– Independent Work Report Spring 2020 –

Color Charts

Karen Ying

Adviser: Alan Kaplan

Abstract

This paper describes a Chrome extension called Color Charts that aims to increase the readability of charts, graphs, and diagrams on the web. Visuals that depend on colors to convey information may be hard to read for people with color blindness. This extension aims to improve upon existing extensions by using researched color blind friendly palettes. It allows for filtering of visuals on a web page, with the option to choose between 4 different color blind friendly palettes. Built with vanilla JavaScript, Color Charts interacts with HTML <img> elements in the Chrome browser, utilizing HTML canvas and base64 encoding to filter the images. This ultimately improves the web experience for color blind users by allowing them to better interpret charts, graphs, and diagrams.

1. Introduction

Color vision deficiency (more commonly known as color blindness) represents a group of conditions relating to the perception of color. In humans, there are many types of color blindness. Three of the most common types in order of frequency are red-green, blue-yellow, and complete. This condition affect approximately one in twelve men and one in two-hundred women of Northern European descent [6]. In America, around thirteen million people are color blind [6].

Color blindness leads to a reduced ability to distinguish between certain colors like red and green, or blue and yellow. Most visual content, such as charts, graphs, and diagrams, rely on the use of many colors to better convey their data and information. Thus it can be difficult for color blind people to effectively process these visuals. Figure 1 shows a what an example graph looks like to a person with normal color vision. Figure 2 is the same graph processed with a protanopia (red-green

color blind) filter, simulating the condition. The absence of the (long wave) L-cone in the eye makes it so that a person with protanopia cannot distinguish between red and green. A simple line graph that may seem intuitive for a person with normal color perception to read can be difficult for a person affected by color blindness to interpret.

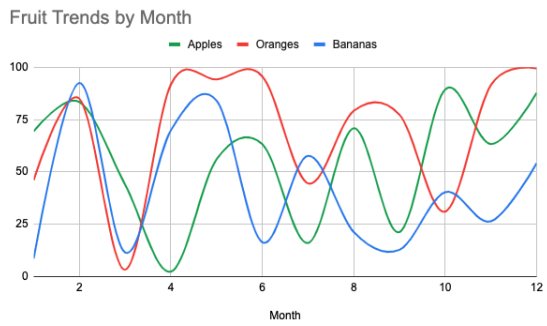


Figure 1: Example graph.

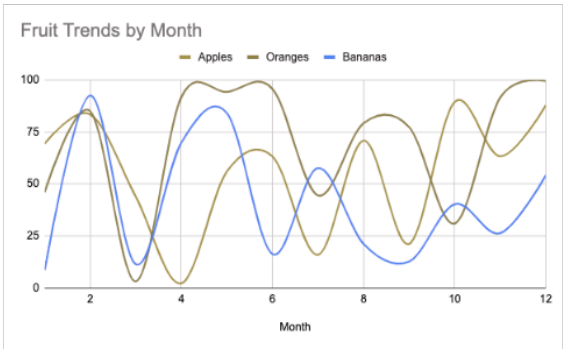


Figure 2: What someone with protanopia sees.

The goal of Color Charts is help color blind people better interpret charts, graphs, and diagrams on the web. Reading articles or papers with such visuals could be difficult for people with color blindness if the creators of the visuals do not consciously choose color blind friendly palettes.

Currently, the Chrome extension store offers a couple of color blind accessibility extensions that process the entire browsing experience. However, these extensions filter based on the original input, creating an output that depends on the input colors. They also do not filter only images, instead applying the color transformation to the entire browser window. Thus a badly colored diagram might not be improved with the use of these existing Chrome extensions. This is where color blind friendly palettes come in. Concurrently, there has been research done on optimal colors for color blind people. Color Charts aims to combine these two applications into a Chrome extension which selectively changes visual content, specifically charts, graphs, and diagrams, to preset color blind friendly palette colors.

Color Charts implements three main steps to create such an extension:

1. **Native image filtering**: recolor images in-line and preserving the structural integrity of the page

2. **Coloring algorithm**: create a coloring algorithm that recolors images into color blind friendly palette colors only
3. **User interface**: design an intuitive and easy-to-use extension interface

To evaluate effectiveness, Color Charts is compared to existing Chrome extensions. The result is that for specific instances of badly colored charts, Color Charts performs better than other Chrome extensions. Because Color Charts is input color agnostic, the resulting recolored visual is presumably easier to interpret for color blind users.

2. Background and Related Work

Color Charts aims to combine these two fields of research — Color blind friendly palettes and available Chrome extensions for color blindness.

2.1. Color Blind Friendly Palettes

Canadian scientist Martin Kyrzywinski has conducted research on color blind friendly palettes, creating suggestions for 12-color, as well as 15-color palettes [7]. Figure 3 shows Kyrzywinski's 12-color palette along with that the twelve colors would look like for people with deuteranopia, protanopia, and tritanopia.

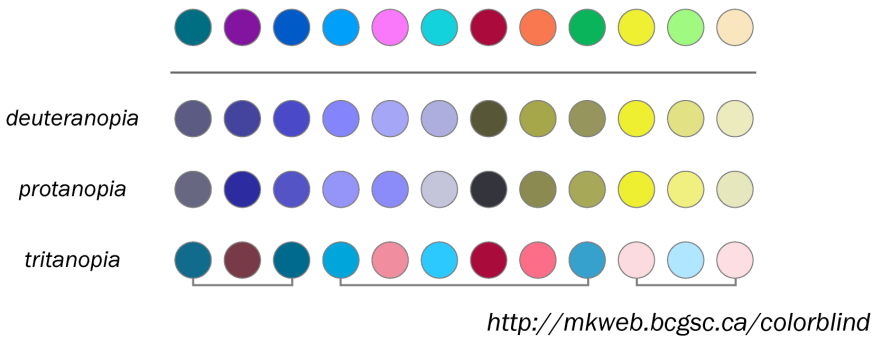


Figure 3: Kyrzywinski's 12-color palette [7].

