{T}$ into one of the three classes: *dense* iff $\nu \geqslant (h_0 - h_\epsilon) \cdot n$; *sparse* iff $\nu + \delta < h_\epsilon \cdot n$; and *uncertain* otherwise.

• When a new cell needs to be created and there is not enough space, we consider each sparse cell according to the increasing order of density. For each sparse cell, we consider the following two options in turn: (a) merge the sparse cells into its closest dense cell [③], or (b) merge the sparse cell with its closest sparse cells with higher dense. The criterion to merge the sparse cell into its closest dense cell is that representative point of the sparse cell is within $\eta \cdot r$ distance from the representative point of the dense cell to be merged, where r is the radius of the dense cell in question. The motivation is that if the sparse cell is too far away from the dense cell, it is unlikely to belong to this dense cell. Similar heuristic is also employed in previous work[5].

• mergeSparseCell is implemented as follows: it loops through all the sparse cells in the increasing order of their density until room for a new cell is created; for each sparse cell it tries to merge it with other cells using the above mentioned merging criteria. When cell x_i merges into another cell x_j, we need to update the entry for cell x_j before x_i is removed from the memory as follows: (a) update the representative vector by

$$\boldsymbol{x}_j^r = \frac{(x_j.\nu + x_j.\varrho)x_j.\boldsymbol{x}_j^r + (x_i.\nu + x_i.\varrho)x_i.\boldsymbol{x}_i^r}{(x_j.\nu + x_j.\varrho) + (x_i.\nu + x_i.\varrho)}$$

(b) update the count of merged external points as: $x_j.\varrho = x_j.\varrho + (x_i.\nu + x_i.\varrho)$.

• createCell is implemented as follows: we initial the entry of the new cell as $(\nu, \varrho, \delta, \boldsymbol{x}_j^r) = (1, 0, h_\epsilon \cdot n - 1, \boldsymbol{t}_i)$. This compensation value $h_\epsilon \cdot n - 1$ is important as some dense cells might be merged into other cells and get deleted because it is not yet dense at certain timestamp.

• incCell is implemented as follows: we increase the number of points of the cell x_j by 1 and update its representative point. That is, we set $x_j.\nu = x_j.\nu + 1$ and $x_j.\boldsymbol{x}_j^r = \frac{(x_j.\nu - 1) \cdot x_j.\boldsymbol{x}_j^r + t_i}{x_j.\nu}$.

The key operation lies in mergeSparseCell. First, the merge process in essence leads to expanding an original cell by absorbing points in neighboring sparse cells. Second, a new cell can be created after its old cell at the same position was merged to a neighboring dense cell due to massive number of cells. The same process is considered in [13] where they insert the elements deleted in previous operations with compensate frequencies. The third, as a result, the number of cells is always kept below an appropriate number.

Cell Merge Strategies in N-Approx. The *N-Approx* is based on the false-negative frequent item mining algorithm, FDPM-1[14], which may miss some frequent items with probability control.

The cell management strategy in *N-Approx* is quite similar to that of *P-Approx*. We highlight the differences as follows.

• We only need to keep $(\nu, \varrho, \boldsymbol{x}_j^r)$ for each cell, as δ is always zero in *N-Approx*. Because the probability of a dense cell being discarded in FDPM-1 algorithm is h_δ, which can be controlled.

[③] In the special case when there is no dense cell, we will consider uncertain cells.

• As the FDPM-1 algorithm dynamically adjusts the error bound[14], h_ϵ does not need to be given to the algorithm. Instead, it is computed as $h_\epsilon = \sqrt{\frac{2h_0 \ln(2/h_\delta)}{n}}$ when needed, where n is the number of data points received and h_δ is the false-negative probability of the FDPM-1 algorithm.

• The criterion of *sparse cells* is now $\nu < (h_0 - h_\epsilon) \cdot n$. Hence there is no *uncertain cells*.

• createCell is implemented without setting the compensation value. That is, we initialize the entry of the new cell $(\nu, \varrho, \boldsymbol{x}_j^r)$ as $(1, 0, \boldsymbol{t}_i)$.

