WINP: A Window-based Incremental and Parallel Clustering Algorithm for Very Large Databases

Zhang Qiang, Zhao Zheng, Sun Zhi wei, Edward daley
School of Electronic Information Engineering, Tianjin University, Tianjin, China
E-mail: zq8622@yahoo.com

Abstract

We introduce a new clustering algorithm called WINP for very large databases. Two different sizes of handling objects were used in WINP to acquire high accuracy and efficiency. WINP creates a window to detect approximate locations of clusters before accurate clustering processing. Clustering on these locations will reduce a lot of computations and get a good performance. WINP is the first algorithm to realize both incremental clustering and distributed parallel clustering. The advantages of our new approach are (1) It is very efficient; (2) It realizes distributed parallel processing and can be run on a number of workstations connected via local area network; (3) It introduces a novel incremental clustering method for new coming data in an already processed database; (4) It is effective in discovering clusters of arbitrary shape; (5) It is not sensitive to noise; and (6) It has some ability to deal with high dimensional points.

1. Introduction

Large amounts of data are being collected in business and scientific domains because of the extensive use of computers and the booming of Internet. But the proportion of useful information is getting smaller and smaller. How to find them out efficiently and accurately becomes more and more difficult, since databases are very large and contain high noise. Being one of the primary data mining tools, clustering has widespread use. Many excellent clustering methods have been developed in this field. The most prominent representatives are partitioning clustering methods such as CLARANS [8]; hierarchical clustering methods such as BIRCH [6]; grid clustering methods such as STING [2] and WAVECLUSTER [5]; and dense clustering methods such as DBSCAN [1]. Each method has its advantages and shortcomings. They are not suitable for processing very large databases.

It is difficult to acquire both high accuracy and efficiency in a clustering algorithm. The two targets always conflict with each other. The key factor of this issue is the size of objects that the clustering algorithm handles. Handling small objects means more work and fine clustering result. In DBSCAN algorithm, a point is looked at as an object. For its small size, DBSCAN achieves a high accuracy but suffers from a low speed. On the other hand, handling big objects means less work and rough results. In STING algorithm, big cells are regarded as objects. Because cells are big and contain many points, STING achieves a high speed and suffers from low accuracy. In theory, a grid based clustering algorithm can achieve a high accuracy by reducing the size of cells. This will cause the number of cells growing rapidly, especially in a high dimensional data space. However, most of the cells are either empty or occupied by isolated points belonging to no clusters. Our analysis showed that the speed of STING would drop rapidly while reducing the size of cells. As its handling objects become small, the performance of STING is much like DBSCAN, losing its efficiency. In short, using only one type of handling objects cannot guarantee accuracy and efficiency at the same time.

To process massive data sets, the power of a single computer is not enough. Parallel clustering is the key technique. It will be highly scalable and low cost to do clustering in a distributed environment. In addition, incremental clustering is another important technique for very large databases. Typically, new data is inserted into databases periodically. Due to the very large size of the databases, it is highly desirable to perform these updates incrementally. But none of the current clustering algorithms we know have abilities of both parallel processing and incremental clustering.

1.1. Contributions of the paper

Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)
1082-3409/05 $20.00 © 2005 IEEE

In this paper we present a new clustering algorithm called WINP (Window-based INcremental and Parallel clustering algorithm) for massive data sets. Two different sizes of objects were used in WINP. A detecting window (the big object) was introduced as a set of cells to speed up clustering. Cells (the small object) can be very dense to fulfill the requirements of accuracy. WINP is the first algorithm to realize both incremental clustering and distributed parallel clustering. This allows us to process much larger data sets than other algorithms that have been reported.

2. Basic WINP algorithm

WINP algorithm introduces a detecting window to integrate the grid method with the dense method. There are three steps in WINP. First the data space is divided into rectangular cells. Each cell sums up the number of points located in it. Second WINP generates a detecting window as a handling object. Because the window is a big object, this step is efficient. Through window detection, WINP finds out the approximate locations of clusters. Most of cells are eliminated in this step, because they are valueless for clustering. Last WINP looks cells as handling objects and searches clusters accurately around the locations from the second step. This step is based on the dense method that can discover arbitrary clusters in a high noise data set.

