An Agent-Based Dual-Tier Algorithm for Clustering Data Streams

DongBin Zhou, LiFeng Jia, Zhe Wang, XiuJuan Xu, ChunGuang Zhou

Abstract—**Characteristics of data stream make it difficult for the clustering algorithms to satisfy the requirements on efficiency and effectiveness. This paper proposes a data stream clustering algorithm on dual-tier structure which employs the agent method. In the on-line process, a set of agents working simultaneously collect similar data points into sub-clusters by applying a heuristic strategy. And in the off-line process, summary information from the on-line component will be further analyzed to obtain the final clusters. The algorithm also supports the time-window queries on streams. The empirical evidence shows that this method can obtain high-quality clusters with low time complexity.**

Index Terms—**Agent, Clustering, Data Mining, Data Stream**

I. Introduction

With the coming of the age of information, commerce, telecom, medical treatment and other industries are being confronted with a large amount of fast-arriving data more and more frequently. The importance of streaming data analysis gradually stands out in the domain of data mining, and clustering data streams has become a heated topic.

A data stream is an ordered sequence of high-speed points which are coming continuously. Usually a data stream can be regarded as a dynamic data set, the scale of which increases infinitely with the lapse of time. As a result, it is too expensive to access a point in stream randomly, and thus requiring a single scan over the stream has become an object of the clustering algorithms. Clustering algorithms partition a data set into several disjoint groups such that points in the same group are similar to each other according to some similarity metric [1]. Clustering on streaming data brings forward challenges for traditional algorithms in the following aspects: obtaining high quality clusters by only one-pass over the data; time-window analysis over an arbitrary period of the stream etc. As for stream clustering, a common method is dividing the streaming data into chunks, and algorithms for static sets can be used on each sub-set separately [2]. In recent years, stream algorithms have developed into a two-phase structure [3], [4]. Usually, a dual framework includes two parts: the on-line component and the off-line component. The former is responsible for the fast but rough processing of streaming data and saving the summary information to meet the one-pass restriction while the latter takes advantage of the information to conduct high-level analysis. At present, stream algorithms are still facing some problems, for example: sensitive to the initial data points; bad quality of clusters due to the loss of global information caused by dividing the stream; high time complexity etc.

A novel dual-tier clustering algorithm for data streams, *AGCluStream*, is proposed in this paper. The on-line algorithm uses agents to make similar points denser in local areas, and record the temporary distribution of data according to the *pyramidal time frame* [3]. The off-line algorithm uses these records to conduct time-window analysis and higher-level clustering analysis. *AGCluStream* dose not divide the stream, and it adopts an incomplete-partition strategy to maintain the global information more effectively.

This paper is organized as follows. In Section *II*, we will discuss the framework of *AGCluStream* including: In *II.A*, we will introduce the framework. In *II.B*, we will introduce the concept of *grid-cluster*. In *II.C*, we will explain the concrete procedure of agent operation. In *II.D*, we will introduce the data condensing method. In *II.E*, we will discuss the details of the on-line algorithm. In *II.F*, we will discuss the process of time-window analysis. In Section *III*, we will show the experiment results. Section *IV* is the conclusion part.

II. The Framework of AGCluStream

A. A Brief Introduction to AGCluStream

AGCluStream is a heuristic data stream clustering algorithm based on agent method, falling into the on-line and the off-line components. The on-line algorithm saves data information in a set of data structure called *grid-cluster*, and the agents adopt heuristic strategy to move data among grid-clusters ceaselessly to increase the degree of the data similarity in individual grid-cluster. In this process, newly arriving points continuously enter the grid-clusters, and then are carried to a proper position, where there are more similar points. Points in grid-clusters need

Manuscript received September 29, 2005. This work was supported by the National Natural Science Foundation of China under Grant No. 60433020, the Key Laboratory for Symbol Computation and Knowledge Engineering of the National Education Ministry of China, and the Computational and Software Scientific and Technological Innovation of 985 Project.

D.B.Zhou Author is with the Department of Computer Science, Jilin University, Changchun, China (e-mail: zdbjob@yahoo.com.cn).

L.F.Jia Author is with the Department of Computer Science, Jilin University, Changchun, China. (e-mail: JuniorJia@yahoo.com.cn)

Z.Wang Author is with the Department of Computer Science, Jilin University, Changchun, China. (e-mail: wz2000@jlu.edu.cn).

