GenIc: A Single Pass
Generalized Incremental Algorithm for Clustering

Chetan Gupta[*]
Department Of Mathematics, Statistics and Computer Science
University of Illinois at Chicago

Robert Grossman[†]
National Center For Data Mining
University of Illinois at Chicago
and Open Data Partners

This is a draft of a paper from the 2004 SIAM International Conference on Data Mining (SDM 04).

Abstract

In this paper we introduce a new single pass clustering algorithm called GenIc designed with the objective of having low overall cost. We examine some of the properties of GenIc and compare it to windowed k-means. We also study its performance using experimental data sets obtained from network monitoring.

1 Introduction

Developing data mining algorithms for streaming data has emerged as an important problem. For streaming data, the assumption is that the data records can be examined only once. More precisely, given n data records, we would like algorithms with $O(n)$ time complexity and $O(1)$ space complexity.

In this paper we introduce a new algorithm called GenIc, which is a single pass **Gen**eralized **Inc**remental algorithm for clustering. We also describe simulation results using GenIc, as well as experimental results applying GenIC to network monitoring data.

Perhaps the most common clustering streaming data is a windowed version of the k-means algorithms [6]. One of our main applications is to cluster large amounts of distributed network monitoring data. Simply integrating the distributed data in real time is expensive and for this reason we were interested in low cost clustering algorithms for streaming data. As we describe in the section on experimental results, GenIc is significantly faster than windowed k-means.

2 Related Work

There is a large research literature on clustering algorithms [4],[15],[16]. Perhaps the most common approach to clustering is to minimize the sums of the squares (SSQ) of the distances between data points and the centers of clusters. The K-means algorithm is a popular heuristic for finding a solution which is a local minimum to this problem since it always converges in a finite number of steps. A wide variety of different algorithms have been proposed to minimize the SSQ, a wide variety of criteria other than SSQ have been proposed, and a wide variety of clustering algorithms adapted to specific applications have been studied.

Another approach is to view the data set as a collection of points and then in a greedy fashion choose the best way to divide the points into two groups. Proceeding inductively in this way leads to what are called hierarchical clustering algorithms and to tree like structures. Similarly, one can work from the bottom up and decide which two best points or clusters to combine. Again either SSQ or either metrics can be used for these types of algorithms.

Recently, clustering algorithms for large data sets and streaming data sets have been developed. The assumption when clustering large data sets is that the data is so large that it resides on secondary storage. BIRCH (Balanced Iterative Reducing and Clustering) clusters large data sets by using specialized trees structures to work with out of memory data [23]. CLARANS (Clustering Large Applications based on RANdom Search) identifies candidate cluster centroids through analysis of repeated random samples of the original data [20].

[*] gupta@math.uic.edu
[†] grossman@uic.edu

The assumption when clustering streaming data is that a data record can be examined only once [12]. One approach to clustering streaming data is to perform local clustering on the data that fits in the local memory. The k-mediod problem is a variation of k-means. Guha, et. al. [13] have presented streaming algorithms using local clustering to solve the k-mediod problem. They use a procedure called LOCALSEARCH. The data stream is divided into chunks and a LOCALSEARCH is done on a weighted representation of the chunks. This is done for all such chunks. Additional work done by Bradley, et. al. [5] and the improvements by Farnstrom [10] repeatedly take k-weighted centers (initially chosen at random with a weight of 1) with as much data that can fit in the main memory to compute k cluster centers. The new centers are weighted by the number of points assigned. The data in the memory is discarded and the process is repeated again until all of the data has streamed through.

GenIc is also a windowed algorithms for clustering streaming data. Unlike the windowed algorithms of Guha et. al. [13] Bradley et. al. [5], and Franstorm [10], GenIC employs incremental clustering. GenIc also uses evolutionary techniques to improve its search for global optimal solutions to the SSQ problem required to find cluster centers. Each window in GenIc is viewed as generation for this purpose.