2.1. Grid space

We divide data space into rectangle cells to form a grid space. The cell's edge length is determined by the target resolution. Because WINP introduces a detecting window, its efficiency is not sensitive to the cell's size. The cell can be very small to acquire a high accuracy. Nonempty cells are numbered depending on their positions in grid space. In this way they are mapped to one-dimensional keys. It is efficient to search nonempty cells with these keys stored in a hash table or a randomized search-tree. The computation complexity of this step is O(N), where N is the number of points in the database. With the grid space on hand, all clustering processing will be done on it. The succeeding computation complexity is only related to nonempty cells' number.

Definition 1: The mass of a cell g, denoted by mass(g), is defined by mass(g)=n, where n is the number of points located in cell g. The mass of a cells' collection is defined by mass($\{g_i\}$)=$\sum_i mass(g_i)$

Definition 2: $\forall\ \varepsilon$>0 (neighborhood radius), The ε-neighborhood of cell g, denoted by N_ε(g), is defined by N_ε(g)=$\{g_i \mid$ |p, p_i|$\le \varepsilon\}$, p being the center of g and p_i being the center of g_i. It also defines g$\in N_\varepsilon$(g).

Definition 3: $\forall\ \varepsilon$>0 and ξ>0 (the minimum mass of a core cell's ε-neighborhood), A nonempty cell g is a core cell, if mass(N_ε(g))$\ge\xi$. This is denoted by core(g, ε, ξ)=true.

Definition 4: Cells p and q are density reachable denoted by p<=>q, if core(p, ε, ξ)=true, core(q, ε, ξ)=true, $\exists$ a sequence $\{g_k \mid$ core(g_k, ε, ξ)=true$\}$ and $g_1\in N_\varepsilon$(p), $g_2\in N_\varepsilon(g_1)$,..., $g_n\in N_\varepsilon(g_{n-1})$, q$\in N_\varepsilon(g_n)$. A nonempty set F is a frame, when it is composed of all the cells that every two of them are density reachable.

Definition 5: A cluster C with respect to frame F is a set of cells, is defined by C={g| mass(g)>0, $\exists$ p$\in$ F lets g$\in N_\varepsilon$(p)}.

2.2. Detecting window

A cluster is defined by a frame, and a frame is built up with core cells. So an area occupied by a cluster is the location of existing core cells. Finding out core cells is the key to searching clusters. The identification of core cells is a time-consuming operation. The computation complexity is O(M*E), where M is the total number of nonempty cells in grid space and E is the average number of nonempty cells in a ε-neighborhood area. M will be very large when the noise percent is high. E rises fast as dimension increases. Because the volume of the ε-neighborhood is growing according to the cell's dimension-th power of ε. WINP algorithm introduces two sizes of handling objects. The big object (detecting window) is used to find out the approximate locations of core cells efficiently. Then the small objects (cells) are used to search clusters accurately only around those locations. In this way WINP algorithm achieves both efficiency and accuracy.

WINP algorithm generates a rectangle with same dimensions of grid space. We call it a detecting window. It moves in grid space step by step (Figure 1). The step length is equal to its edge length. At every step, it sums up the total mass in the window. If the

Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)
1082-3409/05 $20.00 © 2005 IEEE

sum is larger than ξ, then the nonempty cells in the detecting window are regarded as expecting core cells. A core cell may be not detected as expecting core cell by chance, but lemma 1 shows that any core cell can be found out from a detected core cell in the same cluster. Also no core cells may be detected as expecting core cells, but algorithm 1 will distinguish them.

