Scalable Data Exploration and Novelty Detection

J. Carbonell, E. Fink, C. Jin, C. Gazen - Carnegie Mellon University

J. Mathew, A. Saxena, V. Satish, S. Ananthraman, D. Dietrich, G. Mani - DYNAMiX Technologies

J. Tittle, P. Durbin - ManTech CSEC

NIMD “Grand Finale” PI Meeting, April 2006

1 Introduction

Project ARGUS is focused on helping an analyst explore massive, structured data. This scalable exploration includes exact and partial match queries, monitoring hypotheses and discovery of novel patterns in both static and streaming data. We provide these facilities within the context of an analyst workbench interface called Data Explorer.

The remainder of this paper comprises three sections; a) a brief review of the methodology employed within ARGUS for the detection of novelty within massive data, b) monitoring of streaming data and a synopsis of the research originating out of the CMU team on the incremental aggregation on multiple continuous queries, and, c) the development of an analyst workbench environment, called ARGUS Data Explorer that has been developed by the teams from DYNAMiX and ManTech CSEC. The ARGUS Data Explorer is currently being evaluated through the RDEC environment/process as a precursor to possible deployment into live operating environments.

2 Detecting Novelty in Massive Data

ARGUS follows a hybrid model of new hypothesis formation, where the system offers its discovery of novel, potentially interesting patterns for analyst review, leading to new hypotheses being formed and tracked, or to discarding the novelty as coincidental or uninteresting. For instance, if shipments of multiple multi-use precursor chemicals consistent with the production of nerve agents directed to a new location in a potentially hostile country start at a certain date and were not observed before, a hypothesis that something new, possibly of a nefarious or perilous nature, is being produced at that location needs to be considered. In ARGUS this would be detected as a new emerging cluster distinct from background clusters in a data stream. In contrast, a single potential precursor chemical with a dual medical used shipped to a medical facility in small quantities similar to many such past shipments may not trigger an alert, as it is a habitual, rather than novel, happening.

As a different example consider the outbreak of a disease like SARS, which the medical community was slow to recognize because SARS symptoms clusters were masked by similar common cold or influenza symptoms. In this case we need to detect a change in existing clusters – a much greater percentage of patients do not recover within the expected time frame, for instance. This requires detection of change in the density function of a cluster, rather than the onset of a clear new one – i.e. we need to perform a de-convolution process to detect a new component in a mixture of observations. We believe that the second case may be more common, either though accidental masking (as in SARS) or intentional obfuscation, such as combining legitimate medical facilities with potential bio-weapon research lab or development facility. Note that nefarious terrorist preparatory activity may be intentionally masked as normal activities, but cause differences in the density functions of such activities over time – analogous to SARS masquerading as influenza or severe colds – and detectable by our methods given a sufficient signal-to-noise ratio.

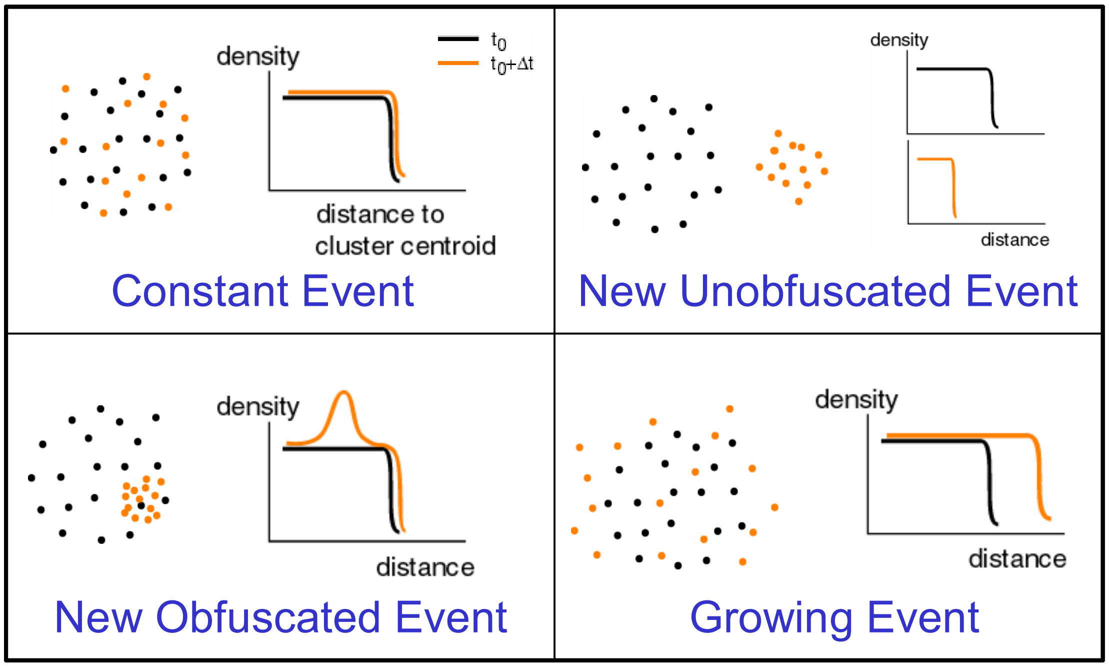
2.1 Clustering, Cluster Density Functions and Novelty Detection