X.J.Xu Author is with the Department of Computer Science, Jilin University, Changchun, China. (e-mail: xuxiujuan666@yahoo.com.cn)

C.G.Zhou Author is with the Department of Computer Science, Jilin University, Changchun, China. (e-mail: cgzhou@jlu.edu.cn)

to be condensed at some time to release the memory for the future points. The on-line algorithm takes snapshots of the data stored in memory in time to record the data distribution at that moment. The off-line algorithm uses these snapshots to conduct further analysis and obtain the final clustering results.

B. Grid-Cluster

A grid-cluster is a data structure used to save streaming data and their statistical information in *AGCluStream*. It is assumed that, the data stream consists of a set of continuous records $X_1 \ldots X_i \ldots$ and each X_i is a multi-dimensional record containing d dimensions which are denote by $X_i = (x_i^1 \ldots x_i^d)$. Then a grid-cluster is defined as follows:

Definition 1. (Grid-Cluster): A grid-cluster is used to store the data points. It always maintains three statistical variables about the data inside: *N*, *LS* and *SS*. It is assumed that there are N points $X_1 \ldots X_N$ saved in the grid-cluster G, these statistical variables are defined as follows:

N: The total number of the points in *G*.

LS: *LS* corresponds to a vector of d entries $LS = (P_1 \ldots P_d)$, where the *i*-th entry $P_i = \Sigma_{k=1}^{N} x_i^k$.

SS: *SS* corresponds to a vector of d entries $SS = (Q_1 \ldots Q_d)$, where the *i*-th entry $Q_i = \Sigma_{k=1}^{N} (x_i^k)^2$. ■

The definition shows that each grid-cluster corresponds to a vector of three entries (*N*, *LS*, *SS*). Actually, grid-cluster is similar to the micro-cluster in *CluStream* [3], except that what is saved in grid-cluster is not only the summary information but the real data points. Another difference between grid-cluster and micro-cluster is that grid-cluster need not maintain the time-stamps of points. In *CluStream*, time-stamps are used to identify the micro-clusters which need to be deleted. However, it is dangerous to delete historical information of a stream unless take the attenuation into account, because some outliers in different time horizons probably constitute a cluster from the global point of view. Therefore, any deletion only depends on the temporary distribution of current data could cause the loss of information. *AGCluStream* avoids deleting operations by condensing data with the incomplete-partition strategy.

C. Agent Operation

Agents can be regarded as a set of independent working threads, there can be several agents working simultaneously. All of the agents share the same algorithm procedure, and thus *AGCluStream* supports the multi-threads run mode. The agent operation aims to increase the density of similar points by moving points among grid-clusters. It is necessary to define the density parameter of data first.

Definition 2. (Density Parameter): The density of similar points in a grid-cluster is denoted by J which is defined as $J = \Sigma_{k=1}^{N} D(X_k, X_c)^2 / N$, where $D(X_k, X_c)$ denotes the distance from X_k to the center X_c. ■

As the definition shows, J is actually the variance statistic of data in grid-cluster and the formula to calculate the value of J is $J = |SS| / N - | (LS / N)^2 |$. For a grid-cluster G_i and a point X_j, there is a conclusion as follows:

Lemma 1. Let J be the density parameter of grid-cluster G_i, ΔJ be the increment of J, X_c be the center of G_i, N be the number of points in G_i, and $N >> 1$. Then,

If $X_j \in G_i$, when X_j is deleted from G_i, the necessary and sufficient condition for $\Delta J < 0$ is: $\Sigma_{k=1}^{d} (x_i^k - x_c^i)^2 > J$ which is referred to as the *pick-up condition*.

If $X_j \notin G_i$, when X_j is added into G_i, the necessary and sufficient condition for $\Delta J < 0$ is: $\Sigma_{k=1}^{d} (x_i^k - x_c^i)^2 < J$ which is referred to as the *release condition*. ■

Proof: Above conclusions are obvious. Skip proof here. ■

We define two states for an agent: *Loading* and *Idling*, and two types of operation: *Pick-up* and *Release*. Agents in idling state first pick up points from the buffer where store the points which have just arrived. When the buffer is empty, the idle agents will pick up points that satisfy the pick-up condition from the grid-clusters. When an idle agent gets a point X_i, it will fall into the loading state. A loading agent tries to look for a grid-cluster which makes X_i satisfy the pick-up condition to release the loading point.

