Constrained Clustering for the Evolving Data Stream

Bi-Ru Dai[1,*] and Ming-Syan Chen[2]

[1] Department of Computer Science and Information Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan, ROC

brdai@csie.ntust.edu.tw

[2] Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC

mschen@cc.ee.ntu.edu.tw

Received June 22, 2007 ; Accepted June 30, 2007

Abstract. In order to import the domain knowledge or application dependent parameters into the data mining systems, constraint-based mining has attracted a lot of research attention recently. However, most of the constraint-based mining algorithms are designed for static data sets, and are not investigated in the data stream environment. In this paper, we devise a framework of *Constrained Clustering for the Evolving Data Stream*, abbreviated as *CCDS framework,* to cluster the data stream under the pairwise range constraint. The CCDS framework proposed consists of two phases, namely the *statistics reserving phase* and the *clustering responding phase*. The statistics reserving phase provides an efficient algorithm to process and maintain the data points in a compact structure named *constraint tree.* The clustering responding phase generates clusters whenever a clustering request is submitted by the user. As shown in our analyses, the time complexity of the statistics reserving phase, which is the time complexity of constructing the constraint tree, is linear in the number of data points. In addition, the clustering time is also reduced by applying the statistics maintained to the clustering algorithm. Therefore, framework CCDS is very suitable for the data stream environment. It is empirically shown that the proposed framework is very efficient for dealing with multiple constrained clustering requests in the data stream environment while producing clustering results of very high quality

Keywords: data mining, data clustering, constrained clustering, data stream

1 Introduction

Data clustering is a useful technique for many applications, including similarity search, pattern recognition, trend analysis, marketing analysis, grouping, classification of documents, and so forth [1][2]. In data clustering, similar data points are grouped together in a cluster. In recent years, several mining capabilities have been explored for the data stream environment [3][4], including those on the association rules [5], frequent patterns [6], data clustering [7-9], and data classification [10][11], to name a few. For data stream applications, the volume of data is usually too huge to be stored on permanent devices or to be scanned thoroughly more than once. It is hence recognized that both approximation and adaptivity are key ingredients for executing queries and performing mining tasks over rapid data streams.

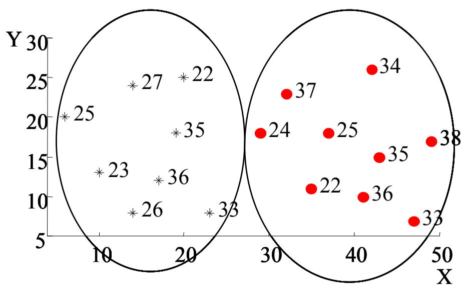
Since data mining is an application dependent technology, the information involving domain knowledge is usually imposed on the mining systems as various constraints. Some algorithms have been proposed for clustering with constraints [12-17]. However, they are mainly designed for static data sets and are usually not able to work well in the data stream environment. On the other hand, the main focus of the research on mining data streams is to solve mining problems under the condition of limited resources, such as time and space, compared to the fast and infinite data points. However, the constraints for clusters or data points are not fully investigated. In this paper, these two concepts, i.e., the properties of constraints and data streams, are combined and consid-

[*] Correspondence author

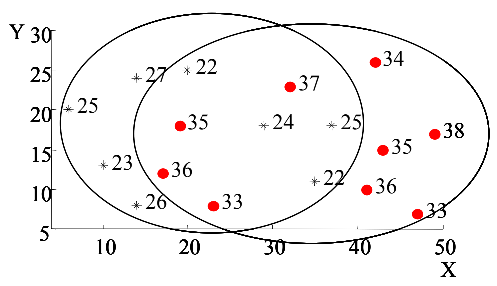
ered at the same time. Note that the effects of these two factors are entangled, thus significantly complicating the problem. We focus our attention on clustering with the pairwise numerical constraint, which was proposed in the work [13], and propose a framework to support this constrained clustering problem in the data stream environment. Specifically, those attributes employed to model the constraints are called constraint attributes whereas those attributes involved in the objective function to be optimized, similar to those in most prior works, are called optimization attributes. The constrained clustering is conducted in such a way that the objective function of optimization attributes is optimized subject to the condition that the imposed constraint by constraint attributes is satisfied. The constraint considered in this work is that the constraint attribute values of any two data items in the same cluster are required to be within the corresponding constraint range.

This pairwise constrained clustering problem can be understood by the following example. Consider the data points in Fig. 1 where X and Y are the conventional optimization attributes which form the coordinate of the residential location and age is the constraint attribute with the constraint range being 5 years. The nodes in Fig. 1(a) are identical to those in Fig. 1(b) and the number next to each node represents the age of that person. By conventional clustering methods which are designed to cluster nearby nodes together, these nodes may be partitioned into the two clusters as shown in Fig. 1(a), in which, however, the age constraint is not obeyed. Instead, one possible solution to this constrained clustering problem is shown in Fig. 1(b), where members within 22 - 27 years old are in one cluster, and people within 33 - 38 years old are in the other cluster. Such a clustering with numerical constraints is called for in many real applications. For example, we may apply this pairwise numerical constrained clustering on the basic data of people in a club to group people provided that we require the range of ages in each group to be no more than 5 years. Notice that we do not indicate a fixed boundary of a cluster explicitly. Instead, the boundaries of constraint attributes of each cluster are dynamically determined by the data points existing in the cluster. In the example in Fig. 1, once a person joins a cluster, the age range of this cluster is adjusted immediately. For example, assume that the first person joining a group is 23 years old, then people between 18 to 28 years old are allowed to join this group afterward. However, if the second person joining this group is 27 years old, the age range of this group is narrowed down to 22 - 28 years old, meaning that the age boundary of this cluster is revised. Finally, groups with similar people are generated based on the 5-year range. Figures 1(c) and 1(d) show the age in the third dimension, and the difference to the result from a conventional clustering method is also shown. Possible applications of this constrained clustering model include bioinformatics data with the control of temperature [18], the group behavior of GPS users along the time line, and the segmentation of a video into story units [19], to name a few. More justification of this problem model can be found in [15][20].

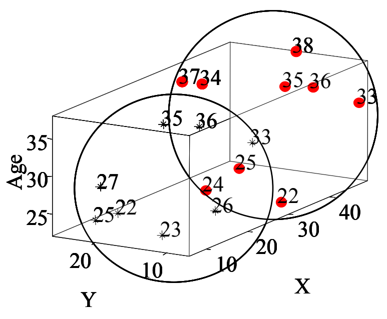
X	Y	Age
6	20	25
10	13	23
14	8	26
14	24	27
17	12	36
19	18	35
20	25	22
23	8	33
29	18	24
32	23	37
35	11	22
37	18	25
41	10	36
42	26	34
43	15	35
47	7	33
49	17	38



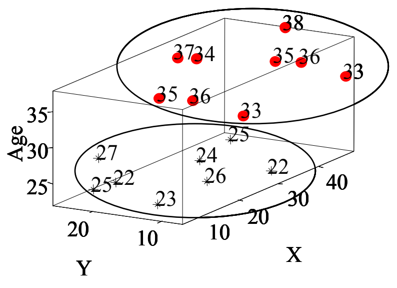
(a) A conventional clustering



(b) A constrained clustering with age constraint range = 5



(c) Three-dimensional graph of (a)



(d) Three-dimensional graph of (b)

Fig. 1. Illustration for the difference between a conventional clustering and a numerical constrained clustering

Note that in the data stream environment, data points arrive continuously. The limited space does not allow us to maintain all the information of data points. Therefore, data points can only be parsed once to collect necessary statistics and then discard afterwards. In addition, to process fast incoming data points, an efficient algorithm of maintaining statistics is also required. Furthermore, along the accumulation of data points, users are possible to have different clustering requirements at different time. For example, when a club has fewer members and fewer resources, 5 clusters with the constraint range being 10 years are good. However, when more and more members join, users may feel that 10 clusters with 5-year range are more appropriate. In addition, clustering requirements are usually unpredictable in advance because they are application dependent and are usually different for various purposes. Therefore, a framework, which is able to handle infinite arriving data points and support various clustering requirements, will be desired.

