

Generation of musical accompaniment for a poem, using artificial intelligence techniques

Caroline-Cristina Stere¹, Ștefan Trăușan-Matu^{1,2,3}

¹Universitatea Politehnica din București

Splaiul Independenței nr. 313, București,

E-mail: caroline.stere@gmail.com, stefan.trausan@cs.pub.ro

²Institutul de Cercetări în Inteligența Artificială

Calea 13 Septembrie nr. 13, București

³Academia Oamenilor de Știință din România

Splaiul Independenței nr. 54, București

Abstract. Music has the gift of adorning everything it touches and of expressing things that cannot be expressed through words alone. This paper describes a software system developed in order to complete the message sent by a poem, with a suitable melody. The system analyzes a poem from both the rhythmical and the general-emotion point of view. This analysis is followed by the actual generation of music, that consists of two equally important phases: the generation of the rhythm and the generation of the musical notes.

Keywords: Sonification, poem, natural language processing, poem analysis, mood classification, rhythm

1. Introduction

Sonification is the process that transforms information, regardless of its nature, into sounds. The word itself comes from the Latin “sonus”, meaning sound, to which the Latin particle “ficatio” has been added, which translates to creation. One of sonification’s first practical applications is the Geiger counter, that produces sounds that are proportional in intensity to the level of radiation present in the device’s area. Another important application is the sonar, that uses sound propagation to detect underwater objects. The utility of data sonification is thus clear.

In this paper, we apply the sonification process to the field of literature, to generate music based on a poem. Michelangelo is credited that he said that “Every block of stone has a statue inside it and it is the task of the sculptor to discover it”. Drawing an analogy, we can say that every poem contains a melody that must be brought to surface.

Music and literature have an intertwined history. They might have

appeared together, but, even if this happened, over time they have developed and resulted in two art forms that continue to influence each other. Just like literature, regardless of its genre (lyrical, epic or dramatic), music is many times used to convey a story. For example, opera tells a story through music and words, but even pure instrumental music, which is devoid of any text, can take a strong narrative form. The symphonies of Mahler and Beethoven are prominent examples of narrative and evocative music (Davis and Mohammad, 2014, p. 1).

1.1 Brief description of the system

In this paper, we analyze a poem and, based on this, we generate a piano composition, whose sound is in harmony with the mood transmitted by the poem.

We analyze the poem from three points of view:

- **Rhythm:** We divide the words into syllables and identify the stressed and unstressed ones.
- **Punctuation:** We identify the punctuation marks and classify them according to the length of the pause they cause in the speech.
- **Mood:** We classify the poem as joyful or sad.

After analyzing the poem, we generate the piano score in two steps: First, the rhythm of the song, i.e. a sequence of note durations is divided into four-quarter notes measures. Second, the actual melody is constructed. After generating the sequence of note durations, we map each duration to an appropriate musical note.

1.2 The motivation behind this paper

This paper has its origin in the desire to connect in an automatic way two art forms that share a common genesis and history, namely literature and music. The text of a poem evokes certain emotions and thoughts, that echo in an inner music and which we have tried to bring to surface.

Moreover, generating musical accompaniment for a poem is the first step towards more complex projects, with practical uses. Among these, we mention (Davis and Mohammad, 2014, p. 1):

- Audio-visual e-books that generate music upon opening certain pages with a higher emotional load. The generated music has the role of emphasizing the action of those pages.
- Generating soundtracks for games or movies.
- Generating a musical composition that helps improve visualization techniques, to communicate information in an efficient and artistic way. A possible application is a world map, which, when clicking on a region, begins to play a song that evokes the mood of the tweets from that region.

2. State of the art

The process of sonifying a poem requires knowledge in the fields of music theory and poetry prosody (such as rhythm, meter, rhyme, and verse measure). Moreover, to classify a poem as cheerful or sad, Natural Language Processing (NLP) is necessary.

2.1 Rhythm in poems

The expressivity of a poem is the result of both carefully chosen words, arranged in evocative figures of speech, and verses' rhythm. Although the words are the ones that set the theme of the poem, it is the rhythm that consolidates the evoked ideas and provides musicality to the verses. These two elements, the text and the rhythm intertwine in a poem, to create verses that enjoy both the mind and the ear.

The alternation of stressed and unstressed syllables determines a poem's rhythm. This alternation is not random though, but instead follows a specific pattern.

To identify the rhythm pattern (i.e. the *meter*) of a poem, the verses' words are divided into syllables, the stressed and unstressed syllables are marked, and the prevalent metric unit is identified. The metric unit represents a fixed pattern of stressed and unstressed syllables, which is repeated throughout the poem.

