Statistical Grid-based Clustering over Data Streams

Nam Hun Park Won Suk Lee
Department of Computer Science, Yonsei University
134 Shinchon-dong Seodaemun-gu Seoul, 120-749, Korea
+82-2-2123-2716
{ zyonix, leewo }@amadeus.yonsei.ac.kr

Abstract
A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to this reason, most algorithms for data streams sacrifice the correctness of their results for fast processing time. The processing time is greatly influenced by the amount of information that should be maintained. This paper proposes a statistical grid-based approach to clustering data elements of a data stream. Initially, the multi-dimensional data space of a data stream is partitioned into a set of mutually exclusive equal-size initial cells. When the support of a cell becomes high enough, the cell is dynamically divided into two mutually exclusive intermediate cells based on its distribution statistics. Three different ways of partitioning a dense cell are introduced. Eventually, a dense region of each initial cell is recursively partitioned until it becomes the smallest cell called a *unit cell*. A cluster of a data stream is a group of adjacent dense unit cells. In order to minimize the number of cells, a sparse intermediate or unit cell is pruned if its support becomes much less than a *minimum support*. Furthermore, in order to confine the usage of memory space, the size of a unit cell is dynamically minimized such that the result of clustering becomes as accurate as possible. The proposed algorithm is analyzed by a series of experiments to identify its various characteristics.

1. Introduction
Recently, several data mining methods[1,2] for a data stream are actively introduced. Researches on a data stream are motivated by emerging applications involving massive data sets such as customer click streams, telephone records, multimedia data, and sets of retail chain transactions can be modeled as data streams. Accordingly, a data stream is defined as a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to this reason, it is impossible to maintain all elements of a data stream. Consequently, data stream processing should satisfy the following requirements[3]. First, each data element should be examined at most once to analyze a data stream. Second, memory usage for data stream analysis should be confined finitely although new data elements are continuously generated in a data stream. Third, newly generated data elements should be processed as fast as possible to produce the up-to-date analysis result of a data stream, so that it can be instantly utilized upon request. To satisfy these requirements, data stream processing sacrifices the correctness of its analysis result by allowing some errors.

Clustering is the process of finding groups of similar data elements which are defined by a given similarity measure. Most conventional clustering algorithms[4,5,6] assume a data set is fixed and focus on how to minimize processing time or memory usage algorithmically. Clustering has been widely studied across several disciplines but only a few of its techniques can effectively support a very large data set. When a data set is enlarged incrementally, it is more efficient to use one of incremental clustering algorithms [7]. They mainly focus on how to utilize the previously identified clusters of data elements in a data set in finding the up-to-date clusters of the data set including a set of newly added data elements efficiently such that only necessary data elements in the previous data set are examined. However, all the data elements of the previous data set should be maintained physically since all of them can be potentially examined in the future. In [8], a partitioning clustering algorithm for a data stream is proposed. It uses an O(1)-approximate k-medoid method for each sub-set of a data stream. In order to overcome the iterative evaluation of the conventional k-medoid algorithm[4], its objective is to maintain only the consistently good set of *k* approximate data elements ,i.e., medoids each of which represents the center of a cluster for the data elements observed so far in a data stream.

This paper proposes a grid-based clustering algorithm for clustering the data elements of a data stream. To find clusters of similar data elements over a data stream, the distribution statistics of data elements in the data space of a data stream are carefully maintained. By keeping only the distribution statistics of data elements in a dynamically partitioned grid-cell, the clusters of a data stream can be effectively found without maintaining the data elements physically. Initially, the multi-dimensional data space of a data stream is partitioned into a set of mutually exclusive equal-size initial cells. As a new data element is generated continuously, each initial cell monitors the distribution statistics of data elements within its range. When the support of an initial cell becomes high enough, one of the dimensions of the data space is chosen as a dividing dimension based on the distribution statistics of data elements in the cell. The range of the cell is dynamically divided into two mutually exclusive smaller cells, called *intermediate cells*, with respect to the selected dividing dimension. In addition, the distribution statistics of the cell are used to estimate those of each divided cell. Similarly, when an intermediate cell itself becomes dense, it is partitioned by the same way. Eventually, a dense region of each initial cell is recursively partitioned until it becomes the smallest cell called *a unit cell*.