In this paper, we devise a framework of *Constrained Clustering for the Evolving Data Stream*, abbreviated as *CCDS framework*, to cluster the data stream under the pairwise numerical constraint. The CCDS framework proposed consists of two phases, namely the *statistics reserving phase* and the *clustering responding phase*. The statistics reserving phase provides an efficient algorithm to process and maintain the data points in a compact structure named *constraint tree*. On the other hand, the statistics maintained will be retrieved in the clustering responding phase to generate clusters according to clustering requests specified by users. Since the clustering requirement is obtained only when a user submits the request, the maintained information is required to have the ability to service various clustering requests without process the original data points again. Note that in addition to minimize the clustering costs, the pairwise numerical constraint should be followed strictly. Accordingly, we design the constraint tree structure to provide a higher priority for the constraint attribute when data points are inserted. Then, data points with higher similarity are summarized and maintained in one record to reduce the storage space. As shown in the complexity analyses, the time complexity of the statistics reserving phase, which is the time complexity of constructing the constraint tree, is of $O(n)$ where n is the number of data points. In addition, the clustering time is also reduced by applying the statistics maintained to the clustering algorithm, which ensures the efficient execution of framework CCDS in the data stream environment. Furthermore, theoretical properties of CCDS are derived in this paper. It is also validated by our empirical studies that the CCDS framework performs very efficiently in the data stream environment while producing clustering results of very high quality.

The rest of this paper is organized as follows. The problem description and framework definitions are given in Section 2. Section 3 presents the proposed algorithms to deal with this constrained clustering problem in the data stream environment. Then empirical studies are conducted in Section 4. This paper concludes with Section 5

2 Problem Description

In this section we will describe the problem studied in this paper, and introduce a framework to solve the constrained clustering for the evolving data stream. As mentioned earlier, among all attributes of the data set, some are specified as *constraint attributes* and some are *optimization attributes*. Constraint attributes and optimization attributes may overlap because some attributes are possibly important in both considerations. Similar to the conventional clustering problems, there is an objective function operating on the optimization attributes to measure the cost of clustering. In addition, an additional *constraint range* R_{a_C} is set for the constraint attribute a_C. For any pair of objects o_i, o_j in a cluster, the constraint distance $d_{a_C}(o_i, o_j)$ of constraint attribute a_C between any two objects o_i and o_j is required to be less than or equal to R_{a_C}. For example, in Fig. 1, age is a constraint attribute, denoted by a_{age}, and its constraint range $R_{a_{age}}$ is 5. The constraint distance $d_{a_{age}}(.,.)$, which is the difference of ages, should be no more than 5 years. For the simplification and clearer discussion of this problem, we focus on the scenario of a single constraint attribute in this paper. Formally, we have the following problem definition.

Definition 1 Clustering Data Stream with Pairwise Constraint:

- *A data stream D of infinite incoming objects is represented as* $\{o_1, o_2, ..., o_i, ...\}$.
- *A clustering request is a pair of (k,* R_{a_C} *),where k is the number of clusters,* R_{a_C} *is the constraint range of the constraint attribute* a_C *, and the value of* $R_{a_C} \times k$ *is larger than or equal to the whole range of the constraint attribute* a_C *.*
- *Given a data stream D and a received clustering request of (k,* R_{a_C} *), the pairwise constrained clustering problem is defined as the problem of determining the k-clustering* $Cl = \{C_1, C_2, ..., C_k\}$ *in such a way that the total cost*

Cost(Cl) is minimized subject to the condition that for any pair of objects (o_i,o_j) in a cluster, we have $d_{a_C}(o_i, o_j) \le R_{a_C}$ *. The cost function Cost(Cl) is calculated based on the optimization attributes.*

Example 1: Given a data stream as follows, D ={(9,11), (2,27), (7,3), (10,9), (6,29), (8,23), (4,22), (5,8), (3,6), (1,25), ...}, where the first value of each data point is the value of the constraint attribute, and the second one is the value of the optimization attribute. Suppose that a clustering request of (*k*, R_{a_C}) = (2, 6) is submitted when 8 data points are received. The constrained clustering result will be C_1 = {(2, 27), (6, 29), (8, 23), (4, 22)} and C_2 = {(9, 11), (7, 3), 10, 9), (5, 8)}. Note that the values of the constraint attribute are in the range [2-8] for cluster C_1 and in the range [5-10] for cluster C_2 to comply with the constraint range 6. Then, after two more data points are inserted, another clustering request of (*k*, R_{a_C}) = (3, 5) is submitted. The clustering result of C_1 = {(2, 27), (6, 29), (4, 22), (1, 25)}, C_2 = {(9, 11), (10, 9), (8, 23)}, and C_3 = {(7, 3), (5, 8), (3, 6)} will be responded to the new request. It is noted that some data points are not in the same clusters as the previous clustering result since the constraint range and the number of clusters are changed.

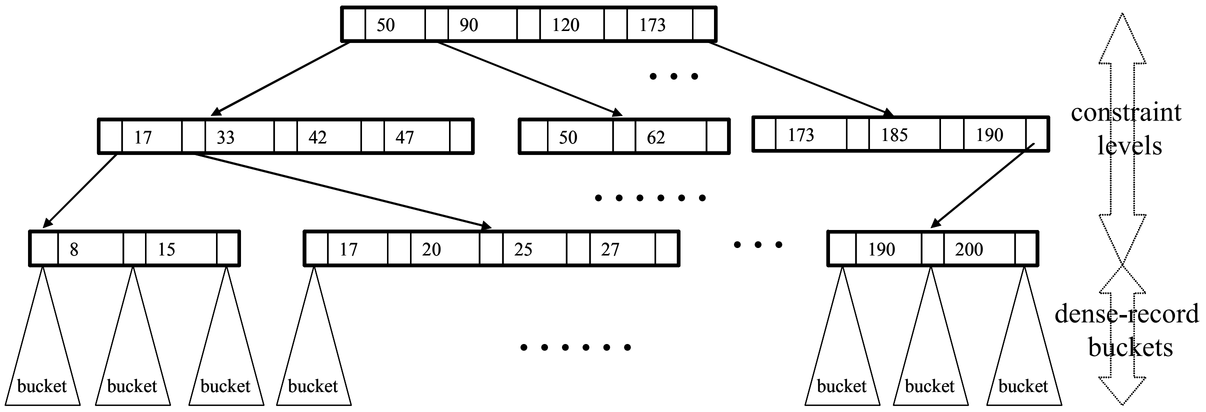


Fig. 2. Illustration of the constraint tree, where the maximum number of keys in a node is 4. Each node contains at least 2 keys and at most 4 keys. The leaf nodes link to dense-record buckets.