Agent operations constitute the main part of the on-line algorithm, and thus the efficiency of judging the pick-up and release conditions has a direct impact on time complexity of the algorithm. As the calculation of *Euclidean* distance usually requires a high cost, we can take the judgment of sufficient conditions as an assistant method to judge the necessary and sufficient conditions by the *Manhattan* distance.

Lemma 2. Let J be the density parameter of the grid-cluster Gi, ΔJ be the increment of J, X_c be the center of G_i, and d be the number of data dimensions. Then,

If $X_j \in G_i$, when X_j is deleted from G_i, the sufficient condition for $\Delta J<0$ is: $Max\{| x_i^k - x_c^k | | k \in [1,d]\} > \sqrt{J}$ (1)

If $X_j \notin G_i$, when X_j is added into G_i, the sufficient condition for $\Delta J<0$ is: $\Sigma_{k=1}^{d}(| x_i^k - x_c^k |) < \sqrt{J}$ (2) ■

Proof: As for the case of *Formula1*, we know from *Lemma1* that points which satisfy the pick-up condition are distributed in the data space outside the *d*-dimension sphere *O* with $\sqrt{J}$ as its radius. Points which satisfy *Formula1* are distributed in the space outside the *d*-dimension circumscribed-polyhedron of *O*, so they satisfy the pick-up condition at the same time. It is similar to the case of *Formula 2*. ■

Lemma 3 Let R be the radius of the sub-cluster constituted by points in grid-cluster G_j, $t = d/2$ and $\Gamma(n) = \int_0^{+\infty} x^{n-1} e^{-x} dx$.

The error function for judging the pick-up condition by Formula 1 is: $f_p(d) = \dfrac{2\Gamma(t+1)(dJ)^t - \pi^t J^t}{\pi^t (R^{2t} - J^t)}$

The error function for judging the release condition by Formula 2 is: $f_r(d) = \dfrac{\pi^t - 2\Gamma(t+1)}{\pi^t}$ ■

Proof: As for the case of the pick-up condition judgment, we know from the proof of *Lemma 2* that errors are determined by the difference between the volume of the *d*-dimension sphere and that of its regular circumscribed-polyhedron. It is similar to the case of release judgment. ■

Let agents be in idling state initially, Q be the maximum exploration steps of a agent, M be the minimum points number in grid-cluster, N be the maximum points number in grid-cluster, and $D(X_i, G_i)$ be the distance from the point X_i to the center of grid-cluster G_i. The agent operation algorithm is summarized as follows:

Algorithm AgentOP (Q, M, N)

```
1. Check whether the buffer is empty, if not empty, pick up a point from the buffer and fall into the loading state, then go to 4; otherwise, go to 2.
2. Skip randomly to a grid-cluster $G_i$, if the point number $C_i < M$, go to 1.
3. Choose a point $X_i$ randomly, if it satisfies the pick-up condition, pick up this point and fall into the loading state, add $G_i$ to a set $S$, exploration counter $T = 0$, go to 4; otherwise, repeat this step.
4. Skip randomly to a grid-cluster $G_j$ ($j \neq i$), if the point number $C_j < M$, add $G_j$ to $S$, go to 6; If $C_j > N$, repeat this step. Otherwise, go to 5.
5. If the loading point satisfies the release condition, release it into the current grid-cluster, go to ends. Otherwise, add $G_j$ to $S$, go to 6.
6. $T = T + 1$, if $T \leq Q$ go to 4; otherwise, release the loading point into the grid-cluster $G_k$, where k is defined as $D(X_i, G_k) = \text{Min}\ \{D(X_i, G_i) \mid G_i \in S\}$, clear $S$. ■
```

Note that S stores G_0 as the original grid-cluster from where the loading point is picked up, if $D(X_i, G_0)$ is the minimum value of the distance from points in S to the center, the point will not move, and we say that the agent has executed a *futile action*. The ratio of the futile action times to the total action times in a unit time is referred to as the *futile action rate*.

D. Data Condensing

There are a great number of points stored in the grid-clusters, and the memory will be exhausted with the accumulation of streaming data. The condensing operation combines the denser points near the grid-cluster center to a *complex-point* with a higher *weight*, witch denotes the number of points it contains, and the peripheral sparse points are left for the future treatment together with later points. This method is referred to as an *incomplete-partition strategy*.

Assume that there are currently C points in memory, condensing operation will sort the points in descending order by distance from their centers, delete the first E points and gather the remaining points towards the center. We denote the condensing rate by the variable $\rho = 1 - E/C$.

