Color Block and Query-Driven Visualization

Report for CPSC 533, Instructor: Tamara Munzner
Ivan Zhao, University of British Columbia

Abstract

This paper discusses a general visualization technique focused on displaying correlations. It is capable of handling structured, high dimensional data, as well as unstructured data such as text. In this technique, we use a rectangular block to represent the *collection* of documents that satisfy a particular query, with the height of the block corresponding to the number of documents. Multiple blocks can be displayed at the same time. To show the entire dataset of a high dimensional structured data, all the distinct values (or range of values) in each data dimension are queried individually, resulting an arrangement is very similar to Parallel Coordinates. To display the correlation of block A with a new query condition B, we first assign a color to the condition B, then vertically paint portion of block A with the color based on the strength of the correlation; therefore, high correlation results in a single dominant color in a block. Moreover, we compare our approach with Scatterplot and Parallel Coordinates, arguing that this *query-driven* approach could be more suitable in certain circumstances.

1. Introduction

The increasing amount of data forces us to compress and abstract it. Visualization is one form of abstraction. You make up a marking scheme that transforms the data from its original format --- either numerical, nominal, or unstructured, such as text --- into a visual format made out of dozen visual features, in hopes that the outcome is perceptually distinguishable to the viewer.

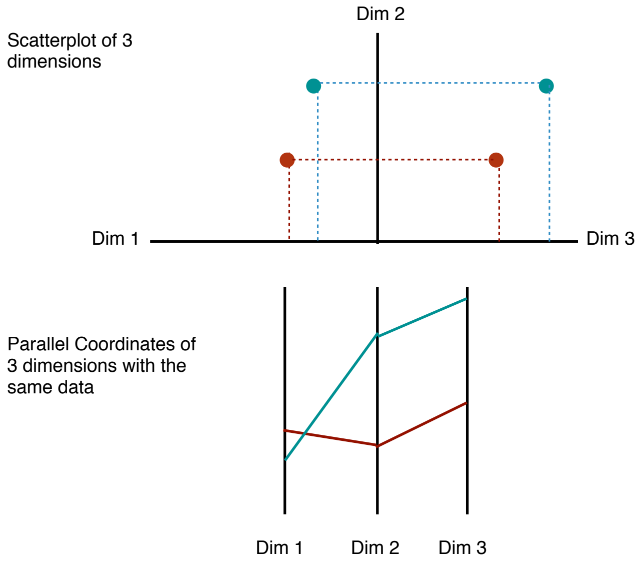
Like any other forms of abstraction, there is the inevitable loss of information during the process. Therefore, which area of original information you are willing to give up is now a design choice. Good design happens when the tradeoff is compensated by quirks of human perception, as well as the characteristics of the information on hand.

In this study, we are investigating a general visualization technique called *Color Block*, with a emphasis on showing the correlation in the data. Traditionally, Scatterplot and Parallel Coordinates are the standard, effective methods for this task; but we decide to seek other paradigms, focusing on alternative design tradeoffs. For example, Scatterplot and Parallel Coordinates share the characteristic that data entries are *individually* rendered, hoping that proper fitting and filtering will reveal their underlying pattern. We suggest a more *query-oriented* approach to display the data, in which only the data that satisfies the query will be shown as a single visual entity (a block); however, we can still show the entire dataset if we ask the disjoint queries that are able to cover the entire universe of each dimension.

Of course, this is not the only design tradeoff we are making in this study. More will be discussed as we address the rationals behind designing *Color Block* and our query-driven approach.

2. Scatterplot, Parallel Coordinates and Mosaic

Scatterplot and Parallel Coordinates [1] are the de facto methods to display correlations. Their underlying principles are the same: every dimension in the data has a correspondent visual axis; and for every piece of correlational evidence between two dimensions, it is displayed as a marker that is an *unique* combination of two values, one from each dimension.


Scatterplot of 3 dimensions
Dim 2
Dim 1
Dim 3
Parallel Coordinates of 3 dimensions with the same data
Dim 1
Dim 2
Dim 3

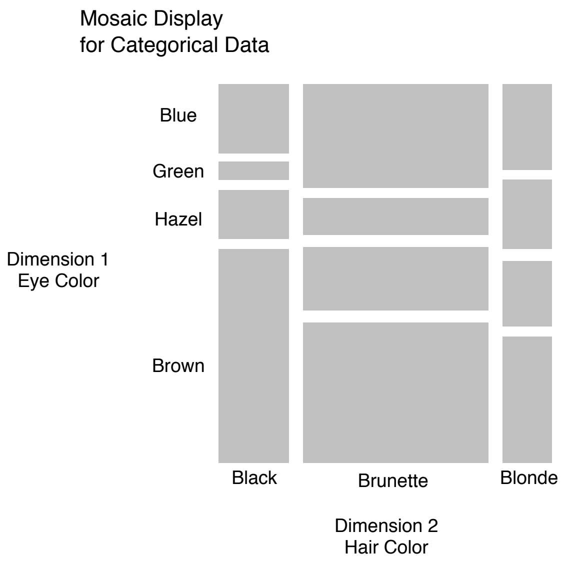
The figure above shows the same data displayed by both methods. In the case of the Scatterplot, the two axes are placed orthogonally, therefore in the Cartesian space between them, a single dot marker is sufficient to represent an unique value combination of the two dimensions.

In Parallel Coordinate the dimension axes are parallel to each other, which saves a lot of space, allowing far more dimensions to be displayed at the same time. But this multidimensional power does not come for free: as the axes are no longer orthogonal, a single dot fails to be an unique combination of values in these two neighboring axes. Therefore, Parallel Coordinate has to use the next most efficient representation, which is a line, to literally draw a connection between the neighboring axes.