5 Performance Study

Our testing environment is a PC with 2.4GHz CPU, 512MB memory, running Microsoft Windows XP Professional. The algorithms we implemented are *D&C*[1], *PLS*[2], *Micro*[5] ④, together with our two proposed algorithms, *P-Approx* and *N-Approx* based on the false-positive and false-negative frequent item mining algorithms[13,14], respectively. Both *D&C* and *PLS* have theoretical guarantee on the approximation ratio ($2^{O(\frac{1}{\epsilon})}$ and $O(1)$, respectively), while *Micro* is one of the latest streaming clustering algorithms and has shown its superior performance to [15]. We used the k-means routine in [16] to get the final k-medians on demand for all the algorithms except *PLS*, as the alternative solutions (e.g., based on online facility location algorithm or local search) have extraordinary high time complexity[15]; *PLS* is an exception and has to rely on the online facility location algorithm[17].

We implemented all the algorithms using Microsoft Visual C++ 6.0. We have also configured all the algorithms using the best known parameters, as listed in Table 1. As a result, we do not impose any limit on the memory usage of the algorithms, but include this measure in our metrics. We note that imposing a uniform space limit will definitely penalize several theoretic algorithms.

Table 1. Default Parameters of Algorithms

Algorithm	Parameters
D&C[1]	segmentLen = 4000, $b = 6$
PLS[2]	$\gamma + 4(1 + 4(\beta + \gamma)) \leqslant \beta\gamma$
Micro[5]	$q = 10 \times k$, $t = 2$, initNumber = 2000
Ours	$h_0 = \frac{1}{8 \times k}$, $h_\epsilon = \frac{h_0}{10}$, $w = 512$

The metrics we used include elapsed time, memory consumption, and clustering quality. Specifically, we used both the total distance (*COST*) and sum square distance (*SSQ*) to measure the quality of the clustering results. We note that the former is the true cost according to the k-medians definition, while the latter is used to compare with previous algorithms[15]. Both of the costs can be represented in their absolute form or in the relative form. The relative cost (or SSQ) is the ratio of the cost over the minimum cost (or SSQ) achieved for the *same* data set by *any* of the algorithms in multiple runs. This ratio can be regarded as an optimistic estimate of the approximation ratio of the algorithms (θ), as it is impractical to find the cost (or SSQ) of the optimal solution. Similar ratios are used in previous work[15].

All the data in figures are calculated by taking the average of multiple runs of the algorithms. We used static and evolving synthetic data sets as well as real data sets in the experiments. The real data sets used are the KDD-Cup'98 (direct marking) and KDD-Cup'99 (network intrusion detection).

We designed a synthetic data generator with the goal of taking into consideration many factors that are likely to affect the performance of the clustering algorithms. To that end, we developed a data generator incorporating some of the *static* features used in BIRCH's data generator[6] as well as some *dynamic* features regulating the order of the data points in the data stream context. Those parameters and their default values are listed in Table 2.

Table 2. Parameters of the Data Generator

Parameters	Values	Default
number of dimensions (ξ)	2..32	16
number of clusters (k)	2..64	11
maximum ratio of radius (r_r)	1..10	5
maximum ratio of size (r_n)	1..5	2
ratio of outliers (r_o, %)	0..10	5
distribution (*distr*)	Gauss	Gauss
order	*RO, CO, FO*	*RO*

For the static features, we included maximum ratio of clusters, r_r, to be able to generate clusters with different spatial extensions; it is defined as the ratio of the radius of the largest cluster over that of the smallest cluster. Similarly, we included maximum ratio of size, r_n, to consider clusters with different numbers of points. We also generate $r_o \cdot n$ number of outliers distributed uniformly in the data space. The k cluster centers are randomly distributed in the space and either Gauss or Uniform distribution is used to generate its points.

We use the order parameter to control the dynamics of the stream. Currently, we support *RO* (random order), *CO*, and *FO*. *RO* randomly selects a cluster and a point within the cluster as the next arrival point, while *CO* (or *FO*) randomly choose a cluster but always choose the unchosen Closest (or Furthest) point to the cluster center as the next arrival point.

5.1 Varying Data Sets

In this subsection, we study the performance of all the algorithms by varying the characteristics of the data set. We use the default settings in Table 2 except for the parameter we studied.

Dimensionality (ξ). We first study the impact of di-

④ *Micro* is a simplified version of the algorithm described in [5] by removing data structures and procedures related to the temporal dimension.