ARGUS focuses on new novelty detection technology built atop clustering techniques, so we use off-the-shelf clustering algorithms to build our background models. In particular, we use the standard leader-follower algorithm and the k-means algorithm [1]. The first is useful in forming an initial set of clusters and especially so because it can handle streaming data incrementally in linear time. The k-means algorithm is then applied to improve the quality of the clusters.

To detect changes in the shape and density of clusters, we analyze the density of points within a cluster as a function of the distance to the cluster's centroid. More formally, we define the density function as $f(\mathbf{r}) = dM(\mathbf{r})/dV(\mathbf{r})$ where $M(\mathbf{r})$ is the number of points within a sphere of radius $\mathbf{r}$ and $V(\mathbf{r})$ is the volume of that sphere. The density function characterizes the shape and density of the cluster. The peaks and valleys of the density function correspond to dense and sparse regions within the cluster. By tracking changes in the density function over time, we can detect changes in both the shape and density of clusters.

The figure below shows four different scenarios in the evolution of a cluster and the corresponding changes in the density functions. The graphs in the figure show the density function $f(\mathbf{r})$ plotted against the distance $\mathbf{r}$ to the centroid. By far the most common of these is the constant event scenario, where the points in the cluster show the same random distribution over time (for example, flu cases over the flu season). Because the distribution of the points remains the same, we expect the density function to remain fairly stable over time as shown in the figure. Another scenario is where recent points cause a new cluster to form. In this case, we detect the formation of the new cluster and track its density function separately from the original cluster. An example for this scenario from the hospital admissions domain is the outbreak of an uncommon disease, e.g. anthrax.

A variation of this scenario occurs when an existing cluster masks the new one. In this case, detecting the formation of new clusters is not enough to trigger an alert: The points from the novel event are clustered into an existing cluster, so no new clusters are created. In this case, we use the density function to detect the novel incident. If the new set of points is gathered in a small region within the existing cluster, the density of points in this region is higher than elsewhere. As a result, a peak forms in the density function. An example for this scenario is a SARS outbreak, where the symptoms of the disease are the same as that of common colds, so the outbreak is likely to be masked by an existing cluster of common cold patients. In the last case, the cluster grows in one or more directions. As a result, the density function extends and tapers off slowly as opposed to a fast drop at the original boundary of the cluster. The spreading of a contagious disease is an example of such an evolution.



Cluster Evolution

To detect changes in a cluster, we take a snapshot of the cluster's density function, process new records, and compare the new density function with the snapshot we had taken. To compare density functions, we use the L_m-distance metric:

$$\sqrt[m]{\int \left| df_1(x) - df_2(x) \right|^m dx}$$

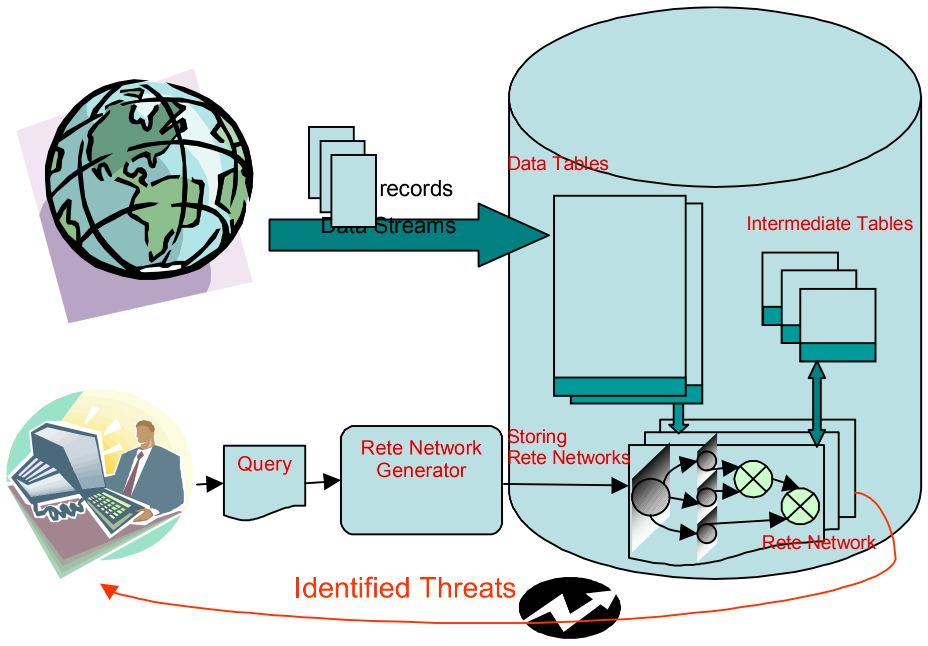
When m=1, the distance between two density function becomes the area between the two curves. However, this metric is not very sensitive to large point-divergences, i.e., the shape of the curves can be significantly different before the metric exceeds the given threshold. In general, by fine-tuning the value of m, we can trade-off between point-divergences and overall shape differences.