Additionally, Japanese scientists Masataka Okabe and Kei Ito have proposed an 8-color palette [8]. Figure 4 shows the palette along with what the palette would look like to people with protanopia, deuteranopia, and tritanopia.

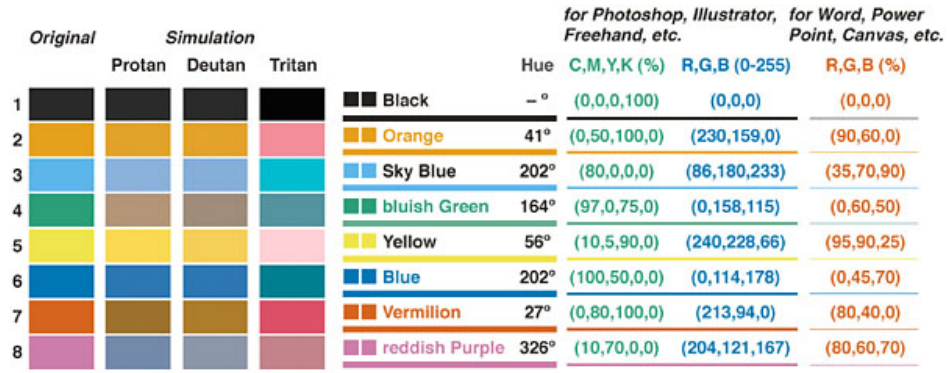


Figure 4: Okabe and Ito's 12-color palette [8].

Finally, Instrument Scientist Paul Tol created several qualitative color palettes in varying shades of brightness. Figure 5 shows the palettes with varying brightness.

Figure 5: Tol's palettes with varying brightness [9].

Color Charts aims to incorporate this existing research done on color blind friendly palettes in the form of a Chrome extension to process charts, graphs, and diagrams.

2.2. Color Blind Accessibility Chrome Extensions

Chrome offers a couple of color blind accessibility extensions in their web store. Under "Use Chrome with accessibility extensions", Chrome suggests a third party extension called Color Enhancer [1]. Under setup, the extension asks the user for their preferences to calibrate the output colors. Figure 6 shows the setup process. Based on user preferences, Color Enhancer filters the entire browser.

After setting up the preferences, the user can enable the filters and adjust the filter strength. Figure

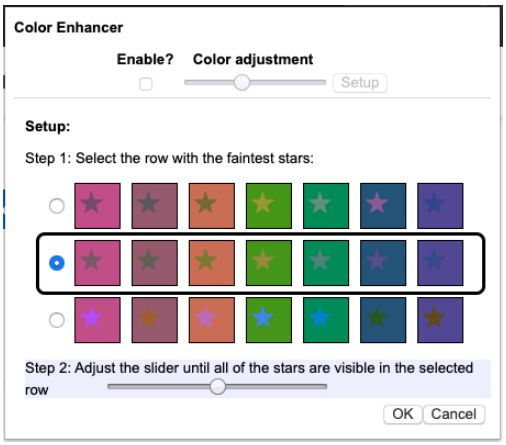


Figure 6: Color Enhancer Calibration Setup.

7 shows the original graph from earlier and Figure 8 shows the filtered image after Color Enhancer is enabled. Based on the setup settings, Color Enhancer takes the original colors and applies a filter to the entire browser, changing the colors to the user's preferences. Output colors *are dependent* on input colors.

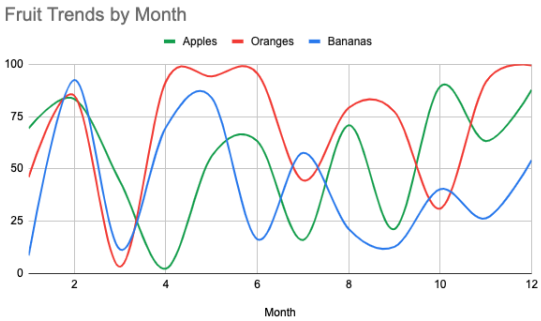


Figure 7: Example graph.

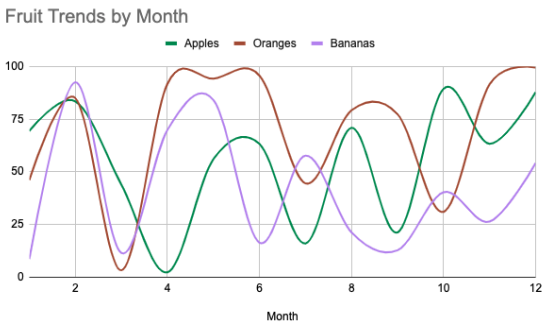


Figure 8: Example graph after Color Enhancer.

However, to someone with protanopia, even after Color Enhancer, the red and green lines in Figure 7 might still not be readable. Figure 9 below shows what the original graph looks like to someone with protanopia. Figure 10 is what the same graph, after applying Color Enhancer, looks like to someone with protanopia.

