Group Members: Di Wang(u1072369)
Shenruoyang Na(u1142914)

Multivariate Data Analysis and Visualization Tool
Final Project Process Book - CS-6630

TABLE OF CONTENTS: LINKS TO COMPONENTS OF PROCESS BOOK

Overview and Motivation:

One important application of information visualization is that it helps domain experts understand multivariate data, which is hard to visualize in conventional ways. Dimension reduction method such as PCA can help visualize the data, but some features of the high dimension space are lost through the dimension reduction process. The main inspiration of this project is the previous work *Visual Exploration of High Dimensional Scalar Functions* by Samuel Gerber. In this paper, a method that involves computation of Morse Smale Complex is used to cluster high dimensional data. Different persistence levels are calculated for each partition to measure the significance level of features in each partition. Then an inverse kernel regression method is used to show the behavior of the point clouds along each attribute. The existing tool that uses this algorithm is written in C++ and python, which does not provide powerful interaction as what javascript and d3 does in frontend. Designing such interface will help users or domain experts do better analysis on multivariate data and understand the high dimension space.

The main goal of this project is to create a tool for the users to analyze specific dataset of user's interest. Such tool provides the ability for the users to understand the high dimensional space (high dimension data is used here instead of multivariate data just to be consistent with the original paper, but it is the same idea). Depending on the persistence level, which is a measure of the significance of the feature, the tool displays how the overall dataset splits into different clusters, shown as a tree structure. It is also able to filter out the outliers. The outliers here are partition of certain size. However, this depends on the user's interest because the user knows whether he/she is interested in partition of certain size. It show also be able to show different plots of the partition that the users are interested in. The interactions between these visualizations would help the users understand more about the high dimensional data.

The dataset used for this visualization is not limited to any single dataset. The first dataset that is going to be analyzed is related to nuclear simulation are obtained from Nuclear Energy University Program. Other multivariate datasets of users' interests from different fields can also be used for this visualization. We got some good feedbacks from through the peer review session about the size of the data to visualize. When the size of the dataset is too large, we would have a function that only selects certain samples from the whole dataset to visualize.

Questions:

One major question we are trying to answer is how users can use our visualizations to understand the dataset they are interested in. In our first case, it would be whether we can find some interesting features from the nuclear dataset.

Also, since there are multiple attributes for the dataset, another question would be understanding the relationship between each attribute. If we consider our multivariate data as point clouds in high dimension space, how are the points clustered and how are the clusters splitted and merged based on persistence level. If we are using the tree to show this, will it make sense to users?

Besides that, another question would be how we would want to design our visualization to help users better analyze the data. Which part of the visualization would the user finds interesting?

Data:
We start with nuclear simulation dataset of 10000 points. Each attribute is numeric. Preprocessing is not required. However, the current available algorithm that calculates the clusters is not in JavaScript. Since we do not want to spend a long time rewriting the algorithm in JavaScript or dealing with python server, we decide to treat this part as preprocessing and run the algorithm locally, which is also the advice from the TA. The result is stored in several csv and json files that can be loaded using javascript. Compared with the original algorithm that saves all the data at each persistence level, our calculated files only store the partitions at base level and give specifications on how the clusters would merge or split as persistence changes. The front-end requires both raw data in the form of csv as well as several processed data files.
Since we mentioned in the proposal that we would like to create a reusable tool that can be used to visualize other dataset, we did not hardcode the project. We try to follow the rule of separation of concerns and make each class independent. The program also does not know about meaning of the data. If a totally different dataset is preprocessed and loaded, the user would be able to see the results from a totally different dataset as well.

Exploratory Data Analysis:
Due to the nature of the algorithm, a tree structure would be prefered to show how different clusters merge or split in high dimensional space. Although it does not capture the geometric information of the points, it tells the users how the overall dataset is clustered based on the persistence (the significance level of the feature), which is something that users might be interested.
Also, plots are linked to the tree node for the users to look at. Some users might be interested in the distribution, while others might be interested in the relationship between different attributes. Several plots such as scatter plot, box plot, pairwise plot and histogram are added for the statistical analysis of each node of user's interest.
The insights that inform our design generally include what the users as domain experts would like to see, what other users might be interested and how we want to design it to make it user friendly and reduce running time.

Design Evolution:

Visualization Design

Layout:

There are many ways to visualize multivariate data. Our idea is to include four major blocks in the design. With the block A contains some button that lets users load the data. After the data is loaded, the information of this data such as number of dimensions, sample size as well as the name of each attribute will shown in this block. Block B and C will contain two type of visualizations of the input data. Block D will include some buttons for interaction commands that the users can play with. The basic framework is shown in. Figure 1.

Block A	Block B
Load Data ● Data Inforamation: ● Selection Information:	**Visualization A** Drop-down box ● Click/Hover/Brush for update
Block C	**Block D**
Visualization B Drop-down box ● Click/Hover/Brush for update	**Interaction Commands:** ● Brush for selection ● Feature 1... ● Feature 2...

Figure 1. Draft Version for the Final Layout of our Design

Focus

As mentioned before, the main focus of this project is to create an interactive interface that users can use to analyze multivariate data. This interface can be reused by different people to work on different datasets. The novel part of this design is to include the algorithm mentioned in the paper *Visual exploration of high dimensional scalar functions* to help the users understand the dataset better. By computing the corresponding Morse Smale Complex for the dataset, a persistence value will be calculated for each cluster that represents the significance level of the feature it represents. Then the users are able to select the specific cluster they are interested in to analyze with the interactions that d3 provides.

Sketches

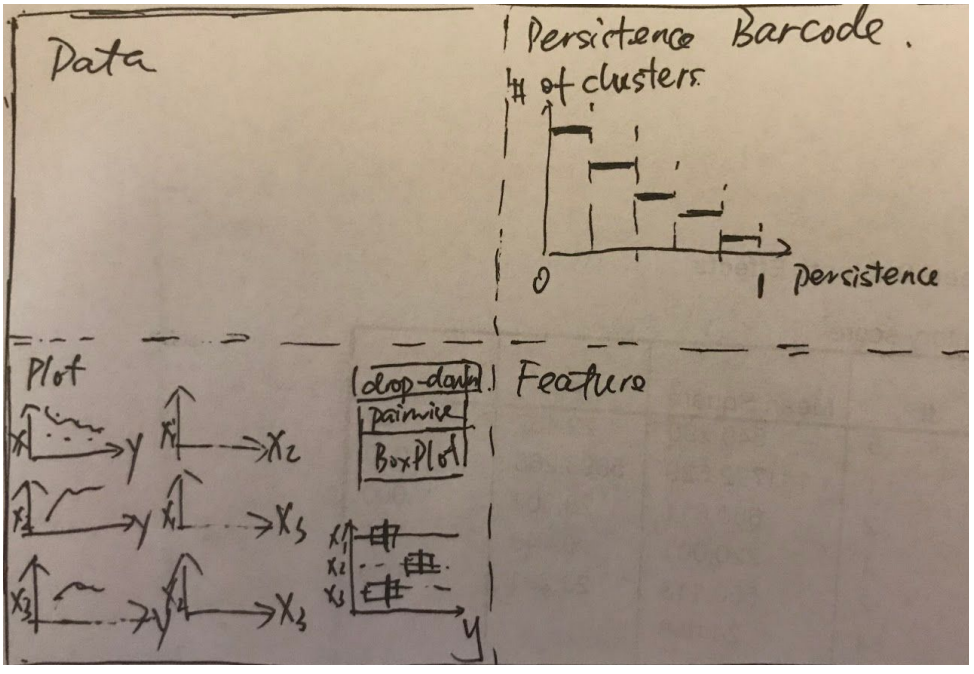

Figure 2. Sketch 1 for the Design

The first sketch is just comprised of certain visualizations from the original paper. Block B includes a chart for persistence barcode is used to show the merging/splitting process of different clusters (in the paper referred to as crystals). However, this visualization lost the geometric property of the data and it was not clear which cluster splits or merges. The plots are capable of giving some information of the data, but other visualizations may also be tried before deciding the final design.