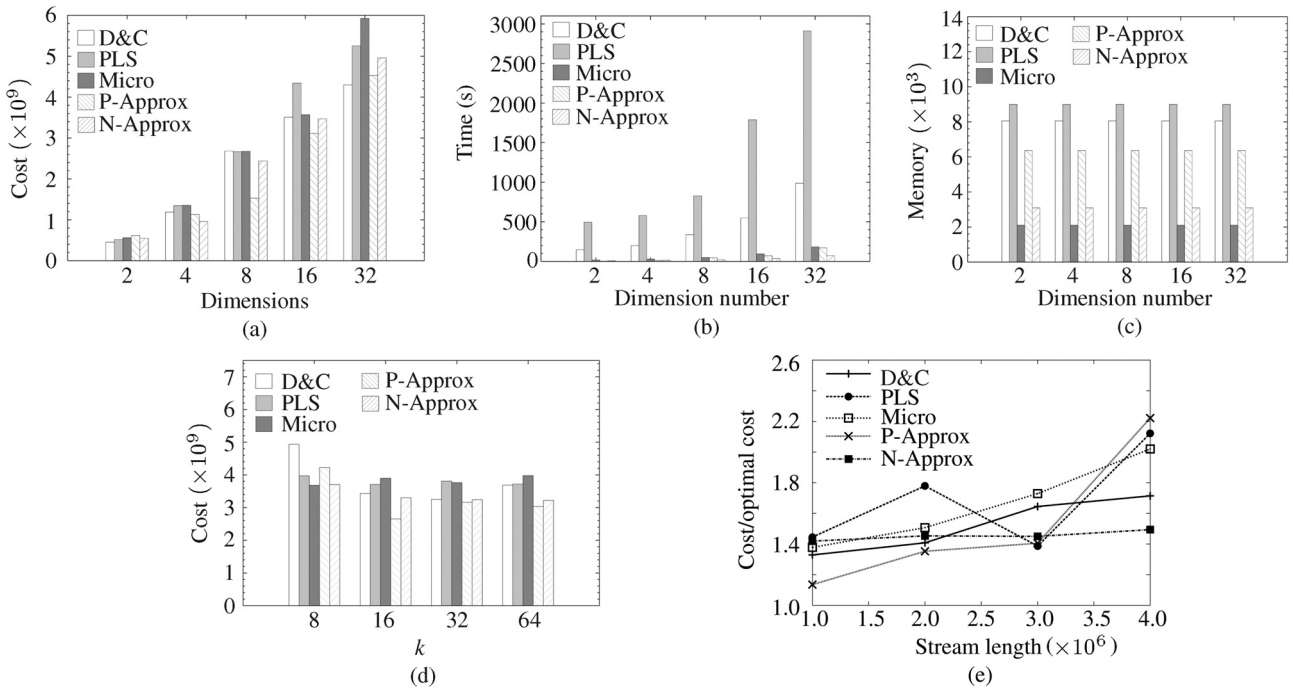


Fig.5. Varying data set characteristics. (a) Varying number of dimensions (cost). (b) Varying number of dimensions (time). (c) Varying number of dimensions (memory). (d) Varying number of clusters. (e) Varying the length of the stream.

mensionality to the algorithms. We varied the number of dimensions, ξ, from 2 to 32, and plot the results in Figs. 5(a), 5(b), and 5(c). Both of our algorithms perform better than the rest of the algorithms in most of the cases. Even for 32-dimension case, the *P-Approx* has comparable performance with *D&C*. However, our algorithms both require less memory and run much faster than *D&C*. We note that although *Micro* uses less memory than our algorithms, increasing its memory budget is not likely to improve its clustering quality[5], but will increase its running time significantly.

Number of Clusters (k). We varied the number of clusters k from 8 to 64, and plot the results in Fig.5(d). In general, none of the algorithms is sensitive to k.

Length of the Stream (L). We note that there is a potential danger that the clustering quality can deteriorate with the increase of the number of points processed. This is because many of the algorithms employ some *lossy* operations to summarize the data stream, and these losses can be accumulated and result in great deviation from the optimal solution. Here, we vary the length of stream from 1×10^6 to 4×10^6 and record the estimated approximation factors of all algorithms at regular intervals. The results are shown in Fig.5(e). It can be observed that the cost ratios of all the algorithm increase with the length of the data stream L, which coincides with our previous analysis. Specifically, *N-Approx* is the most stable one, followed by *D&C*. The rest of the algorithms are more sensitive to the length of the stream, as their quality deteriorates more noticeably as L increases.

