Probabilistic generation of ragtime music from classical melodies

Joel Michelson, Hong Xu, and Phillip B. Kirlin

Rhodes College, Department of Mathematics and Computer Science,
Memphis TN 38112, USA,
{micjp-17, xuho-17, kirlinp}@rhodes.edu

Abstract. This paper examines the computational problem of taking a classical music composition and algorithmically recomposing it in a ragtime style. Because ragtime music is distinguished from other musical genres by its distinctive syncopated rhythms, our work is based on extracting the frequencies of rhythmic patterns from a large collection of ragtime compositions. We use these frequencies in two different algorithms that alter the melodic content of classical music compositions to fit the ragtime rhythmic patterns, and then combine the modified melodies with traditional ragtime bass parts, producing new compositions which melodically and harmonically resemble the original music. We evaluate these algorithms by examining the quality of the ragtime music produced for eight excerpts of classical music alongside the output of a third algorithm run on the same excerpts; results are derived from a survey of 163 people who rated the quality of the ragtime output of the three algorithms.

Keywords: algorithmic composition, ragtime, corpus-based study

1 Introduction

Ragtime is a musical genre that is best described by its syncopated, or ragged, rhythms. Studies of ragtime compositional techniques have concluded that syncopation is the single unifying characteristic of the genre. Though widely assumed to be a term only applied to piano music, ragtime encompasses a wide variety of instrumentations, techniques, and styles [1]. Widespread conceptions and misconceptions regarding the ragtime genre have recently led to the creation of a corpus of digitized ragtime compositions in the MIDI file format [10] to enable large-scale studies of the ragtime music. Not surprisingly, concurrent with the development of this corpus and others like it is the rise of data-driven studies in music informatics, with large-scale data sets being used to discover or confirm trends and tendencies about classical and pop music alike. [6, 3]. In particular, it is tempting to apply corpus techniques to the field of algorithmic composition in order to automatically extract the unifying characteristics of a musical genre and apply those patterns in a new composition. This is the problem we examine here: we test the feasibility of composing ragtime music based solely on probabilistically applying rhythmic patterns extracted from a corpus of ragtime music to existing classical compositions. Specifically, we develop two databases of rhythmic patterns derived from a corpus of roughly 5,000 ragtime pieces and propose two algorithms that alter the rhythms of existing classical melodies to sound more like the ragtime rhythms in the databases. We evaluate these algorithms through a survey of 163 people who rated the quality of the ragtime music produced by these algorithms.

2 Methodology and Algorithms

The goal of this research is to study the feasibility of algorithmically composing ragtime music based solely on realigning existing classical music melodies to fit into ragtime rhythms. We algorithmically discover what rhythms are appropriate in ragtime by using a previously-created corpus of ragtime compositions, known as the RAG-collection, or the RAG-C data set. This data set is a collection of 11,591 MIDI files of ragtime music originally introduced by Volk and de Haas [10] as a first effort in putting together a large-scale database of ragtime music. These ragtime compositions were originally compiled and organized by a group of ragtime music enthusiasts collaborating over the internet, and are sourced from various original ragtime scores, from piano rolls translated into the MIDI format, and from recordings of performances as well. Though the data set contents vary in quality in terms of the MIDI translations, it is probably the most comprehensive collection of ragtime music in a symbolic digitized format known.

