

One Dimensional Color Sorting and Genetic Programming

Abstract

Color sorting is a vast and deep topic. Sorting colors in three dimensions already has various solutions which take the form of color formats that describe colors through a set of properties. RGB, HSL, and HSV are three such formats which sort colors in three dimensions. Sorting colors in two dimensions, while more difficult than sorting in three dimensions, is also widely considered a solved issue. Color palettes and color wheels are examples of two dimensional sorts. Sorting colors in one dimension, however, is an extremely difficult task, and no one attempted solution to this problem is considered to be the correct way of solving the issue. In this paper, a machine learning approach is proposed, utilizing an evolutionary algorithm to evolve an evaluator function which can score colors to be sorted appropriately in one dimension.

Introduction

Sorting colors is a common and useful practice in many fields including art, advertising, and design. It's applications include everything from aesthetics to practical visual clarity. In most situations, color sorting can be done using traditional three dimensional sorting methods. Digital

to creating visually appealing or visually clear designs in three dimensions. Sorting colors in two dimensions has also had plenty of work done, and many methods have been devised to create appealing color maps in this space. One-dimensional color sorts, on the other hand, are not so trivial. Sorting colors in one dimension is deceptively difficult, as the components of a color which make it appear visually appealing next to another color are plenty and non-obvious. Additionally, many methods which have been devised for sorting colors in one dimension are computationally intensive. In this paper a handful of attempts at sorting colors in one dimension are explored, as well as the possibility of how machine learning may help in this pursuit.

Background and Related Works

Background

Many formats have been created for describing colors in such a way that they may be digitally stored and easily referred to. The most prevalent of which is RGB. In the RGB format, a color is described through three additive components: red, blue, and green. These components are given a value between 0 and 255. The values add to one another to create the desired color.



HSL and HSV are two additional commonly used color formats. HSL stands for ‘Hue, Saturation, Lightness,’ and HSV stands for ‘Hue, Saturation, Value.’ Both of these formats have as their first component a hue, which is a directional value between 0 and 360 degrees. The hue is a value which generally determines what a color is ‘known’ as (red, yellow, etc). Both of these formats also have a saturation value, which describes the intensity of the hue. The saturation in HSL and HSV are calculated differently from one another. The third component of both, ‘lightness’ and ‘value’ are separate ways of describing how ‘light’ a color is. Figure 1 shows how these different formats look, visually.

Another commonly used property of digital colors is called ‘luminosity.’ Luminosity describes how bright a color appears to the human eye. It can easily be calculated from the RGB values of a color. In this paper, all of the above properties are utilized in the experiments.

Related Works

In his article [1], Alan Zucconi attempting to solve the problem of one dimensional color sorting through several analytical methods. After all, colors in RGB, HSL, and HSV format are inherently numerical, so several potential methods for color sorting can be contrived by attempting to exploit this. His first set of attempts were to sort directly by each value of the color space, giving priority to the first value, and so on.

Unsorted:



These methods, while they sound nice, do not give very visually appealing results. The RGB sorting looks almost as bad as the unsorted list. The HSL and HSV sorts look a little better, but still have obvious banding where the lightness and luminosity differ between adjacent colors. The article explores several other potential methods, some of which are not entirely without merit. However, the more complex sorting methods can be computationally expensive and still not produce the desired results.

The aforementioned attempts at sorting were all manually devised. Machine learning uses a different approach, and can often find solutions to problems which were never considered. The author presents in this paper an attempt to use machine learning to devise a sorting method which is computationally cheap and pleasing to look at.

Methodology

Overview

In an attempt to sort colors in one dimension, the author used the machine learning strategy of an evolutionary algorithm with genetic programming. In order to complete the sort, a twofold process is used (see Figure 2). First, a color evaluation function takes in various components of each color and assigns the color a floating point value. Then, the colors are sorted numerically by their corresponding floating point evaluation. The key to this process is the color evaluation function, which is evolved over the course of the evolutionary algorithm.