The red and green lines are still almost indistinguishable. Thus a graph that was poorly colored originally, may still be unreadable even after using a color filtering extension like Color Enhancer. This is because Color Enhancer uses the input color to generate the filtered output. Colors that may

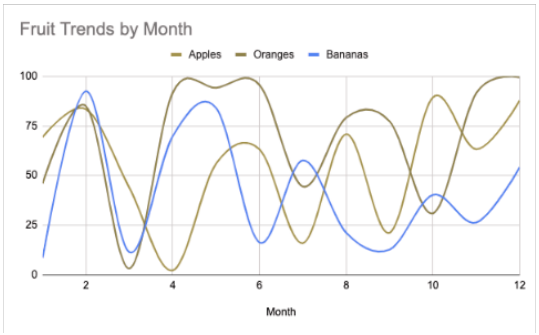


Figure 9: What someone with protanopia not using Color Enhancer sees.

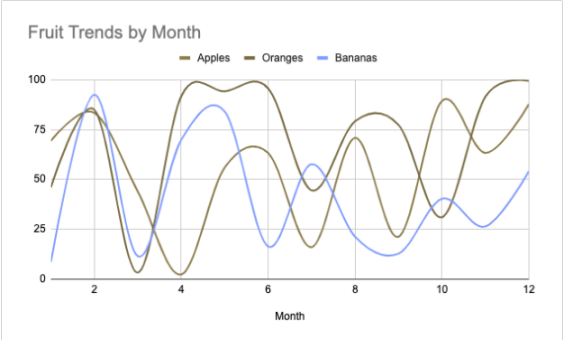


Figure 10: What someone with protanopia using Color Enhancer sees.

have had problematic relationships in the original image may still be mapped in the same relative problematic color scheme in the filtered output.

Another popular Chrome extension is called Colorblind - Dalton for Google Chrome [2]. It operates in a similar manner to Color Enhancer. It also takes in user preferences to filter the entire browser based on the original input colors.

These two extensions have common product designs — the user takes an adjustment test in the beginning which determines how the extension changes their Chrome browser's colors. The initial calibration changes the original colors on the screen according to an algorithm. This means the colors of the end result *depends* on the color of the input. It also means the *entire* page changes colors. This may not be ideal if the color blind user is only concerned with reading graphs, charts, and diagrams. A better solution for increasing readability for color blind users is if one could change the colors of the visual to preset palettes, instead of relying on algorithmic changes to the visual's original colors.

3. Approach

The key idea behind the approach of Color Charts is a Chrome extension that filters images (specifically visuals such as charts, graphs, and diagrams) on a page with researched color blind friendly palettes. The user has the option to apply the filter to *all* images found on the page, or to only apply filtering on selective images. The colors of the filtered images will only consist of the ones in a specific color blind friendly palette. Unlike Color Enhancer and Dalton, the output colors

will *not* be dependent on input colors. This ensures that the colors of these visuals are consistent across different web pages that use different original colors.

Users can select which of the 4 palettes they would like to use. Additionally they can also choose if they want every image on the page to be filtered. If so, every image on the page will be filtered using the chosen palette. If they want to selectively filter, they can right click on the image and choose which palette to filter the selected image. By having preset colors that are known to be color blind friendly, these charts, graphs, and diagrams can be better read by color blind web users.

Color Charts faces a couple of technical limitations. First and foremost, it can only recolor HTML <img> elements. It does not work in PDFs, or on SVGs. Additionally, the extension only applies to images on the same domain as the page it appears on. It is not compatible with cross-origin images (images whose source is from another domain).

4. Implementation

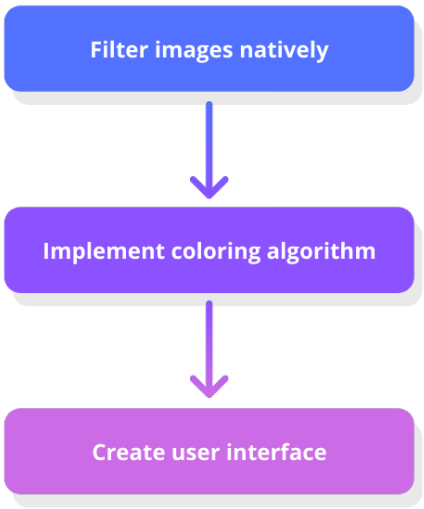


Figure 11: Color Charts problem overview.

At its core, the Color Charts recolors images on the page. For a smooth user experience, the original images should be filtered and replaced without changing the layout of the page — Color Charts filters images *natively*. Next, the extension implements a coloring algorithm. Finally, the user

should also be able to selectively filter images and choose which palette they prefer — Color Charts has a user interface. This section discusses how Color Charts tackles these three steps outlined in Figure 11.

4.1. Filtering Images Natively

In order to filter images a web page, an extension must access the <img> tag of the image that is being filtered. On a page, every <img> tag contains a src attribute that is set to the URL of the image. This is so the browser knows where to pull the image from. Since Color Charts filters this image, it must either replace the image with another kind of HTML element such as a <canvas> element, or it must supply the image's src with the URL of the filtered image.

HTML Canvas allows Color Charts to implement its coloring algorithm by "drawing" the image on the canvas and manipulating the image's pixel data. To replace the <img> element with a <canvas> element, the extension can remove the <img> node in the HTML and replace it with the newly created <canvas> node. However, by doing so, the CSS of the original <img> element does not get applied to the new <canvas> element. This results in a change in the layout of the page.

