David Elkin-Ginnetti

Musical Signal Processing

21 December 2018

# Towards an Algorithmic Harmonic Accompaniment

## I. Scope and Intentions

Over the course of the semester, our class has concentrated on the analysis on audio signals, transforming files from symbolic flows of digital samples into other forms of meaningful data. My electronic music practice works with a different but interrelated form of transformation. I am fascinated with acapellas and vocal sampling, especially with singers like Brandy and Rihanna, whose personae really shine through in their voices. When I chop up and reprocess the acapellas of these singers and others, I re-contextualize their vocals to beyond the point of understanding, while re-emphasizing the qualities or affect that might be more under-the-hood and ephemeral in the original song. I like to think it reveals something different in a small vocal flourish, or a voice's timbre.

The sort of granular re-synthesis of the voice I do usually requires some sort of harmonic contextualization to make sense in the larger scope of a track. Ever since I have been writing these harmonic accompaniments, I have wondered: what would it look like to have an algorithm do this for me? For this final project, I wanted to take the coding knowledge and computational awareness I gained from the course and apply it to a creative outlet. Freeing myself from the composerly duties of accompaniment would add an algorithmic inhuman voice to the conversation. I could even insert myself into the place of these vocalists, and play my saxophone (which for the past few years has been a solitary venture) to a computerized interlocutor. This

second voice is listening and making guesses about my harmonic "meaning" and then adding to the dialogue by outputting live MIDI accompaniment, changing the harmonic direction of the whole as a result. My algorithmic re-harmonizer is a first attempt in this direction, with the final destination being a fully-functional live performance application. The re-harmonizer is necessarily analytic because it must try to arrive at certain conclusions about melodies in order to add its own voice into the conversation, but my main goal is for this function to act as a creative resource.

**II. Prior Research**

Algorithmic composition has been an active line of research in the fields of computer science and signal processing for nearly three decades. Projects have ranged from novel tools designed for computer-assisted composition (Assayag & Laursson 1999) to spectogram analysis based on fuzzy set theory to determine tonal centers, a solid first step in the direction of composing (Vidyamurthy & Chakrapani 1992). More recent work has focused on structures like Markov chains to model harmonic movement (Eigenfeldt & Pasquier 2010), although these have been implemented in melody-generating algorithms since the 1990s (Mozer 1994).

One of the key distinctions mentioned repeatedly in the literature is on the design of an algorithmic compositional tool: should it rely on a programmed intuiting of structures found within a piece of music to model itself after? Or will it be more successful in generating meaningful accompaniment when programmed heuristically, with a set of given constraints that match up to a preexisting compositional model? These questions also center around the challenge of programming an algorithm to creatively compose music with meaning: understanding context

and “tolerating ambiguity in representation” are essential for computerized systems to learn what makes meaning for human listeners (Papadopoulos & Wiggins 1999). These are not easy characteristics to represent computationally, but should be considered when designing systems that attempt to approximate what we as humans understand as creativity.

**III. Project Design**

I tried to incorporate into my dataset a similar collection of melodic samples to those which I might be manipulating in a fully functional performance version of this code. This included songs by Brandy, whose acapellas I often sample and manipulate in my own work, as well as Alanis Morrisette, Faith Evans, Lauryn Hill, Michael Jackson, and other R&B and pop artists whose melodic style and flourishes I thought would prove interesting to harmonize with. The MIDI files were sourced from the clean MIDI subset of the Lakh MIDI dataset. I combed through to find songs with which I would be familiar and that shared stylistic conventions, then processed the full MIDI files in Ableton Live to clean up notes and reduce the files to melody lines which were mostly monophonic. They were then bounced from Live for further processing in Python.

Once it became clear that it would be unfeasible to work with live audio for the project, the Pretty MIDI library emerged as a suitable environment for easily manipulating MIDI data in a human-oriented protocol. The Pretty MIDI library is designed hierarchically, with each MIDI instance containing instruments, each instrument containing notes, and each note containing a start and end time, pitch, and velocity information (Raffel and Ellis 2014). This makes it possible to manipulate notes as individual entities, rather than dealing with two different messages for

note-on and note-off, as the MIDI protocol specifies. Pretty MIDI comes prepackaged with some elegant tools for data visualization, like a piano roll plotting all the notes in a file over time, or weighted and unweighted pitch class histograms, essentially a discrete counterpart to a chroma graph which can be adjusted to assign importance to note duration and velocity. I wanted to scrape data from these histograms in order to make guesses about the tonality of the file. The library did not provide a way to access data from the graph itself, so I needed to write a function to make a quantitative counterpart. What resulted was a pitch probabilities function that scans the file to index instances of each pitch class, then averages these out with a weighting function to get the relative probability of each note over the total file.

I decided to simplify the scope of the project by narrowing down the desired output to some of the standard constraints of Western popular music: the harmony would be in a bass note that stays below the main melody, changing pitch once per measure. I designed the harmonizing function so that the harmonies through every measure might hem closest to fitting into a given major or minor tonality that would be enharmonic to the assumed key of the song. To accompany the melody each measure, the function chooses a note which is either the first, third or fifth step of a triad correlating with the greatest probability to the measure's assumed harmonization. This was essentially the extent of the theoretical "knowledge" that I wanted to give the algorithm. I hoped and expected other interesting tonal coincidences to emerge through the algorithm's heuristic for guessing the next note of a harmony.

