Color Palette Generation for Nominal Encodings

Calvin Ardi
Computer Science
University of California, Berkeley
Berkeley, CA
+1 510 642 8679
calvin@rescomp.berkeley.edu

Simon Tan
Computer Science
University of California, Berkeley
Berkeley, CA
+1 510 915 7789
simtan@eecs.berkeley.edu

Ketrina Yim
Computer Science
University of California, Berkeley
Berkeley, CA
+1 925 922 9152
kyim@berkeley.edu

Abstract

Encoding data using color is a key technique employed by almost all visualizations. A set of colors used to represent data values must be carefully chosen in order to maximize the effectiveness of the visualization in which it is used. We present a method of automatically generating effective color palettes for nominal data encoding based on user specifications. We apply simulated annealing with an appropriate heuristic function to find palettes that have visually distinguishable colors and adhere closely to a user's color preferences. Such palettes can be customized to feature color harmony, a particular variation of contrast or saturation, or support for the colorblind.

Keywords: color, nominal encoding, simulated annealing, color use guidelines, colorblindness

1 Introduction

The use of color provides a number of benefits to visualizations. Color is often used in visualizations to encode data, be it the magnitude of a quantitative value or the category in which a particular data point falls. If used effectively, a color encoding can improve legibility, facilitate data layering, create groupings, and highlight values of interest. Conversely, poor color design can devalue the information presented, create distracting clutter, and lead to false conclusions about the data. These consequences often make selecting a color palette a difficult and time-consuming task, particularly for those inexperienced in visualization design, because there are many factors to consider while choosing colors. Among the most important of these factors is color separability. One must ensure that each color in a palette is visually distinguishable from the others, a property that becomes difficult to maintain as the number of values to encode increases. Other factors are value and contrast control. While it is important for data values or groupings to be differentiable, data points that unintentionally stand out or become subdued can misdirect the audience's attention. In addition, the desire for aesthetics and compatibility with colorblind viewers can also affect color choices.

The complexities of palette selection have led to two main approaches to palette design interfaces in visualization software. One is to present users with a set of predefined color palettes. Expert designers usually create these palettes, so most of the color issues are addressed. However, presets offer little opportunity for users to customize the appearance of their visualizations. The other approach sits at the opposite end of the spectrum, offering full control of the palette to the user. This option offers the most freedom, but places the burden of palette design completely on the user. As mentioned earlier, this can be overwhelming to novices. Thus, there is a need for a compromise between these two interfaces.

The compromise created by our method resides in the fact that the user can customize the palette through a set of preferences, but the palette itself is generated by solving the optimization problem of finding a palette that maximizes distinguishability, limits value and contrast variation, and closely adheres to the user's guidelines. Our system is built on Flare, a Flex adaptation of the Prefuse visualization toolkit [Heer et al. 2005].

2 Related Work

Color palette generation is a topic of interest for both visualization researchers and artists alike. As such, a significant amount of work has been done in the field of palette design.

2.1 Quantitative Palette Generators

The majority of research has gone into palettes for quantitative data, where color often has more responsibility than to just differentiate data points and groupings. Task-based approaches [Tominski et al. 2008] produce palettes according to the purpose that the palette must serve in visualizing the data (*i.e.,* simple lookup, comparison, or localization). Commercial efforts also exist, as seen in IBM Research's PRAVDAColor [Bergman et al. 1995]. This tool generates palettes that preserve the spatial structure of data, facilitate specific analysis tasks, and considers concurrent color use to reduce color-mixing artifacts. Though quite robust, palette generation with PRAVDAColor takes a significant amount of time due to the numerous user inputs required.

2.2 Tools for Nominal Palettes

Palettes for quantitative data are not necessarily applicable to qualitative data. Therefore, separate tools have been created for palettes encoding nominal data. Brewer's ColorBrewer [Brewer and Harrower 2002] presents palettes for encoding categorical data in maps. Though limited to a maximum of twelve colors in a palette, it offers information on a palette's compatibility with colorblind viewers with a variety of displays and each color's value in a range of color space coordinates. However, ColorBrewer does not automatically generate palettes; rather, it presents a series of appropriate presets designed by Brewer herself through years of experience designing maps. As such, ColorBrewer yields effective palettes but restricts customizability like any other palette chooser that presents presets.

2.3 Other Palette Generators

On the Internet, applications exist to generate palettes with a significant amount of customizability. One example is the Color Scheme

Chooser [WebsiteTips.com 2008], an application that produces a palette given a base color and color scheme options. Some applications can even produce palettes based on an uploaded image [Adobe 2008]. In general, these generators produce palettes that are high in aesthetic value but are not meant for data encodings. The interfaces of such tools, however, provided a significant amount of inspiration for our generator's user interface.

3 Methods and Implementation

3.1 User Options

Since our solution provides a way for users to input preferences for influencing the palette generation, it is necessary to describe what each of these preferences are before delving into how our system utilizes them.