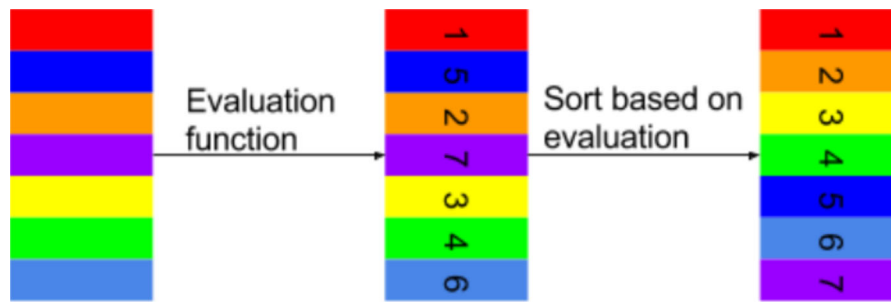


Figure 2

Genetic Programming

The method by which the color evaluation function is selected is genetic evolution of tree structures. These tree structures are made up of root nodes which represent numerical operations, and leaf nodes, which represent both constant values and numerical properties of the color to be evaluated. When the tree is used to evaluate a color, the leaf nodes take the corresponding values of the color being evaluated, and then the values are propagated upward via the unary or binary operations. The final value that is propagated from the root of the tree is the floating point evaluation of the color. Table 1 lists the operations and color information available to the evolutionary algorithm.

Table 1		
Unary Operator Nodes		
Negation	-X	Negates the single child node's value.
Binary Operator Nodes		
Addition	X+Y	Adds the children together.

Leaf Nodes		
RGB Red	[0-255]	The 'red' value of the RGB format of the color.
RGB Green	[0-255]	The 'green' value of the RGB format of the color.
RGB Blue	[0-255]	The 'blue' value of the RGB format of the color.
Luminosity	[0-100]%	The brightness of a color as it appears to the eye.
Hue	[0-360]°	A directional value associated with HSL, HSL color formats.
HSL Saturation	[0-100]%	The 'intensity' of a color calculated for the HSL color format.
HSV Saturation	[0-100]%	The 'intensity' of a color calculated for the HSV color format.
Lightness	[0-100]%	The average of the largest, smallest RGB color components.
HSV Value	[0-100]%	Describes how dark a color is in HSV color format.
Constant	[0-1]	A floating point value between 0 and 1. Not related to the color itself.

The depth of these functional trees is not fixed over the course of each experiment, however, parsimony pressure is employed to penalize complexity in cases where it did not assist in the fitness of the tree. At the beginning of each experiment, a population of configurable size is initialized via a partially random initialization function. When initializing each node in the tree, the function has a $\frac{1}{1+d}$ chance of being a random type of operator (where d is the depth of the node), otherwise the node is initialized as a random type of leaf. This helps to create a good source of genetic material for the initial population, while limiting the average size of each starting tree. The initial population is then scored on its fitness.

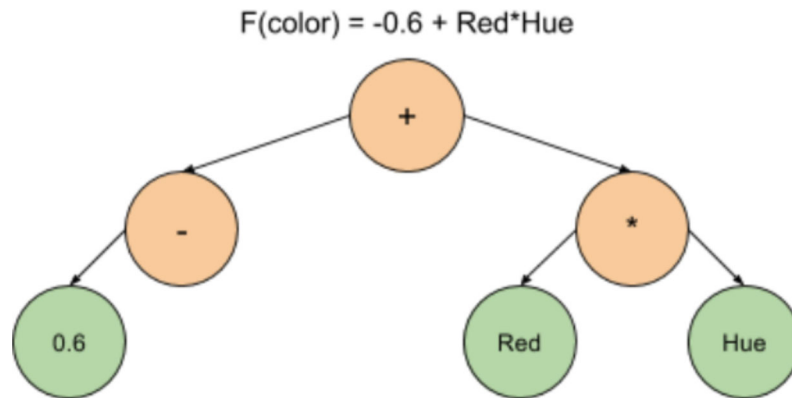


Figure 3: An example tree.

During each generational iteration of the evolutionary algorithm, 2-tournament selection is used to select parents, and crossover is used to create offspring for the next generation. Each pair of parents generates two offspring. The offspring are created by cloning the parents, selecting a random node in each clone (excluding the root), and swapping it and its subtree between the two clones. Each offspring then has a configurable chance to mutate. If an offspring mutates, a random node is selected and re-initialized as a new subtree, using the same methods in initialization. The offspring are then given a fitness score and added into the overall population.

After the new generation is complete and added into the population, 2-tournament selection is once again employed to select survivors and cut the population back down to its original size. This generational iteration then repeats until a configurable amount of generations have occurred. The highest fitness individual in the final population is considered the solution for

‘wins’ the tournament, and becomes the selected individual. For parent selection, the winner for the 2-tournament selection remains in the pool to potentially be selected again for this generation of mating. For survival selection, the individual is removed from the pool so that no duplicate individuals may be selected to survive.