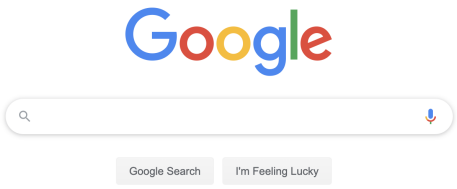


Figure 12: Original image [3].

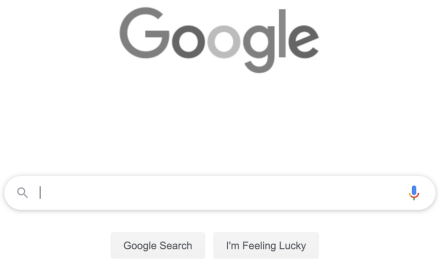


Figure 13: Image replaced with a <canvas>.

Figure 13 shows Figure 12's image replaced with a <canvas> element that applied a grayscale filter to the original image. The canvas did not inherit the image's CSS top padding, resulting in a change page structure. Since Color Charts aims to preserve page structural integrity, this is not a feasible solution.

Instead, Color Chart implements HTML Canvas's *toDataURL()* method. This method takes the

filtered image information and converts it into a data URI representation in base64-encoded binary data. The encoded data is stored as a string with the prefix "data:", representing the canvas data in PNG format. In the place of a URL indicating where the image is stored, an <img> element's src attribute can also directly take a data URI and render the image based on the data embedded in the URI. Thus Color Charts utilizes HTML Canvas's *toDataURL()* method to convert the canvas's filtered image data into a data URI and directly embed the data in the <img>'s src. Since Color Charts replaces an <img> element with a different <img> element, CSS is preserved and images are filtered natively.

4.2. Implementing Coloring Algorithm

Color Charts utilizes a two pass process to take an image with variable numbers of colors and a known color blind friendly palette, and outputs an image consisting of colors only in the palette. This subsection describes the implemented algorithm.

HTML Canvas allows for direct manipulation of the RGB values of the pixels in the current image on the canvas. Each pixel consists of three values: red, green, and blue. The values range from 0 - 255, where a larger number means a larger influence of the corresponding color. In this paper, a color in the RGB color space is represented as (red, green, blue). Red is represented by (255, 0 , 0), green is (0, 255 , 0), and blue is (0, 0 , 255). Every pixel's color is represented in this way in HTML Canvas. Thus, Color Charts iterates over the image's height and width, changing pixels' RGB values to recolor the image. This section explains the decisions of the recoloring algorithm.

Visuals often use grayscale colors such as varying shades of gray, white, or black, for text and daring gridlines. Color Charts leaves these unchanged. Grayscale colors have the property where their RGB values are the same. For example, white is (255, 255, 255), black is (0, 0, 0), and medium gray is (127, 127, 127). Thus Color Charts calculates the differences between the red and green, green and blue, and blue and red values. Only if one of these differences is above a threshold, the algorithm recolors this pixel.

While charts, graphs, and diagrams might only seem to have a couple of colors, in reality, the

image is filled with inconsistencies. A red line on a line chart might seem to only be red to the human eye. However, zooming in reveals that the edges of the line are speckled with a dark red or even orange. This means that a visual that intends to only uses a couple of colors to convey its information, actually contains many times more colors. While iterating over the pixels, the algorithm can easily pick up on these blemishes, when the human eye cannot. In the case of the red line, Color Charts wants to recolor the entire "red" line into one color, not many different ones. To deal with this, the algorithm stores the twelve main color wheel colors shown in Figure 14.

Figure 14: Twelve main colors of an RGB color wheel [4]

.

When given multiple colors, humans can judge the "closeness" of any two given colors. Comparing two colors in the RGB color space does not effectively mimic human perception. Designed to approximate human vision, the CIE L*a*b* (CIELAB) colorspace better compares colors. Much like RGB, CIELAB stores the pixel's information in three values. For each pixel, Color Chart's algorithm converts the pixel from RGB to CIELAB. If CIELAB values are interpreted as a points in 3D, "closeness" between two colors is defined the straightline distance between the two points. Thus the algorithm iterates through the twelve RGB color wheel colors and recolors the pixel with the color wheel color that is "closest" to the pixel. This recoloring to color wheel colors will be referred to as *preprocessing* from now on.

Now the image's colors (not including the grayscale colors) are a subset of the color wheel colors. This preprocessing will make it easier to recolor to the color blind friendly colors. While preprocessing a pixel, a string representation of its color wheel color is stored as a key in a hashmap called *colorMap* and its corresponding value is incremented. At the end of preprocessing,

colorMap's keys are string representations of the color wheel colors that are now present in the recolored image, and the corresponding values is the number of pixels that are this color. Color Charts then iterates through *colorMap*. If the color wheel color in *colorMap*'s value is above a number of pixels, Color Charts deems this color significant enough to recolor into the chosen color blind friendly palette. In this case, the algorithm sets the value as the RGB representation of a color blind friendly palette color. If the color wheel color is not significant, this means the color does not significally contribute to the overall visual. It could be a couple of specks, or an outline. Its value gets set to the RGB representation of white. After this iteration, *colorMap* contains a mapping of color wheel colors to color blind friendly colors.

Finally, Color Charts iterates through the pixels once more to recolor the image into color blind friendly colors. It does so by using *colorMap*: if the pixel is a color wheel color, the algorithm looks up the color wheel color in *colorMap* and recolors the pixel using its corresponding value. The value is a color blind friendly color if the color wheel color is significant, or white in the case that it is not.