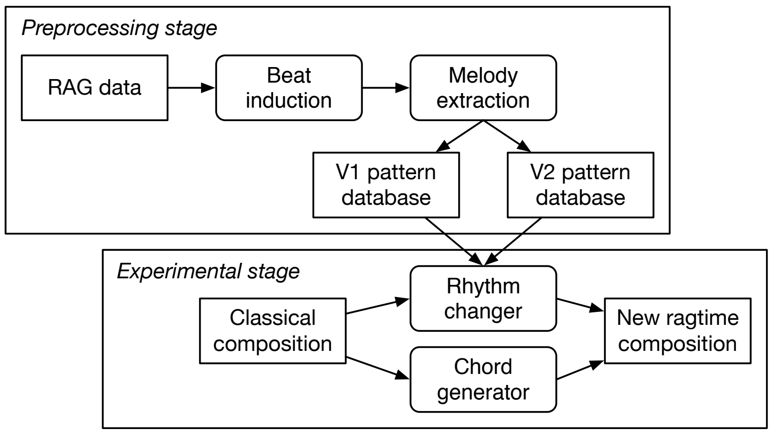

Fig. 1. Methodological setup

Figure 1 illustrates the components in our research setup, which broadly consists of a preprocessing stage and an experimental stage. The goal of the preprocessing stage is to create two databases of rhythmic patterns. We do this by processing the Rag-c data set with a beat induction algorithm; this is a necessary step as the MIDI files in the corpus do not contain enough information to automatically discover rhythmic information such as time signatures or measure boundaries. After the beat induction algorithm aligns the music metrically, we extract the melodies as monophonic sequences of notes using a standard skyline algorithm. This leaves us with a series of measures which we further process into two rhythmic pattern databases.

The goal of the experimental stage is to allow one to transform an existing classical music composition into a ragtime composition. Beginning with a classical music composition, we identify the monophonic main melody of the composition and its harmonic chordal structure. We send the main melody to the rhythm changer algorithms, which alter the metrical placement and duration of the notes in the main melody — but not their ordering — to make the melody sound more like ragtime. This is accomplished with the help of the information in the rhythmic pattern databases. At the same time, we use the harmonic structure of the classical music to compose a prototypical ragtime bass line, which is combined with the altered melody into a final composition in a ragtime style.

We give further details of each step of this process below.

2.1 Beat and measure detection

Though MIDI files can be encoded to include information such as time signature and tempo, the files in the Rag-c data set are derived from a variety of sources, including live performances, some of which do not encode these data. Therefore, we use a version of Dixon and Cambouropoulos's beat detection algorithm [4] to estimate the locations of musical beats in the MIDI data.

The algorithm's beat inducer operates by computing inter-onset intervals, or IOIs — times between pairs of note onsets — for the input MIDI file and then clustering them in the hope that small differences in the IOIs will be smoothed out. The clusters are then ranked by size, and the top-ranked clusters usually correspond to the inter-beat interval or a fraction thereof. With a correctly-predicted inter-beat interval, measure boundaries can be easily calculated for the entire piece.

In practice, however, the inter-beat interval predictions may be slightly miscalculated. We noticed a certain amount of "temporal drift" in the measure boundary predictions for the Rag-c data set. Specifically, as one predicts measure boundaries further and further ahead in a MIDI file, the predicted boundaries deviate more and more from their true locations, most likely due to the accumulation of small errors caused by a slightly miscalculated inter-beat interval. To remedy this, we returned to the top-ranked clusters calculated by the beat induction algorithm, and examined every possible inter-beat interval within 12 MIDI ticks of the cluster's interval, calculated to the tenth of a tick. For every potential inter-beat interval, we calculated the predicted measure boundaries for that interval over the entire MIDI file, then binned all the notes of the MIDI file according to which 16th note of the predicted measure they would fall into. For correctly-predicted measure boundaries, we would expect this frequency distribution of notes across the 16 bins to be weighted more heavily towards the bins corresponding to strong beats, simply because it is more common — even in ragtime — for notes to occur on strong beats. For incorrectly-predicted measure boundaries, we would expect this distribution to be flatter. Therefore, we chose the predicted set of measure boundaries that produced a frequency distribution with the highest standard deviation as our correct measure boundaries.

2.2 Melody extraction and pattern database construction

We use an adapted version of Temperley's streamer algorithm [8] to split the Rag-c MIDI files into streams of notes. In order to isolate the main melodic voice in each file, a skyline algorithm is used as described in [9, 10] to select a set of notes from these streams using the average pitch of all the notes in a given stream as its height.

Though the Rag-c data set contains over 11,000 MIDI files, we used a specific set of 5,176 for this project. We omitted all files with changing tempos — mostly from live performances — because our algorithm for detecting measure boundaries assumed a fixed tempo. Additionally, there were many excessively long MIDI files — containing many repeats of sections of the music — that could not be processed by the melody extraction algorithm due to memory limitations.

Recall that our ultimate goal is to produce new ragtime music by realigning classical music melodies to fit ragtime rhythms. In order to choose ragtime rhythms appropriately during the algorithmic composition phase, we analyze the rhythms of the melodies in the 5,176 MIDI files. We do this by assuming all the ragtime compositions are in 2/4 or 4/4 time (a reasonable assumption for ragtime), and segment each piece at the level of a 4/4 measure (merging consecutive measures of 2/4 pieces). We represent the rhythm of each 4/4 measure using the method described in [10, 5]: the rhythm of a measure is described by a pattern "I"s and "O"s specifying the locations of the note onsets at the granularity of a 16th note: an "I" standing for an onset and "O" standing for no onset at that time. For instance, a 4/4 measure with a quarter note on every beat would be represented by the string "`IOOOIOOOIOOOIOOO`." We refer to these strings as *binary onset patterns* because their contents can be represented by 1s and 0s instead of Is and Os.

