Alex Heyman and Olivia Townsend

COG260 project final report

Color category focus locations also reflect optimal partitions of color space

Abstract

Inspired by the success of Regier, Kay, and Khetarpal (2007) with a method of predicting the boundaries of linguistic color categories, we apply a similar method to the prediction of color categories' foci. We specify a model that seeks to maximize both its predicted focus' similarity to colors in its category and its difference from colors in other categories, backed by a hypothesis that both of those properties are encouraged in foci by linguistic-developmental processes. We test this "similarity-and-difference" (SaD) model by having it reproduce the foci in the World Color Survey dataset. We find that the model's cross-linguistic accuracy at this task is imperfect, but significantly better than both random chance and an alternate model based on the prototype theory of categorization, thus lending support to our hypothesis. We briefly discuss potential limitations of our methods and further research directions.

Introduction

The World Color Survey (WCS) was a cross-cultural study of linguistic color categorization, carried out in the late 1970s. Speakers of 110 languages without written forms were interviewed about the basic color terms (e.g. “yellow” or “purple” in English) that they used. They were also shown a set of 330 colors, consisting of both shades of grey between white and black, and maximum-saturation colors with various combinations of hue and lightness. They were asked to categorize each color under one of their language’s basic color terms, and to select one or more colors in each category as the best examples, also known as the *foci*, of that category. (Kay & Cook, 2015)

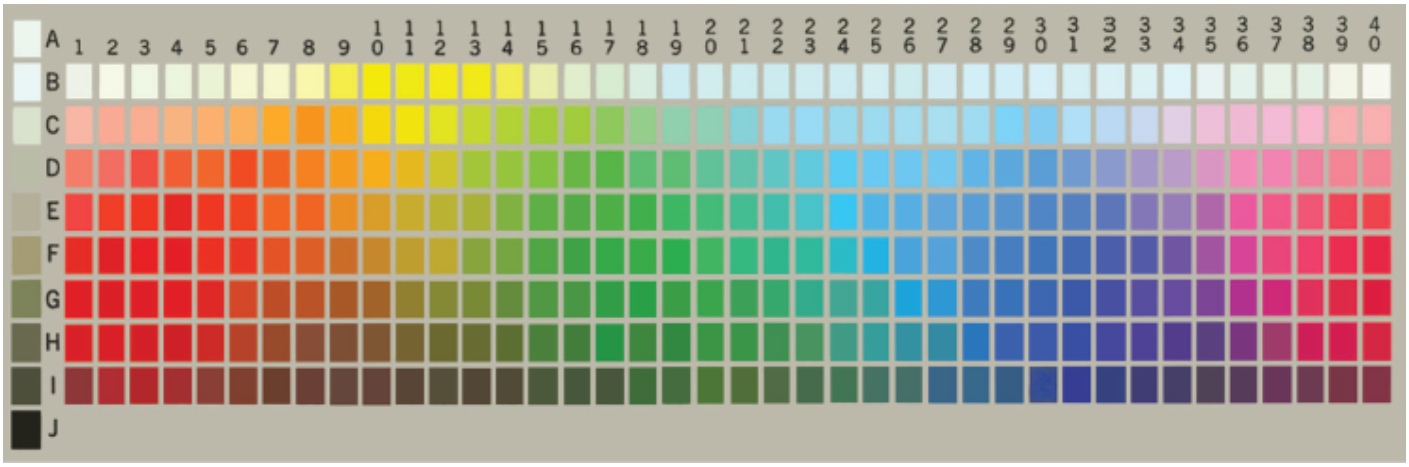


Fig. 1. Visualization of the 330 colors used in the WCS, courtesy of Regier, Kay, and Khetarpal (2007).

In their 2007 paper “Color naming reflects optimal partitions of color space”, Regier, Kay, and Khetarpal (RKK) hypothesized that when linguistic communities sort colors into categories referenced by basic color terms, they tend to sort them in ways that maximize the similarity of colors within each category, and minimize the similarity of colors in different categories. To test this hypothesis, they created a model that predicted the boundaries of color categories in a language, based only on the number of color categories in that language, by assuming the truth of the hypothesis. For any given number of categories, the model generated a categorization scheme for all of the colors used in the WCS, much like the schemes provided by the WCS interviewees. When generating a categorization scheme, the model attempted to maximize a “well-formedness” value W, defined for an entire categorization scheme as follows:

$$S_w = \sum_{\substack{(x,y):\\ \text{cat}(x) = \text{cat}(y)}} \text{sim}(x, y)$$

$$D_a = \sum_{\substack{(x,y):\\ \text{cat}(x) \neq \text{cat}(y)}} (1 - \text{sim}(x, y))$$

$$W = S_w + D_a.$$

where for any two WCS colors x and y, the “similarity” between them $\text{sim}(x, y)$ is calculated as

$$\text{sim}(x, y) = \exp(-c \times [\text{dist}(x, y)]^2)$$

where c is a constant, set to 0.001 in this case, and $\text{dist}(x, y)$ is the Euclidean distance between x and y in CIEL*a*b* (or CIELAB) space. CIELAB space is a three-dimensional space for organizing colors, and it is

suitable for use here because it is designed with the intent that equal CIELAB distances between colors correspond to equal perceptual differences (Hoffmann, retrieved 2019). Note that the outputs of the similarity function sim(x, y) range between 0 and 1, where 0 is "completely different" and 1 is "identical".

RKK then compared the outputs of their model to the categorization schemes provided by speakers of WCS-surveyed languages. In particular, the model's output was compared to the *modal maps* of WCS languages; that is, the categorization schemes where each WCS color was labeled with the color term assigned to it by *the most* speakers of the corresponding language. Each model output was strikingly similar to at least some of the modal maps with the same number of color categories. Furthermore, for almost all of the modal maps, systematic distortions in the form of hue-shifting tended to decrease their well-formedness values, indicating that their well-formedness was near-optimal. These results supported RKK's hypothesis.

Inspired by RKK's success, we hypothesize that the *foci* of languages' color categories are subject to similar linguistic-developmental constraints as the categories' boundaries; that is, foci of a category tend to be those colors which maximize both similarity to other colors in their category and difference from colors in other categories. One intuition behind this is that, for instance, what makes one particular shade of red a good representative of the entire color category "red" is not just that it is similar to other shades of red, but that it is clearly *distinct* from shades of orange, purple, white, etc. Alternative hypotheses include that the *prototype theory* of categorization applies to color categories - i.e. that color category membership is determined by similarity to a "prototype", or typical member of that category - and that foci, which in this view are the same thing as prototypes, thus tend to be those colors that maximize only similarity to other colors in their category. It may also be the case that there is no cross-linguistic relationship between a color category's foci and its boundaries in perceptual color space, and that foci are determined purely by linguistic convention.

Methods

To test our hypothesis, we created a model that predicts focus locations for a language's color categories, based on the number and boundaries of those categories. Our model takes as input a categorization scheme for all of the WCS colors, and for each category C within that scheme, it chooses as the predicted focus of C the WCS color y in C that maximizes the value $O(y)$, defined for individual color chips as follows:

$$S(y) = \sum_{x:\mathrm{cat}(x)=C} \mathrm{sim}(x, y)$$

$$D(y) = \sum_{x:\mathrm{cat}(x)\neq C} (1 - \mathrm{sim}(x, y))$$

$$O(y) = S(y) + D(y)$$

where sim(x, y) is defined the same as in RKK's model, with c still set to 0.001.

To compare the output of our model to the data from WCS, we gave the modal maps from each of the WCS languages as input to our model. Just as we thus summarized color category data across all speakers within each language, we summarized focus data across speakers by choosing as the overall *empirical focus*

of each color category *C* the WCS color in *C* that minimizes its own average distance in CIELAB space to all of the foci of *C* specified by all of the speakers (with each focus' distance weighted so the foci of *C* from a given speaker always have the same *total* weight). We defined the *accuracy* of our model on any given language as the average similarity, using RKK's color similarity formula, of that language's empirical foci to their corresponding predicted foci.

To assess the validity of our "similarity-and-difference" (SaD) model, we repeated its assessment steps with a "prototype" model, in which the predicted focus of any color category *C* is the color chip *z* in *C* that maximizes only $S(z)$; that is, the chip *z* in *C* that is closest to being *C*'s prototype. We also did the same with 20 "random" models that, for any color category, select a random WCS color in that category as its focus. We defined the average random model accuracy of any language as the average of the accuracies of all 20 random models on that language. Comparing the SaD and prototype models' accuracies to their corresponding average random model accuracies allowed us to test whether the non-random models were at least performing better than random chance.