The range of a dense cell can be partitioned by three different methods: *μ-partition*, *σ-partition* and *hybrid-partition*. The μ-partition method divides it based on the average value μ of the data elements of the cell in its dividing dimension while the σ-partition method divides it based on the standard deviation σ of the data elements of the cell in its dividing dimension. The hybrid-partition method chooses the more effective one of the two partition methods. A cluster of a data stream is a group of adjacent dense unit cells. As the size of a unit cell is set to be smaller, the resulting set of clusters is more accurately identified while the number of cells is increased. In order to minimize the number of cells, a sparse intermediate or unit cell is pruned if its support becomes low enough. Furthermore, the size of a unit cell is dynamically adjusted to confine the usage of memory space.

The rest of this paper is organized as follows: Section 2 presents related works. In Section 3, a statistical grid-based clustering algorithm is proposed. In Section 4, several experimental results are comparatively analyzed to illustrate the various characteristics of the proposed algorithm. Finally, Section 5 presents conclusions.

2. Statistical grid-based clustering Algorithm

2.1 Grid Cells and Distribution Statistics

Given a data stream $\boldsymbol{D}$ of d-dimensional data space $\boldsymbol{N}=N_1\times N_2\times \ldots \times N_d$, a data element generated at the j^{th} turn is denoted by $\boldsymbol{e}^j=<e_1^j,e_2^j,\ldots,e_d^j>$, $e_i^j\in N_i$, $1\le i\le$ d. When a new data element $\boldsymbol{e}^t$ is generated at the t^{th} turn in a data stream $\boldsymbol{D}$, all the data elements that have ever been generated so far are denoted by the current data stream $\boldsymbol{D}^t=\{\boldsymbol{e}^1,\boldsymbol{e}^2,\ldots,\boldsymbol{e}^t\}$. The total number of data elements generated in the current data stream $\boldsymbol{D}^t$ is denoted by $|\boldsymbol{D}^t|$. Finding a cluster of similar data elements in the current data stream $\boldsymbol{D}^t$ is identifying a region whose current density of data elements is high enough. In order to define the similarity between two data elements, a cell whose length in each dimension is less than a predefined distance value λ is defined as a unit cell. The current support of a cell is the ratio of the number of those data elements that are inside the range of the cell over the total number of data elements in $\boldsymbol{D}^t$. Therefore, a cluster at $\boldsymbol{D}^t$ is a group of adjacent dense unit cells whose current supports are greater than or equal to a predefined minimum support S_{min} respectively.

The range of each dimension $\boldsymbol{N}_i$ is initially partitioned by p number of mutually exclusive equal-size intervals $I_i^j=[s_i^j, f_i^j)$ $1\le j\le p$ where s_i^j and f_i^j denote the start and end values in the j^{th} interval of the i^{th} dimension. Consequently, p^d number of *initial cells* are formed in $\boldsymbol{N}$ and each initial cell g is defined by a set of d intervals $\{I_1,I_2,\ldots,I_d\}$ $I_i\subseteq \mathrm{N}_i$ $1\le i\le$ d. The initial range $R(g)$ of an initial cell g is a rectangular space $\boldsymbol{rs}=I_1\times\ldots\times I_d$. However, the rectangular space of an initial cell becomes a set of rectangular spaces $\boldsymbol{RS}=\{\boldsymbol{rs}_1,\boldsymbol{rs}_2,\ldots,\boldsymbol{rs}_q\}$ as a series of cell partitioning is performed subsequently. When the rectangular spaces of a cell g are projected to the i^{th} dimension, the intervals of the i^{th} dimension of the cell can be found and they are denoted by $IS_i(g)=\{I_i^1,I_i^2,\ldots,I_i^q\}$. The sum of these intervals is defined as the *interval size* of the i^{th} dimension of the cell g. The range of the cell g is the united spaces of all the rectangular spaces $\boldsymbol{rs}_1,\ldots,\boldsymbol{rs}_q$, i.e., $R(g)=\bigcup_{i=1}^{q} rs_i$. Each cell keeps the current distribution statistics of those data elements that are within its range as defined in Definition 1.