Note that in the data stream environment, data points arrive continuously and users can submit clustering request (*k*, R_{a_C}) at any time. According to the above definition, two main tasks are required to be handled in our framework. First, we have to keep track of the incoming data stream, and maintain sufficient information or summaries for generating clusters with high quality. Second, clusters, which are able to satisfy requirements specified by users, should be generated from the available information instead of from the original whole data set. Note that the maintained information should service various clustering requests without process the original data points again. Therefore, a framework named *Constrained Clustering for the Evolving Data Stream*, abbreviated as *CCDS framework*, containing two phases, which are the *statistics reserving phase* and the *clustering responding phase*, is proposed to deal with these two tasks, respectively. In the statistics reserving phase, the arriving data points are processed and some statistics are maintained. Note that clusters are generated whenever a clustering request is submitted by a user; therefore, we have no idea about what kind of clusters will be generated when data points arrive. In other words, the statistics maintained during the statistics reserving phase should contain enough information for currently unknown clustering requests while keeping the required storage space small. In the CCDS framework, a compact data structure, named *constraint tree*, is proposed to maintain the statistics efficiently for future clustering requests. In the clustering responding phase, clustering requests containing the cluster number *k* and the constraint range R_{a_C} are obtained. Then, according to the clustering requests, the clustering algorithm will be applied to the statistics kept in the constraint tree. Therefore, multiple clustering requests are able to be served without visiting the original data set. The details of the constraint tree and the proposed algorithms will be described in Section 3.

3 Design of CCDS Framework

In Section 3.1, the algorithm for maintaining the constraint tree during the statistics reserving phase is presented. Next, the approach to generate clusters in the clustering responding phase is described in Section 3.2.

3.1 Statistics Reserving Phase

Since the data points in the data stream environment arrive continuously, we are not able to store the information of all data points. Therefore, a compact structure is required to keep enough information for clustering requests in the future without incurring massive storage space.

Note that data points considered in this work contain two types of attributes, which are the constraint attribute and the optimization attributes. The constraint attribute sets a restriction on data points to be put in the same cluster. On the other hand, clusters with lower cost, which is calculated by the optimization attributes, are more desirable. Therefore, these two types of attributes should be handled separately in the statistics reserving phase. Accordingly, the constraint tree is designed to incorporate the information of the constraint attribute and the optimization attributes in a compact structure. This constraint tree consists of two parts, which are *constraint levels* and *dense-record buckets*, as shown in Fig. 2. The index keys in the constraint levels are values of the constraint attribute, where as similar data points in the same leaf node are aggregated into one record in the dense-record bucket. When a new arriving data point is inserted into the constraint tree, it traverses through the constraint levels according to the value of constraint attribute first, and then goes to the dense-record bucket afterwards. As such, among all the attributes, the constraint attribute is able to be considered in a higher priority. Then, the optimization attributes are taken into consideration for data points within a smaller range of constraint values.

The purpose of constraint levels is to distribute data points according to the constraint attribute in a more balanced way. However, the limitation of restricted space in the data stream environment usually does not allow us to keep track of every possible value. Therefore, when more and more data points with different constraint values are received, the precise constraint values are not always be inserted as a new index key value of the constraint tree. Instead, some data points will be assigned to an appropriate leaf node directly without creating a new key. The inner nodes of constraint levels are represented as the following format:

$$<child_0, key_1, child_1, key_2, child_2,..., key_m, child_m> ,$$

where $child_i$ is the pointer to a child node, and key_i is a constraint value where $key_i < key_j$ if $i < j$. There are at most m keys and at least $\left\lceil \frac{m}{2} \right\rceil$ keys in each node. The child node linked by $child_i$ contains records with the constraint values in the range of $[key_i , key_{i+1})$ for $1 < i < m$-1. The child node linked by $child_0$ contains records with the constraint values being smaller than key_1, and the child node linked by $child_m$ contains records with the constraint values being larger than or equal to key_m. The format of leaf nodes is the same as inner nodes. Rather than linking to a child node, a pointer in a leaf node links to a dense-record bucket instead. Similar to the insertion of B+-tree, an incoming record traverse the constraint tree to find a suitable leaf node for insertion. If there is still empty space in the node, a new key is inserted. Otherwise, the node needs to be split and the median key is inserted into its parent node. If the parent node is also full, split it and repeat the procedure until no parent node needs to be split.

However, the constraint tree is not allowed to grow infinitely due to the limited space in the data stream environment. Therefore, a predefined parameter H_{max} , which can be specified by users according to the available space, is assigned to be the maximum height of constraint levels. Note that when the height of the constraint tree is smaller than H_{max}, there are still not many data points arrived. In this situation, we have enough space to allow the constraint tree being built precisely without any approximation. Therefore, data points are inserted into the constraint tree without any restriction. However, when more and more data points have been inserted, not all of them can be maintained precisely. Therefore, the insertion operation will be adjusted to include more data records in a dense-record bucket while do not lose much information.

We can observe that except for the root node, which has at least two branches, every node contains at least $\left\lceil \frac{m}{2} \right\rceil + 1$ branches. Therefore, the constraint tree is a highly balanced tree, where leaf nodes are all at the same level and each node is at least half full. In addition to the balanced structure of the constraint tree, we also desire that the distribution of the constraint values can be highly balanced. This highly balanced distribution can be achieved more easily for static data sets, where all the data points are available for calculation. However, in the data stream environment, maintaining the balance will be an uneasy job because of the evolving of data points and the limitation of available space. Therefore, we propose a mechanism to construct a more balanced constraint tree for the evolving data stream. The following lemmas help us to build a more balanced constraint tree.

Lemma 1 *The number of leaf nodes, denoted as n_L, in the constraint tree of height H_{max} is in the range of*

$$\left[2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max}-2}, (m+1)^{H_{\max}-1} \right].$$

Proof. Except for the root node, which has at least two links, every node contains at least $(\left\lceil \frac{m}{2} \right\rceil + 1)$ links. Therefore, the number of leaf nodes of the tree with height *H_{max}* will not smaller than $2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max} - 2}$. On the other hand, the maximum number of keys in each node is *m*; at most (*m*+1) links will be in a node. The maximum number of leaf nodes will be $(m+1)^{H_{\max} - 1}$.

Note that a link of a leaf node connects to a dense-record bucket. According to the above lemma, the number of dense-record buckets in the constraint tree can be calculated as the following lemma.

Lemma 2 *The number of dense-record buckets, denoted as n_B, in the constraint tree of height H_{max} is in the range of*

$$\left[2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max} - 1}, (m+1)^{H_{\max}} \right].$$

Note that all the dense-record buckets will cover the whole range of the constraint attribute without overlap.

Proof. The minimum number of dense-record buckets in a leaf node is $(\left\lceil \frac{m}{2} \right\rceil + 1)$. Therefore, the number of dense-record buckets of the tree with height H_{max} will not smaller than $2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max} - 1}$. On the other hand, the maximum number of dense-record buckets in each leaf node is (*m*+1); thus, the maximum number of dense-record buckets will be $(m+1)^{H_{\max}}$.

Recall that a dense-record bucket contains a number of data points whose constraint values are in the range specified by the keys in the leaf node of the constraint tree. In order to maintain more precise constraint values for data points, dense-record buckets with smaller ranges of constraint values will be preferred. However, as shown in Lemma 2, the number of dense-record buckets in a constraint tree is limited, and all the dense-record buckets will cover the whole range of the constraint attribute. Since we have no idea about the distribution of constraint values of incoming data points, a balanced distribution of constraint values into all the dense-record buckets will allow us to maintain data points more precisely. Furthermore, because the number of incoming data points is infinite, which causes the range of the constraint attribute being possible to vary along with the arriving data points, we are not able to specify the ranges of the constraint attribute for all the dense-record buckets directly. Therefore, a guideline is introduced in this paper to adjust the ranges of the constraint attribute in dense-record buckets dynamically according to arriving data points. Since the range of constraint values in a dense-record bucket is affected by the number of dense-record buckets in the constraint tree, we have the following observation.

