Automatic Composition from Non-musical Inspiration Sources

Robert Smith, Aaron Dennis and Dan Ventura
Computer Science Department
Brigham Young University
2robsmith@gmail.com, adennis@byu.edu, ventura@cs.byu.edu

Abstract

In this paper, we describe a system which creates novel musical compositions inspired by non-musical audio signals. The system processes input audio signals using onset detection and pitch estimation algorithms. Additional musical voices are added to the resulting melody by models of note relationships that are built using machine learning trained with different pieces of music. The system creates interesting compositions, suggesting merit for the idea of computational "inspiration".

Introduction

Musical composition is often inspired by other musical pieces. Sometimes, the new music closely resembles the inspiring piece, perhaps being an intentional interpretation or continuation of its themes or ideas. Other times the connection between the pieces is not identifiable (or even conscious). And, such sources of inspiration are, of course, not limited to only the musical realm. A composer can be inspired by the sight of a bird, the smell of industrial pollution, the taste of honey, the touch of rain or the sound of a running stream. Since this is the case, an interesting question for the field of computational creativity is whether a similar mechanism can be effected in computational systems. If so, new, interesting mechanisms for the development of (musical) structure become viable.

Many attempts have been made at computational composition. These attempts use mathematical models, knowledge based systems, grammars, evolutionary methods and hybrid systems to learn music theory, specifically whatever music theory is encoded in the training pieces applied to the algorithms (Papadopoulos and Wiggins 1999). Some of these techniques have been shown to be capable of producing music that is arguably inspired by different music genres or artists (Cope 1992). Some computational composers focus on producing melodies (Conklin and Witten 1995), but most focus on producing harmonies to accompany a given melody (Chuan and Chew 2007)(Allan and Williams 2005). Ames (Ames 1989) and others have described training Markov models on existing artists or styles and generating similarly sounding melody lines. No system that we have found models the idea of artistic inspiration from non-musical sources.

We present a computational system which implements a simple approach to musical inspiration and limit our focus to (non-musical) audio inspirational sources. Our system can autonomously produce a melody and harmonies from non-musical audio inputs with the resulting compositions being novel, often interesting and exhibiting some level of acceptable aesthetic.

Methodology

Our approach to automatic composition from non-musical inspirational sources is composed of four steps: (1) audio input and melody generation, (2) learning voice models, (3) harmony generation and (4) post-processing.

Audio Input and Melody Generation

Inspirational audio input was selected from various sources. Our samples included baby noises, bird chirpings, road noises, frog croakings, an excerpt from Franklin Delano Roosevelt's "A Date Which Will Live in Infamy" speech, and an excerpt from Barack Obama's 2004 DNC speech.

The melody generator takes an audio file (.wav format) as input and produces a melody. The input signal typically contains many frequencies playing simultaneously and continuously, and the generator's job is to produce a sequence of non-concurrent notes and rests that mimics the original audio signal. To do so, it uses an off-the-shelf, free audio utility called Aubio to detect the onset of "notes" in the audio file (as well as to estimate their duration) and to extract the dominant pitch at each of these onset times. Aubio is intended for analyzing recordings of musical pieces in which actual notes are played by instruments; however, in our system it is used to analyze any kind of audio signal, which means Aubio extracts "notes" from speeches or recordings of dogs barking or anything else. A thresholding step discards generated notes that are too soft, too high, or too low. The result is a collection of notes, extracted from the raw audio, composing a melody.

Learning Voice Models

To produce harmonization for the generated melody, we employ a series of voice models, M_i, learned from a collection of MIDI files representing different musical genres and artists. Each such model is trained with a different set of training examples, constructed as follows. First, because

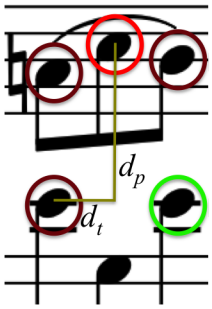

Figure 1: *Finding neighbor notes.* The top center note (circled in red) is the current melody note. In this case, $k = 3$, and, assuming $w_p = w_t$, the k closest neighbors are the two notes surrounding the melody note on the top staff and the first note on the bottom staff (circled in dark red). d_t refers to the distance in time between the melody note and neighbor, and d_p refers to the change in pitch. The $(k + 1)$th note is the rightmost note on the bottom staff (circled in green).

there is no restriction on the time signature of the input or output pieces, note durations are converted from number of beats to seconds.