Definition 1. Distribution Statistics of a grid-cell $g(\boldsymbol{RS},c,\boldsymbol{\mu},\boldsymbol{\sigma})$
For the current data stream $\boldsymbol{D}^t$, a term $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$ is used to denote the distribution statistics of a cell g whose range is defined by a set of rectangular spaces $\boldsymbol{RS}$. Let $\boldsymbol{D}_g^t$ denote those data elements that are in the range of the cell g, i.e.,$\boldsymbol{D}_g^t=\{\ \boldsymbol{e}|\ \boldsymbol{e}\in\boldsymbol{D}^t$ and $\boldsymbol{e}\ \in \mathrm{R}(g)\ \}$. The distribution statistics of the cell g in $\boldsymbol{D}^t$ are defined as follows:

i) c^t : the number of data elements in $\boldsymbol{D}_g^t$

ii) $\boldsymbol{\mu}^t=<\mu_1^t,\ldots,\mu_d^t>$: μ_i^t denotes the average of the i^{th} dimensional values of the data elements in $\boldsymbol{D}_g^t$.

$$\mu_i^t=\sum_{j=1}^{c^t} e_i^j / c^t\ ,\ 1\le i\le \mathrm{d}$$

iii) $\boldsymbol{\sigma}^t=<\sigma_1^t,\ldots,\sigma_d^t>$: σ_i^t denotes the standard deviation of the i^{th} dimensional values of the data elements in $\boldsymbol{D}_g^t$.

$$\sigma_i^t=\sqrt{\sum_{j=i}^{c^t}(e_i^j-\mu_i^t)^2/c^t}\ ,\ 1\le i\le \mathrm{d} \qquad \square$$

When a new data element $\boldsymbol{e}^t$ is generated in the current data stream $\boldsymbol{D}^t$, its corresponding initial cell g among the p^d initial cells is identified based on the initial partitions of the data space $\boldsymbol{N}$. If the distribution statistics of the cell g was updated most recently at the insertion of the v^{th} data element ($v\le t$), its statistics remain the same as $g(\boldsymbol{RS}, c^v,\boldsymbol{\mu}^v,\boldsymbol{\sigma}^v)$ and they are updated to $g(\boldsymbol{RS}, c^t,\boldsymbol{\mu}^t,\boldsymbol{\sigma}^t)$ as follow:

$c^t=c^v+1$,

for $\forall\, i$, $1\le i\le$ d,

$$\mu_i^t=\frac{\mu_i^v\times c^v+e_i^t}{c^t}\ ,\ \sigma_i^t=\sqrt{\frac{c^v}{c^t}\times(\sigma_i^v)^2+\frac{(\mu_i^v)^2+(e_i^t)^2}{c^t}-(\mu_i^t)^2}$$

For the current data stream $\boldsymbol{D}^t$, the current support of an initial cell $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$ is defined by the ratio of its count over the total number of data elements generated so far ,i.e. $c^t/|\boldsymbol{D}^t|$. When the current support of the cell becomes greater than or equal to a *predefined split support* $S_{split}(S_{split}<S_{min})$, two intermediate cells g_1 and g_2 are created as the children of the initial cell. The ranges and distribution statistics of these intermediate cells are determined by a partition method that is used to divide the initial cell. In Section 3.2, three different partition methods are presented in detail.

In case an initial cell corresponding to a newly generated data element is already partitioned, among the children cells of the initial cell, the one whose range includes the new data element e^t is searched. After the target cell g is found, its distribution statistics are updated by the same