When computing the empirical foci, we found that some of the foci specified by individual speakers were for basic color terms that did not appear in their language's modal map, and we could thus not compute empirical *or* predicted foci for these terms. For each language, however, these terms were generally rare in comparison to the terms that were represented in the modal map. Some languages also had modal map categories for which no speaker specified any foci, and for which we could thus not compute empirical foci. These focusless categories were excluded from our analysis. Again, though, they were generally rare in comparison to the modal map categories with speaker-specified foci.

Results

The average accuracy across all languages of the SaD, prototype, and random models were approximately 71%, 65%, and 49%, respectively. Below are the distributions of per-language accuracies for each of the three models on all WCS languages:

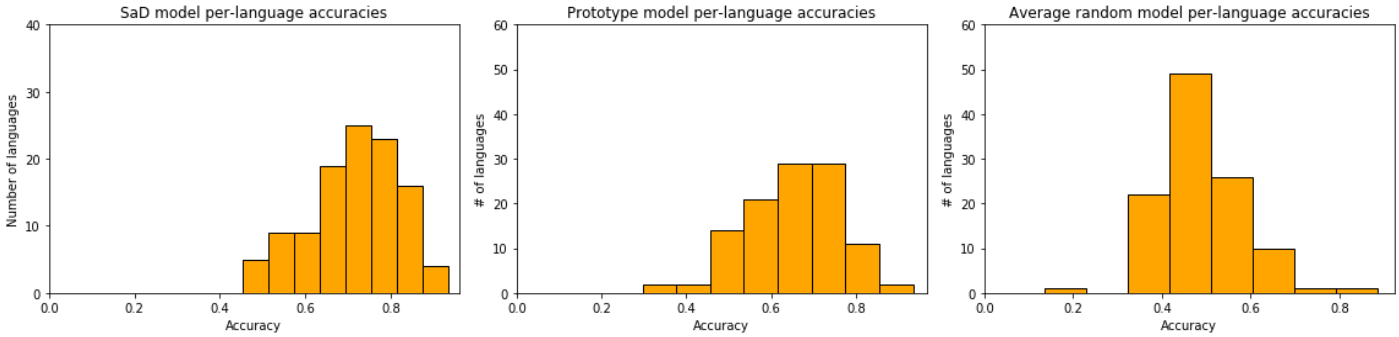


Fig. 2. Histograms of per-language accuracies for the SaD, prototype, and average random models.

All three distributions are approximately normal and thus roughly symmetric, implying that their means (that is, the cross-linguistic average accuracies given above) are appropriate summary measures. Furthermore, we can test for statistical significance in the differences between the three cross-linguistic average accuracies using a paired t-test. Such a test reveals that these differences are statistically significant with p-values well below 0.01. Thus, we can say with confidence that the SaD model's performance is meaningfully better than that of the prototype model, and both have meaningfully better performance than random chance.

We found that, while both the SaD and prototype models have higher accuracy than the average random model on 109 out of 110 languages, the SaD model has higher accuracy than the prototype model on 82 of 110 languages. This demonstrates that the SaD model's superiority in performance is fairly consistent across languages.

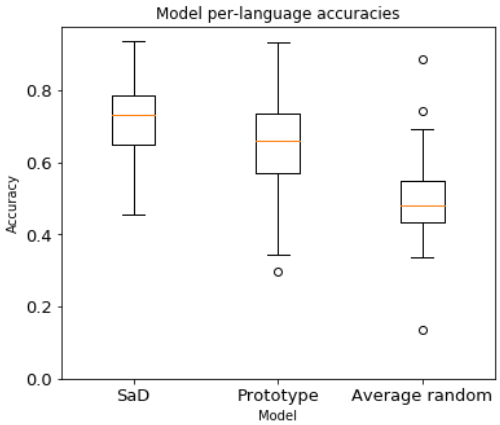


Fig. 3. Side-by-side boxplots of per-language accuracies for the SaD, prototype, and average random models.

We can observe from the boxplots above that the range and interquartile range of the SaD model's accuracy distribution are smaller than those of the prototype model's distribution, and the SaD model's distribution is the only one to contain no true outliers. This coheres with the above evidence for the SaD model's consistency in performance quality. Furthermore, our judgment of the models' accuracy distributions as roughly symmetrical based on their histograms is reinforced by the approximate symmetry of their boxplots. The least symmetrical accuracy distribution is that of the average random model, but the per-language accuracy comparisons above are perfectly sufficient to establish the utility of both the SaD and prototype models as compared to random guessing.

