Master's Thesis

Revision of TPS with Switch Cost in Relative Keys

Sean Haber Arn

School of Information Science
Japan Advanced Institute of Science and Technology
March, 2017

Master's Thesis

Revision of TPS with Switch Cost in Relative Keys

s1510063 Sean Haber Arn

Supervisor:	**Satoshi Tojo**
Main Examiner:	**Satoshi Tojo**
Examiners:	**Satoshi Tojo**
	Nguyen Le Minh
	Kiyoaki Shirai

School of Information Science
Japan Advanced Institute of Science and Technology

February 2017

Abstract

In civilizations all over the world, music plays an important role in people's day-to-day lives. Due to music's universal presence, great amounts of time and money have been spent trying to uncover the underlying features and elements that are present in musical composition and structure. To do so, many computational theories of music have emerged, and these theories have been used to develop comprehensive algorithms by which music can be analyzed. While many theories do exist, music is not a "solved" field, and many questions still remain about music's properties. With the rise of machine learning and automatization, music has become a particularly popular field for researchers in such fields. More than ever, music is of interest, not for its composers, but for those who study it and the results they can yield and demonstrate.

Following along with these previous efforts to demystify music, the research in this paper seeks to further develop new theories of music and improve upon extant musical theories. This paper specifically focuses on Tonal Pitch Space, a musical theory that looks to represent music spatially and calculate distances in such musical space. By doing such spatial representation, one is able to model how listeners perceive musical similarities in a mathematical sense. While some musical distances are easily explained via resonances (C and G resonate well with one another), others are far more vague (C and D as opposed to C and B). TPS seeks to give a better understanding of such musical closeness and gives an apparently robust representation.. The representations and calculations performed in TPS are based on classical theories of music and can analyze classical music quite well. While this theory has its merits, it also has some shortcomings; among them, an insufficient distinction between major and minor keys. In this paper, a switch cost between major and minor keys is proposed so that the distances to and from major and minor keys are now unique. In order to test the switch cost and its merit as an addition to TPS theory, extensive testing was done using a Bebop jazz parser previously constructed by a colleague at JAIST. This particular parser was based on old parsers but used a TPS section and a Combinatory Categorial Grammar section to find cadences from input chords. CCG techniques had been taken from natural language processing, and were retailored to work for music. The previous version of the parser expanded upon existing CCG categories for jazz music and added several new ones to improve analysis. Moreover, the old parser was used to identify pivot chords – chords that are at the scene of a key change and could belong to either key – and gave highly accurate analysis of such pivot chords and their roles in chord progressions.

While the original version of the parser was able to analyze many pieces, it was unable to properly analyze chord progressions in minor keys and switches between relative keys. This problem likely stemmed from the previously mentioned inadequacy in TPS. To improve the parser, two changes were made: the switch cost was added, and a CCG constraint was also implemented. The CCG constraint was to make sure that there was Dominant→Tonic motion in the analyzed cadence. If there was not, then the progression was re-analyzed in the relative key. Both additions were rigorously tested in order to see if the parser is improved. Should the TPS switch cost work, it would be more important than the CCG constraint, as it could be directly implemented into extant TPS theory. Additionally, re-harmonization, the process by which a chord progression is changed to have a slightly different sound, is investigated to see the musical distances that are covered when the original chord sequence is re-harmonized. To perform this investigation, a survey is conducted wherein listeners are presented with three different re-harmonizations and are asked to choose the option that they think is best. Their selections provide insight into whether or not TPS as a theory is sufficient for jazz music. The results of the experiment demonstrate that both the switch cost and the CCG constraint are effective ways of improving the parser; however, the TPS switch cost's success shows that it should be implemented into TPS theory. By adding the switch cost to TPS, major and minor keys become more differentiated, making it possible for analysis to distinguish between major and minor keys. The results of the re-harmonization survey show that listeners generally prefer traditional re-harmonizations, as opposed to jazz re-harmonizations, indicating that perhaps a different theory should be used to analyze jazz music instead of TPS, or that TPS should be re-tuned to better account for jazz harmonies. TPS itself is a valuable theory that could have wide-reaching applications to all types of music, but at the moment may be too grounded in Western classical music.

Going forward, it will be important to re-tool the parser to work on more types of music, as jazz's rejection of classical chords and creation of its own specialized chords makes it unique among musical styles. While jazz has influenced many types of modern popular music, jazz chord patterns do not apply to other types of music. Therefore, a parser developed for the express purpose of parsing jazz music is relatively ill-suited for analyzing other types of music. Additionally, work should be done to analyze modal jazz, as it rejects the concept of a key signature, and instead relies on musical modes. In future work it may also be beneficial to shy away from such constraints as the CCG constraint introduced in this paper. While constraints do seem powerful, they, in many ways, reduce the power of analytical machines by placing rules on what the output analysis should be. In an optimal situation, the parser should always be able to give optimal

analysis without relying on constraints given from users. Such constraints also rely on the user to actually pass valid information to the machine. In the case where a machine is made for the general public, such user reliability does not necessarily exist and such constraints should be discarded. With no such CCG constraints, TPS alone should be able to handle analysis. The parser and the improvements to the parser described in this paper are presented with the goal of eventually yielding machines capable of automated composition and arrangement. If a machine can recognize chord sequences and valid re-harmonizations, then researchers are a step closer to having a machine generate valid chord sequences and re-harmonizations. Future researchers and future work on this topic should have the same goals in mind when furthering the work described in this paper

Table of Contents

List of Figures

1. Introduction

Music is one of humanity's greatest accomplishments, and it is present in nearly everyone's day-to-day lives. Every civilization that has ever existed has had music of some kind of music, and, while musical styles have changed over time, the love of music will certainly never die. As music has developed and been developed over time, great amounts of time and effort have been put into understanding music and its underlying elements and nuances. Such analysis has employed techniques from a broad range of areas, including mathematics as well as linguistics. With such techniques, researchers have been able to not only analyze music but to also develop algorithms for creating new music. The research outlined in this paper was performed with the goal of furthering such analysis and development.

As music has been developed over the centuries, stylization within music has led to the emergence of distinct genres that must be analyzed in different ways. Classical, jazz, rock 'n roll, country, and rap are all popular types of music and all have their own merits for being analyzed; however, the research in this paper focuses specifically on jazz. The structure of jazz differs greatly from that of classical music in that different types of chords and motifs are employed regularly. Such motifs and chords lead to the easily recognizable timbre of jazz music that is known worldwide. While there are many different styles of jazz, the research laid out in this paper focuses on Bebop jazz – a style developed in the 1940's. Bebop jazz is characterized by complex chord progressions, frequent key changes, and a relatively fast tempo. Bebop jazz is an influential style, and the chord progressions used in Bebop jazz music have now found their way into modern pop music.

The project described in this paper was done with two goals in mind. Firstly, it was necessary to improve an extant parser for jazz music. The previous parser, while useful, had some shortcomings that needed to be corrected in order to make it more efficient and accurate. Secondly, it was desirable to improve upon an existing music theory so as to make said theory more appealing for future musical analysis.

This paper is broken into 6 sections. After this introductory section, extant theories of music and language relevant to this research will be explained. Then, recent work on this topic, along with its limitations, will be given. Following that section, the current research will be explained. Next, experimentation methodology and results will be provided. Finally, the contributions of this research will be laid out, and limitations of the current project will be explained along with potential directions for future research and projects.

2. Background Theories

In order to understand the research presented in this paper, it is crucial to understand the theories on which the research is based. The two main theories are Tonal Pitch Space (TPS) and Combinatory Categorial Grammar (CCG). Additionally, a subsection of this paper will cover the Implication-Realization (I-R) Model, although the ideas presented in that model were not used in this particular project. Of TPS and CCG, the most important one for understanding this paper is TPS, so more time will be devoted to its explanation rather than CCG.

2.1 Tonal Pitch Space

As music is an arrangement of tones into some sort of cohesive composition, there exist definitive spaces and distances within music. On a piano, the note G is four white keys above C, and there exists a certain resonance between C and G that produces a pleasant effect for listeners

when the two notes are played simultaneously. For musicologists, such a relationship is known as a fifth, and the properties of fifths are well known. Tonal Pitch Space [1] builds upon basic concepts such as fifths and arranges all musical keys, chords, and scales so as to create a robust representation of musical space. TPS uses the basic formula

$$\delta(x \rightarrow y) = i + j + k$$

where $\delta(x \rightarrow y)$ is the distance between two points in music and where i is regional space, j is chordal space, and k is basic space. Many interpretations are possible, as can be seen in future subsections, but the shortest path should always be taken. With this formula, all distances – between chords, keys, or scales – can be calculated. These three spaces will be explained in the following subsections.

2.1.1 Regional Space

In TPS, regional space is the representation of all musical keys laid out so that keys with similar sets of notes are closer to one another, while keys with dissimilar sets of notes are further away.

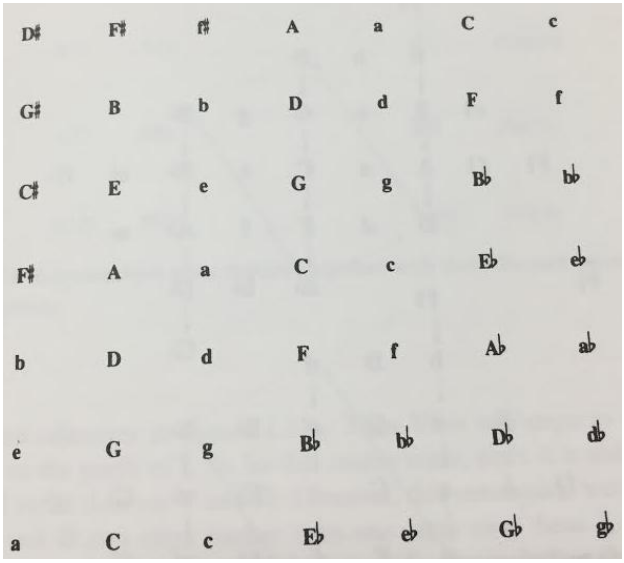


Fig. 1 Regional space
[1] *p. 65*

The capital letters in the grid represent the major keys, while the lowercase letters are used for minor keys. Each major key has its relative minor (the minor with the same set of notes) to its left and its parallel minor (the minor with the same starting note) to its right. Regional space is the most easily recognizable of the spaces as it can also be represented by the Circle of Fifths.

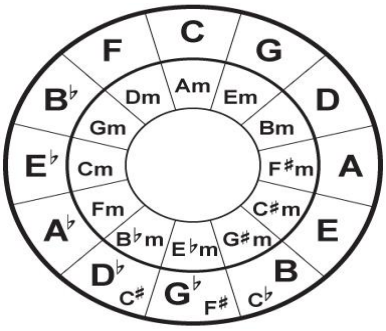


Fig. 2 Circle of Fifths

This space outlines the distances between keys themselves and is well-known by most, if not all people who have played a musical instrument with any regularity. Here, major keys are listed along the outside, and the relative minor of each major is listed on the inside. This pairing makes musical sense, as major keys and their relative minors have the same diatonic sets. Consider the two sets of notes given on the next page.

C Major: {C, D, E, F, G, A, B}
A minor: {A, B, C, D, E, F, G}

The top set is C major, and its notes are ordered; the bottom is A minor and its notes are also ordered. The two sets, it should be clear, are exactly the same when not considering order. It should be noted that the A minor listed here is the natural minor. There exist two other minors, harmonic minor and melodic minor:

Harmonic A minor: {A, B, C, D, E, F#, G}
Melodic A minor: {A, B, C, D, E, F, F#, G, G#}

The melodic minor is strange in that, when ascending, the 6th and 7th notes are both raised a half-step, but when descending, the notes are that of the natural minor.

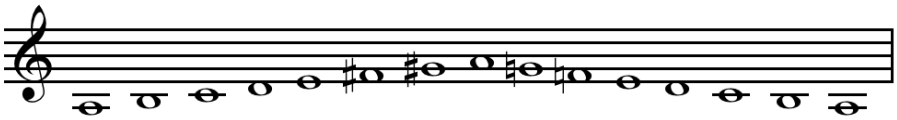


Fig. 3 Melodic A minor

In TPS, minor keys are defined as natural minors, and so they are similarly defined in this research.

Calculating distances in regional space is a matter of finding the shortest path. For example, moving from C to D is a distance of 2. The path will be C→G→D. The Circle of Fifths is potentially easier to understand, but the principle is the same for both the grid and the circle. When calculating distances to or from minor keys, movement between major keys and minor keys incurs no cost. When moving from A minor to D Major the path is a→C→G→D, but the distance is still 2. This calculation will be considered in more detail further into this paper. For now, there is no distinction between a major key and a minor key when calculating regional distance. From this simple calculation, the identity of ***i*** in the distance formula is determined.

2.1.2 Chordal Space

The next distance to be calculated is the distance that exists between chords. In any scale, there are 7 unique tones and thus 7 basic diatonic triads that are built upon those tones.

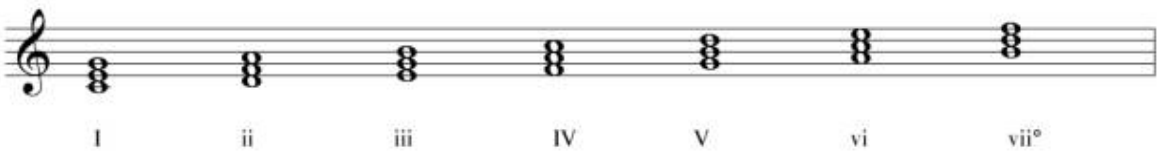


Fig. 4 C Major Triads

For major keys, the set of diatonic triads is *{I, ii, iii, IV, V, vi, vii}*. For minor keys, the set of diatonic triads is *{i, II, III, iv, v, VI, VII}*. Similarly, tetrads can be constructed from the notes of each scale. For the purposes of simplicity, triads will be considered for now, but the calculations are the same for tetrads.

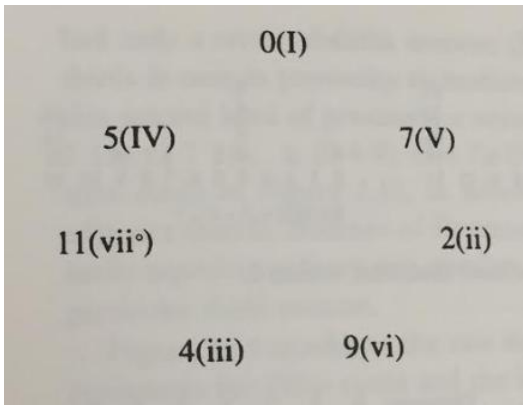


Fig. 5 Chordal circle of fifths
[1] *p. 55*

Figure 4 shows the arrangement of all the diatonic chords of a major scale (minor scales have the same arrangement, but the capitalization is inverted). The Roman numerals in parentheses are the degree of each chord, and the numbers outside the parentheses are the indices of the notes within the scale – in D Major, for example, 0 would be D. When calculating distances between two chords, take the shortest path along the circle. For example, in C Major, moving from the Tonic I chord to the vi chord is a distance of 3. Calculations here are performed in the same manner as those in regional space. Chords not listed on this circle involve a more complicated method of calculation that will be explained later. With this, the quantity ***j*** in the formula is calculated.

2.1.3 Basic Space

The final space in TPS is basic space, and it is the most complicated to calculate. The basic concept is that the tones of any scale can be arranged in a hierarchy, with tones appearing earlier being more resonant and closer to the root of the scale.

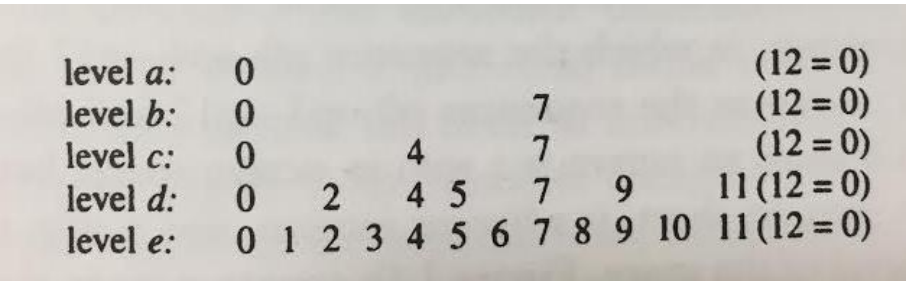


Fig. 6 C Major Basic Space
[1] *p. 47*

In Figure 6, level *e* is all possible notes, while *d* is the diatonic notes of a given scale. Levels *a-c* are the root, fifth, and triadic levels respectively, and they appear in the given order because of the resonances formed with the root (i.e. the root and the fifth at level *b* have better resonance than the root and the third at level *c*). Basic spaces can also be created over a particular tone within a scale. To create another basic space once one space has already been made, the rule is quite simple:

> Shift the elements at levels *a-c* either four steps to the left or four steps to the right (mod 7) [1].

With this rule, all of the basic spaces of a given key can be constructed, and the order of construction will match that of the chordal circle of fifths. Thus, the basic space of V/I (G in C Major) will be different from the basic space of I/V (G in G Major).

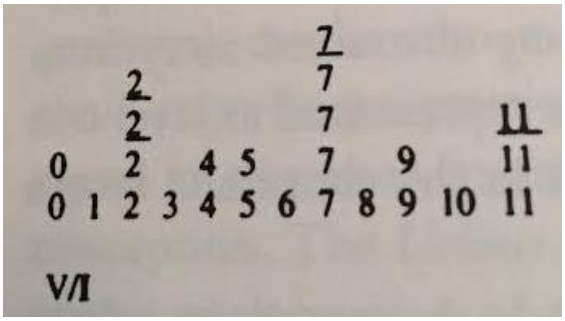


Fig. 7 V/I basic space
[1] *p. 54*

Calculating distances between basic spaces is a complicated process involving multiple steps. First, the start and end tones must be compared to see if they exist in the same scale. For

example, calculating the distance from C to G is viable because G appears in C's diatonic set. If the start and end tones are not in the same scale, the calculation requires extra spaces to be constructed. This will be discussed in a future sub-section. Once the basic spaces have been constructed, distance is calculated by taking the positive differences in the "heights" of the columns of tones.

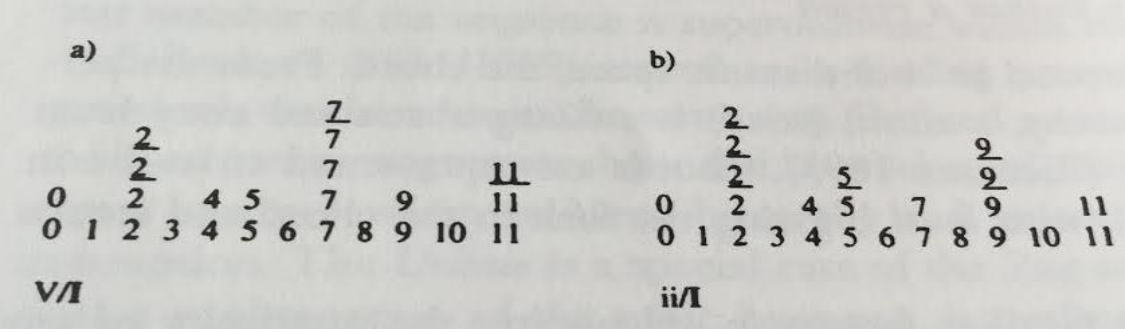


Fig. 8 Basic Space Calculation
[1] *p. 54*

In Figure 8, the basic space of the dominant of C and the minor ii are displayed. The distance between the two is 4. From ii/I, the "height" of the column of 2's is 5, whereas it is only 4 in V/I. Similarly, the 5 column is 1 higher, and the 9 column is 2 higher. The distance is therefore 1+2+1=4. It is important to note that, since the calculation only involves the positive heights, the reverse distance is also 4. This symmetry of distance does not hold when the same chords are interpreted as being of different keys.

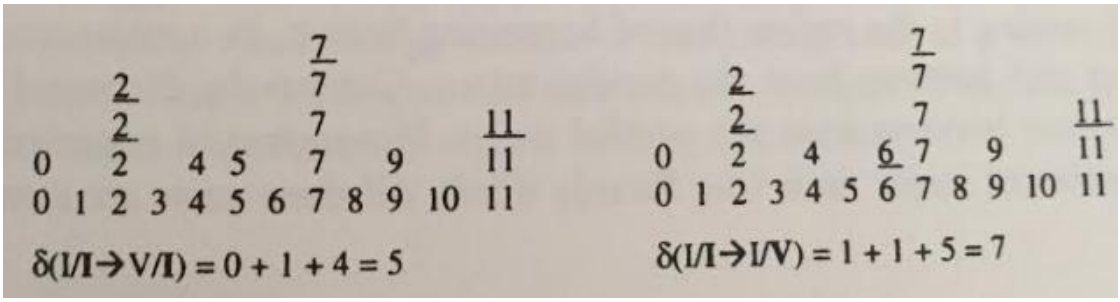


Fig. 9 Distance calculation from C Major tonic to G in two different keys
[1] *p. 61*

Figure 9 shows the distance from the tonic of C Major to the G Major chord. In the left-hand calculation, all distance calculations are performed in C Major. The right-hand side, however, switches from C Major to G Major, so that the basic space shown is the tonic of G Major. Therefore, the distances are different, as the figure shows.

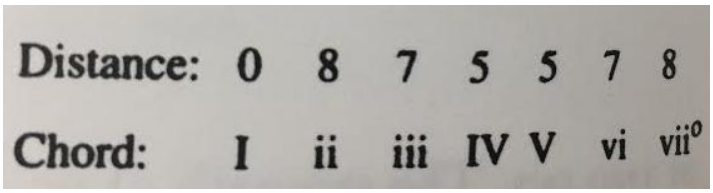


Fig.10 Summary of Distances from I to other Chords of the same key
[1] *p. 56*

These calculations finalizes the ***k*** quantity in the original equation, so all 3 quantities can be determined and the formula $\delta(x \rightarrow y) = i + j + k$ can be computed. Unfortunately, real computation of this equation is not as simple as the basic formula and relies on other important concepts.

2.1.4 Pivot Regions

Up until now, the concept of distance has relied on the assumption that the chords in question occur in the same, or nearby, keys. There are, however, many scenarios where this is not the case. When this happens, pivot regions must be used. Pivot regions allow one to switch to a distant chord via nearby chords.

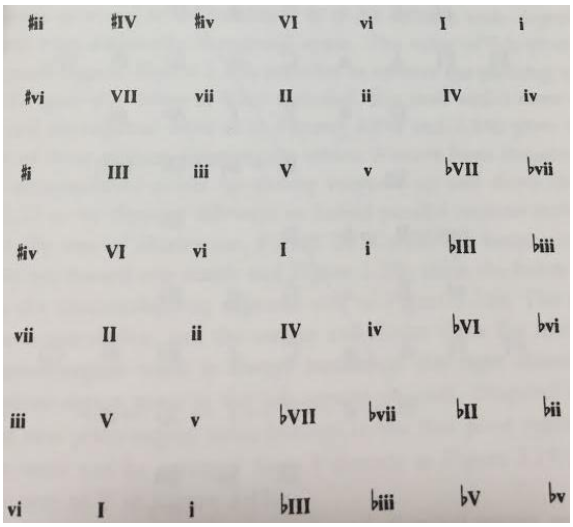


Fig. 11 Spatial representation of all chord locations
[1] *p. 65*

As shown in Figure 11, great distances must be traversed to reach all possible chord from the I chord. If the representation is re-centered on a different chord, the same distances must still be crossed to reach all other chords. Therefore, the concept of pivot regions is crucial in order to actually calculate distances between all chords and basic spaces.

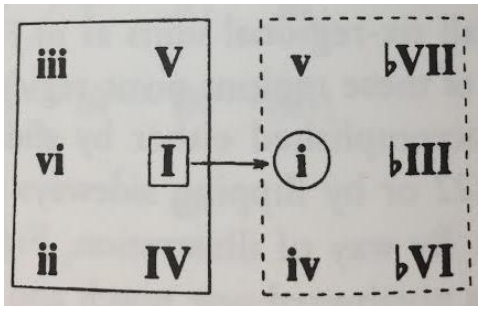


Fig. 12 Pivot Region Example
[1] *p. 68*

In Figure 12, the major ♭ III chord can be reached from I by pivoting through the minor i chord. By using these pivot regions, the computations become more complicated, but it is possible to reach any chord of any key from any chord of any key by shifting repeatedly. For each shift through pivots, a new basic space must be constructed so that the differences in heights can be calculated. As with the basic distance formula, the shortest path should always be taken. By

hand, this is a very time-consuming task; however, computers can perform this calculation very quickly. Thus, pivot regions are an essential part of the distance calculation. In the following figure, all distances from the I chord are shown. These can all be recalculated if the figure is re-centered on a different chord.

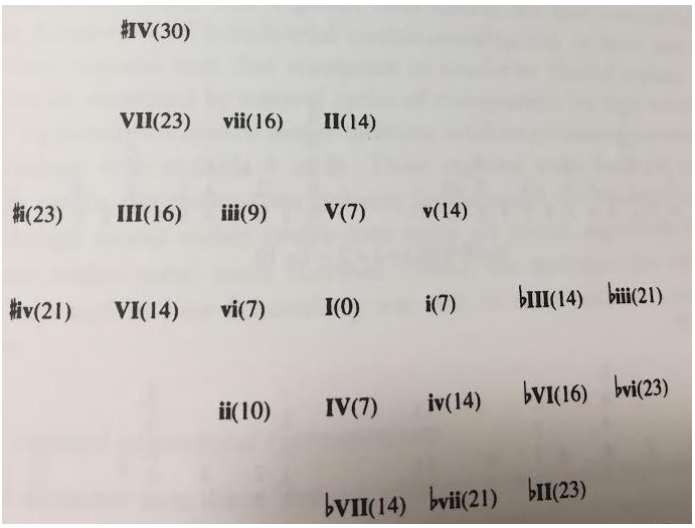


Fig. 13 Summary of distances from I to all other chords
[1] *p. 69*

Figure 13 is a modification of Figure 11 with distances appended to chord names. In 13, I is the center, but the representation can be re-centered to any chord if the distances are recalculated.

2.2 Combinatory Categorial Grammar

In Natural Language Processing, CCG is used to parse and construct sentences. Core to this technique is the idea that language is made up of elements that have features. These elements must be paired with other elements expecting the same features.

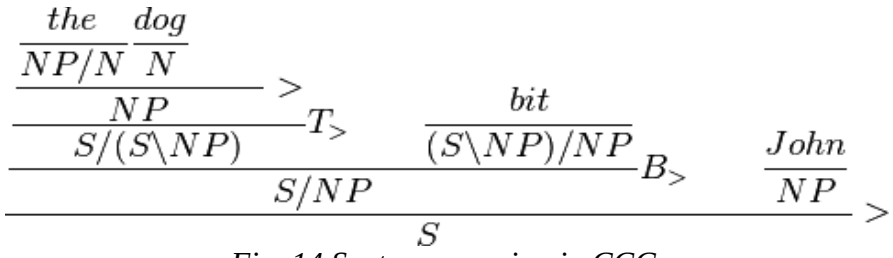


Fig. 14 Sentence parsing in CCG

Figure 14 uses the simple sentence *the dog bit John* to demonstrate these features. *Dog* has the feature N because it is a noun. *The* can form an NP if it finds the feature N in the adjacent word. The two match, so NP is formed using the application combinator >. Using a type-raising combinator, T>, NP yields S/(S\NP). Since *bit* yields (S\NP)/NP, the composition combinator B> allows (S\NP) to be removed since it is present from both NP and *bit*. Lastly, another > combinator is used, so S is derived.

Given that music is a series of elements that are combined according to a series of rules, researchers have sought to apply CCG rules to musical analysis. These rules come in the form of CCG cadence rules. In music, cadence is the melodic flow throughout the composition. A standard cadence is I→IV→V→I, or Tonic→Sub-Dominant→Dominant→Tonic. This particular cadence is essentially an establishment of the key, heightened tension in the middle, and a return to the key at the end. There exist many different variations of cadences, so researchers posit that it should be possible to develop CCG-esque rules that apply to music.

For the original parser, much time was spent developing a robust CCG system so as to parse chord sequences. While some work was done on CCG during the course of this research, the main focus was on TPS. The CCG algorithm was improved upon, but most time was spent on TPS and contributing to that theory. The upcoming main section will give an overview on TPS and CCG parsers for music.

2.3 Implication-Realization Model

Although it is not directly present in this research, the I-R Model [2] may be an important concept for future work on music parsing and generation. At its basis is the idea that music establishes certain implications during the course of a composition. These implications are later concluded at some point during the composition. Such implications throughout the piece create a sense of tension that should be resolved further along. These implications follow two basic patterns: A + B → C, and A + A → A. Here "→" is used to mean implies. At their bases, these two patterns mean two different elements imply a third, different element; and, repetition of the two same elements implies a third repetition of the same element. For these two patterns, Narmour claims that there exist five archetypes for melodies. These five archetypes will be explained in the following subsections.

2.3.1 Process and Duplication

Process, symbolized as [P], is a concept which encapsulates both intervallic motion and registral direction within the A + A pattern. The musical sequence C-D-E creates an upward-moving process where each element is equally separated. Moving from C to D creates an expectation of further upward movement which is later resolved by E. Also existing within the concept of processes is duplication, symbolized as [D]. As can be assumed from its name, duplication is a process in which an element is repeated. Thus, C-C-C is a duplicated pattern that fulfills the initial implications stemming from C-C.

2.3.2 Reversal

As process and duplication are within the A + A pattern, reversal belongs in the A + B pattern. A reversal occurs when "register changes direction as a minor third or more" [2].Therefore, G-E-

D(where the middle E is higher than the initial G and where the final D is lower than the middle E) forms a reversal. As the movement from G to E moves upward on the register, implications are generated so that listeners may expect continued upward movement. Said implications are later reversed with the lower D. Reversal, unlike process and duplication which can create further continued implications, can create a sense of melodic closure. The change of direction can serve to finalize and seal the set of elements.

2.3.3 Registral Return

Similar to reversal, but still distinct, registral return is a principle that applies directly to pitches themselves. Registral return describes an exact pitch pattern where a set of notes starts and ends at or near the starting pitch. G-D-A, for example, is a case of *near* registral return where the ending pitch is almost the same as the starting pitch. G-E-G, therefore, is *exact* registral return, as the starting and ending pitches are the same. Registral return is written as [aba] in cases of *exact* return, and [aba^{1}] in cases of *near* return.

2.3.4 Dyads and Monads

The final two archetypes are dyads and monads. Dyads are two items that are grouped together that form implications but do not have realizations. When dyads occur, they are symbolized by the interval between the two elements of the dyad. Thus, *[3]*, represents an interval of a third between the two notes. Monads are singular elements that form no implications. Said elements are in some way separated from other elements at the level where they occur. Often, such separation is literal (e.g. the monad appears in a different measure from its preceding notes). While monads and dyads do not resolve implications, identifying them is crucial to understanding music as a set of consistent elements.

2.3.5 Using the I-R Model

While the summary of the I-R Model given here is brief, a full understanding could potentially be beneficial for future efforts to automate re-harmonization. The I-R Model gives a complex method for analyzing all types of music. The concepts outlined in the I-R Model are applicable in any genre and, when applied properly, give a robust analysis of the inner workings of a musical composition. Once a piece has been analyzed via the I-R Model, it may be useful to follow the analytical skeleton given when re-harmonizing the same piece. Re-harmonizing or re-writing a piece so that it follows the same processes and registral returns but with different notes may lead to the composition of an equally aurally pleasing work of music. On the other hand, perhaps violating I-R analysis when re-harmonizing may lead to more pleasing music. Regardless, further investigation into the merits and inner workings of the I-R Model would not only be beneficial from an academic standpoint, but may also lead to serious breakthroughs in the field of automated composition and arrangement. While it was not in the scope of this research to do such investigation, future papers and projects may wish to devote more time to understanding the concepts laid out in the two books written on the I-R Model.

3. Existing Parsers and Revisions to TPS

Using the theories explained in the previous section, much work has been done to construct a parser that can analyze music. While musicologists can usually identify a piece's key very easily, it can take both time and effort. Developing a parser that can not only identify the key but can also find the harmonic cadences would save time and would open the door for future generative machines. A machine that could analyze music would hypothetically be able to later take its analysis and use it to construct entirely new works of music. Additionally, with the help

of powerful machines, the possibility to complete previously unfinished works (e.g. Mozart's *Requiem*) via computer algorithm is quite tempting.

3.1 Granroth-Wilding's Jazz Parser

The basis for the current parser comes from a jazz parser developed in 2014 that used Probabilistic CCG (PCCG) and was written in Python [3]. In this particular parser, a set of CCG rules were developed for jazz music and probabilities were assigned to each rule category. This parser focused on CCG only and featured no TPS calculations. This parser performed better than a hidden Markov model that was used as a baseline because of the constraints imposed by the CCG rules. This parser and the results it yielded set the basis for the rest of the work described in this paper.

3.1.1 Limitations of the PCCG Parser

The original PCCG parser, although a step forward, had some limitations. Chief among these were a significant number of incorrect parse trees. The following example, when passed to the PCCG parser as input, yields a cadence composed entirely of tonics. In this case, the better analysis would be Tonic→Tonic→Sub-Dominant→Dominant→Tonic.

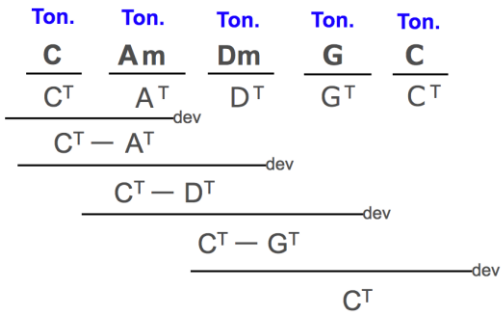


Fig. 15 Incorrect analysis from the PCCG parser

It is important to rectify such mistakes to make a better parser. Generally, sequences of only tonics are not particularly interesting or pleasing to the listener. Giving only tonics as a cadential output therefore cannot be correct and the algorithm leading to such incorrect analysis should be re-examined and corrected.

Furthermore, jazz music uses a wide variety of chords that have their own specific cadential rules (e.g. Sub-Dominant Minor). The Granroth-Wilding parser included a relatively limited number of categories, so expanding the parser to contain more categories would increase its analytical power. Additionally, although PCCG alone worked well in the parser, adding a TPS section to the parser to communicate with the CCG section was seen as a way to increase parser accuracy. TPS would be used to handle key-finding, and CCG would be used to finalize cadential analysis in the key identified by the TPS section. These two sections would, together, improve accuracy and give better outputs when compared to the original parser.

3.2 Sakamoto's Harmonic Parser

As TPS is a useful method by which one can calculate distances in musical space, researchers wanted to create an implementation by which harmonies in music could be identified. One such researcher modified the original TPS pivot formula so as to improve the calculation for very distant keys [4].

$$\delta(x,y) = \begin{cases} region(x,y) + chord(x,y) + basicspace(x,y) & \text{Close Key} \\ \min\{\delta(x, \mathrm{I}/R_1) + \Delta(R_1, R_2) + \delta(\mathrm{I}/R_2, y) | R_1 \in \mathrm{P}(x),\ R_2 \in \mathrm{P}(y)\} & \text{Remote Key} \end{cases}$$

The upper formula is the original, basic TPS formula when chords' keys are close. These close keys can be accessed directly without having to use pivot regions. The lower formula is for calculating distances between chords when pivot regions are necessary. Here, the computation is

done in 3 parts. First, the starting chord *x*'s distance from the I chord in region *R1* is calculated. Then, the distance between the regions *R1* and *R2* (chord *y*'s region) is calculated. Lastly, the distance from the I chord of *R2* to *y* is calculated. There are many possible calculations that must be performed because *R1* and *R2* are regions along the possible paths between *x* and *y*. Lastly, the path with the minimum distance from all calculations is selected and given as output.

By using this formula, Sakamoto was able to develop and implement an algorithm by which harmonies could be identified from input chords. First, chords were given and candidate degrees and keys for each chord were listed. Then, using the formula given above, the distances between all candidates were calculated. Lastly, the shortest distance was selected, and the harmony was identified. This algorithm was then used to create a parser that could identify harmonies when given sequences of chords.

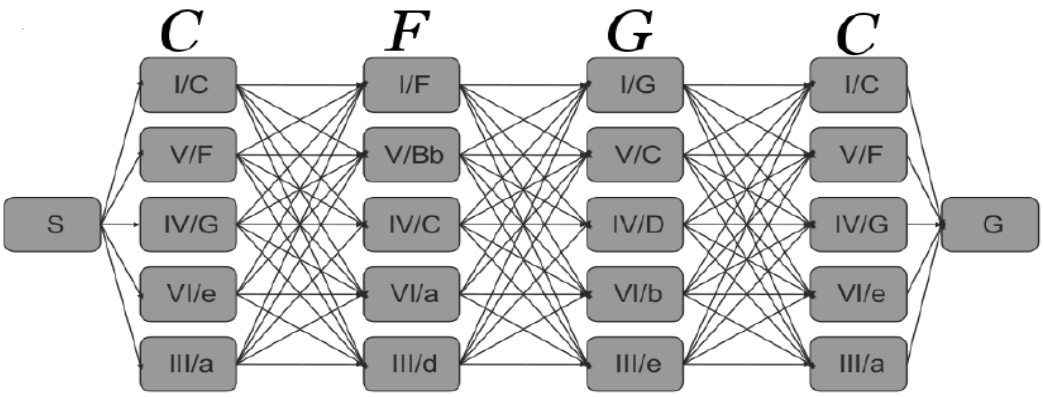


Fig. 16 Example of Sakamoto's algorithm

In Figure 16, the input chord sequence is C→F→G→C. The key is unknown, so each chord has multiple interpretations. These interpretations are enumerated below each chord (e.g. I/C, V/F). After the enumeration, the distances from all nodes to all following nodes are calculated. Here, the correct interpretation of the harmony is **I/C→IV/C→V/C→I/C**. This algorithm is currently implemented in the current parser which will be described later in this paper.

3.3 Yamaguchi's Addition

Another important revision to TPS is the adding of another layer to the basic space hierarchy [5]. In music, non-diatonic notes are often incorporated into compositions. A popular non-diatonic note to include is the one that completes the dominant I_7 chord. Yamaguchi proposes adding an extra layer before the diatonic layer in order to include this non-diatonic note.

Level.a	C											
Level.b	C							G				
Level.c	C				E			G				
Level.d	C				E			G			Bb	
Level.e	C		D		E	F		G		A	*Bb*	B
Level.f	C	Db	D	Eb	E	F	Gb	G	Ab	A	Bb	B

Fig. 17 Yamaguchi's extra non-diatonic level

In Figure 17, level d is an addition to the original TPS theory. In C Major, B ♭ is the non-diatonic note that completes the I_7 tetrad. With Yamaguchi's contribution, it is now possible to consider the I_7 chord as a diatonic chord, meaning that B ♭ is included at the diatonic level, simplifying TPS calculations for chord sequences involving this originally non-diatonic note. When the non-diatonic note is not included in a chord, the basic space remains unchanged. Yamaguchi's contribution is included in the parser described in this paper.

3.4 Fukunari's Parser

To address the limitations of the Granroth-Wilding parser, a new parser was developed by a JAIST student, Takeshi Fukunari [6]. The new parser was written in Java and comprises more than 2000 lines of code. This parser, as implied previously, contained both a TPS and a CCG section. A sequence of chords was given as input, and degree and function were given as output. Moreover, the new parser expanded upon the original CCG rules given in the Granroth-

Wilding parser and added 3 new categories: Sub-Dominant Minor, Passing Diminish, and Related II Minor Seventh.

	Tonic	Sub Dominant	Dominant	Sub Dominant Minor
Major Key	$I := T^{I}$ $I_6 := T^{I_6}$ $I_{M7} := T^{I_{M7}}$ $III_- := T^{III_-}$ $III_{-7} := T^{III_{-7}}$ $VI_- := T^{VI_-}$ $VI_{-7} := T^{VI_{-7}}$ $I_7 := T^{I_7}$ $^{\#}IV^{(\flat5)}_{-7} := T^{^{\#}IV^{(\flat5)}_{-7}}$	$IV := S^{IV}/(T \mid D \mid SM)$ $IV_6 := S^{IV_6}/(T \mid D \mid SM)$ $IV_{M7} := S^{IV_{M7}}/(T \mid D \mid SM)$ $II_- := S^{II_-}/(T \mid D \mid SM)$ $II_{-7} := S^{II_{-7}}/(T \mid D \mid SM)$ $IV_7 := S^{IV_7}/(T \mid D \mid SM)$ $VII_7 := S^{VII_7}/(T \mid D \mid SM)$ $^{\#}IV^{(\flat5)}_{-7} := S^{^{\#}IV^{(\flat5)}_{-7}}/(T \mid D \mid SM)$	$V := D^{V}/T$ $V_7 := D^{V_7}/T$ $VII^{(\flat5)}_{-} := D^{VII^{(\flat5)}_{-}}/T$ $VII^{(\flat5)}_{-7} := D^{VII^{(\flat5)}_{-7}}/T$ $^{\flat}II_7 := D^{^{\flat}II_7}/T$	$^{\flat}VII_7 := SM^{^{\flat}VII_7}/(T \mid D)$ $^{\flat}II_{M7} := SM^{^{\flat}II_{M7}}/(T \mid D)$ $^{\flat}VI_7 := SM^{^{\flat}VI_7}/(T \mid D)$
Minor Key	$I_- := T^{I_-}$ $I_{-6} := T^{I_{-6}}$ $I_{-7} := T^{I_{-7}}$ $I_{-M7} := T^{I_{-M7}}$ $^{\flat}III := T^{^{\flat}III}$ $^{\flat}III_{M7} := T^{^{\flat}III_{M7}}$ $^{\flat}III^{+} := T^{^{\flat}III^{+}}$ $^{\flat}III^{+}_{M7} := T^{^{\flat}III^{+}_{M7}}$ $VI^{(\flat5)}_{-} := T^{VI^{(\flat5)}_{-}}$ $VI^{(\flat5)}_{-7} := T^{VI^{(\flat5)}_{-7}}$	$IV := S^{IV}/(T \mid D \mid SM)$ $IV_6 := S^{IV_6}/(T \mid D \mid SM)$ $IV_7 := S^{IV_7}/(T \mid D \mid SM)$ $II_- := S^{II_-}/(T \mid D \mid SM)$ $II_{-7} := S^{II_{-7}}/(T \mid D \mid SM)$ $VII_7 := S^{VII_7}/(T \mid D \mid SM)$	$V := D^{V}/T$ $V_7 := D^{V_7}/T$ $V_{-7} := D^{V_{-7}}/T$ $VII_{\circ} := D^{VII_{\circ}}/T$ $VII^{(\flat5)}_{-7} := D^{VII^{(\flat5)}_{-7}}/T$ $^{\flat}II_7 := D^{^{\flat}II_7}/T$	$IV_- := SM^{IV_-}/(T \mid D)$ $IV_{-6} := SM^{IV_{-6}}/(T \mid D)$ $IV_{-7} := SM^{IV_{-7}}/(T \mid D)$ $IV_{-M7} := SM^{IV_{-M7}}/(T \mid D)$ $II^{(\flat5)}_{-} := SM^{II^{(\flat5)}_{-}}/(T \mid D)$ $II^{(\flat5)}_{-7} := SM^{II^{(\flat5)}_{-7}}/(T \mid D)$ $^{\flat}VI := SM^{^{\flat}VI}/(T \mid D)$ $^{\flat}VI_{M7} := SM^{^{\flat}VI_{M7}}/(T \mid D)$ $^{\flat}VII := SM^{^{\flat}VII}/(T \mid D)$ $^{\flat}VII_7 := SM^{^{\flat}VII_7}/(T \mid D)$ $^{\flat}II_{M7} := SM^{^{\flat}II_{M7}}/(T \mid D)$ $^{\flat}VI_7 := SM^{^{\flat}VI_7}/(T \mid D)$

	Related II_{-7}		Secondary Dominant	Passing Diminish
Major Key	$II_{-7} := R^{II_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $III_{-7} := R^{III_{-7}}/SD^{[VI_7\mid ^{\flat}III_7]}$ $III^{(\flat5)}_{-7} := R^{III^{(\flat5)}_{-7}}/SD^{[VII_7\mid ^{\flat}III_7]}$ $^{\#}IV_{-7} := R^{^{\#}IV_{-7}}/SD^{[VII_7\mid IV_7]}$ $^{\#}IV^{(\flat5)}_{-7} := R^{^{\#}IV^{(\flat5)}_{-7}}/SD^{[VII_7\mid IV_7]}$ $V_{-7} := R^{V_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $VI_{-7} := R^{VI_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $VII_{-7} := R^{VII_{-7}}/SD^{[III_7\mid ^{\flat}VII_7]}$ $VII^{(\flat5)}_{-7} := R^{VII^{(\flat5)}_{-7}}/SD^{[III_7\mid ^{\flat}VII_7]}$	$^{\flat}VI_{-7} := R^{^{\flat}VI_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $^{\flat}VII_{-7} := R^{^{\flat}VI_{-7}}/SD^{[VI_7\mid ^{\flat}III_7]}$ $^{\flat}VII^{(\flat5)}_{-7} := R^{^{\flat}VI^{(\flat5)}_{-7}}/SD^{[VI_7\mid ^{\flat}III_7]}$ $I_{-7} := R^{I_{-7}}/SD^{[VII_7\mid IV_7]}$ $I^{(\flat5)}_{-7} := R^{I^{(\flat5)}_{-7}}/SD^{[VII_7\mid IV_7]}$ $^{\flat}II_{-7} := R^{^{\flat}II_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $^{\flat}III_{-7} := R^{^{\flat}III_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $IV_{-7} := R^{IV_{-7}}/SD^{[III_7\mid ^{\flat}VII_7]}$ $IV^{(\flat5)}_{-7} := R^{IV^{(\flat5)}_{-7}}/SD^{[III_7\mid ^{\flat}VII_7]}$	$I_7 := SD^{I_7}/S^{[IV\mid IV_6\mid IV_{M7}]}$ $II_7 := SD^{II_7}/D^{[V\mid V_7]}$ $III_7 := SD^{III_7}/T^{[VI_-\mid VI_{-7}]}$ $VI_7 := SD^{VI_7}/S^{[II_-\mid II_{-7}]}$ $VII_7 := SD^{VII_7}/T^{[III_-\mid III_{-7}]}$ $^{\flat}V_7 := SD^{^{\flat}V_7}/S^{[IV\mid IV_6\mid IV_{M7}]}$ $^{\flat}VI_7 := SD^{^{\flat}VI_7}/D^{[V\mid V_7]}$ $^{\flat}VII_7 := SD^{^{\flat}VII_7}/T^{[VI_-\mid VI_{-7}]}$ $^{\flat}III_7 := SD^{^{\flat}III_7}/S^{[II_-\mid II_{-7}]}$ $IV_7 := SD^{IV_7}/T^{[III_-\mid III_{-7}]}$	$^{\#}I_{\circ} := PD^{^{\#}I_{\circ}}$ $^{\#}II_{\circ} := PD^{^{\#}II_{\circ}}$ $^{\#}IV_{\circ} := PD^{^{\#}IV_{\circ}}$ $^{\#}V_{\circ} := PD^{^{\#}V_{\circ}}$ $^{\#}VI_{\circ} := PD^{^{\#}VI_{\circ}}$
Minor Key	$II_{-7} := R^{II_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $II^{(\flat5)}_{-7} := R^{II^{(\flat5)}_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $IV_{-7} := R^{IV_{-7}}/SD^{[^{\flat}VII_7\mid III_7]}$ $V_{-7} := R^{V_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $V^{(\flat5)}_{-7} := R^{V^{(\flat5)}_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $VI_{-7} := R^{VI_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $VI^{(\flat5)}_{-7} := R^{VI^{(\flat5)}_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $^{\flat}VII_{-7} := R^{^{\flat}VII_{-7}}/SD^{[^{\flat}III_7\mid VI_7]}$ $I_{-7} := R^{I_{-7}}/SD^{[IV_7\mid VII_7]}$	$^{\flat}VI_{-7} := R^{^{\flat}VI_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $^{\flat}VI^{(\flat5)}_{-7} := R^{^{\flat}VI^{(\flat5)}_{-7}}/D^{[V\mid V_7\mid ^{\flat}II_7]}$ $VII_{-7} := R^{VII_{-7}}/SD^{[^{\flat}VII_7\mid III_7]}$ $^{\flat}II_{-7} := R^{^{\flat}II_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $^{\flat}II^{(\flat5)}_{-7} := R^{^{\flat}II^{(\flat5)}_{-7}}/SD^{[I_7\mid ^{\flat}V_7]}$ $^{\flat}III_{-7} := R^{^{\flat}III_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $^{\flat}III^{(\flat5)}_{-7} := R^{^{\flat}III^{(\flat5)}_{-7}}/SD^{[II_7\mid ^{\flat}VI_7]}$ $III_{-7} := R^{III_{-7}}/SD^{[^{\flat}III_7\mid VI_7]}$ $^{\flat}V_{-7} := R^{^{\flat}V_{-7}}/SD^{[IV_7\mid VII_7]}$	$I_7 := SD^{I_7}/(\ S^{[IV\mid IV_6\mid IV_7]} \mid SM^{[IV_-\mid IV_{-6}\mid IV_{-7}\mid IV_{-M7}]})$ $II_7 := SD^{II_7}/D^{[V\mid V_7\mid V_-\mid V_{-7}]}$ $^{\flat}III_7 := SD^{^{\flat}III_7}/SM^{[^{\flat}VI\mid ^{\flat}VI_{M7}]}$ $IV_7 := SD^{IV_7}/SM^{[^{\flat}VII\mid ^{\flat}VII_7]}$ $VI_7 := SD^{VI_7}/S^{[II_-\mid II_{-7}]}$ $^{\flat}VII_7 := SD^{^{\flat}VII_7}/T^{[^{\flat}III\mid ^{\flat}III_{M7}]}$ $^{\flat}V_7 := SD^{^{\flat}V_7}/(\ S^{[IV\mid IV_6\mid IV_7]} \mid SM^{[IV_-\mid IV_{-6}\mid IV_{-7}\mid IV_{-M7}]})$ $^{\flat}VI_7 := SD^{^{\flat}VI_7}/D^{[V\mid V_7\mid V_-\mid V_{-7}]}$ $VI_7 := SD^{VI_7}/SM^{[^{\flat}VI\mid ^{\flat}VI_{M7}]}$ $VII_7 := SD^{VII_7}/SM^{[^{\flat}VII\mid ^{\flat}VII_7]}$ $^{\flat}III_7 := SD^{^{\flat}III_7}/S^{[II_-\mid II_{-7}]}$ $III_7 := SD^{III_7}/T^{[^{\flat}III\mid ^{\flat}III_{M7}]}$	$^{\#}I_{\circ} := PD^{^{\#}I_{\circ}}$ $III_{\circ} := PD^{III_{\circ}}$ $^{\#}IV_{\circ} := PD^{^{\#}IV_{\circ}}$ $VI_{\circ} := PD^{VI_{\circ}}$ $VII_{\circ} := PD^{VII_{\circ}}$

Fig.18 CCG Cadence Rules

Figure 18 displays all the rules in the new parser. The blue letters show non-diatonic chords, which is an important improvement over the Granroth-Wilding parser. Jazz music often features such chords, so it is crucial to have a way to analyze these types of chords.

Additionally, Fukunari's parser sought to identify pivot chords. In music, sometimes a chord could be a member of two different keys. In these cases, one interpretation will be better than the other cadentially. The updated parser can successfully find and interpret pivot chords.

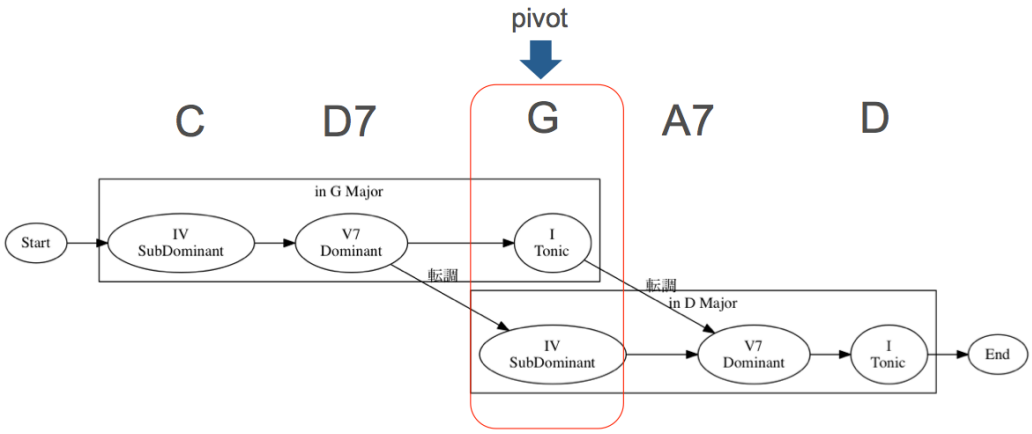


Fig. 19 Pivot chord in a cadence

In Figure 19, G could be I in G Major or IV in D Major. The parser finds both interpretations and lists both as possible degrees of the chord; however, for its final output it chooses the I interpretation as better and shows that. This is an improvement over the Granroth-Wilding parser as, before, there had been no function for identifying pivot chords. The ability to identify pivot chords is important for a jazz parser, as jazz involves frequent key changes. Showing where the change occurs as well as what the correct analysis is makes the parser a powerful analytical tool. In modern music, as well, key changes serve to keep the listener interested in the music, so having the ability to show key changes and the pivoting chords is highly useful.

The parser was tested using roughly 20 jazz compositions, and the outputs were compared to previous analyses that had been done by hand [7]. With the additions made to the CCG section as well as the creation of the entirely new TPS section, the parser was improved greatly. The value of the function to identify pivot chords also made the parser more useful than the original parser made by Granroth-Wilding.

3.4.1 Limitations of the Upgraded Parser

While the new parser was more powerful than the original PCCG parser, some serious problems still remained. Firstly, the parser was generally unable to identify cadences in minor keys. Since major keys and their relative minors share the same diatonic sets, there is no difference in distance when considering one over the other. The cadential output, however, will only make sense for one interpretation. A cadence in a minor key will have no Dominant→Tonic motion when analyzed from the perspective of a major key. The cadence Am→Dm→E→Am is i→iv→V →i in A minor; however it is vi→ii→III→vi in C Major. The latter cadence is nonsensical and should not be given as output. The original parser, while not giving output quite as wrong as the second cadence, analyzes this particular input as vi→iv→V→vi, which is incorrect. This shortfall needed to be corrected in future revisions to the parser.

Moreover, the parser's input method is not intuitive. For example, to give the C Major triad C-E-G as input, the exact input is **{ "0", "M"}**. While this is not difficult for musicologists or those familiar with the parser, it is not easy for first-time users. Additionally, output is only in text format, so it is not particularly visually appealing.

```
input(Chord progression) [ C F G7 C ]

     output: I  in CMajor Tonic
  output: IV in CMajor Sub Dominant
   output: V7 in CMajor Dominant
     output: I  in CMajor Tonic
```

Fig. 20 Example output

As can be seen in Figure 20, while the output is fairly easy to read, it is not truly eye-pleasing. Moreover, when cadences become more complicated, there are many analyses given, so it becomes even harder to read the output. A potentially better output format would be musicXML

so that analysis could be put directly onto analyzed sheet music. These aesthetic changes were not attempted in the course of this research, so it remains to be done in future upgrades.

In all, the parser still requires improvements in order to be both more powerful and more accurate. Fukunari's parser is a significant improvement over the original PCCG parser, but it is not the analytical device it could be. A direct contribution of the work described in this paper is the improvement of Fukunari's parser. The parser itself becomes a means by which more theoretical contributions can be tested. The ways in which the parser was improved will be described in the upcoming sections.

4. Revisions to the Parser and TPS

It has been noted that the parser developed by Fukunari, while useful, has significant problems with progressions in minor keys. When such a progression is passed as input to the parser, the parser will say that at least part of the input is in the relative major.

```
input(Chord progression) [ Am7 Dm7 E7 Am7 ]

        output: VIm7 in CMajor Tonic
  output: IVm7 in AMinor Sub Dominant Minor
       output: V7 in AMinor Dominant
        output: VIm7 in CMajor Tonic
```

Fig. 21 Incorrect analysis of a minor chord progression

The chord progression in Figure 21 should be analyzed as being entirely in A minor. Instead, the parser determines that the beginning and ending tonics are of C Major while the middle two chords belong to A minor. While this cadentially is not wrong – vi→iv→V→vi can be a valid cadence – there is no modulation here, and the sound of the progression is clearly minor in its entirety. This incorrect analysis is output when a minor key is passed as input. Similarly, when there is modulation between major and minor keys, analysis fails.

```
input(Chord progression) [ Cm7 F7 B♭M7 E♭M7 Am7(♭5) D7 Gm ]

output: IIm7 in B♭Major Sub Dominant
output: V7 in B♭Major Dominant
output: IM7 in B♭Major Tonic
output: IVM7 in B♭Major Sub Dominant
output: VIIm7(♭5) in B♭Major Dominant
output: V7 in GMinor Dominant
output: VIm in B♭Major Tonic
```

Fig. 22 Incorrect analysis of Major-minor modulation

For this progression, the last four chords are actually in G minor, but the parser cannot analyze them correctly. In order to correct these errors, changes were made to both the TPS and the CCG sections.

The goal of this research, however, is not solely parser-based. While the parser is an important part of this project, the more important goal is to modify TPS theory itself. While TPS is a useful theory, it suffers from the serious drawback of not being able to properly analyze minor keys. Major keys and minor keys are treated as being essentially the same (i.e. there is no regional distance between a major key and its relative minor). It therefore became necessary to give strong consideration to what exactly was wrong with TPS itself. Modifying TPS is thus the main focus of this research, and the changes to TPS will be described in a subsection 4.2. First, the CCG revision to the parser will be covered.

4.1 CCG Revision

In the parser, the TPS and CCG sections have almost no communication with one another. The TPS section identifies the key, passes it to the CCG section, and then is finished working. Once the CCG section gets the key information from the TPS section, it checks to see if there is a valid cadence, outputs the valid cadences, and then stops working. It was decided that adding some sort of checking to the process would greatly help analytical accuracy. Therefore, a

feedback loop was added to the output process. The problem with Fukunari's upgraded parser was that it identified minor key cadences as belonging to a major key. This meant that the cadence was nonsensical when given as output. Therefore, it was important to check that the cadence made sense via constraints imposed on the parser. Thus, the feedback loop checks the analyzed cadence to see if a Dominant→Tonic or Sub-Dominant→ Tonic condition is met. If the condition is not met, then analysis switches to the relative key (i.e. if initial analysis was in the major, it switches to the minor, and vice versa) and re-analyzes. If no valid cadence is found after the switch, analysis has failed and no output is given.

In order to do this checking, once the CCG section has been given the key from the TPS section, the input is broken up by tonics. This method assumes that if there is a valid cadence within the subsections of the progression, then there is a valid cadence when all the subsections are taken together. Input is broken into phrases ending in tonics and CCG analysis is performed Each subsection ending in a tonic is analyzed under the constraints mentioned earlier. If none of the subsections passes, no output is given. If the beginning of the progression can be analyzed, output is given up until the CCG section cannot find a valid cadence.

```
input(Chord progression) [ CM7 G7 CM7 CM7 B♭m B ]

 output: IM7 in CMajor Tonic
output: V7 in CMajor Dominant
 output: IM7 in CMajor Tonic
 output: IM7 in CMajor Tonic
```

Fig. 23 Partially successful analysis

In the example given, analysis succeeds initially, but the last two chords (added to illustrate the point being made) cannot form a licit cadence and are thus excluded. The same exclusion also occurred in the previous version of the parser; however, the current version, with the CCG constraint, now stringently checks to make sure that all output is actually valid. The exact code

for the CCG feedback can be found in the appendix. With only the CCG feedback active, the parser works as shown in the following diagram.

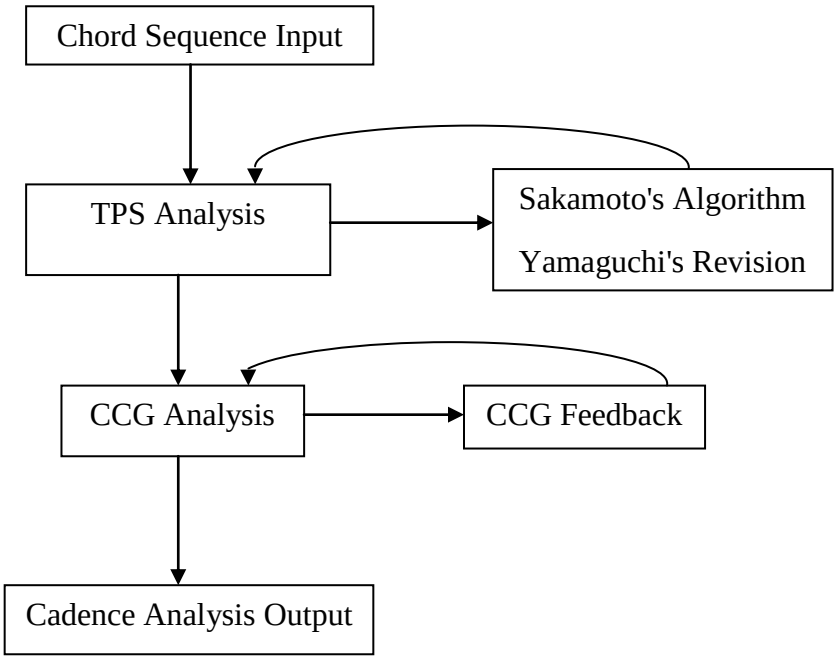


Fig. 24 Diagram of parsing with CCG constraint

While the CCG revision certainly helps make the parser more accurate and powerful, it is not the main focus of this paper. As it improved the original Granroth-Wilding PCCG parser, the union of CCG and TPS was the main feature of Fukunari's parser. With CCG working accurately with such a small modification, it came time to examine the TPS section and see if it was responsible for previous inaccuracies in the parser.

4.2 TPS Revision

Being unable to identify pieces written in minor keys is a problem because it limits the analytical power of the parser. Obviously, increasing said power would be beneficial; however, the problem seems to come from an underlying problem in TPS: major keys and minor keys are

treated as being the same at the regional level. Movement between A minor and C Major has a distance of 0 at the regional level. While C Major and natural A minor have the same diatonic sets, the sound produced by the two keys is distinct, and listeners can usually recognize switches between the two. As implemented in the original parser, TPS cannot yield the correct key analysis when pieces are in minor keys.

To remedy this problem, it was proposed that an extra cost of 0.5 be added to TPS calculations when switches between major and minor keys occurred. The 0.5 value was decided upon because it was thought that, given the overlapping diatonic sets between major keys and their relative minors, movement between major and minor is smaller than movement between two major keys. As was discussed in the TPS theory section of this paper, movement from one major to another major incurs a cost of at least 1 (e.g. C→G is 1). Each full cost of 1 essentially occurs because the diatonic set is slightly changing. Thus, the switch from major to minor and vice versa, as it is a change in tone with no change in diatonic sets, should be of smaller magnitude than a full shift at the regional level. Thus, the original TPS formula:

$$\delta(x \rightarrow y) = i + j + k$$

becomes slightly modified. The quantity ***i*** can now have 0.5 added to it every time there is a switch between major and minor keys. This means that all other TPS calculations will also be slightly changed. Pivot regions, when the pivot is between major and minor keys, will also be increased by 0.5 per major-minor switch. For example, in Figure 13, going from I (which is in C Major) to ii (in D minor) will now have a cost of 10.5 as opposed to its previous cost of 10.

As Fukunari's parser already included TPS distance calculations, only minor modifications needed to be made to the parser's code to implement the 0.5 switch cost. To make

this change in Java, the value of the distance calculation was changed from an integer to a float. Previous TPS calculations only involved integer values, so adding 0.5 to the calculation requires a different variable type. A condition was added such that when the start key's mode (major or minor) doesn't match the end key's mode, 0.5 is added to the calculation. The code for said condition is as follows:

```
if(!fromMode.equals(toMode))
regiondistance = (float)0.5;
```

As mentioned before, the "modes" referred to here are major and minor. The above calculation will take place multiple times when the starting chord and ending chord are far away from one another. The following diagram outlines how the parser works with only the switch cost added.

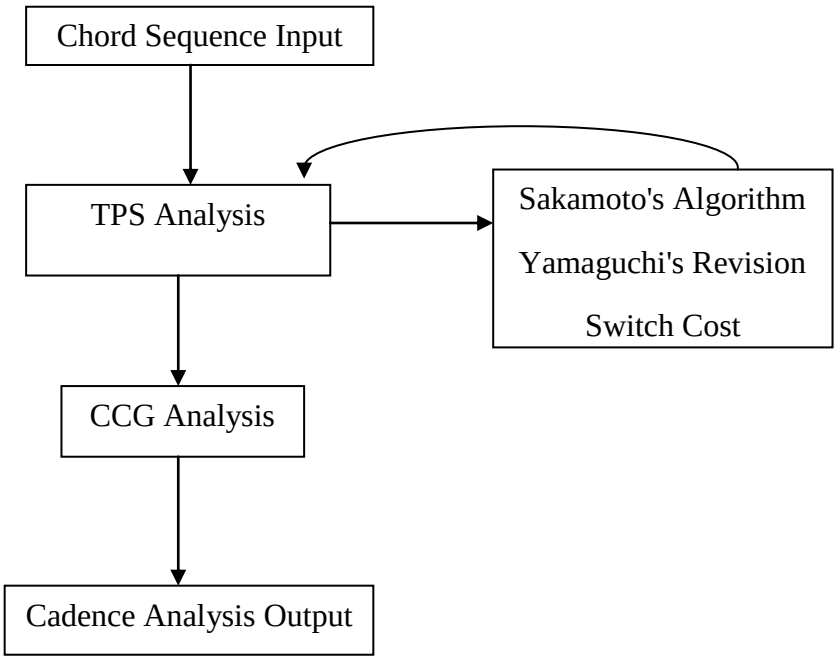


Fig. 25 Diagram of parsing with TPS switch cost

With the switch cost in place, any time there is movement from one key to another, a check is performed to see if the two keys are of the same mode. Thus, in the formula originally given by

Sakamoto, an extra 0.5 will always be added when R1 and R2 are not both minor or both major. Sakamoto's formula has been re-printed here so as to be more easily accessible.

$$\delta(x,y) = \begin{cases} region(x,y) + chord(x,y) + basicspace(x,y) & \text{Close Key} \\ \min\{\delta(x, \mathrm{I}/R_1) + \Delta(R_1, R_2) + \delta(\mathrm{I}/R_2, y) | R_1 \in \mathrm{P}(x),\ R_2 \in \mathrm{P}(y)\} & \text{Remote Key} \end{cases}$$

Since all movement when working with remote keys goes through the tonic chords of pivot regions, it now becomes crucial to check if the pivot region R2 is major or minor based on what R1 is. Thus, the small 0.5 cost can actually become quite large if movement is between very distant chords that requires many switches between major and minor keys.

The merits of the CCG feedback loop and TPS switch cost needed to be tested. As the goal of this project was to modify TPS, the TPS switch cost was tested more rigorously than the CCG condition.

5. Experiment Methodology and Results

As was mentioned earlier in this paper, the most important facet of this research is the addition of the major-minor switch cost to TPS theory. A CCG feedback mechanism was added to the parser, but this addition was more minor than the theoretical contribution of the switch cost. Therefore, the CCG testing will be described first, and the TPS testing will be described later. As the parser is tailored to jazz music and was tested using a set of jazz compositions, testing for this project was done in a similar fashion. Some non-jazz pieces were used, but the main genre for testing was jazz. While the previous tested pieces came from a jazz workshop book [7], the current project took songs from the Weimar Jazz Database, known as Jazzomat [8]. 20 songs were taken from the database, and a handful of other songs from popular music were chosen for testing the parser and the switch cost.

5.1 Experiment

Initially, the experiment was to involve all 299 songs given on the Jazzomat website, as this would give sufficient robustness to testing the switch cost. However, as mentioned previously, the input method in the parser is neither intuitive nor convenient, so improving upon the method would be highly beneficial for both this research as well as future projects using Fukunari's parser. At first, it was decided that the MIDI files given on the Jazzomat database would be perfect inputs for a new system; however, said MIDI files were only the melody lines with no chords present. As chords serve as the basic unit for analysis in the current parser, a melody line alone cannot be analyzed as it does not provide sufficient data.

Following this problem, it was decided that the PDF files given on the Jazzomat Database website could be converted into MusicXML files and could then be passed to a modified parser. The same problem as with the MIDI files again presented itself due to the fact that the PDF files have no written chords, only annotations.

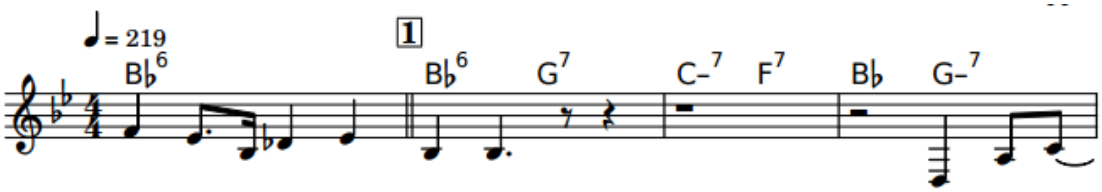


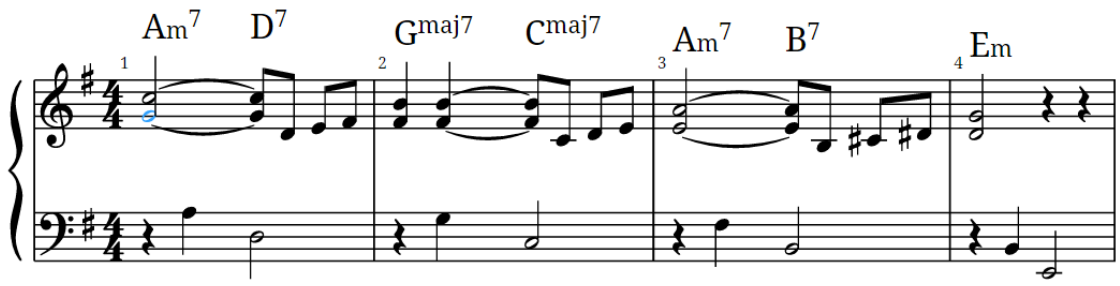
Fig. 26 Excerpt from "Anthropology" by Art Pepper as given on Jazzomat

As can be seen in Figure 26, only the melody is directly printed, with chords being given above the clefs. After noting this setback with both the Jazzomat MIDI and PDF files, it was decided that the input method would remain the same and the scope of experimentation would be reduced from all 299 songs to 20 or 30 songs. Giving chord input from all 299 songs by hand was

determined to be too inefficient, while 20 or 30 songs with chords entered by hand was decided to be powerful enough as a dataset.

The 0.5 switch cost, at this point into the project, functioned as a "magic number", essentially, so it was necessary to test to see if the exact value of the switch cost or merely the existence of the switch cost itself truly mattered. At first, machine learning appeared to be a promising method by which the precise value of the switch cost could be quantified. Unfortunately, due to both time constraints as well as the lack of a setup for machine learning, this idea was set aside, and a different method of testing was decided upon. In order to test the switch cost, pieces in minor keys with pivot chords were to be tested, and the value of the switch cost was to be adjusted while testing. The tested values were set at 0.1, 0.4, 0.5, 0.6, and 0.9. If the value of the switch cost has an impact on the analysis, then pivot chords between major and minor keys should be analyzed differently, and the switch from major to minor (or vice versa) should be analyzed as occurring in a different place. If, on the other hand, the existence of the switch cost is the only important factor for analysis, then parser performance should be more accurate without changing the location of pivots when the switch value is changed. If the switch cost's existence alone matters, then this would potentially resolve the problem in TPS. Testing commenced thusly, and its results shall be provided in the following pages. Sheet music will be provided with the parser input and output. So as to save space, only some of the tested pieces and their analyses will be shown.

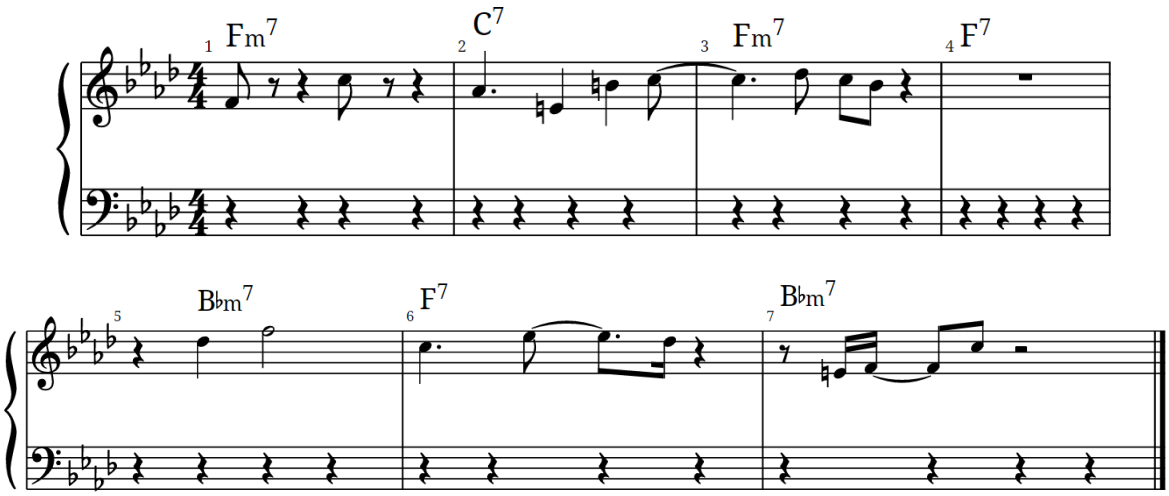
Autumn Leaves by Johnny Mercer




```
input(Chord progression) [ Am7 D7 GM7 CM7 G♭m7(♭5) B7 Em7 ]

output: ⅡMm7 in GMajor Sub Dominant
output: V7 in GMajor Dominant
output: IM7 in GMajor Tonic
output: ♭VIM7 in EMinor Sub Dominant Minor
output: Ⅱm7(♭5) in EMinor Sub Dominant Minor
output: V7 in EMinor Dominant
output: Im7 in EMinor Tonic

output: ⅡMm7 in GMajor Sub Dominant
output: V7 in GMajor Dominant
output: IM7 in GMajor Tonic
output: ♭VIM7 in EMinor Sub Dominant Minor
output: Ⅱm7(♭5) in EMinor Related Second 7th
output: V7 in EMinor Dominant
output: Im7 in EMinor Tonic

output: ⅡMm7 in GMajor Related Second 7th
output: V7 in GMajor Dominant
output: IM7 in GMajor Tonic
output: ♭VIM7 in EMinor Sub Dominant Minor
output: Ⅱm7(♭5) in EMinor Sub Dominant Minor
output: V7 in EMinor Dominant
output: Im7 in EMinor Tonic
```

Before the switch cost and CCG feedback modifications were implemented, the parser analyzed the entire sequence except the B7 chord as being in G Major. Now, the parser correctly analyzes the cadence.

Airegin by Miles Davis




```
input(Chord progression) [ Fm7 C7 Fm7 F7 B♭m7 F7 B♭m7 ]


output: Im7 in FMinor Tonic
output: V7 in FMinor Dominant
output: Im7 in FMinor Tonic
output: V7 in B♭Minor Dominant
output: Im7 in B♭Minor Tonic
output: V7 in B♭Minor Dominant
output: Im7 in B♭Minor Tonic
```

Here, analysis again picks up the minor keys. Also, it detects the pivot F7 chord to shift the key from F minor to B ♭ minor. The parser originally only could analyze the first 3 chords as being in F minor, but could not analyze the remaining chords, missing the key switch entirely.

Blue Mode by J.J. Johnson

For this piece, no sheet music will be given as the progression is long, and the analysis only partially succeeds.

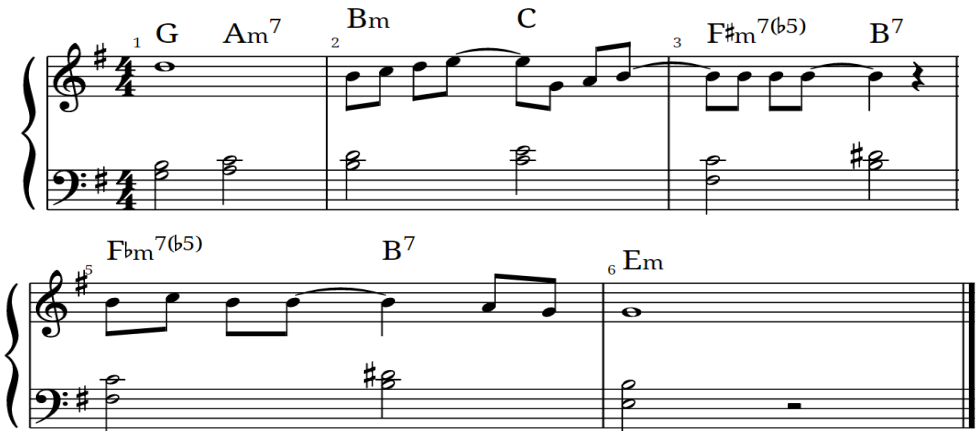
```
input(Chord progression) [ B♭7 Am7 D7 Gm7 Fm7 B♭7 E♭7 D♭7 G7 Cm7 C7 F7 B♭7
G7 Cm7 F7 ]

output: Ⅴ7 in E♭Major Dominant
output: ♭Ⅴm7 in E♭Major Related Second 7th
output: Ⅶ7 in E♭Major Secondary Dominant
output: Ⅲm7 in E♭Major Tonic
output: Ⅱm7 in E♭Major Sub Dominant
output: Ⅴ7 in E♭Major Dominant
output: Ⅰ7 in E♭Major Tonic
output: ♭Ⅱ7 in CMinor Dominant
output: Ⅴ7 in CMinor Dominant
output: Ⅰm7 in CMinor Tonic

output: Ⅴ7 in E♭Major Dominant
output: ♭Ⅴm7 in E♭Major Related Second 7th
output: Ⅶ7 in E♭Major Secondary Dominant
output: Ⅲm7 in E♭Major Tonic
output: Ⅱm7 in E♭Major Sub Dominant
output: Ⅴ7 in E♭Major Dominant
output: Ⅰ7 in E♭Major Tonic

output: Ⅴ7 in E♭Major Dominant
output: ♭Ⅴm7 in E♭Major Related Second 7th
output: Ⅶ7 in E♭Major Secondary Dominant
output: Ⅲm7 in E♭Major Tonic
output: Ⅱm7 in E♭Major Related Second 7th
output: Ⅴ7 in E♭Major Dominant
output: Ⅰ7 in E♭Major Tonic
```

For this cadence, the parser can analyze the first part successfully, but cannot go any further. The CCG rules cannot find a valid cadence, nor can the TPS calculations find a reliable path between the remaining chords.

As testing continued, it became clear that the CCG conditions that had been implemented previously were very strong and could possibly be hindering real analysis of the TPS switch cost. The fact that CCG can call for a switch to the relative major or minor after the TPS section could mean that the TPS switch cost is unnecessary with the CCG constraint turned on. Therefore, it was important to separate the effects of the two sections.

With the TPS cost off and the CCG constraint on, the parser performed as described previously. With the CCG constraint off and the TPS cost on, the previous results were replicated. As will be seen in the following pages, however, some discrepancies occurred when the parser's analyses were compared to musicological analyses . These discrepancies meant that certain outputs were not perfectly correct, but the parser was still able to detect keys correctly. Since it was suspected that the CCG constraint was too powerful, more focus was now directed towards the TPS switch cost changes and the outputs yielded from the varying costs.

Here, There, and Everywhere by The Beatles



The analysis below is the correct output. The melody switches from G Major to E minor, and the C Major chord is the pivot chord.

```
input(Chord progression) [ G Am7 Bm C G♭m7(♭5) B7 G♭m7(♭5) B7 Em ]

output: Ⅰ in GMajor Tonic
output: Ⅱm7 in GMajor Sub Dominant
output: Ⅲm in GMajor Tonic
output: Ⅳ in GMajor Sub Dominant
output: Ⅶm7(♭5) in GMajor Dominant
output: Ⅴ7 in EMinor Dominant
output: Ⅱm7(♭5) in EMinor Sub Dominant Minor
output: Ⅴ7 in EMinor Dominant
output: Ⅰm in EMinor Tonic

output: Ⅰ in GMajor Tonic
output: Ⅱm7 in GMajor Sub Dominant
output: Ⅲm in GMajor Tonic
output: Ⅳ in GMajor Sub Dominant
output: Ⅶm7(♭5) in GMajor Dominant
output: Ⅴ7 in EMinor Dominant
output: Ⅱm7(♭5) in EMinor Related Second 7th
output: Ⅴ7 in EMinor Dominant
output: Ⅰm in EMinor Tonic
```

Here, the parser slightly misses the correct pivot chord, and instead detects the F# $\flat_5$ chord as being the pivot. The parser does, however, detect the keys correctly. With the CCG constraint on, the output matches the true analysis.

5.2 Re-Harmonization Testing

After this round of testing, focus was shifted to re-harmonizing tested pieces to see if the parser would be able to recognize valid reconfigurations of input chord progressions. In music, re-harmonization is an important technique whereby an existing harmonic line – the chord progression – is changed while the melody itself remains untouched. Through these changes, the "flavor" of the music changes in turn. In jazz, re-harmonization is a particularly important concept. In live jazz performances, artists will often perform one of a number of jazz standards – jazz compositions that are well known and frequently performed by jazz musicians – and change the chord progressions. Oftentimes, these changes are done as improvisations and serve to show that the musician who is performing is a true jazz virtuoso. To re-harmonize a piece, chords of a certain function are exchanged with different chords fulfilling the same function. For example, in C Major, the C_7 chord has a tonic function, so $Amin_7$, which also has a tonic function, can be used as a substitute. While this sort of substitution does not drastically change the piece being performed, it does slightly change how the piece sounds; the melody remains intact, while the chord harmonies are altered.

In jazz specifically, tritone substitution is frequently employed to re-harmonize pieces. Tritone substitution involves substituting the dominant V_7 chord with another chord that is three whole steps above the root of the V_7. In C Major, G_7 is the dominant 7^{th}, so the tritone substitution is $C\#_7$ ($D\flat_7$). This particular substitution is shown in Figure 27.

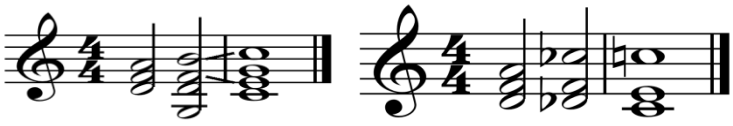
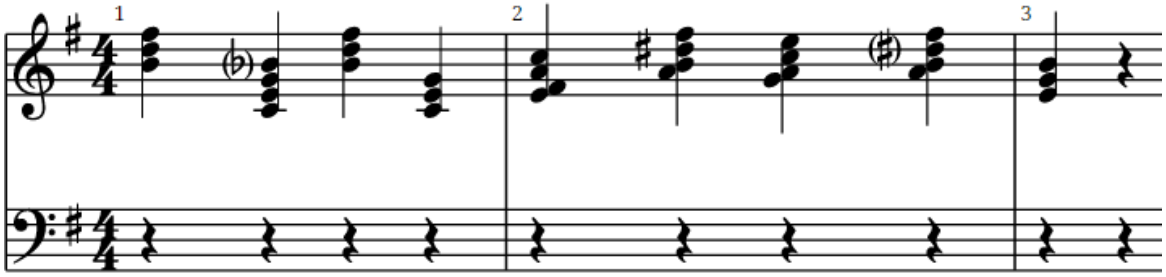


Fig. 27 Tritone substitution

As re-harmonization is such an important part of jazz performance and composition, understanding the process of re-harmonization is crucial to analysis. Given that re-harmonization is only a "re-flavoring" of a given piece rather than a complete re-writing, then pieces that have been re-harmonized with new chords should still be recognized as having valid cadences if the switch cost works properly. For this testing, the CCG constraint was mainly left off, although it was checked to make sure that analysis was correct with the CCG constraint on. For testing, the same pieces as used previously were used again and were re-harmonized. Re-harmonization was performed manually using the techniques of chord substitution, key changing, and tritone substitution. These techniques were also combined so that a piece was re-harmonized and transposed into a different key. As with the previous testing, only some examples are shown in the following pages.

Here, There, and Everywhere Re-harmonization



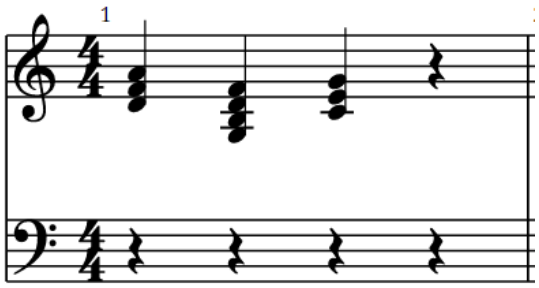
Here, the 1st, 2nd, and 7th chords have been changed to Bm, C_7, and Am_7, respectively.

```
input(Chord progression) [ Bm C7 Bm C G♭m7(♭5) B7 Am7 B7 Em ]

output: ⅢIIIm in GMajor Tonic
output: ⅣIV7 in GMajor Sub Dominant
output: ⅢIIIm in GMajor Tonic
output: ⅣIV in GMajor Sub Dominant
output: ⅦVIIm7(♭5) in GMajor Dominant
output: ⅤV7 in EMinor Dominant
output: ⅣIVm7 in EMinor Sub Dominant Minor
output: ⅤV7 in EMinor Dominant
output: ⅠIm in EMinor Tonic
```

When compared to the original harmony, the analysis should look familiar. While the pivot is still slightly incorrect, it is clear that the parser found the same general cadence. This shows that the switch cost is working correctly for both the original and the re-harmonized melody.

Tritone re-harmonization, however, is not recognized.

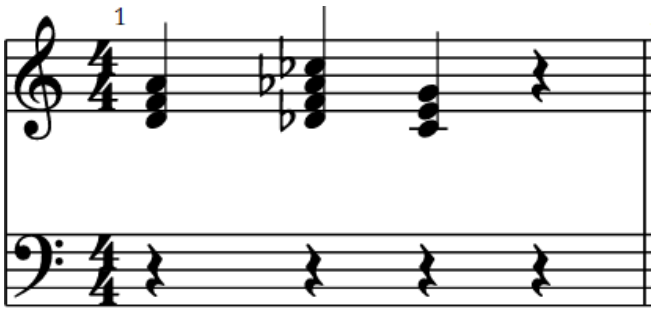
Consider the example cadence given:



This is a classic ii-V-I turnaround, and the parser analyzes it as such.

```
input(Chord progression) [ Dm G7 C ]

output: Ⅱm in CMajor Sub Dominant
output: Ⅴ7 in CMajor Dominant
output: Ⅰ  in CMajor Tonic
```

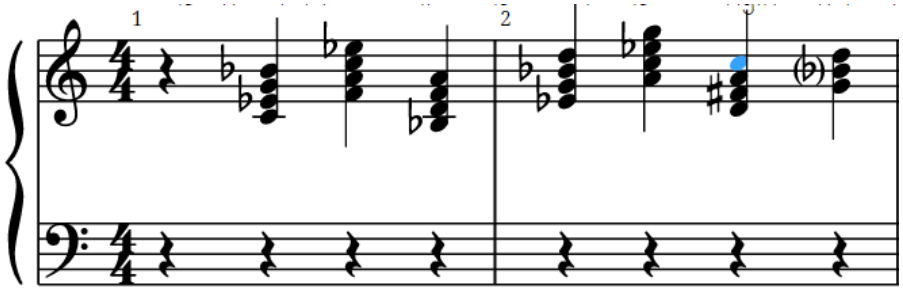
When the piece is re-harmonized using a tritone substitution, the cadence becomes:



While not necessarily common, such a cadence is utilized in jazz music and should be analyzed as being a valid cadence. The parser, however, does not recognize it and gives no output.

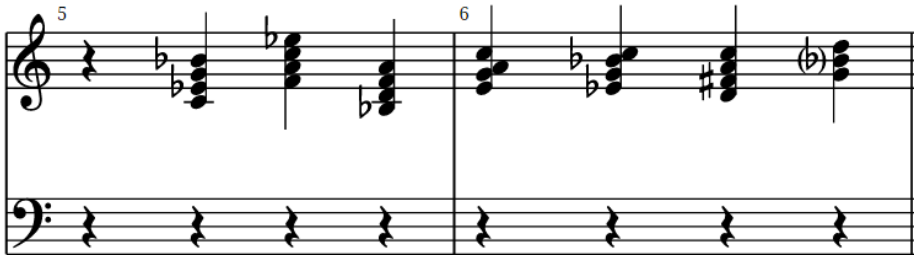
```
Chord progression [ Dm D♭7 C ]

input: Dm

input: D♭7

input: C
```

Additionally, it became desirable to quantify the distances generated by re-harmonizations and invalid re-harmonizations. In the following modulation



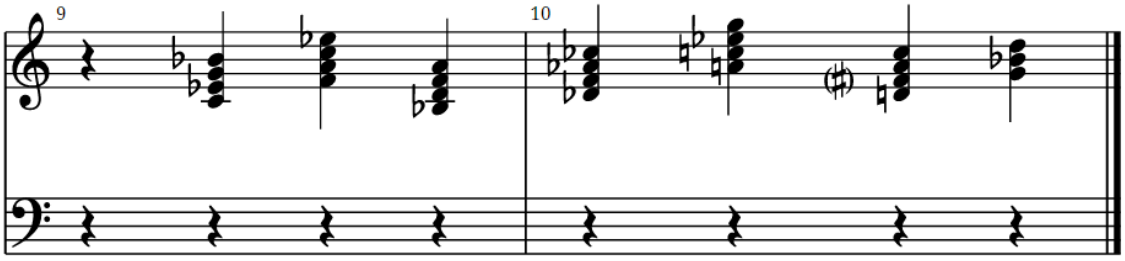
the distance from the 3rd chord to the 5th is 19.5.

When re-harmonizing the cadence in the following manner



the distance becomes 26.

When incorrectly re-harmonizing the cadence to include a completely unrelated chord



the distance from the 3rd chord to the 5th becomes 44.

Similar incorrect re-harmonizations also incur such large distances, explaining why the parser, and also the key finding algorithm, will not recognize them. When re-harmonized distances are closer to the original distances, re-harmonization will be successfully analyzed. When the new distance is not close to the old distance, analysis will fail and the resulting music will not be

aesthetically pleasing for the listener. Similarly, when comparing the distances in the re-harmonization of *Here, There, and Everywhere*, the first three chords in the original progression cover a distance of 16, while the distance covered by the three in the re-harmonization is 14. As re-harmonization is a changing of the chord progression without entirely changing the piece, it is reasonable to think that re-harmonized distances should be close to the original distances. The parser with the switch cost implemented is able to find these new distances, re-analyze the piece, and then give the analysis as output. Without the switch cost, both analysis of the original progression and the re-harmonization would be incorrect.

With this work on re-harmonization, it became necessary to know if TPS distances matched listener's perceptions of musical quality. Thus, a survey was conducted to determine if close distances were more preferable from listeners' points of view. 15 MIDI files were constructed consisting of three re-harmonizations of different chord progressions. Two of the re-harmonizations were close to the original progression, while the third was very far. If listeners are able to discern musical distances in the manner listed in TPS, then they should prefer one of the close re-harmonizations. Additionally, within the two close re-harmonizations, they should prefer the closest one.

5.3 Results

For the first set of testing on the parser, the results were clear: both the CCG constraint and the TPS switch cost improved parser accuracy and power. Where once the parser was unable to identify cadences in minor keys, it now can. Both the CCG constraint and the TPS switch cost improve accuracy, but the CCG constraint is more accurate. As was demonstrated with the *Here, There, and Everywhere* example, the CCG constraint leads to more accurate pivot chord locations. The TPS switch cost still finds the correct keys but has slightly more trouble with the

pivot chord analysis. This inaccuracy is unfortunate, but it still is a marked improvement over both the previous parser as well as the Granroth-Wilding parser mentioned earlier in this paper.

Additionally, the results of the re-harmonization investigation showed that the parser can recognize licit re-harmonizations. While the re-harmonizations constructed in this paper were not created by a professional musician or musicologist, the calculations performed on the re-harmonization distances were done in accordance with TPS theory. In terms of TPS theory, re-hmaronized cadences should cover similar distances to those in the original harmony. From the survey used to test this, the following table was made using the participants' responses. The columns are the re-harmonizations, with the "Far Cadence" column being the most distant re-harmonization in terms of TPS calculations. The "Tritone" column marks re-harmonizations that were performed substituting the dominant V chord for a tritone. The "Close Cadence" column marks re-harmonizations that had minimal distance changes when TPS distances were calculated. In each row, the number of participants choosing a particular cadence is marked.

	Tritone	Close Cadence	Far Cadence
Melody 1	1	3	0
Melody 2	2	2	0
Melody 3	2	1	1
Melody 4	2	2	0
Melody 5	0	2	2
Melody 6	1	3	0
Melody 7	3	1	0
Melody 8	2	2	0
Melody 9	0	3	1
Melody 10	0	3	1
Melody 11	1	2	1
Melody 12	1	3	0
Melody 13	1	3	0
Melody 14	0	2	2
Melody 15	4	0	0
Total	**20**	**32**	**8**

As can be seen, participants almost never chose the far cadence and favored the close cadence over the tritone. These results match TPS distances and show that listeners' perceptions of musical space functions similarly to how TPS describes. The size of this survey was fairly limited, so it may be prudent in the future to increase the number of participants. In this project, time was fairly limited, so a large number of listeners could not be surveyed.

6. Discussion and Future Directions

As the results have been laid out in the previous section, it is now important to discuss their meaning as well as the importance of the research that has been detailed throughout this paper. Since a survey was taken, it is also important to discuss the possible meanings of the collected answers. These subjects will be covered in the following subsections.

6.1 Discussion of Results

For the most part, the results yielded by this research are self-explanatory. The CCG constraint and TPS switch cost both fulfilled their intended purposes and greatly improved the parser. The parser can now recognize minor chord progressions more successfully and can recognize re-harmonizations of such minor progressions. The parser is also able to discriminate "good" re-harmonizations from "bad" re-harmonizations and will not give output for substitutions that are musically implausible. The switch cost does not fully resolve the problems with the parser, nor the problems with TPS as a theory. As was shown above, analysis with the switch cost can contain inaccuracies, so these need to be improved.

The calculation of the spatial distances covered in re-harmonization shows that re-harmonizations that are closer spatially to the original melody are preferable to those that are distant. This matches the hypothesis outlined earlier and, in a way, continues to validate TPS as an accurate model for representing how listeners perceive music. The survey reinforces this

finding, as listeners almost never chose very distant re-harmonizations, and preferred the closest re-harmonization provided. The 8 selections of the far cadence could be due to the experimenter's lack of expertise in re-harmonization: it was mentioned by a listener that the voice-leading in the re-harmonized progressions could confuse perceptions. Future similar investigations should try to use better re-harmonizations that are constructed by those more familiar with re-harmonization processes. Regardless, the results do support the hypothesis proposed earlier and show that listeners are certainly able to recognize the distances present in TPS. The fact that tritones were chosen relatively frequently (20 times out of 60 possibilities), is interesting due to the fact that tritone substitution is frequently employed in jazz music. In TPS, tritone substitution is somewhat distant (but not as distant as the far cadences constructed in this survey), so it could be the case that TPS is not particularly suitable for jazz music. Indeed, TPS is designed around classic music and classical music theories. As jazz changes many features of classic music, TPS may need to be further modified to better suit jazz music.

Furthermore, the CCG constraint proved to be slightly too strong. The constraint looking for Dominant→Tonic motion seems good at first, but it may actually be post hoc in nature. In an ideal world, the parser should be able to find the cadences of all pieces without being forced to find certain types of motion. The Dominant→Tonic rule was written after observing that many pieces followed such a rule. Applying this constraint is not optimal and merely represents a sort of stop-gap measure to temporarily improve the parser's accuracy. While adding these constraints may seem to be useful, addition can actually decreases the parser's analytical power, as eventually the parser will just regurgitate a constraint rather than actually analyzing the given chord progression.

6.2 Contribution

The research described in this paper has thus contributed to TPS as a theory directly, the parser itself, and future work on re-harmonization. The addition of a switch cost has proven valuable for analysis of progressions in minor keys, and therefore holds merit as a theoretical contribution to TPS. By using the switch cost, TPS itself can now better distinguish between major and minor keys. As was mentioned in Section 2 on TPS, there was previously no distinction between major and minor keys in terms of regional distance due to the fact that the diatonic sets of relative major and minor keys were the same. Now, with the switch cost, such distinctions can be made, and these distinctions assist distance calculations, thus leading to more accurate and efficient analysis of musical compositions. With these improvements also comes an extra, more abstract contribution to work on automated musical arrangement. As of now, the parser is only able to analyze and give output relating to input chord progressions. This step, however, is quite important. Work on the parser has proved that the parser can recognize valid cadences and re-harmonizations of said cadences, so the next logical step is generating new progressions. When generation is automated, it is desirable to have the output be aesthetically pleasing in a musical sense. By creating a robust parser that can find valid progressions and reject invalid progressions, the step to automated arrangement becomes much smaller. Moreover, the addition of the switch cost means that valid minor arrangements can now be recognized, opening the door to future generation of such chord progressions in minor keys. While automated arrangement and composition was not the direct focus or contribution of this particular research, the implications of the results yielded by this research contribute significantly to future work on this subject. It would be beneficial, in fact, to develop another

addition to the parser that could start generating musical output in addition to the already-developed analytical section.

While this research was successful in achieving its goals and contributing to the established body of musical research, there were some limitations and there remain many tasks that should be done in the future. The following two subsections go into further detail about these two topics.

6.3 Limitations

In its current state, the parser is not perfect. The slight inaccuracies in locating pivot chords when the CCG constraint is turned off is not desirable and should be fixed as soon as possible. The cause of this inaccuracy seems to stem directly from TPS itself, so more changes to TPS should be made in the future. Additionally, testing was done via hand-entered chord progressions. The parser's strength has been demonstrated at this point, but it would be good, for the purpose of replicating the results presented here, to do further testing on a larger set of data. To do this, remodeling the parser's input system to allow MusicXML or MIDI files as input would be highly beneficial. This would not only allow for easier input, but would also simplify the system so that entering chords is less error-prone for future users.

The work on re-harmonization was also useful in that it gives more mathematical insight into what a valid re-harmonization truly is, but the re-harmonizer in this case was a researcher, not a musicologist. Thus, the quality of the re-harmonizations cannot be verified and may be unsound. While the survey did show a listener preference matching the hypothetical preference for re-harmonizations which are close in TPS, the survey was quite small. Only having time to survey four people means that the results yielded by the survey conducted in this project are not as powerful as would be preferable. To rectify these issues, further testing using a more highly

trained re-harmonizer and a larger subject pool for the survey would yield more significant results. The current parser is also set up to analyze jazz music. While this certainly pleases jazz aficionados, there are many other types of music that people listen to, so expanding the parser to allow analysis of other genres of music would remove another one of this parser's limitations. If these limitations can all be fixed, the parser would be greatly improved, as would TPS. It would greatly behoove future researchers in this area to make the changes listed here so as to further develop this field of musical analysis and composition.

6.4 Future Work

Keeping in line with the limitations mentioned in the preceding subsection, there exist a variety of directions that those who are interested in this current research can undertake. Perhaps the most fashionable at the moment would be the addition of a generative section to the parser. As has been demonstrated throughout this paper and in the previous paper regarding this particular parser, the current version of the parser can only analyze, not generate. As technology moves ever closer to developing sentient artificial intelligence, it is also sensible to make musical AI that can compose and arrange music. Being able to recognize validity in music is an important first step, so using the parser developed here would be a logical way to start the process of building a generative system. This work would not be easy, but it would make a large contribution to both the field of AI and musical science. Along these lines, it would be useful to also consider modal jazz. Unlike Bebop jazz, modal jazz does not rely heavily on the concept of a key – the music switches through the modes of a given scale. This style became popular after bebop jazz and was pioneered by such artists as Miles Davis and John Coltrane. Since this type of music does not focus on keys, TPS and the TPS switch cost developed in this paper may not

apply to analysis of modal jazz. It would be interesting to see how such music could be analyzed and arranged via machine.

Another interesting topic for further research is automated re-harmonization. Initially, it was the goal of this project to developed an automated re-harmonization function and add it to the parser, but the aim changed as time was limited. Using the I-R Model, it would be interesting to construct re-harmonizations that match and do not match the structure of the original piece as analyzed by the I-R Model. Moreover, this would allow the theory that closer re-harmonizations are preferable to more distant re-harmonizations to be tested more rigorously. For this research, manually re-harmonizing was time-consuming, so automated re-harmonization would expedite the process greatly.

The future works proposed here are only some of the projects that can be undertaken to continue the research established here. It will be interesting to see what direction is taken by future researchers and investigators. The contributions established in this paper by this research should greatly benefit future musical research and will hopefully lead to revision of and additions to extant musical theories. Music has by no means been solved or figured out, so all research with the purpose of demystifying music is both beneficial and highly interesting.

References

[1] Lerdahl, F. (2001). *Tonal pitch space*. Oxford: Oxford University Press.

[2] Narmour, E. (1992). The analysis and cognition of melodic complexity: the implication-realization model. Chicago: University of Chicago Press.

[3] Shouki Sakamoto, Sean Arn, Masaki Matsubara, Satoshi Tojo, Harmonic analysis based on Tonal Pitch Space, KSE2016

[4] Naohiko Yamaguchi, Noboru Sugamura, Improving TPS to Tackle Non Key Constituent Notes, Information Processing Society of Japan (IPSJ2011), Vol.2011-MUS-89 No.10, Feb 2011.

[5] Mark Granroth-Wilding, Mark Steedman, Robust Parser-Interpreter for Jazz Chord Sequence, Journal of New Music Research, Volume 43, Issue 4, 2014.

[6] Takeshi Fukunari, Sean Arn, Satoshi Tojo, CCG Analyzer with Tonal Pitch Space for Non-Classical Chords, KSE2016.

[7] Oyama, D. (1980). *Jazz Theory Workshop*. Musashino Music Institute Publishing Department.

[8] The Jazzomat Research Project. (n.d.). Retrieved January 31, 2017, from http://jazzomat.hfm-weimar.de/dbformat/dbcontent.html

Appendix

The code used to modify the parser's CCG section is given below.

```
	private static void toRelative(int s, int g){
		for(int i=s;i<g+1;i++)
		{
			if(AnalyzeTPS.ChordOutput[i][3].equals("M"))
			{
				AnalyzeTPS.ChordOutput[i][2] =
(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])+9)+"";
				if(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])>11)

	AnalyzeTPS.ChordOutput[i][2]=(Integer.parseInt(AnalyzeTPS.ChordOutput[i
][2])-12)+"";
				AnalyzeTPS.ChordOutput[i][3] = "m";
			}
			else
			{
				AnalyzeTPS.ChordOutput[i][2] =
(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])+3)+"";
				if(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])>11)

	AnalyzeTPS.ChordOutput[i][2]=(Integer.parseInt(AnalyzeTPS.ChordOutput[i
][2])-12)+"";
				AnalyzeTPS.ChordOutput[i][3] = "M";
			}
		}
	}

	private static void unify(int s, int g)//btwn s & g everything can be
relative major/minor, make everything the major
	{
		for(int i=s;i<g+1;i++)
		{
			if(AnalyzeTPS.ChordOutput[i][3].equals("m"))
			{
				AnalyzeTPS.ChordOutput[i][2] =
(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])+3)+"";
				if(Integer.parseInt(AnalyzeTPS.ChordOutput[i][2])>11)

	AnalyzeTPS.ChordOutput[i][2]=(Integer.parseInt(AnalyzeTPS.ChordOutput[i
][2])-12)+"";
				AnalyzeTPS.ChordOutput[i][3] = "M";
			}
		}
	}
```

```
private static void chunkCCG(int s, int g, String[][] LM, String[][] Lm)
	{

	//with pivots 1st & last key may be different.
	//keep them in a corner and change them to the same as the rest for the
analysis.
		String firstkey = AnalyzeTPS.ChordOutput[s][2];
		String firstmode = AnalyzeTPS.ChordOutput[s][3];
		String lastkey = AnalyzeTPS.ChordOutput[g][2];
		String lastmode = AnalyzeTPS.ChordOutput[g][3];

		AnalyzeTPS.ChordOutput[s][2] = AnalyzeTPS.ChordOutput[s+1][2];
		AnalyzeTPS.ChordOutput[s][3] = AnalyzeTPS.ChordOutput[s+1][3];
		AnalyzeTPS.ChordOutput[g][2] = AnalyzeTPS.ChordOutput[g-1][2];
		AnalyzeTPS.ChordOutput[g][3] = AnalyzeTPS.ChordOutput[g-1][3];


//			if(AnalyzeTPS.ChordOutput[s][3].equals("m"))
//				toRelative(s,g);
		unify(s,g);


		int froms = s;
		int fromM = lastTonic(s,g,LM)+1;

		toRelative(s,g);
		int fromm = lastTonic(s,g,Lm)+1;
		toRelative(s,g);

		while(froms<g)
		{
			if(fromM == g+1 && fromm == g+1)
				flag = false;

			int next = nextTonic(froms,fromm,fromM,g,LM,Lm);

			if(next==g+1)
			{
				froms = g+1;
				flag = false;
			}
			else
			{
				if(AnalyzeTPS.ChordOutput[froms][3].equals("M"))
				{
					int last=lastTonic(next,g,LM);
					innerCCG(froms,last,LM);

					if(flag)
					{
						froms = last+1;
						fromM = last+1;
						fromm = last+1;
					}
					else
					{
						if(fromm<fromM)
```

```
                                    fromm = fromM;
                              fromM = last+1;
                        }
                  }

                  else
                  {
                        int last = lastTonic(next,g,Lm);
                        innerCCG(froms,last,Lm);

                        if(flag)
                        {
                              froms = last+1;
                              fromM = last+1;
                              fromm = last+1;
                              if(last<g)
                                    toRelative(last+1,g);
                        }
                        else
                        {
                              if(fromM<fromm+1)
                                    fromM = fromm+1;
                              fromm = last+1;
                              toRelative(froms,g);
                        }
                  }
            }
      }

      AnalyzeTPS.ChordOutput[s][2] = firstkey;
      AnalyzeTPS.ChordOutput[s][3] = firstmode;
      AnalyzeTPS.ChordOutput[g][2] = lastkey;
      AnalyzeTPS.ChordOutput[g][3] = lastmode;

}
```

```
	//returns last consecutive tonic starting at fromT (fromT-1 if fromT
isn't a tonic)
	private static int lastTonic(int fromT, int g, String[][]L)
	{
		String[][] Degree = degreeNameConversion(fromT,g);
		boolean tonic_flag = isTonic(fromT,fromT, Degree, L);
		int last=fromT;
		while(last<g && tonic_flag)
		{
			last++;
			tonic_flag = isTonic(fromT,last, Degree, L);
		}
		if(!tonic_flag)
			return last-1;
		else
			return g;
	}
```

```
	//returns first tonic in major or minor. if minor, turns s->g into
minor.
	private static int nextTonic(int s, int fromMin, int fromMaj, int g,
String[][]LM, String[][]Lm)
	{
		int next=g+1;
		int nextm = 0;
		if(fromMaj<g+1)
			next = firstTonic(fromMaj,g,LM);

		if(fromMin<g+1)
		{
			toRelative(s,g);
			if(next<g+1)
				nextm = firstTonic(fromMin,next,Lm);
			else
				nextm = firstTonic(fromMin,g,Lm);
			if(nextm<next)
				next=nextm;
			else
				toRelative(s,g);
		}
		return next;
	}

	//returns first tonic according to L (LM or Lm) between fromT and g. if
none, returns g+1.
	private static int firstTonic(int fromT, int g, String[][] L)
	{
		String[][] Degree = degreeNameConversion(fromT,g);
		int tonic = fromT;
		boolean tonic_flag = false;
		//look for first tonic in major
		while(!tonic_flag && tonic<g+1)
		{
			tonic_flag = isTonic(fromT, tonic, Degree, L);
			tonic++;
		}
		if(tonic_flag)
			tonic--;
		return tonic;
	}

	//is chord (#chord in Degree) a tonic ?
	private static boolean isTonic(int s, int chord, String[][] Degree,
String[][] L)
	{
		boolean tonic_flag = false;
		int i = 0;
		//check if it's a tonic
		while(L[i][4].equals("Tonic") && !tonic_flag)
		{
			if(Comparison(chord-s,i,Degree,L))
				tonic_flag = true;
			else
				i++;
		}
```

```
            return tonic_flag;
      }

      private static boolean isValid(String[][] Annotation)
      {
            boolean cadence_flag = false;
            int i = 0;
            while(i<Annotation.length-1 && !cadence_flag)
            {
                  cadence_flag = ((Annotation[i][4].equals("Dominant")) ||
Annotation[i][4].equals("Sub Dominant"))
                                                &&
Annotation[i+1][4].equals("Tonic");
                  i++;
            }
            return cadence_flag;
      }
```