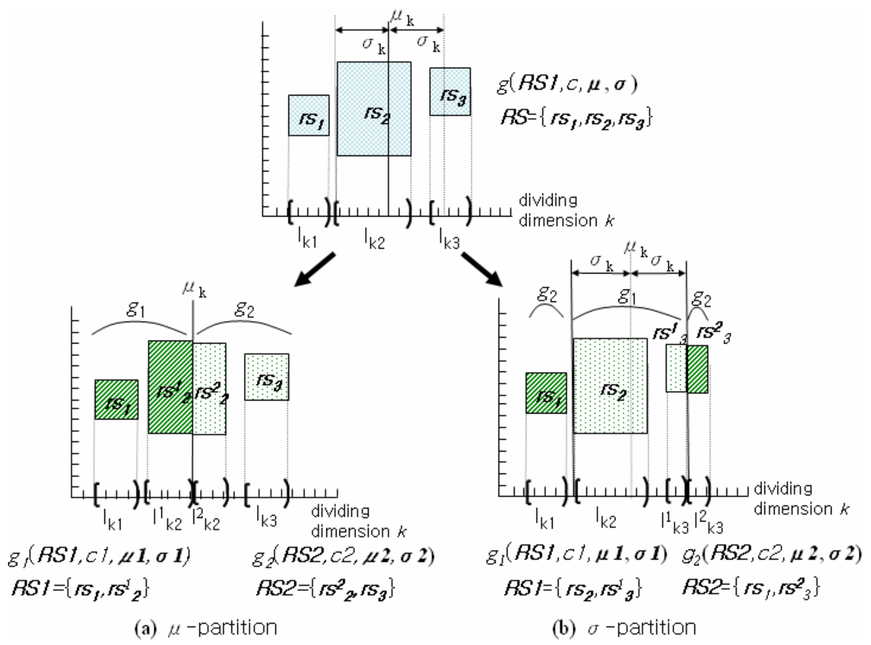


Fig. 1. μ-partition and σ-partition methods

way as described for an initial cell. If the cell g is an intermediate cell and its updated support becomes greater than or equal to S_{split}, the cell g is divided into two smaller cells. However, unlike an initial cell, the intermediate cell g is replaced by the two divided cells. Consequently, the parent initial cell of the intermediate cell g becomes the parent of each divided cell.

2.2 Cell Partition methods

There are three different ways of partitioning a dense initial or intermediate cell, namely *μ-partition*, *σ-partition* and *hybrid-partition*. Among the dimensions of the data space $\boldsymbol{N}$ for a data stream, a dividing dimension is chosen based on the distribution statistics of a dense cell to be partitioned. Subsequently, the range of the cell in the multidimensional data space $\boldsymbol{N}$ is split into two sub-ranges with respect to the selected dividing dimension.

When a dense cell g is split into two cells g_1 and g_2 at the current data stream $\boldsymbol{D}^t$ by the μ-partition method, among the dimensions whose interval sizes are larger than λ, the one with the largest standard deviation, say $\sigma_k^t (1 \leq k \leq d)$, is chosen as a dividing dimension. The standard deviation of each dimension in the distribution statistics of a cell represents how the data elements of the cell are distributed with respect to the dimension. When the standard deviation of a dimension is large, the corresponding dimensional values of most data elements in the cell are far from the average of the dimension. Therefore, the dividing dimension selected by the μ-partition method is the one in which the data elements can be evenly partitioned into two groups as clearly as possible. The rectangular spaces of the cell g are split into two mutually exclusive sets with respect to the average μ_k^t of the dividing dimension k. One contains those rectangular spaces whose intervals in the dividing dimension are less than μ_k^t and the other contains those rectangular spaces whose intervals are greater than or equal to μ_k^t. If an interval of the dividing dimension k of the cell g includes μ_k^t, the interval is actually divided. Figure 1-(a) illustrates how to split the rectangular spaces of a dense cell $g(\boldsymbol{RS},c,\boldsymbol{\mu},\boldsymbol{\sigma})$ in a two-dimensional data space by the μ-partition method. The two sets of the rectangular spaces are assigned to the ranges of the two divided cells g_1 and g_2 respectively.

By assuming the distribution of data elements in a dense cell g is a normal distribution, the distribution statistics of its two divided cells $g_1(\boldsymbol{RS1}, c1^t, \boldsymbol{\mu 1}^t, \boldsymbol{\sigma 1}^t)$ and $g_2(\boldsymbol{RS2}, c2^t, \boldsymbol{\mu 2}^t, \boldsymbol{\sigma 2}^t)$ in the current data stream $\boldsymbol{D}^t$ are initialized. The count of the cell g is evenly distributed into the two divided cells g_1 and g_2 i.e., $c1^t = c2^t = c^t/2$. Let $\varphi(x) = \frac{1}{\sqrt{2\pi}\sigma^t} e^{-\frac{(x-\mu^t)^2}{2(\sigma^t)^2}}$ be the normal distribution function of the data elements in the range of the cell g. The remaining distribution statistics of the two divided cells can be estimated respectively as follows.

$\boldsymbol{\mu 1}^t = \boldsymbol{\mu 2}^t = \boldsymbol{\mu}^t$ except $\mu 1_k^t$ and $\mu 2_k^t$,

$$\mu 1_k^t = \int_{s_k(g_1)}^{f_k(g_1)} x\varphi(x)dx \quad \mu 2_k^t = \int_{s_k(g_2)}^{f_k(g_2)} x\varphi(x)dx$$

$\boldsymbol{\sigma 1}^t = \boldsymbol{\sigma 2}^t = \boldsymbol{\sigma}^t$ except $\sigma 1_k^t$ and $\sigma 2_k^t$,

$$\sigma 1_k^t = \sqrt{\int_{s_k(g_1)}^{f_k(g_1)} x^2\varphi(x)dx - \left(\mu 1_k^t\right)^2} \quad \sigma 2_k^t = \sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2\varphi(x)dx - \left(\mu 2_k^t\right)^2}$$

where $s_k(g_w)$ and $f_k(g_w)$ denote the smallest start and largest end value of the intervals of the dividing dimension k for the divided cell g_w, w=1 or 2. If the dense cell g is an initial cell, its distribution statistics are cleared as c^t=0 and μ_i^t=σ_i^t=0 for $\forall i$, 1≤ i≤ d since they are carried to those of g_1 and g_2.