Second, to identify the melody line of the training piece (and later to identify the melody line of the output piece), we use a simple heuristic assumption that the highest pitched note at any given time is the melody note.

Third, for each melody note, we find the $k + 1$ nearest neighbor notes using the distance function (see Figure 1):

$$d(n_1, n_2) = \sqrt{w_t d_t(n_1, n_2)^2 + w_p d_p(n_1, n_2)^2}$$

where n_1 and n_2 are notes, and weights w_t and w_p allow flexibility in how chordal or contrapuntal the training data will be. d_t and d_p compute absolute difference in onset time and pitch, respectively, so

$$d_t(n_1, n_2) = |\text{onset}(n_1) - \text{onset}(n_2)|$$

and

$$d_p(n_1, n_2) = |\text{pitch}(n_1) - \text{pitch}(n_2)|$$

Training instances are constructed from a musical piece's melody notes and its $k + 1$ closest notes. The training inputs are the melody note and its k nearest neighbors, while the $(k + 1)$th closest note is used as the training output (see Figure 1). The melody note is encoded as a 2-tuple consisting of the note's pitch and duration. The neighbor notes and the output note are encoded using a 3-tuple consisting of the time (d_t) and pitch (d_p) differences between the neighbor note and the melody note and its duration (see Figure 2). When building the training set for voice model M_i (with i indexed from 0), $k = i + 2$. So, after training, voice model M_i computes a function, $M_i : \mathbb{R}^{3i+8} \rightarrow \mathbb{R}^3$.

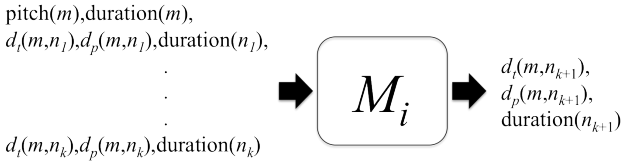

Figure 2: *Training the voice models.* For each melody note m of each training piece, a training instance is created from the melody note and the $k + 1$ closest neighboring notes $(n_1, ...n_{k+1})$. The k closest neighbors are used, along with m as input, and, as training output, the $(k + 1)$th closest neighbor is used. The melody note is represented as a pitch and a duration. Each of the other notes is represented as a 3-tuple consisting of d_t, d_p, and duration, where d_t and d_p refer respectively to the differences in start time and pitch between the neighbor note and the melody note.

Harmony Generation

The harmony generator is applied iteratively to add notes to the composition. Each pass adds an additional voice to the composition as follows. For the iteration 0, $k = 2$ and voice model M_0 is used with the melody as input. Each note, in turn, is used as the melody note, and it and it's two nearest neighbors are used as input to the model, which produces an output note to add to the harmonizing voice. This does not imply that each harmony note is produced to occur at the same time as its associated melody note. For each melody note the model produces as output values for d_t, d_p, and duration; the harmony note will only start at the same time as the associated melody note if $d_t = 0$.

When all melody notes have been used as input, the additional harmonic voice is then combined with the original melody line and the first iteration is complete. For iteration 1, $k = 3$ and voice model M_1 is used with the new two-voice composition as input, and the process is repeated, with the following caveat. We use the "melody" notes of the current piece (that is, the highest pitched notes) instead of the original melody notes (along with their k neighbors) as input to the model. This allows the melody notes to change from iteration to iteration, since the system can output notes that are higher than the (current) melody. The end result is another harmonic voice that is combined with the two-voice composition to produce a three-part musical composition (see Figure 3).

This process is repeated for v iterations, so that the final composition contains $v + 1$ voices in total. Empirically, we found that $v = 3$ resulted in the most pleasing outputs. With $v < 3$ there was not enough variation to distinguish the output from the original melody. For higher values of v, the less musical and more cluttered the output became.

Post-processing

After the output piece has been composed, the composition is post-processed in two ways which we call *snap-to-time* and *snap-to-pitch* (and to which we refer collectively as *snap-to-grid*).

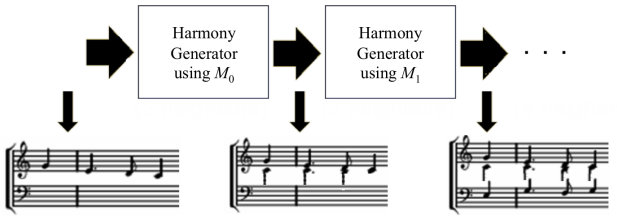

Figure 3: *Adding voices.* The harmony generator is applied iteratively over the melody line and generated harmony lines, using successively complex voice models. These iterations add successive voices to a composition.