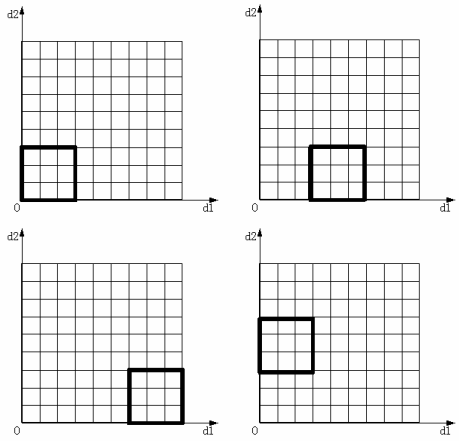

Figure 1: **Movement of detecting window**

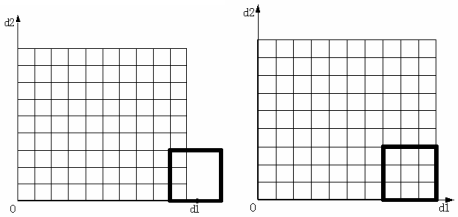

Figure 2: **Movement of detecting window at boundary**

Lemma 1: If F is a frame and set E$\subseteq$F, then { g_k | $\exists$p$\in$ E, let p<=> g_k } = F.

Proof:

(1)From Definition 4 ("all") we know F $\supseteq$ { g_k | $\exists$p$\in$ E, let g_k <=>p}.

(2) $\forall$ e$\in$ E, E$\subseteq$F $\therefore$ e$\in$ F. for $\forall$ f$\in$ F, From Definition 4 we know f<=>e, $\therefore$ f$\in$ { g_k | $\exists$p$\in$ E and let g_k <=>p} $\therefore$ F$\subseteq$ { g_k | $\exists$p$\in$ E, let g_k <=>p}.

From (1) (2) we get { g_k | $\exists$p$\in$ E, let g_k <=>p} = F.

The edge length of the detecting window is defined by L_w=[2*ε / L_c]* L_c, where L_c is the edge length of a cell and ε is neighborhood radius ([a] means the maximum integer smaller than a). So the window's edge length is divisible by the cell's length. The movement of the detecting window is shown in Figure 1. If the window is moving out of the grid space, the window should return back several cell lengths to let its edge snap back to the grid boundary (Figure 2). The computation complexity of window detecting is O(M), where M is the number of nonempty cells in grid space. When this step is finished, the expecting core cells are found out. The clusters should be located around these expecting core cells.

2.3. Finding accurate clusters

In this step, WINP algorithm uses small objects (cells) to search the accurate locations of clusters. It is based on dense method, which was chosen for its ability of discovering arbitrary shape clusters with high accuracy. Searching only around expecting core cells reduces much of the computation and makes this step very fast. In algorithm 1, Clustering function is the main function. IsaCoreCell function tests a cell to determine whether it is a core cell according to the parameters ε and ξ.

Algorithm 1 (clustering in a low-dimensional space)

Function IsaCoreCell (g, ε , ξ)

//Test whether cell g is a core cell

(1) Sum up the mass of all nonempty cells in ε -neighborhood area of g.

(2) IF sum< ξ return false. //not a core cell

(3) Else return true.

Function Clustering (Core)

//Core is the set of expecting core cells

(1) If Core==null then terminate, else remove a cell g from it.

(2) If g is already marked, then go back to 1.

(3) Test whether g is a core cell.

(4) If g is not a core cell, then mark it with NOISE and go back to 1.

(5) If g is a core cell, then put all nonempty cells that are located in g's ε -neighborhood area into a set E (pick g out of E).

(6) Generate a new cluster C, mark g as a member of C_F (the frame of C), mark all the cells in E as members of C.

(7) If E==null, then go back to 1, else remove a cell h from it.

(8) Test whether h is a core cell.

(9) If h is not a core cell, then go back to 7.

(10) If h is a core cell, put all nonempty and not marked cells that are located in h's ε -neighborhood area into set E (pick h out of E).

(11) Mark h as a member of C_F. Mark all nonempty cells that are located in h's ε -neighborhood area as members of C.

(12) Go back to 7.