Novelty detection results using this approach have been reported at an earlier NIMD meeting [2].

3 Monitoring of Streaming Data

After a novel event is detected, the analyst needs a way of tracking it going forward. For instance, in the above example of novelty detection, the system generated the hypothesis that there is a new disease outbreak whose symptoms might be masked by those of influenza. If the analyst is not interested – e.g. it is off-topic, or already known via other means – then no further action is taken. However, if the new event generates a hypothesis of direct or potential interest, then a new persistent hypothesis tracker is generated, and the input streams are filtered for information pertinent to this hypothesis using the Rete algorithm [3] to correlate data efficiently. Hence, novel event detection adds a new dimension by providing hypothesis genesis in a semi-automated manner – where the analyst remains in the driver's seat to guide which hypotheses are tracked, which are promoted, and which are eliminated due to lack of supporting data or lack of topical pertinence.

The Rete-based approach has been described in earlier papers [4][5]. In brief, the approach avoids re-computation of JOINs of unchanged data when new data is added. The figure below shows the overall system architecture for ARGUS Profile Monitoring. New data elements are appended to



database tables. The continuous queries formulated by the analyst are converted from SQL to Rete networks with our ReteGenerator, and installed in the database. The Rete networks run periodically on the newly arrived data, store and update intermediate results, and generate alarms when any query matches the new data.

Because Rete networks perform incremental query evaluation over the delta part (new stream data) and materialized intermediate results, they can execute much faster than a traditional RDBMS in many cases. However, if materialized intermediate tables become very large, performance can deteriorate severely. Only when the intermediate tables are fairly small, can the incremental evaluation scheme work to best advantage. Fortunately, when monitoring queries are not satisfied frequently, there are usually highly selective conditions that make the intermediate tables fairly small naturally.

To minimize the intermediate tables, we can apply following techniques, namely, a) Transitivity inference, b) Single query optimization, c) Computation sharing among multiple queries, and, d) Incremental aggregation.

Transitivity inference, described in an earlier paper [4], infers hidden conditions from a given query. If inferred conditions are highly selective, performance can be improved significantly. We also implemented single query optimization and computation sharing. Single query optimization is similar to traditional database query optimization. Computation sharing is described in a recently submitted paper [5]. The following sub-section describes incremental aggregation, our current focus of activity.

3.1 Incremental Aggregation on Multiple Continuous Queries

3.1.1 Incremental Aggregation

Many aggregate functions (distributive or algebraic functions), including MIN, MAX, COUNT, SUM, AVERAGE, STDDEV, and TrackClusterCenters (tracking cluster centers), can be incrementally updated upon data changes without revisiting the entire history of grouping elements; while other aggregates (holistic functions), e.g. quantiles, MODE, and RANK, can not be done this way. We implement incremental aggregation on algebraic aggregates, and apply re-aggregation methods on holistic aggregates.

Consider a query A monitoring the number of visits and the average charging fees on each disease category in a hospital everyday. When new tuples from the stream Med arrive, the aggregates COUNT(*) and AVERAGE(fee) can be incrementally updated if COUNT(*) and SUM(fee) are stored.

```
SELECT dis_cat, hospital, vdate, COUNT(*), AVERAGE(fee)            FROM  Med
GROUP BY CAT(disease) AS dis_cat, hospital, DAY(visit_date) AS vdate;
```

