COMPUTER AIDED MELODIC ANALYSIS USING SUFFIX TREE

Matevž Jekovec, Janez Demšar, Andrej Brodnik

Faculty of Computer and Information Science
University of Ljubljana, Slovenia
{matevz.jekovec,janez.demsar,andrej.brodnik}@fri.uni-lj.si

ABSTRACT

Quality melodic analysis of the score is one of the most significant, but also awkward and time consuming musicological tasks. In this paper we propose a computer aided melodic analysis based on suffix tree. Our data structure represents hierarchically combined melodic patterns for a given score. The potential interestingness of melodic patterns to the musicologist is then estimated from their diversity, length and frequency. We tested the proposed method on 48 fugues from J. S. Bach's Well-Tempered Clavier opus. All our approaches were integrated into the free musicological application *Harmonia*, which allows musicologists to explore the most common theme, its submelodies, common motifs and other melody-based score features.

1. INTRODUCTION

Melodic analysis is a category in the systematic musicology. It investigates where and how the imitation compositional technique was used, what kind of themes, motifs and phrases were composed, how many times and on which degree the theme or a motif was imitated, what variations were introduced and similar. Analysing the compositional techniques used in the score is one of the most awkward and time consuming tasks, but to our knowledge there are no widely used methods or computer software packages for musicologists. In this paper we design a method to detect and store significant melody patterns found in the analysed score.

Temperley [9] used prediction to analyse and predict the melody contour in German folk songs. He showed that probability is not useful only for analysing or fixing the score but also for composing new melodies. Weyde [11] proposed the segmentation of the motifs. He showed that the motif candidates can be split into multiple segments based on the internal melodic and rhythmic similarities. His approach is important because he combined submelodies from different melodies in a similar way as the method presented in this paper does. Takasu et al. [8] proposed a complex approach to automatically extract the theme phrase from the score stored in MIDI format. They used the submelodic similarities using the Longest Common subsequence, a decision tree to classify the candidates based on their pitch and frequency and improved the results by using a finite state machine built on subsequent submelodies of three. We used similar melody attributes in our research to build a simpler version of the melody evaluation function. Hsu et al. [3] introduced an RP-tree based on the covariance matrix of the repeating substrings in the score. They argued that the suffix tree itself was too time and space consuming for their use case. In this paper we show that using the path compression technique and correctly choosing the characters representing the score, suffix trees can be efficiently used to store and manage string suffixes.

There are not many end-user applications focused on music analysis from the musicologists point of view. JRing [5] allows importing MIDI files, annotating musical features and exporting results to a text file for further processing. Analysis [1] was developed during the harmony statistics research. It supports importing the MIDI file and determining the key, chord type and a tonal degree of the chord.

In this paper we propose an innovative approach for computer aided melodic analysis based on suffix tree. The approach was initially described in the bachelor's thesis [4]. In Section 2 we present the suffix tree data structure commonly used for manipulating strings. Section 3 describes a procedure to construct a suffix tree for storing the note pairs and then transforming it into the compressed suffix tree for storing the melodies. A separate Section 4 is dedicated to the evaluation of the melodies. We integrated the proposed methods into a computer program *Harmonia* and we briefly present the solution and the empirical results in 5. We conclude in Section 6.

2. SUFFIX TREE

The suffix tree data structure [10] is a trie [2] in which we insert all suffixes of a given text. Although we use this data structure to find melodic patterns, we will present it first on an example of its original use for alphabetic strings.

Suffix tree supports efficient operations on a text $w = \langle c_1 c_2 ... c_n \rangle$, where $|w| = n$ and characters are from a finite alphabet $c_i \in \Sigma$. Suffix tree, as any other data structure, can be augmented to also contain the number of leaves in each subtree, the reference to the linked list of leaves in a subtree etc., and make the operations like searching easier to implement and run faster.

The straightforward way to build a suffix tree from a text w is to insert into it all possible suffixes of w from

$\langle c_1...c_n\rangle$, $\langle c_2...c_n\rangle$ and all the way to $\langle c_n\rangle$.