The SaD model was most accurate (~94%) on the language Mi'kmaq, spoken by indigenous peoples in what is now Canada and Maine (McGee, 2008). Below is a visualization of the modal maps and the foci, both empirical and predicted, for Mi'kmaq.

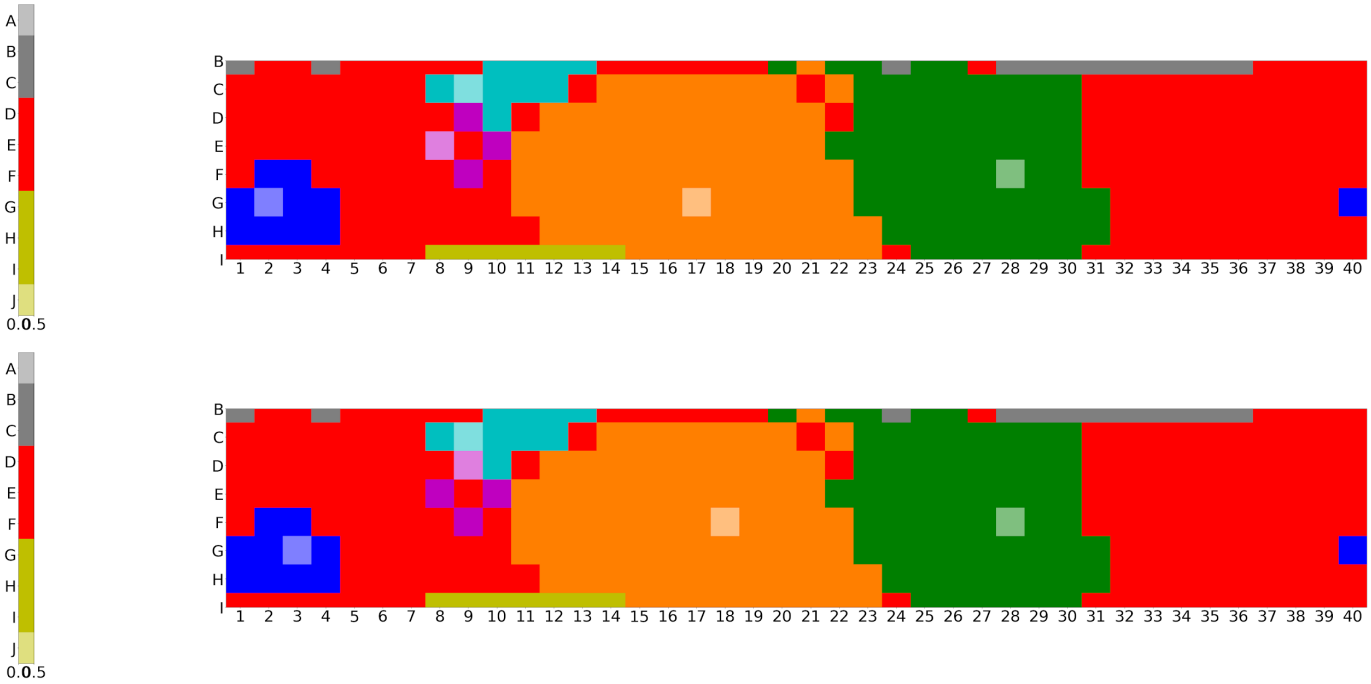
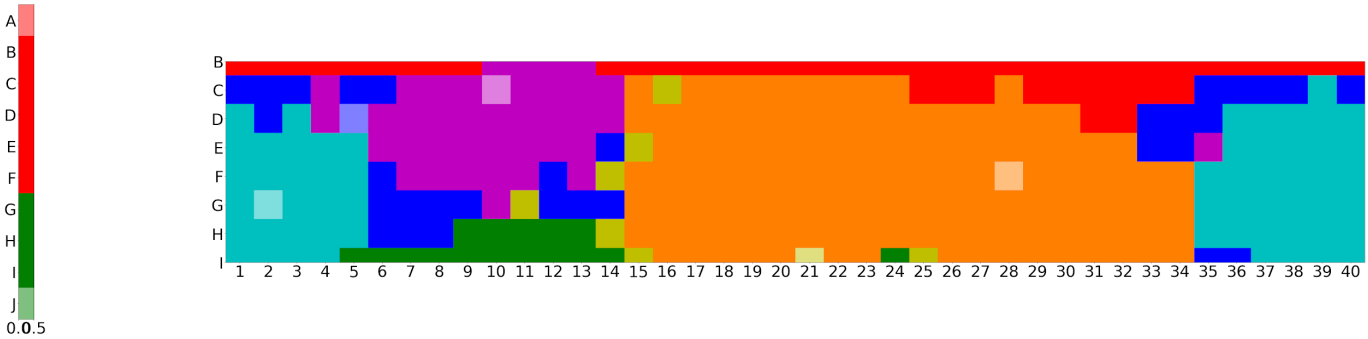


