A Preliminary Study on Text-Based Music Generation

Niklas Zechner

Department of Computing Science,
Umeå University
zechner@cs.umu.se

Abstract

Generating music from a text, considering both the phonetics and the semantic contents of the text, is a large and complex task, but all the parts necessary have been widely studied, and some are even commercially available. We look at a prototype of a system for combining all the different steps, to generate mood-appropriate sung music directly from unrestricted text. Well-established methods for speech synthesis can be separated into text-to-phoneme and phoneme-to-audio conversion, so we can insert music generation algorithms between those steps. Using sentiment analysis to analyse the mood of the text, we can fit the musical output to the text. With the modular nature of the method, it is easy to extend the system with more complex algorithms.

1. Introduction

Writing about combinations of notes into chords and melodies, Guido of Arezzo (1025) makes the indubitable claim "...some are suitable, some are more suitable, and some are the most suitable". In the millennium since, as the accepted views on which combinations are the most suitable have changed, many have tried to formalise the rules, so that a person - or a machine - could write music by just applying them.

Different studies vary in their start and end points. Högberg (2005) starts from only a choice of style, and generates form, melody, and accompaniment using tree transducers. Toivanen et al. (2013) start from a single word, using that to generate both lyrics and music. Chan and Ventura (2008) start from a given melody and a mood, and let the algorithm create backgrounds that adapt the melody to the mood. This can be useful for video games and other adaptive media, where we can let the music seamlessly develop as the setting changes with the character for example moving between locations.

Another situation where mood-appropriate music is valuable is when generating music to fit given lyrics, which is the topic of this study. Here, the algorithm takes as input plain text, and creates suitable music, fitting not only the natural rhythm of the text, but also the mood. We try to develop a modular prototype, where each part can be replaced with more advanced software.

We want to work with the whole real-world application all at once - go straight from unrestricted text to audio output. The problem consists of many parts: We must extract information from the text, both phonetic and semantic, generate the music to match it, and produce the audio, considering both textual and musical information.

The good news is that none of these tasks are in themselves new. Text-to-speech tools are found on regular home computers - all we need to do is to separate the parts, so we can insert the other processes in the middle. So the whole process becomes:

Use sentiment analysis to extract the mood of the text

Use a text-to-speech system to convert the text to phonetic data

Extract the rhythm information from the phonetic data

Generate music based on the rhythm and mood

Modify the phonetic data accordingly

Use the text-to-speech system to produce audio from the modified phonetic data

In this experiment, we will use the MacinTalk text-to-speech system available in Mac OS X. It has a convenient feature that lets us specify pitch and rhythm, making it easy to produce output which is sung rather than spoken. For the other steps, we use simple algorithms as proof of concept.

Sample output from the program can be found on www.cs.umu.se/~zechner/singer.

2. The process

2.1 Analysing the text

The first thing we want to do is analyse the input text to determine the mood. In this initial version, we look for certain words that suggest whether the text is happy or sad - a rudimentary form of sentiment analysis. We use that to automatically create the music in a major or minor key.

We use the existing software to transform the input text into phonemic information. This includes a list of phonemes with stress pattern, duration, and pitch. We will use this intermediate data to create a musical version, by replacing the pitch and modifying the duration.

2.2 Generating the rhythm

To determine a suitable rhythm, we extract the stress pattern. To keep things as simple as possible, we ignore vowel length and secondary stress, and think of the spoken rhythm as a bitstring, with 1 for stressed syllables and 0 for unstressed. We can then divide it into substrings by cutting it off before each 1, so the substrings are on the form (1, 10, 100, 1000...). We use a fixed 4/4 time signature, and assign each such substring to one bar.

Next, we need to figure out the exact rhythm for each bar. In the simplest case, we can use a fixed pattern for each number of notes. Then we need, for each n, a sequence of lengths adding up to 1. Perhaps the most straightforward way is going from n to $n+1$ by picking the last longest note and cutting it in half. So for increasing number of syllables, we get the length sequences

1
1/2, 1/2
1/2, 1/4, 1/4
1/4, 1/4, 1/4, 1/4
1/4, 1/4, 1/4, 1/8, 1/8
1/4, 1/4, 1/8, 1/8, 1/8, 1/8
...