3.1.1 Size

This is the number of colors desired in the resulting palette. We can theoretically support any number of colors in a palette, but we find that very large palettes would inevitably contain colors that would be difficult to differentiate from each other. We allow anywhere from 3 to 50 colors per palette in our implementation, but resulting palettes of 20 colors or more are usually undesirable. The default setting is 10 colors.

3.1.2 Harmony

The concept of color harmony refers to palettes that have "thematic" colors. For example, a "warm" palette would primarily have colors from the red-orange-yellow part of the hue spectrum. In our implementation, we allow the user to specify these types of color harmony in the resulting palettes:

Warm colors palettes that have a majority of their colors in the red-yellow/green hue range

Cool colors palettes that have a majority of their colors in the green-violet hue range

Specific color palettes that tend to have their hues near the hue of a specified color

3.1.3 Variations

Color palette variations are other types of color schemes that might be desirable for a visualization designer using nominal color encodings. We allow the user to specify one of the following options for color variation:

Pastel colors palettes with colors that have low saturation values

Bright colors palettes with colors that have high saturation values

High contrast colors palettes with colors that have high contrast with each other and a specified background color

Low contrast colors palettes with colors that have lower contrast levels with each other and a specified background color

3.1.4 Colorblindness

One of the more difficult tasks for a palette designer is in choosing colors that would not be confused by a person with colorblindness. We offer the ability to generate palettes that avoids sets of colors that would look too similar to a colorblind person:

Red-Green colorblindness Avoids having *both* red and green hues appear in the palette

Blue-Yellow colorblindness Avoids having *both* blue and yellow hues appear in the palette

Together, these two options allow for the creation of palettes that have colors fully distinguishable from each other for the vast majority of those who have some sort of colorblindness. ($\sim$1.3% of the people in the United States [WrongDiagnosis.com 2008])

3.1.5 Background Color

The user can specify the hex-value of the color used as a background for the visualization with this option. This is needed to prevent the generation of palettes with colors too similar to the background (one of Brewer's Color Use Guidelines [Brewer 1999]). This option is white (`0xFFFFFF`) by default.

3.2 Simulated Annealing

Our use of simulated annealing to approach this problem was born out of a desire for an algorithm with these traits:

- The opportunity to use a heuristic function to judge the quality of color palettes
- The ability to traverse the possible space of color palettes iteratively, in order to use our heuristic in such a way as to determine the "best" palettes of that space.

Our implementation of simulated annealing is not the pure form of annealing that is described in most artificial intelligence literature. We have made modifications to make it more suitable for our purpose. An outline of our algorithm follows in Algorithm 1.

Algorithm 1 Simulated Annealing Algorithm

$currentPalette \leftarrow$ default palette
$currentScore \leftarrow \text{heuristic}(currentPalette)$
for $i = 1$ to $numRounds$ **do**
  $newPalette \leftarrow \text{perturb}(currentPalette)$
  $newScore \leftarrow \text{heuristic}(newPalette)$
  if $newScore > currentScore$ or $\text{rand}(0,1) < \exp^{-\frac{\Delta \text{score}}{T}}$ **then**
    $currentPalette \leftarrow newPalette$
    $currentScore \leftarrow newScore$
  end if
  $i \leftarrow i + 1$
end for

3.2.1 Starting Palette

Depending on the user's options for desired "color harmony" (none, warm, cool, or a revolving around a specific color), we started the algorithm with different initial palettes. For warm and cool options, we start with palettes generated from the Flare library's `ColorPalette.hot()` and `ColorPalette.cool()` functions respectively. If the user wants a palette that harmonizes around a specific color, we start with a palette composed only of that particular color. Another option would be to start with a diverging palette centered on that particular color, which we did not explore.

If the user has no desired color harmony for the palette, we start with the nominal encodings given by the Flare library `ColorPalette.category()` function. Although this means that the process begins with a quality, hand-chosen color palette,

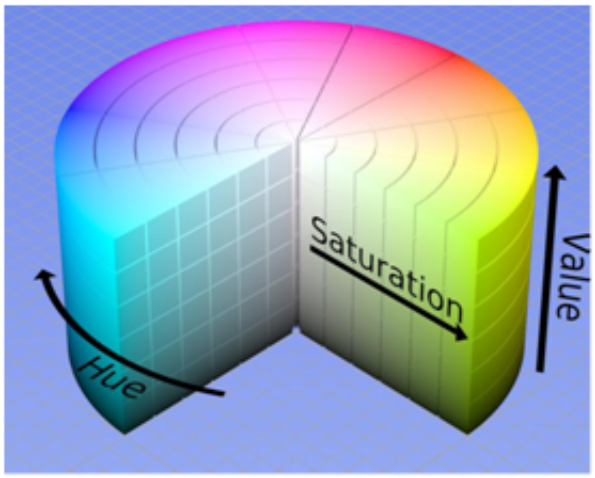

Figure 1: *Adjustments of Hue, Saturation, and Value in a 3-dimensional representation of the color space. [SharkD 2008]*

we hope that the random properties of the algorithm result in movement away from the Flare palette to other unique palettes, perhaps just as high or higher in score.