Observation: From Lemma 2, we can observe that the constraint tree of height H_{max} contains n_B densr-record buckets, where $2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max} - 1} \le n_B \le (m+1)^{H_{\max}}$. Therefore, if the whole range R_{whole} of the constraint attribute is distributed to these dense-record buckets uniformly, each dense-record bucket is desired to cover the range between $\frac{R_{whole}}{(m+1)^{H_{\max}}}$ and $\frac{R_{whole}}{2 \times (\left\lceil \frac{m}{2} \right\rceil + 1)^{H_{\max} - 1}}$.

Because the number of branches in each node is between $(\left\lceil \frac{m}{2} \right\rceil + 1)$ and (*m*+1), we choose a middle value for approximating the range of the constraint values in a dense-record bucket. Accordingly, a guideline is provided as follows.

Lemma 3 *The approximating range R_{app} is assigned as* $\dfrac{R_{whole}}{(\left\lceil \frac{3m}{4} \right\rceil + 1)^{H_{\max}}}$. *When a data point arrives at a leaf node, if the constraint value of the new data point, denoted as key_{new}, is larger than the largest key in the leaf node, denoted as key_{max}, insert the new key if ($key_{new} – key_{max}$) > R_{app}. If it is smaller than the smallest key in the leaf node, denoted as key_{min}, insert the new key if (key_{min}– key_{new}) > R_{app}. Otherwise, this data point is put into a dense-record bucket directly without creating a new key.*

The procedure of constructing the constraint tree is described as follows.

Procedure of Constructing the Constraint Tree

1. When a data point with the constraint value key_{new} is inserted, update the minimum value and the maximum value of the constraint attribute if so necessarily.
2. Traverse the tree to an appropriate leaf node according to the constraint value. Then three cases are considered:
   2.1 Case 1: If the key value key_{new} already exists in the leaf node, insert this data point into a dense-record bucket. If this data point has high similarity with an existing dense-record in the bucket, merge the date point to that dense-record.
   2.2 Case 2: If the height of constraint tree is smaller than H_{max}, insert the constraint value of the data point as a new key. A dense-record bucket with this data point is also created. If this leaf node is full, split it into two nodes and insert the middle key to their parent node. This step of split and insertion will be repeated until no splitting of a parent node is required.
   2.3 Case 3: If the height of constraint tree is equivalent to H_{max}, two cases will be considered:
      Case 3.1: if (key_{new}–key_{max}) $\le R_{app}$ and (key_{min}–key_{new}) $\le R_{app}$, this data point is inserted into a dense-record bucket directly.
      Case 3.2: Otherwise, insert the new key, create the corresponding dense-record bucket, and split nodes as Case 2 while needed. After this insertion, if the height of the constraint tree increases to (H_{max}+1), which exceeds the maximum height allowed, the constraint tree needs to be adjusted. In order to maintain the height of the constraint tree not to exceed H_{max}, the leaf nodes of the constraint tree will be regarded as dense-record buckets afterwards, and the parents of original leaf nodes will become new leaf nodes. The dense-record buckets in an original leaf node are combined into a single dense-record bucket, and dense-records with high similarity are merged. Consequently, the height of the constraint tree is reduced to H_{max} again.

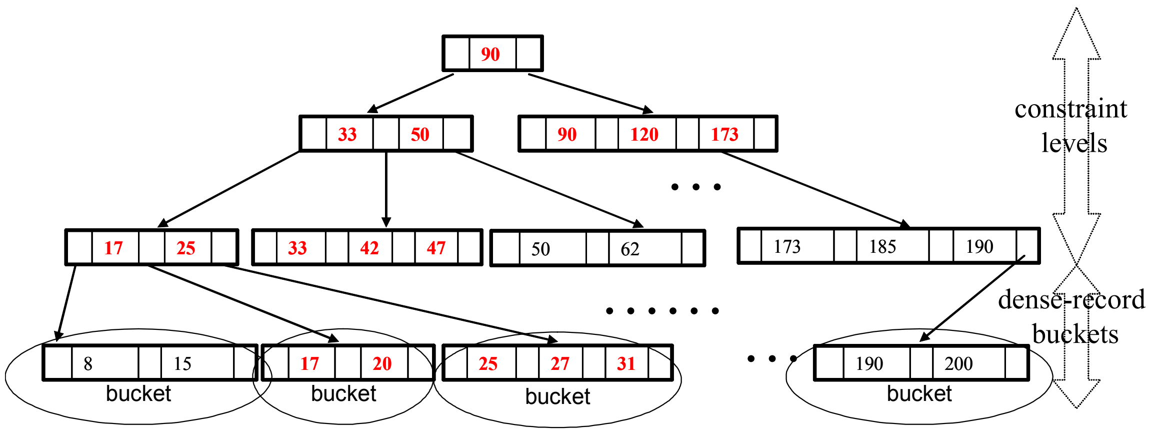


Fig. 3. Illustration of inserting a data point with constraint value 31.

Example 2: Consider the constraint tree in Fig. 2 as an example and recall that m =4 and H_{max} =3. Assume that a data point with key 31 is inserted. Since the constraint tree already reaches the maximum height, Case 3 in the Procedure of Constructing the Constraint Tree will be followed. R_{app} is calculated by $\dfrac{200-8}{\left\lceil \frac{3\times 4}{4} + 1 \right\rceil^{3}} = 3$,

key_{max} of the leaf node is 27, and we will have that (31–27) = 4 > R_{app}. Therefore, Case 3.2 will be followed next. After this insertion, several nodes are split and the height of the constraint tree becomes (H_{max}+1), as shown in Fig. 3, where the split nodes are marked with red and bold key values. Therefore, the original leaf nodes of the constraint tree will be regarded as dense-record buckets afterwards, and the parents of original leaf nodes will become new leaf nodes. Then, the height of the constraint tree is reduced to H_{max} again.

It is noted that if the values of the constraint attribute do not have much correlation with time, the constraint tree will become stable soon, and most of new data records will be inserted to appropriate dense-record buckets accordingly. However, if the values of the constraint attribute are dependent on time, the whole range of the constraint attribute will getting larger and larger. In this situation, our approach is able to adjust the constraint tree dynamically according to the increasing range.

After introducing the properties of constraint levels and the construction of the constraint tree, we start to describe the details of dense-record buckets. A dense-record bucket contains multiple dense-records, and a dense-record, which is an aggregation of similar data points, can be regarded as a micro-cluster within a smaller range of the constraint attribute. A dense-record is represented as the following format,

$$\left\langle [a_C.s, a_C.e], \left(N, \overrightarrow{LS}, \overrightarrow{SS}\right)\right\rangle,$$

where $a_C.s$ and $a_C.e$ are the smallest and largest constraint attribute values of data points in this dense-record respectively, and the part $\left(N, \overrightarrow{LS}, \overrightarrow{SS}\right)$ is the same as the Clustering Feature defined in BIRCH [21]. Accordingly, N is the number of data points in the dense-record, $\overrightarrow{LS}$ is the linear sum of the N data point, i.e., $\sum_{i=1}^{N} \overrightarrow{X_i}$, and $\overrightarrow{SS}$ is the square sum of the N data points, i.e., $\sum_{i=1}^{N} \overrightarrow{X_i^2}$. By the Clustering Feature, the distance between two dense-record $d(F_i, F_j)$, where $F_i = \left\langle [a_C.s_i, a_C.e_i], \left(N, \overrightarrow{LS}, \overrightarrow{SS}\right)\right\rangle$, and $F_j = \left\langle [a_C.s_j, a_C.e_j], \left(N, \overrightarrow{LS}, \overrightarrow{SS}\right)\right\rangle$, can be calculated efficiently.