At first, this does not appear to be a bad tradeoff, but as the number of data element increases, the connecting lines between axes could become very clustered, in a rate much faster than its Scatterplot counter part; this poses the biggest challenge to the Parallel Coordinates technique.

Moreover, both Scatterplot and Parallel Coordinates have limited performance with categorical data. Because there are only few possible values in each dimension, numerous lines and dots would all compile at the same locations, difficult to tell the actual quantity.

Mosaic Display [2] effectively addresses this issue. In the figure below, the correlation between the nominal dimension *Eye Color* and *Hair Color* are shown as blocks; the width and the height of each block corresponds to the *percentage* of its underlying categorical value; therefore, a large block entails a strong correlation between the two values it represents.



The limitations of this approach are obvious: there can be only two dimensions at the same time, and you also "waste" a large space in the middle of each block.

So isn't there a way to display correlation that is categorical-data-friendly, and also supports multi-dimensions at the same time? If we were still in the Victorian age, with limited B&W printing and zero interactivity, these three cases above might very likely cover up most the ground, as we can either represent a correlational entry as a dot/glyph (Scatterplot), a line (Parallel Coordinates), or an area (Mosaic). But with the help of interactivity and color display, we gain new dimensions to seek alternative tradeoffs that can satisfy all the requirements above; our result is the *Color Block* visualization.

3.1 Color Block Visualization, the *Pile* Metaphor

Let's start with a metaphor: imagine you are in the 1970s and all your documents are papers. Every time a interest comes into mind, such as "I wonder how many people in my company are aged over 40", you make a request to your assistant, who would then provide *copies* of all the satisfied documents. So after a while, your desk would be covered by numerous piles of documents, each ties to a request you made; they would probably have different thickness and the thicker a pile, the more documents satisfy that particular request.

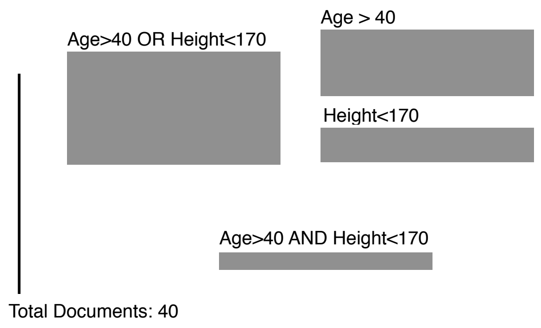
Our study is tightly related to this metaphor. There are two main components. First is the query system (the assistant in the metaphor), which is essentially a SQL server. When the user performs a SQL statement started with "SELECT (*) ... WHERE *conditionX*", the server returns a collection of documents that satisfy *conditionX*.

The second component is the visualization, which we call *Color Block*. In this visualization, the collection of documents returned by each query would be visually represented as a rectangular block. All blocks have the same width (important), and the height of a block corresponds to the number of documents in that collection.

Let's look at an example: you have just performed the following four queries in your company's human resource database, each returned a number of documents.

1) Age > 40	12 docs
2) Height < 170cm	8 docs
3) Age > 40 AND Height < 170cm	4 docs
4) Age > 40 OR Height <170cm	20 docs

Based on the Color Block visualization, we use four blocks of different heights to represent the four collections of documents:



Age>40 OR Height <170cm is the thickest, because it's easier for documents to satisfy that OR condition; you might also noticed that its height is the exactly the sum of blocks *Age>40* plus *Height<170*. You can also compare each block with the reference height at the bottom left, which represents the height of entire dataset as if presented as a block. There's also a minimum height limit (10 pixels in the application), just to ensure that queries with very low count get properly displayed.

Just like you can flip through a pile of documents on your desk, you can click on a block to access all the original documents that are in that block. You can also toss away or combine any number of blocks, or perform a secondary query only based on the blocks that are currently selected.

3.2 Color Block Visualization, Correlation

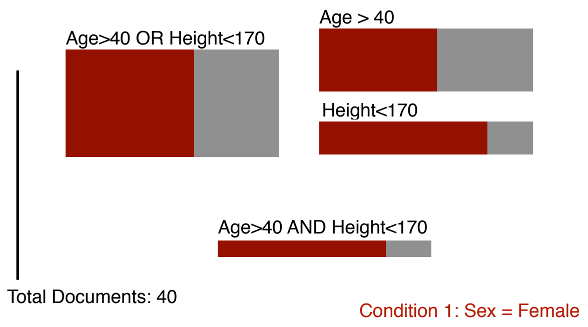
The focus of this visualization is to display correlation; but before we introduce the technique, we shall revisit our metaphor one more time.

Let's say you are sitting in front of your desk and looking at all the piles of documents, "I wonder how many people are female in each of them?" So you start calculating the *percentage* of female employees in each of the piles. After you get all the numbers down, you use a red pen to paint on the front *edge* of each pile, with the painted width proportional to the percentage in your calculation. Now you can lie back to your chair, and still have a very good idea of how each pile satisfies your interest.