Let C be the number of points in current memory, D_c^i be the center of grid-cluster G_i, C_i be the number of points in G_i, W_i be the weight of the point D_i, and ρ be the condensing rate. We summarize the condensing algorithm as follows:

Algorithm Condensing (C, ρ)

```
1. Sort all the points in descending order of distance from their centers, and get the ordered sequence $D_1 \dots D_c$.
2. Calculate the number of the points to be deleted $E = C \times (1 - \rho)$. $S$ is a set of the deleted points. For each $i \in [1, E]$, delete $D_i$ from the grid-cluster it belongs to, and add it to $S$.
3. For each grid-cluster $G_i$, create a new point $D_i = D_c^i$, with the weight $W_i = C_i$. Delete all the points in $G_i$ and then add $D_i$ to $G_i$.
4. Start agent threads to pick up the points in $S$, and release them to proper grid-clusters until S is empty. ■
```

The grid-clusters need to be stored at snapshots in time in the on-line stage. Snapshot can be regard as a mirror image of current memory usually stored on disk, and it is also necessary to condense the points before the storage of snapshot to reduce the space complexity.

E. On-Line Algorithm

The more points the algorithm maintains, the better it can reflect the data distribution. Assume that there are G grid-clusters with N as their capacity, the maximum number of points can be maintained in the algorithm is $C < G \times N$ (When C approaches to $G \times N$, the condensing process will be triggered). As a result, there are C points and G grid-clusters stored in memory at the same time, the space complexity is $O(C + G)$. It is assumed that the probability for picking up a point that satisfy the pick-up condition is p, the time complexity for the pick-up operation is $O(f(p))$, and the probability density function is $P(f(p) = k) = (1 - p)^k$. Let Q be the maximum exploration steps, then the time complexity for the release operation is $O(Q)$.

The storage of the snapshots needs to put out all the data in memory. There are two types of the output data points, the complex-points formed in the condensing process and the origin points still remaining in memory. As the time window-analysis process requires identifying the different types of points on snapshots, we have to assign a unique *id* to each point when a snapshot needs to be stored after a condensing operation. An origin point simply corresponds to a unique *id*. A complex-point D_c represents a point set S, if there is no point that has already got an id in S, D_c can be assigned a unique *id* simply; otherwise, an *id* list L needs to be created for D_c and all the *ids* in S should be copied to L, if S also contains some *id* lists, they should be copied to L either.

AGCluStream needs not accumulate data for initialization and is insensitive to the initial points. Once start an agent thread, the on-line algorithm begin to run immediately. The futile action rate increases gradually while similar points gather towards to each individual grid-cluster. Futile actions may cause a waste of the system resources, and thus the agents should be reduced as the futile rate rises to save system resources. Once the number of agents is reduced to be zero, the algorithm will end automatically.

F. Time-Window Analysis

During the time-window analysis, *AGCluStream* chooses the points of a given time horizon and performs clustering on them. Time-window analysis relies on the snapshots which are stored on the disk. In fact, because of the condensing operation, the output O_A of *AGCluStream* is much smaller than that of *CluStream* O_c ($O_A \approx \rho\ O_c$), this proves that the storage requirement of *AGCluStream* is quite modest.

Time-window analysis needs two input parameters: the current time t_c and the length of the time window h. Time window $W(t_c, h)$ represents a past time horizon of h from t_c. Let t_n be the current time, T be the time horizon from the beginning of the stream to t_n, a common case in practice is $t_c = t_n$ and $h = T$, i.e., it is required to cluster all the data in stream by now. From the above statement we know that the snapshot S_n stored at t_n contains information of all the historical data since the beginning of the stream, so we could get the final cluster result by S_n. A complex-point represents a set of points at the same position and could be chosen as a cluster centroid. If the input cluster number is K, we first employ the *K-Means* [5] method to cluster the complex-points on S_n to get K clustering centroids, and then apply *K-Means* again over all the weighted points on S_n with these centroids to obtain the final result.