Extreme Settings. We study the performance of algorithms for data sets with certain extreme settings. Specifically, we chose to use the maximum values in Table 2 (we use both *CO* and *FO* for the order parameter) to generate "hard" synthetic data sets. The results are depicted in Fig.6. Several observations can be made as follows.

- All algorithms except *Micro* are relatively insensitive to the extreme values of the parameters, both in terms of the running time and the quality of the clustering result.
- The clustering quality of *Micro* can be easily affected by the settings of those parameters. It seems *Micro* tends to work better for clusters with similar radius, number of points, and that there is no particular order among the points in the stream.
- All the algorithms except *D&C* achieve better quality under *CO* order than under *FO* order.

5.2 Evolving Data Streams

Some of the clustering algorithms have the implicit assumption that a sample of the data reflects the same data distribution characteristics of the entire data set. Therefore, we designed a data set whose distribution evolves over the time and evaluated all the algorithms on this data set. Such data set is also known as "concept drifting" data set[18]. The data set is generated by splitting the entire stream into segments of length 10,000, and a random shift within radius of 400 is added to every point within the segment. The costs of the algorithms are shown in Fig.7.

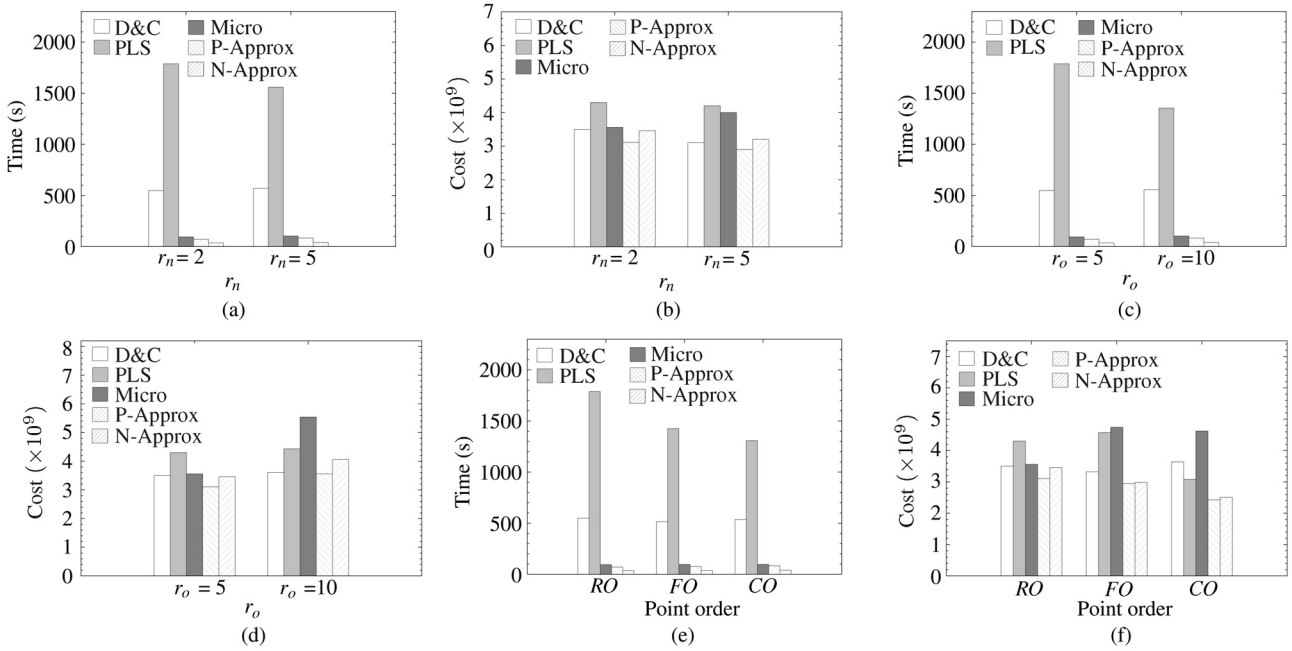


Fig.6. Extreme data set. (a) Time. (b) Cost. (c) Time. (d) Cost. (e) Time. (f) Cost.

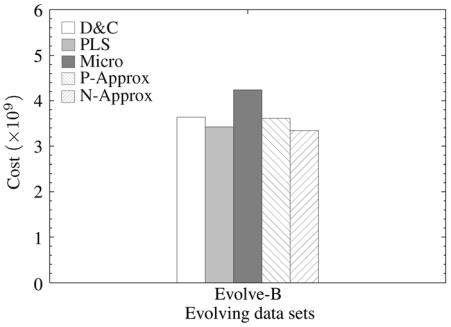


Fig.7. Evolving data set.