Fig. 4. Division of the WCS color space in the Mi'kmaq modal map. Category color-coding has no relationship to correspondence with English color terms. Lighter-colored areas are locations of empirical foci (top) and SaD predicted foci (bottom).

The SaD model was least accurate (~45%) on the language Arabela, a critically endangered language spoken by indigenous peoples in what is now Peru (Glottolog, retrieved 2019).



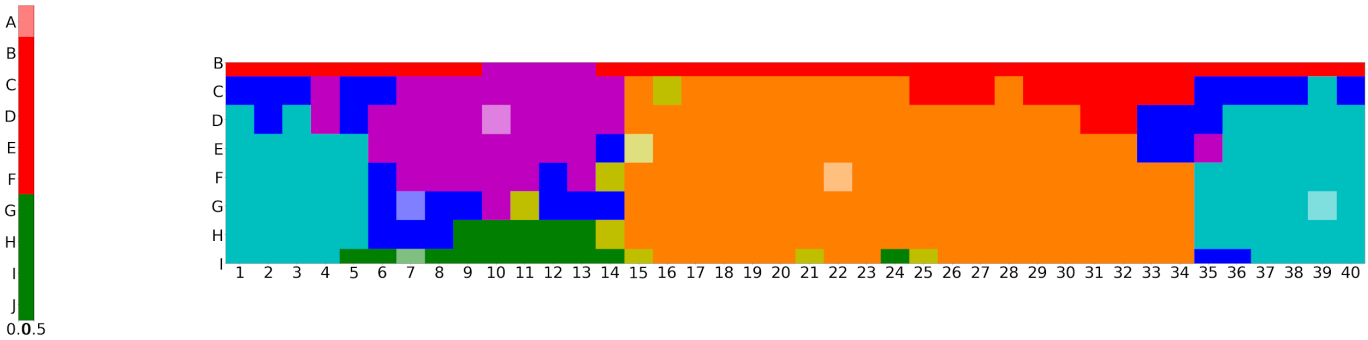


Fig. 5. Division of the WCS color space in the Arabela modal map. Lighter-colored areas are locations of empirical foci (top) and SaD predicted foci (bottom). Note the scattered distribution of the color terms that are here color-coded blue and yellow.

Conclusion

We are pleased to conclude that our results support our hypothesis. The SaD model performed consistently and statistically significantly better than both random chance and the prototype model. This suggests that the theoretical assumption behind the SaD model - that is, our hypothesis about how foci develop - is at least a real contributing factor in the determination of focus locations.

One potential further research direction is searching for features of a language that relate, correlationally and perhaps causally, to the accuracy of the SaD model on that language. One potential factor that we have observed from looking at languages' modal maps is that high-SaD-accuracy languages, more so than low-SaD-accuracy ones, tend to have color categories that each form a single, continuous, compact mass in WCS color space. Low-SaD-accuracy languages' modal maps tend to have at least one category scattered across disparate regions of the space.

The relative failure of the SaD model on these languages may be attributable to the fact that, like with the other models used in this study, the SaD model only generates one predicted focus per category, and we test the model by comparing that predicted focus to only one summarized empirical focus. In cases where a color category's speaker-specified foci are distributed such that a single point is inadequate to summarize them all, more sophisticated testing against multiple empirical foci may be required, and the models may need to be extended accordingly to generate multiple predicted foci.

References

"Arabela." Glottolog 4.1. Accessed December 8, 2019. https://glottolog.org/resource/languoid/id/arab1268.

Hoffman, G. (n.d.). CIELab Color Space. Retrieved December 8, 2019, from http://docs-hoffmann.de/cielab03022003.pdf.

Kay, P., & Cook, R. S. (2015). World Color Survey. *Encyclopedia of Color Science and Technology*, 1–8. doi: 10.1007/978-3-642-27851-8_113-10.