Algorithm 1 Snap-To-Time. This algorithm adjusts note start times in the final composition to compensate for lack of uniform timing across input and training pieces. First, δ_{min}, the minimum difference in start time between any two notes in the melody, is calculated. Each note is then shifted so that its start time is an integer multiple of δ_{min} from the start time of the composition's initial note.

$\delta_{min} \leftarrow \infty$
for all notes n_1 **do**
  for all notes n_2 **do**
    $\delta \leftarrow |\text{onset}(n_1) - \text{onset}(n_2)|$
    if $\delta < \delta_{min}$ **then**
      $\delta_{min} \leftarrow \delta$
    end if
  end for
end for
for all notes n **do**
  $\Delta \leftarrow \lfloor \text{onset}(n)/\delta_{min} + .5 \rfloor * \delta_{min} - \text{onset}(n)$
  onset(n)←onset(n)+Δ
end for

Due to the beat-independent durations of the generated notes, the note onsets in the composition can occur at any time during the piece, which can result in unpleasant note timings. To correct this, we implement a snap-to-time feature.

To do so, we first analyze the melody line to determine the shortest time, δ_{min}, between any two (melody) note onset times. Then each composition note onset is shifted so that it is an integer multiple of δ_{min} from the onset of the first note in the composition (see Algorithm 1). In other words, each note is snapped to an imaginary time grid whose unit measure is δ_{min}, with the result being music with a more regular and rhythmic quality.

Because each voice is generated independently, there is no explicitly enforced (chordal) relationship between notes which occur at the same time. The voice models may provide some of this indirectly; however, this implicit relationship is not always strong enough to guarantee pleasing harmonies—there exists the possibility of discordant notes. To remedy this, we implement the snap-to-pitch algorithm.

If two notes occur at the same time, the difference in their pitches is computed. The pitches are then adjusted until the pitch interval between the notes is acceptable (here, for simplicity, acceptable means one of {*major third*, *perfect fourth*, *perfect fifth*, *major sixth*}). See Algorithm 2.

Algorithm 2 Snap-To-Pitch. The notes n_1 and n_2 start at the same time. If the interval between them is not one of {*major third*, *perfect fourth*, *perfect fifth*, *major sixth*}, snap-to-pitch modifies the pitch of one of n_2 so that it is.

$\delta \leftarrow \text{pitch}(n_1) - \text{pitch}(n_2)$
if $\delta > 0$ **then**
  if $\delta < 4$ **then**
    $\delta = 4$
  else
    while $\delta \notin \{4, 5, 7, 9\}$ **do**
      $\delta \leftarrow \delta - 1$
    end while
  end if
else if $\delta < 0$ **then**
  if $\delta > -3$ **then**
    $\delta = -3$
  else
    while $|\delta| \notin \{3, 5, 7, 8\}$ **do**
      $\delta \leftarrow \delta + 1$
    end while
  end if
end if
$\text{pitch}(n_2) \leftarrow \text{pitch}(n_1) - \delta$

As a summary, Algorithm 3 gives a high-level overview of the entire compositional process.

Results

Musical results are better heard than read. We invite the reader to browse some of the system's compositions at http://removedforblindcopy.

In some cases the melody generator produces melody out-

Algorithm 3 Algorithmic Overview Of System. A melody is generated by detecting pitch, onset, and duration of "notes" in an inspirational audio sample. Additional voices are added by creating increasingly complex voice models and iteratively applying them to the composition. The entire composition is then post-processed so that it incorporates a global time signature of sorts and to improve its tonal quality.