It is obvious that *Micro* performs worst on such evolving data set. The reason is that *Micro* relies on a subset of the first segment to find the initial cluster centers, which does not agree with the point distributions in the subsequent segments.

5.3 Sensitivity to the Algorithm Parameters

We first conducted a set of experiments to test the performance of the two proposed algorithms against different parameter settings.

We generate the synthetic data using the default settings in Table 2. We then fixed the width of the cell w to be 512. As there is no way to set an appropriate density threshold h_0 *a priori*, we empirically set the density threshold $h_0 = \frac{1}{\alpha \times k}$. The larger the parameter α is, the lower the density threshold becomes. We varied the density threshold h_0 by varying α from 2 to 32 (Recall that h_ϵ is defaulted to be $h_0/10$ for *P-Approx* and h_ϵ is not needed for *N-Approx*). The time, memory, and cost of *P-Approx* and *N-Approx* are shown in Fig.8. It can be observed that the memory required by both algorithms grows linear with the decrease of density threshold as the sketch will contain more cells; the runtime of *P-Approx* starts to increase sharply when the density threshold drops below a certain level (about 1.11%), due to the fact that more mergeSparseCell calls need to be made by *P-Approx* as it admits false-positive "dense" cells. In terms of the cost, both algorithms are insensitive to the threshold, while *P-Approx* generally achieves better clustering results than *N-Approx*.

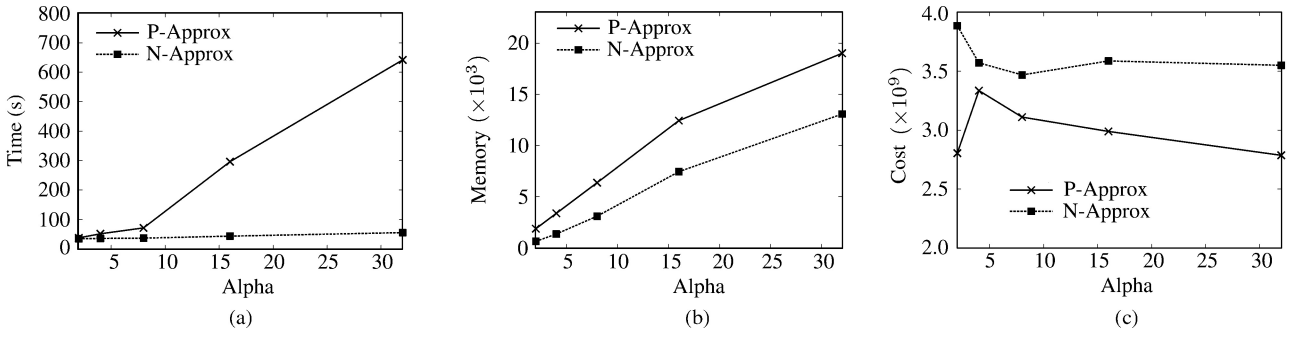


Fig.8. Varying the density threshold h_0. (a) Time. (b) Memory. (c) Distance cost.

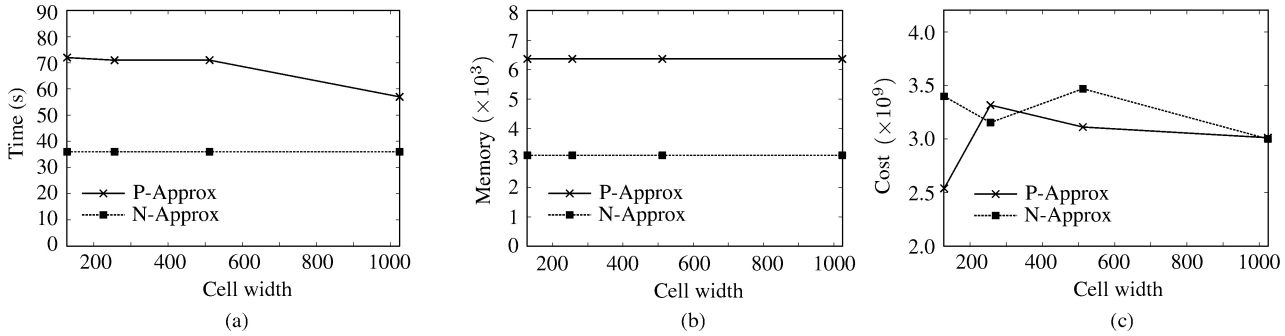


Fig.9. Varying the width of the cell w. (a) Time. (b) Memory. (c) Distance cost.