Once every ragtime composition is converted into a sequence of binary onset patterns, we create two rhythmic pattern databases, Version 1 (V1) and Version 2 (V2). The V1 database simply records the frequencies of every possible binary onset pattern observed in the melodies of the 5,176 ragtime MIDI files. The V2 database records the frequency of *transitions* between binary onset patterns corresponding to every pair of adjacent measures in the corpus. In Section 3, we describe some noteworthy facts that can be learned from examining the information in the rhythmic pattern databases.

2.3 Experimental phase

The experimental phase is designed to harness the information in the pattern databases in order to produce new ragtime compositions from classical music files. For testing and evaluation, we use a set of eight excerpts of classical music. These excerpts are taken from “Dance of the Sugar Plum Fairy” by Tchaikovsky; “Eine kleine Nachtmusik,” by Mozart; Concerto No. 1 in E major, Op. 8, RV 269, “Spring” by Vivaldi; three Christmas carols: “Deck the Halls”, “Hark! The Herald Angels Sing”, “Jingle Bells”; and two traditional tunes: “Old MacDonald Had a Farm” and “Yankee Doodle.” We chose these pieces for their easily-identified melodies and duple meters.

We encoded the melody and harmony separately for these eight testing files and used them as input to the chord generation and rhythm changing algorithms, described next.

2.4 Chord generation

The chord generation algorithm generates a ragtime-style bass line consisting of a sequence of chords. The input to the algorithm is a sequence of chord symbols, in this case from one of the eight classical music excerpts which have had their harmonies manually labeled. We turn these chord symbols into into ragtime-style chord progressions based on a subset of the guidelines prescribed in [2]. We use a straightforward algorithm: we choose octaves or single bass notes on the first and third beats of a measure and chords on the second and fourth beats. The first beat is always the root of the current harmony, and the third beat is always the fifth. Additionally, we stochastically change some of the second- or fourth-beat chords into passing tones if the surrounding harmonic structure allows for this transformation. We found a probability of 1/6 works well for choosing whether or not to insert a passing tone.

As an example, Figure 2 shows the bass line generated from the first eight measures of the fourth movement of Beethoven’s Ninth Symphony (“Ode to Joy”). Notice how there is a passing tone generated in the transition from the end of measure 4 into measure 5.

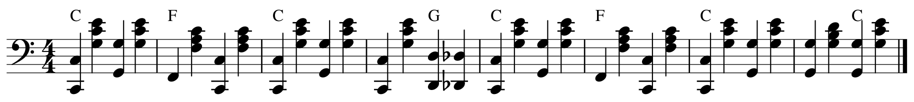

Fig. 2. Illustration of chord generation for “Ode to Joy.” The chord symbols above the staff are used as input, and the notes displayed are the output. Note the passing tone in measure 4.

2.5 Rhythm changing algorithms

At the heart of this algorithmic composition system is the rhythm changing algorithm. Recall that our goal is to adjust the rhythm of a classical melody to fit a ragtime rhythm. We do this by identifying, for every measure of the classical input composition, a corresponding ragtime measure with the same number of notes, and altering the classical measure to fit the ragtime measure's rhythm. For example, consider Figure 3, which shows (on the top left) a measure of music taken from the Christmas carol "Deck the Halls," and also (on the top right) a measure taken from the ragtime composition "The Entertainer," by Scott Joplin. The rhythm changer would combine these measures into the new measure of music at the bottom of the figure, which aligns the notes of the "Deck the Halls" melody with "The Entertainer"'s rhythm.

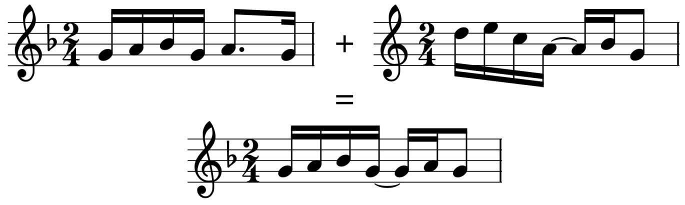

Fig. 3. An example of the rhythm-changing algorithm. We combine a classical melody (top left) with a ragtime rhythm (top right), producing a new measure of ragtime-sounding music (bottom).