In the σ-partition method, among the dimensions whose interval sizes of a cell g to be partitioned are larger than λ, the one with the smallest standard deviation, say $\sigma_l^t (1 \leq l \leq d)$, is chosen as a dividing dimension. When the standard deviation of a dimension in the distribution statistics of a cell is small, the corresponding dimensional values of most data elements in the cell are adjacent to the average of the dimension. Since the one with the smallest standard deviation is selected as a dividing dimension, the data elements of the cell are most closely distributed to the average of the dividing dimension. Based on the standard deviation σ_l^t of the data elements in the selected dividing dimension l, the rectangular spaces of the cell are split into two mutually exclusive sets. One contains those rectangular spaces whose intervals in the dividing dimension are within $[\mu_l\text{-}\sigma_l, \mu_l+\sigma_l)$. In this set, the 68 percentage of data elements in the cell g is assumed to be distributed according to a normal distribution. The other contains the remaining rectangular spaces. As in the μ-partition method, if an interval of the dividing dimension l includes either $\mu_l\text{-}\sigma_l$ or $\mu_l+\sigma_l$, the interval is actually divided. As a result, the σ-partition method makes the range of one of the two divided cells be as small as possible. In other words, it makes one of them be as dense as possible. Figure 1-(b) illustrates how to divide the rectangular spaces of a dense cell g ($\boldsymbol{RS}$,c,$\boldsymbol{\mu}$,$\boldsymbol{\sigma}$) in a two-dimensional data space by the σ-partition method.

When a cell $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$ is partitioned by the σ-partition method into two cells $g_1(\boldsymbol{RS_1}, c1^t, \boldsymbol{\mu 1}^t, \boldsymbol{\sigma 1}^t)$ and $g_2(\boldsymbol{RS_2}, c2^t, \boldsymbol{\mu 2}^t, \boldsymbol{\sigma 2}^t)$, the counts of these cells g_1 and g_2 are initialized respectively by the normal distribution function $\varphi(x)$ of the cell g as follows:

$$c1^t = c^t \times \int_{\mu_k^t - \sigma_k^t}^{\mu_k^t + \sigma_k^t} \varphi(x)dx, \quad c2^t = c^t - c1^t$$

The distribution statistics $\mu 1_k^t$, $\mu 2_k^t$, $\sigma 1_k^t$ and $\sigma 2_k^t$ of the dividing dimension k for the divided cells are estimated similarly as follows:

$$\mu 1_k^t = \int_{s_k(g_1)}^{f_k(g_1)} x\varphi(x)dx \quad \text{and} \quad \sigma 1_k^t = \sqrt{\int_{s_k(g_1)}^{f_k(g_1)} x^2 \varphi(x)dx - \left(\mu 1_k^t\right)^2}$$

if $s_k(g_2) < \mu 1_k^t < f_k(g_2)$,

$$\mu 2_k^t = \int_{s_k(g_2)}^{f_k(g_2)} x\varphi(x)dx - \mu 1_k^t, \text{ and}$$

$$\sigma 2_k^t = \sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2 \varphi(x)dx - \int_{s_k(g_1)}^{f_k(g_1)} x^2 \varphi(x)dx - \left(\mu 2_k^t\right)^2}$$

$$\text{else} \quad \mu 2_k^t = \int_{s_k(g_2)}^{f_k(g_2)} x\varphi(x)dx \quad \text{and}$$

$$\sigma 2_k^t = \sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2 \varphi(x)dx - \left(\mu 2_k^t\right)^2} \; .$$

The distribution statistics of the other dimensions remain the same as in the μ-partition method.