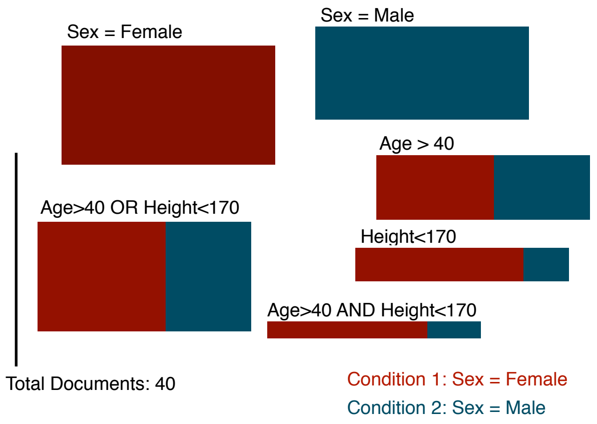
Bridging the metaphor into our visualization, we first associate the new query *Sex = Female* with a color (red); we then combine this new query with each of the old ones through an AND relationship, and obtain the updated collections of documents:

1. *Sex=Female AND Age>40* .... 7 docs, 0.58%
2. *Sex=Female AND Height<170cm* .... 6 docs, 0.75%
3. *Sex=Female AND Age>40 AND Height<170cm* .... 3 docs, 0.75%
4. *Sex=Female AND (Age>40 OR Height<170cm)* .... 13 docs, 0.65%

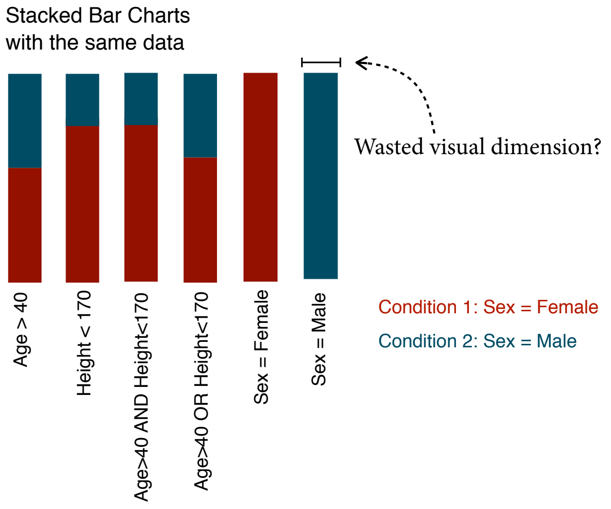
Just like we would paint certain portion of a pile's edge with a particular color, we are doing the same in the Color Block visualization. By adding color condition *Sex = Female* (red), all four blocks from previous example are updated to the graph below (Fig). As you can see, the blocks of *Heigh<170* and *Age>40 AND Height<170* are dominated by red color, that means most documents in these two block satisfy the red coloring condition.



A key to point out is that a coloring condition is an autonomous entity; it is not necessary to be associated with a specific block. Of course, you can always add a query block that is equal to a coloring condition. Let's do that, and also add a complementary condition *Sex = Male* with color blue. It would result in the following:



Doesn't it look amazingly similar to a stacked bar chart? In fact you can say it is a stacked bar chart, a *normalized* one and flip it over 90 degrees.



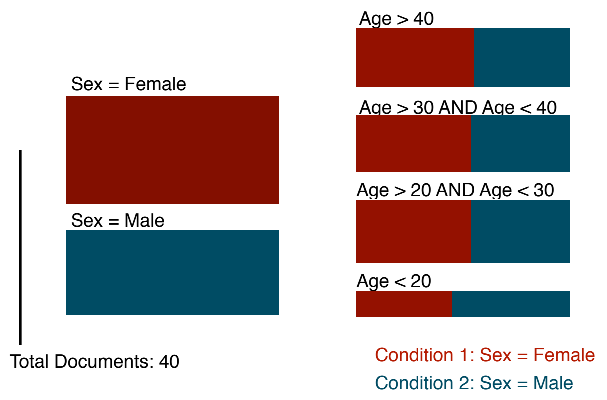
Now, if we carefully question the design of a stacked bar, the width of each normalized stacked bar is in fact a "wasted" visual dimension, that does not encode any information. You may counter argue that their uniform width provides a visual nicety, but in each Color Block, this nicety still exists as all blocks have the same fixed width. Moreover, this uniform width is necessary for a more important reason: when you are measuring correlations, it is not the absolute *count* that matters, just like you cannot dismiss the strong correlations between certain natural phenomenon only because they are observed very rarely. Therefore, in terms of correlation to a particular coloring condition, an uniform width allows all blocks to be treated "equally", disregarding how many documents they represent. In this case, the exclusiveness of a color in a block indicates the correlations, and the size of the block indicates how strongly this correlation is supported by the entire dataset. A single-colored "thick" block will obviously draw a lot of attention, but at the same time very rare blocks with little "thickness" can still raise enough attention of the viewer by being dominated by a single color.

3.2 Display Entire Dataset

So far we are only creating an individual Color Block for each of our queries. To display the entire dataset at once, it is equivalent to querying all possible values in each dimension of the dataset and displaying them all simultaneously.

This can be easily done for nominal/categorical dimensions. For a nominal dimension DimX, you only need to obtain its distinct values DimX = {V1, V2, V3...Vn}, then query them individually *DimX=V1*, *DimX=V2* ...etc. The SQL statement "SELECT DISTINCT DimX ...." will help you get the distinct values.

It gets trickier for numerical dimensions. If we query all the distinct values, very likely it would result in too many blocks over flooding our screen. Therefore we need to bin each dimension to multiple mutually exclusive ranges, and query the ranges instead. This will lead to information loss, because certain outliner cases now have to be binned and "suppressed", but it's a design tradeoff we are willing to give up.



The graph above displays the entire data dimension of *Sex* and *Age*; each block is mutually exclusive from others, and you can view it as two copies of the original datasets (one for each dimension). You might also notice that the *Age* column and the *Sex* column oddly have the different height; this is due to the labels and gaps of the blocks, which is unavoidable even if we put the labels on the side, then we still need the gaps to distinguish different blocks. This is another design tradeoff we are willing to make, the underlying rational is that it is the *relative height* of a block that matters; we assume the the importance of each query (the height of the block) is only meaningful when there are competing queries, and the analysis is performed through block-to-block comparison, instead of block-to-database comparison.