About half of the length of the harmonizing function is a series of if/else statements that I hoped would be a suitable analog to the heuristic that a bass player in a band would use for harmonizing a melody, given a set of chord changes and an understanding of how to shape a bass

line, with the memory of how the last note will relate to the next. In hindsight I think this section got a little too convoluted with trying to imagine every possible permutation of notes, but it did end up creating patterns that mirrored what I would expect from a human musician in my preliminary tests. At the end of the function, all of the notes in the harmony array are written to a new MIDI instrument that is appended to the file containing the melody.

**IV. Evaluation**

I evaluated the results of the algorithm on the purely qualitative basis on harmonic novelty. Results that I valued most were were not just producing one note for a track, as some of the melody files ended up with, even if that was "harmonically fitting." Nor did I want something that had a totally chaotic harmonic output. I was trying to approximate the subtle shades of structure and unpredictability that are often sought after in the idiom of jazz improvisation, for instance. As I expected, it was very difficult to write this degree of subtlety into a relatively simple Python function with the time resources I had allotted for this project. The results did not converge on any one localized failure or success, but depended strongly on the harmonic content within the MIDI file the function was operating on. Some of the melodies, such as Faith Evans' "Love Like This" and "My Girl" by The Temptations, were complemented with harmonies that accentuated some parts of the existing melody, while highlighting potential re-harmonizations not present in the original chord changes. These were successes for me, for in these cases the algorithm helps us see something in the harmony that we did not glimpse before. These moments of spontaneous creation on the part of the function are what I was really trying to get from the project, and in a few cases it produced results that were interesting in that respect.

The code also produced more modest results, like with Brandy's "Have You Ever." The harmonizing function returned a series of notes that remained on the minor 3rd of a song that was clearly in a major key, which the algorithm should have been able to figure out clearly from the pitch probabilities table. I did not have the time to do a thorough debugging of this behavior, but I know it was the result of one of the many conditionals in the if/else structures I implemented for the harmonization function. My initial impulse to design a heuristic based on the decisions of an actual bass player did not translate well into the code. Musicians do not think in if/else statements; we float many different possibilities, try out a couple, and see what works. In light of this, I am going to begin future revisions of the function by implementing a decision structure that relies less on the rote mechanics of if statements, and more on a randomly seeded probability function to pick the next note, selecting a suitable guess among a few random options after checking its harmonies against the rest of the notes in the melody. I will also add larger context for the decision function, so that it relies not just on what the last note and the current harmony is, but maybe will look for broader patterns and try to replicate or imitate them. My conjecture is that these structures would allow for a smoother harmonic line, or at least one more interesting than in my current implementation. Perhaps it will give the algorithm a chance to construct antiphonal dialogue rather than just simple downbeat accompaniment.

To further improve this code in the future would call for more experimentation with relative weightings of guesses about localized harmony, e.g. interpolations based off one measure of information, versus guesses based off the harmonic content of the whole melody. The current state of the algorithm puts primary importance on the harmonic content summed over the whole file, with which it makes one probabilistic generalization about "key." (I only put this in

quotation marks because it is never explicitly named in the function, except as a starting note for the harmonization. This is another weakness of my design; in general the implementations that most match a human perception of pitch and harmony will be most successful at creating interesting and novel results.) A more flexible implementation of the harmonizing algorithm might have an adjustable weighting system that could prioritize local guesses over the general. This would improve the its performance on songs with a key change, with which the algorithm fared poorly in these tests– see Michael Jackson's "Rock With You," in which the harmony blunders through a modulation towards the end of the song. By experimenting with combinations of adjustable weightings for local/global harmony recognition, along with alternative structures for pitch decisions, I think I will be able to obtain results in the future that are more harmonically novel, and that might even approach creativity.

Overall, the experiment in algorithmic harmony was a success in that it showed me the areas to improve upon to bolster the algorithm's "creativity." The prior research I had read was right: the best computational systems for algorithmic composition will be those that are flexible, comfortable with ambiguity, and have some sort of knowledge representation that approximates human understanding of music. In further work on this harmonization function, I will be researching how to build on these ideas. Maybe the algorithm will eventually be a good musical conversation partner, helping myself and others see harmony in a different light.

References Cited

Assayag, G., Rueda, C., Laurson, M., et al. (1999). Computer-assisted composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal, 23*(3), 59-72.

Eigenfeldt, A., and Pasquier, P. (2010). Realtime generation of harmonic progressions using controlled Markov selection. Retrieved from http://computationalcreativity.net/iccc2010/papers/eigenfeldt-pasquier-1.pdf

Mozer, M. (1994). Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing. Connection Science, 6(2&3), 247-280.

Papadopoulos, G., and Wiggins, G. (1999). AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects, presented at AISB Symposium on Musical Creativity, Edinburgh.

Raffel, C., & Ellis, D. P. (2014). Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi. In *Proceedings of the 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers*.

Vidyamurthy, G., & Chakrapani, J. (1992). Cognition of Tonal Centers: A Fuzzy Approach. *Computer Music Journal, 16*(2), 45-50. doi:10.2307/3680715