2.2 Machine learning

To generate a melody that fits the mood of the poem, we use a *Naïve Bayes* classifier, that classifies a text as "happy" or "sad". Naïve Bayes is a type of

supervised learning, where the learning hypothesis is determined based on labeled data. A set of data (X, T) is required, where X is the training set and T is the set of classes. The training set X contains several examples, each example being labeled with a class from the set T . Based on this labeled dataset, the classifier can label new objects (Berariu, 2014).

In our case, the examples to be classified are texts. Therefore, the attribute set for each poem is represented by the *TF-IDF* (term frequency – inverse document frequency) values of the words. The TF-IDF value reflects the relevance of a word in a document that is part of a data set. It is directly proportional to the word's frequency within that document and inversely proportional to the word's frequency within the data set. This inverse proportion decreases the value of those words that appear in many documents and therefore are not relevant to determining the class of the document.

In the following paragraphs we will explain what changes must be made to the vocabulary of the texts, to be able to compute their TF-IDF values.

To transform a text into a set of attributes, three processing steps are required:

1) Removing stop words

Stop words appear frequently in a data set and therefore are not considered to provide relevant information about the class of the document (e.g. conjunctions, articles, prepositions, etc.). There are two ways to remove them:

- Using a list of stop words from a certain language.
- Sorting all words in the dataset in descending order of their frequencies, and removing from each text the first N words of that list.

2) Bringing the remaining words to the root form

To compute the frequency of each word, we need to bring it to a root form. For example, it is very likely that words such as: “receive”, “receiving”, “received”, “receives” or, a more complicated situation: “is”, “am”, “are” to be interpreted as different words, because they're not identical, even if, in the second case, they are different conjugations of the verb “to be”. Here too there are two ways of getting the root of a word:

- Stemming, which can generate the roots of the words in the first case.
- Lemmatization, that tries to obtain valid roots and lexical data even in the second case, which is why it is more resource intensive than the latter.

3) Obtaining the attribute set

At this point, the document only contains the roots of relevant words. Next, it must be transformed into an array whose size is equal to the total number of distinct words in all documents. This array will hold on position i the TF-IDF value of the word i .

2.3 Similar projects

Although the project presented herein, which analyzes a poem and generates a suitable melody to accompany it, is an original one, there are some other projects that aim to sonify different kinds of literary works. One of these projects is TransProse (Davis and Mohammad, 2014).

TransProse is a system that sonifies a novel using the connection between emotions and various musical elements, such as scale or tempo. The application receives the text of a novel and produces a piano composition that transmits the various feelings encountered in the book.

TransProse generates music in three basic steps (Davis and Mohammad, 2014):

- 1) First, it analyzes the text of the novel and creates a “map” of feelings. This map is an ensemble of statistics on the usage density of emotion-depicting words. For this purpose, it uses a lexicon that maps words to the evoked feelings, such as fear, joy, anger, surprise, etc.
- 2) Next, it generates the octave, the scale and the tempo based on the feelings identified in the previous step. It also produces notes sequences for each section of the novel.
- 3) Finally, these notes and values are introduced into JFugue, a Java API for music generation. Thus, JFugue combines all this information and produces a suitable audio file.

On the project’s web page are examples of compositions generated from various novels (TransProse, 2017).

3. Implementation details

We implemented the sonification system in Java. For creating and playing the musical composition, we used the *JFugue* library, whereas for the interface of the application we used *JavaFX*, a library for creating graphical interfaces.

JFugue is an open source library that offers an easy way to program music in Java. It first appeared in 2002 and its current version is 5.0. The JFugue format for representing music is called “Staccato” and it’s designed to be easy to understand and write. Besides numerous features such as generating a chord progression based on the chord (e.g. I vi IV V) and the scale (e.g. C major), JFugue also converts the Staccato format into and from various other musical formats (MIDI, MusicXML, LilyPond, etc.).

3.1 The workflow of the system

The system receives a poem as input and generates an appropriate piano song. The composition is two-handed, namely on two voices: one voice for chord progressions and one for the actual melody (the notes sequences). This process follows roughly four basic steps (see also Figure 1):

- 1) Analyze the text of the poem
 - Determine the general mood of the poem and, based on this, set the tempo and the scale of the composition.
 - Divide the words in each verse into syllables and mark the stressed ones.
 - Identify the punctuation marks and classify them by the pause they introduce in the speech.
- 2) Generate the rhythm of the composition
 - Construct a set of suitable note values for each type of syllables and punctuation marks.
 - Map each syllable and punctuation mark to a note value, randomly chosen from its corresponding set, while maintaining the four-quarter notes measures.
- 3) Generate notes sequences
 - Construct sequences of melodic intervals, following the basic rules of music theory.
 - Construct chord progressions.