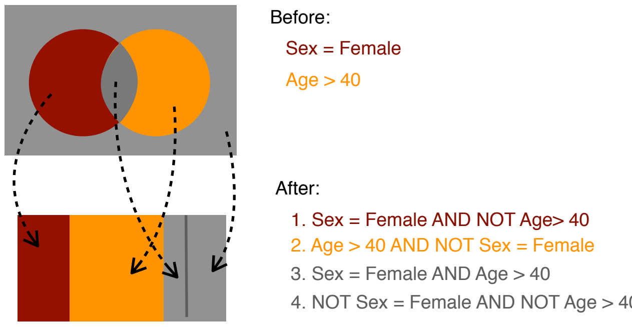
3.3 Logic Sets

By now you might have noticed a problem in the coloring scheme. What if the color conditions are not disjoint? In the example we provide, conditions *Sex=Female* and *Sex=Male* are disjoint and covering the entire dataset; but for conditions such as *Sex=Female* and *Age>40,* which are necessarily mutually exclusive, our solution is to automatically modify existing coloring conditions, and adding in extra ones so the resulting conditions can cover the entire universe of the dataset.

For example, imagine you currently have the coloring condition *Sex=Female* (red), and by adding the new condition *Age>40* (orange), the system automatically updates the current coloring conditions to:

1. Sex = Female AND NOT Age > 40
2. Age > 40 AND NOT Sex = Female
3. Sex = Female AND Age > 40
4. NOT Sex = Female AND NOT Age > 40

Then, all the blocks would be colored based on these new conditions, in the listed order. A helpful way might be to imagine the conditions in a Van Diagram:


Before:
Sex = Female
Age > 40
After:
1. Sex = Female AND NOT Age> 40
2. Age > 40 AND NOT Sex = Female
3. Sex = Female AND Age > 40
4. NOT Sex = Female AND NOT Age > 40

As you can see, only the "exclusive" cases (*Sex = Female ONLY*, *Age > 40 ONLY*) can keep their previous coloring. The newly created conditions are assigned the default color gray, and separated by a subtle dark gray line. Still, the user processess the full ability to change, delete, or reorder any of the automatically created color conditions (more on this in Section 4).

Similarly, if there are three mutually non-exclusively conditions, it would result in 8 possible outcomes, composed of 3 colored blocks, and 5 uncolored gray ones, (think of a three-circled Van Diagram). It might be confusing to distinguish among the 5 gray blocks, but the design rational is that you don't *have to,* nor you are *forced* to (given that they are very subtly separated) ---- *until*, a single disproportional gray block unavoidably draws your attention. Then, you can either mouse-over, or click on that block to investigate that particular "abnormal" coloring condition.

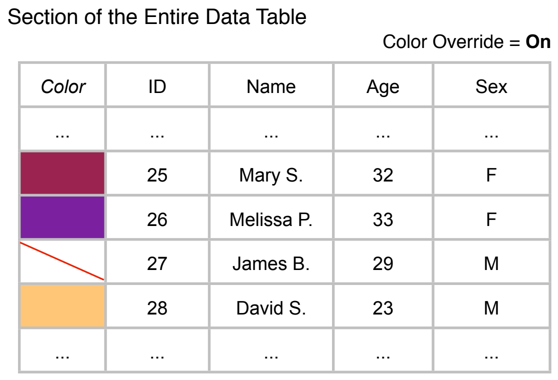
3.4 Brushing

Brushing is a very useful feature, for example in Parallel Coordinates, when certain documents are selected in the table view, their correspondent polylines in the parallel coordinates will be highlighted as well. It is particular useful when the view is very clustered, because the highlighting brings out the documents of interest.

The same philosophy could be applied to Color Block as well. When browsing on the level of individual documents, we can selectively assign each document a color, then decide to *temporally* override the current block-level color coding with the document-level colors we just assigned. Because a single document would only take up a tiny fraction of each block's width, we then need to normalize its width to take up the entire width.

For example, we assign document ID 25, 26, 28 each with a distinct color, then temporally turn on the *Color Overriding* function. Document 25, 26 satisfy the *Sex = Female* condition, so they split that block evenly. Document 28 is the only one that satisfies *Sex = Male*, so its color (yellow) fills that entire block. The same principle applies to the rest of the blocks. For a block that does not satisfy any coloring condition, it is left as the default gray color.

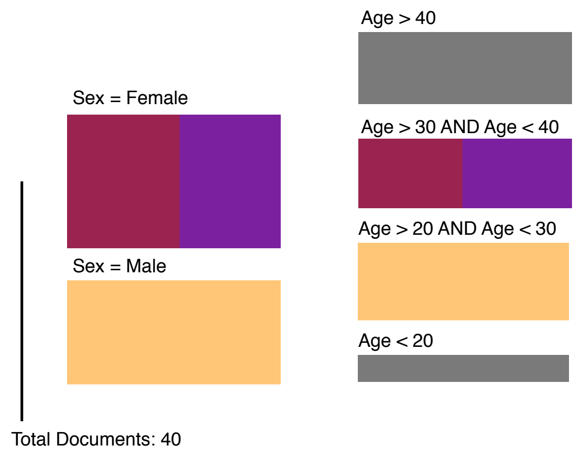
In our brushing case, positive correlation reveals, if the same *combination* of colors always appears in a single block (disregarding what that block is specifically). Of course, you can always turn off the brushing by switch off the *Color Override* function.