3.2.2 Perturbations

At each iteration, we perturb the current palette by altering a random feature of a random color within it. In order to better control the effects we have on the colors in the dimensions we care about within the heuristic, we perturb by randomly incrementing or decrementing a color's hue, saturation, or value at each step. The Hue-Saturation-Value (HSV) color space is shown in Figure 1.

The amount we affect these values by at each iteration is constant (0.05, where hue, saturation, and value $\in [0, 1]$). However, we divide this factor in half for Hue modifications when we detect that the user has chosen a color harmony preference, since this implies that colors should not stray much from a defined range of hues.

By affecting palettes in this way, we essentially have the ability to efficiently traverse the entire three-dimensional space of color for each element of the palette. With an infinite amount of rounds, we can theoretically reach every possible combination of colors for palettes of any size.

3.2.3 Iterations

We calculate a score for every palette we generate through our perturbations using our modular heuristic function (see Heuristics), and decide whether to accept or reject each new palette by comparing it with the previous palette.

Normally, we accept a new palette if it scores better than the previous one. However, to prevent our algorithm from inflexibility at a local maximum, we accept palettes that are worse with a small probability that goes down as more iterations are completed. The probability that a worse palette is accepted is $e^{-\frac{\Delta \text{scores}}{T}}$, where T is the number of rounds of the algorithm left to run. Hence, as T approaches zero, our algorithm gradually reduces to the pure greedy algorithm which will only accept palettes if they are better.

3.2.4 Stopping Condition

We did not consider a more intelligent stopping condition for our algorithm; rather, we decided to have a set number of rounds of simulated annealing and adjusted that to study the impact of our annealing on the resulting palettes. We generally ran about 10,000 rounds, but could feasibly run 100,000 rounds before the runtime environment became unstable (see Performance).

3.3 Heuristics

In order to evaluate a generated palette, a set of heuristics is employed in the form of a general scoring function. The scoring function returns a linear combination of various weights multiplied with scores returned from different modules that evaluate certain aspects of each color within the palette, or the palette itself. All of the modules evaluate the palette based on the HSV color model, which allows for easier identification of the colors: a specific hue value defines a color, while a color represented in the RGB additive model is a combination of the three primary colors.

A default set of "good" values are created. If none of the user options are selected, the scoring function will evaluate the palette based on values previously prescribed (Figure 2). Options selected otherwise will modify certain values and ranges based on the preference (e.g., if the cool "color' harmony' option is selected, we explicitly define the range in hue values that are acceptable).

All modules in the heuristic return normalized scores with range $[-1, 1]$. This allows for greatest flexibility when determining weights, as the individual can tailor the scoring function based on his or her preferences. This also opens up future work regarding how the weights are calculated.

Aspect	Values
Contrast	$[0.2 - 0.8]$
Hue	$[0^\circ - 360^\circ]$
Saturation	$[0.5 - 1.0]$
Value	$[0.5 - 1.0]$

Figure 2: *Prescribed default values*

3.3.1 Pre-Generation Modules

Before the scoring function is run, these modules make the appropriate modifications to acceptable HSV values based on what palette preference or color harmony the user selects. Depending on the configuration or preference that a user has, the preferred values for each characteristic of a color is redefined and updated if the user changes input values.

In particular, "harmony" choices change optimal hue values, while choices in "variations" change contrast, saturation, or value ranges (Figure 3). Pastel colors, for example, require a saturation range between 0.30 and 0.65; in all cases, as color is highly subjective, values were chosen based on group consensus. Unless otherwise noted, any ranges not modified were left to the default values listed in Figure 2.

3.3.2 HSV Evaluation Modules

Several modules in the heuristic are dedicated to the validation of various HSV values; although implemented in separate modules in the Flare library, their discussion and implementation can be summarized together under the same heading. In general, these modules iterate through the palette, checking to make sure each aspect of the

t **Palette Preference**	**Aspect**	**Values**
Warm	Hue	$[0\,^\circ - 63\,^\circ, 330\,^\circ - 360\,^\circ]$
Cool	Hue	$[90\,^\circ - 300\,^\circ]$
Specific base color	Hue	$[hue - 10\,^\circ, hue + 10\,^\circ]$
High contrast	Contrast	$[0.6 - 1.0]$
Low contrast	Contrast	$[0.0 - 0.5]$
Pastel colors	Saturation	$[0.30 - 0.65]$
Bright colors	Value	$[0.85 - 1.0]$

Figure 3: *Preferences and Corresponding Ranges*

colors fit within a specified threshold (Figure 2, 3) and score the palette appropriately.

Brewer [Brewer 1999] also notes that most people prefer blue colors to yellow; this has been implemented, but weighted lower relative to the other heuristic modules as it would not be ideal to have a palette that consisted of only blue hues.