$composition \leftarrow \text{extractMelody}(inspirationAudio)$
for $i = 0$ to v **do**
  $k = i + 2$
  $trainset \leftarrow \emptyset$
  for all training pieces t **do**
    $trainset \leftarrow trainset \cup \text{extractInstances}(t, k))$
  end for
  $\text{trainModel}(M_i, trainset)$
  $composition \leftarrow \text{addVoice}(M_i, composition)$
end for
$composition \leftarrow \text{snapToTime}(composition)$
$composition \leftarrow \text{snapToPitch}(composition)$

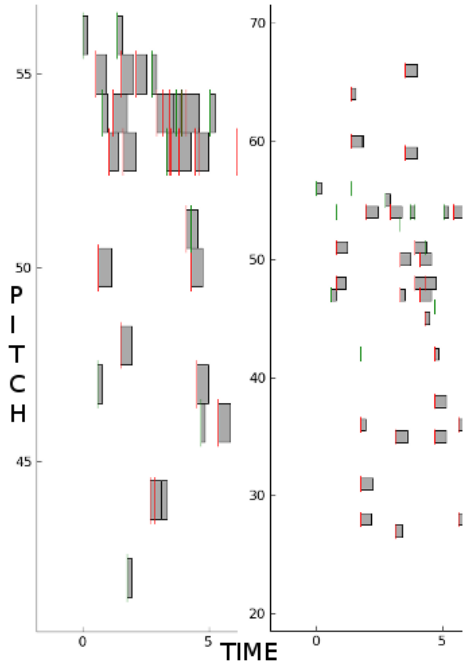

Figure 4: *Snap-to-grid*. The first graph shows the layout of an output composition based on CarSounds without snap-to-grid post-processing. The second graph shows another CarSounds output with snap-to-grid. Note the change in the pitch scale that reflects the increase in pitch range which is a result of adjusting concurrent notes to an aesthetically pleasing interval.

puts which are readily identifiable with their inspirational source audio files. Examples include compositions inspired by a speech by President Obama and by a bird's song. In both cases, the resulting melody line synchronises nicely with the original audio when both are played simultaneously. In contrast, other compositions sound very different from their inspirational source. Examples include a recording of a frog's repetitive croaking and a monotonous recording of road noise in a moving car. In the case of the road noises one would expect an output melody that is monotonous, mirroring the humanly-perceived characteristics of the input audio file. However, the melody generator composes a low-pitched, interesting, and varied melody line when given the road noise audio file, making it hard to identify how the melody relates to its source.

In all outputs there is a general lack of traditional rhythm and pitch patterns. This is, of course, not surprising given that our audio sources for inspiration are not required to be in any particular musical key or to follow traditional key changes, nor do they have any notion of a time signature. Additionally, we do not restrict our training sets in either of these traditional ways. As a consequence, it is likely that in any given training set there will be instances which are in different keys and/or time signatures than the melody. In light of these conditions, it is to be expected that the output would not be traditional music.

Training	w_t	w_p	Percent Chords
TwoDance	1	1	83
TwoDance	1	3	44
TwoDance	3	1	80
TwoBlues	1	1	67
TwoBlues	1	3	47
TwoBlues	3	1	71

Table 1: This table shows the effect of the weights w_t and w_p. The input was the FatFrog audio file and voice models were trained using either two songs from the Dance genre or two songs from the Blues genre. Generally, as w_p increases (with respect to w_t), the number of chords produced in the output composition decreases.

The snap-to-grid feature is helpful. We have posted audio examples on the web comparing outputs with and without snap-to-grid. An example graph of each is given for visual comparison in Figure 4. Snap-to-time doesn't significantly change the landscape of the pieces, but it proves to be essential in synchronizing notes which were composed as chords but are not easily recognized as such because of the high precision of start times. Snap-to-pitch has a dramatic effect on the pitch of certain notes but is limited to those notes which occur at the same time.

We explored several values for w_t and w_p (see Table 1), and, as expected, when $w_p > w_t$ there are less chordal note events than single notes compared to when $w_p < w_t$. Interestingly, the baseline $w_t = w_p = 1$ for the case of voice models trained with two Dance songs is slightly more chordal even than $w_t = 3, w_p = 1$.

We could not detect any significant difference in effect when using different genres or artists for training the voice models. No distinguishable qualities of dance music were discernible in the outputs composed using models trained only on dance music. No distinguishable qualities of Styx songs were discernible in the outputs composed using models trained only on songs by Styx. In short, each variable on training input successfully introduced novel variations in the output compositions in an untraceable way. Choice of training pieces did not produce a predictable pattern for aesthetic quality. The fact that our (admittedly simple) voice models failed to capture the distinct qualities of certain artists or genres suggests that our methods for encoding the musical qualities of training pieces are less effective at capturing such information than they are at capturing interesting note combinations and timings (see Figure 5).

As described, the standard system uses the $k + 1$ closest neighboring notes of each melody note for training the voice models, and this works. However, as a variation on this approach, randomly sampling $k + 1$ notes from the $4k$ closest notes adds some extra variation in the composition and can lead to more aesthetically pleasing outputs.