- Iterate through the previously generated note values and map them to actual notes.
- 4) Create the output
- Generate the MusicXML (<http://www.musicxml.com/>) file, which contains the music score of the composition.
 - Play the song.

We continue by presenting each module, one by one, focusing on the functionality and on some implementation details.

3.2 Poem analysis: the general feeling

The message of the poem is a key factor in generating adequate sound accompaniment. The theme of the poem can be happy or sad, and the song must evoke this feeling with the help of the scale. The scale of a composition is either *major* or *minor*. Major scales have a joyful, optimistic sound, whereas minor scales evoke anxiety, fear.

To classify the mood of the poem, we have used an already implemented classifier called MusicMood. This is an open-source project, that classifies song lyrics as happy or sad. Because song lyrics resemble a poem, we have considered it fit for classifying poems.

MusicMood is a Naïve Bayes classifier implemented in Python (Raschka, 2014; MusicMood, 2017). The training data set is selected from Million Song Dataset, a collection of audio features and metadata for a million popular songs from the last decades (Million Song Dataset, 2017). Out of these, he chose a set of 1,200 songs with lyrics in English, which he labeled manually. MusicMood is also available as a web application (MusicMood, 2017).

After we classify the poem as happy or sad, we set the *tempo* and the *scale* of the song. The following table shows the connection between these two notions.

Table 5. How the mood of the poem influences the scale and the tempo

| The mood of the poem | Scale | Tempo |
|----------------------|---------|-------------------------|
| Happy | C Major | Faster (e.g. Moderato) |
| Sad | C Minor | Slower (e.g. Andantino) |

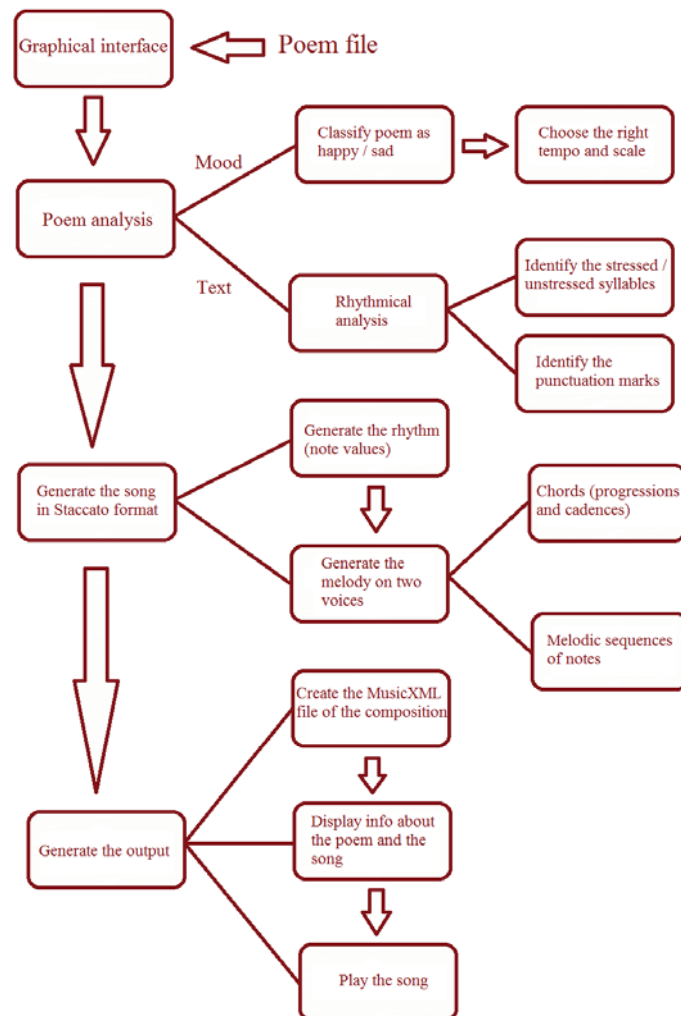


Figure 18. The workflow diagram

3.3 Poem analysis: the rhythm

After analyzing the general mood of the poem, we have focused on the rhythmic analysis of the text. First, we split the words into syllables, identifying the stressed and unstressed syllables. Then we identify the punctuation marks and categorize them, based on the length of the pause that

they produce in the speech.