Moreover, since the suffix tree is just a trie, we can merge several suffix trees into a single, combined suffix tree in a natural way. So, if we have a set of texts $\{w_1, w_2...., w_m\}$, where $|w_i| = n_i$ we insert each text w_i with all its suffixes into a combined suffix tree in the same way as the first text obtaining a final, combined suffix tree.

Finally, we do path compression on the tree by merging the nodes with only a single child (cf. Patricia, [6]). This results in a tree where edges represent more than one character. The size of a path-compressed suffix becomes $\sum_{i=1}^{m} n_i$ and therefore it is minimal. Consequently, this makes our further operations described in the following sections time and space optimal with regard to the size of the text – i.e. the analysed composition. Further details of rigorous time and space complexity of our solution are here omitted. Figure 1 shows an example of a path-compressed suffix tree built from the text BANANA.

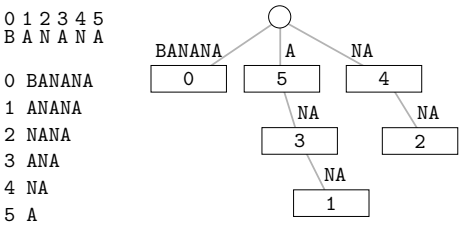

Figure 1. Compressed suffix tree for the text BANANA. Each node represents a suffix from the root to that node. The number in the rectangle denotes the position of the suffix in the text.

3. SUFFIX TREE FOR MELODIC ANALYSIS

In this section we describe how to store all the melodies in the score into suffix tree. Initially we need a music score represented in a notation format (e.g. MusicXML, MIDI). The first step is to linearise the input data to obtain voices singing at most one note at a time. We will not describe this step in this paper (details in [4]) and assume the input data is already organized in such a way. Then we re-express the melody as a sequence of note pairs. The note pairs represent our alphabet Σ and their sequence represents text w from which we construct a suffix tree. Afterwards we do path compression on the note pairs suffix tree to obtain the *melody suffix tree*.

3.1. Note pairs generation

We have a list of voices $v_1, ..., v_k$. Each voice consists of a sequence of notes $\langle p_1, r_1\rangle, ..., \langle p_m, r_m\rangle$ where p_i denotes the note's pitch and r_i denotes the note's length. We define the note's length as the difference between the note's start time and the next note's start time. This way we treat the melody played in different style (staccato, legato, with pedal) the same. If the time between the notes is too long, we introduce a new voice.

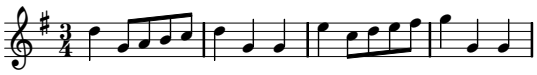

Figure 2. Melody of initial two bars of the J. S. Bach's Menuet No. 1 in G-major.

For each voice we generate note pairs. The ith note pair is defined as $\langle d_p(p_i, p_{i+1}), d_r(r_i, r_{i+1})\rangle$, where $i = 1..m-1$. Functions d_p and d_r describe note's pitch and rhythmic distance respectively. We define the following strategies for the function d_p: a musical interval based distance (quality and quantity or quantity-only), pitch difference in semitones and relative note names.

The first strategy consists of the interval quality, quantity and direction. This strategy is used to determine which melody occurs in a specific gender (eg. major, minor). It requires the note accidentals to be correctly determined. The value of the first note pair in the Figure 2 is P5↓. An alternative approach is to only keep the interval quantity and the direction. This way we detect repetitions of the same melody on different degrees (also called sequences). In the mentioned example the two lines consisting of 4 eight notes and a quarter in the first and the third bar are resolved as the same melody.

The second strategy consists of an integer describing the number of semitones between the notes. This method works well if the note accidentals are not correctly determined. This is common when importing music from the MIDI format because the format does not contain any information on enharmonic tones. The value of the first note pair in the example is -7.

The third strategy consists of the note names including the accidentals and the direction. This is useful for finding occurrences of the melody in exactly the same pitch regardless of the octave, for example when analysing the main theme in sonata form where we wish to separate same melodies in the exposition (Tonic) and the development part (Dominant). In the example, the value of the first note pair is D-G↓.