Finally, our palette should have "spacing" between attributes and values of each color. For example, it will not be an optimal palette if there are two colors with similar hues. Likewise, palettes containing colors with different hues yet low values of saturation and value would not be optimal either. All the colors would have muted hues and be almost indistinguishable from each other. The "neighboring values" module penalizes palettes that have colors whose values are too similar with one another.

3.3.3 Saturation/Lightness Variation Modules

In addition to variations in hue, Brewer [Brewer 1999] noted that qualitative schemes benefit from small variations in lightness (value) and saturation. The standard deviation of the lightness and saturation is taken from all the colors in the palette and subtracted by an optimal saturation and lightness deviation value, 0.3 and 0.086, respectively. The resulting score is the difference between the absolute difference between the two standard deviations and 1.0.

3.3.4 Colorblindness Module

Palettes can be created and tailored to colorblind audiences. In each major type of colorblindness (red-green and blue-yellow), we not only check to make sure that certain hues are avoided, but also that they must fit within a specific saturation and value threshold in order for a "penalty" to be assessed. By navigating the HSV color space, one can determine that most colors with saturation or value less than 0.5 tend to have their hues indistinguishable. A color with saturation and value equal to 0.0 will be black, regardless of hue.

3.3.5 Contrast Modules

Colors in palettes, in general, should have an appropriate amount of contrast between each other, and with the background. First, the relative luminance, defined by equation (1) [Stokes et al. 1996], of each color is calculated. We then calculate the Michelson contrast, equation (2) [Michelson 1927], found by dividing the sum of the two luminance values by the absolute value of their difference.

$$\text{luminance} = 0.2126R + 0.7152G + 0.07522B \tag{1}$$

$$\text{contrast} = \frac{I_{\max} - I_{\min}}{I_{\max} - I_{\min}} \tag{2}$$

The Michelson contrast is used when comparing the contrast of a color to the background color due to potential divide-by-zero errors associated with the Weber contrast (3).

$$\text{contrast} = \frac{I - I_{\text{background}}}{I_{\text{background}}} \tag{3}$$

Module	**Weight** (α).
hue	10.0
blue/yellow	1.0
colorblindness	30.0
contrast	1.0
background	1.0
neighbor	15.5
saturation threshold	5.5
saturation variation	1.5
lightness variation	1.5
language	1.0

Figure 4: *Weights and Modules*

3.3.6 Language Module

Brewer [Brewer 1999] notes "[in] addition to red, green, yellow, and blue, the other basic colors named in all fully-developed languages are pink, purple, orange, brown, gray, white, and black." A module was developed that encompasses these ranges of colors and positively affects the heuristic score if these hues exist within the palette. To avoid excessive influence on the palette results, however, the weight associated with this module is significantly lower than the weights of other modules.

3.3.7 Scoring Function

Given the modules, we can then generalize the scoring function in equation (4), where N is the number of modules that exist, *module* refers to a heuristic module that evaluates and scores a palette for a certain quality (normalized to the range $[-1, 1]$) and α is the associated weight given to that particular module.

$$\text{heuristic score} = \sum_{i=1}^{N} \alpha_i \times \text{module}_i \tag{4}$$

The list of weights and their corresponding modules used can be found in Table 4.

3.4 Extension to Flare

Because Flare is an open-source visualization toolkit, we were able to implement our palette generator as a Flex library based on Flare, in hopes of presenting it as an extension package. The generator consists of three classes:

NominalColorEncodingGenerator.as This class contains the simulated annealing algorithm used to generate palettes iteratively. It also contains the perturbation function used to traverse the color space.

NCEGHeuristic.as This Strategy class contains the heuristic function used to evaluate each intermediate palette during the annealing process.

ColorOptions.as This class is the type of the object used to pass user-specified options to the palette generator. It also contains the static variables used to refer to each parameter value by name.

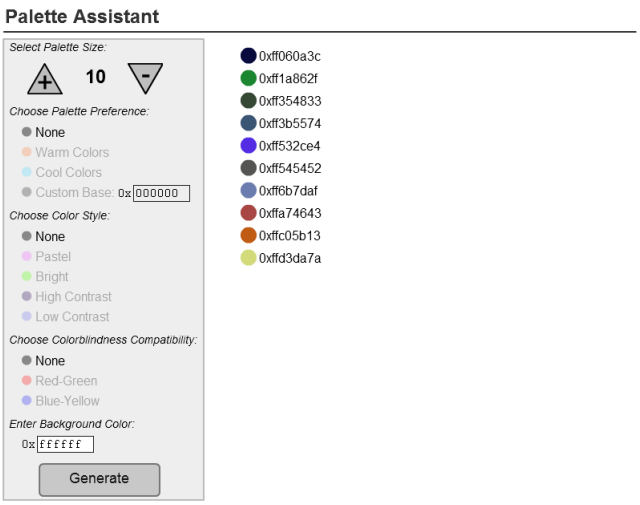

Figure 5: *A screenshot of the graphical frontend*

3.4.1 Graphical User Interface Wrapper

To provide a visual demonstration of how our palette generator works, we implemented a graphical user interface to our Flex library. It displays the available palette options as a control panel and the results of running the algorithm with these options on the right. The resulting palettes are represented as a list consisting of each color and its corresponding 32-bit hexadecimal value (with the first two digits being the color's alpha, which is always `0xFF`). The graphical user interface is shown in Figure 5.