Incremental clustering has also been used for clustering streaming data and large data sets by others [4],[8],[17]. Hartigan's leader algorithm [15] uses a one-pass, incremental approach. If a point lies within a threshold it is added to the cluster. Otherwise it is used to make a new cluster center. Charikar, et. al. [8] gave theoretical limits for some existing incremental hierarchical clustering algorithms and suggested new approaches. Unlike the incremental clustering described by Hartigan [15] and Charikar [8], GenIC updates each center with each new data point and merges clusters only at the end of a generation (i.e. window of data). DIGNET [21],[22] uses a similar approach in that the cluster centers are pushed or pulled towards a new point, with each new data point; however, the approach used by DIGNET depends strongly on the data ordering. ART [7] is similar to DIGNET. Other incremental clustering algorithms include the Cobweb system of Fisher [11].

Evolutionary techniques have been used in clustering for sometime. They have been used as part of an optimization to reduce overall sum of the squares error by Babu et. al. [2],[3]. Another example is the GGA (Genetically Guided Algorithm) used for fuzzy and hard k-means by Hall et al. [14]. Simulated annealing techniques have likewise been used [18] in clustering. Cowgill has also introduced a evolutionary clustering algorithm [9]. GenIC is different than these techniques since it views each consecutive set of n data points as a generation rather than looking at all the points before applying evolutionary techniques. For this reason, GenIc is better suited to streaming data.

3 The GenIc Algorithm

In this section, we describe the GenIc algorithm in detail.

An *incremental clustering algorithm* maintains a list of k centers. Each new point which is presented is either i) assigned to one of the clusters or ii) is used to start a new cluster and two of the existing clusters are merged.

By a *generalized incremental algorithm*, we mean an incremental algorithm which can move a center in the list of centers using a weighted sum of the existing center and the new point presented.

Our approach is to divide the stream into chunks or windows as is common with streaming algorithms. We view each chunk of n data points as a generation and think of the "fitness" of a center as being measured by the number of points assigned to it. In general, the fittest centers survive to the next generation, but occasionally new centers are selected and old centers are killed off.

Here are the details:

1. **Select parameters.**
   - Fix the number of centers k.
   - Fix the number of initial points m.
   - Fix the size of a generation n.
2. **Initialize**
   - Select m points, $c_1, \ldots, c_m$ to be the initial candidate centers.
   - Assign a weight of $w_i = 1$ to each of these candidate centers.
3. **Incremental Clustering.** For each subsequent data point p in the stream: do
   - $Count = Count + 1$.
   - Find the nearest candidate center c_i to the point p
   - Move the nearest candidate center using the formula:
     $$c_i = \frac{(w_i * c_i + p)}{(w_i + 1)}$$
   - Increment the corresponding weight
     $$w_i = w_i + 1$$

- When $Count \equiv 0 \ mod \ n$, goto Step 4.

4. **Generational Update of Candidate Centers.** When $Count$ equals n, $2n$, $3n$, $\ldots$, for every center c_i in the list L of centers, do:

    - Calculate its probability of survival using the formula
    $$p_i = \frac{w_i}{\sum_{i=1}^{m} w_i}$$
    - Select a random number δ uniformly from $[0, 1]$. If $p_i > \delta$, retain the center c_i in the list L of centers and use it in the next generation of n points.
    - If $p_i < \delta$, kill the center c_i and select a new random point from the current generation to replace it as a center in the list L of centers.
    - Set the weight $w_i = 1$ back to one. Although some of the points in the stream will be implicitly assigned to other centers now, we do not use this information to update any of the other existing weights.
    - Goto step 3 and continue processing the input stream.

5. **Calculate Final Clusters.** The list L contains the m centers. These m centers can be grouped into the final k centers based on their Euclidean distances.

Note that every point can be assigned to a cluster center during the incremental clustering phase of the algorithm. This information can be used in the final phase of the algorithm which computes the final clusters to assign each point to one of the final clusters. For this to work, those points assigned to a cluster center which is dropped can be reassigned to the closest cluster at the end of each generation.

We found that taking m to be a small multiple of k, say $m = 3, 4, 5$ seem to produce the best results.