In common cases, $h < T$, and the time frame ensures that it is always possible to find a snapshot at t_s, $t_s < t_c - h$, and $h' = t_c - t_s$, is within a pre-specified tolerance of h. Assume that snapshots saved at t_c, t_s are S_c, S_s, points in time window $W(t_c, h)$ are those which exist only in S_c but not in S_s, i.e. they are in the point set $S(t_c, h) = S_c - S_s$. $S(t_c, h)$ can be obtained by the point ids as follows: Initially, $S(t_c, h) = S_c$, for each point D_i in S_s, let ID_i be its *id* and W_i be its weight, there are two different cases:

(1) If D_i is an origin point in S_s, delete D_i from $S(t_c, h)$.
(2) If D_i has been combined into a complex-point D_c in S_c, ID_i must be in the *id* list of D_c. Let W_c be the weight of D_c, create a new point $D_c' = (W_c \cdot D_c - W_i \cdot D_i) / (W_c - W_i)$ with its weight $W_c' = W_c - W_i$, then replace D_c by D_c' in $S(t_c, h)$.

Suppose that the input cluster number is K, we first cluster the complex-points in $S(t_c, h)$ to get K cluster centroids, and then cluster all the weighted points in $S(t_c, h)$ with these centroids to obtain the final result.

III. Empirical Results

All of our experiments are conducted on a PC with Intel Pentium IV processor and 256 MB memory, which runs Windows XP professional operating system. The data set is generated by the DataGenerator 2.0.

At the moment T =100, we analyze points in the two time windows W (100, 100) and W (100, 40) respectively, with the stream speed = 1000/*s*. The quality of clustering results is usually measured using the sum of square distance (SSQ). Figure 1 shows the comparison on SSQ between the clustering results of the two algorithms and the real clusters. *AGCluStream* and *CluStream* present similar performances in the best results of each algorithm, but *AGCluStream* possesses better stability. In our experiment for testing the accuracy of the algorithms identifying the cluster centers, 85.7% of the results of *AGCluStream* fall in an error range of 16.7%, but the proportion for *CluStream* in this error range is only 42.9%.

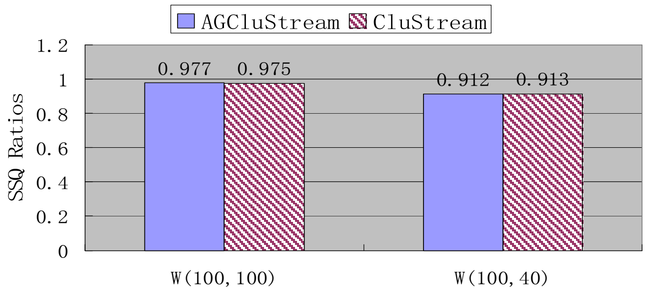

Figure 1 SSQ Ratios of The Real Clusters to the Best Results Of Each Algorithm

The experiment has shown that *AGCluStream* can get a lower time complexity. The iteration times that *AGCluStream* takes for convergence almost crease linearly with the number of data points. We set the stream speed to the maximum value that can be accepted by the algorithm, and figure 2 shows the time costs of *AGCluStream* over each period of the stream.

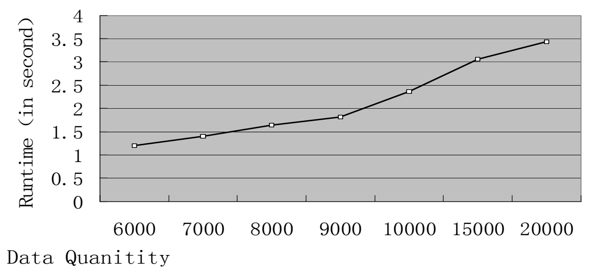

Figure 2 Test On Efficiency

IV. Conclusions and Future Work

In this paper, we have developed a novel data stream clustering algorithm based on the agent swarm intelligence technique. The algorithm avoids dividing the streaming data into chunks so as to maintain the integrity of the global information to get higher-quality clusters. This method needs no initial points and is insensitive to the initialization information. Meanwhile, it allows time-window analysis on streams by the dual-tire structure. Experiment evaluations show the excellent performance of our algorithm in the aspects of effectiveness and efficiency. In future work, we will underline the attenuation strategy in this framework which could reflects the various tend of clusters in streaming data.

References

[1] R. Duda and P. Hart. Pattern Classification and Scene Analysis. J. Wiley and Sons, 197

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

[3] CC Aggarwal, J. Han, J. Wang, and PS Yu. A framework for clustering evolving data streams. In Proceeings of the 29th VLDB conference, 2003

[4] Zhe Wang, Bin Wang, Chunguang Zhou, Xiujuan Xu. Clustering Data Streams On the Two-tier Structure. APWeb 2004. 416-425

[5] Huang Z. Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Mining and Knowledge discovery, 1998,2: 283-304