The control panel consists of six components. From top to bottom, they are:

Palette size input adjusted by clicking the "+" and "−" buttons on either side of the displayed number

Color harmony input when the "Custom Base" option is selected here, a text box to enter the custom base color as a 24-bit hexadecimal value becomes active

Color style input

Colorblindness compatibility input

Background color input affects both the heuristic and the background that the generated palette is displayed against

"Generate" button when pressed, this reads in the user input specified above it and runs the generator with those options

4 Results

4.1 Sample Palettes

Figure 6 presents a set of screenshots of palettes generated through the graphical interface.

4.2 In Context with Data

In her guidelines, Brewer explains that the appearance of a color is often affected by its context [Brewer 1999]. Therefore, a true measure of a palette's effectiveness can only be obtained by actually mapping the colors to some data and examining the resulting visualization. We present our generated palettes in context with data

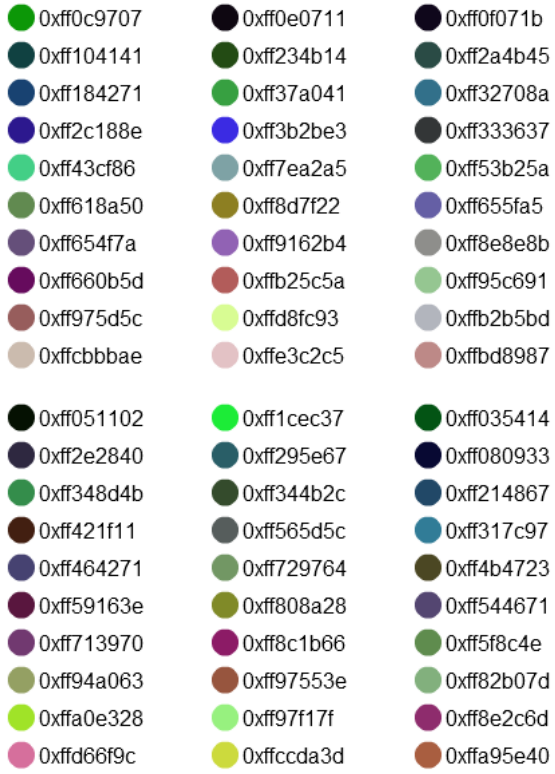

Figure 6: *Samples of generated 10 color palettes*

by using them to encode the nominal variable of decay mode in an interactive table of isotopes [Yim 2008] implemented in Flex/Flare. Screenshots of the resulting visualizations are shown in Figures 8 and 9.

4.3 Performance

Performance requirements vary depending on the usage scenario. That is, the number of rounds may require adjustment based on the application. As stated earlier, the color palette generator is a Flex library which outputs a color palette that can be used immediately for visualization. An alternative is to use the included graphical frontend to view the generated colors and export them for use in another application or visualization tool. Thus, if being used with visualizations designed in Flare, the number of rounds should be decreased such that there is no excessive delay in generating a palette. Likewise, a user selecting colors from the interface can afford to take the time to increase the number of rounds and implement the colors into his or her visualization manually.

Color generation was done with the default settings: a palette size of ten with no additional options enabled. Debug statements were suppressed and the user interface was run using Adobe Flash 9 and Mozilla Firefox 3.0.4 on Apple's Mac OS X on an Intel Core 2 Duo 2.4 GHz system. Running 1,000 rounds resulted in times varying from 690ms to 774ms, with a linear correlation between time and rounds (Figure 7). Running at 40,000 rounds resulted in run times around 28,000ms. We found that running anything greater than 100,000 rounds resulted in a timeout built into Adobe Flex; regardless, the amount of time required running that many rounds would be highly impractical for Flare visualizations. If this many rounds were needed, the generator should be implemented in a more

efficient language for more optimal run times.

Rounds	**Times** (*ms*)
1,000	$690 - 774$
10,000	$7101 - 7261$
20,000	$13976 - 14368$
40,000	$28427 - 28989$

Figure 7: *Generation running times*

Part of the performance issue is due to the way we implemented the heuristic. The heuristic is designed to remain flexible and allow quick modification or removal of each of the individual scoring modules. A more optimized version could potentially be implemented by combining several of the modules with related calculations into a more efficient larger module.

5 Discussion

Currently, there are two main issues with our generated palettes. First, palettes sometimes have colors that are visually difficult to distinguish, even though the colors have significantly different hexadecimal values. Our heuristic does take into consideration the pairwise contrast of colors in a palette, so it is likely that these undesirable subsets emerge when the simulated annealing is trapped at a local maximum. This can occur despite the fact that the simulated annealing algorithm has provisions to continue exploring the color space, even when local maxima are hit. Another issue is that since aesthetic value is difficult to encode as a numeric feature, and thus was not included as part of our heuristic, a high heuristic score does not necessarily mean the palette is aesthetically pleasing. A user may have to go through several rounds of palette generation with the same parameters before an attractive palette is obtained.

From our work, it can be realized that evaluating a palette's overall "quality" with a score is nontrivial. Expert designers who create color palettes for visualizations consider many factors beyond what we have encoded in our heuristic function. For designers who require good color schemes without the time expense associated with handpicking colors for the most optimal palette, we believe our color palette generation system can be used for its efficiency and effectiveness.

6 Future Work

We have several ideas on how to build upon the work that has been presented here.

6.1 Heuristic Modules and Weights

Potential work could be done with regards to adjustments to the weights of the scoring function. Weights and values are subjectively chosen and attempt to optimize palettes that cater to a variety of audiences and users. Using machine learning or supervised learning, weights and other values can be learned and automatically adjusted based on user preference. In particular, one could train the system to adapt to a user's preferences by showing two different palette choices multiple times, then adjusting and saving the weights based on their choices. For future use, especially if implemented using the Flare visualization library, these weights can be passed in as parameters so the system does not have to be continuously retrained.

In addition, the library has been developed such that new modules can be added with ease. Modules may be created in response to new work in color representation, newfound color guidelines, or the changing preferences of a target audience.

6.2 Aesthetics and "Frozen Colors"

Often, palette designers are highly selective about the colors they wish to use. Our system may try its best to follow color use guidelines, but it currently cannot consider the aesthetics of palettes it creates. One could imagine drawing from a source of color aesthetics guidelines to complement our heuristic. However, it should be noted that aesthetics are very subjective and even this approach would not be guaranteed to produce palettes that are aesthetically pleasing to all people.

Alternatively, our graphical frontend to the system could provide a way for users to specify which colors they definitely want in their palette and which they do not care for. This could be done by allowing users to "freeze" colors after a palette is generated and then run the generation again with those colors protected from perturbation.

6.3 Improved Termination Condition

Currently, our simulated annealing terminates at a set number of rounds. A better solution would be to stop when a certain score threshold is crossed, when perturbations are detected to do more harm than good to the palette's score, or when perturbations have little effect on the score.

Another approach would be to determine an appropriate minimum number of rounds of simulated annealing to run. This could be accomplished by running many trials where scores are recorded at each round. These records could then be examined to figure out where scores typically converge, leading to an ideal baseline of iterations.

6.4 Using Data

We note that some of Brewer's Color Use Guidelines [Brewer 1999] require a palette to be judged in context with the data that the colors are going to be encoding. For example, Brewer suggests using saturation to emphasize smaller categories as they are naturally hard to see on a graphic.

Currently, our algorithm produces palettes with no consideration of the data they will be used to encode. In addition, the colors are displayed in the style of a legend, which Brewer warns is a poor view for palette evaluation. These issues can be addressed with one of two methods.

One would be to do as Brewer herself does and generate fake data to visualize with the palettes. This would involve dynamically generating fake data for each palette generation or loading a prepared set of fake data as part of the application. However, this does not factor into the algorithm and may require users to run the algorithm many times over to find a palette that works with a data set that they would not even use. Alternatively, real data sets could be passed into the algorithm to be analyzed as part of a heuristic module. This would require much overhead in data processing, which may not be desired in a package such as ours. However, it does open the doors to many possibilities for data analysis to play a role in determining which colors should be placed near each other on a graphic, which our algorithm currently cannot consider.

7 Conclusion

We have presented a method for algorithmically generating original, high-quality color palettes for use in nominal (categorical) encodings. We have applied the technique of simulated annealing to traverse the space of color palettes iteratively, using a heuristic to

score palettes and work our way towards palettes that follow sound color use guidelines.

Our novel method for iterating through the space of color palettes is a modified version of the simulated annealing algorithm that is affected by the palette options it is initialized with. While based on a randomized search, our iterative process begins with palettes that are not actually random; rather, they are tailored to the user's preferences for color harmony. In addition, the amounts by which we perturb color values are dependent on this preference as well.

We have also introduced a modular heuristic function that is able to score palettes based on many sound color use guidelines. This heuristic could possibly be used in other applications to judge the quality of a color palette used for visualizations.

This work is presented as an extension to the Prefuse/Flare visualization library by Jeffery Heer, and we hope that it can be useful to anyone using the library to create visualizations requiring color as an encoding for nominal variables.

Acknowledgements

We thank Professor Maneesh Agrawala for allowing us to produce this system as part of a graduate course on visualization. We also thank Jeffery Heer for his Prefuse/Flare toolkit, on which this system and graphical frontend is built.

