Graph Theoretic Analysis of Parsimonious Voice Leading Using Pitch Class Set Theory

Zack Fravel

C.E. Undergraduate - University of Arkansas - 2018

1 Abstract

This paper is intended to expand upon the work done by [1],[2], and [3] by providing composers, mathematicians, and other readers alike with the tools to engage in deeper problems in the field of music theory. The idea is to study the Neo-Riemannian concept of parsimonious voice leading with graph and network theory as our lens to look through. Parsimonious voice leading has to do with how a given composer might wish to arrange chord progression given N voices. The intended goal is to further quantify the relative importance of triads (or chords) given a particular scale.

This paper expands upon previous work in the field by analyzing 4-note (seventh) chords, N - TET scales, as well as establishing a framework that is straight forward to customize to any new project studying similar concepts (i.e. temperament, chord structure, voice leading parameters, etc). Along with analyzing 4-note chords, I also used NetworkX to perform novel analysis measurements on voice leading graphs that should bring new insight to this emergent field. All code used to find results is included at the end paper.

2 Background

Before getting into the graph theoretic definitions and algorithms implemented, an overview on some basic music theory is important to understanding the contributions made by this paper. I will begin the background discussion by providing some basics on pitch class set

theory as well as a discussion on equal temperament. Music theory provides many different ways for mathematicians and musicians to speak a common language, geometry for example. This paper specifically aims to focus on the potential insight that could be gleaned from graph theory and expands upon previous work.

2.1 Equal Temperament and Pitch Class Set Theory

In music, equal temperament refers to the system of tuning being implemented at a given time. Equal temperament defines the frequency interval between pairs of notes within the same ratio. What this means musically is how many notes there are in an octave before the scale wraps back around on itself. For the last few hundred years, 12-TET tuning has been the dominant system in Western music. This is more commonly referred to as the chromatic scale. The notes being C, C♯, D, E♭, E, F, F♯, G, A♭, A, B♭, B. In other parts of the world however, other tuning systems come into play. For example, Persian and Arabic music commonly implement a 24-TET or quarter-tone tuning system. Most of the previous work done in the field focuses on 12-TET, however I have outlined and provided a program that makes it easy to analyze different tuning systems.

Now onto more mathematical definitions. Pitch class set theory provides use with an adequate framework for classifying scales and denoting notes mathematically in a simple way. The chromatic scale can be represented by the ordered pitch class $C = \{0, 1, 2, 3, ..., 11\}$ since there are 12 notes in an octave. We assume octave equivalence (meaning all notes in octaves below and above will be accurately represented mathematically). We are concerned specifically with looking at major, minor, augmented, and diminished triads and sevenths given an ordered set of pitches. Triads are 3-note chords and Sevenths are 4-note chords. On all graphs, we denote scales as follows: {C Maj. C Min. C Aug. C Dim.}. So, given an ordered set $P = \{p_1, p_2, ..., p_n\}$, there exists the following chord definitions:

Chord	Major	Minor	Augmented	Diminished
Triad	$\{p_i, p_i + 4, p_i + 7\}$	$\{p_i, p_i + 3, p_i + 7\}$	$\{p_i, p_i + 4, p_i + 8\}$	$\{p_i, p_i + 3, p_i + 6\}$
Seventh	$\{p_i, p_i + 4, p_i + 7, p_i + 11\}$	$\{p_i, .., p_i + 10\}$	$\{p_i, .., p_i + 11\}$	$\{p_i, .., p_i + 9\}$

2.2 Graph Theoretic Definitions

Translating these definitions to work within a graph theoretic context is very straight forward. Again, we are concerned with the Neo-Riemannian concept of parsimonious voice leading. Parsimonious voice leading provides a defined way to assist composers in figuring out exactly how a chord progression could be played provided some notion of harmonic proximity. In our case, we are concerned with triads that are related by two common notes and one different note that is one semitone apart in the pitch set. This is slightly modified with how I set up how sevenths graphs are analyzed, with those I allow two notes to be different neighbors. This, of course, could be further expanded to any number of alternate paramters.

Graph construction is fairly simple, the major, minor, augmented, and diminished chords form our set of nodes, or vertices V. An edge exists between two vertices if and only if the intersection of $v_i \cap v_j$ contains exactly two elements and if and only if the two elements of the symmetric difference $v_i \Delta v_j$ are neighbors in P, that is they are a semitone apart.

I have provided three sample graphs generated by my code to illustrate the definitions described above. Major chords are red, minor chords are green, augmented are blue, and diminished are tan. The chromatic scale, as described above, is the set $P = \{0, 1, 2, ..., 11\}$. The C Major scale (the most common scale in western music) is the set $P = \{0, 2, 4, 5, 7, 9, 11\}$. It can be seen that the simplicity of this graph makes it clear why it is such a universal scale for musicians to learn on. It's worth noting the C Minor scale exhibits the same structure. The final graph is the set $P = \{0, 2, 4, 5, 7, 8, 9, 11\}$, which is the C Major scale with one added note.