Fitness Function

Determining the fitness of a function tree is a multi-step process. For any given experiment a sorting order scoring function is selected to be used in fitness evaluation. In addition to a scoring function, two configurable parameters are selected: a ply, and sort length. To determine the fitness, a number of randomly generated lists of colors are created. The number of lists is the ply, while the lengths of the lists are the sort length. The individual being scored is then used to evaluate each color in each list and the lists are then sorted according to their evaluated number. Each now sorted list is then scored based on the sort scoring function to determine how well sorted they are (see Table 2 for a list of scoring methods). All scoring functions score better sorts with lower values. These scores are then added up and divided by the ply to find an average score of how well the tree evaluated the colors to be sorted. After the average is found, a parsimony pressure penalty of 0.5% per node is applied to the trees, to avoid trees that are too deep. After this, the individual’s fitness complete, and is saved.

Table 2. Sort Scoring Methods

TSP_RGBHSLSV	The total 'path length' is found where every property of each color is considered at once, not just in sets of three. This mimics the traveling salesman problem in 8-dimensional space.
--------------	--

Table 3. Configurable Parameters		
Parameter	Default Value	Description
Evaluations	500	The number of generations to be executed in the experiment.
μ (Mu)	200	The number of individuals (trees) to be in a given population.
λ (Lambda)	100	The number of offspring to be created in each generation.
Fitness Ply	3	The number of color sorts to be averaged when scoring a tree.
Sort Length	50	The length of the color lists to be sorted when scoring a tree.
Node Exclusion List	['Hue']	A list of node types to be excluded from this experiment.
Sort Scoring	TSP_RGB	The function to be used for scoring how well a color list is sorted.
Mutation Rate	5%	What percent of the time does a newly created offspring mutate.

Results

A series of experiments were run, varying up the parameters and configurations of the evolutionary algorithm. Below are the results of a handful of those runs. Some runs which produced different results use the same configuration. Each result is a list of 1000 colors sorted numerically by score using the best evaluation function evolved. Above each result, a list of configurations which vary from the default are listed. Additionally, the evaluation function itself is listed.

Using TSP_RGBHSLSV scoring function



$$f(\text{color}) = \text{Green} / \text{Blue}$$

Using TSP_RGBHSLSV scoring function



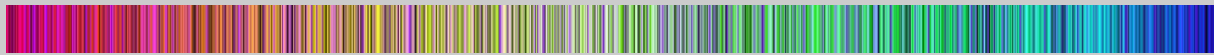
$$f(\text{color}) = (((\text{Min}(\text{Red}, \text{Green}) + 0.339) / \text{Green}) + \text{Min}(\text{Min}(\text{Blue}, \text{Min}(\text{Blue}, \text{Green})), \text{Green})) / \text{Green})$$

1000 evaluations



$$f(\text{color}) = \text{Max}(\text{Luminosity}, \text{Blue})$$

2000 evaluations



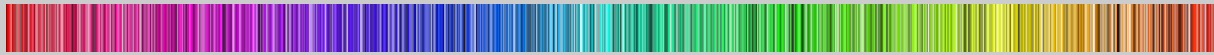
$$f(\text{color}) = (-\text{Red} + (\text{Blue} - \text{Red})) / \text{Green}$$

Not excluding 'Hue'



$$f(\text{color}) = \text{Max}(\text{Hue}, \text{Luminosity})$$

Not excluding 'Hue', excluding 'Max' and 'Min'



$$f(\text{color}) = -\text{Hue}$$

Fitness ply of 5



$$f(\text{color}) = ((\text{Lightness} + (-\text{Lightness} * (\text{Blue} * (\text{Lightness} + -\text{Luminosity})))) / \text{Red})$$

Sort length of 100



$$f(\text{color}) = (\text{Luminosity} / \text{Max}(\text{Blue}, (\text{Lightness} + \text{Green})))$$







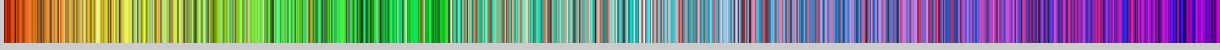

Sort length of 100, using TSP_RGBHSLSV scoring function



$$f(\text{color}) = ((\text{Green} + -\text{Blue}) * ((\text{Blue} / (\text{Red} / (\text{Green} / (\text{Red} / \text{Green})))) / (\text{Red} / (\text{Green} / (\text{Red} / \text{Blue}))))))$$