The *accent* of the syllables, along with the *punctuation* marks, contribute to the musicality of a text, which strengthens the meaning of the words. To support this statement, we give as example the first stanza of Edgar Allan Poe's narrative poem "The Raven", in which we underline the stressed syllables.

Once u – pon a mid – night dre – ary, while I pon – dered, weak and wea – ry,
 O - ver ma – ny a quaint and cu – rious vo – lume of for – got – ten lore –
 While I nod - ded, near – ly nap – ping, sud – den – ly there came a tap – ping,
 As of some one gent – ly rap – ping, rap – ping at my cham – ber door.
 "Tis some vi – si – tor," I mut – tered, "tap – ping at my cham – ber door –
 Only this and not – hing more." (Poe, 2017).

The musicality of this stanza is obvious from the first reading. It is determined by the trochaic rhythm, which consists in a stressed syllable, followed by an unstressed one. The trochee is a descending rhythm, that gives the poem rapidity, dynamism, while also creating a feeling of hesitation and doubt. The dynamism of the lyrics thus contributes to the gradually built sensation of anxiety, while the feeling of hesitation is very clear in the last two verses, in which the poet tries to convince himself that behind the door is just a visitor.

In addition to the *meter*, *punctuation* has also an important role in supporting the musicality of the verses. In this stanza there is an overwhelming use of commas, which create a rhythm that resembles the beating of a drum.

To identify the stressed syllables, we have used a modified version of the *CMU* (Carnegie Mellon University) word pronunciation dictionary. The *CMU* dictionary is an open-source project, created at the Carnegie Mellon University in Pittsburgh, Pennsylvania. This project offers pronunciation models for more than 134 000 of words, in a form that is easily processed by computers (CMUDICT, 2017).

The *CMU* dictionary describes the word pronunciation using the ARPAbet phoneme set (Advanced Research Projects Agency). A *phoneme* is a unit of sound, namely how we perceive a sound, which helps distinguish words from one another. The *CMU* dictionary also maps vocals to a digit indicating the accent: 0 for no accent, 1 for the main accent and 2 for the secondary accent.

For example, the mapping of the word *happiness* in the dictionary is *HH*

AEI P IYO N AH0 S, the phonemes being separated by spaces. We notice that the accent falls on the AE phoneme, which corresponds to the vocal “a”. If we wanted to mark the stressed syllable, this would be the first syllable, consisting of the phonemes *HH AE*.

The shortcoming of this dictionary is that it does not split the phonemes into syllabic groups, whereas in the present project we need to identify the stressed syllables. Therefore, we have used a *modified version of the CMU dictionary*, which groups the phonemes into syllables (Bartlett, Kondrak, and Cherry, 2009). This operation is done automatically, using SVM (Support Vector Machines). Thus, for the word *happiness*, the modified version of the CMU dictionary will produce *HH AEI - P IYO - N AH0 S*, the syllables being visually separated by hyphens. This version of the dictionary can be downloaded from <https://webdocs.cs.ualberta.ca/~kondrak/cmudict.html> (last accessed on June 27, 2017).

- In this step of the rhythmic analysis, we examine each verse of the poem, transforming it into an array of values 0, 1, 2 or 3. We iterate through the words and punctuation marks of each verse and process them accordingly:
- The *words*: They are searched in the CMU dictionary, the returned value being a set of phonemes grouped into syllables. We process each syllable, checking if one of the phonemes is followed by the digit 1, which means that the syllable is stressed, so we add the value 1 to the array; Otherwise it means that the syllable is unstressed, so we add the value 0 to the array. To avoid situations where a monosyllabic linking word, such as a conjunction or a preposition, is marked as stressed, we use a list of words that are usually unstressed in speech. If the processed word is found in this list, we mark it in the array as being unstressed, by adding the value 0. We consider that for words with more syllables, there must be a stressed one (e.g. for the conjunction *whenever*, the second syllable is stressed).
- The *punctuation marks*: We group them in two sets, based on the length of the pause they produce in speech: shorter (“,” “;” “:” “-”, “- -”) or longer (“.” “!” “?” “...”). If the punctuation mark is from the first group, we add the value 2 to the array, otherwise, we add the value 3.

After the rhythmic analysis of the poem, we obtain a set of arrays, one for each verse. Each element in these arrays describes a syllable or a punctuation mark:

- 0 – unstressed syllable
- 1 – stressed syllable
- 2 – short pause in speech, caused by one of the punctuation marks in the group (“,” “;” “:” “-” “- -”)
- 3 – a longer pause in speech, caused by one of the punctuation marks in the group (“.” “!” “?” “...”)