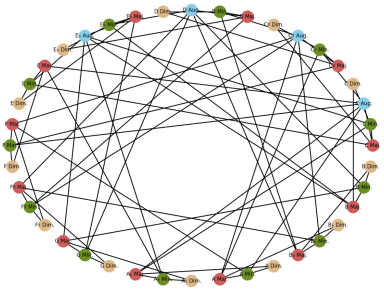

Figure 1: Chromatic Scale

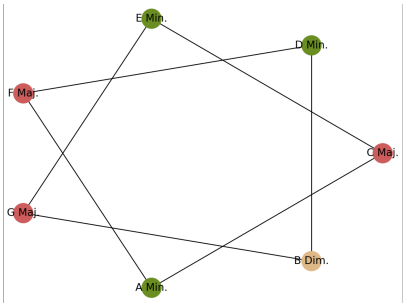

Figure 2: C Major Scale

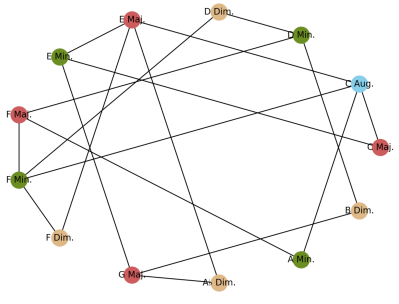

Figure 3: C Major Scale with an Added Note

3 Problem

The problem I aim to solve is to provide a way to analyze graphs for not only triads but to expand the previous research to allow for seventh chords and temperaments other than 12-TET. The underlying goal is to encourage more researchers to dig into the possible insights to be found by applying concepts from complex network theory to smaller graphs. Even though a lot of these tools were build for large graphs, the notions of connectedness and centrality provide a composer with very useful information about chords and how they might efficiently fit into a progression given a set of pitches. The mathematical definitions of all analysis measurements, along with why they might be musically relevant, will be found in the next section.

4 Implementation

4.1 Analysis Definitions and Equations

In the following subsections I will provide a detailed description of both mathematically and musically relevant information about each of the measurements I chose for analyzing voice leading graphs. A lot of these measurements can be found in [1], however I did not find discussion of measurements found in sections 4.1.5 to 4.1.7 in any other sources.

4.1.1 Degree Centrality

Degree centrality is defined simply as the fraction of nodes in the graph a given node is neighbors with. It is found by taking the degree of a node and dividing by the maximum possible degree in the graph.

$$\frac{deg(V)}{n-1}$$

Musically, this translates to the immediate amount of musical choice available from the current triad. While this simple measurement is useful, more complex algorithms meant for complex networks might help provide more detailed information.

4.1.2 Closeness Centrality

Closeness centrality is the reciprocal of the sum of the shortest path distances from u to all n - 1 other nodes. This measure is different in that it takes shortest paths to all other nodes in the graph into account in the calculation.

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(u,v)}$$

This measure provides a more accurate representation of harmonic choice at a given chord using local information.

4.1.3 Betweenness Centrality

Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v. $\sigma(s,t)$ is the number of shortest (s, t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s, t.

$$C_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

Musically, betweenness centrality is an interesting measure in that it provides information about how often a chord appears on paths in the graph. This means that a chord with a lower betweenness centrality might be more likely to begin or end a chord progression rather than being in the middle of one.

4.1.4 Katz Centrality

Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. Eigenvector centrality is the measure of "influence" a node has on a network. A is the adjacency matrix of the graph G with eigenvalues λ.

$$x_i = \alpha \sum_j A_{ij} x_j + \beta$$

Musically, Katz centrality is different from degree centrality in that it measures harmonic choice among ALL chord progressions given an initial triad. In network theory, this is typically the measure of how influential friends are.

4.1.5 Load Centrality

Load centrality is another measure that measures the fraction of all shortest paths that pass through a given node. This measure's algorithm is given by [6] and is slightly different than betweenness centrality. As far as I know, this is a novel measurement of voice leading graphs. Musically, this would give similar information to a composer to betweenness centrality.

4.1.6 Degree Vitality

Closeness vitality of a node is the change in the sum of distances between all node pairs when excluding that node. This is another measurement I haven't found in any other sources, but think would be another useful measurement to consider. Musically, this seems to give a sort of inverse measurement of betweenness where it's showing how other chord progressions could potentially be effected by removing that triad. It could be seen as another measure of importance.

4.2 Seventh Chords and N-TET Temperament

First I'll describe the process of implementing sevenths graphs. Most of the code is the same for analyzing triads, all that was really needed was to add the logic for adding 4-note chords as described in the background (this ammounted to adding an extra conditional to the node creation process) and modifying the edge creation portion of code. In my code, I still hold that $v_i \Delta v_j$ contains two elements, this allows two notes to be different in each pair of nodes, however they still are only 1 semitone apart. Instead of only comparing two values, we're comparing four values in the symmetric difference so the logic had to be slightly modified to account for that. Figure 4 shows the graph for $P = \{0, 2, 4, 5, 7, 8, 9, 10, 11\}$.

The second way in which I expanded on previous work in the field was creating a framework for analyzing different tuning systems. I designed the program to be very simple to change what tuning system the user wishes to implement. At the top of the code, there is a dictionary called "temperament" where the user can define a new tuning system. No other changes need to be made to analyze different tunings. Figure 5 shows the graph for the full 24-TET scale.