Section of the Entire Data Table

Color Override = **On**

Color	ID	Name	Age	Sex
...	...	...	...	...
🟥	25	Mary S.	32	F
🟪	26	Melissa P.	33	F
╲	27	James B.	29	M
🟨	28	David S.	23	M
...	...	...	...	...



4. The Application

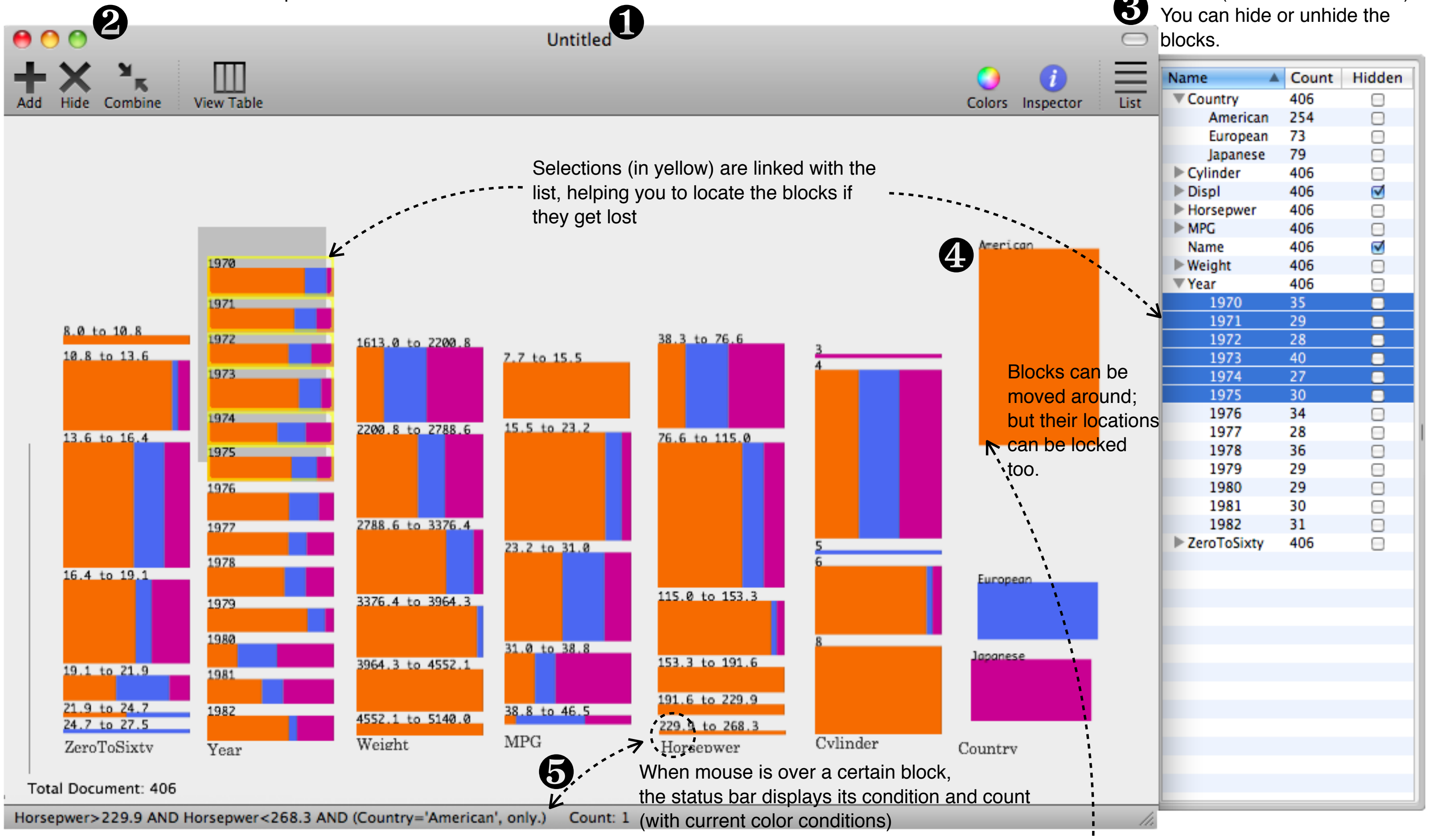
We built a Mac OSX application to implement our Color Block design. The screen shot and description are on the next page. Please note that the Brushing function is not supported currently. We will try to include it in a later release.

Mac OSX Implementation of Color Block Visualization

Operations you can perform with one block or multiple blocks.

Each document is a visualization of a single SQL table, storing all the query results as *Color Blocks.*

The List View showing all the blocks (or columns of blocks). You can hide or unhide the blocks.



Name	Count	Hidden
▼ Country	406	☐
American	254	☐
European	73	☐
Japanese	79	☐
▶ Cylinder	406	☐
▶ Displ	406	✓
▶ Horsepwer	406	☐
▶ MPG	406	☐
Name	406	✓
▶ Weight	406	☐
▼ Year	406	☐
1970	35	☐
1971	29	☐
1972	28	☐
1973	40	☐
1974	27	☐
1975	30	☐
1976	34	☐
1977	28	☐
1978	36	☐
1979	29	☐
1980	29	☐
1981	30	☐
1982	31	☐
▶ ZeroToSixty	406	☐