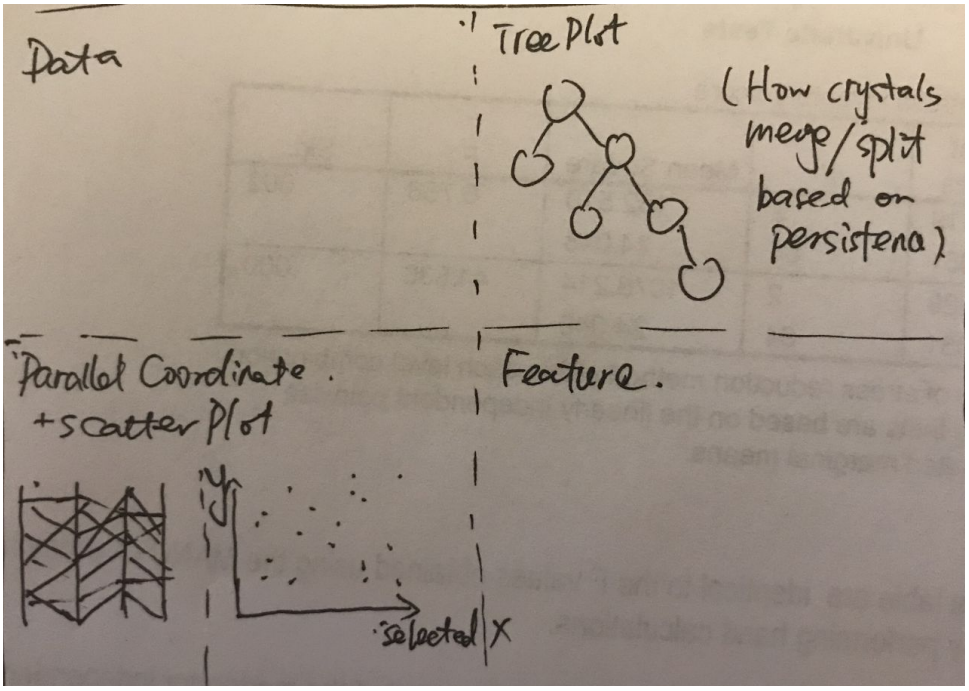

Figure 3. Sketch 2 for the Design

Since the persistence barcode is not something easy for the users to understand and it does not show how crystals are merged/splitted based on persistence, we want to think of some other way to visualize this. One option we come up with next is the tree plot, which will fit our design and show the information we want. For Block C, we are trying parallel coordinates witg scatter plot. There is no specific reason for this. We would just like to try different sketches before we decide which one we want to use.

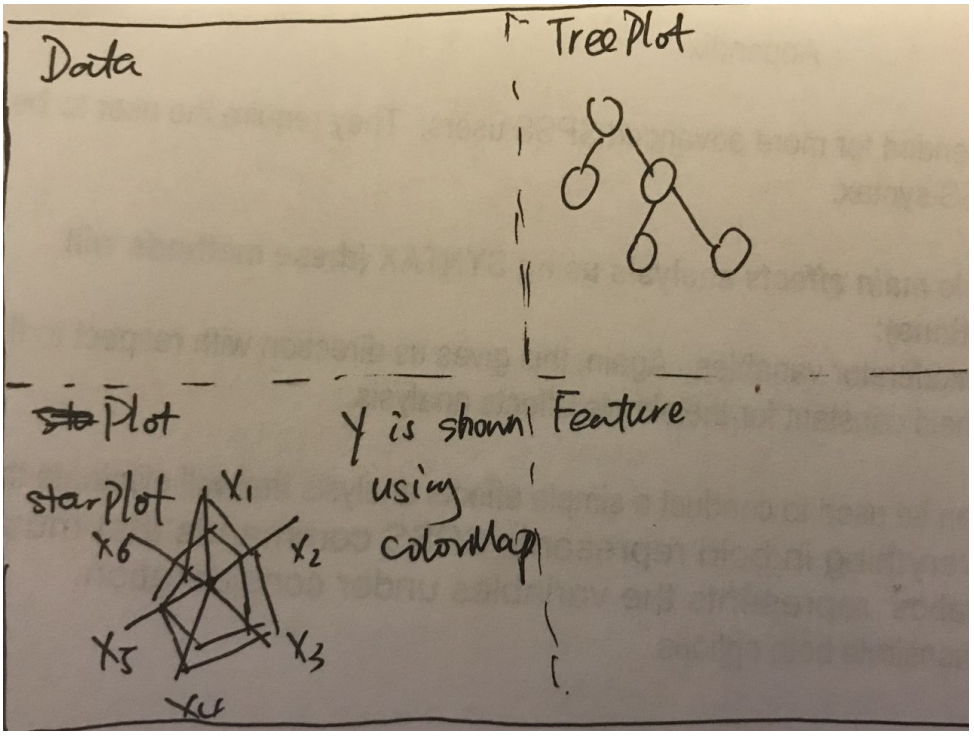

Figure 4. Sketch 3 for the Design

We did not come up with better idea for Block B and kept the tree plot. However, we may include star plot for Block C, which might look neater than parallel coordinates in some sense. We also use the colormap to represent the value of our dependent variable of interest. This might show some interesting feature of the dataset.

Discussion

The treeplot seems a good visualization technique for our interface. We compare the treeplot with dendrogram and find that the traditional treeplot seems more intuitive for this case. However, when the tree is really long, there might be some problem with it. That is something we will keep in mind.

As we show with the three sketches, there are many ways to visualize the points. When we talk with the people that are working on high dimensional data analysis, many of them prefer the traditional ways (as the one in Figure 2). One reason behind this is that it is straightforward and easy to understand. It is still robust when the size of the data increases. Also, if we are going to fit some mathematical model to certain clusters, the best way would be showing it directly in the 2D plot. However, we will have a drop-down menu for different kinds of traditional plots that users might be interested in.

Final Design

Our final design is shown by Figure 5 with some instructions in each of the block. We switch the location of Block C and Block D because we think the users always prefer have the interactions, buttons or commands on the left side and visualization on the right side.