Next, for the same data set, we fixed the density threshold $h_0 = \frac{1}{8 \cdot k}$ and varied the width of the cell from 128 to 1024. The results are shown in Fig.9. The time and memory remains almost the same, as they are more related to the density threshold rather than the width of the cell. The cost of the two algorithms remains steady too, while *P-Approx* still marginally outperforms *N-Approx*.

5.4 Real Data Sets

We also tested all the algorithms on two real data sets: KDD-Cup'98 and KDD-Cup'99 data sets. Both of them are popularly used in previous work[15,19]. The following preprocessing is done: (a) we take the first 400,000 and 95,000 points from the two data sets, respectively; (b) we randomly extracted 32 numerical dimensions and applied normalization. We set $k = 5$ for the KDD-Cup'98 data set as there is one type of normal traffic and four types of traffic belonging to different kinds of attacks. We set $k = 11$ for the KDD-Cup'99 data set. The approximation of all algorithms is shown in Fig.10 under the sum of distance ($COST$) and sum of square distance (SSQ) metrics. We can observe that both *P-Approx* and *N-Approx* outperform *Micro* on both data sets using both metrics. In fact, possibly due to the different characteristics, *P-Approx* and *N-Approx* are the best algorithms for the KDD-Cup'98 and KDD-Cup'99 data sets, respectively. Depending on the metrics, *Micro* or *PLS* can be the worst algorithm for KDD-Cup'99 data set.

Summary. Our experiment with the previous state-of-the-art data stream clustering algorithms demonstrated that our proposed algorithms can not only achieve good clustering quality, but also features a small memory footprint and extremely fast processing time. They also have the salient feature that they are not sensitive to the characteristics of the data and program parameters.

6 Related Work

Guha and Mishra spearhead the problem of approximating k-medians in data streams in [1], where they achieved $2^{O(\frac{1}{\epsilon})}$-approximation using $O(n^{\epsilon})$ memory. The first constant approximation algorithm, denoted as PLS, in a random way was provided in [2].

The reported algorithms for k-median in a data stream environment include [5, 15, 20, 21]. In [15], it adopts a divide and conquer approach as reported in [1], with facility location algorithms to approximate k-median in each segment. Their results showed its advantage in less fluctuation. In [5, 21], the authors consider the time dimension of data stream elements. Based on the compressed structure of time stamps, EH[21] and pyramidal time frame[5] can be used to approximate k medians by repeatedly applying any existing k-median approximation algorithms on selected part of data streams.

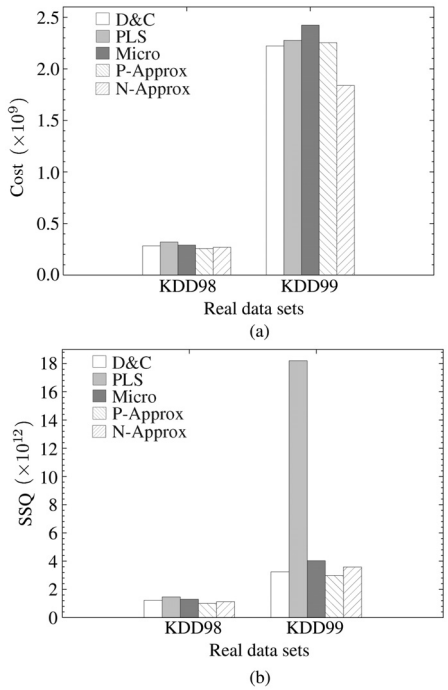


Fig.10. Real data sets. (a) Cost. (b) SSQ.

There exist algorithms based on computational geometry for k-median problem. Piotr Indyk[4] gives a randomized algorithm, denoted as XLS, to achieve a bicriterion $[1+\epsilon, O(\log \Delta(\log \Delta + \log(1/\epsilon)/\epsilon))]$-approximation in a low dimension space, devising an XS-Count sketch. In [3], Peled and Mazumdar introduce a

notion called coresets, give solutions in low dimension[3]. These algorithms provide theoretic quality guarantee, but are not efficient.