Snap-to-grid proved to be very useful for contributing to the aesthetic quality of the compositions. Compositions without snap-to-grid have more atonal and discordant chords which play at undesirable intervals. Using snap-to-grid allows a compromise between the uniqueness of the compo-

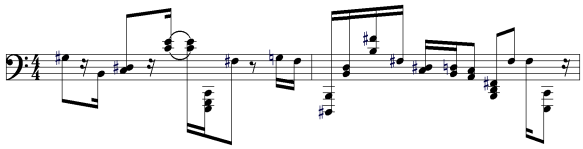

Figure 5: *Composition sample*. These two measures are taken from one of the compositions produced by our system. The system produces interesting rhythms with varying chordal texture.

sitional style and regular timing intervals and chordal structure.

Future Work

At this point, our system is quite simple and many of the techniques it employs are somewhat naïve musically. Some of this naïveté is for convenience at this early stage of system development, and some of it is design decisions that allow for greater variety in system output. The snap-to-grid processing is a post-hoc attempt to impose some level of musical "correctness" on the system's output. Given the unconstrained nature of the inspirational input, it is an interesting question to ask how one might naturally incorporate useful aspects of music theory directly in the melody generation process while still allowing significant effect from the source. Also, it is natural to suggest incorporating more traditional and mature harmonization schemes for the generated melodies. Finally, to this point, only the melody has been (directly) affected by the inspiring piece; it would be interesting to develop methods for using the inspirational source to directly influence other musical characteristics such as harmonization, style, texture, etc. However, all of these necessary improvements are relatively minor compared to the real open issues.

The first of these is the development of an evaluation method for judging aesthetic and other qualities of the compositions. To this point, our measure of "interestingness" has been only our own subjective judgment. The development of more principled, objective metrics would be useful as a filtering mechanism, and, at a more fundamental level, as feedback for directing the system to modify its behavior so that it produces better (novel, interesting, and surprising) compositions. In addition, such results may also be vetted in various kinds of human subject studies.

The second of these is the development of a mechanism for autonomously choosing which inspirational sources the system will use as input. This requires the development of some type of "metric" for inspiration. Or, perhaps another way to think about this problem is to ask the question, "what makes a sequence of sounds interesting (or pleasing, or arousing, or calming, or ...)?" Is this quantifiable or at least qualifiable in some way? Some potential starting points for this type of investigation might include work on identifying emotional content in music (Li and Ogihara 2003; Han et al. 2009) as well as work on spectral composition methods (Esling and Agon 2010).

This, in turn, introduces further considerations, such as in which quality or qualities the system might be interested and how those interests might change over time. An additional consideration is that of a second level of inspiration – rather than the system being inspired by the aural qualities of the input alone (as it is at present), is it possible to construct a system that can be inspired by metaphors those aural qualities suggest? And is it then possible for the system to communicate the metaphor to some degree in its output?

References

Allan, M., and Williams, C. K. 2005. Harmonising chorales by probabilistic inference. In *Advances in Neural Information Processing Systems 17*, 25–32.

Ames, C. 1989. The Markov process as a compositional model: A survey and tutorial. *Leonardo* 22(2):175–187.

Chuan, C. H., and Chew, E. 2007. A hybrid system for automatic generation of style-specific accompaniment. In *Proceedings of the 4th International Joint Workshop on Computational Creativity*.

Conklin, D., and Witten, I. H. 1995. Multiple viewpoint systems for music prediction. *Journal of New Music Research* 24:51–73.

Cope, D. 1992. Computer modeling of musical intelligence in EMI. *Computer Music Journal* 16(2):69–83.

Esling, P., and Agon, C. 2010. Composition of sound mixtures with spectral maquettes. In *Proceedings of the International Computer Music Conference*, 550–553.

Han, B.; Rho, S.; Dannenberg, R. B.; and Hwang, E. 2009. SMERS: Music emotion recognition using support vector regression. In *Proceedings of the 10th International Conference on Music Information Retrieval*, 651–656.

Li, T., and Ogihara, M. 2003. Detecting emotion in music. In *Proceedings of the 4th International Conference on Music Information Retrieval*, 239–240.

Papadopoulos, G., and Wiggins, G. 1999. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *Proceedings of the AISB Symposium on Musical Creativity*, 110–117.