3.4 Generating the song

Every musical composition consists of two basic elements: the *rhythm* and the *melody*. In addition, the harmonic structure (the chords) has the role of supporting and completing the melody, giving it a well-defined form (Williams, 2017).

Therefore, to automatically generate a song, we have split the process in two steps: one for generating the rhythm (the notes’ durations) and another one for mapping each duration to a musical note and for generating an appropriate chord progression. In the following subsections we describe each step in detail.

Generating the rhythm of the song

The rhythm of a song is a sequence of note durations divided into measures, without having associated any actual musical notes (C, D, E, etc.). For example, the following figure contains a few measures from the beginning of the song “Happy Birthday to you”, written on a one-line staff.




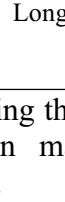
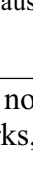
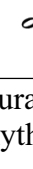
Figure 19. The rhythmic notation of a couple of measures from the song “Happy Birthday to You” (Williams, 2017)

To generate the song’s rhythm, we have used the results from the rhythmical analysis of the poem (the array with possible values from 0 to 3). Each of these values determine the choice of a note duration from a set of possible values. This way, each syllable and punctuation mark have a corresponding

duration in the melody's rhythm.

In the following table, we present the connection between the elements of the array and the note durations.

Table 6. The connection between the poem's rhythm and the notes durations

| Array element | Element's meaning | Possible note durations | Explanation |
|---------------|---------------------|--|---|
| 0 | Unstressed syllable |  | The unstressed syllables are represented by a short value, i.e. an eighth note. |
| 1 | Stressed syllable |  | The stressed syllables are mapped to longer durations, i.e. two eighth notes, one quarter note or one half note. |
| 2 | Short pause |  | The short pause is represented by an imperfect cadence, followed by a rest. Each chord and the rest are mapped to a quarter note. |
| 3 | Long pause |  | The longer pause is represented by a perfect cadence, followed by a rest. Each chord and the rest are mapped to a half note. |

By setting the note durations based on the accent of the syllables and the punctuation marks, rhythm-like verses will generate similar melodic sequences.

This step generates an array of measures, each measure containing a sequence of note durations. The time signature is $\frac{4}{4}$, as it is the most common one. Next, we describe the algorithm by which we create four-quarter-note measures.

For each verse of the poem:

- For each value in the array that describes the verse's rhythm (the array obtained from the rhythmic analysis of the poem):
 - We choose the appropriate note durations (as shown in Table 2). For the value 1 (stressed syllable), we randomly choose one of the three possible note durations.
 - Next, we introduce these note durations into measures, following the rule that each measure has exactly four quarter notes. So, for each note duration, we check the following:

- If it fits into the current measure, we add it there.
- If the current measure is complete, then we create a new measure and add the note value there.
- If the current measure is incomplete, but the duration that we want to add is longer:
 - We complete the current measure with the note value needed to reach four quarters.
 - We create a new measure and add the note value there.

Generating the melody

After producing the rhythm of the song, i.e. the sequence of note durations divided into measures, the next step is to map those note durations to musical notes. These musical notes are organized in *two voices*: one voice for the *chord progressions and cadences* and another one for the *melodic sequence*. The musical notes for these two voices are not random, but follow the basic rules of music theory:

- Chords are part of a progression (a progression is a sequence of several chords that sound good when played one after another);
- Cadences are a particular type of progressions, consisting of two chords, that play the role of punctuation marks in a musical composition;
- The musical notes of the melodic sequence are chosen based on the notes of the chord that is being played at that moment:
 - The first note (the one played at the same time as the chord) must be part of the chord;
 - The following notes can be:
 - Either notes that belong to the chord;
 - Or notes within a two-semitone (i.e. one tone) interval from the last note that was part of the chord.

In the following, we describe the steps to generate musical notes for the two voices. Firstly, we choose one of the chord progressions that are suitable for the current scale (major or minor). We have chosen some commonly used chord progressions, which we enumerate in the following table:

Table 7. Chord progressions for the major and minor scales

| Scale | Examples of chord progressions | | | | |
|-------|--------------------------------|---------------|----------|------------|---------------------|
| Major | I IV V I | iii vi ii V I | I V IV | I vi IV V | I V vi IV |
| Minor | i VI III VII | I VII VI VII | i iv v i | VI VII i i | ii ⁰ v i |

Next, we create two Java objects of type *Pattern* (a JFugue class that will contain the musical notes and their durations), one for each voice: one for the voice that plays the chords, called *leftHand*, and one for the voice that plays the melodic sequence, called *rightHand*.