$$d(F_i,F_j) = \left(\frac{\sum_{a=1}^{N_i} \sum_{b=N_i+1}^{N_i+N_j} \left(\overrightarrow{X_a} - \overrightarrow{X_b}\right)^2}{N_i N_j} \right)^{1/2} = \left(\frac{\sum_{a=1}^{N_i} \sum_{b=N_i+1}^{N_i+N_j} \overrightarrow{X_a^2} + \overrightarrow{X_b^2} - 2\overrightarrow{X_a}\overrightarrow{X_b}}{N_i N_j} \right)^{1/2}$$

$$= \left(\frac{N_j \sum_{a=1}^{N_i} \overrightarrow{X_a^2} + N_i \sum_{b=N_i+1}^{N_i+N_j} \overrightarrow{X_b^2} - 2\sum_{a=1}^{N_i} \overrightarrow{X_a} \sum_{b=N_i+1}^{N_i+N_j} \overrightarrow{X_b}}{N_i N_j} \right)^{1/2} = \left(\frac{N_j \overrightarrow{SS_i} + N_i \overrightarrow{SS_j} - 2\overrightarrow{LS_i}\,\overrightarrow{LS_j}}{N_i N_j} \right)^{1/2}$$

When two dense-records F_i and F_j are merged, the new dense-record will be

$$\left\langle [a_C.s',\ a_C.e'], \left(N_i + N_j,\ \overrightarrow{LS_i} + \overrightarrow{LS_j},\ \overrightarrow{SS_i} + \overrightarrow{SS_j}\right)\right\rangle,$$

Where $a_C.s' = \min(a_C.s_i, a_C.s_j)$ and $a_C.e' = \max(a_C.e_i, a_C.e_j)$.

Example 3: Assume that F_i contains two data points (3, 20, 26) and (5, 22, 24), and F_j contains two data points (6, 21, 25) and (7, 23, 22),where the first value of each data point is the value of the constraint attribute, and others are values of the optimization attributes. Dense-record F_i is represented as

$$\begin{aligned} F_i &= \left\langle [a_C.s_i, a_C.e_i], \left(N_i, \overrightarrow{LS_i}, \overrightarrow{SS_i}\right)\right\rangle \\ &= \left\langle [3,5],(2,(20+22, 26+24),(20^2+22^2, 26^2+24^2))\right\rangle \\ &= \left\langle [3,5],(2,(42, 50),(884, 1252))\right\rangle. \end{aligned}$$

Similarly, dense-record F_j is represented as

$$F_j = \langle [6, 7], (2, (21+23, 25+22), (21^2+23^2, 25^2+22^2)) \rangle$$
$$= \langle [6, 7], (2, (44, 47), (970, 1109)) \rangle.$$

When dense-records F_i and F_j are merged, the new dense-record will be

$$\langle [a_C.s', a_C.e'], (N_i + N_j, \overrightarrow{LS_i} + \overrightarrow{LS_j}, \overrightarrow{SS_i} + \overrightarrow{SS_j}) \rangle$$
$$= \langle [3, 7], (2+2, (42+44, 50+47), (884+970, 1252+1109)) \rangle$$
$$= \langle [3, 7], (4, (86, 97), (1854, 2361)) \rangle.$$

Theorem 1: *The time complexity of the statistics reserving phase is $O(n)$,where n is the number of data points inserted.*

Proof. The time complexity of reading n data points is $O(n)$. When the height of the constraint tree is smaller than H_{max}, a data point traverse $O(\log_m n)$ levels to the leaf node, and the maximum number of split caused by this insertion is also $O(\log_m n)$. Therefore, the overall complexity is $O(n)+O(\log_m n)= O(n)$. On the other hand, when the height of the constraint tree is H_{max}, a data point traverse H_{max} levels to the leaf node, and the maximum number of split caused by this insertion is also H_{max}. However, if the height exceeds H_{max} during the insertion, the leaf nodes and dense-record buckets should be processed in the complexity of $O((m+1)^{Hmax})$. Therefore, the total time complexity of insertion will be $O(H_{max} + (m+1)^{Hmax})$, which is independent to the data number n. Consequently, the overall time complexity of the statistics reserving phase will be $O(n)+O(H_{max}+(m+1)^{Hmax})= O(n)$.

As will be shown in the performance studies, the time complexity of the statistics reserving phase, which is the time complexity of constructing the constraint tree, is linear in the number of data points. Therefore, framework CCDS is very efficient and suitable for the data stream environment.

3.2 Clustering Responding Phase

As the clustering request shown in Section 2, users want to inspect clusters with the cluster number being k and the constraint range being R_{a_C}. Note that the clustering algorithm is not applied to the original data set. Instead, the dense-records maintained in the leaf nodes of the constraint tree are retrieved for clustering. We apply the constrained clustering algorithm progressive CCL, which is proposed in the work [13], to generate clusters according to the clustering requests. The distance between clusters is defined as follows.

$$dist(c_i, c_j) = \max\{d(F_a, F_b) \mid F_a \in C_i, F_b \in C_j\}.$$

Here we take the time constraint attribute a_{time} as example, i.e., the constraint attribute a_C represents a_{time}. For a cluster C, we define the start constraint value, denoted by $C.ts$, and end constraint value, denoted by $C.te$, as the smallest and largest constraint attribute values of data points in the cluster respectively, i.e., $C.ts = \min\{a_C.s_a \mid F_a \in C\}$ and $C.te = \max\{a_C.e_a \mid F_a \in C\}$. Then, the constraint distance between two clusters C_i and C_j is determined as

$$d_{a_C}(c_i, c_j) = \max\{C_i.te - C_j.ts, C_j.te - C_i.ts\}.$$

The distance measurement of two clusters is modified for this constrained clustering problem. Instead of the original distance $dist(.,.)$, the distance measurement between two clusters is defined as follows:

$$\widehat{dist}(c_i, c_j) = \begin{cases} dist(c_i, c_j), & if\ d_{a_C}(c_i, c_j) \le R_{a_C}, \\ \infty, & otherwise. \end{cases}$$

The outline of algorithm progressive CCL is shown in Fig. 4. The basic idea of algorithm progressive CCL is that by using a tighter constraint range in early iterations of merging, we will have more room to seek for better solutions in subsequent iterations. In addition to the relaxation of the constraint range, the desired cluster number is also temporarily modified accordingly. The whole process starts with a small local constraint range and a large local desired cluster number. The procedure continuously relaxes the constraint range and reduces the number of desired clusters, and eventually results in the actual constraint range and the cluster number specified by the user. Explicitly, a parameter *level* is used to control the number of relaxation steps.

```
Progressive CCL
//Input: an input data set, the number of clusters, k, the relaxation level, and the constraint range $R_{ac}$
1. For $i = 1$ to level, do Step 2 and Step 3.
2. Calculate the local_k and the local_$R_{ac}$,
   where $Local\_k=(n\times(level-i)+k\times i)/level$, and $local\_R_{ac} = R_{ac}\times(k/local\_k)$.
3. Run the algorithm CCL based on local_k and local_$R_{ac}$.

Algorithm CCL
//Input: an input data set, the number of clusters, k, and the constraint range $R_{ac}$
1. Initially, each data point forms a cluster by itself.
2. The algorithm repetitively merges the two closest clusters.
3. Repeat Step 2 until exactly k clusters left or all pairs of points exceed the constraint range.
```

Fig. 4. Outline of algorithm progressive CCL

Theorem 2: *The time complexity of the clustering responding phase is*

$$O\left(n_R r_c k \log(\frac{n_R r_c k}{level})\right) + O\left(\frac{n_R^2 r_c}{level^2}\log(\frac{n_R^2 r_c}{level^2})\right) + O(f \times level),$$

where r_c is between 0 and 1, n_R is the number of dense-records retrieved from the constraint tree, $O(f)$ represents the lower order terms which can be ignored in the analysis of CCL, and level is iterations of algorithm CCL performed in algorithm progressive CCL.