For our tests we used a slight variant of this algorithm. We worked with a list L of length $2m$. In Step 3, only the first half of the list L was used when testing for the nearest candidate center. In Step 4, we tested the first half of the list to decide whether to drop a center. If the center was dropped in this test, we then tested the second half of the list L to see whether its weight was higher than the lowest weight in the second half. If so, then it was added to the second half of the list L, replacing the center with lowest weight.

For some of the applications, it is useful to update the list L in a different way: when a center say A, is “killed” we look to see if it lies within a certain distance from one of the existing centers in the list L. If it lies within this distance, a new center is created and its coordinates are the weighted mean of the two. The weight of the new center is the sum of the two initial centers. If the center A does not lie within a certain distance and if its weight is greater than the lightest weight on the list, it will be added to the list.

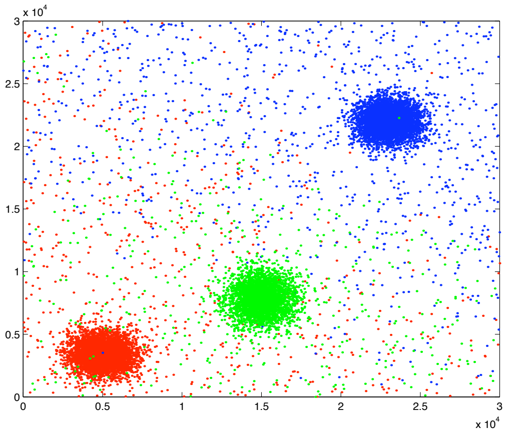

Figure 1: 2-d view of Clustering with GenIc.

4 Experimental Results

We conducted a series of experiments with simulated data using GenIc. Several different data distributions used, including random, normal, Poisson and Cauchy, as well as a variety of different parameters. For these experiments, we randomly chose the centers, distributed data around the centers using the chosen distribution, and then distributed data randomly in the space. Some typical results are shown in Figures 1 and 2.

Accurately identifying cluster centers. In the first series of simulations, we created approximately 50 data sets containing data points distributed around three clusters, as well as other randomly place points, and used GenIc to try to identify the centers of the clusters. The results for some of these experiments are contained in Table 1. For every simulation, the upper row contains the true centers, while the lower row contains the centers as calculated by GenIc.

In most cases, the difference between the calculated centers and the true centers was less than one percent. Occasionally, anomalous centers were identified. This happened more often for points distributed using the Cauchy distribution, when the clusters overlapped, or when one cluster contained almost all of the points.

Quality of clusters. In the second series of simulations, we compared the quality of the clusters by comparing the SSQ of errors of GenIc and windowed k-

