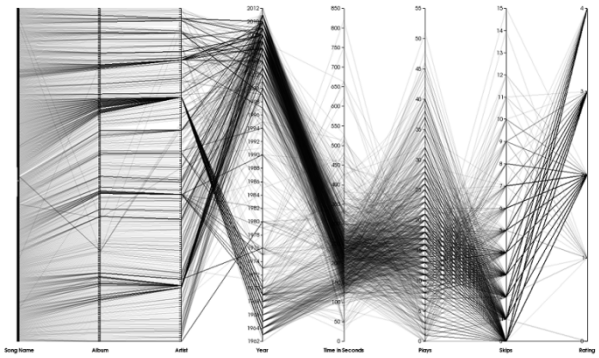
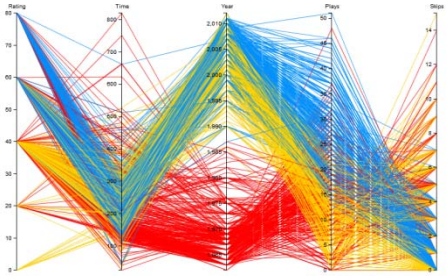
Motivation

My final project is a way for to make parallel coordinates graph work for a project I am working on through the Tetherless World Constellation, TWC. The project is a visualization system which allows researchers to place their data behind. Then users can select specific dimensions. Then scatter plots are made for every dimension and a parallel coordinates graph is made of all of dimensions. A major part of designing a parallel coordinates chart is the ordering of the axes. The system has to support any data, so it is not possible for the axes to be order for all possible combinations of dimensions. I wanted to create a system that could automatically order the axes based on the data. Another design problem was the randomness of the k-means clustering. The data can selected for a specific time and would be the graphs were re-rendered. This caused the k-means clusters to appear to move. This made analysis confusing as it appeared the data radically shifted even when nothing changed. I wanted to keep the clustering because it aided in the analysis of trends.

Evolution of the Visualization

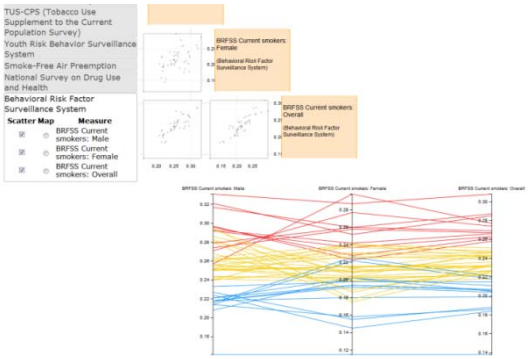
This visualization started as my submission for Assignment 5 on high dimensional data, which is shown to the right. The data was my personal listening habits on Itunes. The original version was created using VTK. It had no clustering of any kind.





For my midterm project, which is show to the left, I wanted to improve upon this visualization. I made multiple changes to improve the visualization only for the specific data

set. Instead of using VTK, D3 was used for this version. D3 offered more flexibility and control over the details of the visualization. I implemented a version of k-means clustering which initially was only able to support the specific data set.



After this version I was tasked with was adding the parallel coordinates to the project for the TWC. This required generalizing the K-means algorithm to support any general set of data. The final result is shown to the left.

Hypothesis

My plan was to use polynomial interpolants and the error between the data and the interpolant as a scale of how well the data exhibits a specific trend. I would combine these errors with information about the interpolant as the weights of edges in a weighted undirected graph, where each node was a dimension. I could then find the shortest Hamiltonian path in this graph to find the optimal ordering based on the weights I assigned. I was unsure how to deal with normalization, and what degree polynomials for interpolation to be used.

Feedback

My feedback stage focused on what characteristics are shown well on a parallel coordinates chart. I needed this information in order to implement a system that would automatically order the axes based on the data. Participants in the user study made it clear that linear trends, whether they be positive or negative, are easy to see when graphed in a parallel coordinates chart. Users also could see quadratic relationships, especially when the region where the data fits is was only

increasing or decreasing. When the data crossed a maxima or minima the parallel coordinates became cluttered. Users also confirmed a proposed way to normalize the weights.

Design issues were also answered. Everyone agreed that k-means clustering helped; however, some were confused by its lack of inherent meaning. The choice of colors was also well liked. Some people suggested slight changes to increase contrast.