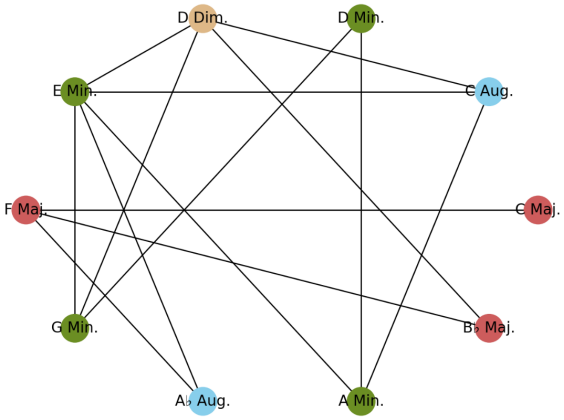

Figure 4: 4-note Chords with Set $\{0, 2, 4, 5, 7, 8, 9, 10, 11\}$

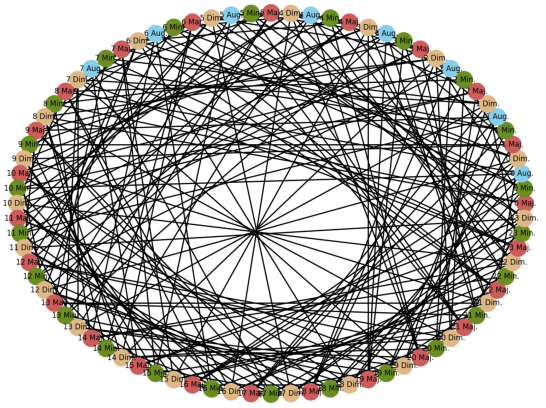

Figure 5: 24-TET Chromatic Scale

5 Results

The following section outlines the results found using the graph definitions and algorithms described in the previous section. To reiterate, I modified the original graph definitions from [1] to handle seventh (4-note) chords, along with a framework for working in different equal temperaments other than 12-TET.

5.1 Seventh Chords

Below are the results of comparing triad graphs with seventh graphs using the following pitch set $P = \{0, 2, 4, 5, 7, 8, 9, 10, 11\}$

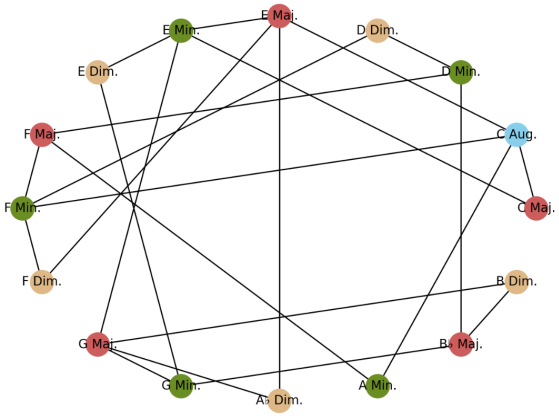

Figure 6: Triads Chords with Set $\{0, 2, 4, 5, 7, 8, 9, 10, 11\}$

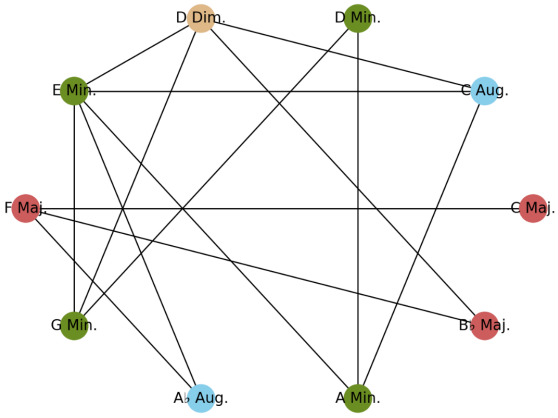

Figure 7: Seventhes Chords with Set $\{0, 2, 4, 5, 7, 8, 9, 10, 11\}$

Triad	Degree Centrality	Closeness Cent.	Betweenness Cent.	Katz Cent.	Load	Vitality
C Maj.	0.133	0.394	0.047	0.231	0.051	38.0
C Aug.	0.266	0.428	0.193	0.277	0.190	16.0
D Min.	0.2	0.384	0.179	0.248	0.179	1.0
D Dim.	0.133	0.365	0.041	0.228	0.043	41.0
E Maj.	0.266	0.441	0.209	0.277	0.205	14.0
E Min.	0.266	0.428	0.193	0.277	0.190	16.0
E Dim.	0.133	0.365	0.031	0.228	0.033	41.0
F Maj.	0.2	0.384	0.105	0.251	0.104	33.0
F Min.	0.266	0.405	0.165	0.274	0.162	20.0
F Dim.	0.133	0.384	0.044	0.231	0.047	39.0
G Maj.	0.266	0.405	0.165	0.274	0.162	20.0
G Min.	0.2	0.384	0.105	0.251	0.104	33.0
A♭ Dim.	0.133	0.384	0.044	0.231	0.047	39.0
A Min.	0.133	0.365	0.031	0.228	0.033	41.0
B♭ Maj.	0.2	0.384	0.179	0.248	0.179	1.0
B Dim.	0.133	0.365	0.041	0.228	0.043	41.0

Seventh	Degree Centrality	Closeness Cent.	Betweenness Cent.	Katz Cent.	Load	Vitality
C Maj.	0.111	0.321	0.0	0.253	0.0	28.0
C Aug.	0.333	0.5	0.027	0.328	0.027	18.0
D Min.	0.222	0.391	0.013	0.287	0.013	23.0
D Dim.	0.444	0.6	0.226	0.355	0.229	7.0
E Min.	0.555	0.642	0.342	0.384	0.340	2.0
F Maj.	0.333	0.45	0.25	0.305	0.25	∞
G Min.	0.333	0.5	0.120	0.325	0.118	16.0
A♭ Aug.	0.222	0.529	0.203	0.291	0.201	13.0
A Min.	0.333	0.473	0.074	0.322	0.076	18.0
B♭ Maj.	0.222	0.5	0.129	0.288	0.131	16.0

It can be seen that the constraints, even with the ability to change two notes, is a bit more limited when analyzing seventh chords.

5.2 Other Temperaments

Below are measurements using 24-TET of the pitch class $P = \{0, 1, 4, 5, 6, 7, 13, 14, 16, 21, 23\}$.

24-TET	Degree Centrality	Closeness Cent.	Betweenness Cent.	Katz Cent.	Load	Vitality
0 Min.	0.166	0.166	0.0	0.332	0.0	NaN
1 Dim.	0.333	0.444	0.0	0.382	0.0	NaN
5 Aug.	0.333	0.444	0.033	0.378	0.033	NaN
7 Min.	0.5	0.533	0.1	0.416	0.1	NaN
16 Maj.	0.166	0.166	0.0	0.332	0.0	NaN
23 Maj.	0.5	0.533	0.1	0.416	0.1	NaN
23 Min.	0.333	0.444	0.033	0.378	0.033	NaN

Vitality measurements didn't seem to calculate correctly on 24-TET graphs.

6 Conclusions

In summary, this paper expands upon previous work in the field of musical graph theory by expanding the horizon of parsimonious voice leading graph analysis to include N-TET tuning systems and seventh chords. Hopefully this paper provides an adequate outline for other new ways to expand upon the work done by [1], [2], and [3]. The work being done is still very rudimentary and there are plenty of questions to be asked in regards to the relationship between graph theory and music. All we are looking at here are chord progressions, but other researchers focuses on melody formation or rhythm formation I believe could benefit by trying their hand at applying graph theoretic concepts to their research analysis.

7 References

[1] Susannah Wixey and Rob Sturman. A Graph-Theoretic Approach to Efficient Voice-Leading. School of Mathematics, University of Leeds, Leeds LS2 9JT, UK. 2016.

[2] Azer Akhmedov and Michael Winter. Chordal and Timbral Morphologies Using Hamiltonian Cycles. Journal of Mathematics and Music, 8(1):1–24, 2014.

[3] Adrian Walton. A Graph Theoretic Approach to Tonal Modulation. Journal of Mathematics and Music, 4(1):45–56, 2010.

[4] Richard Cohn. Introduction to Neo-Riemannian theory: a survey and a historical perspective. Journal of Music Theory, pages 167–180, 1998.

[5] Dave Carlton. Analysis of Chord Patterns of 1300 Popular Songs. http://www.hooktheory.com/blog/i-analyzed-the-chords-of-1300-popular-songs-for-patterns-this-is-what-i-found. 2012.

[6] Mark E. J. Newman: Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. Physical Review E 64, 016132, 2001.

8 Code

```
# ------------------------------------
# Zack Fravel
# 010646947
# CSCE 5014 Applications of graph Theory
# ------------------------------------
# Triads.py
# ------------------------------------
# Graph Theoretic Voice Leading Measure
# Using NetworkX and MatPlotLib.
# ------------------------------------

import networkx as nx
import matplotlib.pyplot as plt

# ---------------------------
# --- Graph Construction ----
# ---------------------------
#
# Define dictionary of chromatic scale (12-tone)
# This is where a user can modify the program to define
# any other n-tone temperament set or scale to be analyzed.
# In short, this tells you how many unique steps there are
# in a single octave.
#
# 12-Tone Equal Temperament is the most common system used in
# Western music.
```

```
#
# If user doesn't want to use 12-TET, just specify a dictionary
# with values that are numbers corresponding to the index.

chromatic = {0: 'C',
             1: ' C  ',
             2: 'D',
             3: ' E  ',
             4: 'E',
             5: 'F',
             6: ' F  ',
             7: 'G',
             8: ' A  ',
             9: 'A',
             10: ' B  ',
             11: 'B'}

# Define set of pitches to consider here

pitches = [0, 2, 4, 5, 7, 8, 9, 10, 11]

# Initialize data structures

major = []          # Major Chords
minor = []          # Minor Chords
augmented = []      # Augmented Chords
diminished = []     # Diminished Chords
triads = []         # Triads
colorMap = []       # Color map to designate chord type
g = nx.Graph()      # Initialize Graph

# Find Triads and add nodes to graph
# Triads in list and graph nodes are added in the same order.
# This makes comparisons between triads with iterations
# simple through direct mapping.
#
# Nodes are assigned a 'chord' attribute which is a set
# containing the three notes in the triad. The set class
# makes comparisons between triads easier.

# 3 - Note Chords
for notes in pitches:

    # Major (Red)
    if (notes + 4) % 12 in pitches:
        if (notes + 7) % 12 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 4) % 12, (notes + 7) % 12]) in major:
                pass
            # If not, add triad for analysis
            else:
                major.append([notes, (notes + 4) % 12, (notes + 7) % 12])
                triads.append([notes, (notes + 4) % 12, (notes + 7) % 12])
                g.add_node(chromatic[notes] + ' Maj.', notes={notes, (notes + 4) % 12, (notes + 7) % 12}, feel='happy')
                colorMap.append('indianred')

    # Minor (Green)
    if (notes + 3) % 12 in pitches:
        if (notes + 7) % 12 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 3) % 12, (notes + 7) % 12]) in minor:
                pass
            # If not, add triad for analysis
            else:
                minor.append([notes, (notes + 3) % 12, (notes + 7) % 12])
```

```
                triads.append([notes, (notes + 3) % 12, (notes + 7) % 12])
                g.add_node(chromatic[notes] + '_Min.', notes={notes, (notes + 3) % 12, (notes + 7) % 12}, feel='sad')
                colorMap.append('olivedrab')

    # Augmented (Blue)
    if (notes + 4) % 12 in pitches:
        if (notes + 8) % 12 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 4) % 12, (notes + 8) % 12]) in augmented:
                pass
            # If not, add triad for analysis
            else:
                augmented.append([notes, (notes + 4) % 12, (notes + 8) % 12])
                triads.append([notes, (notes + 4) % 12, (notes + 8) % 12])
                g.add_node(chromatic[notes] + '_Aug.', notes={notes, (notes + 4) % 12, (notes + 8) % 12}, feel='happy')
                colorMap.append('skyblue')

    # Diminished (Tan)
    if (notes + 3) % 12 in pitches:
        if (notes + 6) % 12 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 3) % 12, (notes + 6) % 12]) in diminished:
                pass
            # If not, add triad for analysis
            else:
                diminished.append([notes, (notes + 3) % 12, (notes + 6) % 12])
                triads.append([notes, (notes + 3) % 12, (notes + 6) % 12])
                g.add_node(chromatic[notes] + '_Dim.', notes={notes, (notes + 3) % 12, (notes + 6) % 12}, feel='sad')
                colorMap.append('burlywood')

# Print scale to console
print("\nChromatic_Scale""\n")
for num, note in chromatic.items():
    print(num, note)

# Print major triads to console
print("\nMajor\n")
for elements in major:
    print(chromatic[elements[0]], elements)

# Print minor triads to console
print("\nMinor\n")
for elements in minor:
    print(chromatic[elements[0]], elements)

# Print augmented triads to console
print("\nAugmented\n")
for elements in augmented:
    print(chromatic[elements[0]], elements)

# Print diminished triads to console
print("\nDiminished\n")
for elements in diminished:
    print(chromatic[elements[0]], elements)

# Print total number of triads to console
print("\n\nTotal_Triads\n")
print(triads.__len__())

print('\n')

# Add edges between nodes.
# The relationship in chord structure between two nodes
# that determines edge creation can be modified here.
for u in g.nodes():
```

```
    # Compare nodes with all other triads.
    for v in g.nodes():
        # Add edge when the intersection of u and v shares two elements
        # and the two elements of the symmetric difference of u and v
        # are neighbors in the ordered pitch class set.
        #
        # Check here if intersection has 2 elements
        if len(g.node()[u]['notes'] & g.node()[v]['notes']) == 2:
            # Assign both values of symmetric difference to variables.
            # Using min and max makes index comparisons with pitch set simple.
            a = min(g.node()[u]['notes'] ^ g.node()[v]['notes'])
            b = max(g.node()[u]['notes'] ^ g.node()[v]['notes'])
            # Add edge if symmetric difference is 1 semitone apart (neighbors in 'pitches')
            if pitches.index(a) + 1 == pitches.index(b):
                g.add_edge(u, v)
            # Pitches on opposite ends of the list are considered neighbors
            # since scales wrap around on each other on the octave jump.
            elif pitches.index(a) == 0 and pitches.index(b) == len(pitches) - 1:
                g.add_edge(u, v)

# ------------------------------
# ------- Tone Analysis -------
# ------------------------------
#
# Measure properties of popular scales
print("Connected?: ", nx.is_connected(g), '\n')
for nodes in g.node():
    print(nodes)
    print()
    print("   Degree                    : ", nx.degree(g, nodes))
    print("   Degree Centrality         : ", nx.degree_centrality(g)[nodes])
    print("   Closeness Centrality      : ", nx.closeness_centrality(g, nodes))
    print("   Betweenness Centrality    : ", nx.betweenness_centrality(g)[nodes])
    print("   Katz Centrality           : ", nx.katz_centrality(g)[nodes])
    print("   Load Centrality           : ", nx.load_centrality(g)[nodes])
    print("   Closeness Vitality        : ", nx.closeness_vitality(g, nodes))
    print('\n')

# ------------------------------
# ------ VISUALIZATION --------
# ------------------------------
#
# Create a visualization of user's
# graph of tones. Chords are colored
# coded using the following mapping:
#       Major - Red
#       Minor - Green
#       Aug.  - Blue
#       Dim.  - Tan
#
# Set node, edge, and font size (scale) of
# plot depending on the number of nodes.

if len(g.nodes()) < 12:
    nodeSize = 500
    fontSize = 12
    edgeWidth = 1
elif len(g.nodes()) < 24:
    nodeSize = 350
    fontSize = 10
    edgeWidth = 1
elif len(g.nodes()) < 36:
    nodeSize = 250
    fontSize = 8
    edgeWidth = 1
```

```
else:
    nodeSize = 180
    fontSize = 6
    edgeWidth = 1

# Set resolution of the plot
plt.figure(dpi=200)

# Draw graph
# Shapes available: circular, random, spectral, spring, or shell
nx.draw_circular(g, node_color=colorMap, node_size=nodeSize, width=edgeWidth,
                 with_labels=True, font_size=fontSize)

# Show plot
plt.show()
```

```
# ----------------------------------------
# Zack Fravel
# 010646947
# CSCE 5014 Applications of graph Theory
# ----------------------------------------
# Sevenths.py
# ----------------------------------------
# Graph Theoretic Voice Leading Measure
# Using NetworkX and MatPlotLib.
# ----------------------------------------

import networkx as nx
import matplotlib.pyplot as plt

# ------------------------------
# --- Graph Construction -----
# ------------------------------
#
# Define dictionary of chromatic scale (12-tone)
# This is where a user can modify the program to define
# any other n-tone temperament set or scale to be analyzed.
# In short, this tells you how many unique steps there are
# in a single octave.
#
# 12-Tone Equal Temperament is the most common system used in
# Western music.
#
# If user doesn't want to use 12-TET, just specify a dictionary
# with values that are numbers corresponding to the index.

chromatic = {0: 'C',
             1: ' C  ',
             2: 'D',
             3: ' E  ',
             4: 'E',
             5: 'F',
             6: ' F  ',
             7: 'G',
             8: ' A  ',
             9: 'A',
             10: ' B  ',
             11: 'B'}

# Define set of pitches to consider here

pitches = [0, 2, 4, 5, 7, 8, 9, 10, 11]

# Initialize data structures

major = []          # Major Chords
minor = []          # Minor Chords
augmented = []      # Augmented Chords
diminished = []     # Diminished Chords
sevenths = []       # Seventh Chords
colorMap = []       # Color map to designate chord type
g = nx.Graph()      # Initialize Graph

# Find sevenths and add nodes to graph
# sevenths in list and graph nodes are added in the same order.
# This makes comparisons between sevenths with iterations
# simple through direct mapping.
#
# Nodes are assigned a 'chord' attribute which is a set
# containing the three notes in the triad. The set class
```

```
# makes comparisons between sevenths easier.

# 3 - Note Chords
for notes in pitches:

    # Major (Red)
    if (notes + 4) % 12 in pitches:
        if (notes + 7) % 12 in pitches:
            if (notes + 11) % 12 in pitches:
                # Check if variation of triad is already in list
                if sorted([notes, (notes + 4) % 12, (notes + 7) % 12, (notes + 11) % 12]) in major:
                    pass
                # If not, add triad for analysis
                else:
                    major.append([notes, (notes + 4) % 12, (notes + 7) % 12, (notes + 11) % 12])
                    sevenths.append([notes, (notes + 4) % 12, (notes + 7) % 12, (notes + 11) % 12])
                    g.add_node(chromatic[notes] + ' Maj.', notes={notes, (notes + 4) % 12, (notes + 7) % 12,
                    (notes + 11) % 12}, feel='happy')
                    colorMap.append('indianred')

    # Minor (Green)
    if (notes + 3) % 12 in pitches:
        if (notes + 7) % 12 in pitches:
            if (notes + 10) % 12 in pitches:
                # Check if variation of triad is already in list
                if sorted([notes, (notes + 3) % 12, (notes + 7) % 12, (notes + 10) % 12]) in minor:
                    pass
                # If not, add triad for analysis
                else:
                    minor.append([notes, (notes + 3) % 12, (notes + 7) % 12, (notes + 10) % 12])
                    sevenths.append([notes, (notes + 3) % 12, (notes + 7) % 12, (notes + 10) % 12])
                    g.add_node(chromatic[notes] + ' Min.', notes={notes, (notes + 3) % 12, (notes + 7) % 12,
                    (notes + 10) % 12}, feel='sad')
                    colorMap.append('olivedrab')

    # Augmented (Blue)
    if (notes + 4) % 12 in pitches:
        if (notes + 8) % 12 in pitches:
            if (notes + 11) % 12 in pitches:
                # Check if variation of triad is already in list
                if sorted([notes, (notes + 4) % 12, (notes + 8) % 12, (notes + 11) % 12]) in augmented:
                    pass
                # If not, add triad for analysis
                else:
                    augmented.append([notes, (notes + 4) % 12, (notes + 8) % 12, (notes + 11) % 12])
                    sevenths.append([notes, (notes + 4) % 12, (notes + 8) % 12, (notes + 11) % 12])
                    g.add_node(chromatic[notes] + ' Aug.', notes={notes, (notes + 4) % 12, (notes + 8) % 12,
                    (notes + 11) % 12}, feel='happy')
                    colorMap.append('skyblue')

    # Diminished (Tan)
    if (notes + 3) % 12 in pitches:
        if (notes + 6) % 12 in pitches:
            if (notes + 9) % 12 in pitches:
                # Check if variation of triad is already in list
                if sorted([notes, (notes + 3) % 12, (notes + 6) % 12, (notes + 9) % 12]) in diminished:
                    pass
                # If not, add triad for analysis