The hybrid-partition method selects the more effective method of the two partition methods whenever a specific dense cell needs to be divided. Creating unit cells quickly is an important property to improve the accuracy of the proposed algorithm. It can be accomplished by minimizing the overall number of cell partition steps to produce a unit cell. Applying the more effective one of the two methods for a specific dense cell can reduce the overall number of cell partition steps. Let $\sigma^e{}_i$ denote the standard deviation of the i^{th} dimensional values of data elements in a cell g when the data elements are uniformly distributed over the intervals of the i^{th} dimension of the cell. The *relative effectiveness rate* $\beta_k(g)$ of dividing a dense cell g by a partition method with a dividing dimension k is defined as follows:

$$\beta_k(g) = |\sigma^e{}_k - \sigma^t{}_k|, \quad \text{where} \quad \sigma^e{}_k = \sqrt{\frac{1}{f_k(g) - s_k(g)} \int_{s_k(g)}^{f_k(g)} x^2 dx - (\mu_k^t)^2}$$

It is the absolute difference of the two standard deviations σ_k^t and σ_k^e. It can be used to measure the relative congestion rate of data elements in the cell g. Given a dense cell g to be partitioned, let $k1$ denote the dividing dimension selected by the μ-partition method and let $k2$ denote the dividing dimension selected by the σ-partition method. In other words, the standard deviation of the dimension $k1$ is the largest and that of the dimension $k2$ is the smallest. Since both effectiveness rates $\beta_{k1}(g)$ and $\beta_{k2}(g)$ are measured relatively to σ_k^e, the partition method with the larger rate divides the cell g more effectively. If $\beta_{k1}(g) > \beta_{k2}(g)$, the congestion rate of data elements in the cell g on their average μ_{k1}^t in the dimension $k1$ is larger than the evenly separated rate of the data elements from their average μ_{k2}^t in the dimension $k2$. As a result, the cell g is partitioned in terms of $k1$ by the μ-partition method. On the other hand, if $\beta_{k1}(g) < \beta_{k2}(g)$, the σ-partition method is applied to the dividing dimension $k2$. If $\beta_{k1}(g) = \beta_{k2}(g)$, either of the two partition methods can be applied. By selecting an appropriate partition method, the number of cell partition steps as well as the number of cells can be minimized.

2.3 Cell Pruning

When the current support of an intermediate or unit cell g becomes less than a *predefined pruning support* S_{prn} ,i.e., $c^t/|\boldsymbol{D}^t| \leq S_{prn}$, the probability of finding a cluster in the range of the cell in the near future is very low. Consequently, the cell is removed and its distribution statistics $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$ are returned back to its parent initial cell g_p. Suppose the distribution statistics of the parent cell g_p were updated lastly at the v^{th} element($v \leq t$) and they are denoted by $g_p(\boldsymbol{RS_p}, cp^v, \boldsymbol{\mu p}^v, \boldsymbol{\sigma p}^v)$ where $\boldsymbol{\mu p}^v = <\mu p_1^v, \mu p_2^v, \ldots, \mu p_d^v>$ and $\boldsymbol{\sigma p}^v = <\sigma p_1^v, \sigma p_2^v, \ldots, \sigma p_d^v>$. After the cell g is pruned, the statistics $g_p(\boldsymbol{RS_p}, cp^t, \boldsymbol{\mu p}^t, \boldsymbol{\sigma p}^t)$ of the parent cell at $\boldsymbol{D}^t$ are updated as follows:

$cp^t = cp^v + c^t$,

$$\mu p_i^t = \frac{\mu p_i^v \times c^v + \mu_i^t \times c^t}{cp^t} \quad \text{and for all dimensions } i \ (1 \le i \le \mathrm{d})$$

$$\sigma p_i^t = \sqrt{\frac{cp^v \times (\sigma p_i^v)^2 + c^t \times (\sigma_i^t)^2}{cp^t} + \frac{(\mu p_i^v)^2 + (\mu_i^t)^2}{cp^t} - (\mu p_i^t)^2}$$

A sparse intermediate or unit cell can be pruned whenever a data element in the range of the cell is newly generated. However, a considerable number of such sparse cells may not be pruned since the possibility of encountering a data element in the range of a sparse cell is very low. All sparse intermediate or unit cells can be forced to be pruned together by examining their current supports. This mechanism is called as a *force-pruning operation*. Since the distribution statistics of all intermediate or unit cells should be examined, the processing time of a force-pruning operation takes relatively long. Due to this reason, it can be performed periodically or when the current usage of memory space reaches a predefined threshold value.

