User-interface elements for discovering correlations in multi-dimensional data sets

Mihai Budiu, MSR SVC

November 8, 2015

# Abstract

Visualization is a crucial part of data manipulation. However, exploring interactively complex visualizations of large amounts of information is not easy. To navigate through highly dimensional data the user needs easily used tools to discover patterns and correlations. We are proposing a very simple and intuitive user-interface element based on drag-and-drop to help the user discover correlations in multi-dimensional data sets. The UI allows the user to (a) assign colors to entities according to values of their attributes, (b) drag-and-drop colors between different views displaying entity attributes and (c) drag-and-drop data selections between views. We exploit functional relationships in the underlying data model to transport colors between views in a sound way.

# Setting

We assume that the data to be displayed is contained in a database. For simplicity, we will use a traditional relational database (e.g., SQL Server), but this technique can be applied to other types of databases, such as object-oriented databases, excel spreadsheets, etc. The objects stored in the database are not restricted to scalar values; our techniques can accommodate complex objects, such as time-series. The only essential ingredient we require from the database is to provide (a) a schema for the data stored in each table (or view), and (b) a set of relationships which link columns in different tables to each other (e.g., foreign key relationships, or one-to-many relationships between tables).

We start by analyzing how our method can be applied when the data is in a single table with multiple columns. We exemplify with both numeric and categorical columns. We use as sample data measurements of a cloud computing application; however, our techniques are applicable arbitrary data domains. Later we show how our methods can be used to correlate information from multiple database tables.

## Terminology

The bold terms in this section are used consistently in this document.

Throughout this text we will use the table in Figure 1, which describes a set of entities; the ***entities*** are (computational) processes running in a distributed system. Each process is identified through a unique ***key*** (the process id). The process ***attributes*** are scalar-valued measurements, such as its running time (shown in column “time”), average disk utilization, network packets retransmitted, average memory utilization, and the name of the machine where the process ran. The time, disk and memory and network columns contain ***numeric*** data, while the machine column contains ***categorical*** data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process id [Key]** | **Time** | **Avg disk** | **Avg memory** | **Network** | **Machine** |
| D6B98074 | 263.45 | 15547255.13 | 0.411 | 0.499 | m-113 |
| 6656D9B5 | 246.53 | 16271201.53 | 0.323 | 0.566 | m-045 |
| 9584176B | 241.74 | 16953056.72 | 0.412 | 0.471 | m-022 |

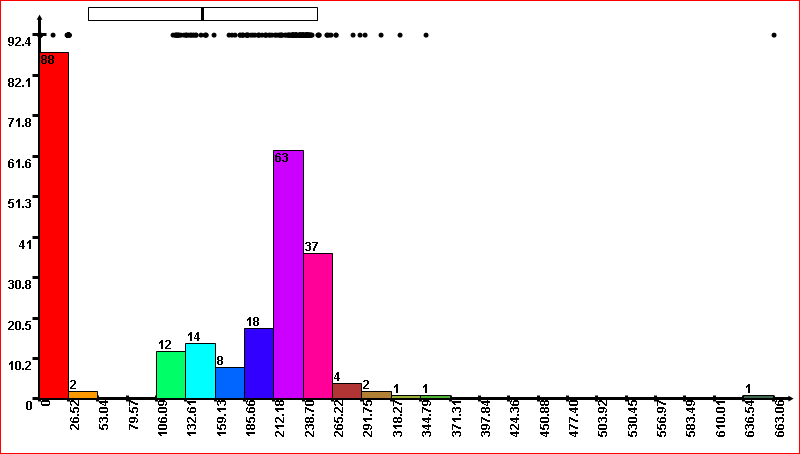
Figure : fragment of a database table used throughout this document.   
The “process id” column is the primary key.  
The data in a particular numeric column (e.g., time) can be seen as a distribution of values;   
most of the views in this document show data distributions.

Given a table with a primary key, we can see the values in any numeric or categorical column of the table as a ***distribution*** of values. We plot these distributions as histograms of values, by dividing the values in a number of buckets. (The buckets do not have to be of the same size; however, all our examples use equal-sized buckets). Figure 2 displays the distribution of the execution times of the processes in our data set (the dots in the top part of the figure), and the histogram of the execution times using 25 equal-sized buckets. In all histogram plots the X axis spans the range of the plotted attribute (so time values range between 0 and 663.06), while the Y axis is the count of entities falling in each bucket. Each bucket is assigned a different color. We call such colored plots ***views*** of the data. We ignore the units of measure for the magnitudes (e.g., time is in seconds).

Running time distribution

12 points in green bucket

Y axis: count in each bucket



Number of values in bucket

Value histogram (density)

X axis: attribute value

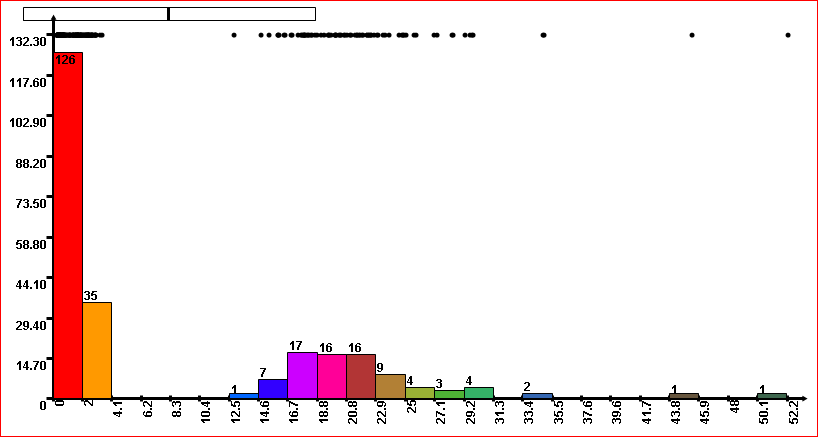
Figure : view of the values in the "time" column as a distribution and histogram.  
In this case the X axis is the running time of each process.