Mu of 400, lambda of 200, using TSP_RGBHSLSV scoring function



<p><i>Using TSP_RGB_HSL_HSV</i></p>  <p>$f(\text{color}) = \text{Blue} / \text{Green}$</p>
<p><i>1000 evaluations, using TSP_RGB_HSL_HSV scoring function</i></p>  <p>$f(\text{color}) = \text{Blue} / \text{Green}$</p>
<p><i>1000 evaluations, not excluding 'Hue', using TSP_RGB_HSL_HSV scoring function</i></p>  <p>$f(\text{color}) = -\text{Hue}$</p>
<p><i>Not excluding 'Hue,' using TSP_RGB_HSL_HSV scoring function</i></p>  <p>$f(\text{color}) = -\text{Hue}$</p>
<p><i>Not excluding 'Hue,' using TSP_RGB_HSL_HSV scoring function, sort length of 1000, 200 evaluations</i></p>  <p>$f(\text{color}) = -\text{Hue}$</p>
<p><i>Not excluding 'Hue,' using TSP_RGBHSLSV scoring function, sort length of 1000, 200 evaluations</i></p>  <p>$f(\text{color}) = \text{Max}(\text{Hue}, \text{Luminosity})$</p>
<p><i>Mutation rate 1%, 200 evaluations</i></p>  <p>$f(\text{color}) = (\text{Green} / (-\text{Max}(\text{Green}, (\text{Blue} + \text{Luminosity}))) + \text{Luminosity})$</p>
<p><i>Using TSP_RGBHSLSV scoring function, sort length of 1000, 200 evaluations</i></p>  <p>$f(\text{color}) = \text{Blue} / \text{Green}$</p>

Conclusion & Future Works

Over the course of these experiments, many things became apparent. These small

First of all, in any experiment where hue was not excluded as a potential node, hue tended to dominate in the population. Since trees with only one node were not allowed in the population, the function 'f(color) = -Hue' was very common among these runs. The author concludes this is because of the non-trivial calculation that goes into finding hue. Since hue is found by compounding other information about a color, particularly its RGB values, in any run where hue was not excluded it quickly became a local optima to which the population converged. This guided the decision to make excluding hue a default among other experiments.

The next discovery of note is what happens when the scoring function included more than just RGB path length. In most runs where TSP_RGB_HSL_HSV or TSP_RGBHSLSV was used, green and blue became especially important. In many of these experiments, the populations converged to 'f(color) = Green / Blue,' or some very similar function including green and blue.

Another find of note is that typically when sort length was higher, results appeared overall better. The author believes that this is due to the fact that having more colors to sort when determining fitness tends to find and exploit the finer differences between evaluation functions, guiding the population more steadily toward better solutions. The author believes that this trend would synergize well if the mutation rate was raised and the number of evaluations increased, so that a higher variety of combinations can be tried against the finer scoring. This is more computationally expensive however, and would require more time.

Overall the various results had a few things in common. Over the course of all the

selected for over the course of evolution. Numerically, some colors can be close in RGB space, for example, without being close in luminosity, thus leading to color banding for adjacent colors.

In all the experiments, the resulting evaluation functions were fast, proving that there is at least a computational benefit to machine learning driven color sorting. Many improvements to the algorithm, however, can be made to increase the overall quality of these solutions. Including luminosity in fitness scoring would be a good first step to improving color banding. Attempting to include other color spaces like LAB (Lightness, Green-red, Blue-yellow) or CMYK (Cyan, Magenta, Yellow, Key) could increase the probability of appealing properties being found. Program efficiency can be improved to increase the potential for higher ply values, more evaluations, or longer sort lengths, which tend to produce better results. New nodes could be introduced to allow other functionality for evaluation functions. In any case, these results go to show that machine learning and evolutionary algorithms may be a potential source of color sorting solutions.

References

[1] Zucconi, A. (2015, September 30) *The Incredibly Challenging Task of Sorting Colours*. Retrieved from <http://www.alanzucconi.com/2015/09/30/colour-sorting/>

Images in figure 1

<https://upload.wikimedia.org/wikipedia/commons/thumb/c/c2/AdditiveColor.svg/2000px-AdditiveColor.svg.png>

<http://www.beliefmedia.com/r/2017/05/485/hsv-hsl-color-models-800.png>