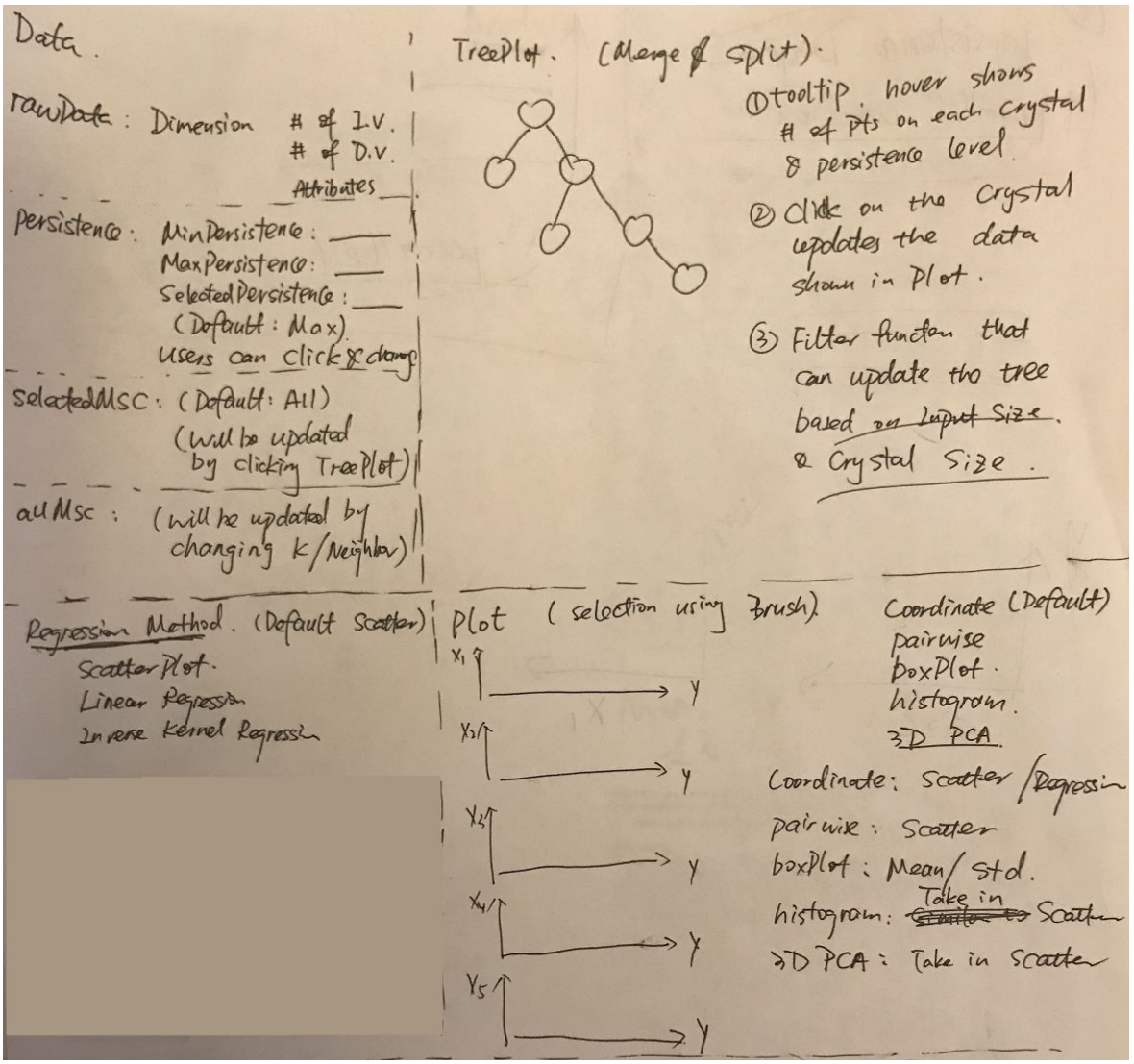

Figure 5. Draft for Final Design

Although there is not tree structure mentioned in the original paper, we try to create the tree-view first since it gives some notion of the algorithm. Since what we want to show through other views are clusters existing in the current tree structure, the tree is the main view that all other views need to be linked to. The designs and interactions we want to include in the tree view are most based on user's interests.

There should also be a plot section. The plot section includes several visualizations of the plots that the user might be interested in. Each plot should give different information of the data. The plot should be updated by the user to show only certain partition.
Besides,there is the data section that gives the information of the whole dataset as well as the selected partition of the data. The data section should also get updated based on user's selection.

In plotting parts, we designed 4 kinds of visualization plots for each cluster (a.k.a treenode). To illustrate the plots intuitively, we visualize the data which has only 3 attributes below as an example.

	attr1	attr2	attr3
1			
...			
N			

These four plots are:
1. rawdata plot: scatter plots to show the relationship between each attribute(independent variable) and some specific attribute(dependent variable) of the data. Suppose we have K attributes, we will have K-1 scatter plots. If we choose attr3 as the independent variable, we will plot two scatter plots, the first one is the plot of attr1 and attr3, the second is the plot of attr 2 and attr3.
2. pairwise plot: scatter plot of the every two different attribute in the data set, in the example, If we choose attr3 as the Y value of the dataset. there will be 1 scatter plot, which the x axis and y axis are attr1 and attr2, the Y value of the point will be shown by the color of the point.
3. Box Plot: box plot of each individual attribute, in the example, if we choose attr3 as the Y value of the dataset, there will be 2box plots of attr1,attr2.
4.Histogram plot: Histogram plot of each individual attribute, in the example if we choose attr3 as the Y value of the dataset, there will be 2 histogram plots of attr1,attr2.

Our visualization obey the design principles, the marks and channels are used correctly. such as the circle of node in tree represent the cluster, and circle in first two plots represent the value of each data point. The rectangle in box plot and histogram represent the the statistic property of each cluster, we can clearly see the variance and difference of each attribute of the cluster. Position is used as channels of the tree and all the other plots, which is the best channel applied in the visualization. And color channel is also used in the scatter plots to show the difference of the value. The length of the rectangle is used in the box plot and the histogram. The highlight of the tool tip of each point in the scatter point obey the rules of the popout. The links between the treenode and the plots shows the relationship of the multiple view. The interaction that clicking the node of the tree and using the button on the to change the cluster's persistence or partition's size enable user to focus on different cluster of the data. and buttons to choose different plot and different attribute as the Y value make user see the data in different angle of view.

Implementation:

Implementation Process

- **Data Processing**

First of all we need to run the algorithm locally in python to calculate the partitions. To make sure the visualization runs smoothly, we did not attachment all the points or partition to the tree nodes. Instead, we only attach the children information and some indexing information that relates the node to the raw data information, which is shown in Figure 6. In this way, the program can fetch out the corresponding raw data information during the runtime. One reason for doing this is that the tree would consist of more than 10000 nodes when the leaf nodes are the ones with lowest persistence. Attaching all the data to every node would crash the browser.

Thus, after we run the algorithm to calculate the partition, we write some python code to process the results and save the output files that contain the information of the basic partitions and how the partitions merge at different persistence levels as parent-children relations. These files together with the raw data would be loaded by JavaScript to show the visualization. We believe the existing algorithm should be robust for other multivariate data.

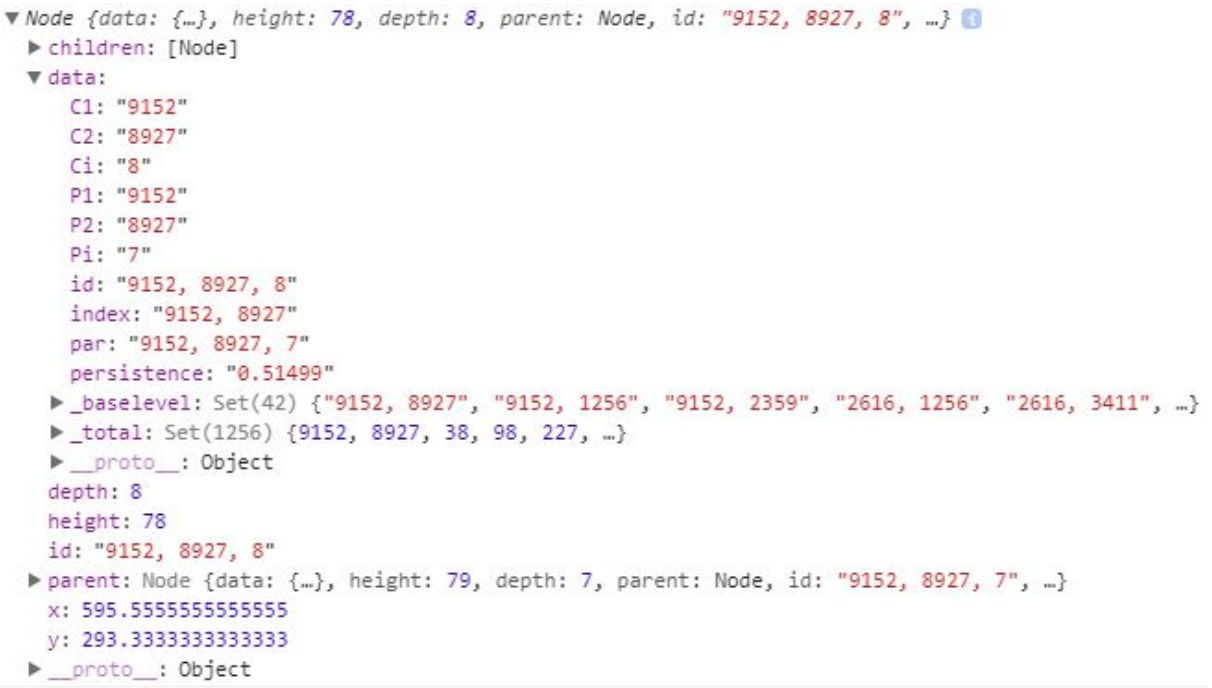

Figure 6. Only the Indexing Information is Stored at Each Node.

As mentioned before, we would like to have different visualizations to show different aspects of the data based on the algorithm. Here the visualizations are not listed in chronological order of implementation since the visualizations are implemented back and forth when new interactions are created or when new bugs are shown in the design. The tree structure, which represents the backbone of the algorithm is implemented first. After the tree is implemented, all other views will connected to the tree for interaction.