Since available memory space is confined, the memory usage of the proposed algorithm is adjusted adaptively by resizing the size λ of a unit cell dynamically. Given the value of λ, if the confined memory space is full, all unit cells are pruned and their distribution statistics are added to their parent initial cells respectively. Similarly, those intermediate cells whose interval size in at least one dimension is less than 2λ are pruned by the same way. Subsequently, the value of λ is doubled and the normal operations of the proposed algorithm are resumed. Since the value of λ is doubled, the memory requirement of the proposed method is reduced while its clustering accuracy is degraded. On the other hand, if the amount of unused memory space stays to be larger than the two-fold of the total size of all unit cells, the value of λ is dynamically adjusted to be the half of its value.

3. Experimental Results

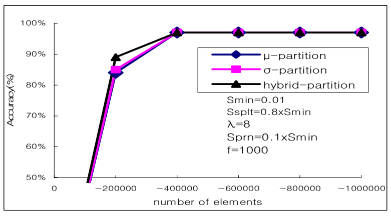
In order to analyze the performance of the proposed algorithm, a data set containing one million 10-dimensional data elements is generated by the data generator used in ENCLUS [9]. Most of data elements are concentrated on randomly chosen 100 distinct data regions whose sizes in each dimension are also randomly varied from 10 to 20 respectively. Although the proposed algorithm is based on grid-based clustering, the accuracy of the proposed algorithm is compared with that of DBSCAN since, unlike conventional grid-based approaches, a grid cell is dynamically created. The values of a pruning support S_{prn} and a split support S_{splt} are assigned relatively to a predefined minimum support S_{min}. The multidimensional data space of the data set is divided into 4 initial cells. In all experiments, data elements are looked up one by one in sequence to simulate the environment of a data stream.

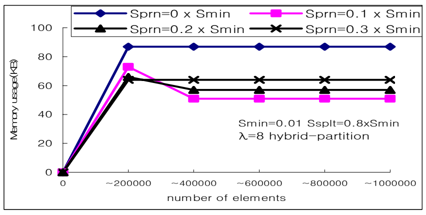
Figure 2 shows the variation of clustering accuracy by each partition method. The accuracy of the proposed algorithm is measured by the ratio of the number of correctly clustered elements by a specific partition method of the proposed algorithm over the total number of data elements clustered by DBSCAN. In DBSCAN, the value of *MinPts* is set to $S_{min} \times |\boldsymbol{D}^t|$ and the value of *Eps* is set to the half of λ. The sequence of generated data elements is divided into 5 intervals each of which consists of 200,000 elements. The average accuracy of each interval is shown in this figure. The force-pruning operation is performed whenever 1,000 new elements are processed(f=1000). The clustering accuracy of the first interval is relatively lower than those of the other intervals. This is because the support of each intermediate cell is too sensitively varied in the first interval. As a result, a lot of cell partition operations are performed in the first interval to produce a set of meaningful unit cells. After the first interval, the accuracy of the proposed algorithm is stabilized regardless of which partition method is used. The hybrid-partition method provides the most accurate result in the first interval. This is because it applies the most appropriate partition method to divide a dense cell. Consequently, meaningful dense unit cells are generated quickly via less number of cell partition operations.

Figure 3 shows the variation of memory usage in this experiment. After most of dense unit cells are generated, i.e. the proposed algorithm is stabilized, its memory usage can also be decreased by setting the value of S_{prn} adequately. When S_{prn} is set to $0.3 \times S_{min}$, the memory usage is the largest. The reason is that most of divided intermediate cells are pruned too quickly and their initial cells are repeatedly partitioned again. On the contrary, when the value of S_{prn} is set to $0.1 \times S_{min}$, the memory usage is minimized since dense intermediate cells are successfully divided into dense unit cells while sparse ones are pruned properly. The pruning support can be effectively used to minimize the usage of memory space with small loss of accuracy.

Figure 4 shows the variation of clustering accuracy by varying the value of λ. The predefined size λ of a unit cell determines the resolution of clustering. As λ is set to be smaller, the boundary of a cluster is more precisely identified. By varying the value of λ, Figure 5 shows the variation of memory usage. As the value of λ is increased, the memory usage is decreased enormously.