This view assigns a different color to each bucket. The color values can be maintained as a separate relation (table), mapping the keys in the view to the colors; this ***color mapping*** is dynamically computed, and isa property of the view of the data being displayed.

|  |  |
| --- | --- |
| **Process id [Key]** | **Color** |
| D6B98074 |  |
| 6656D9B5 |  |
| 9584176B |  |

Figure : The color mapping is relation assigning to each process (entity) one color.

Figure 4 shows the histogram plot of the attribute “average memory utilization”:



X axis: average memory

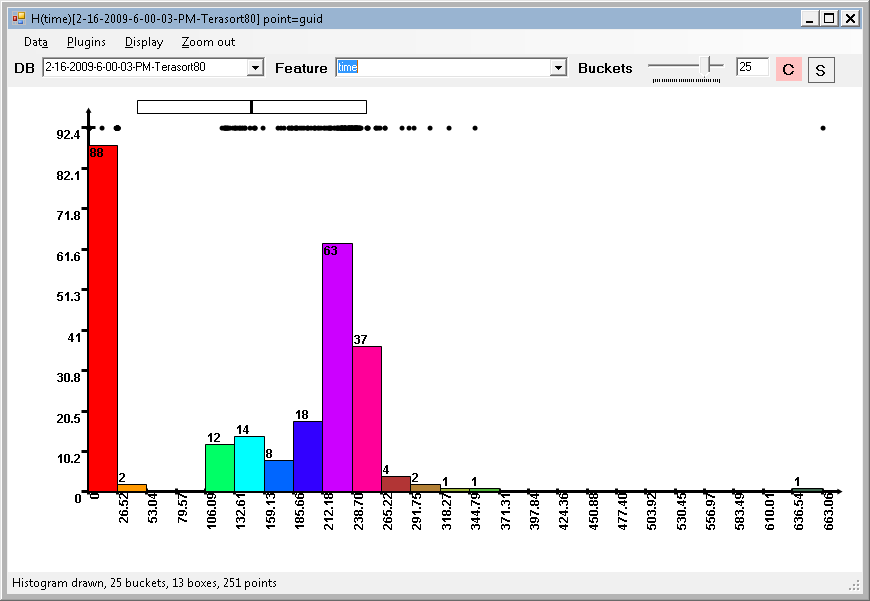
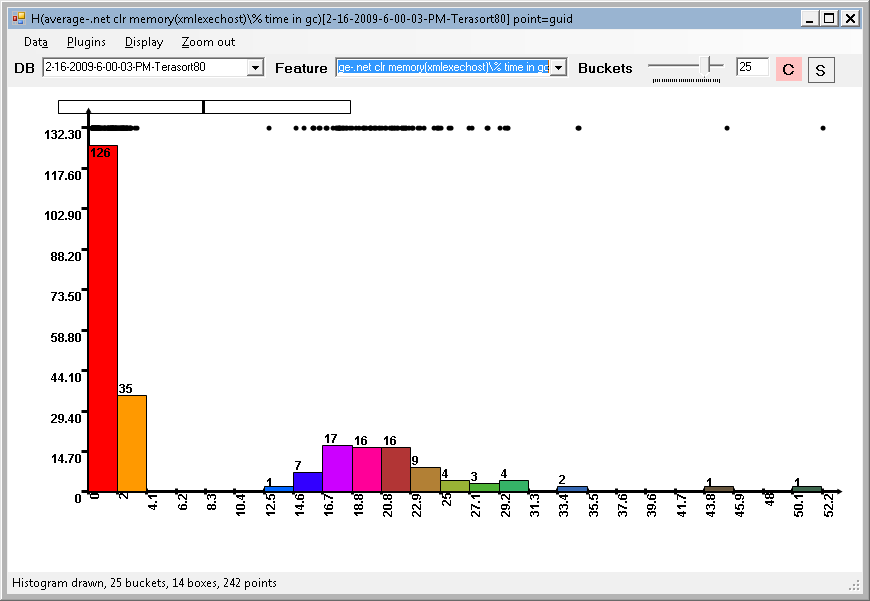
Figure : view of the "average memory" column as a distribution.   
Note that each process has a different color from Figure 2.

This distribution has a different shape, but it is also bimodal (i.e., it has two clusters). Note that there is no relation between the color mappings used in Figure 2 and Figure 4.

## Displaying data correlations

We are interested to discover whether there is a correlation between the running time (Figure 2) and the memory utilization (Figure 4). For this purpose, we display them in two separate views. Then we use the metaphor of drag-and-drop to transport the color mapping from one view to the other. In our implementation we have used a little icon to denote the color mapping of a view; by dragging this icon to another view, the user essentially moves the “color” mapping from the source view to the destination view.

Icon for dragging the color mapping

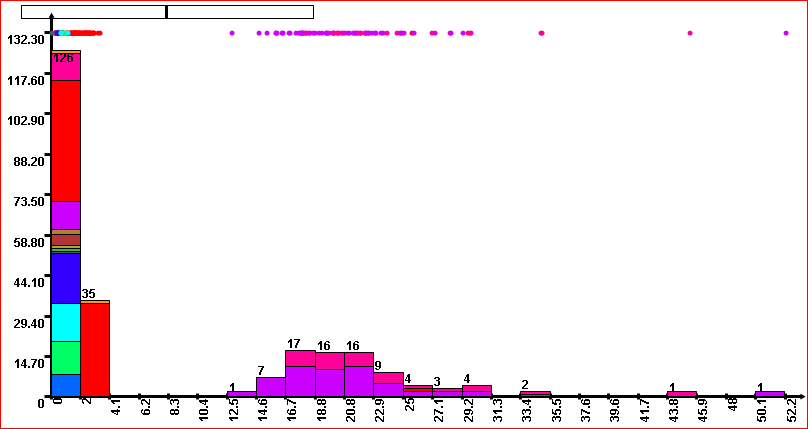
Drag and drop color map

Source view with time distribution

Destination view with memory distribution

Figure : using the mouse the user can drag-and-drop the colors from one view to another view of the data.   
In this case the user is dragging the color map from the time view to the memory view.  
A little icon is used to indicate the color map; the user drags the icon between windows.