We iterate through each measure and analyze what type of notes it contains, more precisely if it includes any part of a cadence or a rest, since these two elements interfere with the model described above.

If it does not contain a part of a cadence or a rest:

- To the *leftHand*: We add the chord in line from the chosen progression, with the duration of a whole note (four beats).
- To the *rightHand*: We add a sequence of notes based on the previously chosen chord (following the rule described above). These notes are mapped to the durations generated in the previous section, 3.4.1.

If the measure contains a part of a cadence or a full cadence:

- To the *leftHand*: We add the appropriate cadence chord, depending on the type of the cadence: perfect or imperfect.
- To the *rightHand*: We add a rest of the right duration.
- For the remaining time of the measure, we add a chord to the *leftHand* and a sequence of notes to the *rightHand*, as shown above.

If the measure contains a rest:

- We add the rest to both the *leftHand* and the *rightHand*.
- For the remaining time of the measure, we add a chord to the *leftHand* and a sequence of notes to the *rightHand*, as shown above.


Finally, after we have mapped all the notes, we combine the two *Pattern* objects, *leftHand* and *rightHand*, into one object *poemPattern*, to which we specify the tempo that we set earlier, based on the mood of the poem.

3.5 The output

The output of this system consists of three components:

- The *music* played through the *play* method of the *JFugue Player* class. This method receives the music score in Staccato format and plays it to the speaker.
- We also convert the music score from the Staccato format to the more general *MusicXML* format and generate a file in this format. For this purpose, we use classes and methods specifically designed in *JFugue*.
- In the application interface, we display some information obtained from the poem's analysis: the *mood* of the poem, the *tempo* and the *scale* of the song. This information helps the user check whether the poem has been interpreted correctly.

3.6 The Graphical User Interface

For creating the GUI, we use the JavaFX library. The interface is intuitive and easy to use. Pressing the “open folder” button  a browse window is opened, which allows the user to select a file containing a poem.

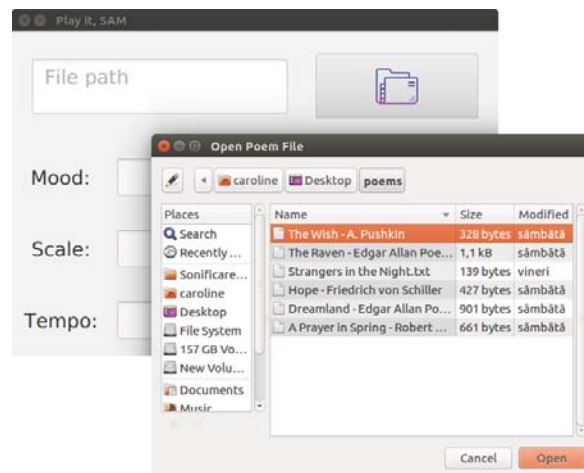




Figure 3. Selection of a file containing a poem

Next, the user can press the “play button” . This causes the application to start analyzing the poem and display the information in the three text boxes,

as shown in Figure 4. Also, the generated song starts playing, which can be stopped by pressing the stop button . Then the user can re-run the application for the same file or select another poem.

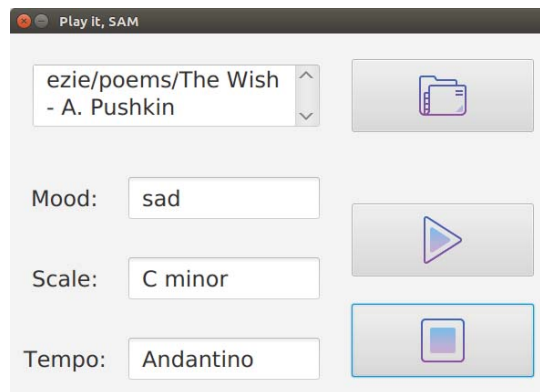


Figure 4. The output of the application

Figure 4 shows that the application classified Pushkin's poem *The Wish* as sad, which is why the generated composition is played in C minor and the tempo is Andantino.

4. Testing the application

The application works correctly, creating a harmonious sound accompaniment for poems. The generated music evokes the general feeling of the poem, by using a major or minor scale and changing the tempo according to the mood of the text.