Query A

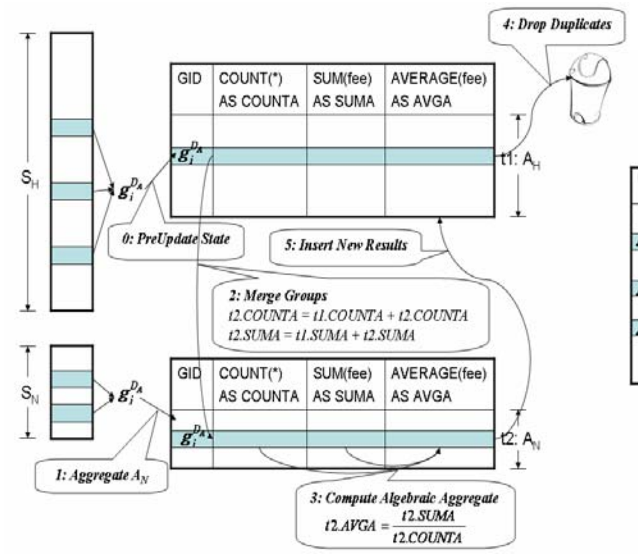


Fig3.1 Incremental Aggregation

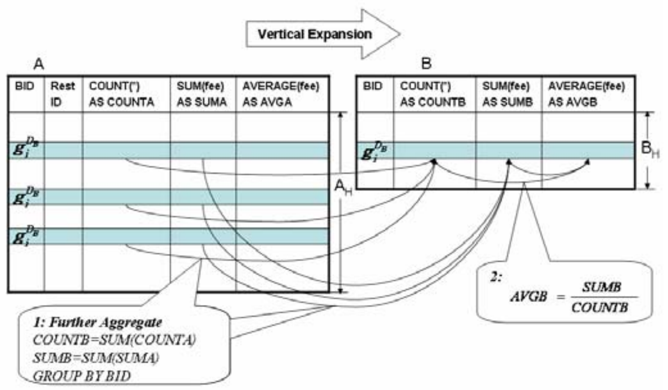


Fig3.2 Vertical Expansion Initialization

The incremental aggregation procedure is shown below, and is illustrated in Fig 3.1 to compute AVERAGE(fee) for query A. The basic aggregates (for AVERAGE, they are COUNT and SUM) and the incremental update rules for each aggregate function are stored in the system catalog. The aggregates in the query are parsed and analyzed, and corresponding rules are applied to instantiate the incremental aggregation code.

Incremental Aggregation

0. **PredUpdate State. A_H contains update-to-date aggregates on S_H.**
1. **Aggregate S_N, and put results into A_N.**
2. **Merge groups in A_H to A_N.**

3. **Compute algebraic aggregates in A_N from basic statistics (omitted for distributive functions).**
4. **Drop duplicates in A_H that have been merged into A_N.**
5. **Insert new results from A_N to A_H, preferably after A_N has been sent to the users.**

3.1.2 Sharing on Multiple Aggregates

Now consider that a new query B arrives. It monitors the daily average fees in a hospital. B groups can be obtained by compressing A groups on the CAT(disease) dimension. Further, B's aggregate can be obtained from A as well. Thus the system shares A's results to evaluate B. This sharing process is called vertical expansion.

```
SELECT hospital, vdate, AVERAGE(fee) FROM  Med GROUP BY hospital, DAY(visit_date) AS vdate;
```

Query B

Vertical expansion is not applicable to holistic queries. However, holistic queries can still be shared if they share the same GROUP BY expressions. On initialization, vertical expansion computes aggregates B_H from A_H instead of from S_H. The incremental aggregation on a vertical-expanded node B is similar to other nodes except the first step which performs further aggregation from A_N instead of from S_N.

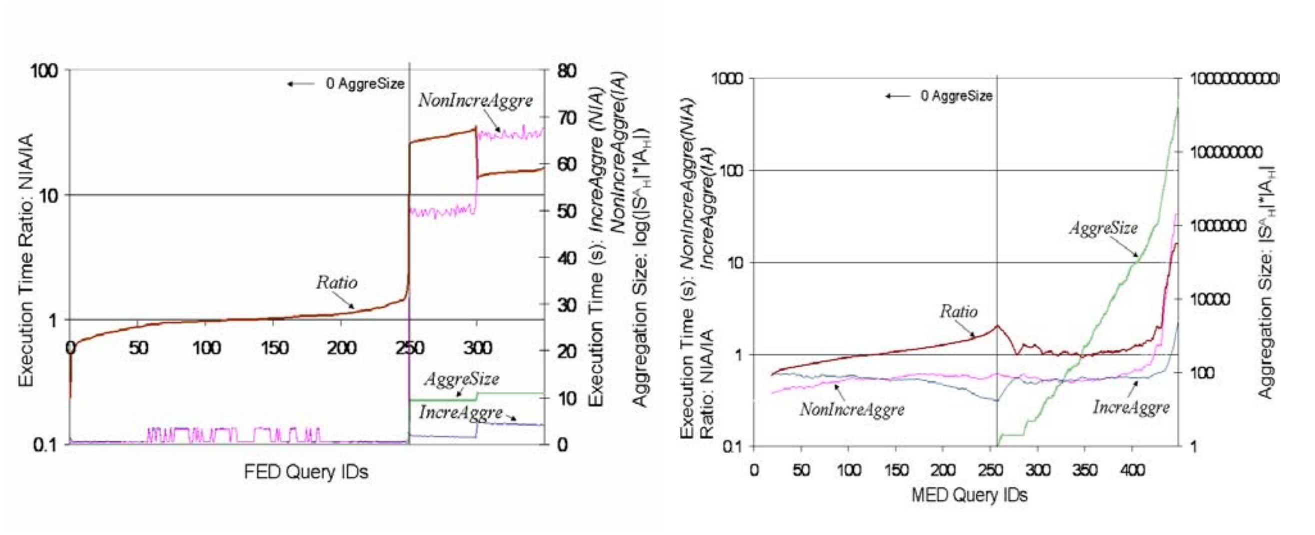
We implemented two sharing strategies. The first one is to choose the optimal sharing node when there are multiple sharable ones. Assuming the data distribution does not change, the optimal choice is the node whose historical part A_H is the smallest. The second strategy is rerouting. After a new node B is created, the system checks if any existing nodes can be improved by being evaluated from B. These nodes are disconnected from their original parents and connected to B by vertical expansion. The system applies a simple pruning heuristic. If a node C satisfies both conditions, and a set of nodes {N} satisfying the first condition are descendants of C, then any node in {N} should not be rerouted, and so are dropped from consideration. If the two queries A and B mentioned above arrive in the reverse order, the sharing can still be achieved by applying rerouting.

3.1.3 Evaluation Study

We conducted experiments to study the effect of incremental aggregation, and the effect of shared networks. Two databases are used. One is the synthesized FedWire money transfer transaction database (Fed) with 500,000 records. And the other is the Massachusetts hospital patient admission and discharge record database (Med) with 835,890 records. Both databases have a single stream with timestamp attributes. To simulate the streams, we take earlier parts of the data as historical data, and simulate the arrivals of new data incrementally.

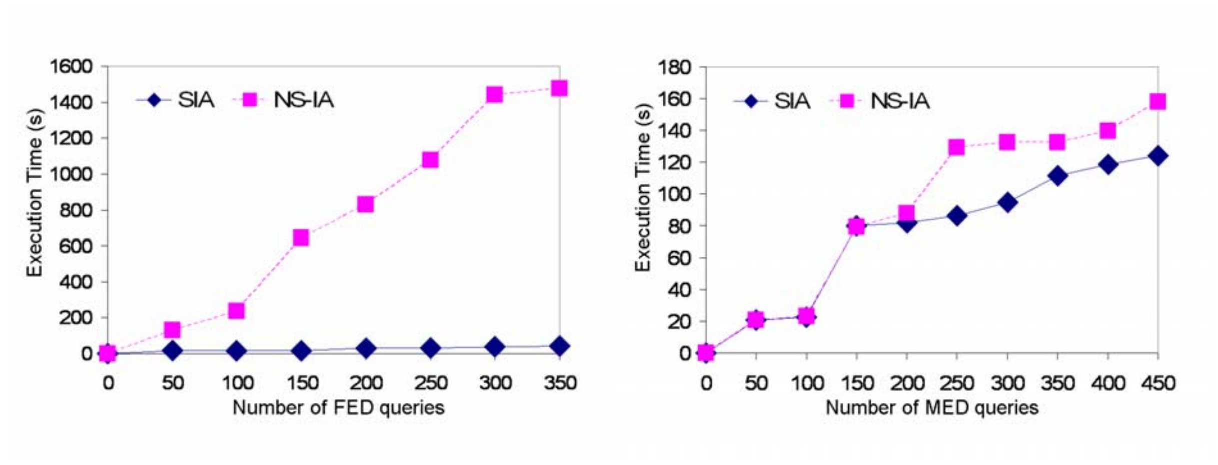
We use 350 queries on Fed and 450 queries on Med. These queries are generated systematically. Interesting queries arising from applications are formulated manually. Then more queries are generated by varying the parameters of the seed queries. Some of these queries aggregate on selection and self-join results.

The following figures show the execution times of incremental aggregation (IA) and non-incremental aggregation (NIA) on each single query, and the ratio between them, NIA/IA. A NIA/IA ratio above 1 indicates better IA performance. Since there are more fluctuations on diversified Med queries, we show running averages over 20 consecutive queries to make the plot clear. The queries are sorted in the increasing order of AggreSize = $|S_H(A)|*|A_H|$, shown on the second Y axis. $|S_H(A)|$ is the actual aggregation data size of the historical part after possible selections and JOINs. We experimented with two other metrics, $|S_H(A)|$ and $|S_H(A)|+|A_H|$, which show less consistent trends to the time growth and the NIA/IA ratio. This indicates that the DBMS aggregation operator on S_H has time complexity of $|S_H(A)|*|A_H|$. AggreSize indicates the query characteristics. There are about 250 queries in both Fed and Med whose $|S_H(A)|$ is 0, shown to the left of the vertical cut lines and are sorted by the NIA/IA ratio. Unsurprisingly, their execution times are very small. The small time fluctuations are caused by the DBMS file caching.



For the remaining queries, incremental aggregation is better than non-incremental aggregation. In particular, it gains more significant improvements when the aggregation size is large. These large-size queries dominate the execution time in multiple-query systems. Thus significant improvements on such queries are significant to the whole system performance, counting for up to 10 times improvement in the total execution time.

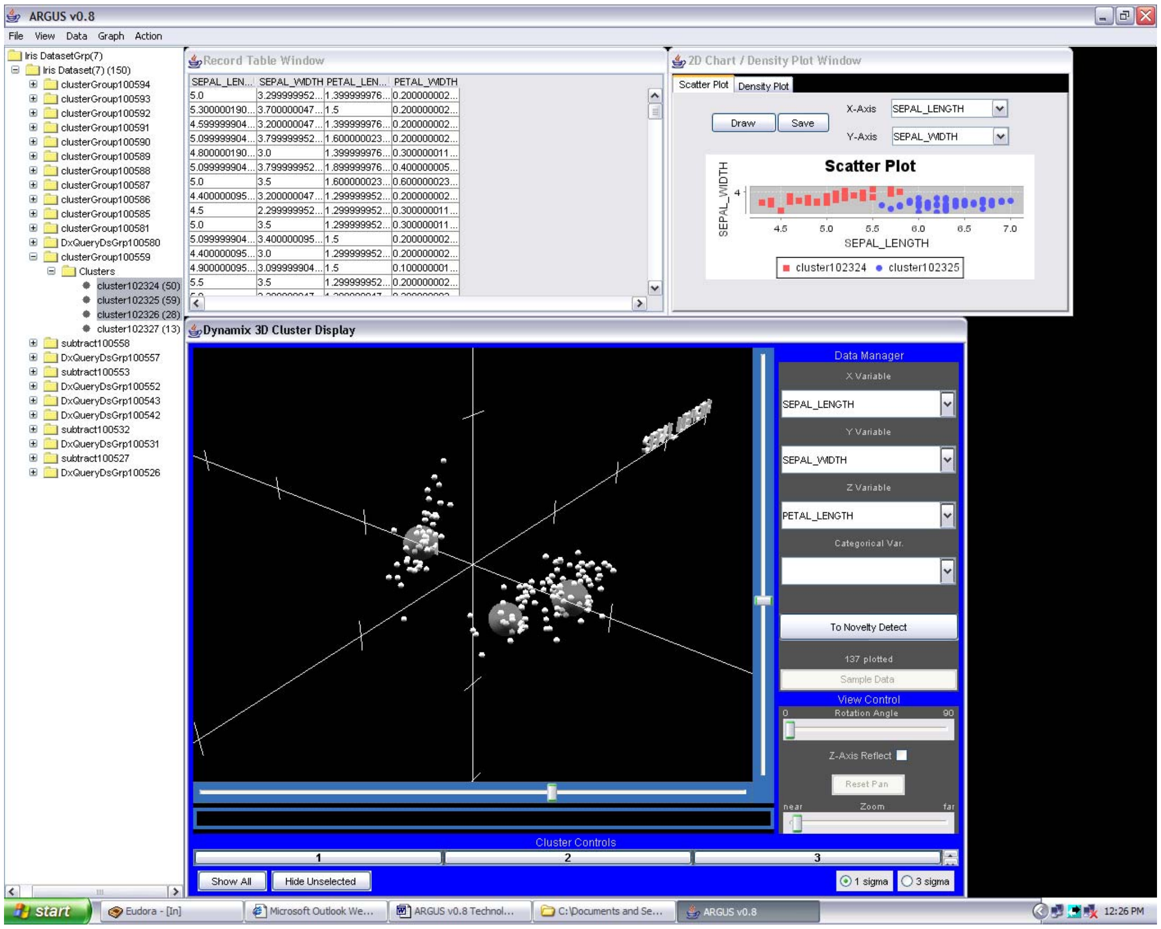
The following figures show the total execution times of shared (SIA) and non-shared (NS-IA) networks by scaling over the number of queries. Clearly, incremental sharing provides improvements, particularly on Fed where queries share more overlap computations.



4 ARGUS Data Explorer

To make the facilities described above accessible to an analyst for work on structured data sets, we have developed an integrated environment called the ARGUS Data Explorer, a screenshot of which is shown below. This environment permits analysts to identify various subsets of structured data sources using querying, clustering, and novelty detection. These methods can be combined by applying one method to the results of another; along with standard set operations (union, intersection, difference). These facilities provide analysts with great flexibility in working with structured data, and are intended to convey a sense of "rolling around in" the data. Access to data sources and the results of data manipulation operations can be shared between analysts subject to a fine-grained security system.

The screenshot below illustrates some of the operation of the interface. The Windows Explorer type tree on the left hand side of the screen keeps track of all the data sets and subsets that the user has chosen or created. The tree structure visually conveys the derivation history of each subset in the tree, and the tree can be expanded or collapsed as the analyst finds useful, in particular allowing the analyst to focus on one part of the analysis without being swamped with full detail on all data subsets created during the course of a session. The right-hand side of the screen is available for various views of the data including tabular record data and 2D/3D plots.



- Iris DatasetGrp(7)
  - Iris Dataset(7) (150)
    - clusterGroup100594
    - clusterGroup100593
    - clusterGroup100592
    - clusterGroup100591
    - clusterGroup100590
    - clusterGroup100589
    - clusterGroup100588
    - clusterGroup100587
    - clusterGroup100586
    - clusterGroup100585
    - clusterGroup100581
    - DxQueryDsGrp100580
    - clusterGroup100559
      - Clusters
        - cluster102324 (50)
        - cluster102325 (59)
        - cluster102326 (28)
        - cluster102327 (13)
    - subtract100558
    - DxQueryDsGrp100557
    - subtract100553
    - DxQueryDsGrp100552
    - DxQueryDsGrp100543
    - DxQueryDsGrp100542
    - subtract100532
    - DxQueryDsGrp100531
    - subtract100527
    - DxQueryDsGrp100526

SEPAL_LEN...	SEPAL_WIDTH	PETAL_LEN...	PETAL_WIDTH
5.0	3.299999952...	1.399999976...	0.200000002...
5.300000190...	3.700000047...	1.5	0.200000002...
4.599999904...	3.200000047...	1.399999976...	0.200000002...
5.099999904...	3.799999952...	1.600000023...	0.200000002...
4.800000190...	3.0	1.399999976...	0.300000011...
5.099999904...	3.799999952...	1.899999976...	0.400000005...
5.0	3.5	1.600000023...	0.600000023...
4.400000095...	3.200000047...	1.299999952...	0.200000002...
4.5	2.299999952...	1.299999952...	0.300000011...
5.0	3.5	1.299999952...	0.300000011...
5.099999904...	3.400000095...	1.5	0.200000002...
4.400000095...	3.0	1.299999952...	0.200000002...
4.900000095...	3.099999904...	1.5	0.100000001...
5.5	3.5	1.299999952...	0.200000002...

X-Axis SEPAL_LENGTH
Y-Axis SEPAL_WIDTH

🟥 cluster102324 ● cluster102325

X Variable SEPAL_LENGTH
Y Variable SEPAL_WIDTH
Z Variable PETAL_LENGTH
Categorical Var.
To Novelty Detect
137 plotted
Sample Data
0 Rotation Angle 90
Z-Axis Reflect ☐
1 2 3
(● 1 sigma) ○ 3 sigma

View showing the 4 constituent windows of ARGUS v0.8 – Tree, Records, 2D and 3D

4.1 Menu Functions within the ARGUS Data Explorer

Below is a brief description of the functionality available within the Data Explorer. The software is scheduled to be demonstrated on April 19th, 2006 at the Poster Session of the NIMD PI Meeting.

File > Exit, under this menu item is used to exit the program.

View > Records is used to view a selected dataset in the Records Table Window.

Data > Subset is used to partition the data into subsets. The subsetting action can be exact or approximate. The result of executing a subsetting query results in the return of a matched and an unmatched subset.

For exact subsetting, the user is provided with two slider range bars for each field that is available in a given dataset. The upper bar sets the low range of the subset and the lower bar sets the upper range for the subset. So for a given dataset if a field has a range of 1 to 10, setting the upper bar at 3 and the lower bar at 7 will return all records in the 3 to 7 interval for that field. This will be returned as the matched data subset. The remaining records in the dataset are returned as the unmatched data subset. A "Check" button is available to automatically check data ranges and logic.

Approximate subsetting uses the same parameter setting dialog box as the exact subsetting except for a key difference in that the Distance Threshold comes into play. If the Distance Threshold is set to 0.0 then approximate subsetting is reduced to exact subsetting. For values of the Distance Threshold greater that 0.0, exact matching is transformed into approximate matching and increases the "catchment region" of the match using a distance function (fixed to Euclidean Distance in v0.8). So the example stated in the exact subset section above with a Distance Threshold greater than 0.0 would, in all probability, return a greater number of matched records as compared to the exact match.

Data > Union is used to combine two datasets.

Data > Intersect is used to glean the records that represent the intersection of two datasets.

Data > Subtract is used to subtract one dataset from another.

Data > Statistics fires up a statistics routine that automatically generates the summary statistics on a chosen dataset. These summary statistics are used by the analyst to get an overview of the data so as to better plan their hypothesis exploration.

Graph > Density Plot is a 2D plot that is used to visualize one or many cluster densities. The X axis represents the distance from the center of the cluster going outwards and the Y axis is the magnitude of the density for a given cluster.

Graph > Scatter Plot is a 2D plot that is used to visualize the scatter plot for any two fields of interest in a given dataset. The X axis represents the first selected field and the Y axis the second selected field. The user, for a selected dataset can re-draw the plot for any two fields of choice.

Graph > Plot3D is a multi-functional 3D tool that can be used by the analyst to study multiple clusters of interest. Firstly note that in ARGUS v0.8, this display is separate from the window shown on the previous page. The 3D tool allows the analyst to analyze 3D cluster data. An analyst can study multiple clusters at a given time. There are 3 distinct parts in this tool, namely the Data Manager, the View Control, and the Cluster Controls.

In the Data Manager, for a given view, the analyst can select the X, Y, and Z fields of choice. Additionally a categorical variable can also be chosen should this be available in the dataset being analyzed. There are two types of data views accessible via a toggle button titled Novelty Detect or Same Time View depending on toggle state. The Same Time View (default) shows analyst selected clusters from the same time period. The Novelty Detect view is used to depict cluster evolution, i.e., how does the same cluster evolve over time. This can be used to spot cluster density changes, emergence of novel clusters, novel sub-clusters and the like. The Data Manager also shows the total number of 3D records plotted in the current view.

The View Control has a set of intuitive controls used to manipulate the current view. Rotation and zoom are most useful to obtain appropriate views,

The Cluster Controls help show and hide clusters of interest. A most useful feature is the 1-sigma and 3-sigma toggle radio buttons that enable the user to render spheres (around the actual data points) that represent the one standard deviation and three standard deviation ranges for that cluster.

Action > Cluster on a selected dataset is used to cluster the data based on the leader-follower clustering algorithm. A parameter dialog window enables the user to set two categories of parameters. The first is the Cluster Distance Threshold, which in essence defines the clustering neighborhood, so the smaller this number the smaller the neighborhood, hence smaller the number of points within each cluster. Hence the total number of clusters needed to cover the entire data space is proportionately large. In contrast, for a large Threshold, the total number of clusters is relatively small, but the number of points within each cluster is large. The second parameter (of set thereof) is the importance weight of each field in relation to the others. So with all weights set to 1.0, equal weighting is assumed, a weight of 0.0 effectively removes that particular field from the clustering process, and so on.

Action > Recluster works on clusters generated by the leader-follower (Cluster) process above and is used to refine the clusters generated by the leader-follower algorithm. Recluster uses the k-means algorithm, hence is relatively slower, but it produces higher quality clusters.

Action > Novelty works on clusters generated by Cluster or Recluster and is used to typically compare pairs of clusters. Ideally these are clusters that are contiguous in time over the same dataset and the ideal is to track and detect novelty in cluster evolution through time. There are two parameters used here, the Density Threshold which is a real valued number that determines the size of the radial density bins and the Cluster Size Threshold, an integer that determines the number of points in novel clusters.

References

[1] Pattern Classification, Second Edition. *Richard O. Duda, Peter E. Hart, David G. Stork*, Wiley, 2001.

[2] Novelty Detection in Data Streams: A Small Step Towards Anticipating Strategic Surprise. *Cenk Gazen, Jaime Carbonell, Phil Hayes*, NIMD PI Meeting, Washington, DC, June 2005

[3] Rete: A fast algorithm for the many patterns/many objects match problem, *Charles Forgy*. Artificial Intelligence, 19(1), pages 17–37, 1982.

[4] ARGUS: Rete + DBMS = Efficient Continuous Profile Matching on Large-Volume Data Streams, *Chun Jin and Jaime Carbonell*, To appear in Proceedings of 15th International Symposium on Methodologies for Intelligent Systems, May 2005.

[5] Computation Sharing on Continuous Queries, *C. Jin and J. Carbonell*, Submitted to VLDB, 2006.