McGee, Harold F. "Mi'kmaq." The Canadian Encyclopedia, August 13, 2008. https://www.thecanadianencyclopedia.ca/en/article/micmac-mikmaq.

Regier, T., Kay, P., & Khetarpal, N. (2007). Color naming reflects optimal partitions of color space. *Proceedings of the National Academy of Sciences*, *104*(4), 1436–1441. doi: 10.1073/pnas.0610341104.

Appendix (Python code)

wcs_helper_functions_alt.py:

```
# Code developed and shared by:
# Vasilis Oikonomou
# Joshua Abbott
# Jessie Salas
# Alex Heyman

import numpy as np
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.pyplot as plt
import re
from random import random, shuffle
import numpy as np
from matplotlib import gridspec
import warnings
import string
warnings.filterwarnings('ignore')

def readNamingData(namingDataFilePath):
    """
    Read all of namingDataFilePath into a dictionary, and return it.  Assumes data file
follows WCS format:
    language number\tspeaker number\tchip number\tlanguage term for chip\n

    Parameters
    ----------
    namingDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted color
naming data from.


    Returns
    -------
    namingData : dictionary
      A hierarchical dictionary: namingData[languageNumber][speakerNumber][chipNumber] =
languageTerm


    Example Usage:
    --------------
    import wcsFunctions as wcs
    namingDictionary = wcs.readNamingData('./WCS-Data/term.txt')

    """
    namingData = {}  # empty dict
    fileHandler = open(namingDataFilePath,'r')

    for line in fileHandler:                          # for each line in the file
```

```
        lineElements = line.split()                    # lineElements are denoted by
white space

        # WCS format for naming data from term.txt:
        # language number\tspeaker number\tchip number\tlanguage term for chip

        languageNumber = int(lineElements[0])    # 1st element is language number, make
it an int
        speakerNumber = int(lineElements[1])           # 2nd is speaker number, make int
        chipNumber = int(lineElements[2])              # 3rd is chip number, make int
        languageTerm = lineElements[3]           # 4th is languageTermegory assignment
(term), keep string

        if not (languageNumber in namingData.keys()):
      # if this language isn't a key in the namingData dict
            namingData[languageNumber] = {}
            # then make it one, with its value an empty list
        if not (speakerNumber in namingData[languageNumber].keys()):                  #
if this speaker isn't a key in the languageNumber dict
            namingData[languageNumber][speakerNumber] = {}
      # then make it one, with its value an empty list

        namingData[languageNumber][speakerNumber][chipNumber] = languageTerm    # fill in
these empty lists to make a GIANT namingData dictionary

            # where each entry looks like this: {1: {1: {1: 'LB'}}

            # and thus namingData[1][1][1] returns string 'LB'

    fileHandler.close()                    # close file after reading it in, for neatness
    return namingData                      # return the dict


def readFociData(fociDataFilePath):
    """
    Read all of fociDataFilePath into a dictionary, and return it. Assumes data file
follows WCS format:
    language number\tspeaker number\tterm number\tterm abbreviation\tWCS chip grid
coordinate

    Paramaters
    ----------
    fociDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted color foci
data from.

    Returns
    -------
    fociData : dictionary
      A hierarchical dictionary:
fociData[languageNumber][speakerNumber][languageTerm].append(modGridCoord)
```

```
    Example Usage:
    --------------
    import wcsFunctions as wcs
    fociDictionary = wcs.readFociData('./WCS-Data/foci-exp.txt')

    """

    fociData = {} # empty dict
    fileHandler = open(fociDataFilePath,'r')
    for line in fileHandler:                    # for each line in the file
        lineElements = line.split()        # elements are denoted by white space

        # WCS format for naming data from foci-exp.txt:
        # language number\tspeaker number\tfoci number in term\tlanguage term for
chip\tWCS grid coordinate

        languageNumber = int(lineElements[0])    # 1st element is language number, make
it an int
        speakerNumber = int(lineElements[1])     # 2nd is speaker number, make int
        termNumber = int(lineElements[2])        # 3rd is term number for which foci goes
to which term
        languageTerm = lineElements[3]                 # 4th is term abbreviation