The result of dragging colors between views is shown in Figure 6. Each bucket in the memory distribution is now divided into a set of sub-buckets. The sub-buckets are drawn as stacked colored bars. (The sub-buckets are drawn sorted on their color, from bottom to top.). In Figure 6 we have drawn each point in the distribution (top of plot) with the assigned color, but in the rest of this document we will draw the distributions always in black. A sub-bucket corresponds to all entities with the same color in the bucket.



This bucket has 17 points with values between 16.7 and 18.8. There are 6 pink and 11 magenta points.

Point colors assigned from time distribution color map

Cluster 2

Cluster 1

Figure : the memory plot done using the colors from the time view.   
A strong correlation is apparent between the two measurements due   
to the segmentation in two clusters that have almost completely disjoint colors.

The data distribution is exactly the same in Figures 5 and 6, but each bucket has been subdivided into sub-buckets. From this figure it is immediately apparent that there is a good correlation between running time and average memory consumption for some processes: all the buckets in the second cluster have colors corresponding to long running times.

For comparison, here is a view of the “network packets retransmitted” attribute which implies a weak correlation, since the colors are almost evenly distributed between the buckets.

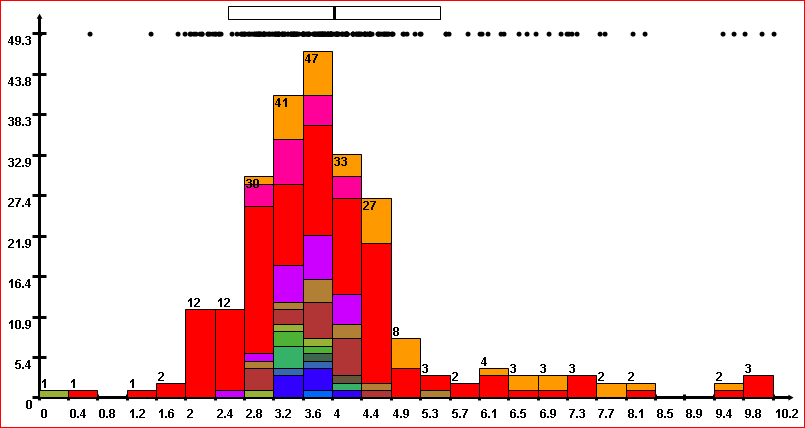
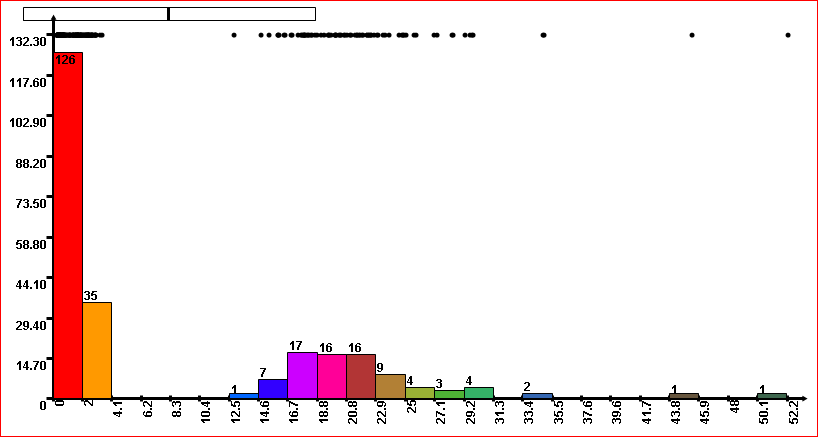


Figure : example plot which implies a very weak correlation between the   
column defining the colors (source) and the displayed column (destination).

If we assume that the distributions of values are actual samples from a real underlying probabilistic distribution, we can interpret the stacked bars histogram in Figure 6 as a plot of the join probability distribution between the source view of the colors (running time) and the destination view (memory consumption).

## Generalization: data selection (zoom-in)

We can extend the usage of drag-and-drop to other operations on the data. For example, the user can draw using the mouse a box around the data to select just a subset of the points (in the sense of traditional relational algebra selection: restricting the view to a subset of the points), as shown in Figure 8. (The user can also select points by dragging the mouse around the distribution plot at the top of the view.)



Drag box to select data

Figure : a subset of the data can be selected visually by dragging a box around with the mouse.

For example, here is the result of the selection, drawn using the same number of buckets; notice that the range on the values on the X axis has changed to reflect the range of the selection (between 12.8 and 34.8). The selection plot uses the same color map as the original plot.