For the rhythmic distance function d_r the notes' rhythm ratio or absolute rhythm are used. The first distance is defined as r_i/r_{i+1}. This method is robust because of a rhythmic invariance to the same melodies written for a factor of shorter (*diminution*) or longer notes (*augmentation*). The value of the first note pair in the example is 2. The absolute rhythm distance is defined as $\langle r_i, r_{i+1}\rangle$ and is used when we need exact information on which melody is represented using which note lengths. The value of the first note pair in the example is $\langle 1/4, 1/8\rangle$.

3.2. Suffix tree of music patterns

First we build a suffix tree containing all voices (w_i). The alphabet of characters Σ consists of the note pairs as transformed by d_p and d_r. Note, that our voices are originating from the same piece of music, although this need not to be the case. Indeed, we could use our technique to analyse

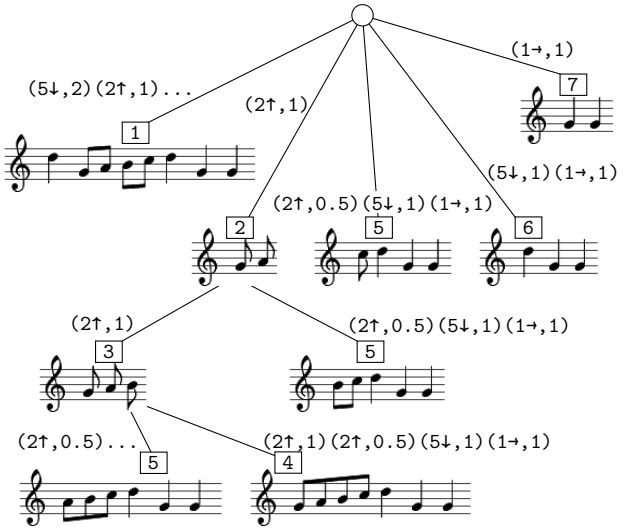

Figure 3. The m-suffix tree for the first two bars of J. S. Bach's Menuet No. 1 in G-major as seen on Figure 2. The first component in the braces contains the interval quantity and direction, the second component contains the rhythm ratio. The number in rectangle denotes the index of the of the pattern in the score. No pruning is used.

several pieces of the same or of different composers.

The second step is to prune the suffix tree based on maximum suffix length and minimum number of occurrences. The former pruning is implemented by stopping building the suffix tree at a node v of a specified depth, but still counting the number of patterns ending in a subtree rooted at v. This makes the latter optimization, where we prune the nodes corresponding to patterns with less than the prescribed number of occurrences in the sequence, feasible. We call such a suffix tree a *p-suffix tree* because it contains pairs of notes.

Finally, we do path compression the p-suffix tree as described in section 2 obtaining an *m-suffix tree* – because it contains melodic patterns or simply melodies. Figure 3 shows the m-suffix tree built for the example shown in Figure 2. We visualize it by showing the whole suffixes (melodies) in vertices from the root to the corresponding node.

4. MELODY EVALUATION

The m-suffix tree described in the previous section is comprehensive. We examined the J. S. Bach's Menuet No. 1 in G-major which consists of two voices (right and left hand) containing 146 notes and 49 notes, respectively. The p-suffix tree contained 1332 nodes and the m-suffix tree 174 nodes. After limiting the minimum number of occurrences to 2 and the maximum pattern length to 30, the number of nodes in the trees reduced to 142 note pairs and 57 melodies.

In order to determine the melodies which may be of interest to musical analysis (musicians call these figures, motifs, phrases or even a theme), we evaluate each pattern based on four attributes: pattern length, number of occurrences in the score (frequency), melodic and rhythmic diversity.

We obtain the pattern length and the number of occurrences directly from the suffix tree. To calculate the pattern's diversity we use the Shannon's normalized entropy [7]

$$H_O(p_1,..,p_k) = \frac{H}{H_{max}} = -\sum_{i=1}^{k} \frac{p_i \cdot \log p_i}{\log k}$$

We calculate the diversity by substituting $p_1,...,p_k$ with the share of the specific melodic interval or rhythm ratio where k denotes the total number of note pairs in each pattern. For instance, let's calculate the melodic diversity for the first bar in Figure 2. It contains quantity intervals $-5,2,2,2$, so $p_{-5} = 0.25, p_2 = 0.75$ and $H_O = 0,406$.