Final Version

For the final version I implemented several algorithms in order to determine how well a set of data exhibits a trend. The main part of my algorithm is interpolation. Below I outline the specific algorithm for a linear least squares, then I generalize to any degree polynomial:

-Input set of *n* data points (x_i,y_i) where *i* is between 1 and *n*
-Find a polynomial *p(x)=a+bx* that minimizes *E*, where

$$E = \sum_{i=1}^{n} (a + bx_i - y_i)^2$$

-In order to minimize *E* the partial derivatives of *E* with respect to *a* and *b*.

$$E_{a=} \sum_{i=1}^{n} 2(a + bx_i - y_i)$$

$$E_{b=} \sum_{i=1}^{n} 2x_i(a + bx_i - y_i)$$

-Then these derivatives must be set to 0 and the values of *a* and *b* should be found

$$0_{=} \sum_{i=1}^{n} 2(a + bx_i - y_i)$$

$$0_{=} \sum_{i=1}^{n} 2x_i(a + bx_i - y_i)$$

-This leads to the matrix problem of

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \end{bmatrix}$$

-Using Gaussian Elimination and Back substitution this can be solved for *a* and *b*. With *a* and *b*, it is possible to compute E which is then used to compute the weights.

-For a polynomial of degree m, $p_m(x)=c_1+c_2x+c_3x^2+\ldots+c_{m+1}x^m$, the following matrix problem has to be solved to find ci for I between 1 and m+1

$$\begin{bmatrix} n & \sum x_i & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & & \sum x_i^{m+1} \\ \vdots & & \ddots & \vdots \\ \sum x_i^m & \sum x_i^{m+1} & \cdots & \sum x_i^{2m} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \vdots \\ \sum y_i x_i^m \end{bmatrix}$$

The Least Squares algorithm is used to compute the error for the data for linear and quadratic polynomials, but I did implement a general algorithm which I used to consider the use of higher degree polynomials. I decided only to use the two types of polynomials, because they are the fastest to compute, and higher degree polynomials have a higher possibility of being cluttered on a parallel coordinates chart. The algorithm is $O(nm+m^2+O(\text{Gaussian Elimination and Back Substitution})$. Gaussian Elimination and Back substitution are somewhat more computationally inefficient. In Gaussian Elimination the problem is reduced to a upper triangular matrix. At the ith iteration, the jth row for j between i+1 and the number of rows in the matrix and constant vector is replaced by the difference of the jth row and the product of the ith row and the quotient of the value at the ith row and column and the value at the jth row and ith column. This leaves all values below the ith row in the ith column equal to zero. Once this is performed for i from 1 to the number of rows minus 1, back substitution is used to find the values of the variables. Gaussian elimination is $O(n^3)$ and back substitution is $O(n^2)$. Therefore for small degree polynomials, the least squares algorithm has a running time of $O(n^3)$.

Once I have the error from the algorithm I must combine them. For each type of error and each set of dimensions I get the error for one dimension as x and the other as y, and then I swap which dimension is x and which is y. This is because the error is only measured between the interpolant value at an x value and the y value it should be. For the linear version I average these two errors to create a single linear error. However, for the quadratic error I select the smaller error and use that.

This is because, the inverse of a line is a line, but the inverse of a quadratic polynomial is a square root function which cannot be found using my least squares error. The quadratic error also includes a penalty for when the data is not fit to a region of the parabola that is only decreasing or increasing. After I have the linear error and the quadratic error, I select the smaller of the two to be the error. This is because if the data shows a very good quadratic trend its associated linear error would be very high. All of these errors are normalized before compared. Each error is divided the range of the y values. This method of normalization was supported in the user study.

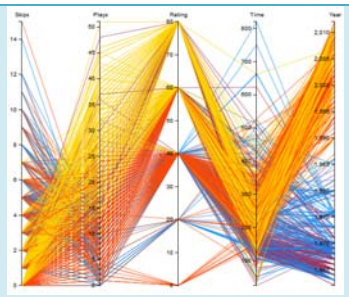
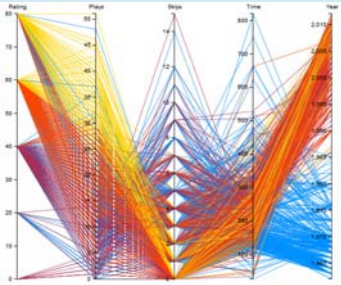
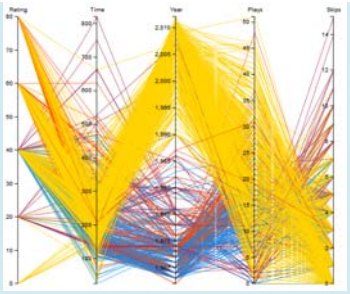
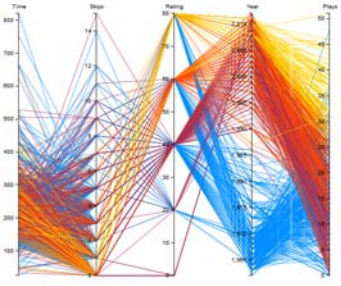
Once the errors for each combination of dimensions are determined, the ordering with the lowest score is chosen. This problem is similar to the Traveling Salesman problem, which has algorithms to solve it better than brute force. However, for time reasons I used a brute force method. Brute force requires O(n!) time, but for my use, n would remain relatively small.

As well as developing an algorithm to automatically order the dimensions, I also created an algorithm which limited the randomness of k-means clustering. After computing the labels, I reassign the labels based on the average normalized vector distance of each cluster. The vector distance is normalized so that the smallest value for a dimension is changed to zero and the largest value is changed to one. Then the distances of all vectors for a specific label are averaged. The labels are then assigned based on the ranking of their average normalized vector distance. This caused k-means to appear to be less random each time it renders.

Results

To inspect my algorithm I used the data from my midterm. Each point in this data represents a song. The dimensions are how many times the song has been skipped and played, the rating I assigned it, how long it is, and the year it was released. I looked at using both types of error, and only one at a time.

The following table shows various orderings of the data used for my midterm report.

Chart	Order	Score
	Skips, Plays, Rating, Time, Year	102.8840
	Rating, Plays, Skips, Time, Year	111.4572
	Rating, Time, Year, Plays, Skips	163.3445
	Time, Skips, Rating, Year, Plays	216.6495

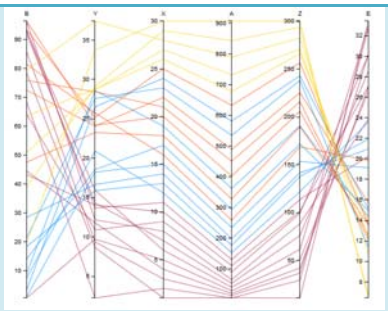
The first chart is what my algorithm said was the best using both linear and quadratic error. The second chart is what my algorithm decided was the best when using only linear error. The third chart is

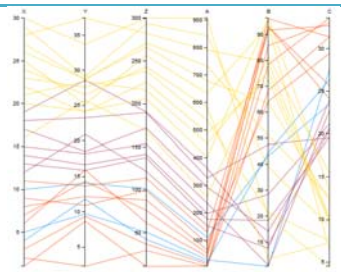
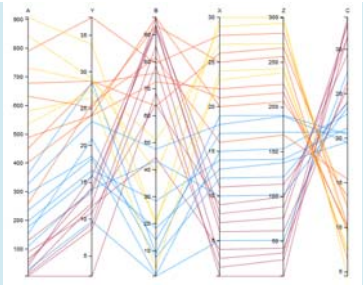
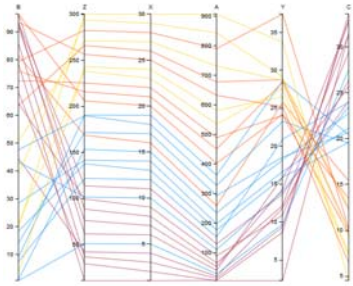
what I decided to use on my midterm report. The final chart is the worst possible ordering. All scores are shown using both linear and quadratic error.

By inspecting all the errors I found that there were large differences, and that in some cases the linear error was smaller, and others the quadratic error was smaller. This affirms the use of both to get a better understanding of the data.

The itunes data is not the best suited data for the automatic ordering, because many of the trends are not linear or quadratic, but are much more complex. The system I am planning on working with is going to use a lot of biomedical data. The data currently being used to test it either has linear trends or no trends at all. In order to test on this type of data I also did a similar analysis on contrived data with these characteristics.

For the data, dimension X is all integers between one and thirty. Dimension Y is X plus a random number between zero and 10. Dimension Z is dimension X times 10 plus a random number between zero and ten. Dimension A is dimension X squared plus a random number between zero and ten. Dimension B is a set of random data between zero and one hundred. Dimension E is thirty minus dimension X plus a random number between one and ten.

Chart	Order	Score
	B, Y, X, A, Z, C	3.0943

	X, Y, Z, A, B, C	6.61386
	A, Y, B, X, Z, C	6.1919
	B, Z, X, A, Y, C	3.6807

All of these used both linear and quadratic error. The first graph is the best ordering, the second graph is the worst. The third graph was a randomly selected ordering. The final graph is what I decided to use for the ordering. My ordering did receive a relatively low score, but not the lowest. This is alright, because in the system we are setting there is a scatter plot matrix of every combination of matrices. When selecting the ordering of dimensions, interesting combinations might be selected. This trend may not lead to a low error in this system, and the lowest ordering might not have that combination adjacent to each other. However, the interesting trend would not be hidden; it could still be found in the scatter plot.

Conclusion and Future Changes

The visualization system is able to create the best ordering given a set of weights. The weights may not be perfect, but they do effectively find linear and quadratic trends within the data. For data with these trends, it is good at ordering the axes in such a way that these trends are shown. The design decisions I made myself had lower scores; however, they were not the best ordering usually. Most data will not have simple trends, so it would be useful to try and numerically represent more complex and interesting trends.

Bibliography

Bostock, Michael. "d3.js." bost.ocks.org. N.p., n.d. Web. 24 Apr. 2012. <http://mbostock.github.com/d3/>.

Heath, Michael T. Scientific computing: an introductory survey. 2nd ed. Boston: McGraw-Hill, 2002. Print.