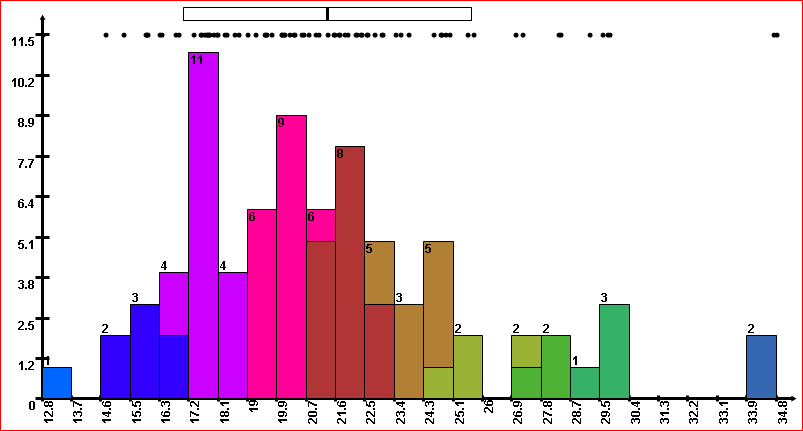
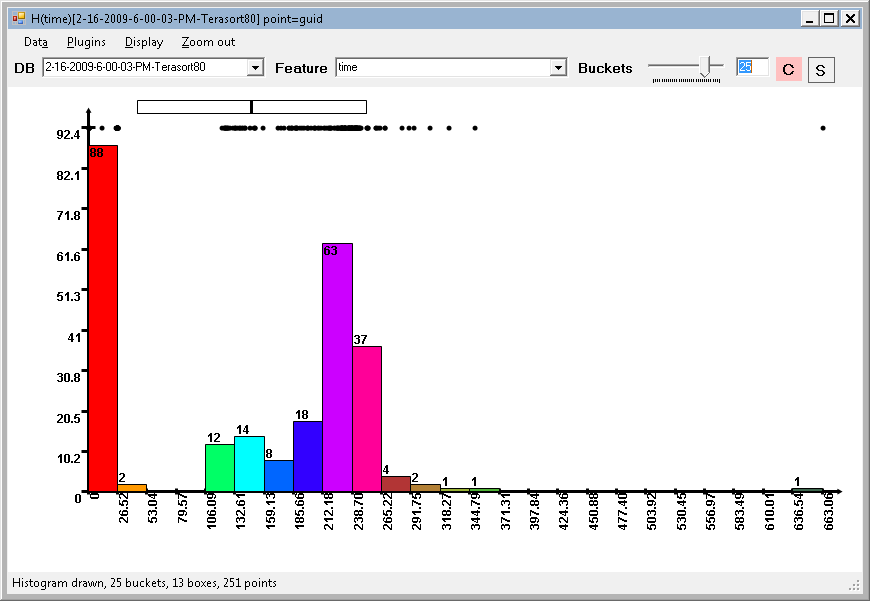
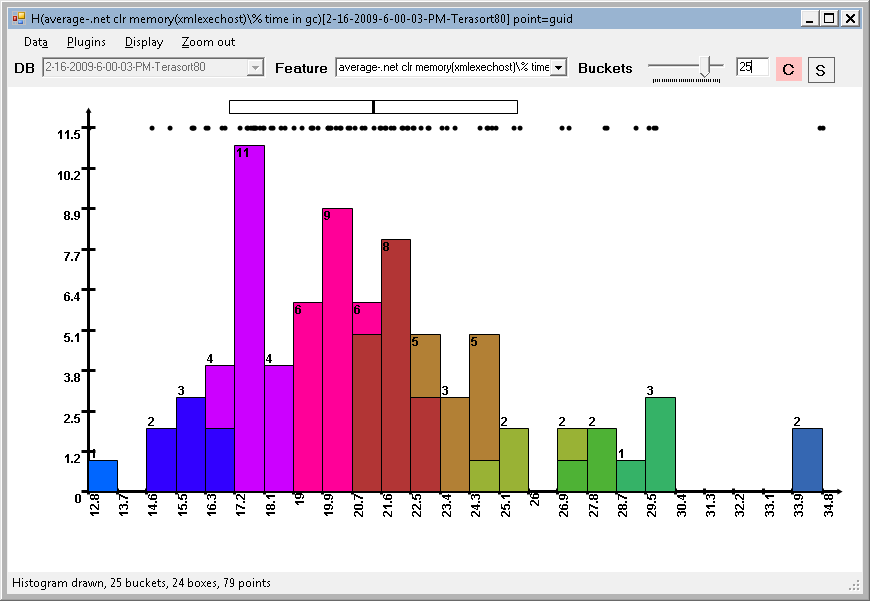


Figure : view of the data selected from Figure 8.   
The color map is preserved through the selection operation.

The user can also use drag-and-drop to move the current selection to another view. In our implementation we have added a small icon to the view, signifying “current selection”; the user drags this icon to “move” the data selection:

Icon for dragging selection



Drag and drop selection

Source view with selected data  
memory distribution

Target view with complete data;  
time distribution

Figure : the user can drag the selection of the data from the memory view and drop it into the time view.

We can define the meaning of the selection drag-and-drop operation in several ways; our particular implementation performs the following operations: (a) selection dragging implies color dragging (this is less confusing for the users), and (b) dragging a selection to a view which already contains a selection performs the *intersection* between the two selections. (Alternative reasonable choices are to just copy the selection, discarding the original selection in the target view, or to perform the union of the two selections; specific controls such as using a modifier key -- control or shift – could be used to distinguish these cases.)

The result of dragging the selection in Figure 10 is shown in Figure 11:

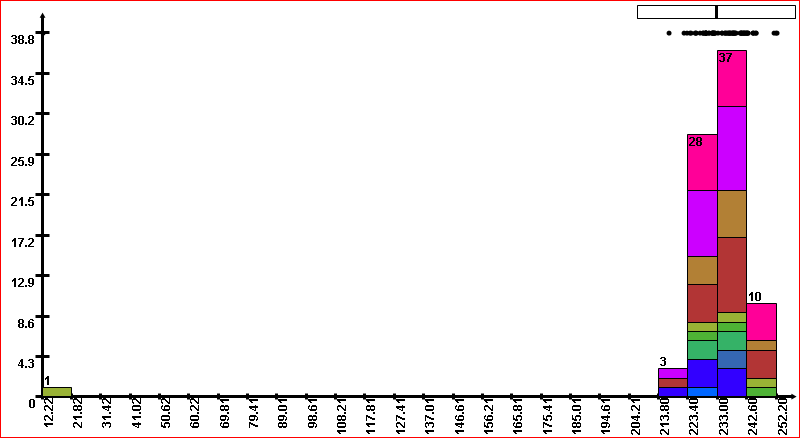
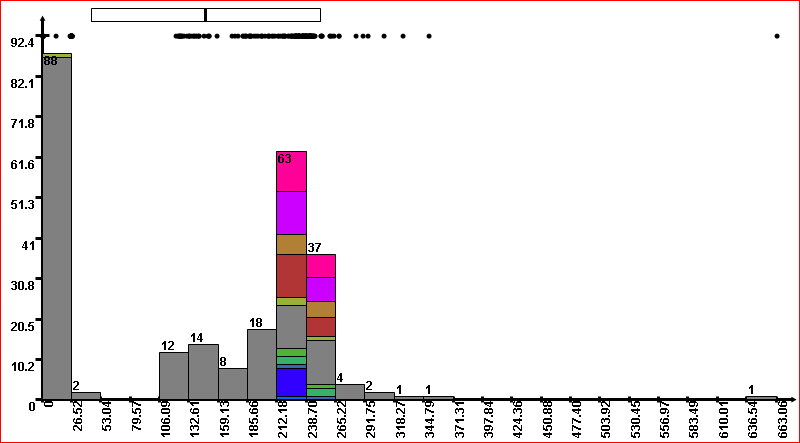


Figure : the result of the operation of dragging the selection in Figure 10. The resulting view only displays the distribution of time values corresponding to the processes selected in Figure 9.

The uniform distribution of colors in this plot indicates that there is no strong correlation between memory consumption and running time *for the selected processes*.

When dealing with selections we have to handle entities in the destination view which do not have a corresponding entity in the source color map. For example, we can drag colors from a view of a subset of processes (Figure 9) to a view containing all processes (Figure 2). In our implementation we have chosen to assign the default grey color to all such entities; this color can never be assigned to a histogram bucket, so it is visually distinct. This makes it easy to find correlations restricted just to selections of the data.



Grey processes are   
absent in the source view.

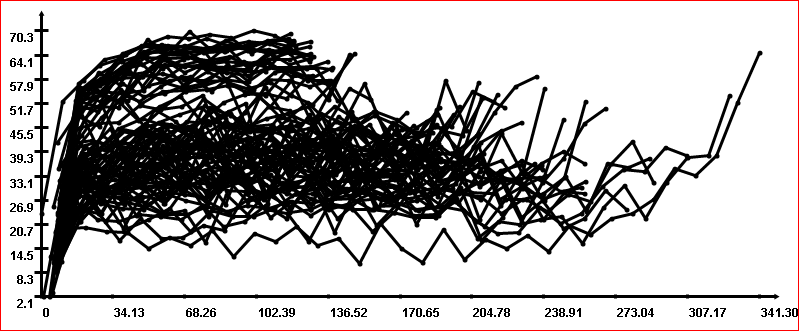
Colored processes are present in the source view.

Figure : Dragging information from a small view (containing a selection as in Figure 9) to a complete view (Figure 2).  
The entities which do not exist in the source view are drawn using a default distinct color, grey in this case.

## Generalization: Application to richer data types

The use of drag-and-drop between views to carry color or selection information around is not limited to distributions and histograms. For illustration, we show two examples: (1) time-series displays and (2) scatter-plot displays.

The time series data in this case is the CPU-utilization of each process during the measurement period; logically, the time-series data is just another column of the processes table; (however, a traditional relational database would have to use a separate table to store time-series data).



X axis: time

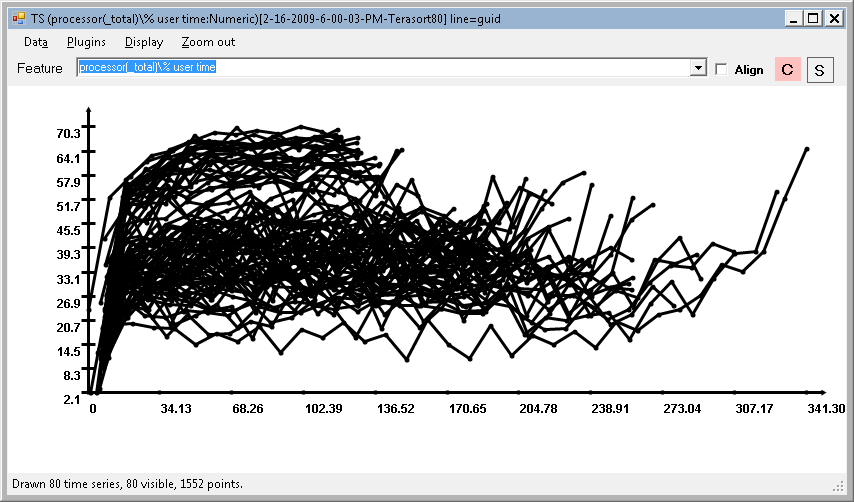
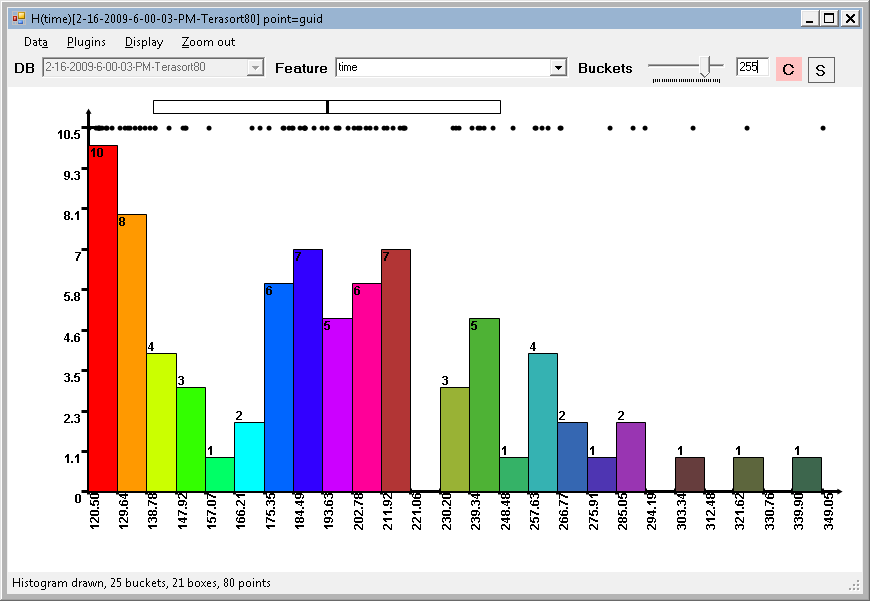
Y axis: CPU

Figure : time-series view of a time-dependent magnitude.   
In this case for each entity (process) we display the CPU utilization as a function of time.