References

ADOBE, 2008. Kuler. http://kuler.adobe.com.

BERGMAN, L. D., ROGOWITZ, B. E., AND TREINISH, L. 1995. A rule-based tool for assisting colormap selection. In *IEEE Visualization*, 118–125.

BREWER, C. A., AND HARROWER, M. A., 2002. Colorbrewer. http://www.colorbrewer.org.

BREWER, C. 1999. Color use guidelines for data representation. In *Proceedings of the Section on Statistical Graphics*, American Statistical Association, 55–60.

HEER, J., CARD, K., S., LANDAY, AND A., J. 2005. prefuse: a toolkit for interactive information visualization. In *Proceedings of ACM CHI 2005 Conference on Human Factors in Computing Systems*, vol. 1 of *Interactive information visualization*, 421–430.

MICHELSON, A. 1927. *Studies in Optics*. U. of Chicago Press.

SHARKD, 2008. Comparison of the hsl and hsv color spaces when mapped to a cylinder, with corner cut-away shown.

STOKES, M., ANDERSON, M., CHANDRASEKA, S., , AND MOTTA, R., 1996. A standard default color space for the internet - srgb. http://www.w3.org/Graphics/Color/sRGB.

TOMINSKI, C., FUCHS, G., AND SCHUMANN, H. 2008. Task-driven color coding. In *IV*, IEEE Computer Society, 373–380.

WEBSITETIPS.COM, 2008. Color scheme chooser. http://websitetips.com/colortools/sitepro.

WRONGDIAGNOSIS.COM, 2008. Prevalence and incidence of color blindness. http://www.wrongdiganosis.com/c/color_blindness/prevalence.htm.

YIM, K., 2008. Interactive table of nuclides. http://vis.berkeley.edu/courses/cs294-10-fa08/wiki/index.php/A3-KetrinaYim.