Proof. The time complexity of algorithm progressive CCL is

$$O\left(n r_c k \log(\frac{n r_c k}{level})\right) + O\left(\frac{n^2 r_c}{level^2}\log(\frac{n^2 r_c}{level^2})\right) + O(f \times level).$$

In the CCDS framework, only the dense-records retrieved are used for the clustering. Therefore, the number of data points n will be replaced by the number of dense-records n_R.

Since only dense-records are applied to the clustering algorithm, framework CCDS is more efficient than algorithm progressive CCL. As will be shown in the performance studies, when more and more data points are inserted, the superiority of framework CCDS over algorithm progressive CCL is more significant because more similar data points have been merged into dense-records.

4 Performance Studies

To assess the performance of these algorithms, we have conducted a series of experiments. These experiments are performed on a computer with an 1.7 GHz Intel CPU and 1 GB of memory. In order to generate the synthetic data used in our experiments, we adopt a method similar to the one in [13] and [21], by which n data points for k clusters are generated. Without loss of generality, every generated data point contains two optimization attributes and one constraint attribute. Several such synthetic data sets are combined together to simulate the phenomenon of clusters with time constraint. Both of the optimization attributes are in the range of {–100, 100}, and the range of the constraint attribute is {0, 10000}. Four such data sets, each of which contains 5000 data points for 5 clusters, are combined into 20000 data points with 20 clusters which may overlap with one another in optimization and constraint attributes. All the data sets in our experiments are random samples of this data set. Therefore, these data sets have the same distribution as described above. In the performance studies, we will compare our CCDS framework with algorithm progressive CCL [13]. Note that algorithm progressive CCL can only deal with static data set; therefore, all data points are given in advance. On the other hand, to show the merit of framework CCDS on supporting streaming data sets in which data points are collected continuously, we let each data point be received at an individual time stamp.

In Section 4.1, the CCDS framework is evaluated over different values of parameters. In Section 4.2, we conduct a series experiments of different clustering requests to compare the performance of CCDS framework with algorithm progressive CCL. Finally, we examine the scalability of CCDS framework in Section 4.3.

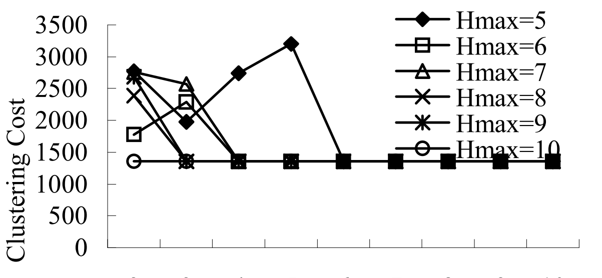
In the following experiments, we use the average squared error of all points as the evaluation function for clustering results, i.e.,

$$Cost(Cl) = sq(Cl) = \frac{\sum_{C_i} \sum_{p \in C_i} (p - C_i.center)^2}{\sum_{C_i} \| C_i \|}$$

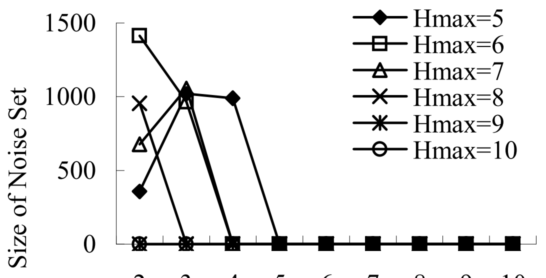
where C_i denotes a cluster of the clustering result Cl, and C_i.center is the center of cluster C_i.

4.1 Parameter Sensitivity of CCDS

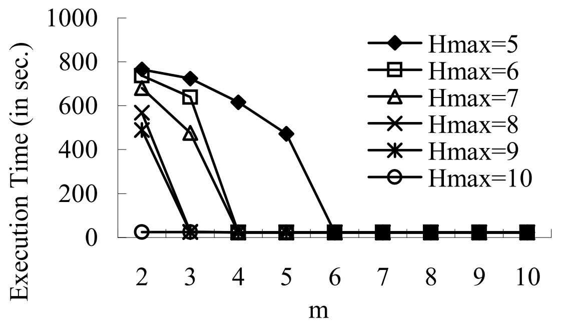
In this section, two parameters of the CCDS framework, which are the maximum number of keys in each node (m) and the maximum height of the constraint tree (H_{max}) are investigated with a fixed clustering request (k, R_{a_C})=(20, 4000). The data set of 20000 data points is used in this experiment. The size of noise set represents the number of data points which cannot join a cluster without violating the constraint range. As shown in Fig. 5, when the number of keys m and the height of the constraint tree H_{max} are too small, the number of dense-record buckets will be too few. Therefore, the ranges of the constraint attribute covered by each dense-record bucket will be too large or biased, and the dense-records maintained will not be good representatives for similar data points within small constraint ranges. Consequently, the clustering quality will be poor and many data points willbe regarded as noises since they cannot be inserted into clusters subject to the constraint. However, when parameters m and H_{max} increase to sufficiently large values, most of the data points can be maintained effectively, and the performance of framework CCDS becomes stable.



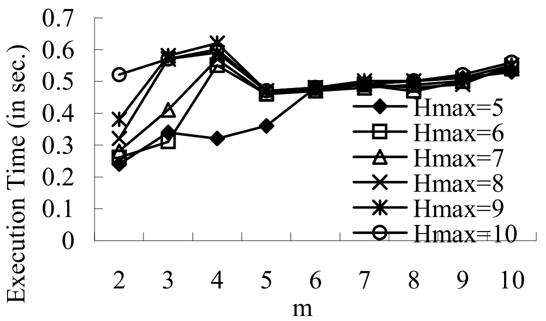
(a) Clustering Cost



(b) Size of Noise Set

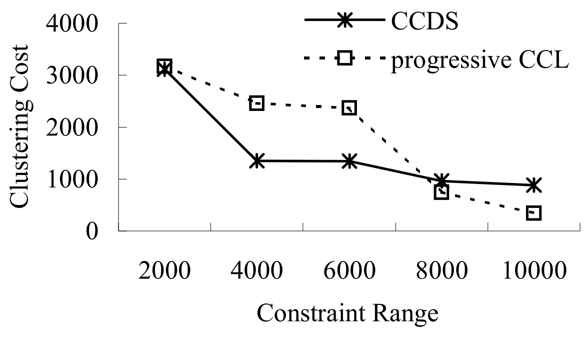


(c) Responding Time to a Clustering Request

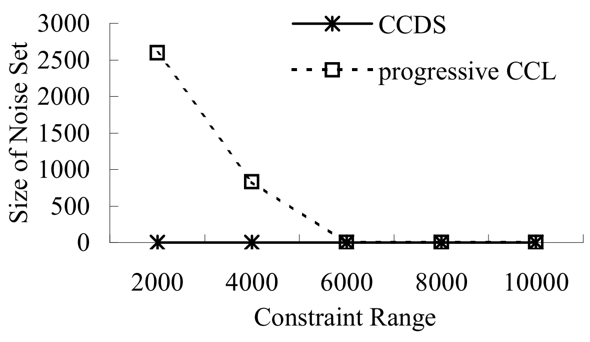


(d) Construction Time of Constraint tree

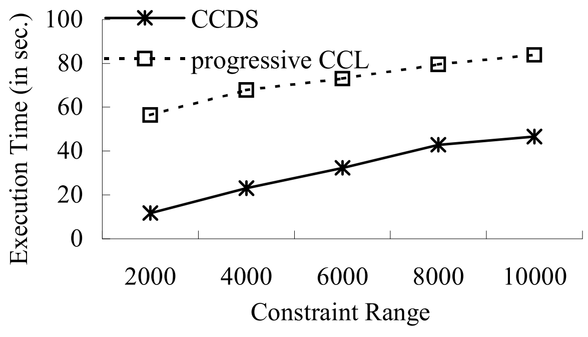
Fig. 5. Performance of CCDS for different parameter settings.