We can drag colors from the time distribution view to the time-series view:

Icon dragging color map

One line per process



Drag and drop colors

Figure : the user can drag colors (or selections) between different views of the data.   
In this case, the user drags colors from a selection of the time view to the CPU utilization view.

The result assigns to each CPU usage curve a color depending on the total running time of the process, as shown in Figure 15.

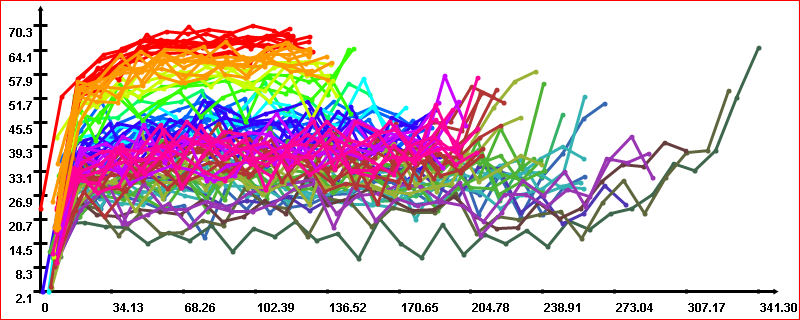
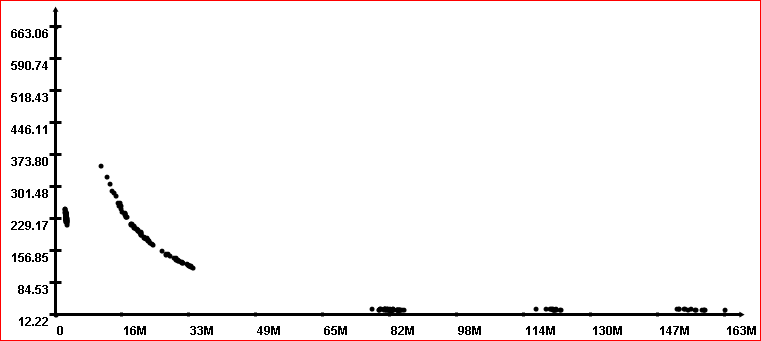


Figure : result of the drag-and-drop operation performed in Figure 14.   
Each time series is colored as a function of the process running time.   
The horizontal bands of color indicate a very strong (inverse) correlation between running time and CPU utilization.

This view makes it very apparent that there is a very strong correlation between running time and CPU utilization: the lines with the same color are clustered together. Processes which finish quickly have high CPU utilization. This information was not apparent at all from Figure 13; moreover, such a correlation between distribution and time-series data is non-trivial to define statistically in a rigorous way; the ability to drag-and-drop colors enables the human user to instantly discover correlations.

Figures 16 and 17 show an example dragging colors between a distribution view and a scatter-plot view. The scatter-plot view in Figure 16 plots the average disk utilization on the X axis and running time on the Y axis. Each point is a different process (entity) in our data set.

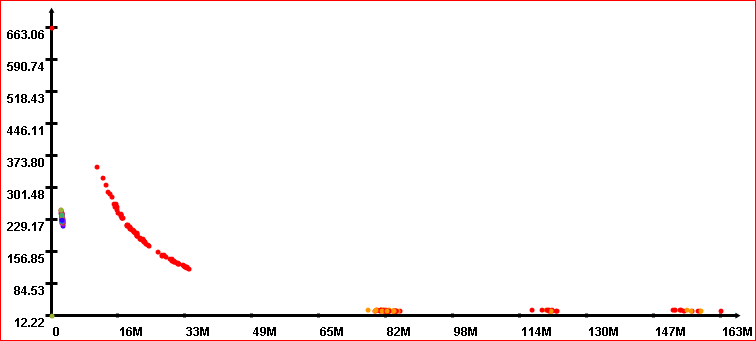


Y axis: running time

X axis: average disk utilization

Figure : scatterplot view of two columns of the table: running time versus average disk utilization.

We can drag the colors from the view containing average memory utilization in Figure 4 to the view in Figure 16, to obtain Figure 17.



Cluster with many colors: weak correlation

Cluster with uniform color: strong correlation

Clusters with few colors: good correlation

Figure : the result of dragging the colors from the memory utilization  
 view in Figure 4 to the scatterplot view in Figure 15.   
The uniformly colored clusters at the right indicate a strong correlation.

The color information can also be moved in the other direction, from time-series or scatterplot views to histogram views; the only requirement is for each view to maintain the entity->color mapping.

## Generalization: visualizing databases with multiple tables.

So far we have assumed that the plotted data originates from a single relational database table. However, the technique of dragging colors between views can be applied even across distinct table views, if the tables can be joined together using a one-to-many (or one-to-one) relationship.

For example, let us assume that our database contains a second table “machines”, which maps each machine to a different server rack (pod):

|  |  |
| --- | --- |
| **Machine [Key]** | **Rack** |
| m-113 | Pod4 |
| m-045 | Pod2 |
| m-022 | Pod1 |

Figure : fragment of a database table mapping each machine to a rack.

We can view the “rack” column as a distribution of (categorical) values, which we can plot using a histogram, shown in Figure 19. Note that this is now a distribution over *machines*, not a distribution over *processes*. The entity that is being plotted is different.

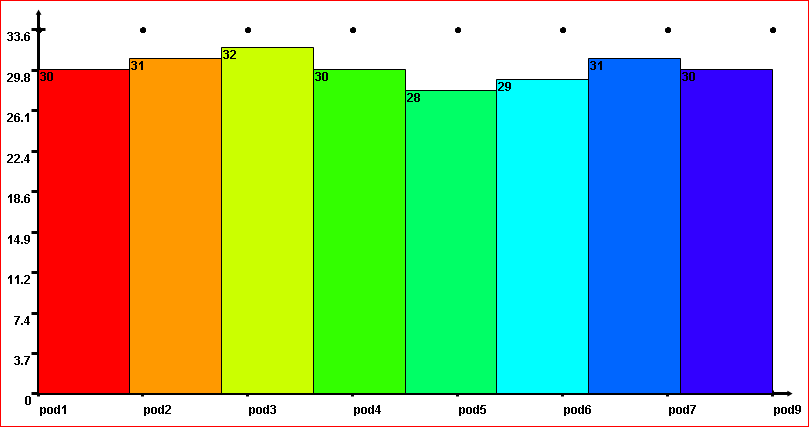


Figure : histogram of rack distribution of machines.   
For each rack the histogram plots the number of machines belonging to the rack.   
The racks are named "pod1", to "pod9".

We are interested to explore additional correlations between the “processes” and “machines” tables. There are two approaches to achieve this: (1) join the two tables into a single table, reducing the problem to the previous case, or (2) extend the drag-and-drop operation to carry information between views of different tables. These two methods produce different outcomes, which we discuss in the next two sections. Note that we can apply this method to drag both color maps and selections, but we will only illustrate the transport of colors in our examples.

### Joining database tables

When joining the “processes” and “machines” tables the result is a table having the “process id” a primary key, because the join column is a primary key of the “machines” table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process id [Key]** | **Time** | **Avg disk** | **Avg memory** | **Network** | **Machine** | **Rack** |
| D6B98074 | 263.45 | 15547255.13 | 0.411 | 0.499 | m-113 | Pod4 |
| 6656D9B5 | 246.53 | 16271201.53 | 0.323 | 0.566 | m-045 | Pod2 |
| 9584176B | 241.74 | 16953056.72 | 0.412 | 0.471 | m-022 | Pod1 |

Figure : schema of table created by joining the processes and machines tables:   
an additional “rack” column has been added to the table from Figure 1.

We can then plot the distribution of the “rack” attribute, where the height of each bar is the number of processes that ran in the corresponding rack; note that this is a histogram plot of categorical values (pod names).

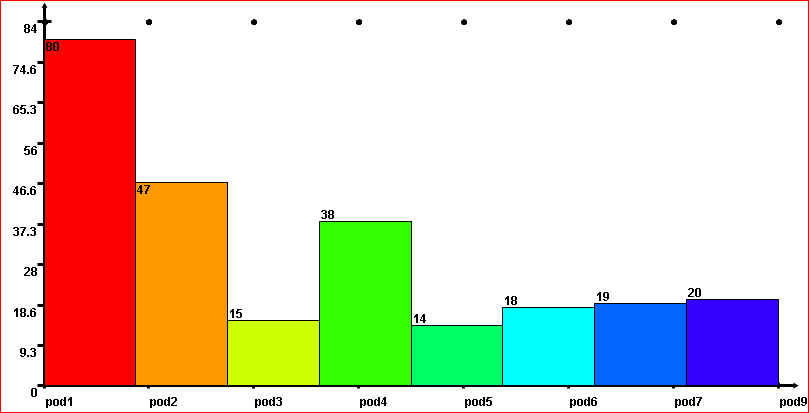


Figure : Histogram of the distribution of processes across pods.  
 This is obtained by plotting the histogram of the "pod" column in Figure 20.

We can now explore correlations between running time and pod allocation by dragging colors from the view in Figure 2 and the view in Figure 21, obtaining Figure 22. This figure indicates a good correlation between placement in pods 1, 2 and 4 and long-running processes. Since these pods also have the largest number of processes, we can hypothesize that they are overloaded.

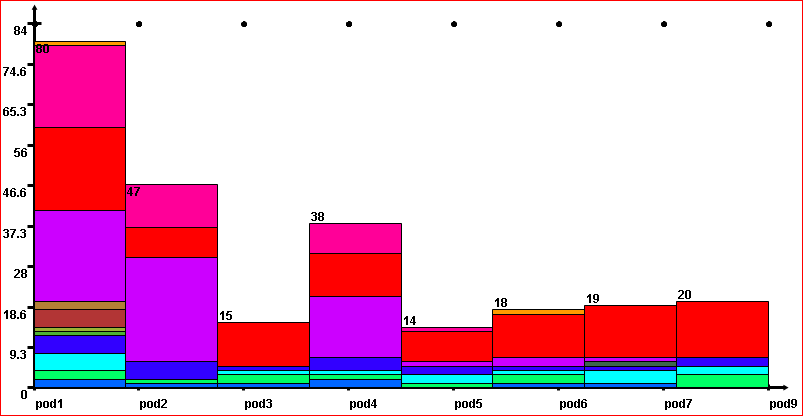


Figure : correlation between pod allocation and running time.   
There are lots of long-running processes in pods 1, 2, and 4.

### Dragging colors across many-to-one relationships

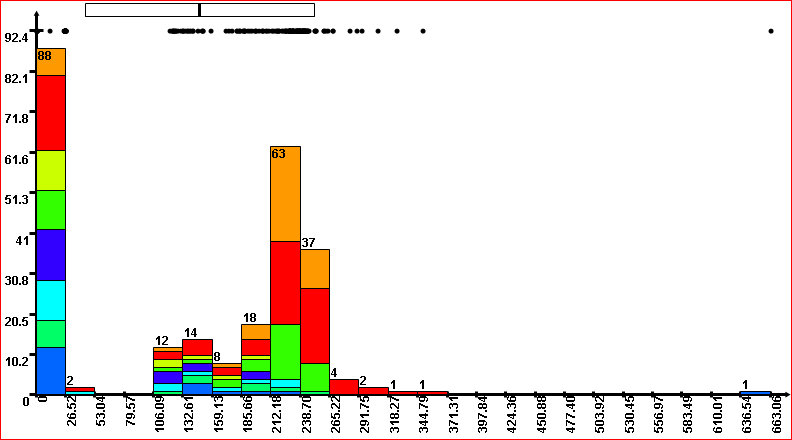
In general, given a database and a set of relationships between its tables, one can transport “color” information in a single direction across many-to-one relationships. Notice that the single-table with primary key is a just special case: there is an implicit many-to-one relationship between the columns of that table and the key column. Also, notice that the flow of information goes from the “many” to the “one” endpoints, because this implies a well-defined, functional relationship for transporting information. For example, we can transport the colors from the “rack” table to the “processes” table, through the common “machine” attribute, but not vice-versa. When dragging colors from Figure 19 to Figure 2, the color of machine A is assigned to all processes that ran on machine A. In the reverse direction there is no well-defined unique color of A, since multiple processes may have run on A. The result of dragging colors between Figure 19 and Figure 2 is shown in Figure 23. (In a sense, this is the “reverse” of Figure 22: Figure 23 shows how running time depends on pod allocation, while Figure 22 shows how pod allocation depends on running time.)  


Figure : correlation between running time and pod allocation.   
Many of the long-running processes are in pods 1, 2, and 4.

In general, one can compute the transitive closure of several many-to-one relationships across the whole database and colors and selections can be dragged-and-dropped across an arbitrarily long chain of relationships. We are in essence performing the relational composition of the color mapping with all many-to-one relationships in the chain. When there are multiple possible chains between a pair of tables the user can be presented a choice, or the system can signal an error. The issue of multiple paths is a property of the database schema, well known in the database literature (see for example the paper “Closures of database hypergraphs” JACM 32(4), 1985, by Domenico Sacca), so we will not discuss it further.

## Maintaining the color mapping

When dragging colors from the “rack” table to the “processes” table we translate a color mapping   
machine->color to a color mapping process->color. Although the translated mapping contains essentially the same information as the original mapping, we are unable to drag the process->color mapping back to the “rack” view, because the process->machine relationship is not a many-to-one relationship. This asymmetry is unfortunate, but it can be resolved by maintaining a little more information for each view: instead of having just one color map for each view, we maintain two color maps: the displayed color map (which maps entities to colors) and the *source* color map. Often the two mappings are the same, but when dragging colors across different tables the mappings are different. If the user interactively changes the displayed process->color mapping, the source mapping has to be discarded, since it is no longer equivalent to the displayed mapping.

|  |  |
| --- | --- |
| **Process id [Key]** | **Color** |
| D6B98074 |  |
| 6656D9B5 |  |
| 9584176B |  |

|  |  |
| --- | --- |
| **Machine [Key]** | **Color** |
| m-113 |  |
| m-045 |  |
| m-022 |  |

Figure : For each view we maintain two color mappings: the entity->color mapping used to display the data, and the source color mapping, which was used to derive by composition the current color mapping.

The entity color mapping is used to draw the data, while the source color mapping is used when dragging colors from the view. Because the source color mapping is indexed on machines, this will enable the user to drag the colors back from the processes view to the racks view, as long as the processes color mapping has not been changed. This method can be also applied to selections.

## Generalization: arbitrary color definitions

So far we have always defined the color mapping of an entity by the histogram bucket where the entity falls. This method is very practical, especially if the number of buckets is small, since it is easy to choose a small number of highly contrastive colors. However, everything we have described so far is still applicable even when colors are defined by arbitrary computations on the entity attributes. However, when the number of colors becomes very large (more than a hundred or so) the usability of the displayed results diminishes severely, since the histogram bars start to be composed of many very small sub-buckets, which cannot be visualized well on a screen.

# Conclusions

In this paper we have explored a method of displaying information by assigning distinct colors to entities, according to the values of their attributes. We have also described how the user-interface paradigm of drag-and-drop can be used to transport color information between different views of the information, allowing the user to discover data correlations quickly and naturally.

# Summary

Visualizing and discovering patterns and correlations in multi-dimensional data is difficult. We propose a user-interface metaphor for simplifying this task.

The core idea is to assign dynamically a color to each item visualized. The correspondence item->color is maintained into a color map associated to each data plot. The user can drag-and-drop the color maps between different views of the data. The resulting view can then easily be analyzed visually for discovering correlations (or lack of correlations). The interaction cost is very small, so the method is very suitable for rapid iteration.

The basic method can be extended in several ways: (a) the users can use drag-and-drop to perform data selection between views (“zoom-in”), (b) the views can encompass multiple database tables (then the operation of dragging color maps requires a recomputation of the map, which can be done based on the existing relationships in the data, such as many-to-one), (c) the views can accommodate complex displays, such as time-series or scatterplots, and (d) the colors themselves can be defined by computations on the data.

We have applied the method successfully to several production-based Microsoft datasets, including health data from Hotmail and performance data from Office Live. The method is very successful in analyzing performance measurements for cloud-computing properties, so it could be immediately applied in products such as MS System Center or business intelligence. We estimate that the implementation cost is very moderate, but the benefits in manageability are large.

Related work has been implemented in products from other companies, most notably Tableau Software, which is the state-of-the-art in data visualization. Their work deals with visualizing relational data and performing relational operations visually. From our knowledge, our methods complement their approach.