        gridcoord = lineElements[4]                    # 5th is chip grid coord

        # fix WCS bad entries - collapse A1-40 to A0 and J1-40 to J0
        if (gridcoord[0] == 'A'):
            if (int(gridcoord[1:]) > 0):
                gridcoord = "A0"

        if (gridcoord[0] == 'J'):
            if (int(gridcoord[1:]) > 0):
                gridcoord = "J0"

        modGridCoord = gridcoord[0] + ":" + gridcoord[1:]     # make it nicer for parsing
purposes later


        if not (languageNumber in fociData.keys()):     # if this language isn't a key in
the fociData dict
            fociData[languageNumber] = {}              # then make it one, with its value
an empty list
        if not (speakerNumber in fociData[languageNumber].keys()):  # if this speaker
isn't a key in the languageNumber dict
            fociData[languageNumber][speakerNumber] = {}                  # then make it
one, with its value an empty list
        if not (languageTerm in fociData[languageNumber][speakerNumber].keys()):      #
if this term isn't a key in the speakerNumber dict
            fociData[languageNumber][speakerNumber][languageTerm] = []                #
then make it one, with its value an empty list
```

```
        if not(modGridCoord in fociData[languageNumber][speakerNumber][languageTerm]): #
if they provided multiple A0 or J0 hits, only record 1
            fociData[languageNumber][speakerNumber][languageTerm].append(modGridCoord)

    fileHandler.close()
    return fociData


def readChipData(chipDataFilePath):
    """
    Read all of chipDataFilePath into two dictionaries, one maps from row/column code to
WCS chip number,
      the other maps in the other direction.  Assumes data file follows WCS format:
    chip number\tWCS grid row letter\tWCS grid column number\tWCS grid rowcol
abbreviation\n

    Parameters
    ----------
    chipDataFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted chip data
from.


    Returns
    -------
    cnum : dictionary
      cnum[row/column abbreviation] = WCS chipnumber, thus cnum maps from row/col
designation to chip number

    cname : a dictionary
      cname[WCS chipnumber] = row letter, column number (a tuple), thus cname maps from
chip number to row/col designation


    Example Usage:
    --------------
    import wcsFunctions as wcs
    cnumDictionary, cnameDictionary = wcs.readChipData('./WCS-Data/chip.txt')

    """

    cnum = {}    # empty dict to look up number given row/column designation
    cname = {}   # empty dict to look up row/column designation given number
    fileHandler = open(chipDataFilePath, 'r')    # open file for reading
    for line in fileHandler:                # for each line in the file
        lineElements = line.split()      # elements are denoted by white space
        chipnum = int(lineElements[0])   # 1st element is chip number, make it an int
        RC = lineElements[3]             # 4th is row/column designation, leave str (NB
dictionaries don't exactly reverse each other)
        letter = lineElements[1]         # 2nd is the letter (row) designation
        number = str(lineElements[2])    # 3rd is the number (column) designation, make
string so we can combine it with letter as a tuple in cname dict
```

```
        # cnum[rowcol] maps from row/column designation to chip number
        cnum[RC] = chipnum
        # cname[chipnum] maps from chip number to row/column designation (a tuple)
        cname[chipnum] = letter,number

    fileHandler.close()

    return cnum,cname            # return both dicts


def readSpeakerData(speakerFilePath):
    #lANUGUAGE[SPEAKER NUMBER]
    """
    Parameters
    ----------
    speakerFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted speaker
data from.

    Returns
    ------
    speakers : dictionary
               The keys are ints corresponding to the language IDs and the values are
lists of tuples, where
               each element of the list contains (AGE,GENDER) corresponding to the
speakers recorded for each language

    Example Usage:
    -------------
    >>> from pprint import pprint
    >>> speakerDictionary = readSpeakerData('./WCS_data_partial/spkr-lsas.txt')
    >>>  pprint(speakerDictionary)
    """
    speakers = {}                     #Initialize the dictionary
    f = open(speakerFilePath, 'r')    #Open the file containing the input data
    for line in f:                    #Iterate through each line
        content = line.split()        #split input data by whitespace