                else:
                    diminished.append([notes, (notes + 3) % 12, (notes + 6) % 12, (notes + 9) % 12])
                    sevenths.append([notes, (notes + 3) % 12, (notes + 6) % 12, (notes + 9) % 12])
                    g.add_node(chromatic[notes] + ' Dim.', notes={notes, (notes + 3) % 12, (notes + 6) % 12,
                    (notes + 9) % 12}, feel='sad')
                    colorMap.append('burlywood')
```

```
# Print scale to console
print("\nChromatic Scale""\n")
for num, note in chromatic.items():
    print(num, note)

# Print major sevenths to console
print("\nMajor\n")
for elements in major:
    print(chromatic[elements[0]], elements)

# Print minor sevenths to console
print("\nMinor\n")
for elements in minor:
    print(chromatic[elements[0]], elements)

# Print augmented sevenths to console
print("\nAugmented\n")
for elements in augmented:
    print(chromatic[elements[0]], elements)

# Print diminished sevenths to console
print("\nDiminished\n")
for elements in diminished:
    print(chromatic[elements[0]], elements)

# Print total number of sevenths to console
print("\n\nTotal sevenths\n")
print(sevenths.__len__())

print('\n')

# Add edges between nodes.
# The relationship in chord structure between two nodes
# that determines edge creation can be modified here.
for u in g.nodes():
    # Compare nodes with all other sevenths.
    for v in g.nodes():
        # Add edge when the intersection of u and v shares two elements
        # and the two elements of the symmetric difference of u and v
        # are neighbors in the ordered pitch class set.
        #
        # Check here if intersection has 2 elements
        if len(g.node()[u]['notes'] & g.node()[v]['notes']) == 2:
            # Assign both values of symmetric difference to variables.
            symmetric_difference = list((g.node()[u]['notes'] ^ g.node()[v]['notes']))
            a = symmetric_difference[0]
            b = symmetric_difference[1]
            c = symmetric_difference[2]
            d = symmetric_difference[3]
            # Add edge if symmetric difference is 1 semitone apart (neighbors in 'pitches')
            if (pitches.index(a) + 1 == pitches.index(b)) or (pitches.index(c) + 1 == pitches.index(d)):
                g.add_edge(u, v)
            # Pitches on opposite ends of the list are considered neighbors
            # since scales wrap around on each other on the octave jump.
            elif (pitches.index(a) == 0 and pitches.index(b) == len(pitches) - 1) or (pitches.index(c) == 0 and pitches.index
                g.add_edge(u, v)

# ----------------------------
# ------ Tone Analysis -------
# ----------------------------
#
# Measure properties of popular scales
print("Connected?: ", nx.is_connected(g), '\n')
for nodes in g.node():
    print(nodes)
```

```
        print()
        print("   Degree                    :", nx.degree(g, nodes))
        print("   Degree Centrality         :", nx.degree_centrality(g)[nodes])
        print("   Closeness Centrality      :", nx.closeness_centrality(g, nodes))
        print("   Betweenness Centrality    :", nx.betweenness_centrality(g)[nodes])
        print("   Katz Centrality           :", nx.katz_centrality(g)[nodes])
        print("   Load Centrality           :", nx.load_centrality(g)[nodes])
        print("   Closeness Vitality        :", nx.closeness_vitality(g, nodes))
        print('\n')

# ------------------------------
# ----- VISUALIZATION ----------
# ------------------------------
#
# Create a visualization of user's
# graph of tones. Chords are colored
# coded using the following mapping:
#       Major - Red
#       Minor - Green
#       Aug.  - Blue
#       Dim.  - Tan
#
# Set node, edge, and font size (scale) of
# plot depending on the number of nodes.

if len(g.nodes()) < 12:
    nodeSize = 500
    fontSize = 12
    edgeWidth = 1
elif len(g.nodes()) < 24:
    nodeSize = 350
    fontSize = 10
    edgeWidth = 1
elif len(g.nodes()) < 36:
    nodeSize = 250
    fontSize = 8
    edgeWidth = 1
else:
    nodeSize = 180
    fontSize = 6
    edgeWidth = 1

# Set resolution of the plot
plt.figure(dpi=200)

# Draw graph
# Shapes available: circular, random, spectral, spring, or shell
nx.draw_circular(g, node_color=colorMap, node_size=nodeSize, width=edgeWidth,
                 with_labels=True, font_size=fontSize)

# Show plot
plt.show()
```

```
# ----------------------------------------
# Zack Fravel
# 010646947
# CSCE 5014 Applications of graph Theory
# ----------------------------------------
# Temperament.py
# ----------------------------------------
# Graph Theoretic Voice Leading Measure
# Using NetworkX and MatPlotLib.
# ----------------------------------------

import networkx as nx
import matplotlib.pyplot as plt

# ----------------------------
# --- Graph Construction ----
# ----------------------------
#
# Define dictionary of temperament scale (24-tone)
# This is where a user can modify the program to define
# any other n-tone temperament set or scale to be analyzed.
# In short, this tells you how many unique steps there are
# in a single octave.
#
# 24-Tone Equal Temperament is the most common system used in
# Western music.
#
# If user doesn't want to use 24-TET, just specify a dictionary
# with values that are numbers corresponding to the index.

temperament = {0: '0',
               1: '1',
               2: '2',
               3: '3',
               4: '4',
               5: '5',
               6: '6',
               7: '7',
               8: '8',
               9: '9',
               10: '10',
               11: '11',
               12: '12',
               13: '13',
               14: '14',
               15: '15',
               16: '16',
               17: '17',
               18: '18',
               19: '19',
               20: '20',
               21: '21',
               22: '22',
               23: '23'}

# Define set of pitches to consider here

pitches = [0, 1, 4, 5, 6, 7, 13, 14, 16, 21, 23]

# Initialize data structures

major = []          # Major Chords
minor = []          # Minor Chords
augmented = []      # Augmented Chords
```

```
diminished = []      # Diminished Chords
triads = []          # Triads
colorMap = []        # Color map to designate chord type
g = nx.Graph()       # Initialize Graph

# Find Triads and add nodes to graph
# Triads in list and graph nodes are added in the same order.
# This makes comparisons between triads with iterations
# simple through direct mapping.
#
# Nodes are assigned a 'chord' attribute which is a set
# containing the three notes in the triad. The set class
# makes comparisons between triads easier.

# 3 - Note Chords
for notes in pitches:

    # Major (Red)
    if (notes + 8) % 24 in pitches:
        if (notes + 14) % 24 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 8) % 24, (notes + 14) % 24]) in major:
                pass
            # If not, add triad for analysis
            else:
                major.append([notes, (notes + 8) % 24, (notes + 14) % 24])
                triads.append([notes, (notes + 8) % 24, (notes + 14) % 24])
                g.add_node(temperament[notes] + ' Maj.', notes={notes, (notes + 8) % 24, (notes + 14) % 24}, feel='happy')
                colorMap.append('indianred')

    # Minor (Green)
    if (notes + 6) % 24 in pitches:
        if (notes + 14) % 24 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 6) % 24, (notes + 14) % 24]) in minor:
                pass
            # If not, add triad for analysis
            else:
                minor.append([notes, (notes + 6) % 24, (notes + 14) % 24])
                triads.append([notes, (notes + 6) % 24, (notes + 14) % 24])
                g.add_node(temperament[notes] + ' Min.', notes={notes, (notes + 6) % 24, (notes + 14) % 24}, feel='sad')
                colorMap.append('olivedrab')

    # Augmented (Blue)
    if (notes + 8) % 24 in pitches:
        if (notes + 16) % 24 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 8) % 24, (notes + 16) % 24]) in augmented:
                pass
            # If not, add triad for analysis
            else:
                augmented.append([notes, (notes + 8) % 24, (notes + 16) % 24])
                triads.append([notes, (notes + 8) % 24, (notes + 16) % 24])
                g.add_node(temperament[notes] + ' Aug.', notes={notes, (notes + 8) % 24, (notes + 16) % 24}, feel='happy')
                colorMap.append('skyblue')

    # Diminished (Tan)
    if (notes + 6) % 24 in pitches:
        if (notes + 12) % 24 in pitches:
            # Check if variation of triad is already in list
            if sorted([notes, (notes + 6) % 24, (notes + 12) % 24]) in diminished:
                pass
            # If not, add triad for analysis
            else:
                diminished.append([notes, (notes + 6) % 24, (notes + 12) % 24])
```

```
                triads.append([notes, (notes + 6) % 24, (notes + 12) % 24])
                g.add_node(temperament[notes] + ' Dim.', notes={notes, (notes + 6) % 24, (notes + 12) % 24}, feel='sad')
                colorMap.append('burlywood')

# Print scale to console
print("\ntemperament Scale""\n")
for num, note in temperament.items():
    print(num, note)

# Print major triads to console
print("\nMajor\n")
for elements in major:
    print(temperament[elements[0]], elements)

# Print minor triads to console
print("\nMinor\n")
for elements in minor:
    print(temperament[elements[0]], elements)

# Print augmented triads to console
print("\nAugmented\n")
for elements in augmented:
    print(temperament[elements[0]], elements)

# Print diminished triads to console
print("\nDiminished\n")
for elements in diminished:
    print(temperament[elements[0]], elements)

# Print total number of triads to console
print("\n\nTotal Triads\n")
print(triads.__len__())

print('\n')

# Add edges between nodes.
# The relationship in chord structure between two nodes
# that determines edge creation can be modified here.
for u in g.nodes():
    # Compare nodes with all other triads.
    for v in g.nodes():
        # Add edge when the intersection of u and v shares two elements
        # and the two elements of the symmetric difference of u and v
        # are neighbors in the ordered pitch class set.
        #
        # Check here if intersection has 2 elements
        if len(g.node()[u]['notes'] & g.node()[v]['notes']) == 2:
            # Assign both values of symmetric difference to variables.
            # Using min and max makes index comparisons with pitch set simple.
            a = min(g.node()[u]['notes'] ^ g.node()[v]['notes'])
            b = max(g.node()[u]['notes'] ^ g.node()[v]['notes'])
            # Add edge if symmetric difference is 1 semitone apart (neighbors in 'pitches')
            if (pitches.index(a) + 1) or (pitches.index(a) + 2) == pitches.index(b):
                g.add_edge(u, v)
            # Pitches on opposite ends of the list are considered neighbors
            # since scales wrap around on each other on the octave jump.
            elif pitches.index(a) == 0 and pitches.index(b) == (len(pitches) - 1) or (len(pitches) - 2):
                g.add_edge(u, v)

# ----------------------------
# ------ Tone Analysis -------
# ----------------------------
#
# Measure properties of popular scales
print("Connected?: ", nx.is_connected(g), '\n')
```

```
for nodes in g.node():
    print(nodes)
    print()
    print("   Degree                    : ", nx.degree(g, nodes))
    print("   Degree Centrality         : ", nx.degree_centrality(g)[nodes])
    print("   Closeness Centrality      : ", nx.closeness_centrality(g, nodes))
    print("   Betweenness Centrality    : ", nx.betweenness_centrality(g)[nodes])
    print("   Katz Centrality           : ", nx.katz_centrality(g)[nodes])
    print("   Load Centrality           : ", nx.load_centrality(g)[nodes])
    print("   Closeness Vitality        : ", nx.closeness_vitality(g, nodes))
    print('\n')

# ----------------------------
# ----- VISUALIZATION --------
# ----------------------------
#
# Create a visualization of user's
# graph of tones. Chords are colored
# coded using the following mapping:
#       Major - Red
#       Minor - Green
#       Aug.  - Blue
#       Dim.  - Tan
#
# Set node, edge, and font size (scale) of
# plot depending on the number of nodes.

if len(g.nodes()) < 12:
    nodeSize = 500
    fontSize = 12
    edgeWidth = 1
elif len(g.nodes()) < 24:
    nodeSize = 350
    fontSize = 10
    edgeWidth = 1
elif len(g.nodes()) < 36:
    nodeSize = 250
    fontSize = 8
    edgeWidth = 1
else:
    nodeSize = 180
    fontSize = 6
    edgeWidth = 1

# Set resolution of the plot
plt.figure(dpi=200)

# Draw graph
# Shapes available: circular, random, spectral, spring, or shell
nx.draw_circular(g, node_color=colorMap, node_size=nodeSize, width=edgeWidth,
                 with_labels=True, font_size=fontSize)

# Show plot
plt.show()
```