- **Tree Design**

Initially our tree structure would show the all levels, which is shown by Figure 7. However, since there are so many levels and would not make much sense to the users, we would like to show the tree structure at different resolution depending on the persistence level of user's

interest. To make the tree look better, the size of the node that represents the cluster will also depend on the number of levels of the tree. The algorithm would compare the persistence value of each cluster with the persistence of user's interest. Then it would build the tree based on the clusters that have persistence level above user's interest. Since we do not want to remove and rebuild the tree each time, we use a filter function to class the nodes and links either as true or false to make only certain levels of the tree visible. The visible nodes will then be rescaled and repositioned based on the visible tree size, which can be shown by Figure 8. In this way, we avoid rebuilding the tree each time and the update function for the tree runs smoothly.

However, this creates two problems. First of all, since the tree is not reconstructed each time, the visible tree nodes are only positioned using a linear scaling function based on the overall tree structure. As a result, some nodes would be very close to each other, leading to visually bad visualizations. The other problem was that it would lead to a problem for other interactions such as collapsing and expanding for the tree. Ideally when we expand or collapse the tree, we only need to flip between node.children and node._children. This idea is quite different from the current design. Thus we want to switch to a different approach to update the tree structure so that different interactions for updating the tree would share a consistent updating method.

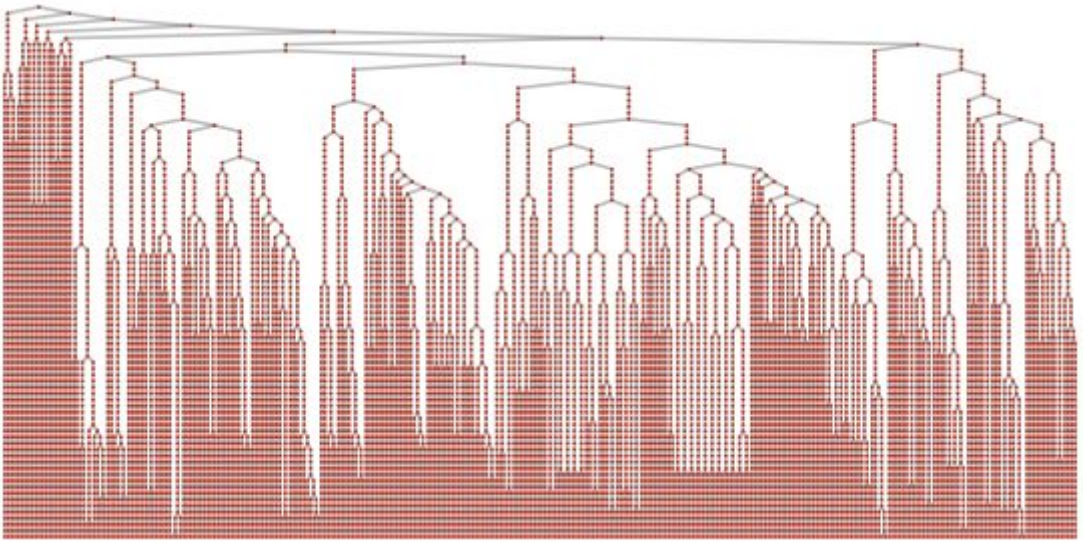

Figure 7. Tree Layout that Consists of all the Nodes during the Merging Process.

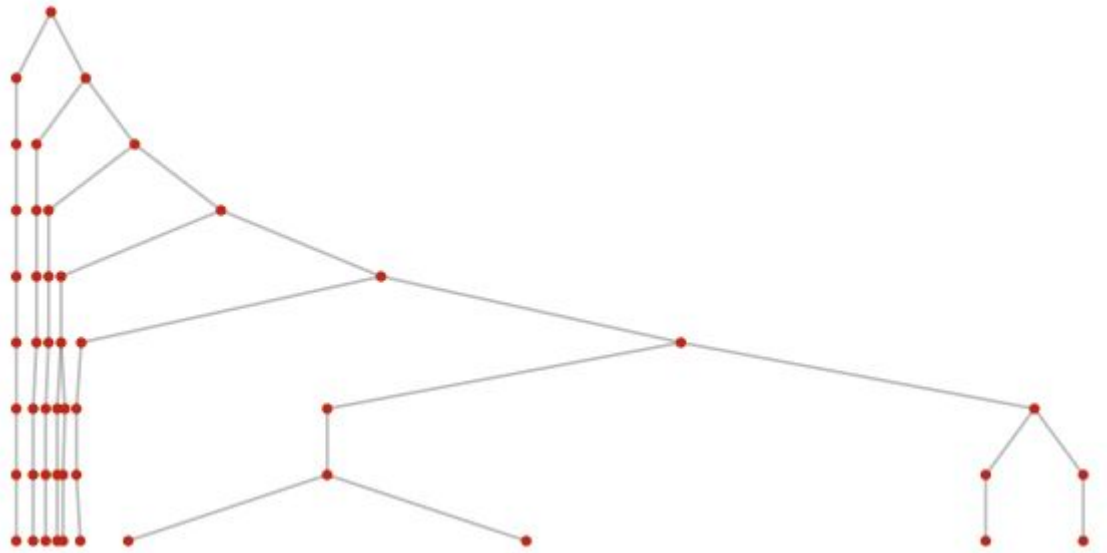

Figure 8. Tree Layout at Certain Persistence Level using Resizing and Repositioning Method

Then we change the update function. Instead of using a filter function to directly classify nodes, we change the children to _children if we do not want to show certain levels of the tree. In this way it would be consistent with the interaction for collapsing and expanding. Original we remove all the tree nodes and rebuild the tree. However, this causes a problem when the event function is outside the tree class. Due to separation of concerns, we do not want to have the tree listen to the events or call other classes since it should not know about other classes. Thus, instead of removing and appending nodes everything, we still use the similar method to make nodes that have children attribute visible. The nodes that have _children attribute will be invisible.
In this case, it would lead to better tree layout since it uses d3.tree to relayout the tree, which is shown by Figure 9.
Compared with the previous tree structure, this would lead to better tree visualization.
We then add the interactions such as buttons and sliders for the user to update the tree. Interaction with other visualizations are also added such that when users click on the tree node, the plots would display the partition represented by the node and the information shown as texts would be updated in the data section. When the mouse hovers over the node, a tooltip would show up to give the information of the tree node. When the users double click the tree, the node would be collapsed or expanded.
Animations are also added in the end to make the transition look better when the tree updates.

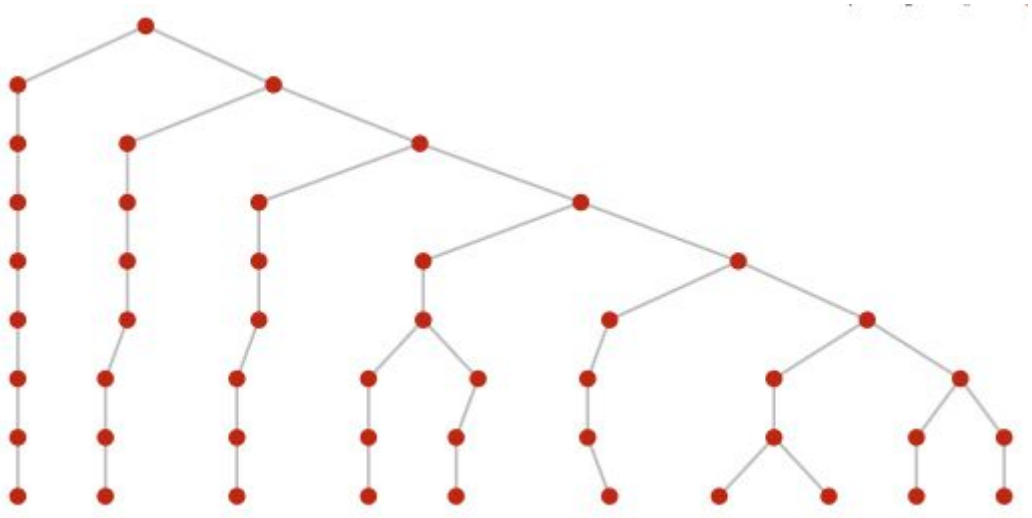

Figure 9. Tree Layout at same Persistence Level as Figure 8, using d3.tree() to Relayout

- **Plot Design**

The plots would display the raw data based on user's selection. To implement the plot section, we create a pull down button for users to choose which kind of the plot they want to see. After clicking the specific type of plot, the plots will change accordingly. The button is shown at the upper right side of the following image.

We will illustrate the 4 kinds of plots for our example of the nuclear simulation dataset which has five attributes in total.

In the rawdata plot, n-1 plots would be displayed if there are n attributes in the data. In this case, the Pu_TOT attribute is the dependent variable. Thus there would be 4 scatter plots of the relationship between the dependent variable and four other attributes, which are BU_UOX, BU_MOX, FR_MOX, TC_MOX. In interactive design of the plot, when user hovers the mouse on some specific point of the plot, the tool tip will show the information of the point which includes the value of all the attributes of the point. The color of the points will variate according to the value of the dependent variable from blue to red. The label of the axis show the meaning of the corresponding axis. What's more, we would implement the link feature in the future if we have time, which means if we click on some point on one plot, the corresponding point on the other plots will be highlighted accordingly. The example of the rawdata plot is shown in Figure 10, where x-axis represents the dependent variable and y-axis represents the independent variable.

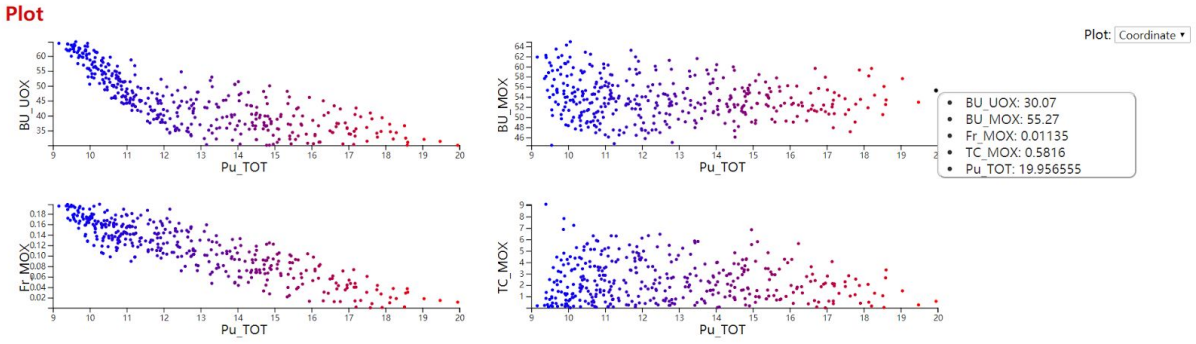

Figure 10. Coordinate Plots of each Independent Variable vs Dependent Variable

In pairwise plot, the relationship between each independent variable will be displayed. There would be C(n-1,2) plots displayed if there are n attributes in the data. In this case, the Pu_TOT attribute represents the dependent variable. Thus there would be C(4,2) = 6 scatter plots of the relationship between four other independent attributes, which are BU_UOX, BU_MOX, FR_MOX, TC_MOX. In interactive design of the plot, when user hover the mouse on some specific point of the plot, the tool tip will show the information of the point which is the value of all the other attribute of the point. The color of the points will variate according to the value of the dependent variable for the point from blue to red. The label of the axis shows the meaning of the corresponding axis. What's more, we would implement the link feature in the future if we have time, which means if we click on some point on one plot, the corresponding point on the other plots will be highlighted accordingly. The example of the pairwise plot is shown by Figure 11.

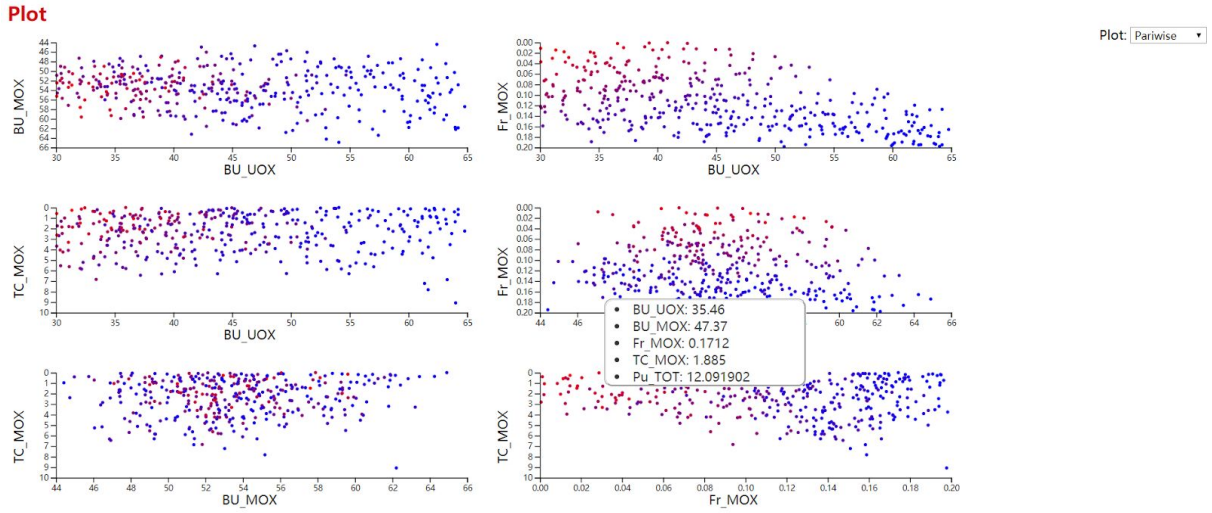

Figure 11. Pairwise Plot that Shows the Relationship between each Independent Variable

In box plot, n plots would be displayed if there are n attributes in the data. In this case, there would be five boxplots to represent the statistical information of five attributes from the data, which are BU_UOX, BU_MOX, FR_MOX, TC_MOX and Pu_TOT. An example of the box plot is shown by Figure 12.

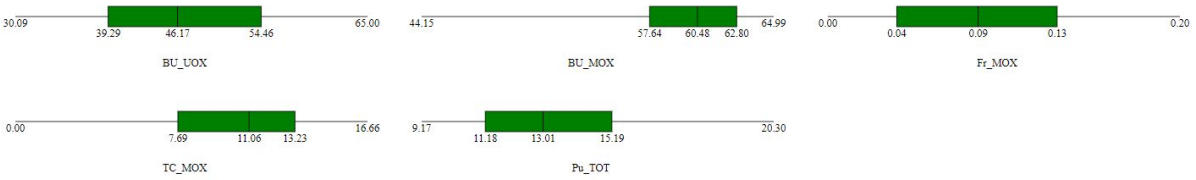

Figure 12. Box Plot for Selected Partition

In histogram plot, n plots would be displayed if there are n attributes in the data. In this case, there would be five boxplots to represent the distribution of five attributes from the data, which are BU_UOX, BU_MOX, FR_MOX, TC_MOX and Pu_TOT. An example of the histogram plot is shown by Figure 13.

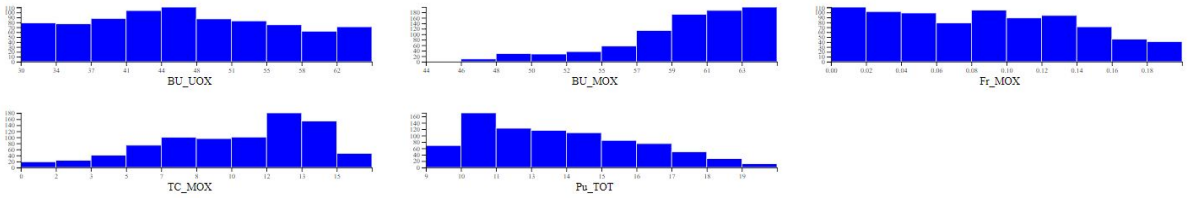

Figure 13. Histogram for Selected Partition

The interaction between the Tree and the Plot is added such that when the user clicks on specific tree node, the plot would be updated to represents the partition represented by the tree node.
Another interaction is added to allow the user to select the dependent variable, which is shown as a pull down button on the upper left corner of the plot. This would give the user different scatter plots.

- **Data Design**

A data section is also implemented to show the information of the data. This would help the user to understand the basic information of the raw dataset. It also shows the current persistence level and partition size, which are the filters used to show the current tree structure. When the user selects a specific partition by clicking on a tree node, it also shows the information of the selected partition. The interaction is added so that the information in the data section gets updated when the event fires.

- **Manager**

To follow the rule of separation of concerns, we make sure different classes do not know about each other. Thus we create a manager class that knows about all the classes. The main function as well as all the event listeners are created in this class.

- **Partition**

This class includes the the partition information of the raw data, which would be used to create other visualizations.

- **Event**

Though called “Event”, this class only represents the slider that is used in the tree structure.

- **Optional Features**

Different methods have been tried to display the PCA Projection as well as the regression curve for each plot.

Final Implementation

The tree can be updated based on the persistence level of user’s interest, which can be achieved by clicking on the button or using the slider, which is shown by Figure 15. It can also be updated based on the partition size of user’s interest since sometimes the user is not interested in the partition that only consists of a few points, which might be outliers. The buttons labelled increase/decrease partition size allow the user to do this, which is shown by Figure 16. Also, the the user hovers the mouse over a node, it will display the node information as a tooltip, which is shown by Figure 14. When the user clicks on the tree node, the partition will be displayed in the plot section and the partition information will be updated

in the data section, which is shown by Figure 17. When the user double clicks on the tree node, the node would either expand or collapse. Which is shown by Figure 18.

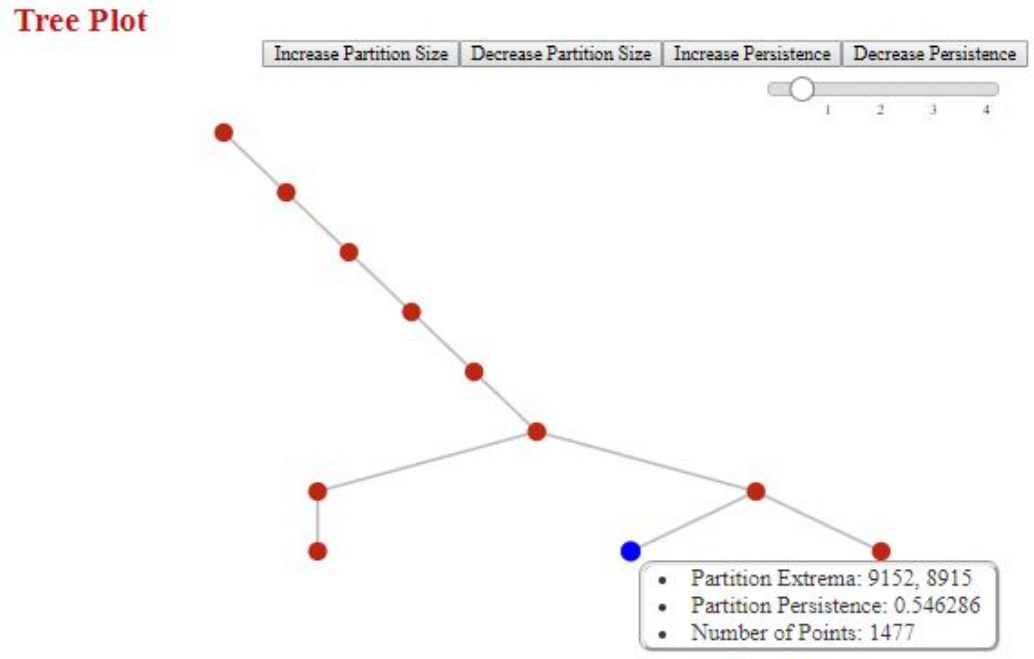

Figure 14. Partition Information Shows up when User Hovers Mouse over the Node

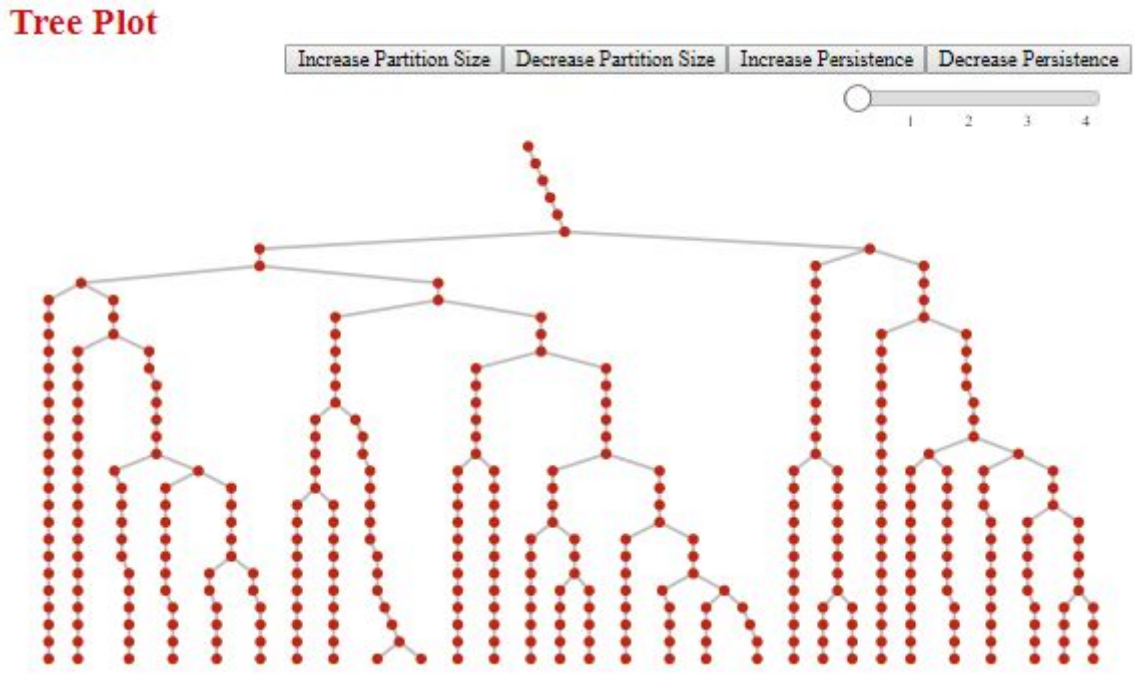

Figure 15. Tree Layout at different Persistence Level from Figure 9.

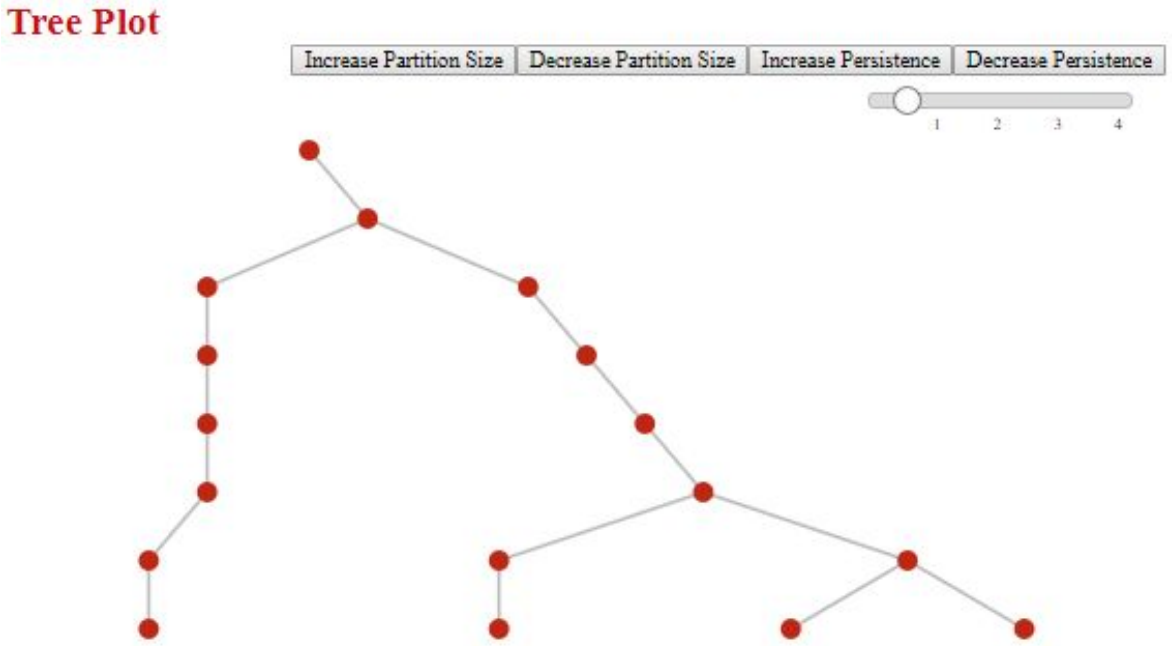

Figure 16. Tree Layout at different Partition Size from Figure 9.

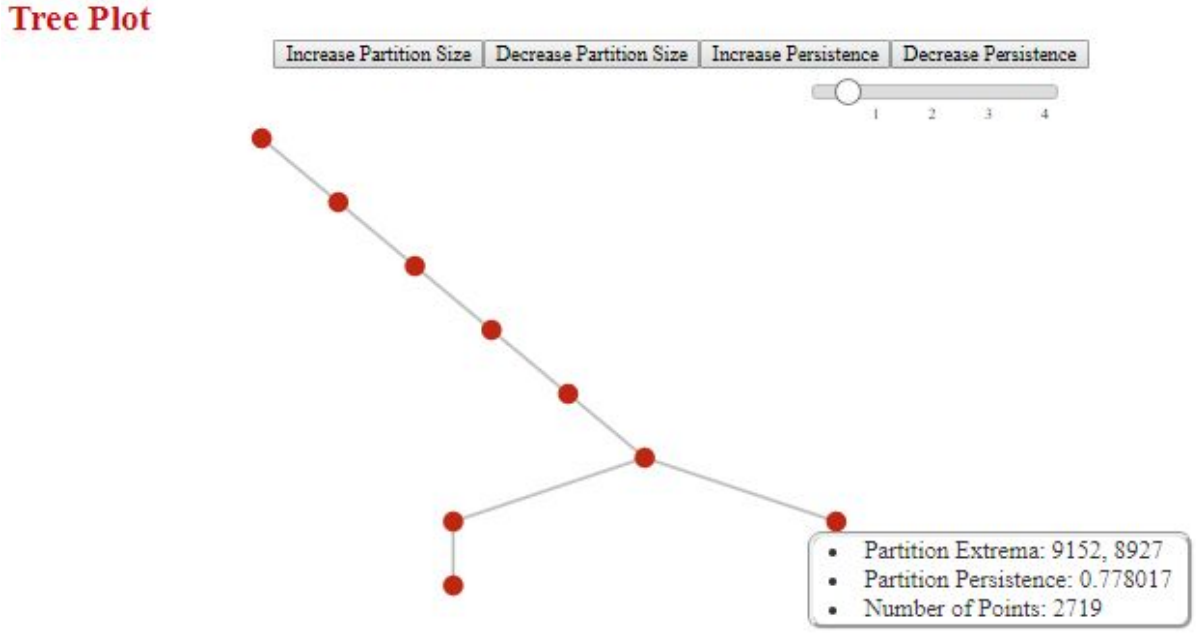

Figure 17. Tree Layout from Figure 9. with a Tree Node Collpased

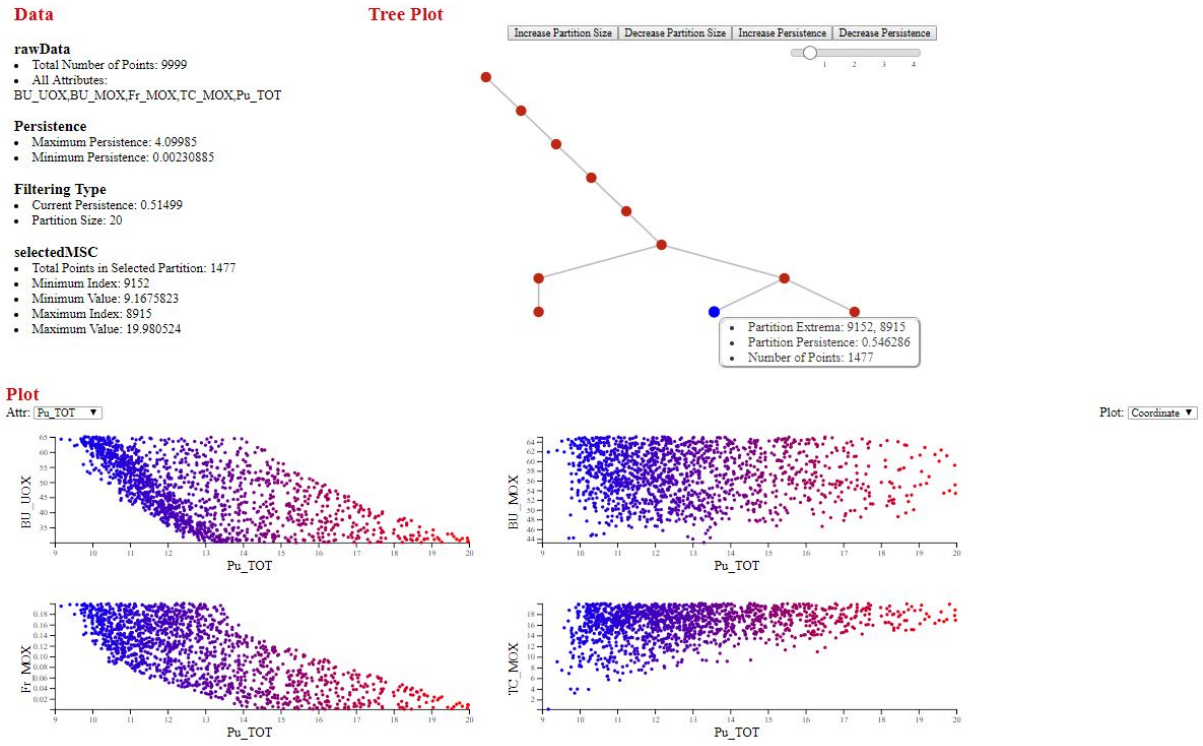

Figure 18. Plot Section Displays Partition when a Tree Node is Clicked

The final plot section includes four kinds of plots as well as a pull down button for user to select dependent variable of interest. The user is able to select the dependent variable of interest in the dataset, which is shown by Figure 19. The user is also able to select the type of plot they are interested in, which is shown by Figure 20. For the scatter plot, when the user hovers the mouse over the point, the value of all the attributes of the point will show up as a tooltip. An example of this is shown by Figure 21.

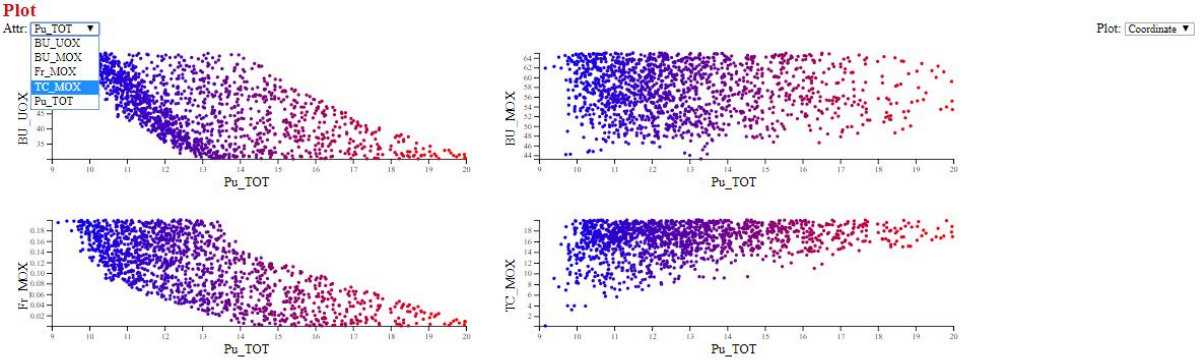

Figure 19. Pull down Button for the User to Select Dependent Variable

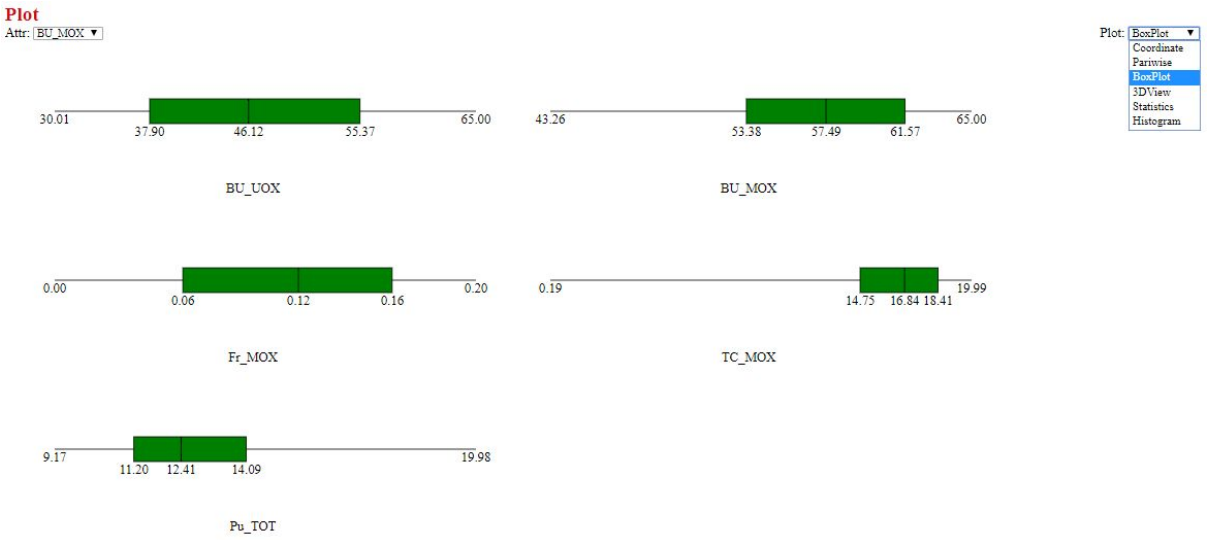

Figure 20. Pull down Button for the User to Select Plot

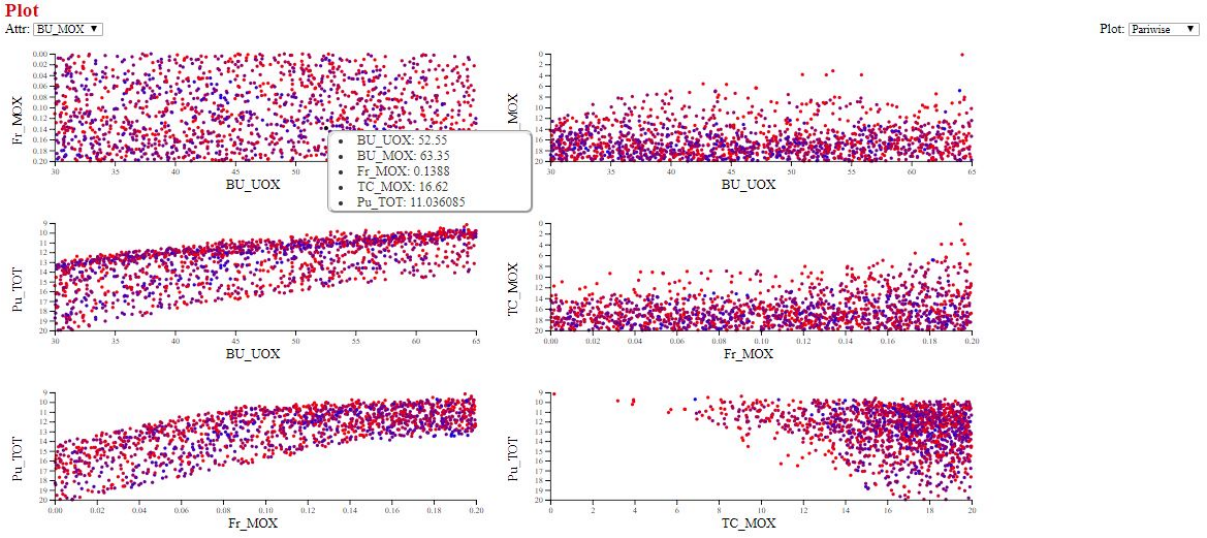

Figure 21. Point Information Shows up when User Hovers Mouse over a Point

The data section also gets updated based on current persistence level, partition size as well

as user's selection, which is shown by Figure 22.

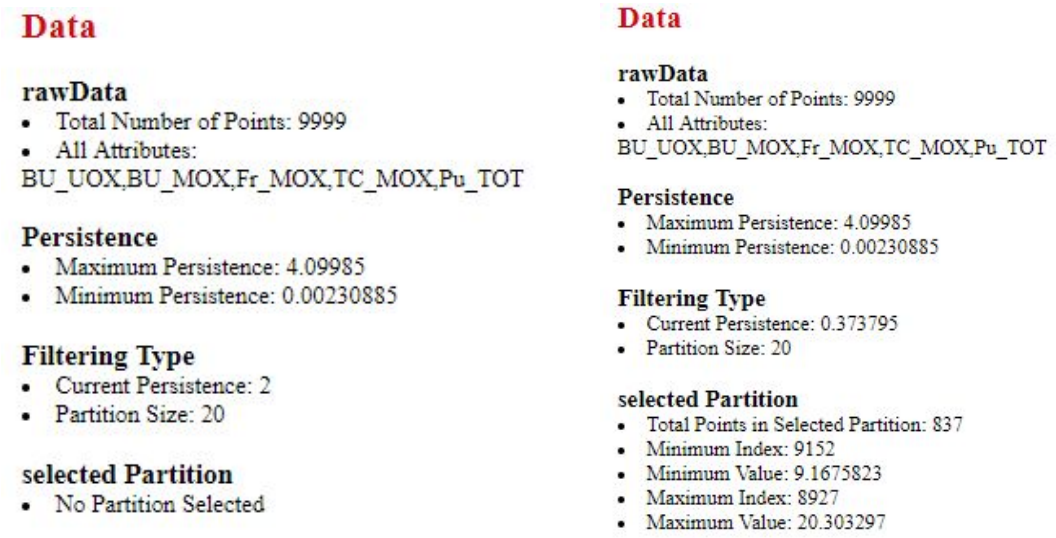

Figure 22. Data Section Gets Updated during Interaction

Evaluation:

There would be two main target user groups for our visualizations. The first group would be data analysts. And the second would be domain experts. In many cases, these two groups would work together to see in the visualizations make sense.

For example, when we see the top few levels of the tree, we find that many children partition has comparatively small size. Although we are not sure whether those partitions are considered as outliers, they would not make much sense when they are displayed by the plot. Thus we want to increase the partition size to filter out those small partitions.

After we increase the partition size to 50, we see that at persistence level 2.899, the tree node is splitted into two comparatively large partitions. When we compare the scatter plots of the two children, we see that the scatter plot between Pu_TOT and BU_MOX behaves differently.

At persistence level 0.778, a partition is splitted into two children of large size. When we compare the scatter plots of the two children, we see that the scatter plot between Pu_TOT and TC_MOX behaves differently.

At persistence level 0.546, a partition is also splitted into two children of comparatively large size. When we compare the scatter plots of the two children, we see that the scatter plot between Pu_TOT and BU_MOX behaves differently.

Different scatter plots show that the point clouds might be clustered differently in the high dimensional space. We can see that there is something interesting going on. However, we need to work with the domain experts to understand more about these features.

From the domain experts, we understand the physical meaning of each attribute shown as following.

- PuTOT: total plutonium produced
- BU_MOX: Burn-ups for Mixed uranium-plutonium oxide fuel
- BU_UOX: Burn-ups for Uranium oxide fuel
- FR_MOX: Amount of mixed oxide fuel in fast-reactor
- TC_MOX: Technetium-99 in the mixed oxide fuel

The factor that the Pu_TOT vs BU_MOX scatter plots of two nodes at same persistence level behaves differently means there might be some internal reactions going on in the simulation. For example, the burn-ups for mixed uranium-plutonium oxide fuel might follow two or more reaction processes to produce plutonium directly or indirectly.
Similarly, the mixed oxide fuel in fast-reactor might also follow two or more reaction process to produce plutonium directly or indirectly.
Thus our visualization shows that there might be some interesting features in the high dimensional space, which would help domain experts to better understand the data. However, it is up to the domain experts to extract the corresponding features or explain the internal process.
One thing that may help the user understand the visualization better would be fitting regression curves to each scatter plot. We mentioned this as an optional feature in the proposal, but was not able to achieve it since calculating kernel regression in the runtime would significantly slow down the visualization. Simply fitting linear regression curves would not analytically help the users either. However, we believe certain types of regression curves would definitely help the users to understand the data better.