        language_ID = int(content[0]) #ID is the first element of row, cast as int
        speaker_ID = int(content[1])
        speaker_age = content[2]      #Age is the third element of row
        speaker_gender = content[3]   #Gender is the fourth element of row
        if not (language_ID in speakers.keys()):
            speakers[language_ID] = {}
        if not (speaker_ID in speakers[language_ID].keys()):
            speakers[language_ID][speaker_ID] = []
        if not((speaker_age, speaker_gender) in speakers[language_ID][speaker_ID]):
            speakers[language_ID][speaker_ID].append((speaker_age, speaker_gender))
    return speakers

def readClabData(clabFilePath):
    """
```

```
    Parameters
    ----------
    clabFilePath : string
        The path (and filename, with the extension) to read the WCS-formatted clab data
from.

    Returns
    -------
    clab : dictionary
           The keys are ints and the values are tuples (n1,n2,n3), representing the clab
coordinates

    Example Usage:
    -------------
    >>> clabDictionary = readClabData('./WCS_data_core/cnum-vhcm-lab-new.txt')
    >>> print(clabDictionary[141])
    (96.00, -.06,.06)

    """
    clab = {}                   #Initialize the dictionary
    f = open(clabFilePath,'r') #Open the file containing the input data
    for line in f:              #Iterate through each line
        content = line.split() #split input data by whitespace
        ID = int(content[0])        #ID is the first element of row, cast as int
        n1,n2,n3 = content[-3],content[-2],content[-1] #coords are the last three
elements of row
        clab[ID] = (n1,n2,n3)  #Add ID, coordinate pairs to the dictionary
    return clab


def plot_rgb_values(values):
    """
    Shows a matplotlib plot of the specified array of RGB values (given as
    3-tuples) in the positions of the WCS color chips in Munsell space.
    """
    #read in important information for reordering
    plt.rc(['ytick', 'xtick'], labelsize=50)
    cnumDictionary, cnameDictionary = readChipData('./WCS_data_core/chip.txt')
    #reorder the given values
    lst = [values[cnumDictionary['A0']-1], values[cnumDictionary['B0']-1],
       values[cnumDictionary['C0']-1], values[cnumDictionary['D0']-1],
values[cnumDictionary['E0']-1],
      values[cnumDictionary['F0']-1], values[cnumDictionary['G0']-1],
values[cnumDictionary['H0']-1],
      values[cnumDictionary['I0']-1], values[cnumDictionary['J0']-1]]
    for letter in list(string.ascii_uppercase[1:9]):
        for num in range(1, 41):
            lst.append(values[cnumDictionary[letter+str(num)]-1])
    values = np.array(lst)
    #plot
    ha = 'center'
    fig = plt.figure(figsize=(80,40))
```

```
    fig.suptitle('WCS chart', fontsize=80)
    gs = gridspec.GridSpec(2, 2, width_ratios=[1, 8], height_ratios=[1,1])
    ax1 = plt.subplot(gs[1])
    ax2 = plt.subplot(gs[0])
    core = values[10:].reshape((8, 40, 3))
    ax1.imshow(core, extent = [0, len(core[0]),len(core), 0], interpolation='none')
    labels = ["B", "C", "D", "E", "F", "G", "H", "I"]
    ax1.set_yticklabels(labels)
    ax2.imshow([[i] for i in values[:10]], extent = [0, 0.5, 0, 10],
interpolation='none')
    ax1.yaxis.set(ticks=np.arange(0.5, len(labels)), ticklabels=labels)
    ax2.yaxis.set(ticks=np.arange(0.5, len(["A"]+labels+["J"])),
ticklabels=(["A"]+labels+["J"])[::-1])
    ax1.xaxis.set(ticks=np.arange(0.5, 40), ticklabels=np.arange(1, 41))


viz_colors = [(1, 0, 0), (0, 0.5, 0), (0, 0, 1), (0.75, 0.75, 0), (0, 0.75, 0.75),
              (0.75, 0, 0.75), (1, 0.5, 0), (0.5, 0.5, 0.5), (0.5, 0.25, 0),
              (0.5, 0, 0), (0, 0, 0.5), (0.5, 0.5, 0.5)]

def generate_rgb_values(ar):
    """
    Returns a dictionary that maps each unique element in the list ar to a
    random RGB value, given as a 3-tuple. The RGB values are chosen to be easily
    distinguishable from one another, at least for reasonably small numbers of
    unique elements.
    """
    ar = list(set(ar))
    shuffle(ar)
    num_values = len(ar)

    colors = []
    while len(colors) < num_values:
        if len(colors) < len(viz_colors):
            colors.append(viz_colors[len(colors)])
        else:
            colors.append((random(), random(), random()))

    d = {}
    for i in range(num_values):
        d[ar[i]] = colors[i]
    return d

def map_array_to(ar, d):
    """
    Returns a copy of the list ar in which each element is replaced with the
    value to which it maps in the dictionary d.
    """
    return [d[i] for i in ar]
```

COG260_project_code_AlexOlivia.ipynb: