Detecting And Correcting Voice Leading Errors

The formal rules of classical voice leading are the result of hundreds of years of study by composers and musicologists. Many composers undertaking study in the composition of tonal music, even today, undergo some training in species counterpoint and its successor, voice leading. The principles of “good” voice leading are still understood to produce music which is pleasing to the listener, even when outside the context of the four-part harmonies from which the principles are derived.

As in many other subjects, the rules of voice leading are often taught by having the student complete composition exercises, which are then corrected by a teacher. This project has resulted in software which is capable of detecting and correcting the voice leading errors, as a teacher would do. The program is written in Haskell, and leverages the Euterpea library extensively.

Exceptions

The rules and guidelines which govern classical voice leading are by no means absolute. Some of the most famous composers of history were known to break these rules, on occasion. Nonetheless, as the goal of this project was to produce software which could be used as an aid in assisting music theory students as they practice composition, the program is strict in its error detection. To accommodate situations in which students may wish to knowingly violate common rules or go against common guidelines, the program is written in such a way that voice leading rules are applied modularly, and can be enabled or disabled at any time by the programmer.

Error Detection

The software is able to detect (and correct) the following voice leading errors:

- Doubled leading tone
- Direct Octaves
- Direct Fifths
- Direct Unisons
- Voice Crossing
- Parallel Octaves
As mentioned above, the program is written in a modular manner so that if a student wishes to omit any of these rules, or even write their own, doing so is straightforward. Instructions for how to do so are included in the included source code. Upon detecting an error, the program reports the location of the error, which rule was violated, and the notes which make up the identifying chord.

Atomization

A challenge faced in allowing the algorithm to appropriately detect errors in a piece was handling cases in which parts move at different times. It can become hard to detect non-chord tones in this scenario, which hinders our ability to check certain harmonic errors. In order to deal with this, the program breaks the score down into vertical “atoms” which represent the four pitches which are playing at a given time in the piece. Whenever one voice moves, a new atom begins, and this atom lasts until another voice moves. This also simplifies the process of reporting errors and correcting them. An example is shown below.
Error Correction

Whereas error detection is relatively straightforward, correction requires a much more complex algorithm. The algorithm I have designed scans a piece from left to right, searching for errors, and correcting them in place as they are discovered. Corrections are made by analyzing the harmony at a given point, producing a list of all possible chords which fit into that harmony without breaking any voice leading rules, and then ranking the chords based on their voice leading quality, where chords which move the least are preferred over chords which move the most. The scoring mechanism which ranks the chords penalizes movement in the middle two voices harshly, movement in the soprano less harshly, and movement in the bass barely at all. In order to keep the program as flexible as possible, the scoring function is completely modifiable by the user, and can be modified to apply different weights to certain types of movement even on the same piece.

The correction algorithm takes into account the possibility of having incomplete chords (in which one member of a triad is missing). It also allows the user to mandate that one or more parts not be changed. This is because it is not uncommon in voice leading exercises for a bass or soprano line to be given.

While progressing from left to right, the algorithm ensures that it does not induce any errors before the point to which it has already progressed. In the event this results in the algorithm “composing itself into a trap” in which the options for replacement chords would all produce voice leading errors, the algorithm begins to see if rewriting nearby chords simultaneously would help avoid the trap. The algorithm could theoretically attempt to correct all nearby chords (until it had reach either
end of the piece) using this strategy, but doing so comes at a tremendous computational cost. On a personal computer, it is reasonable to limit this strategy to rewriting 4 chords simultaneously, as a typical chord has around 40 possible replacements. There is a limit set within the program which controls this feature and can be adjusted as necessary.

Performance of correction algorithm

In pieces which have few errors, the algorithm tends to be very successful in finding acceptable corrections. With the exception of harmonic progressions which are known for having no acceptable voice leading (such as the progression of the Ger+6 to the V7 chord), there has not been an exercise which the piece has failed to correct, keeping in mind that these exercises are short and often designed to be easy to compose.

A particularly interesting case is that the algorithm has even been able to “correct” a piece composed of 1) 2 opening chords, 2 closing chords, and 9 nonsensical chords in between -- what I have affectionately dubbed “Garbage”. This piece violates some sort of rule at almost every point. Using the starting chord, the ending cadence, and the harmonic progression input by the user, a suitable piece is still generated by the algorithm.

“Garbage” before correction
Additional examples

MIDI files and JPGs representing “Before and After” for the correction of examples are included on the project webpage.

Future work

The correction algorithm could be significantly improved if modification of harmonies by the algorithm were allowed. This could allow the program to find a suitable harmony quicker. Rather than sorting through millions of combinations of chords, it could first check to see if switching a harmony would be acceptable. The ability to correct poor harmonic motion would allow for even better-sounding corrected scores.

Acknowledgements

Professor Paul Hudak for his guidance throughout this project, not to mention the Euterpea library.

Xiao Shi ‘16, and John Clayton ‘13 for providing completed voice-leading exercises of their own for testing purposes.
References