The computation complexity of Algorithm 1 is O(F*E), where F is the number of cells occupied by clusters and E is the average number of nonempty cells in an ε-neighborhood area. It is much smaller than O(M*E), when the detecting window is not introduced. M is the number of nonempty cells in grid space, and M>>F. The overall computation complexity of WINP algorithm is made up of grid creating O(N), window detecting O(M), and accurate clusters finding O(F*E), summed up to O(N+M+F*E), where N is the point's number in the database. In a low dimensional space, N>>M and N>>F*E, so the complexity of WINP is about O(N).

Algorithm 1 works well in a low dimensional grid space. But in a high dimensional grid space, the ε-neighborhood area will be big and makes F*E very large. So a compromise between accuracy and efficiency should be made. If expecting core cells are assumed to be core cells without an additional test, the clustering processing will be speeded up. Under this assumption the computation complexity has no relation with dimensions any more. The accuracy of big clusters is affected little, but small clusters are affected much. In order to resolve this problem, the small clusters will be reprocessed by Algorithm 1. Algorithm 2 introduces two parameters to control the tradeoff between accuracy and efficiency in a high dimensional grid space. Parameter COREMAX is for efficient and CLUSTERMIN is for accuracy.

Algorithm 2 (clustering in a high dimensions space)

(1) If $\frac{NumberofExpectingCoreCells}{NumberofNonemptyCells}$ > COREMAX, then jump to 4.
(2) Use Algorithm 1 to process.
(3) Terminate.
(4) If Core ==null, then jump to 11, else remove a cell g from Core.
//Core is the set of expecting core cells
(5) If g has been already marked, then go back to 4.
(6) Else generate a new cluster C, mark g as a member of C_F (the frame of C).
(7) Mark all nonempty cells that are located in g's ε-neighborhood area as members of C. Put all expecting core cells (in g's ε-neighborhood area) into set E (pick g out of E), and mark them as a member of C_F.
(8) If E is null, then go back to 4, else remove a cell h from it.
(9) Mark all nonempty and not marked cells that are located in h's ε-neighborhood area as members of C. Put all expecting core cells (in h's ε-neighborhood area) into set E (pick h out of E), and mark them as members of C_F.
(10) Go back to 8.
(11) If $\frac{N_i}{F}$ >CLUSTERMIN, (N_i is the number of expecting core cells in cluster i and F is the number of nonempty cells), then use Algorithm 1 to process cluster i.

The computation complexity of Algorithm 2 is O(F+ F_{small} *E), where F is the number of cells occupied by clusters and F_{small} is the number of cells occupied by small clusters. It is much faster than O(F*E) of Algorithm 1, for F>> F_{small}. The overall complexity of WINP in a high dimensional space is O(N+M+F+ F_{small} *E). Users can control efficiency and accuracy by the selection of COREMAX and CLUSTERMIN.

2.4. The selection of parameters

Two parameters ε (neighborhood radius) and ξ (the minimum mass of a core cell's ε-neighborhood) have to be determined before window detecting. According to our experience, It's fit to let ξ =3*dimension for very large database with high noise. We define ξ distance (denoted by σ) of a nonempty cell g by σ =min(Δ), $\{\Delta | \text{mass}(N_\Delta(g)) \geq \xi\}$. Algorithm 3 shows how to acquire ε through ξ distance.

Algorithm 3 **(parameter selection)**

(1) Select m (about 1% of total nonempty cells) nonempty cells from grid space at random.
(2) Calculate ξ distance of these cells.
(3) Sort these distances in descending order.
(4) Make a plot of sorted distances. The sorted cells are on the X-axis and their ξ distances are on the Y-axis.
(5) Choose the ξ distance of the first valley point or break point in the plot as parameter ε.

3. Incremental clustering algorithm

As new data is being inserted into databases periodically, the clustering result should be updated as well. Due to the very large size of databases, it is perfect to process these new points incrementally. WINP algorithm presents a new incremental clustering method that takes full advantage of previous clustering results. In order to guarantee the accuracy, WINP algorithm employs a different policy for large insertion and little insertion. Too many points insertion will

change the distribution of original points entirely. Clustering parameters (ε, ξ) should be updated. The previous clusters become valueless under new parameters. But the grid space can be reused. As stated in section 2, the generation of grid space is the bottleneck of clustering. Grid reuse can reduce operation significantly. When the number of inserted points is small, its influence on the distribution of original points can be omitted. Not only grid space but also clusters can be reused. We will merely consider the influence of updated cells on previous clusters. Figure 3 shows four results of updated cells, (a) No influence (b) birth of a new cluster (c) The growth of cluster (d) the merging of clusters.

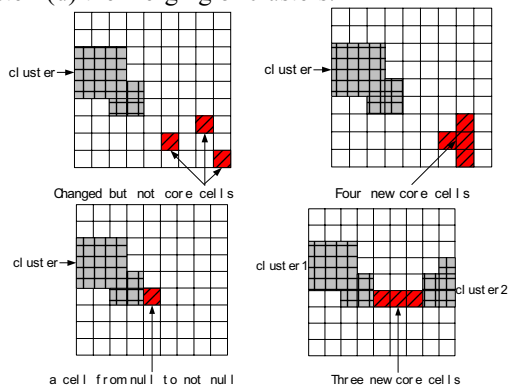

Figure 3: **The influence of update cells on previous clusters**

Algorithm 4 realizes incremental clustering. The basic idea is to check whether an updated cell is rising to a core cell that will cause the birth of a new cluster or clusters merging. Algorithm 4 introduces an equal clusters table to merge clusters efficiently. For example, the item C2=C3=C5 in the table means clusters C2, C3 and C5 are parts of the same cluster.

Algorithm 4 (incremental clustering)

(1) Put inserted points into grid space through adding the mass of corresponding cells. Add updated label to them.

(2) If ΔN>5%N (ΔN is the number of inserted points and N is the number of original total points), then use Algorithm 2 to process and then terminate.

(3) If the number of updated cells is larger than the number of original core cells, then use Algorithm 2 to process and then terminate.

(4) If there is no updated cell, then jump to 14, else pick up an updated cell h and remove the updated label from it.

(5) If h is a core cell in previous clustering, then go back to 4.

(6) Else test whether h becomes a core cell.

(7) If h is not a core cell and is a member of a cluster already, then go back to 4.

(8) If h is not a core cell and belongs to no clusters, then find out core cells (by checking their labels) in its ε-neighborhood area and mark h belonging to the same cluster as the found core cell.

(9) If h is a core cell, then search core cells (by checking their labels) in its ε-neighborhood area.

(10) If there is no core cell, then generate a new cluster C and mark h belonging to the frame of C. Mark the cells in its ε-neighborhood area that belong to no clusters and are not empty as members of C.

(11) If there are core cells in h's ε-neighborhood area, test whether they belong to the same clusters.

(12) If these core cells belong to the same clusters U, then mark h belonging to the frame of U. Mark the cells in its ε-neighborhood area that belong to no clusters and are not empty as members of U. Go back to 4.

(13) If these core cells belong to clusters $C_i, C_j \ldots C_x$, then mark h belonging to the frame of C_i. Mark the cells in its ε-neighborhood area that belong to no clusters and are not empty as members of C_i. Put the item $C_i = C_j = \ldots = C_x$ in the equal clusters table. Go back to 4.

(14) Remove duplicate items from the equal clusters table. Merge the equal items. Then terminate.

When the inserted points are more than 5% of the original points' number, only grid space is reused. The computation complexity of Algorithm 4 is O(ΔN+M+F+F_{small}*E). Compared with Algorithm 2, it saves the time of O(N). When the inserted points are within 5%, both grid and clusters are reused. The incremental clustering algorithm gets the biggest speed-up factor with the computation complexity of O(ΔN+I*E), I being the number of updated cells.

4. The distributed clustering algorithm

WINP realized parallel clustering for distributed environment. It is a shared-nothing architecture shown in Figure 4. Every unit has its processor, memory, and disk. They are connected via local area network (LAN). The same program runs on every processor but processes different parts of the data set. Processors communicate with each other through LAN. The communication latencies would waste much time. Exchanges of a few large messages are more efficient than many small messages.

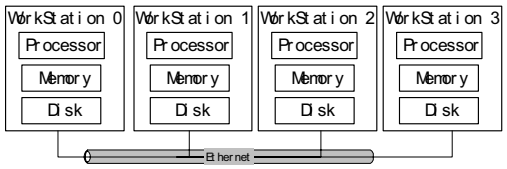

Figure 4: **Shared-nothing architecture**

First, each processor starts to build the grid space with the data of size N/n on its local disk, where N is the total number of points and n is the number of processors. Here we assume that the data is available on the local disk of a processor. This step needs no communication. Then, the processors exchange grid information with each other. Each processor selects 1/n of global grid space as its clustering space, and acquires the cells mass in this space from other processors. Next, each processor will do clustering on its own grid space independently. Last, the processors whose clustering spaces are connected exchange the information of core cells around the common boundary. Through these core cells, the connected clusters will be merged. Figure 5 shows the merging of clusters at the boundary. Lemma 2 tells us when the two clusters should be merged. Algorithm 5 shows the detail of parallel clustering. To merge clusters efficiently, an equal clusters table (section 3) is used in Algorithm 4.

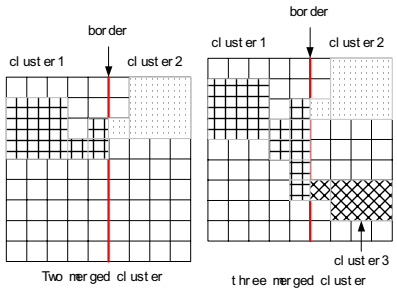

Figure 5: **Clusters' merging at boundary**

Lemma 2: Two clusters C_1 C_2, and F_1 F_2 are their frames. If $\exists$ $g_1 \in F_1$, $g_2 \in F_2$ let $g_1 \in N_\varepsilon(g_2)$, then $C_1 = C_2$.

Proof: $g_1 \in N_\varepsilon(g_2)$ and $g_2 \in F_2$ $\therefore g_1 \in F_2$ (from definition 4). $\forall$ g$\in F_1$ $g_1 \in F_1$ $\therefore$ g<=> g_1 $g_1 \in F_2$ $\therefore$ g$\in F_2$ $\therefore F_1 \subseteq F_2$ $\therefore C_1 \subseteq C_2$.

In the same way, we can get $C_1 \supseteq C_2$. So $C_1 = C_2$.

Algorithm 5 (parallel clustering)

(1) Processor i (i=1,2...n) reads data block N_i to generate grid blocks $G_{i,1}, G_{i,2} \ldots G_{i,n}$.

(2) Exchange grid blocks among processors. At last processor i get $G_{1,i}, G_{2,i} \ldots G_{n,i}$.

(3) Processor i makes $G_{1,i}, G_{2,i} \ldots G_{n,i}$ together and gets its own grid space $\widetilde{G}_i$ by summing up the mass of same cells.

(4) Processor i uses Algorithm 2 to do clustering in grid space $\widetilde{G}_i$ and gets clusters $C_{i,1}, C_{i,2}, C_{i,p}$.

(5) Processor i finds the processors sharing a grid boundary with i and uses Algorithm 6 to merge the clusters.

(6) Processor Q (only a computer does this work) collects all equal clusters tables and creates a global equal clusters table.

(7) Terminate.

Algorithm 6 (clusters merging for parallel clustering)

// Processor i and j share a boundary X, Processor i does the following process:

(1) Find out all core cells within ε distance of X, getting sets $\{E_{i,1}\}$, $\{E_{i,2}\} \ldots \{E_{i,k}\}$ belonging to clusters $C_{i,1}, C_{i,2} \ldots C_{i,k}$. The total number of core cells around the boundary is S_i.

(2) Compare S_i with S_j, let the small one send its core cells around the boundary to the big one for reducing the communication. Here we assume $S_i > S_j$.

(3) Processor i gets core cells sets $\{E_{j,1}\}$, $\{E_{j,2}\} \ldots \{E_{j,m}\}$ from j.

(4) If $\{E_{j,r}\}$ (r=1,2...m) is null, then process $\{E_{j,r+1}\}$, else remove a cell g from it. If all sets have been processed, then jump to 8.

(5) In g's ε-neighborhood area, check whether there are core cells in the clusters of processor i.

(6) If core cells in clusters $C_{i,q1}, C_{i,q2} \ldots C_{i,qm}$ are found, then generate an item $C_{i,q1} = C_{i,q2} = \ldots = C_{i,qm} = C_{j,r}$ in its equal clusters table.

(7) Go back to 4.

(8) Remove duplicate items from the equal clusters table. Merge the equal items.

(9) Terminate.

The overhead of parallel clustering algorithm is composed of grid bulk exchanging (steps 2, 3 in Algorithm 5) and clusters merging (Algorithm 6). Because the number of core cells around the grid boundary is very small, the overhead is mainly caused by grid bulk exchanging. To save time only nonempty

cells are transmitted. The algorithm efficiency is defined by $\eta = 1 - \frac{TimeofGridExchanging + TimeofClustersMerging}{TotalTime}$. In parallel test (section 5.4), η is 85.4%.

5. Performance evaluation

5.1. Accuracy test

The first data set is to test whether WINP has the ability to distinguish clusters with different sizes. Figure 6a shows three clusters of different sizes, and with some noise. Figure 6b is the clustering result (the grid density is 100*130), (a) Data set (b) Clustering Result.

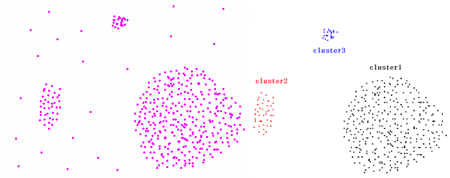

Figure 6: **Test on clusters of different sizes**

The second data set has three arbitrary shape clusters as shown in Figure 7a. They are found in Figure 7b (the grid density is 100*170). This shows that WINP is effective in discovering clusters of arbitrary shape.

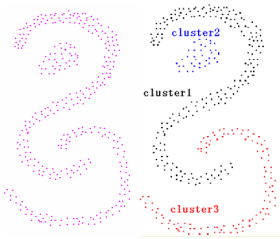

Figure 7: **Test on clusters of different shape**

5.2. Efficiency test of single workstation

We created a data generator to produce data sets used in our experiments (sections 5.2, 5.3 and 5.4). Our data sets can have arbitrary cluster shape. In section 5.2 the number of clusters in the data set is 6. The noise level is 20%. The dimensions are 2.

In the first test, we compare the efficiency of WINP with DBSCAN (indexed by R* tree). The workstation is HASEE 715D. The program language is JAVA. The grid density is 1000*1000. The result is shown in Table 1.

Table 1: **Comparative Operation time (second) in a 2 dimensional space (grid density 1000*1000)**

	Points' number	100,000	300,000	500,000
WINP	Create grid	0.471	1.272	2.113
	Clustering	0.801	0.809	0.922
	Total	1.272	2.079	3.035
DBSCAN	Create R*tree	39.237	161.773	290.228
	Clustering	29.984	415.016	987.62
	Total	69.221	576.789	1277.848

In the second test, larger data sets are used. The test environment is the same as above. The grid density is 4000*4000. The result is shown in Table 2. The total processing time has an approximate linear relation with the number of points.

Table 2: **WINP's Operation time (second) in a 2 dimensional space on one workstation (grid density 4000*4000)**

Points' number	2,000,000	4,000,000	8,000,000	16,000,000	32,000,000
Create grid	8.406	17.767	37.143	77.073	167.367
Clustering	2.327	2.344	2.484	2.501	2.532
Total	10.733	20.211	39.627	79.574	169.899

5.3. Incremental clustering test

The test data sets are divided into two groups. In group one, the percent of incremental points is 1% (less than 5%). In group two, the percent of incremental points is 5.5% (more than 5%). The dimensions of the data sets are 2. Grid density is 4000*4000. Figure 8 depicts speed-up factors of two groups. X-axis is the number of points in the original data sets. Y-axis is the speed-up factors, which are defined, by $\frac{T_{reclustering}}{T_{incremental}}$, where $T_{reclustering}$ is the time of fully reclustering, and $T_{incremental}$ is the time of incremental clustering (Algorithm 4).

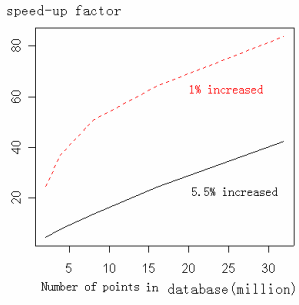

Figure 8: **Speed-up factors of incremental clustering in a 2 dimensional space**

5.4. Distributed clustering test

The number of points in the test data set is $1.2*10^8$. The dimensions are 5. There are 7 clusters and 20% noise in the data set. Four workstations connected via 100M Ethernet do parallel clustering. Each station has a Pentium processor (1.2GHz) and 256M memory. Programming language is JAVA. All the clusters are found. The detail clustering time is shown in Table 3. Figure 9 shows the speed-up factor of multiprocessors.

Table 3: **Operation time (in seconds) for 1.2* records in a 5 dimensional space on 4 workstations**

Create grid	Exchange bulk cells	Clust ering	Connect clusters	Total	η
417	98	174	3	692	85.4%

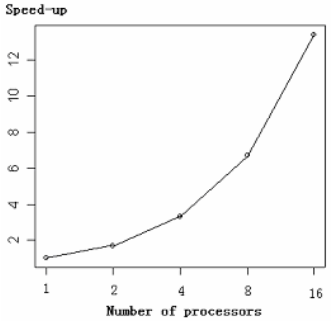

Figure 9: **Speed-up factors of multiprocessors**

6. Conclusion

In this paper we present WINP, a window-based incremental and parallel clustering algorithm for massive data sets. WINP algorithm introduces two sizes of handling objects in clusters searching. A detecting window is used to seek the approximate locations of clusters. Cell is used to find out the clusters exactly around these locations. WINP also realizes incremental clustering and distributed parallel clustering to process massive data sets.

Performance evaluation shows that WINP is effective in discovering clusters of arbitrary shape; the computation complexity of WINP is O(N), where N is the number of points in Databases; WINP is much faster than DBSCAN; incremental clustering can speed up processing significantly; parallel clustering can deal with massive data sets in high dimensions. In conclusion, WINP algorithm is an ideal clustering method for very large databases.

7. References

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. KDD'96.

[2] W. Wang, J. Yang, R. Muntz, STING: A Statistical Information Grid Approach to Spatial Data Mining, VLDB'97.

[3] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. SIGMOD'98.

[4] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure, SIGMOD'99.

[5] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. VLDB'98.

[6] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH : an efficient data clustering method for very large databases. SIGMOD'96.

[7] Ng R. T., Han J.: "Efficient and Effective Clustering Methods for Spatial Data Mining", Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, Morgan Kaufmann, 1994, pp. 144-155.

[8] Ng R. T., Han J.: "Efficient and Effective Clustering Methods for Spatial Data Mining", Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, Morgan Kaufmann, 1994, pp. 144-155.

[9] G., HAN, E.-H., and KUMAR, V. 1999a. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling, COMPUTER, 32, 68-75.

[10] A. Hinneburg, D.A. Keim, An Efficient Approach to Clustering in Large Multimedia Databases with Noise, Proc. 4th Int. Conf. on nowledge Discovery and Data Mining, 1998.