We develop and evaluate two different strategies for using the rhythm changer in conjunction with the V1 and V2 rhythmic pattern databases.

Recall that the V1 rhythm database simply stores the frequency of every binary onset pattern in the corpus of ragtime MIDI files. Given a piece of classical music as input, our goal is to probabilistically generate a set of rules that map every binary onset pattern in the input music to a ragtime binary onset pattern, to which we then apply the rhythm changer algorithm as described above. We generate this set of rules by enumerating all the onset patterns in the input classical composition on a measure-by-measure basis, and sorting them in order of descending frequency. For every one of the classical onset patterns, we choose a corresponding ragtime onset pattern proportionally to its frequency in the V1 database (keeping in mind that the number of onsets in the two patterns must be equal), and then create a rule associating those two binary onset patterns. There are two caveats. First, because un-syncopated rhythms (e.g. `I000I000I000I000`) are so common in the data, rules that map an onset pattern to itself are never permitted. Additionally, rules that would shift onsets by a total of more than eight positions (where a position is an individual 16th note shift) are rewritten to prevent drastic changes, such as a note at the beginning of a half measure being shifted to the end of the measure. This technique is presented as Algorithm 1.

The strategy for using the V2 database is similar to that of the V1 database, except we examine *pairs* of binary onset patterns in the classical input and in the ragtime corpus. This technique is presented as Algorithm 2.

Algorithm 1 Version 1

```
for each unique binary onset pattern $X$ in the input song do
    Count the number of onsets in $X$
    while a rule has not yet been generated do
        Choose a random binary onset pattern $Y$ with the same number of onsets in
        the data set, weighted by its frequency
        if $Y \neq X$ and onset shifting distance from $X$ to $Y \leq 8$ then
            Generate new rule $X \to Y$
        end if
    end while
end for
for each measure in the input song with binary onset pattern $X$ do
    Generate measure in output song with note positions $Y$ from the rule $X \to Y$ and
    pitches from the original measure
end for
```

Algorithm 2 Version 2

```
for each unique binary pair of onset patterns $X$ and its subsequent measure $Y$ in
the input song do
    Count the number of onsets in $X$ and the number of onsets in $Y$
    while a rule has not yet been generated do
        Randomly select a transition $Z$ from the transition table in which a measure
        with count of $X$ transitions into a measure with count $Y$
        if $Z \neq Y$ and onset shifting distance from $Y$ to $Z \leq 8$ then
            Generate new rule $Y \to Z$
        end if
    end while
end for
for each measure in the input song with binary onset pattern $Y$ do
    Generate measure in output song with note positions $Y$ from the rule $Y \to Z$ and
    pitches from the original measure
end for
```

2.6 Syncopalooza Rhythm Changer

To serve as a baseline algorithm, we implemented the Syncopalooza algorithm as described in [7]. This algorithm is neither data- nor corpus-driven, but rather manipulates the syncopations in a composition by shifting note onsets to stronger or weaker metrical positions individually, rather than by rewriting an entire measure of rhythm at once.

3 Results, Survey, and Evaluation

Some interesting results can be gleaned from the V1 pattern database which records the frequency of every binary onset pattern in the 5,176 ragtime compositions. In general, the V1 database frequencies display a long-tailed distribution as can be seen in Figure 4; the correspondence between the frequency of a pattern and its rank in the list follows a power law relationship. Furthermore, the most commonly-occurring binary onset patterns in ragtime music do not correspond to syncopated rhythms at all, but rather to simple rhythms such as a measure of two regularly-spaced half notes (the most frequent pattern), a measure with one whole note (the second-most frequent pattern), or a measure of four regularly-spaced quarter notes (the third-most frequent pattern). Though the data set contains 8,803 different binary onset patterns, these three patterns account for roughly 11% of all the measures in the corpus. It is noteworthy that the fourth-most common rhythm, with onsets on beats 1, 2, and 4 (but not 3), displays the characteristic "short-long-short" pattern of note durations which is especially prevelant in ragtime [5].