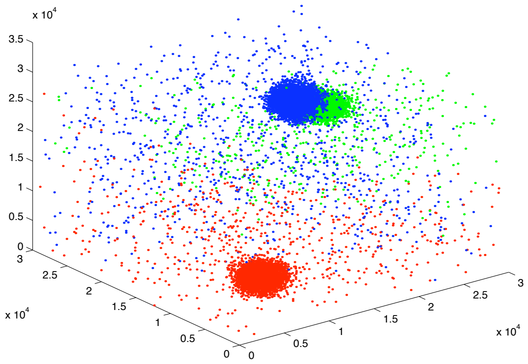

Figure 2: 3-d view of Clustering with GenIc

Exp. No.	Cluster1	Cluster2	Cluster3
1	5000,3400,7600 5092,3404,7667	15000,7800,30000 14884,8167,29747	23000,22000,20000 22823,22039,20084
2	5000,3400,7600 5080,3698,7687	15000,7800,30000 14932,7634,29572	23000,22000,20000 22792,21831,19766
3	5000,3400,7600 5185,3692,7635	15000,7800,30000 14895,7779,30017	23000,22000,20000 23102,22199,19835
4	15000,13400,17600 14645,13210,17064	15000,7800,30000 14726,7803,29942	23000,22000,20000 23153,22066,20044
5	15000,13400,17600 15044,13349,17596	1111,2222,3333 1094,2281,3304	23000,22000,20000 23347,21797,19483
6	6781,9989,10600 6738,9859,10818	4561,8976,7899 4449,9076,8050	23000,22000,20000 22902,21863,19755
7	16781,9989,10600 15114,9737,10149	4561,18976,7899 4664,18875,8105	25000,6780,11232 25324,6884,11307
8	16781,9989,10600 15932,9664,10304	4561,18976,7899 4792,19498,8112	25000,6780,11232 24908,6805,11281
9	16781,9989,10600 16798,10065,10599	9561,18976,7899 9499,19013,7914	25000,8780,11232 25001,8864,11137
10	16781,9989,10600 16203,9580,11198	9561,18976,7899 9399,19549,7883	25000,8780,11232 24698,8886,10872
11	5000,3400,7600 4997,3497,7594	15000,7800,30000 15124,7731,29879	23000,22000,20000 22625,22115,20087
12	5000,3400,7600 5116,3458,7582	15000,7800,30000 14896,7794,29771	23000,22000,20000 23022,21979,19989
13	5000,3400,7600 6957,4223,11811	15000,7800,30000 15163,7699,29913	23000,22000,20000 23039,22027,19903
14	5000,3400,7600 6908,22879,6721	15000,7800,30000 14719,7710,29212	23000,22000,20000 23016,21073,18908
15	5000,3400,7600 5611,4021,7516	15000,7800,30000 14560,7922,27874	23000,22000,20000 22993,22007,20055
16	5000,3400,7600 5655,4434,8596	15000,7800,30000 16018,7492,28847	23000,22000,20000 23006,22074,20251
17	5000,3400,7600 5148,3590,7550	15000,7800,30000 0,0,0	23000,22000,20000 0,0,0
18	5000,3400,7600 5041,3433,7596	15000,7800,30000 14955,7861,29996	23000,22000,20000 23264,22177,20262
19	15000,3400,17600 14993,3487,17544	15000,17800,30000 15080,17836,29883	23000,22000,20000 15080,17836,29883
20	5000,3400,7600 5033,3400,7646	15000,7800,30000 15102,7756,29885	23000,22000,20000 22969,22003,19963
21	5000,3400,7600 0,0,0	15000,7800,30000 0,0,0	23000,22000,20000 23014,22133,19987
22	5000,3400,7600 5251,4268,6499	15000,7800,30000 14724,7639,28289	23000,22000,20000 23199,21920,20014
23	14000,5400,2800 14071,5405,2841	15000,7800,30000 15021,7778,29965	23000,22000,20000 23020,21999,20000
24	14000,5400,2800 14491,6743,28764	15000,7800,30000 14491,6743,28764	23000,22000,20000 22861,21798,20216
25	4000,15400,28000 3783,15893,27742	15000,7800,3000 14111,6875,3204	13000,6000,2900 12282,5951,3414
26	24000,5400,8000 24056,5421,8095	15000,7800,3000 14975,7925,3041	3000,6000,2900 3008,6001,2910

Table 1: Simulation Results for test for finding cluster centers using GenIc

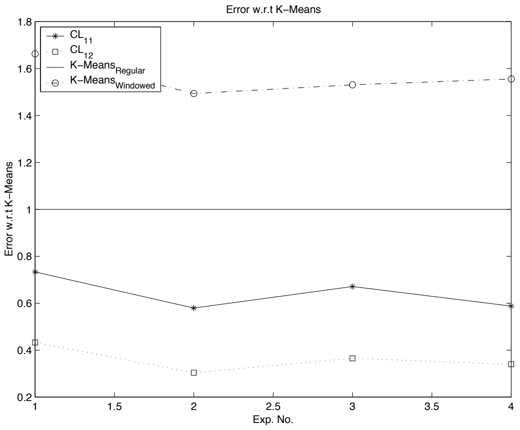

Figure 4: Comparing the ratio of errors between two variants of GenIc and windowed k-means vs. k-means using data as suggested in [23].

means to the SSQ of the errors of standard k-means. For this series, we created 24 data sets: 8 each using the Normal, Cauchy and Poisson distributions. Each set was comprised of 50000 points in three dimensions and with $k = 3$ cluster centers. Each center had a varying number of points assigned to it and there was up to 10 percent noise in each data set. The results can be seen in Figure 3, which compares the ratio of error of an algorithm over the error of k-means. GenIc generally has less error than windowed k-means, and sometimes has less error than k-means.

Changing the number of clusters. For the third series of experiments, we examined the effect of the number of clusters on GenIc. We created four data sets as suggested by the BIRCH paper [23]. There were $k = 100$ centers distributed on a two-dimensional grid. There was no noise and all of the centers had an equal number of points assigned to them. The total number of points for each set was 150000. The results are shown in Figure 4. Again, we used the error of k-means as the standard and plotted the ratio of an algorithm error to the k-means error.

Effect of initial starting points. In this series of experiments we tried to understand how the effect of starting points effected GenIc and k-means. We created 18 data sets of 100000 points in 2 dimensions. We ran experiments using 100 centers, 90 centers, . . ., 10 centers, 9 centers, 8 centers, . . ., and 2 centers. Since in these experiments the error in the k-means depended fairly strongly on the choice of starting points, we ran k-means 10 times and computed both the minimum error and a trimmed error. To calculate the trimmed error,

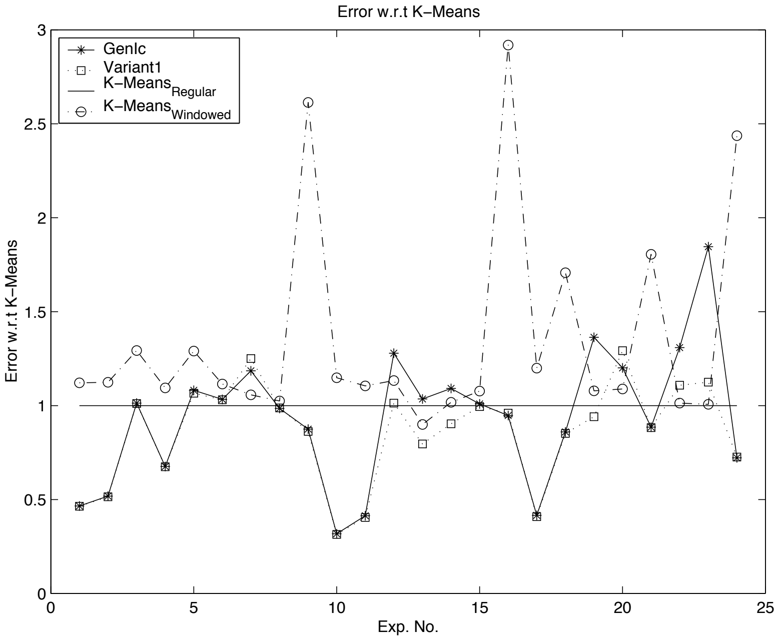

Figure 3: Comparing the ratio of errors between two variants of GenIc and windowed k-means vs. k-means.

we removed the two largest errors and took the mean of the other eight. To calculate the minimum error we took the minimum of the 10 runs.

Figure 5 plots the sum of square distances between the points and the assigned cluster centers. As seen on the graph, the error of GenIc is generally less than the error of k-means when there are a large number of centers. For a sufficiently small number of centers, the minimum error of K-means become less than that of GenIc. On the other hand, the trimmed error is larger than GenIc's, showing that for these experiments k-means is more strongly dependent on the choice of initial center than GenIc.

Independence of data order. The next series of simulations tested the sensitivity of GenIc to the order of the data, a problem faced by some incremental clustering algorithms. Sixteen data sets, grouped into four sets of four, were used. Let A be a set of all points belonging to cluster 1. Let B be the set of all points belonging to cluster 2. Let C be the set of all points belonging to the cluster 3. Each group contained the following four orders: A+B,C;A,B,C;A,B+C;A+B+C, for a total of sixteen data sets. The results are contained in Figure 6 where the relative error between GenIc and windowed k-means as defined by

$$PercentError = 100 \cdot \frac{error_{k-means} - error_{GenIc}}{error_{k-means}}$$

is plotted. Again GenIc does very well.

