Towards Musical Analysis Tools

Curt Hill and Sara Hagen
Valley City State University
Valley City, ND 58072
Curt.Hill@vcsu.edu Sara.Hagen@vcsu.edu

Abstract.

The golden age of musical analysis is now dawning with computer assistance. Traditional analysis of music is a tedious and time consuming process for even the experienced analyst. There are many well known rules for this, but music is an art where a balance must exist between the expected and surprises.

The raw data for such a study abounds, but finding the preferred format is an issue. The data can generally found in two different classes. The first class is something that could be performed. The second class includes the file formats of numerous score producing programs. There are some tradeoffs for either class of data.

Several experimental programs are described. The first detects composition errors. The second recognizes chords and even this is not trivial. The third tabulates chord sequence pairs within collections of works to give statistical insight into the important topic of chord progressions.

Introduction

The entire process of publishing has been revolutionized in recent memory. The time honored method was to write out the manuscript in longhand, have it typed and then typeset. Each copy introduced errors that had to be found and removed. The arrival of word processors and desktop publishing in the 1970s and 1980s streamlined the process considerably. A similar but less well known series of events has done similar things to music composition and publication. The advent of score production programs has saved countless hours in the production of publishable music. In about the same time frame Musical Instrument Digital Interface (MIDI) and a plethora of MIDI-compatible musical instruments has made it possible to play the score directly from the program.

The area of analysis of music is just beginning to benefit from these advances. Music analysis is a well known process of decomposing music into recognizable pieces. Just as a sentence may be analyzed to find its subject and verb so may music be analogously examined. Music analysis is looking for chord sequences, cadences, tonicisations and modulations rather than for nouns, verbs and adjectives. Although the low order techniques are quite different as the subject matter is different, the goal is the same – to find the structure placed there by its author.

Music and grammar analysis have something else in common: both require considerable knowledge of the subject domain and are time consuming and tedious. Tasks that are knowledge based and tedious are exactly the kind to which computer application is most suitable. Therefore, we will consider two important topics: the form of the data on which to begin the analysis and some types of analysis that have been accomplished. The former will focus on the kinds of data available and conversion to usable format. The latter will focus on some specific analyses that have been done as well as what could be done. We will also attempt at least an informal definition of the technical musical terms as we consider them.

Form of Musical Data

Music exists in a number of data formats that are readily available [Castan,2008], but few are easy to process. What is desired is a format that is easy to process and captures as much usable information as possible. As always, these goals are in conflict, so finding the middle ground is important.

The obvious form is the recorded performance. There are a variety of formats for recorded music such as MPG, WAV, AIFF, etc. Although this is the obvious form it also has the obvious problems. Processing an audio file into its separate components, that is the separate instruments, is extremely difficult. The human has a very well developed sense of hearing and an extremely proficient audio processor, yet very few people can identify the next to lowest note played by the piano. In most cases our attention focuses on the highest notes. Of course, as the number of instruments increases the difficulty of separating out each line of notes also increases. Sadly, we have a substantial way to go in order to render a score from a performance [Bello, 2000].

A much more tractable alternative is Musical Instrument Digital Interface (MIDI) format data. MIDI may be generated via a score program or recorded live from a large variety of electronic musical instruments. MIDI is composed of a series of signals indicating events and is relatively easy to parse. An event is typically the depressing or releasing of a key. This may include velocity information, which describes how vigorously the musician pressed the key. This velocity translates into the loudness of the note. In addition there is also information concerning the particular instrument tone that is being played. The modern digital piano can approximate the sound of virtually any instrument. Although, a Roland piano will never sound just like a Stradivarius violin, the differences in the two are not significant for the purpose of musical analysis.

In contrast to live recordings, either in audio or MIDI format, are the files used by the score production programs. Since a score is seldom finished in one session, these programs must store the data in a file in a retrievable way. An analysis program could parse the file and reconstruct the score as it would be published. The problem with this format is the difficulty of the parse. The manufacturers of such programs are not generally open about the format of the programs. There are at least two exceptions to this that are worth noting: ETF and MusicXML.

ETF is the acronym for Enigma Transportable File. This was an exchange format used by the MakeMusic! company (formerly known as Coda Music) in its Finale product line. The Finale series routinely stores the score in a proprietary format that typically changes with each new release in an upward compatible way. This format was a binary file with no specifications – it was not intended that any other program could process it. However, most of these could also store the score in ETF format. This was an ASCII file that was programmatically simple to read. It appears to be a people readable dump of the contents of the internal tables.

ETF was originally designed as a means to transport files from one version of Finale to another. Coda published the original specifications [Coda, 1998]. Margaret Cahill [Cahill, 2006] expanded these in her thesis, which seems to be no longer available. The types of coding used in ETF has increased over the versions and the available documentation has not. The first author has augmented this documentation as a side effect of constructing an ETF parser, however there is much that is not yet documented. This parser is the basis of several analysis tools to be discussed in the next section.

Unfortunately, MakeMusic! has ceased its support of ETF. Finale 2007 will not write out an ETF file, although it may still read it. According to their documentation it turned into a support problem as users tried to modify the ETF directly and produced inconsistent files. The Finale program could read the file but apparently did insufficient checking which led mysterious errors and eventually to finger pointing between the company and its users. This would appear to be a good corporate decision but disappointing from a short-term research perspective.

A superior format does exist and this is MusicXML. It was designed primarily to be an interchange format for score programs. Unlike ETF, which had little use outside of those who used Finale, it was designed to be able to represent the score in an independent way. It was pioneered by Recordare company [Recordare, 2008] but most of the score programs now support it. This includes Capella, Finale, Sibelius among many others [Recordare, 2008]. Strangely, there is no support for either the direct publication of a score nor the playing of a piece from MusicXML. The underlying assumption is that every person with an interest in MusicXML has one of the score programs, will import a MusicXML file into their score program and then print or play from there.

There are some tradeoffs between a score format like MusicXML and a performance format like MIDI. Both are relatively easy to parse. The score format will contain extra information that is tricky to extract from a performance format. It always contains both a key signature (such as key of Eb) and a time signature (such as 2/4), since these are essential parts of the score. When a piece modulates, that is moves from one key to another, the key signature generally changes on the score. The measure separators (that is the bar lines) should be of no consequence, but can give significant clues as to the structure. For example, a cadence (a sequence of keys that brings the piece to a temporary conclusion) almost always terminates at the end of a measure. Such things can be deduced from a performance but it is a complicated and error-prone process. Moreover, the performance of a score by a person always has minute variations from what is printed.

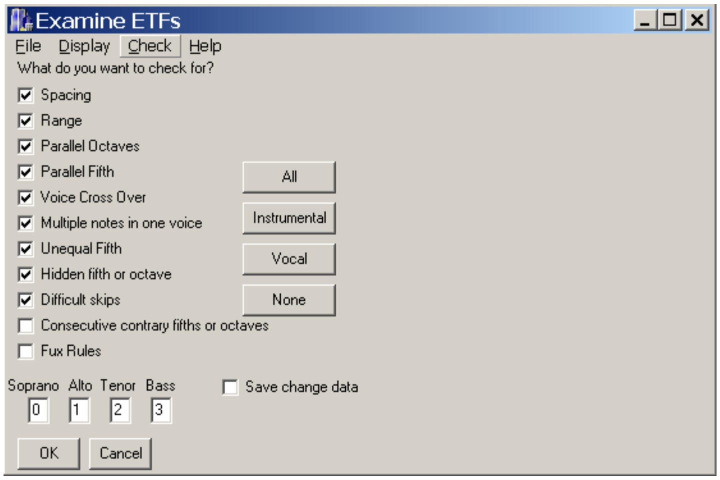
The comparison, however, is not one-sided. The performance format has some advantages of its own, when the file was created by a talented musician. Although a good musician should stay within close tolerances of the score, there is always room for interpretation that a score does not capture. For example, when looking for a cadence in a MIDI format file, one would look for slightly louder notes along with the other features. This is not evident from the score at all, but the musician has already done a de facto analysis and will emphasize things like a cadences.

Analysis Tools

Even though MusicXML would appear to be the preferred format, for historical reasons an ETF parser was constructed and the results in this section are based upon that. Work on converting the work to MusicXML is proceeding.

Once a parser of the file is complete there are a variety of data available for processing. Clearly the most important form of data is the information on the individual notes. This will necessarily include the note value and octave, as well as starting time and duration. Auxiliary information includes the measure information, which includes its starting time, key signature and time signature. This is the bulk of the raw data that needs to be analyzed. What will be considered next are three experimental programs that perform differing analyses on this raw data.

Document authors have a variety of checking tools available for their use. These include spelling checkers, grammar checkers as well as the determination of reading level. Many programming languages have types of verifiers as well, such as the lint program which checks for certain potential errors in source code. The first of the described programs does exactly that for compositions. Its intended audience was novice composers. It searches for a variety of conditions that should not normally exist in certain types of compositions. A screen shot of the dialog box that allows condition selections to be checked is in Figure 1.



What do you want to check for?

- ☑ Spacing
- ☑ Range
- ☑ Parallel Octaves
- ☑ Parallel Fifth
- ☑ Voice Cross Over
- ☑ Multiple notes in one voice
- ☑ Unequal Fifth
- ☑ Hidden fifth or octave
- ☑ Difficult skips
- □ Consecutive contrary fifths or octaves
- □ Fux Rules

Soprano	Alto	Tenor	Bass
0	1	2	3

□ Save change data

Figure 1. Checking dialog box.

There are several types of checking that may be done. Vertical checking compares a note with the other notes that are to be played at the same time. Horizontal checking looks at successive notes in the same voice. There are other possibilities as well including combinations.

Perhaps an explanation of a few of these conditions is in order. Most of these conditions are checked in a vocal piece for three or four part harmony. The idea of a spacing error is if there is more than an octave between the soprano and alto or alto and tenor voices. This is an example of vertical checking. A range error occurs when a note is beyond the usual range of that voice. This is neither vertical nor horizontal, just a case of demanding too much from a performer. A difficult skip is when a single voice is required to skip an interval that is beyond the average vocalist. This includes a skip of more than an octave, a seventh and the augmented fourth. This would be found with horizontal checking. A parallel octave or parallel fifth error occurs when two voices move from one note to another and maintain the parallel distance. This usually occurs in non-adjacent voices and requires a combination of horizontal and vertical checking.

Is this really necessary? Certainly to the experienced composer these rules are so ingrained as to require no additional work. However, consider the number of possibilities that a novice must deal with. In a ten measure piece, which is little more than an advertizing jingle, there might be approximately 40 notes in each voice. This requires about 40 range checks, 156 horizontal checks, 240 vertical checks as well as 117 horizontal and vertical checks. This is more than 500 things to check for in a trivial piece and this does not take into consideration that checking for a parallel octave is slightly different that checking for a parallel fifth. When these numbers are considered, as well as other conditions that may be selected but not here described, it should be clear that verifying a piece is a formidable obstacle for the novice composer.

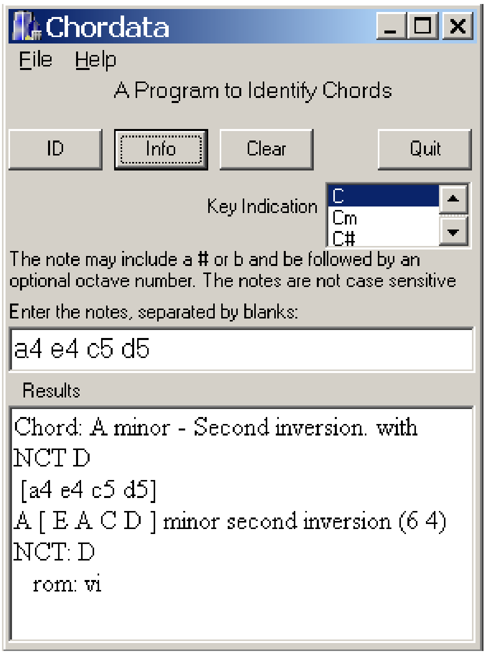
A chord is usually three or four notes with well known intervals that are played simultaneously. For example a major chord has a major third interval between the lower two notes and a minor third interval between the upper two notes. This would seem to be a simple thing to recognize, but this is not the case because of the many variations. In actual music any chord may be complicated by inversions, duplicated or omitted notes and extra notes that are not part of the chord. The marvelous human ear can appreciate all of these as more or less equivalent.

An inversion is a rearrangement of the notes in any way other than the standard notation. For example a standard C major chord consists of the notes, C, E and G in the same octave. There are six permutations of this order. They have somewhat different tonality but they are for the most part interchangeable. In a four part harmony at least one of the notes needs to be duplicated, although probably in a different octave. Thus any note may occur more than once. It is also the case that we will recognize the chord even if one of the notes is missing, so that possibility must be accounted for as well. The more difficult problem is non-chord tones. There are a variety of these that have function in the linear structure of the piece but are not a part of the chord itself. Notice that the number of combinations of tones that can form the same chord is quite large and there are a fair number of chords that one can expect in a particular key. The pattern recognition capability of people is rather amazing, for chords with any combination of these items is recognizable, not to mention pleasing. Since chord progressions, that is the sequential motion of chords, is an important foundation for many higher level structures, it is essential that chord recognition be rather robust.

The pattern recognition used involves representing the chord as a set of integers. Ultimately, a chord should fit in an octave. Thus after the notes are represented in the set the range is gradually reduced by plucking off the highest note and adding it back in an octave lower until the entire chord fits in the octave. This will also collapse duplicates into a single note. There are approximately 50 bit patterns that capture the tonality of the chord. Thus a D chord has exactly the same shape (ie. bit pattern) as a C chord, but it is based upon a higher note. If the set matches none of the patterns and has too many notes a backtracking process removes a note and tries again. If the note removed was an extra-chord tone the pattern should be discovered, otherwise that path fails.

This process is embedded in several objects that have a general usage in the analysis process. A simple program that identifies a chord entered as a series of notes is used as a test bed for these objects. Figure 2 shows a screen shot of this program. An explanation of what is shown follows.

The series of notes is entered into the window. Each note must specify the note name, such as D# or G. This may be followed by an optional octave number. Octaves always start with C and end with B, so C4 is one half-step above B3. Once the notes are entered the user may get a simple ID by clicking the ID button or more information by using the Info button.



A Program to Identify Chords

Key Indication: C

The note may include a # or b and be followed by an optional octave number. The notes are not case sensitive

Enter the notes, separated by blanks:

a4 e4 c5 d5

Results

```
Chord: A minor - Second inversion. with
NCT D
 [a4 e4 c5 d5]
A [ E A C D ] minor second inversion (6 4)
NCT: D
   rom: vi
```

Figure 2. A chord analyzed.

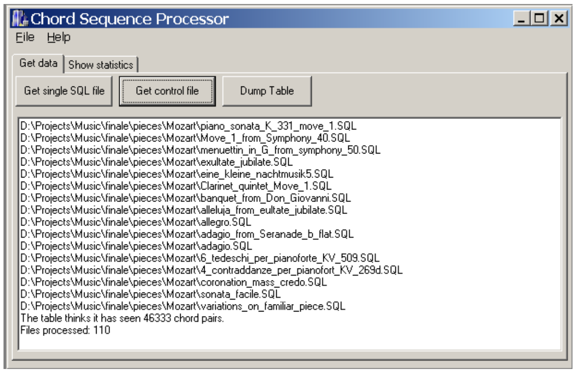
In the example shown four notes are entered. The ID button shows the first two lines. It identifies this chord as an A minor. Since the E and not the A is the lowest note it is labeled as an inversion. The program has also identified the D as a non-chord tone (NCT).

The last four lines results from using the Info button. It gives mostly the same information as the ID button, but offers some additional information as well. It shows the figured base notation (6 4) as well as the Roman numeral designation: vi. The Roman numeral notation is handy because it is independent of key. A piece transposed into another key has exactly the same Roman numeral notation for its chords, which makes comparisons of chords and chord sequences much easier. Roman numeral notation requires knowledge of the key that is being used when the chord is used. The key indication in the screen shot shows that this should be in the key of C major. Within C major an A minor is the sixth chord. An upper case roman numeral indicates a major chord, while lower case signifies a minor chord. Thus the vi signifies a minor sixth chord.

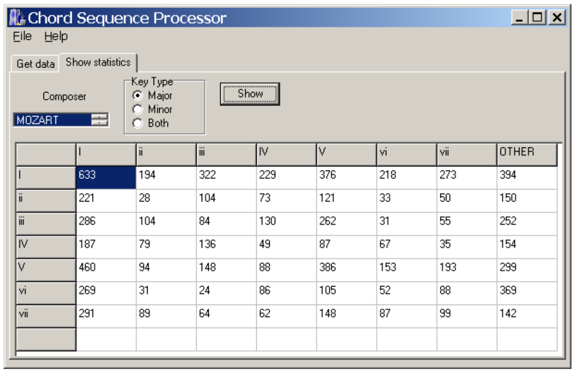
A chord progression is a sequence of chords. Of course, not any random sequence will be enjoyable and an important question is what chord may follow another in a pleasing fashion. The question has many variants depending on whether the focus is on just pairs of chords or on longer sequences. A very common sequence is the IV, V, I. With the ability to find and process large quantities of music this can be considered from a statistical point of view.

A two step approach was attempted to shed some light on this. The first step is to use the parsing of ETF files and the recognition of chords to create temporary file that captured chord transitions. This would provide data that indicate a starting and ending chord as well as measure information. (The form of these temporary files was that of SQL Insert statements even though a relational database was not used in this experiment.) The second step was that of a tabulator, that would select the data in some fashion and create a table counting transitions.

Statistical approaches are of dubious value with small sample sizes. It is of little interest if one or two Beethoven pieces are tabulated, but twenty would be of some value. Such a tabulator was created and the user may gather as many of the SQL temporary files as possible and create an in-memory database. This database may be selected by composer or by minor or major key and a table produced. Figures 3 and 4 show this program.



```
D:\Projects\Music\finale\pieces\Mozart\piano_sonata_K_331_move_1.SQL
D:\Projects\Music\finale\pieces\Mozart\Move_1_from_Symphony_40.SQL
D:\Projects\Music\finale\pieces\Mozart\menuettin_in_G_from_symphony_50.SQL
D:\Projects\Music\finale\pieces\Mozart\exultate_jubilate.SQL
D:\Projects\Music\finale\pieces\Mozart\eine_kleine_nachtmusik5.SQL
D:\Projects\Music\finale\pieces\Mozart\Clarinet_quintet_Move_1.SQL
D:\Projects\Music\finale\pieces\Mozart\banquet_from_Don_Giovanni.SQL
D:\Projects\Music\finale\pieces\Mozart\alleluja_from_eultate_jubilate.SQL
D:\Projects\Music\finale\pieces\Mozart\allegro.SQL
D:\Projects\Music\finale\pieces\Mozart\adagio_from_Seranade_b_flat.SQL
D:\Projects\Music\finale\pieces\Mozart\adagio.SQL
D:\Projects\Music\finale\pieces\Mozart\6_tedeschi_per_pianoforte_KV_509.SQL
D:\Projects\Music\finale\pieces\Mozart\4_contraddanze_per_pianofort_KV_269d.SQL
D:\Projects\Music\finale\pieces\Mozart\coronation_mass_credo.SQL
D:\Projects\Music\finale\pieces\Mozart\sonata_facile.SQL
D:\Projects\Music\finale\pieces\Mozart\variations_on_familiar_piece.SQL
The table thinks it has seen 46333 chord pairs.
Files processed: 110
```



Composer: MOZART

Key Type
● Major
○ Minor
○ Both

	I	ii	iii	IV	V	vi	vii	OTHER
I	633	194	322	229	376	218	273	394
ii	221	28	104	73	121	33	50	150
iii	286	104	84	130	262	31	55	252
IV	187	79	136	49	87	67	35	154
V	460	94	148	88	386	153	193	299
vi	269	31	24	86	105	52	88	369
vii	291	89	64	62	148	87	99	142

Figure 3. Loading data.

Figure 4. Displaying the table

Figure 3 shows the loading of the temporary files included over 100 separate scores. The bulk of these were obtained from the Finale Showcase [MakeMusic, 2008] which contains a large variety of scores suitable for downloading. A score could be a movement from a symphony or a much smaller piece. In this there were about 30 pieces from Mozart, which will be summarized in Figure 4. A particular piece may modulate through various keys, minor or major, but only those that are in a major key are summarized in the table.

The analysis of this table exceeds the scope of this paper. However, a casual glance asserts that Mozart was more likely to transition a V back to a I, which could hardly be a surprise since this is the basis of many cadences and was common well before Mozart and continues to this day. However, it seems that he was not a big fan of transitioning a IV to a V as is often common elsewhere.

Conclusions and Future Work

The purpose of this paper was to demonstrate potential, so no conclusions are needed. Yet, it seems clear from Figure 4 that what could have taken hundreds to thousands of hours is now quick and painless. The analysis of the data will still engage us but the tedium of generating it should be done by program.

The authors still have much to do. The construction of a suitable MusicXML parser that will produce data in way parallel to the ETF parser is in process along with the preservation of the existing analysis tools. The detection and classification of cadences is the next two project that is planned.

The goal of music analysis is to illuminate the structure of the work. Yet at the heart of it is the desire to find why we like some pieces and not others in order that every composition is a pleasing one. If computerized analysis assistance can bolster this effort in any way then it will be time and energy well spent.

References

Bello, Juan Pablo, Guiliano Monti and Mark Sandler(2000). Techniques for Automatic Music Transcription. Proceedings of the International Symposiumon Music Information Retrieval (ISMIR 2000). October 23-25, 2000, Plymouth, MA. http://www.ismir2000.net/papers/bello_paper.pdf. Date accessed 5 March 2008.

Cahill, Margaret (2006). Publications. http://www.csis.ul.ie/staff/Margaretcahill/publications.htm. Date accessed 5 March 2008.

Castan, Gerd (2008). Musical notation codes. http://www.music-notation.infor/en/notationformats-a4.pdf. Date accessed 5 March 2008.

Coda Music (1998). Enigma Transportable File Specification. http://www.xs4all.nl/~hanwen/lily-devel/etfspec.pdf. Date accessed 5 March 2008.

MakeMusic (2008). Finale Showcase. http://www.finalemusic.com/showcase/ Date accessed 7 March 2008.

Recordare (2008). MusicXML Definition, Version 2.0. http://www.recordare.com/xml.html and http://www.musicxml.org/xml.html. Date accessed 5 March 2008.