In order to evaluate the quality of the rhythm-changer algorithms presented earlier, we conducted two separate surveys. The surveys differed in length and in the participant demographics, but contained the same basic type of question. Each question in the survey asked the participant to listen to three different algorithmically-produced ragtime excerpts, one each derived from the V1 and V2 databases paired with the rhythm-changer algorithm, and the third from the Syncopalooza algorithm. The order of the three excerpts was randomized for every question. After listening to each excerpt as many times as the participant desired, they were asked how much they agreed with the statement "This excerpt sounds like ragtime" for each of the three excerpts. Their answers were recorded on a five-point Likert scale, with the choices of strongly disagree (1), disagree (2), neither agree nor disagree (3), agree (4), and strongly agree (5).

The first survey was taken by 33 college undergraduates with some familiarity with ragtime music. Each participant was asked to evaluate 6 sets of excerpts (listening to 18 excerpts in total) according to the schema above. The second survey was taken by 130 different people solicited from internet message board about piano music.

The college students' average responses to the questions on the Likert scale were 3.50, 2.83, and 2.99 for Syncopalooza, V1, and V2, respectively; while the internet users' average responses were 3.36, 2.55, and 2.64, respectively. These values indicate that while the college students rated the output of all three algorithms slightly higher than the general internet population did, both groups preferred Syncopalooza to the algorithms presented in this study, though by less than one point. Furthermore, because even the best-performing algorithm —

Syncopalooza — did not surpass a 3.50 rating, there is clearly plenty of room for improvement in the algorithms.

Figure 5 illustrates the survey results grouped by the classical music piece used as input. We can see that Syncopalooza consistently outperforms both the V1 and V2 algorithms, though there are cases where all three scores are clustered closely together. It is also noteworthy that neither V1 nor V2 consistently outperforms the other; their ratings are usually close together.

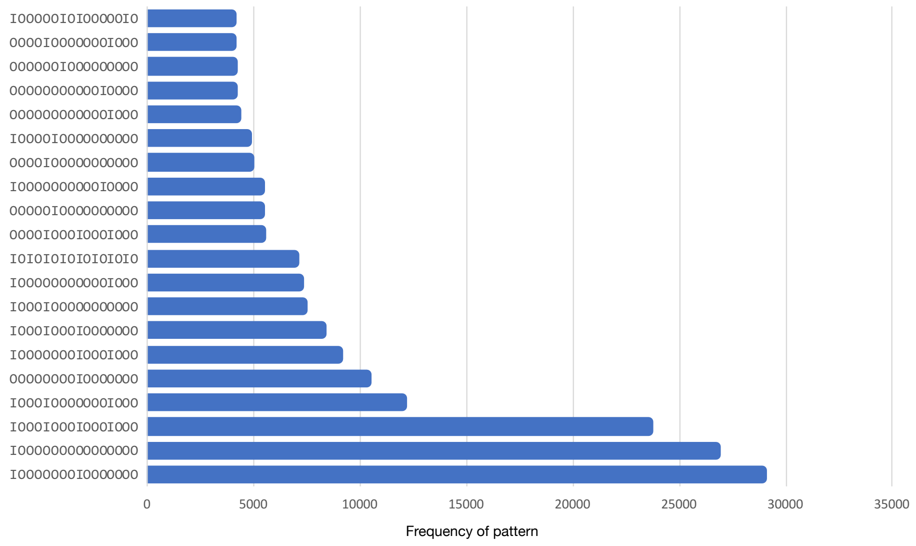

Fig. 4. This graph illustrates the frequencies of the 20 most common binary onset patterns found in the ragtime corpus.

Anonymous comments solicited from the survey participants revealed some of the reasons for their ratings. Multiple users noted some inconsistencies and mismatches between the beats of the melody and the bass notes of the accompaniment. We believe this may be due to (1) the algorithms moving notes that were originally consonant with their underlying harmonies to new metrical locations that then become dissonant with the underlying harmonies; (2) the algorithms choosing ragtime rhythmic patterns that, while technically common, do not "flow" with the preceding or following music; (3) beat induction errors. Multiple users also commented on the low register of the accompaniment chords; some thought this rendered the audio muddy and distracted from the overall sound. We plan on investigating these issues fully in the next iteration of this project.