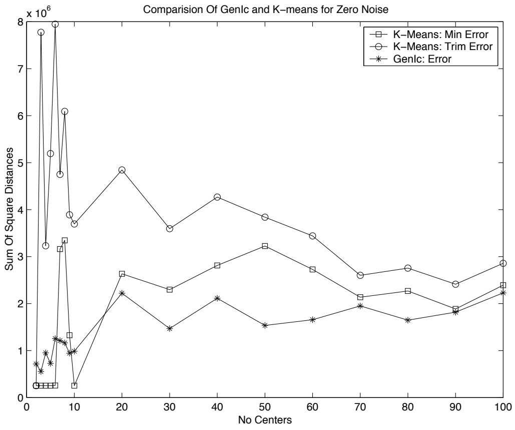

Figure 5: Comparing the ratio of errors between GenIc and k-means. The min error and trimmed error of k-means are plotted.

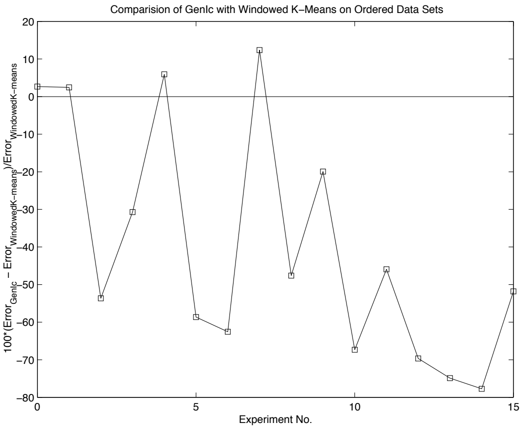

Figure 6: Comparing the effect of data ordering on the error of GenIc and windowed k-means.

GenIc	**Windowed k-means**
338	14,543

Table 2: Time in seconds using GenIc and windowed k-means to find 4 clusters in 3 million four dimensional feature vectors extracted from 3 million Snort alerts.

Clustering experimental data generated by Snort. Our main interest in developing GenIc was to cluster large experimental data sets, such as those which arise in network monitoring, intrusion detection, threat detection, and related applications.

As a simple test of its speed, we compared GenIc to windowed k-means [6] on a data set 3 million four dimensional feature vectors extracted from 3 million Snort alerts. For GenIc, we used $m = 16$ and $n = 2000$. In order for windowed k-means to obtain a realistic execution time, the window size was set at 100000 points, and for every window, up to 30 iterations were permitted.

5 Summary and Discussion

There is no end to the creation of new clustering algorithms, just a weariness of the flesh. Perhaps the only justification that can be provided for a new clustering algorithm, such as GenIc, is that we have found it useful in applications, such as the clustering of network data, and have found it to be fast.

Here are some of the reasons that we have found GenIc to be useful in practice.

- GenIc is fast. Indeed in some of our experiments it was over 40 times faster than windowed k-means.

- GenIc usually identifies well separated clusters quite well, as confirmed by our simulation studies.

- GenIc is flexibile in the face of the unknown values of k. It does not need an explicit initial input of k. Indeed, it can predict the value of k at the end of the run. Previously, genetic techniques had been used for improving the value of k itself [19].

- Previous incremental algorithms ART [7],[21],[22] are dependent upon the order in which the data comes. GenIc overcomes this by taking every batch of n points independently of the previous n points. As previously shown, we conducted tests by varying the order of streaming data, taking care that the order was not random.

- GenIc is less affected by the choice of initial centers than k-Means.

Any clustering algorithm faced with certain data sets does the wrong thing and GenIc is no exception. If one of the centers has an overwhelming concentration of points near to it, points around it will occupy all of the m points in our list L, and GenIc will perform poorly. One obvious remedy is to exclude points too near to each other. L.