In the following, we give a few examples that support this statement. Although deciding the mood of a poem is a subjective matter, we have chosen poems in which the general emotion is rather obvious.

Sad poems

- *The Raven*, by Edgar Allan Poe

*Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore,*

*While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
"Tis some visitor," I muttered, "tapping at my chamber door-
Only this, and nothing more." (Poe, 2017)*

- *The Wish, by Aleksandr Pushkin*
*I shed my tears; my tears – my consolation;
And I am silent; my murmur is dead,
My soul, sunk in a depression's shade,
Hides in its depths the bitter exultation.
I don't deplore my passing dream of life --
Vanish in dark, the empty apparition!
I care only for my love's infliction,
And let me die, but only die in love! (Pushkin, 2017)*

Figure 5 shows how the application interprets the poems listed above. For both poems, the mood is correctly identified as sad, which is why the song is played in C minor, and the tempo is a slow one.

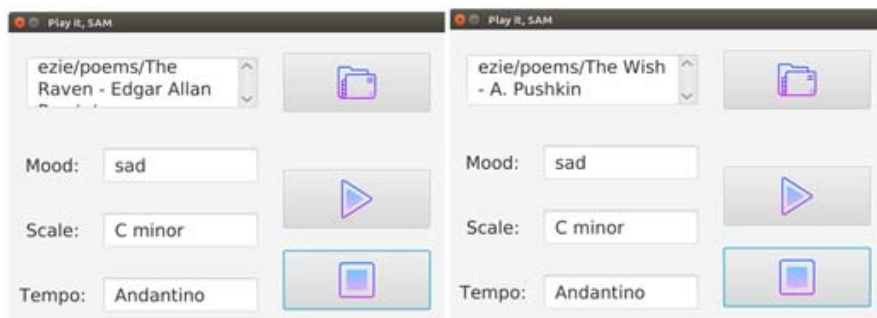


Figure 5. Running the application on sad poems

Happy poems

- *A Prayer in Spring, by Robert Frost*
*Oh, give us pleasure in the flowers today;
And give us not to think so far away
As the uncertain harvest; keep us here
All simply in the springing of the year. (Frost, 2017)*

- *Hope, Friedrich von Schiller*

*At the threshold of life hope leads us in--
 Hope plays round the mirthful boy;
 Though the best of its charms may with youth begin,
 Yet for age it reserves its toy.* (Schiller, 2017)

The following figure shows how the application interprets the poems listed above. For both poems, the mood is correctly identified as happy, which is why the song is played in C major, and the tempo is Moderato.

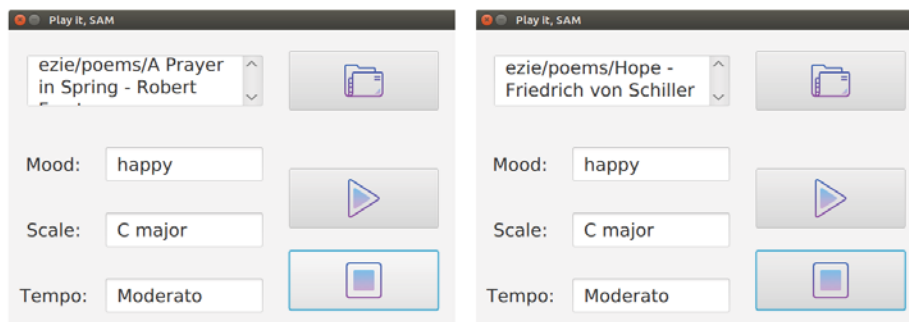


Figure 6. Running the application on happy poems

The application also generates a MusicXML file, which contains the music score of the composition. This way, it can be saved, and analyzed or played on another occasion. The following figure shows an excerpt of a MusicXML file generated by our app.

```

    <note>
      <pitch>
        <step>C</step>
        <octave>4</octave>
      </pitch>
      <duration>16</duration>
      <type>whole</type>
    </note>
    <note>
      <chord/>
      <pitch>
        <step>E</step>
        <octave>4</octave>
      </pitch>
      <duration>16</duration>
      <type>whole</type>
    </note>
  
```

Figure 7. An excerpt from a MusicXML file generated by our system

The information is rather difficult to follow by the human eye. So, we used a web application called SoundSlice MusicXML Viewer (SoundSlice, 2017) which extracts the notes from a MusicXML file and places them on a staff, so they are easier to read by a human.

Figure 8 shows the music generated from the first stanza of the poem *Hope* by Friedrich von Schiller.