With a few exceptions[4,5], most existing algorithms for k-median problem in data streams need frequently applying a k-median algorithm to approximate $O(k)$ medians as the sketch[1−3], which is time consuming and accumulates errors.

There are also incremental algorithms like BIRCH[6] that can be adapted to the k-median problem. In BIRCH, its farthest pair split, closest criteria and the reducibility of CF tree makes BIRCH store the necessary summary information of massive data set in memory. This heuristic is similar with that used in [5] by greedily choosing the farthest or the closest pairs with specified threshold.

The time-efficient algorithms for approximating k medians or centers in static data set have been highlighted in [22, 23]. The high scalability algorithms for handling massive data sets are reported in [6, 8, 24].

[25, 26] are two comprehensive surveys for general clustering techniques. [27] contains a recent survey of clustering algorithms for data streams.

7 Conclusion

In this paper, we study efficient algorithms to approximate k-medians over high speed data streams. Our algorithms are based on the relationship of density and dominance, which lends us to leveraging frequent items mining paradigm to approach the k-medians problem. We conducted extensive experimental studies, and show that our approaches outperform other well-known approaches. In the future, we will extend our algorithms to other data stream clustering problems, such as sliding window and automatic subspace clustering.

References

[1] Guha S, Mishra N, Motwani R, O'Callaghan L. Clustering data streams. In *FOCS'00: Proc. the 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, USA, 2000, p.359.

[2] Moses Charikar, Liadan O'Callaghan, Rina Panigrahy. Better streaming algorithms for clustering problems. In *STOC'03: Proc. the 35th Annual ACM Symposium on Theory of Computing*, San Diego, CA, USA, 2003, pp.30–39.

[3] Sariel Har-Peled, Soham Mazumdar. On coresets for k-means and k-median clustering. In *STOC'04: Proc. the 36th Annual ACM Symposium on Theory of Computing*, Chicago, IL, USA, 2004, pp.291–300.

[4] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC'04: Proc. the 36th Annual ACM Symposium on Theory of Computing*, Chicago, IL, USA, 2004, pp.373–380.

[5] Charu C Aggarwal, Jiawei Han, Jianyong Wang, Philip S Yu. A framework for clustering evolving data streams. In *VLDB'03: Proc. 29th International Conference on Very Large Data Bases*, Berlin, Germany, 2003, pp.81–92.

[6] Tian Zhang, Raghu Ramakrishnan, Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD'96: Proc. the 1996 ACM SIGMOD Int. Conf. Management of Data*, Montreal, Quebec, Canada, 1996, pp.103–114.

[7] Moses Charikar, Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS'99: Proc. the 40th Annual Symposium on Foundations of Computer Science*, New York, NY, USA, 1999, p.378.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96: Proc. the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, USA, 1996, pp.226–231.

[9] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD'99: Proc. the 1999 ACM SIGMOD Int. Conf. Management of Data*, Philadelphia, Pennsylvania, USA, 1999, pp.49–60.

[10] Markus M Breunig, Hans-Peter Kriegel, Peer Kröger, Jörg Sander. Data bubbles: Quality preserving performance boosting for hierarchical clustering. *SIGMOD Rec.,* 2001, 30(2): 79–90.

[11] Samer Nassar, Jörg Sander, Corrine Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. In *SIGMOD'04: Proc. the 2004 ACM SIGMOD Int. Conf. Management of Data*, 2004, Paris, France, pp.467–478.

[12] Carlos Ordonez, Edward Omiecinski, Norberto Ezquerra. A fast algorithm to cluster high dimensional basket data. In *ICDM'01: Proc. the 2001 IEEE Int. Conf. Data Mining*, San Jose, California, USA, 2001, pp.633–636.

[13] Gurmeet Singh Manku, Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB'02: Proc. 28th Int. Conf. Very Large Data Bases*, Hong Kong, China, 2002, pp.346–357.

[14] Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Aoying Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *VLDB'04: Proc. the 30th Int. Conf. Very Large Data Bases*, Toronto, Canada, 2004, pp.204–215.

[15] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani *et al.* Streaming-data algorithms for high-quality clustering. In *ICDE'02: Proc. the 18th Int. Conf. Data Engineering*, San Jose, California, USA, 2002, p.685.

[16] MacQueen J. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math*, Stat. and Prob, University of California Press, 1967, pp.281–297.

[17] Meyerson A. Online facility location. In *FOCS'01: Proc. the 42nd IEEE Symposium on Foundations of Computer Science*, Las Vegas, Nevada, USA, 2001, p.426.