References

[1] N. Alon, Y. Matias, and M. Szegedy "The Space Complexity of Approximating the Frequency Moments," Journal of Computer and system sciences, 58, 1999.

[2] G. P. Babu, and M. N. Murty, "A Near Optimal Initial Seed Value Selection in K-means Algorithm Using a Genetic Algorithm," Pattern Recogn. Lett. , 14(10), 1993.

[3] G. P. Babu, and M. N. Murty, "Clustering with Evolutionary Strategies," Pattern Recognition, 27(2), 321-329,1994.

[4] P. Berkhin, "Survey Of Clustering Data Mining Techniques," to appear.

[5] P. S. Bradley, U. M. Fayyad, and C. A. Reina, "Scaling Clustering Algorithms to Large Databases," In proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining, New York, NY, Aug 27-31, pages 9-15, 1998.

[6] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-Data Algorithms for High-Quality Clustering," Proceedings of IEEE International Conference on Data Engineering, March 2002.

[7] G. Carpenter, and S. Grossberg ART3: "Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," Neural Networks 3, 129-152. 1990.

[8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, “Incremental Clustering and Dynamic Information Retrieval,” In proceedings of the 29th annual ACM Symposium on Theory of Computing, 1997.

[9] M. C. Cowgill, R. J. Harvey, and L. T. Watson, “A genetic algorithm approach to cluster analysis,” Computational Mathematics and its Applications, Volume 37, 1999, page 99–108.

[10] F. Farnstrom, J. Lewis, and C. Elkan, “True Scalability of Clustering Algorithms,” SIGKDD Explorations, 2000.

[11] D. Fisher, “Knowledge Acquisition via Incremental Conceptual Clustering,” Mach. Learn. 2, 139-172. 513. 1987.

[12] P. B. Gibbons, and Y. Matias, “Synopsis Data Structures for Massive Data Sets,” Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization, A, 1999.

[13] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering Data Streams,” In the Annual Symposium on Foundations of Computer Science, IEEE, 2000.

[14] L. O. Hall, I. B. Ozyurt, and J. C. Bezdek, “Clustering with a Genetically Optimized Approach,” IEEE Transactions on Evolutionary Computation, Volume 3, No. 2, pp. 103-112, 1999.

[15] J. Hartigan, “Clustering Algorithms,” John Wiley and Sons, New York, 1975.

[16] A. K. Jain, R. C. Dubes, “Algorithms for Clustering Data,” New Jersey, Prentice Hall, 1988.

[17] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review,” ACM Computing Surveys, vol. 31, no. 3, pp. 264-323, 1999.

[18] R. W. Klein, and R. C. Dubes, “Experiments in Projection and Clustering by Simulated Annealing,” Pattern Recogn. 22, 213-220, 1989.

[19] C-Y Lee, and E. K. Anotonnson, “Dynamic Partitional Using Evolutionary Strategies,” In Proceedings of the Third Asia-Pacific conference on Simulated Evolution and Learning, Nagoya, Japan, 2000.

[20] R. Ng, and J. Han, “Very Large Data Bases,” In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB’94, Santiago, Chile, Sept.), VLDB Endowment, Berkeley, CA, 144-155. 1994.

[21] S. C. A. Thomopoulus, D. K. Bougoulias, and C. Wann, “Digent: An Unsupervised-Learning Clustering Algorithm for Clustering and Data Fusion,” IEEE transactions on aerospace and electronic systems, 31(1):21-38, 1995.

[22] C-D Wann, and S.C. A. Thomopoulus, “A Comparative Study of Self Organizing Clustering Algorithm Dignet and ART2,” Neural Networks, 10(4) 737-743, 1997.

[23] T. Zhang, R. Ramkrishnan and M. Linvy, “BIRCH: An Efficient Data Clustering Method for Very Large Databases,” SIGMOD Rec. 25, 2, 103-114. 1996.

[24] www.snort.org, The Open Source Network Intrusion Detection System.