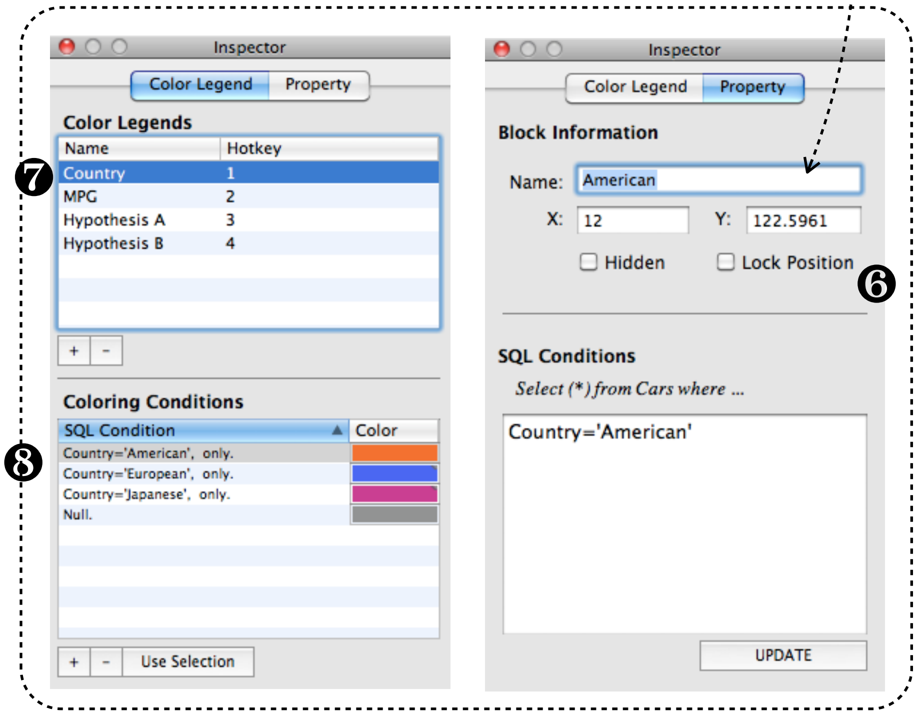
Selections (in yellow) are linked with the list, helping you to locate the blocks if they get lost

Blocks can be moved around; but their locations can be locked too.

When mouse is over a certain block, the status bar displays its condition and count (with current color conditions)

Horsepwer>229.9 AND Horsepwer<268.3 AND (Country='American', only.) Count: 1

Two views of the Inspector window.



Color Legends

Name	Hotkey
Country	1
MPG	2
Hypothesis A	3
Hypothesis B	4

Coloring Conditions

SQL Condition	Color
Country='American', only.	🟧
Country='European', only.	🟦
Country='Japanese', only.	🟪
Null.	⬜

Block Information

Name: American

X: 12 Y: 122.5961

☐ Hidden ☐ Lock Position

SQL Conditions

Select (*) from Cars where ...

Country='American'

Each legend contains a set of coloring conditions. You can assign a hot key to each legend, for fast switching.

Within each legend, there could be multiple coloring conditions, each based on a query and associated with a color.
You can also automatically create conditions based on the currently selected blocks.

When click on a single block, you can update its properties. Note that the *Name* property is different from its SQL condition. You can give a block a shorter, easier to remembered name.

The SQL condition of a block can be updated after creation.

5. Scenarios of Usage

We will look at two scenarios of usage, both with real world datasets. The first scenario uses the famous Parallel Coordinates "cars" dataset, which contains about 400 of cars' performance specs. The resulting visualization has already been shown in Section 4 on the previous page.

As you can see at the bottom left corner, the reference height indicates that there are 407 total documents in the dataset. The List View on the left shows that there are 9 dimensions in total in the dataset. We are hiding the dimension *Displ* because we are unclear what it represents; dimension *Name* is hidden because it has too many distinct nominal values that cannot all fit in the screen (but we can always look them up using the List View, or in the Table View browsing the entire documents).

If we ignore the coloring for a second and only focus on the shape (height) of the blocks, there is already a lot information in the display. For example, from the similar sized blocks in the *Year* dimension, we know that the annual car production is pretty even from 1970 to 1982; we can also see that the *American* car production is almost the combination of *European* and *Japanese*; and there are more 4 cylinder cars than other cylinder types...etc.

Now let's see some correlations. Presume we are interested in how the manufacturing country correlates to other specs of the car, so we make 3 coloring conditions based on the 3 values in the *Country* dimension (*American, European, Japanese*); if there's another country, it would be the default color gray, however there's not such entry in our dataset.

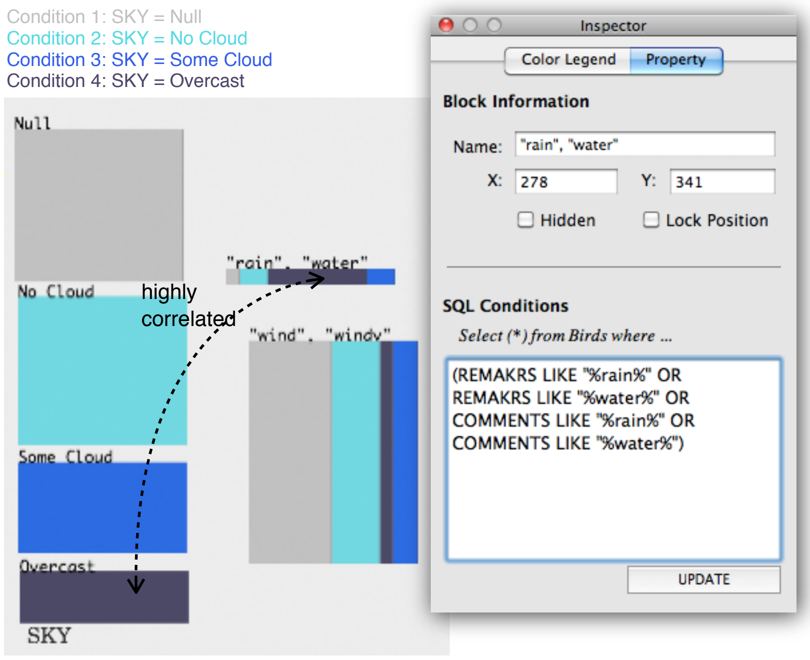
Immediately, the American muscle cars of the 70s are unavoidably exhibited in the display. As *American* cars are colored in orange, we can see that high *Cylinder* numbers, high *Acceleration*, large *HorsePower* as well as low *MPG* are all dominated by orange, stereotypically depicting the muscle cars of that era. On the other hand, if you lack basic, prior knowledge about car specs, it might still lead you to suspect that these speed and the fuel efficiency dimensions are correlated, as they all share the similar coloring characteristics (increasing/decreasing amount of orange).