4.3. Creating User Interface

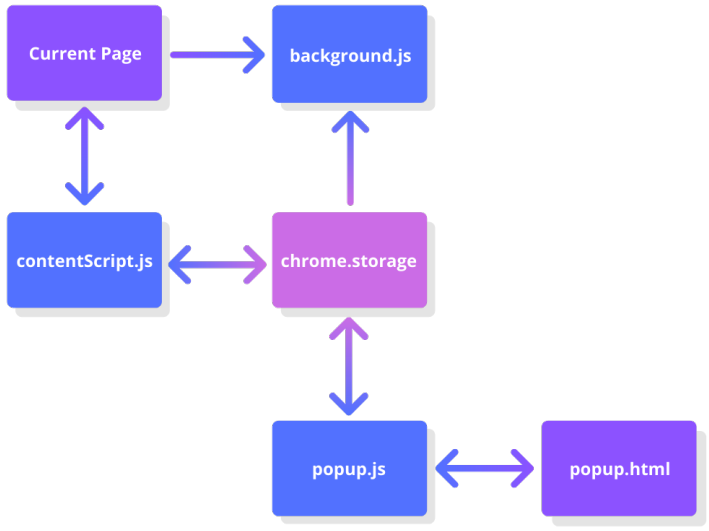


Figure 15: Color Charts architecture.

The front facing user interface of Color Charts consists of the popup and the right-click context

menu. The scripts that run these interfaces read and write from Chrome's Storage (chrome.storage) to achieve persistence. Built on vanilla JavaScript, Color Charts consists of several scripts and HTML files. Figure 15 diagrams how the different files interact with each other. An arrow pointing from object *A* to object *B* means that *A* writes to *B*, and *B* reads from *A*. This figure will be a useful reference throughout this section.

Upon clicking the extension's logo to the right of Chrome's address bar, the popup displays on the screen. Figure 16 shows Color Chart's popup.

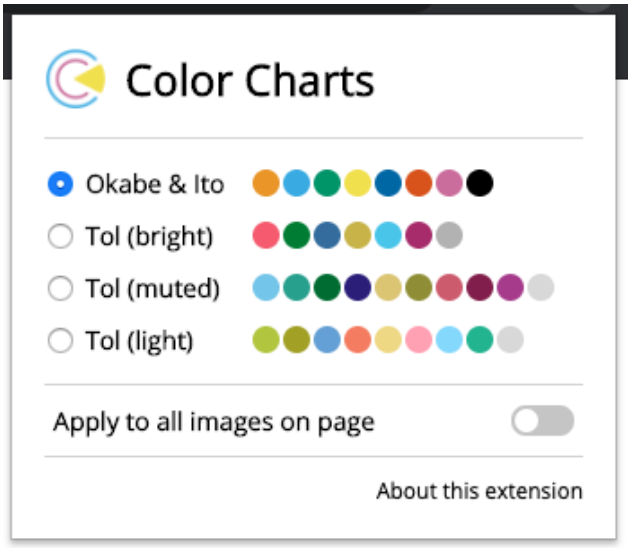


Figure 16: Color Charts popup interface.

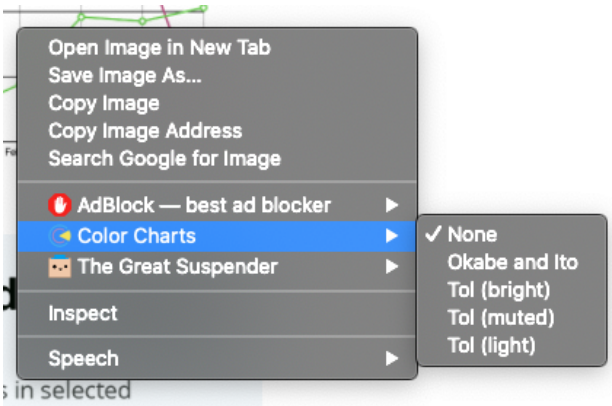


Figure 17: Color Charts context menu.

The popup section below the title is a menu of researched color blind friendly palettes. The credited names and palette are shown next to each option. It includes Okabe and Ito's 8-color palette

[8], as well as three Tol palettes [9]. The next popup section features a toggle “Apply to all images on page”. When the toggle is off, users are able to selectively choose which images they want to filter by right-clicking on the image. Upon right-clicking the desired image, the context menu part of the user interface appears as shown in 17. When the toggle is on, the right-click menu does not appear.

As briefly mentioned before, Color Charts uses Chrome’s Storage API to achieve persistence across the user interface. Table 1 illustrates the implemented caching schema.

Key	**Value type**	**Example values**
applyAll	Boolean	true, false
paletteSelected	String	’Okabe and Ito’, ’Tol (bright)’
originalLinks	Array	[’https://karenying.com/color-charts-dummy-page/images/1.png’, ’https://karenying.com/color-charts-dummy-page/images/2.png’ ’https://karenying.com/color-charts-dummy-page/images/3.png’]

Table 1: Color Charts caching schema.