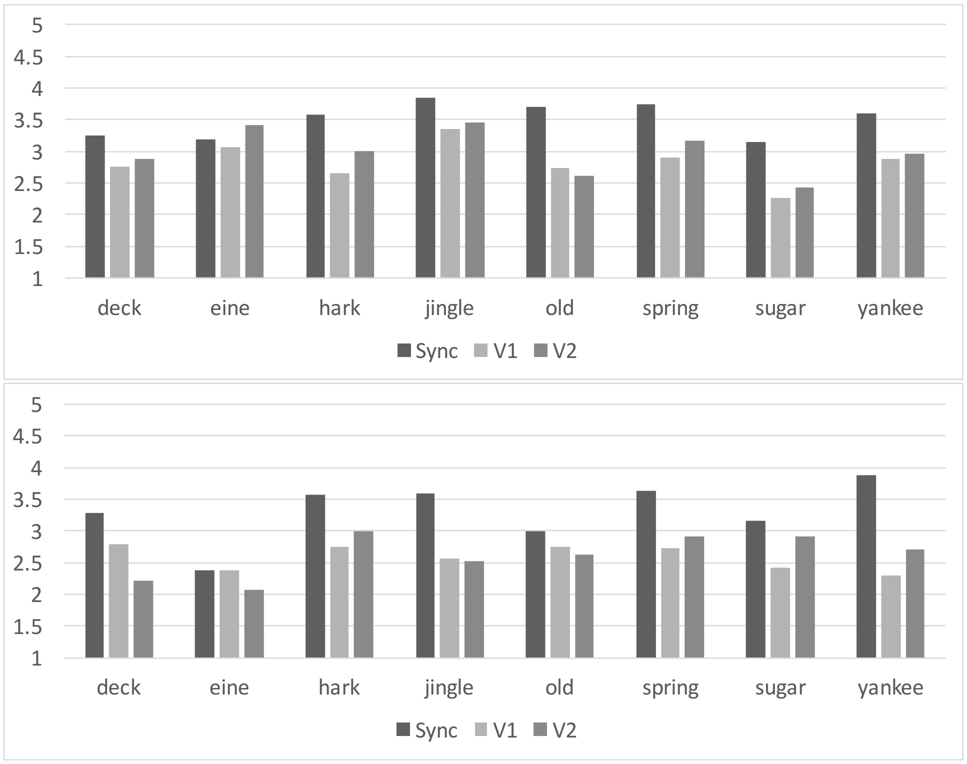

Fig. 5. Survey responses from college undergraduates (top), and internet respondents (bottom), separated by classical input composition and by ragtime algorithm (Syncopalooza, rhythm shifter version 1, rhythm shifter version 2). Participants were asked how much they agreed that the output music sounded like ragtime.

References

1. Berlin, E.A.: Ragtime. In: Grove Music Online. Oxford Music Online. Oxford University Press (March 2017), http://www.oxfordmusiconline.com/subscriber/article/grove/music/A2252241, web
2. Bradlee, S.: Ragtimify: How to Turn Any Song into Ragtime and Stride Piano. Scott Bradlee (2013)
3. de Clercq, T., Temperley, D.: A corpus analysis of rock harmony. Popular Music 30(1), 47–70 (2011)
4. Dixon, S., Cambouropoulos, E.: Beat tracking with musical knowledge. In: Proceedings of the 14th European Conference on Artificial Intelligence. pp. 626–630 (2000)
5. Koops, H.V., Volk, A., de Haas, W.B.: Corpus-based rhythmic pattern analysis of ragtime syncopation. In: Proceedings of the 16th International Society for Music Information Retrieval Conference. pp. 483–489 (2015)
6. Rohrmeier, M., Cross, I.: Statistical properties of tonal harmony in Bach's chorales pp. 619–627 (2008)
7. Sioros, G., Miron, M., Cocharro, D., Guedes, C., Gouyon, F.: Syncopalooza: Manipulating the syncopation in rhythmic performances. In: Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research. pp. 454–469 (2013)
8. Temperley, D.: A unified probabilistic model for polyphonic music analysis. Journal of New Music Research 38(1), 3–18 (Mar 2009)
9. Uitdenbogerd, A.L., Zobel, J.: Manipulation of music for melody matching. In: ACM Multimedia. pp. 235–240 (1998)
10. Volk, A., de Haas, W.B.: A corpus-based study on ragtime syncopation. In: Proceedings of the 13th International Society for Music Information Retrieval Conference. pp. 163–168 (2013)