More interestingly, you can see that the heyday of American Muscle cars ended at 1979: as we hit our first energy crisis, smaller, more fuel efficient European and Japanese cars started to flourish in the market.

The first scenario is based on a highly structured tabular dataset, with clearly defined dimensions (columns) that either contain nominal or numerical values. In our second scenario, we are looking at an aviation safety dataset of airplane incidents/accidents information. There are 11000 documents, with over 80 dimensions, but our primary interest is on two dimensions only: *REMARKS* and *COMMENTS*, which both contain *textual* information filled out by the pilot or ground crew, to describe the incident/accident.

Assume we are interested in incident cases of rainy and windy weather conditions. We created two custom blocks, each based on keyword queries that might describe each condition (*"rain" OR "water"; "wind" OR "windy"*). Result shows that there are way more documents of the "wind", "windy" pair, than the "rain", "water" pair. However, if we also create four coloring conditions, each based on a nominal value in the *SKY* dimension in the dataset (describing sky condition), we can see a much stronger correlation between the *SKY = Overcast* and the documents that contain "rain", "water" keywords. This is of course a trivial finding, as we all know overcast weather leads to rainfall; but the point is that the keywords of *"rain", "water", "wind"...* etc, are not tied to any specific values in the dataset, nor are they tied to a single dimension (both *COMMENTS* and *REMARKS* in this case). This demonstrates the flexibility of our query-driven approach, in terms of supporting unstructured data.

6. Discussion



Block Information

Name: "rain", "water"

X: 278 Y: 341

SQL Conditions

Select (*) from Birds where ...

(REMAKRS LIKE "%rain%" OR
REMAKRS LIKE "%water%" OR
COMMENTS LIKE "%rain%" OR
COMMENTS LIKE "%water%")

6.1 Query-driven Display

As we addressed in the introductions, our study intentionally seeks alternative approaches to display correlation. We believe these differences could result in multiple advantages.

First, our biggest difference from the standard Scatterplot/Parallel Coordinate paradigm is the level of information we are showing. In a typical Scatterplot or Parallel Coordinate, all data entries are individually displayed; and the analyst's task is essentially filtering out the less useful pieces. In our query-driven approach, the minimal visual unit is no longer individual data entries, but query results that are collections of the data entries.

This unit difference provides us improved compactness in the display because in any collection of documents, there are always far fewer values (or ranges of values) that satisfying a query than the actual number of documents. Through setting the minimal visual unit as query, it bypasses the problem of visual occlusion, as there are no longer renderings of individual data entries that can overlap. Therefore, a display setup of hundreds of documents could easily scale to support millions more documents from the same dataset, with no decrease in visual performance. For example, we are showing over 11000 documents in Scenario 2, and it looks just as clean as the examples in Section 3, which have 40 documents.

Second, our approach "associates" correlations differently. In our method, one coloring condition is projected and correlated to the *entire* display; whereas in the Parallel Coordinates, only the correlation in neighboring axes are shown. We speculate that our method might be more in tune with human attention, because it allows the viewer to focus on a single topic at a time, instead of constantly "shuffling" the orders of axes to see the correlation of interest in Parallel Coordinates. However, this speculation still needs to be verified by proper user studies.

Third, the query-driven approach also provides us a different layout philosophy, which we believe to be more flexible. In Scatterplot, Parallel Coordinate and Mosaic Display value entities have to be projected onto dimensional axes to be meaningful; but in Color Block, because each block already self-contains all its information (in its label, height, and coloring), there is a larger freedom in the layout arrangement. The user can spatially group, order and reorder the blocks to assign them different meta-meanings. It might also lead to better usage of spacial memory, however this also needs to be verified by user studies.

Fourth, we believe our query-driven approach can reduce the analyst's cognitive burden of keeping tracks of hypotheses. For example, in the Second Scenario, if an analyst of the aviation data suspects that nighttime might lead to more birds striking to the airplane, then this suspicion/hypothesis is first transformed into a keyword query, describing both nighttime and birds (e.g. "night, dusk, dark, bird, Canada goose"...etc), and finally a color condition. In this case, when interacting with the visualization, the analyst no longer needs to remember the details of the hypothesis (like what keywords are specifically in the query); all he/she needs to care about is the significance of the *color* patterns, as all the hypotheses have transcend into colors. A similar idea also applies to query blocks, as each block can also carry a query, hence a hypothesis. In some extreme cases, users can constantly switch between multiple coloring conditions, with no need to recall what each one represents, until a dominated/salient color pattern merges. In fact, the *Hot Key* function in the application was designed for this purpose, allowing a speedy switch between color legends.

Last, we think the biggest difference of our approach is not in terms of visualization but the *relationship* between the user and the data. In most traditional methods, data is passively display *as is*, so each visualization is essentially a "read-only" display, with certain filtering and reordering capability. In our approach, by displaying data as queries, it provides us more freedom accessing the data, breaking apart the structural boundaries of columns in tabular datasets. For example, In the Second Scenario, we accessed and combined the results from both *REMARKS* and *COMMENTS* dimensions at the same time. This is like treating each document entry as a single text file, so we no longer need to worry about wether a piece of information is numerical, nominal or textual. After all, dimensions/columns are frequently man-made artifacts; when pilots fill out forms to describe their incidents, different people could have completely different understandings of where to fill out what information. In this case, why should we treat data as the reality, when in fact there is already an extra artificial layer on top of it?

6.2 Limitations

As mentioned earlier, we see design as a play of tradeoffs. As this stage of our study, we can already deduce multiple limitations of our visualization from the tradeoffs, with potentially more after proper user studies.

First, because we display data entries as a collection (a block), this is essentially an extra abstraction/compression process, that unavoidably leads to the loss of visual information of individual entries. As a result, many outliers will inevitably be missed in the display, and we believe this is a tradeoff that cannot be compensated.

Second, although we praise our query-driven approach's ability to display cross dimensions, very likely the user would still prefer to work within well-defined dimensions/columns. In this case, for numerical dimensions, range adjustment could be a real a problem. When the user adjusts the range in one block, he/she would have to manually adjust the ranges of neighboring blocks, which can be very tedious. Therefore for utilitarian purposes, we might have to break the "purity" in block-to-block independence, and to link the ranges of neighboring blocks; besides, how to design a interface for range adjustment can also be another challenge.

Third, the display for logic sets addressed in Section 3.3 could be too complicated. By having all the color-stripes in the same direction, our method forces the user to pay extra attention to the ordering of the stripes, or to use mouse-over/mouse-click to obtain further information. While in methods such as Treemaps, all the logic conditions are built in the visual structure, and little interaction is required. But we decided not to adopt the Treemap method because it makes correlation comparison more in the area-to-area manner, instead of the width-to-width manner (of each color stripe) that we believe to be more accurate to human perception. A possibly improvement could be to use better interaction techniques, such as tooltips, to display the logical condition of each block, instead the current status-bar method.

Last, all color-related visualizations have to inevitably face the issue color abusing. When there are more colors than you can distinguish, it would be very difficult to associate one color with its represented query condition. (Interestingly, we also noticed that it is often the *ordering* of colors, instead of color per se, that help us to discriminate underlying query conditions). Our proposed solution is to use a finely tuned color library that can hopefully auto-assign highly identifiable colors to conditions.

7. Implementation

The SQL system in our study is based on SQLite, which has the advantage of being lightweight, fast, and does not require a server. We created a minimal Objective-C wrapper on top of this C library.

The application and visualization are implemented with Objective-C and Apple's Cocoa framework. Cocoa is an excellent framework for creating OSX native applications; we get all the window GUI for free from Cocoa; however, the actual drawings of Color Blocks, as well as all the associated mouse and event handling have to be manually created and linked with the Cocoa framework (where most of the bugs were).

8. Lessons Learned & What I Would Have Done

This study initiated as a reaction against the messy lines/edges in the Parallel Coordinates. My original goal was to improve the edge rendering/filtering used in that method, but many hours of fruitless sketching forced to me seek otherwise. From this, it reinforced my learning that before solving a problem, it is necessary to see where the problem comes from, and to see if we can *bypass* it. Inspired by Cleveland's paper on visual features, I tried to integrate new dimensions (color) into the visualization. The result definitely bypassed many previous issues faced in the Scatterplot/Parallel Coordinates paradigm.

The query-driven idea wasn't intentional either. Initially I used the SQL backend only to store files, with a very thick layer of wrapper class that nicely formats the data, before hand them into the rendering classes. But then I got inspired by the contrast in data handling philosophy, between two commercial visualization tools. One is Starlight [3] , which has absolutely zero customize-ability (all data is real-only); in contrast, Tableau [4] allows you to reformat the data, and to perform some simple queries on the fly, even after the data have been imported. I found myself much more productive with Tableau, as I could offload a lot of my thinking back to the tool, (which then could be used as new data, resulting in a healthy analysis "loop"). This project, especially this paper, crystalized many of my thinkings on this issue.

If there were more time, I would definitely come up with better scenario examples for this paper. The two scenarios in Section 5 only demonstrates the basic functionality of this visualization, not its potential power.

Also, I wish there was time to do a more extensive literature review, especially to relate to the "query-driven visualization" idea. In this study, we have almost no idea if the approach has been addressed before or not, therefore we cannot declare the novelty of this approach (we suspect it must have been done before).

Moreover, real world testing and user studies are a must. Too often we are "designing" solutions to non-existed problems. This is especially a pitfall for a general technique such as this one (instead of improving on preexisted, proven ones). I wish there was more time to actually drive the visualization in real world analysis environment (which I will in my coming up internship at Boeing, for aviation data analysis), and also let other people try it out (which I would probably release it as an Open-source application).

9. Conclusion

In this study we designed a general visualization technique called *Color Block*, which is capable of displaying correlation in multidimensional structured data, as well as unstructured data such as text. We argued that its underlying approach, which is to display data queries instead of data entries, could be more advantages than the Scatterplot/Parallel Coordinates' approach in certain circumstances. However, because we did not perform any user studies, the full potential as well as many problems of the *Color Block* technique is still unclear; but we think there won't be many problems because the design is simple and flexible.

10. Bibliography

[1] Wegman, EJ. (1990). Hyperdimensional data analysis using parallel coordinates. Journal of the American Statistical Association
[2] Friendly, M. (1994). Mosaic displays for multi-way contingency tables. Journal of the American Statistical Association
[3] Starlight Information Visualization System. http://starlight.pnl.gov/
[4] Tableau Software. http://www.tableausoftware.com/