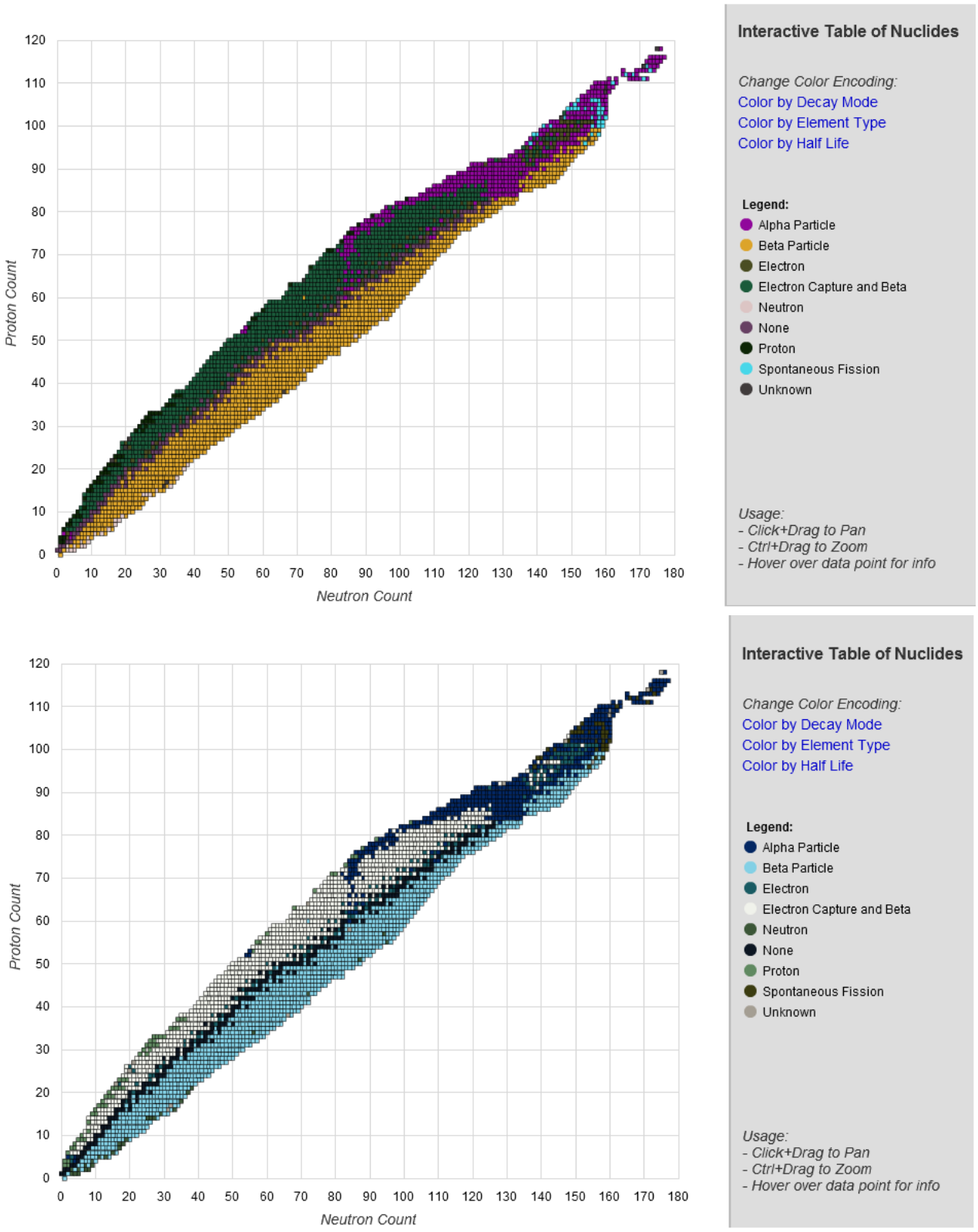

Figure 8: *Samples of generated palettes applied to a table of nuclides implemented in Flare.*

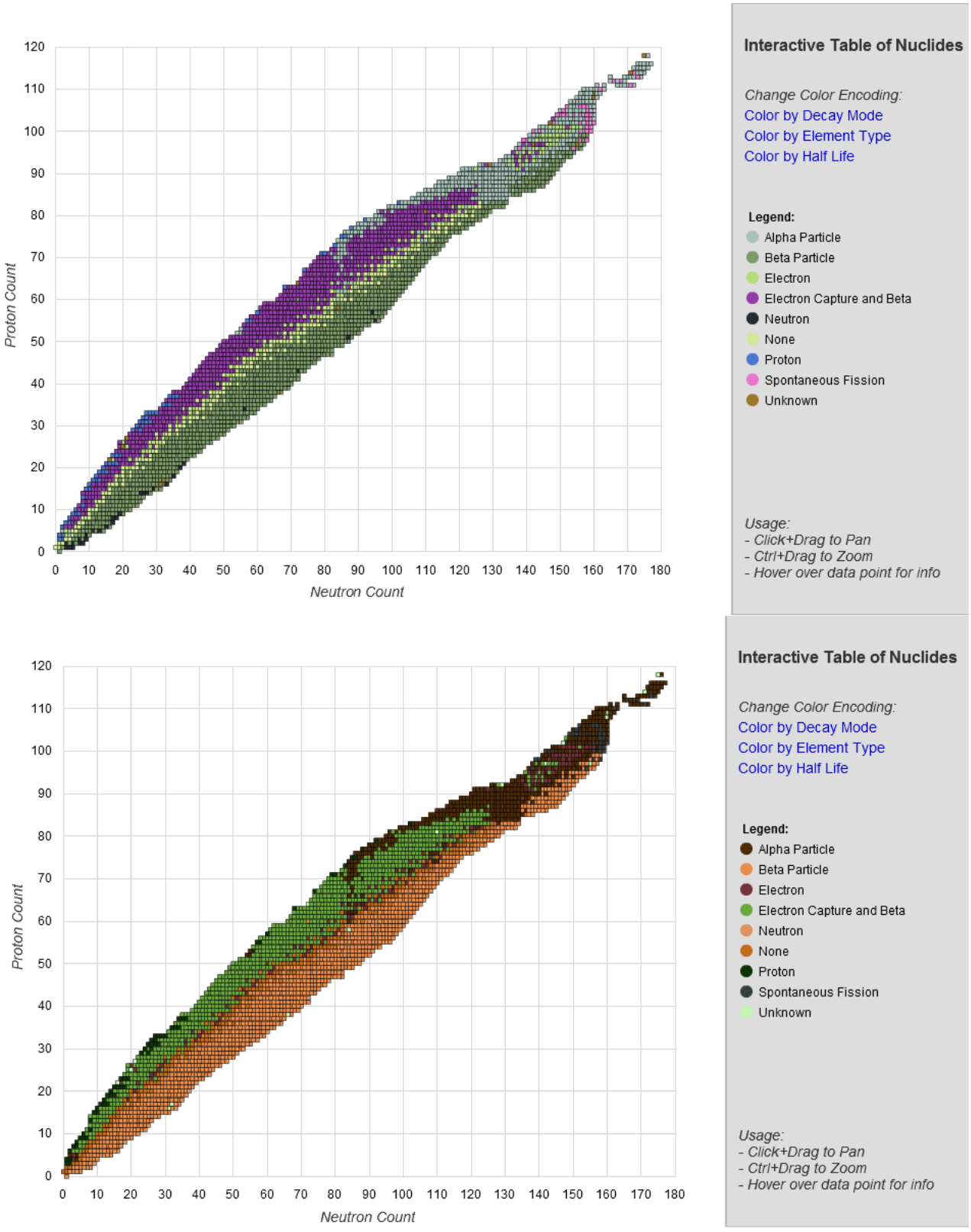

Figure 9: *Additional samples of generated palettes applied to the table of nuclides.*