Depending on the style of music, we might want to add more rhythmic variation, dotted rhythms, or syncopation, but as a first approximation, this works quite well; for example, a three-syllable sequence is much more likely to have a (1/2, 1/4, 1/4) rhythm than (1/4, 1/4, 1/2), let alone (1/4, 1/2, 1/4), or for that matter (1/3, 1/3, 1/3) (assuming the music is in 4/4 time). This is presumably because stress is associated not only with amplitude but also with length. (It is also strongly associated with pitch, which is certainly something we could use in an algorithm like this, but that could get more complicated.)

This approach works better for some languages than others. Toivanen et al. (2013) work with Finnish, where vowel length is very important, so they take that into account when determining the length of notes. In syllable-timed languages like Italian or Cantonese, where each syllable takes roughly the same time, this kind of rhythm assignment might be problematic, but since English is a stress-timed language, where each stress point takes roughly the same time, this approach closely mirrors natural speech. We are also basing the music on a European tradition; some other traditions would not divide things into bars at all.

Note that these stress-initial substrings are not meaningful divisions in any semantic or syntactic sense, but they do not need to be. We do however need to think of phrases, and leave pauses in between them. The easiest way to do that is to assume that each sentence is a phrase. We want a pause between the phrases, but we also want to allow phrases starting with unstressed syllables. This can be easily achieved by thinking of the phrase break as a silent but stressed syllable.

2.3 Adjusting phoneme length

At this point, we have a length associated with each syllable. Now, we need to assign each phoneme to a note, and divide the allotted time between the phonemes. The obvious way would be to separate the syllables where they naturally break, so for example the word "bitstring" would be divided into "bit" and "string", but that would be a mistake. Consider the song fragment "twinkle twinkle little star". The stress pattern is clearly "1010101". If you read it rhythmically, clapping your hands at the stressed syllables, you'll notice that the /s/ clearly occurs before the last clap. You probably see the same thing with "tw" in "twinkle" - it comes before the clap corresponding to that syllable. The clap occurs not at the beginning of the syllable, but at its sonority peak, typically the vowel. So, just as we divided the sequence of syllables into substrings starting with the stressed part, we should do the same with phonemes, each note containing not the phonemes of one syllable, but rather one vowel and the following consonants.

Some consonants (mainly plosives, such as p b k g t d) are usually the same length, at least in English. Others may vary in length, and this may be noticeable in singing; one example is the aforementioned "twinkle", where, if you sing very slowly, you will probably extend the n-sound rather than the i-sound. But this is usually not essential - if you extend the i-sound instead, it may come out as "tweenkle", but it is probably not too much of a concern. So, as a first approach, we let all consonants keep their length, and extend the vowel until the allotted time is filled. We see again the advantages of using an existing system for speech synthesis, as we already have natural-sounding consonant lengths without having to think about it.

2.4 Generating music

Having established a rhythm, we turn to generating a melody. Many advanced algorithms have been devised for this purpose, but we will keep things simple for this preliminary demonstration.

For many music styles, we can think of suitable melodies and harmonies in terms of constraints, including cost functions on intervals between concurrent notes as well as between sequential notes. But constraint algorithms can get slow and complex. We can reach an adequate result using incremental methods, basing each step on previous steps.

For simple Western music, keys and chords are very useful in finding appealing melodies. There is no need to include key changes or accidentals (that is, deviations from the key), so we assume all songs will be in C major or C (harmonic) minor, depending on the results from the sentiment analysis, and we completely exclude notes not in this scale (effectively setting their cost to infinity). As for chords, we want to keep things simple while highlighting the difference between major and minor, so we use only the three most common chords for each key. In terms of chromatic scale positions, those are ({0, 4, 7}, {0, 5, 9}, {2, 7, 11}) and ({0, 3, 7}, {0, 5, 8}, {2, 7, 11}). We choose chords randomly, with equal probabilities, but always set the final chord to the first of the set (the tonic). Using chords is an effective way to get natural-sounding melodies, and makes it easy to extend the system by adding voices or other harmonic backgrounds.

For each note, we rate each possible scale position, by combining several criteria:

> whether the note is in the chord; this is more important if the note is stressed
>
> the distance between this scale position and that of the previous note; smaller intervals are better, and this is less important if the note is stressed
>
> whether the note is the same as the previous; although small distances are good, we may get more interesting melodies with a penalty for repetition

the distance between this scale position and the middle of the available range; closer is better

a random factor

3. Example

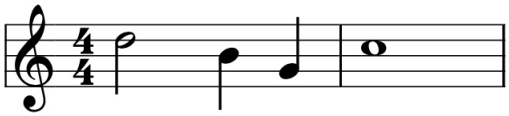
As an example, we try the text "give me a hug". We take the output from the speech synthesis, ignoring syllable breaks:

```
g {D 95; P 102.4:0 112.2:47}
1IH {D 140; P 130.9:0 130.7:14 118.4:61 111.1:79}
v {D 85; P 99.1:0}
m {D 80; P 101.3:0 102.0:44}
IY {D 85; P 106.4:0 106.3:71}
IX {D 75; P 104.9:0 102.3:60}
h {D 85; P 94.0:0 99.5:29 105.4:47 119.2:71}
1UX {D 170; P 139.9:0 146.3:12 144.9:24 109.1:71}
g {D 145; P 80.9:0 74.2:59 73.4:86 75.0:100}
```

The "1"s denote stressed vowels, so the algorithm determines the stress pattern to be 1001. We get two bars of music, and the note lengths will be (1/2, 1/4, 1/4, 1). No negative words are found in the text, so we get a major scale. The chords are chosen by the program to be ({0, 5, 9}, {0, 4, 7}). The notes are chosen to be (14, 11, 7, 12). The phoneme lengths and frequencies are adjusted, and we get the modified data:

```
g {D 95; P 146.832627925494:0 146.832627925494:100}
1IH {D 603; P 146.832627925494:0 146.832627925494:100}
v {D 85; P 146.832627925494:0 146.832627925494:100}
m {D 80; P 146.832627925494:0 146.832627925494:100}
IY {D 384; P 123.47103046483:0 123.47103046483:100}
IX {D 299; P 97.9990218237347:0 97.9990218237347:100}
h {D 85; P 97.9990218237347:0 97.9990218237347:100}
1UX {D 1391; P 130.813:0 130.813:100}
g {D 145; P 130.813:0 130.813:100}
```

which corresponds to the following notes:



4. Future work

We have purposely designed the process so that each part can be replaced by something more advanced. There are many ways the different algorithms could be extended.

Much research has been done on sentiment analysis that could be applied here. We could try to classify the text as energetic or relaxed, and use that to set a tempo. Taking it to a more advanced level, we could check for other things that can be translated into music styles - dialectal differences might affect regional differences in music, archaic words lead to older music styles, or age and gender classification determine which voice to use.

For the music generation, there are also many more advanced algorithms, some of which would be applicable here. As a first step, we could include more rules, such as "avoid intervals of 6" or "if the previous note was much higher than the one before that, prefer a slightly lower note now". It would also be an easy task to add harmonic backgrounds, which can also vary in style depending on the sentiment analysis.

Other possible approaches include working with specific styles of music; styles such as gregorian chant, traditional four-part harmony, or baroque minuets follow many rules that would work in a system like this. It might also be interesting to explore the differences between languages and geographic music traditions, as mentioned above, and the relation between them.

References

Guido of Arezzo. 1025. *Micrologus de disciplina artis musicae.*

J. Högberg. 2005. Wind in the willows – generating music by means of tree transducers. *International Conference on Implementation and Application of Automata*, 2005: 153–162.

J.M. Toivanen, H. Toivonen, and A. Valitutti. 2013. Automatical composition of lyrical songs. *The Fourth International Conference on Computational Creativity*.

D.A. Ventura and H. Chan. 2008. Automatic composition of themed mood pieces. *Computational Creativity*.