Figure 8. The staff of the song generated for Hope

5. Future development

We currently analyze the general mood of the poem. This way, the tempo and the scale of the song are set from the very start, causing the song to evoke a single emotion all along, which is only partially correct.

Aside from the general theme, be it cheerful or sad, a poem conveys various other emotions: a bit of hope in a dark poem, a melancholic thought in an otherwise cheerful text and so on. In the future, we intend to analyze the intermediate emotions too, so that the song can evoke the emotional changes in the text. We can alter the mood of the song by changing the scale (a process called *modulation*) or the tempo.

For sentiment analysis, we could use the lexicon NRC Word-Emotion Association Lexicon (EmoLex, 2017). This lexicon maps words to the evoked emotions (fear, anger, happiness, wonder, disgust, hope or sadness) and the general feeling (positive or negative).

We also intend to analyze the musicality of a text from other points of view, apart from the syllables' accent and the punctuation marks. For example, we could identify *repetitions* (repetitive use of a word or combinations of words) and “translate” them into similar melodic sequences. Or end the verses that *rhyme* with similar melodic sequences.

As one can see, our system is just at the beginning. There are many ways in which the generated song can be improved, ways that we intend to implement soon.

6. Conclusions

We implemented a system that analyzes a poem and, based on its text, generates a piano composition. To this purpose, we classify the poem by the general emotion, analyze the rhythm of the verses and then, based on this information, generate a piano composition with two voices. The song generation is done in two steps: first we build the rhythm (i.e. the notes durations, organized in four-quarter notes measures) and then we map the note durations to musical notes.

The system also has an easy-to-use graphical interface, which allows the user to select a text file containing a poem and run the algorithm. Next, it displays some general information (mood of the poem, tempo and scale of the song) and the song starts playing. The playback can be stopped at any time by pressing the stop button. The performed tests show that the system generates a harmonious song, appropriate to the input poem.

By developing this project, we have explored the connection between literature and music. These two art forms have been at the basis of human society since ancient times, influencing and shaping it, and being in turn influenced by its progress. Looking at things from this perspective, the action of generating one based on the other is only logical and more than welcome.

References

- Bartlett, S., Kondrak, G., Cherry, C. (2009) *On the Syllabification of Phonemes*, NAACL-HLT 2009
- Berariu, T. (2014) *Învățare Automata – Laboratorul 11: Învățare Bayesiană*, Departamentul de Calculatoare, Universitatea Politehnica din Bucuresti
- Davis, H., Mohammad S.M. (2014) *Generating Music from Literature*, in Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLfL) @ EACL 2014, pp. 1–10
- CMUDICT (2017) *The CMU Pronouncing Dictionary*. Downloaded from <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>, accessed on June 27, 2017
- Million Song Dataset (2017) Downloaded from <https://labrosa.ee.columbia.edu/millionsong/>, accessed on June 27, 2017
- MusicMood (2017) *MusicMood Web Application*. Downloaded from <http://rasbt.pythonanywhere.com> accessed on June 27, 2017
- Emolex (2017) *NRC Word-Emotion Association Lexicon*. Downloaded from

- <http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>, accessed on June 27, 2017
- Frost, R. (2017) *A Prayer in Spring*. Downloaded from <https://www.poemhunter.com/poem/a-prayer-in-spring/>, accessed on June 27, 2017
- Poe, E.A. (2017) *The Raven*. Downloaded from <https://www.poetryfoundation.org/poems-and-poets/poems/detail/48860>, accessed on June 27, 2017
- Pushkin, A. (2017) *Poems*. Downloaded from http://famouspoetsandpoems.com/poets/alexander_pushkin/poems/1909.html, accessed on June 27, 2017
- Raschka, S. (2014) *MusicMood – A Machine Learning Model for Classifying Music by Mood Based on Song Lyrics*. Downloaded from <https://sebastianraschka.com/blog/2014/musicmood.html>, accessed on June 27, 2017
- Schiller, F. (2017) *Hope*. Downloaded from <https://www.poemhunter.com/poem/hope-16/>, accessed on June 27, 2017
- SoundSlice (2017) *Free MusicXML Viewer*. Downloaded from <https://www.soundslice.com/musicxml-viewer/>, accessed on June 27, 2017
- TransProse (2017) *Translating Literature into Music*. Downloaded from <http://www.musicfromtext.com>, accessed on June 27, 2017
- Williams, V. (2017) *Composing a Melody – General Tips*. Downloaded from <https://www.mymusictheory.com/for-students/grade-5/58-12-composing-a-melody-general-tips>, accessed on June 27, 2017