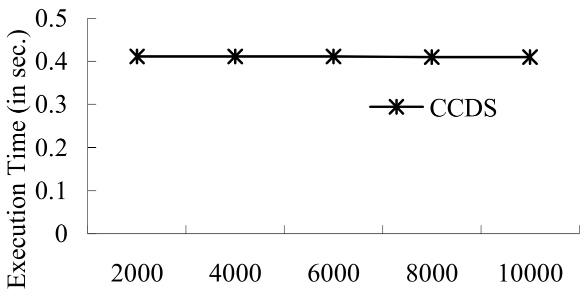
(a) Clustering Cost



(b) Size of Noise Set

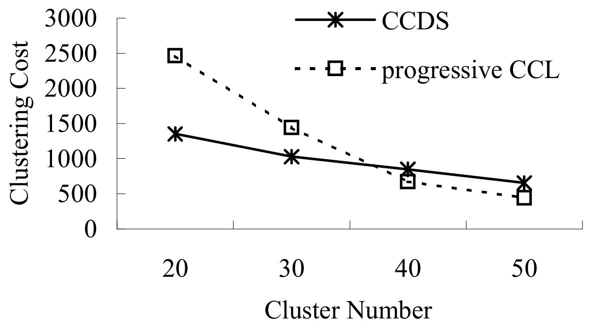


(c) Responding Time to a Clustering Request

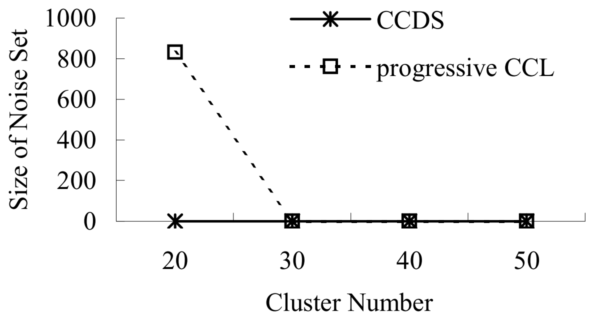


(d) Construction Time of Constraint tree

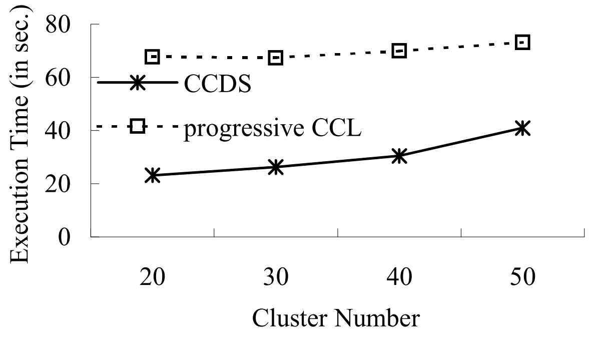
Fig.6. Comparison between CCDS and progressive CCL on (a) clustering costs, (b) size of noise set, (c) execution time of clustering, (d) construction time of constraint tree when the constraint range varied.



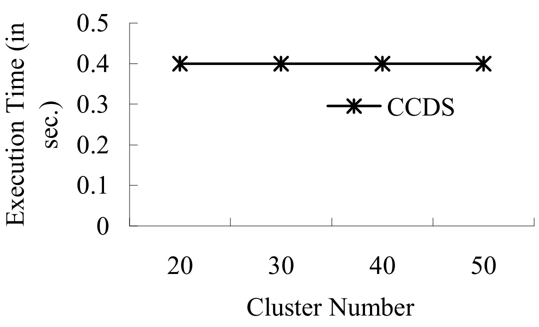
(a) Clustering Cost



(b) Size of Noise Set



(c) Responding Time to a Clustering Request



(d) Construction Time of Constraint tree

Fig.7. Comparison between CCDS and progressive CCL on (a) clustering costs, (b) size of noise set, (c) execution time of clustering, (d) construction time of constraint tree when the number of clusters varied.

4.2 Clustering Result by CCDS

In this section, various clustering requests (k, R_{a_C}) are submitted to evaluate the performance of CCDS framework. As mentioned earlier, algorithm progressive CCL is also executed on the same clustering requests for comparison. The same data set of 20000 data points is used. First, given a fixed cluster number k being 20, the performance against different constraint ranges R_{a_C} is investigated. As shown in Fig. 6(a), when the constraint range is very small, more similar data points are assigned into different clusters because they exceed the constraint range. Therefore, the clustering costs of two algorithms are both higher. Note that although framework CCDS applies the same clustering algorithm as algorithm progressive CCL, it assigns much more data points into appropriated clusters without violating the constraint range because the constraint tree structure of framework CCDS provides higher priority for the constraint attribute, as shown in Fig. 6(b). On the other hand, when the constraint range is very large, say, near to the whole range of the constraint attribute, clustering results similar to those without constraint will be generated. Therefore, the clustering costs will be lower. In this situation, the performance of algorithm CCL is slightly better than framework CCDS because that the approximation of data points in the constraint tree sacrifices a little accuracy for the limited space. Except these two extreme cases discussed above, the clustering quality of framework CCDS is much better than algorithm progressive CCL. These results show that framework CCDS is able to solve the problem of clustering with pairwise numerical constraint gracefully, while requiring much shorter execution time for generating clusters, as shown in Fig. 6(c). It is also noted that the execution time for constructing the constraint tree is very short, as shown in Fig. 6(d). As shown in Fig. 7, similar results can also be obtained from the experiments of varying the number of clusters (k) with a fixed constraint range. Therefore, our CCDS framework is able to maintain the statistics of the data stream very efficiently, while generating clusters with high quality by much shorter clustering time.

4.3 On Scalability

To evaluate the scalability of the proposed framework, the scale-up experiments on the number of data points are conducted. Recall that one data point is received at each time stamp for framework CCDS. Therefore, the value of the time stamp shown in Fig. 8 is also the number of data points accumulated at that time. As shown in Fig. 8(a) and Fig. 8(b), the properties of the constraint tree, such as considering the constraint attribute with a higher