[18] Haixun Wang, Wei Fan, Philip S Yu, Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *KDD'03: Proc. the Ninth ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Washington DC, 2003, pp.226–235.

[19] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan. Geometric Approximation via Coresets. http://valis.cs.uiuc.edu/∼sariel/papers/04/survey/

[20] Nam Hun Park, Won Suk Lee. Statistical grid-based clustering over data streams. *SIGMOD Record,* 2004, 33(1): 32–37.

[21] Brian Babcock, Mayur Datar, Rajeev Motwani, Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS'03: Proc. the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2003, San Diego, CA, USA, pp.234–243.

[22] Piotr Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *FOCS'99: Proc. the 40th Annual Symposium on Foundations of Computer Science*, New York, NY, USA, 1999, p.154.

[23] Mettu R R, Plaxton C G. The online median problem. In *FOCS'00: Proc. the 41st Annual Symposium on Foundations of Computer Science*, Redondo Beach, CA, USA, 2000, p.339.

[24] Alexander Hinneburg, Daniel A Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in

high-dimensional clustering. In *VLDB'99: Proc. 25th Int. Conf. Very Large Data Bases*, Edinburgh, Scotland, UK, 1999, pp.506–517,

[25] Jain A K, Murty M N, Flynn P J. Data clustering: A review. *ACM Comput. Surv.,* 1999, 31(3): 264–323.

[26] Pavel Berkhin. Survey of clustering data mining techniques. Technical Report, Accrue Software, San Jose, CA, 2002. http://citeseer.nj.nec.com/berkhin02survey.html

[27] Mohanmed Medhat Gaber, Arkady Zaslavsky, Shonali Krishnaswamy. Mining data streams: A review. *SIGMOD Record,* 2005, 34(2): 18–26.

Zhi-Hong Chong received his B.S. degree in computer science from Nanjing Meteorological Institute and M.S. degrees in economics from Institute of Fiscal Studies, Ministry of Finance, P.R. China, in 1991, 1999, respectively. He is currently a Ph.D. candidate in Fudan University, China. His research interests cover data mining, data streams. He has published several research papers in these areas in major international conferences and reputable journals.

Jeffrey Xu Yu received his B.E., M.E. and Ph.D. degrees in computer science, from the University of Tsukuba, Japan, in 1985, 1987 and 1990, respectively. Jeffrey Xu Yu was a research fellow (Apr. 1990–Mar. 1991) and a faculty member (Apr. 1991–July 1992) in the Institute of Information Sciences and Electronics, University of Tsukuba. From July 1992 to June 2000, he was a lecturer in the Department of Computer Science, The Australian National University. Currently, he is an associate professor in the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. Jeffrey Xu Yu is a member of ACM, and a member of IEEE Computer Society. His interests cover XML database, data mining, data warehouse and on-line analytical processing, web-technology, query processing and query optimization, bioinformatics, wireless information systems, design and implementation of database management systems.

Zhen-Jie Zhang received his B.S. degree from Department of Computer Science and Engineering, Fudan University in 2004. Currently he is a Ph.D. candidate in the School of Computing, National University of Singapore. His current research interests include skyline query, clustering and ranking database.

Xue-Min Lin is an associate professor (reader) in the School of Computer Science and Engineering, the University of New South Wales. Currently, he is the head of database research group. Dr. Lin received his Ph.D. degree in computer science from the University of Queensland (Australia) in 1992 and his B.Sc. degree in applied math from Fudan University (China) in 1984. During 1984–1988, he studied for Ph.D. in applied math at Fudan University. Dr Lin's principal research areas cover databases and graph visualisation.

Wei Wang is currently a lecturer in the School of Computer Science and Engineering, The University of New South Wales, Australia. His current research interests include query processing and optimization for XML, integration of database and information retrieval technologies, data warehousing and OLAP, approximate query processing and data mining. He has published over twenty research papers in these areas in major international conferences. He received his Ph.D. degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, China, in 2004, and B.Eng. degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 1999.

Ao-Ying Zhou received his B.Sc. and M.Sc. degrees from Computer Department of Sichuan University in 1985 and 1988, Ph.D. degree in computer software from Fudan University in 1993. From 1996 to 1999 he was appointed the vice director of Computer Science Department of Fudan University and the director from 1999 to 2002. Prof. Zhou is engaged in research on data mining and business intelligence, web data management and peer-to-peer computing.