The current state of the “Apply all” toggle is stored in the key *applyAll* in the cache. When the toggle is on, Color Charts filters every image on the page. In this case, *applyAll* is set to true. It first reads in the *paletteSelected* value from cache to know which filter to use. Then the extension caches all the links of the images in the *originalLinks* array of the cache. This is necessary so that the page can be reverted to normal if the user desires to toggle “Apply all” off. Finally, Color Charts iterates through every image on the page, recoloring them using the filtering process described before, generating a base64 URI which replaces the original image. After this process, every image on the page is recolored to the selected color palette. If at this point, the user wishes to change the palette selected, they can choose another palette from the popup menu. This action changes the value of *paletteSelected* in cache. The cache has an *onChange* listener. Once detecting the change, Color Charts repeats the filtering process for every image on the page using the newly selected palette.

When the user toggles “Apply all” off. The value of *applyAll* in the cache changes to false. Color Charts detects this change and reverts all the images on the page back to the original images by retrieving the original links from cache. Upon right clicking an image, the user sees the context menu in Figure 17. When the user clicks the desired palette from the menu, a message is created with a body consisting of the url of the image right-clicked and the palette selected. Color Charts reads this message, iterating through all the images on the page to find the image that matches the url in the message. Once found, the image is filtered using the palette from the message. If the user decides to switch the palette for the image, they can right-click on the same image and the whole message-sending and filtering process repeats.

4.4. Installing Color Charts

The source code for Color Charts is available as open source software on GitHub.

To use:

1. Clone the GitHub repository
2. Head over to Chrome Extensions (usually “chrome://extensions” in the address bar) and turn “Developer Mode” on the top left.
3. Click “Load unpack” and select the destination as the cloned repository folder
4. The Color Charts icon should now appear to the left of the address bar. Click on the icon to trigger the popup and utilize the extension as described in this section.

Throughout development, this page was used for testing. It includes a couple of visuals that are same-origin images. This makes it the optimal page for Color Charts usage.

5. Evaluation

We will be evaluation Color Charts against the existing Chrome Extension Color Enhancer. To simulate color blindness, we will use an online color blind simulator tool called Coblis [5]. Coblis takes in an uploaded image and simulates what people with different types of color blindness see. The following tables all follow the same format: given a sample chart of three colors, each box shows what the same chart after using an extension (or not) roughly looks like to someone with

normal color perception or someone with a type of color blindness. Each experiment features a different graph with different colors and a different type of color blindness.

For the first experiment, the primary colors of the graph is red, green, and blue. The color blindness type is protanopia. A person with protanopia is unable to perceive red and green. Table 2 shows the results of this experiment.

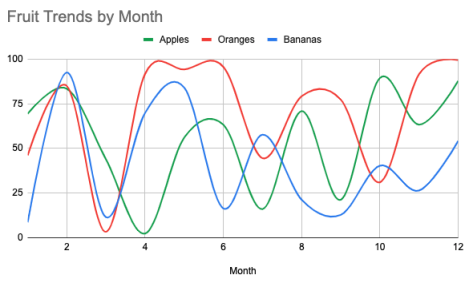
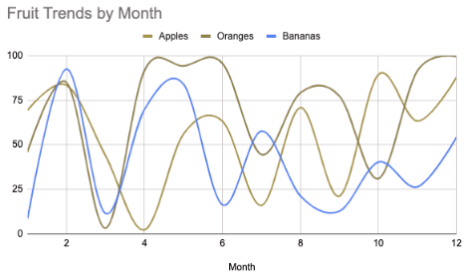
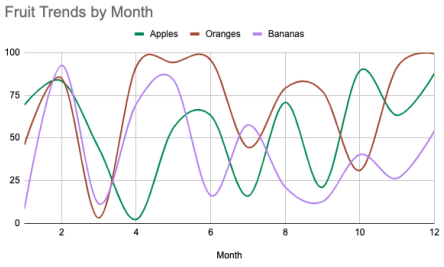
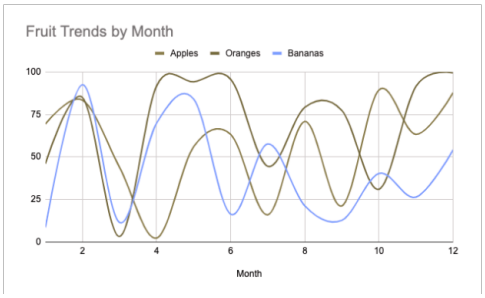
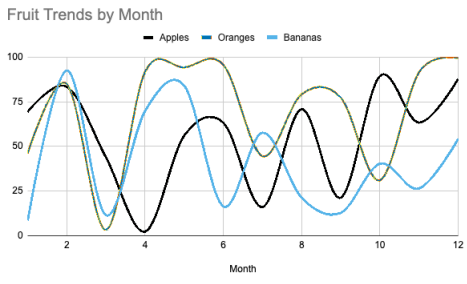
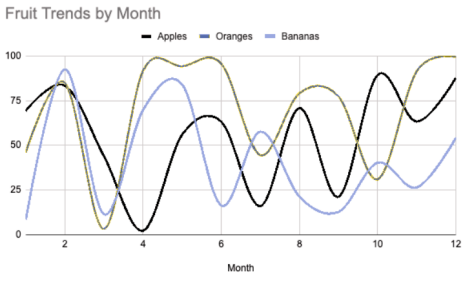
	Normal color perception	**Protanopia**
No extension		
Color Enhancer		
Color Charts		

Table 2: Color Enhancer vs. Color Charts for Protanopia.

While Color Enhancer changes the colors of the original images, to someone with protaniopia, there is almost no difference between using Color Enhancer or not. This is the same problem as mentioned before: Color Enhancer adjusts colors based on input colors. Thus the problematic

color combination of red and green is not improved using Color Enhancer — they look the same to someone with protanopia. On the other hand, Color Charts completely remaps the red, green, and blue, using Okabe and Ito's 8 color palette [8]. Red becomes brown, blue becomes a lighter blue, and green becomes black. To a user with protanopia, this is a much better color combination as seen in the Color Charts x Protanopia graph.

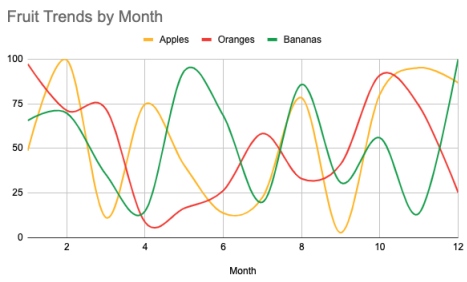
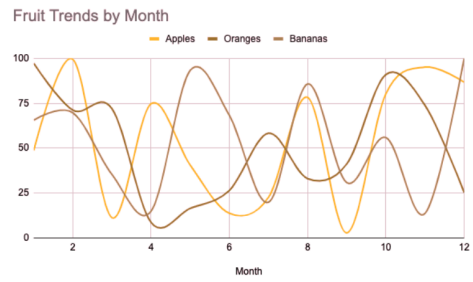
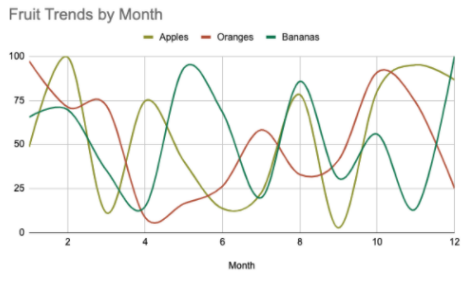
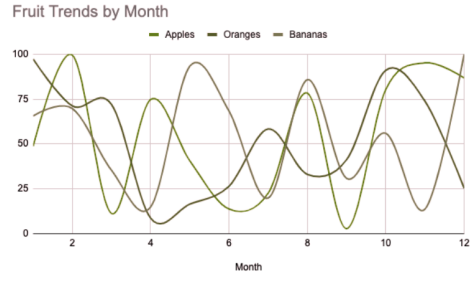
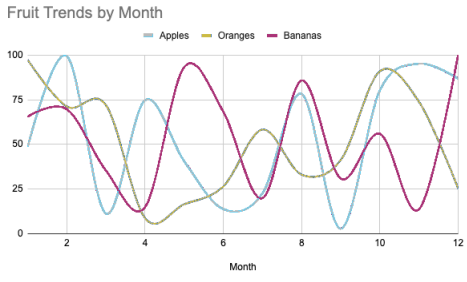
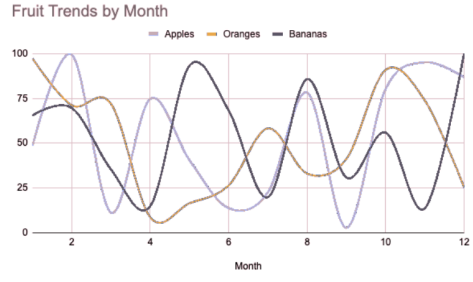
	Normal color perception	**Deuteranopia**
No extension		
Color Enhancer		
Color Charts		

Table 3: Color Enhancer vs. Color Charts for Deuteranopia.

The second experiment focuses on different type of red green color blindness called deuteranopia. Similar to protanopia, a person with deuteranopia has difficulty distinguishing between red and

green. The chart this time features red, green, and yellow. Table 3 shows the results of this experiment.

As with the first experiment, Color Enhancer fails to adjust for deuteranopia and does not sufficiently adjust the red and green of the chart. Color Charts uses the Tol Bright 7-color palette [9] to completely recolor the graph into blue, magenta, and green. This color combination is much more deuteranopia-friendly.

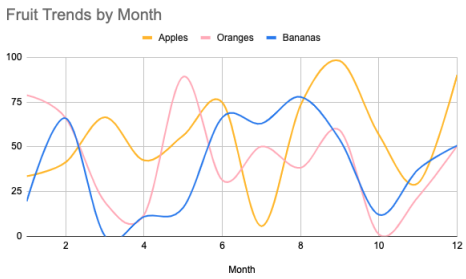
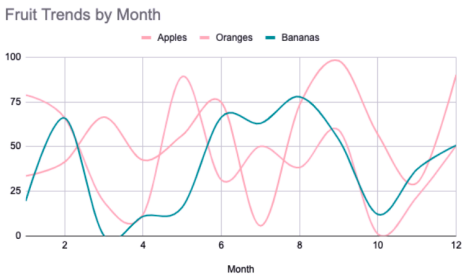
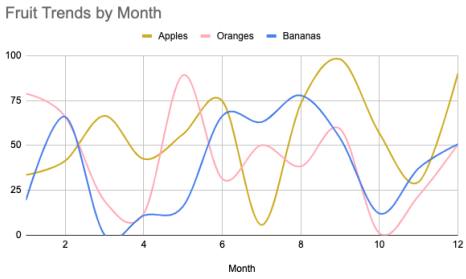
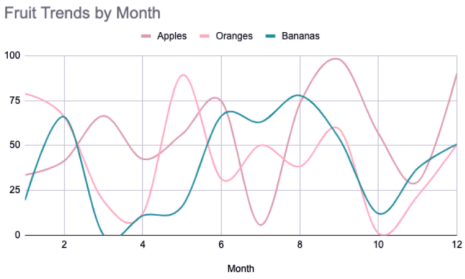
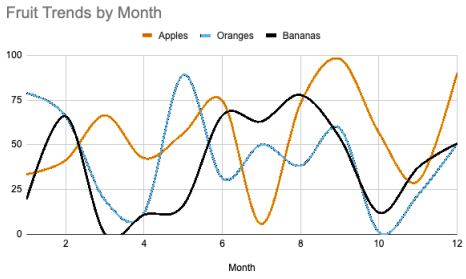
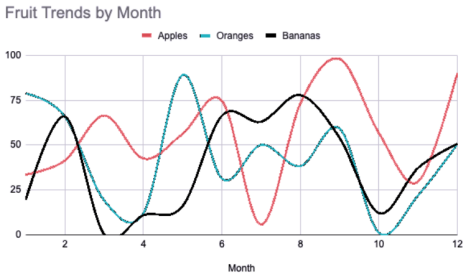
	Normal color perception	**Tritanopia**
No extension		
Color Enhancer		
Color Charts		

Table 4: Color Enhancer vs. Color Charts for Tritanopia.

Finally, the last experiment features a chart with pink, yellow, and blue lines. The color blindness

type tested is tritanopia. A person with tritanopia cannot distinguish between blue and yellow. Table 4 shows the results of this experiment.

Again, there is not much difference between using Color Enhancer and not. The pink and yellow remains hard to read for someone with tritanopia. Instead, Color Charts uses Okabe and Ito's 8 color palette [8] to recolor the problematic graph into a much easier to read graph for someone with tritanopia.

In each of these experiments, Color Charts is compared to Color Enhancer, the recommended color blindness accessibility extension. Three types of charts and color blindness are utilized to evaluate the effectiveness of the two extensions. Compared to Color Enhancer, Color Charts succeeds to recoloring visuals into more color blind friendly palettes.

6. Summary

Through filtering images natively, implementing a coloring algorithm, and creating a user interface, Color Charts increases the readability of charts, graphs, and diagrams on the web. It improves upon existing color blindness accessibility extensions using researched color blind friendly palettes. In doing so, it can be considered a success compared to other extensions. However, Color Charts also faces a variety of issues that limits its current usefulness as an extension.

6.1. Limitations

First and foremost, it does not deal with cross-origin images. A cross-origin image is an image that appears on a page in domain A, however, its source resides in domain B. An example would be embedding an image from domain B on a page in domain A. This is very common across webpages because often images reside in a different domain from the one they appear on for reasons such as page performance and scalability. The problem with cross-origin images is that HTML Canvas produces error messages such as "Unable to get image data from canvas because the canvas has been tainted by cross-origin data" when trying to pull image data from a cross-origin image. HTML Canvas does not work with cross-origin images. Since Color Charts heavily relies in HTML Canvas,

it is unable to recolor such images if HTML Canvas cannot render them. Thus the extension is effectively useless on the majority on web pages.

Another issue is that when users toggle between different palettes either in the popup or the context menu, Color Charts recolors the image on top of the filtered image. This is problematic if the filtered image contains colors that are close to each other. Upon applying the second palette, the extension might interpret the two close colors as the same color and output a chart with less than the original number of colors. However, if the user refreshes the page, this issue is resolved. This hints at a deeper issue with the coloring algorithm, if colors are too close, they might be colored to the same color. The number of colors that are recolored is also limited by the number of colors in the chosen palette.

Finally, Color Charts only works on visuals that are represented as an HTML Image. While <img> elements are prevalent across the web, new image formats such as Scalable Vector Graphics (SVG) are also used and Color Charts cannot be applied to such formats. Color Charts also does not work in PDFs.

6.2. Future Work

While Color Charts relies on researched color blind friendly palettes, a possible improvement is to generate a sort of calibration, or set up test. What this could look like is: immediately upon installation of the extension, a test similar to the Color Enhancer one in Figure 6 appears. The user sees a wide variety of potential color blind friendly palettes and they can choose which colors are preferable to them. Then Color Charts will utilize their preferences and only recolor images in these colors. This calibration accounts for the different types of color blindness and creates a much more personalized user experience. Additionally, users should ideally be able to click on colors in an image that they wish to change, and choose the color they want to change it to.

As mentioned in the previous section (Section 6.1), Color Charts faces many limitations that affect its usage across different pages and mediums. Resolving these limitations would also result in a stronger extension software.

References

[1] [Online]. Available: https://chrome.google.com/webstore/detail/color-enhancer/ipkjmjaledkapilfdigkgfmpekpfnkih

[2] [Online]. Available: https://chrome.google.com/webstore/detail/colorblind-dalton-for-goo/afcafnelafcgjinkaeohkalmfececool

[3] [Online]. Available: https://www.google.com/

[4] [Online]. Available: https://www.timvandevall.com/info/rgb-color-wheel-hex-values-printable-blank-color-wheel-templates/

[5] [Online]. Available: https://www.color-blindness.com/coblis-color-blindness-simulator/

[6] "Color vision deficiency - genetics home reference - nih." [Online]. Available: https://ghr.nlm.nih.gov/condition/color-vision-deficiency

[7] M. Kyrzywinski, "Color palettes for color blindness." Available: http://mkweb.bcgsc.ca/colorblind/

[8] M. Okabe and K. Ito. Available: https://jfly.uni-koeln.de/color/

[9] P. Tol. Available: https://personal.sron.nl/~pault/#sec:qualitative