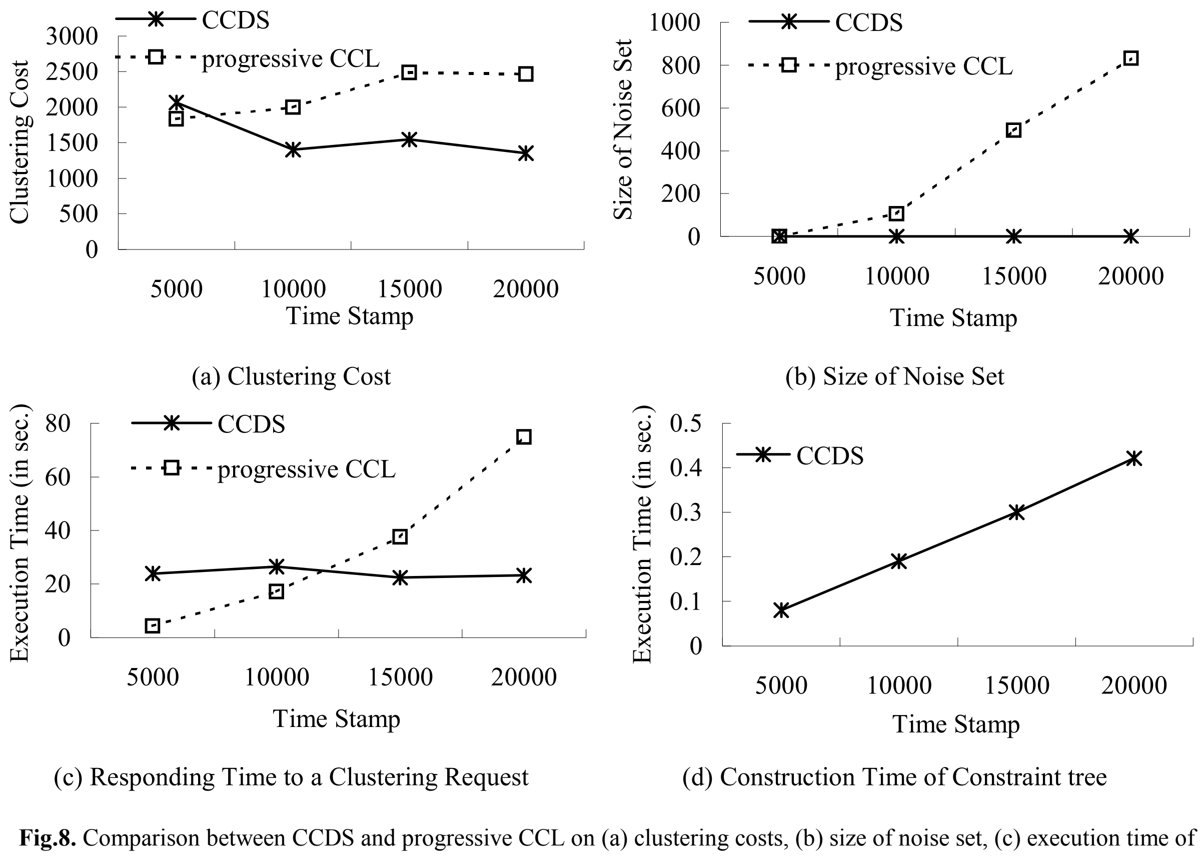


Fig.8. Comparison between CCDS and progressive CCL on (a) clustering costs, (b) size of noise set, (c) execution time of clustering, (d) construction time of constraint tree when the number of data points varied.

priority and merging similar data points in dense-record buckets, allow the clustering responding phase of framework CCDS to produce clusters with higher quality than algorithm progressive CCL when the data set becomes larger in the data stream environment. Also illustrated by Fig. 8(c), clustering on the summarized statistics is very efficient for large data sets. This result conforms with Theorem 2, which states that the time required for responding to a clustering request is dependent on the number of dense-records rather than the number of data points. In addition, as shown in Fig. 8(d), as the number of data points increases from 5, 000 to 20, 000, the execution time of constructing the constraint tree is very short and grows linearly. This experiment also validates the time complexity analyzed in Theorem 1, which expresses that the time required for the statistics reserving phase is in $O(n)$. Therefore, our CCDS framework is able to handle fast data streams very efficiently while producing excellent clustering results.

5 Conclusions

In this paper, we devised a framework named CCDS framework to cluster the data stream under the pairwise numerical constraint. The CCDS framework proposed consists of two phases, namely the statistics reserving phase and the clustering responding phase. The statistics reserving phase provides an efficient algorithm to process and maintain the data points in a compact structure named constraint tree. On the other hand, the statistics maintained will be retrieved in the clustering responding phase to generate clusters according to clustering requests specified by users. Since the clustering requirement is obtained only when a user submits the request, the maintained information in the constraint tree has the ability to service various clustering requests without processing the original data points again. As shown in the complexity analyses and also validated by our empirical studies, the CCDS framework performs very efficiently in the data stream environment while producing clustering results of very high quality.

References

[1] M.-S. Chen, J. Han, and P. S. Yu, “Data mining: An overview from database perspective,” *IEEE Trans. on Knowledge And Data Engineering,* Vol.5, No.1, pp.866–883, Dec. 1996.

[2] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems,” *Proceedings of PODS*, June 2002.

[4] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, “Computing on data streams,” *DIMACS*, Vol.50, pp.107–118, 1999.

[5] G. S. Manku and R. Motwani, “Approximate frequency counts over streaming data,” *Proceedings of VLDB*, pp. 346–357, Aug. 2002.

[6] W.-G. Teng, M.-S. Chen, and P. S. Yu, “A regression-based temporal pattern mining scheme for data streams,” *Proceedings of VLDB,* Sep. 2003.

[7] C. C. Aggarwal, J. W. Han, J. Y. Wang, and P. S, Yu. “A framework for projected clustering of high dimensional data streams,” *Proceedings of VLDB*, pp. 852–863, 2004.

[8] S. Guha, N. Mishra, R. Motwani, and L. O’ Callaghan, “Clustering data streams,” *Proceedings of FOCS*, pp. 359–366, Nov. 2000.

[9] L. O’ Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, “Streaming-data algorithms for high-quality clustering,” *Proceedings of ICDE*, 2002.

[10] C. C. Aggarwal, J. W. Han, J. Y. Wang, and P. S. Yu, “On demand classification of data streams,” *Proceedings of ACM SIGKDD,* pp. 503–508, 2004.

[11] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” *Proceedings of ACM SIGKDD*, pp. 97–106, Aug. 2001.

[12] P. S. Bradley, K. P. Bennett, and A. Demiriz, *Constrained K-Means Clustering*, MSR-TR-2000-65, Microsoft Research, May 2000.

[13] B.-R. Dai, C.-R. Lin, and M.-S. Chen, "On the techniques for data clustering with numerical constraints," *Proceedings of SDM*, 2003.

[14] D. Klein, S. D. Kamvar, and C. Manning, "From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering," *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002), Sydney, Australia*, 2002.

[15] C.-R. Lin, K.-H. Liu, and M.-S. Chen, "Dual clustering: Integrating data clustering over optimization and constraint domains," *IEEE Trans. on Knowledge and Data Engineering*, Vol.17, No.5, pp.628–637, May 2005.

[16] A. K. H. Tung, J. Han, L. V. S. Lakshmanan, and R. T. Ng, "Constraint-based clustering in large databases," *Proceedings of 2001 International Conference on Database Theory,* Jan. 2001.

[17] O. R. Zaïane, A. Foss, C.-H. Lee, and W. Wang, "On data clustering analysis: Scalability, constraints, and validation," *Proceedings of PAK DD*, pp. 28–39, 2002.

[18] S. L. Huang, L. C. Wu, H. K. Laing, K. T. Pan, M. T. Ko, and J. T. Horng, "Pgtdb: a database providing growth temperatures of prokaryotes," *Bioinformatics*, Vol.20, No.2, pp.276-278, January 2004.

[19] M. Yeung and B. L. Yeo, "Time-constrained clustering for segmentation of video into story units," *Proceedings of the International Conference on Pattern Recognition,* pp. 375–380, May 1996.

[20] B.-R. Dai, C.-R. Lin, and M.-S. Chen, "Constrained Data Clustering by Depth Control and Progressive Constraint Relaxation," *Very Large Data Base Journal (VLDBJ)*, Vol. 16, No. 2, pp. 201-217, April 2007.

[21] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large database," *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 103–114, 1996.