Figure 6 shows the memory space adaptability of the proposed algorithm when available memory space is confined to 30KB. Two different schemes of the proposed algorithm, namely *adjusted* and *fixed* schemes, are compared. The value of λ is dynamically adjusted in the adjusted scheme while it is statically set in the fixed scheme. All the previous experiments are based on the fixed scheme. Initially, the value of λ is set to 16. In the first interval, the two schemes use the same amount of memory space. However, the memory usage of the fixed scheme is dropped rapidly in the second interval since lots of sparse cells are pruned. It is stabilized in the third interval. On the contrary, in the second interval of the adjusted scheme, the value of λ is adjusted to its half, i.e., λ=8 since a considerable amount





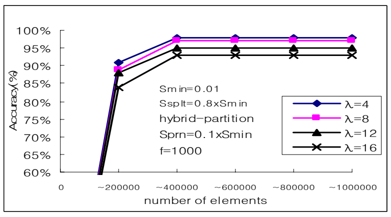
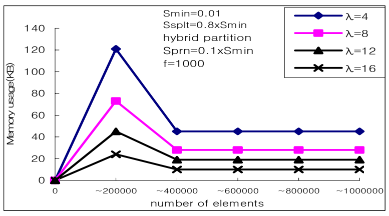
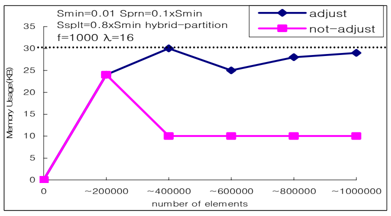


Fig. 2. Clustering accuracy **Fig. 3.** Memory usage by varying S_{prn} **Fig. 4.** Clustering accuracy by varying λ





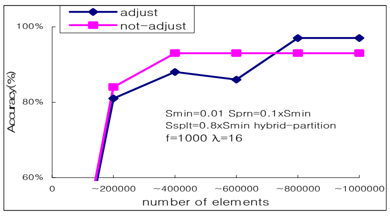


Fig. 5. Memory usage by varying λ **Fig. 6.** Memory space adaptability **Fig. 7.** Adjusted clustering accuracy

of unused memory space is left. Subsequently, the memory usage of the adjusted scheme is increased in the second interval. When the memory usage reaches the limit in the third interval, the value of λ is doubled ,i.e., λ=16. After sparse cells are pruned in the third interval, its value is readjusted to be the half again, i.e., λ=8. Ultimately, the memory usage of the adjusted scheme is stabilized to the confined memory space. Figure 7 shows the clustering accuracy of this experiment. Before the memory usage of the adjusted scheme is stabilized, its accuracy is lower than that of the fixed scheme. This is because all unit cells and some of intermediate cells are compulsorily pruned when the value of λ is dynamically increased.

4. Conclusion

In this paper, a grid-based statistical clustering algorithm for a data stream is proposed. The multi-dimensional data space of a data stream is dynamically divided into a set of cells with different sizes. By maintaining only the distribution statistics of data elements in each cell, its current support is precisely monitored. A dense region of a data space is partitioned repeatedly until it becomes a dense unit cell. Three different partition methods are proposed in this paper. The μ-partition and σ-partition methods are intended to split every cell in a fixed way while the hybrid-partition method is designed to use the more effective one of the two methods for the distribution statistics of a specific dense cell. Two thresholds S_{splt} and S_{prn} are proposed to control the performance of the proposed algorithm in a data stream. A split support S_{splt} is used to determine how fast dense unit cells are created. A pruning support S_{prn} is used to remove meaningless sparse intermediate or unit cells. Furthermore, in order to confine the memory usage of the proposed algorithm, the size of a unit cell can be dynamically adjusted based on the current usage of memory space. As a result, it is possible to maximize the accuracy of clustering for the available amount of memory space.

References

1. M. Datar, A. Gionis, P. Indyk and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. Of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002
2. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. Of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.
3. M. Garofalakis, J. Gehrke and R. Rastogi. Querying and mining data streams: you only get one look. In *the tutorial notes of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.
4. L. Kaufman and P.J. Rousseeuw. Finding Groups in Data. An Introduction to Cluster Analysis. Wiley, New York, 1990.
5. S. Guha, R.Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. SIGMOD*, pages 73-84, 1998
6. M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases, 1996.
7. M. Ester, H. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment, In *Proc. VLDB 24th*, New York, 1998
8. Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. STREAM-data algorithms for high-quality clustering. In *Proc. of IEEE International Conference on Data Engineering*, March 2002.
9. Cheng, C., Fu, A., and Zhang, Y. Entropy-based subspace clustering for mining numerical data. KDD-99, 84-93, San Diego, August 1999.