Finally, we define the function that evaluates the pattern. A pattern needs to be both long and frequent at the same time to be interesting. As the pattern grows longer, the interestingness is increasing slower; the same goes for frequency. Based on this reasoning, we compute the product of logarithms of the length and frequency. Logarithms can have different bases for balancing the importance of length and frequency. In addition to this, the pattern's interestingness depends on its melodic and rhythmic diversity as described in 4. The final evaluation function is

$$C_1 \cdot \log_{BL} k \cdot \log_{BF} f + C_2 \cdot H_O(pm_{1...k}) + C_3 \cdot H_O(pr_{1...k}),$$

where C_1, C_2 and C_3 represent weights, k is the length of the pattern and f is its frequency.

5. EVALUATION

Harmonia (http://harmonia-music.sf.net) is a free cross-platform application for music analysis. It is written in Python, uses *Canorus* score editor for score manipulation and *LilyPond* for rendering the melodies. It allows the user to import scores in MIDI and MusicXML file formats and then perform various analysis operations.

We added support for the melodic analysis described in sections 3 and 4. The most important views show the p- and the m-suffix trees. The former allows the user to observe the frequency of different intervals in the score. For instance, we analysed 48 fugues from J. S. Bach's Well-Tempered-Clavier and observed the most common sequence of intervals. These were the major and the minor seconds for the works written in the major and in the minor keys, respectively.

The m-suffix tree shown in Figure 4 allows user to view all the melodies in the score and their correlation to submelodies including the information on their locations in the score, the frequency, the melody length and the diversity. We analysed the Bach's fugues and observed the melody rank calculated by the melody evaluation function as described in section 4. Fugues have a strict form and the main theme is always present at the beginning of the

score, which allows for objective evaluation of the proposed algorithm. Our results show that among 48 fugues, 19 had the main theme in the top ten most significant melodies. This greatly helps the user as there was an average of 483 melodies per fugue. By empirically tuning all the parameters the following combination turned out to work best: pitch difference in semitones, notes' rhythm ratio, maximum theme length: 23, required min. number of occurrences: 2, $B_{freq} = 4$, $B_{len} = 60$, $C_1 = 128$, $C_2 = 23$, $C_3 = 91$.

Application also offers other functionalities. It allows user to listen to the whole score, selected voices or only specific melodies. If the analyst is interested in a specific melodic or rhythmic pattern he can search for it using the *LilyPond* syntax. The p- and m-suffix tree views allow user to flatten the tree for example to sort the results by specific attribute like the melody rank. Tight integration with *Canorus* allows selecting a melody and showing all its occurrences on all degrees in the score. *Harmonia* also provides score statistics like the note count per voice or instrument and the number of nodes in both p- and m-suffix trees. It can also visualise score statistics, for instance the melody distribution according to their length and frequency.

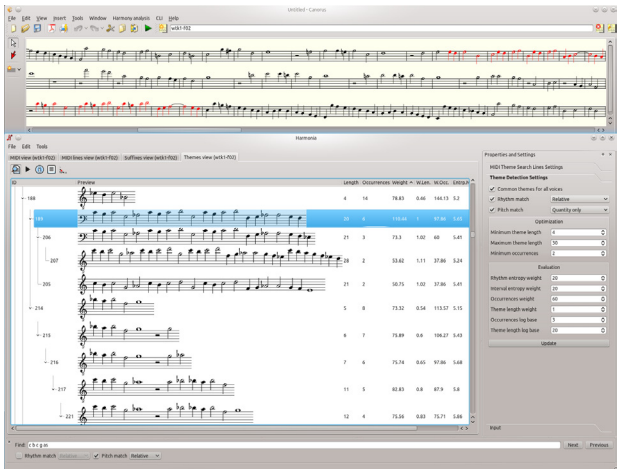

Figure 4. Screenshot of *Harmonia* showing the m-suffix tree for J. S. Bach's Fugue no. 2, WTK 1. Upon selecting the melody, all its occurrences were also selected in *Canorus* visible on top.

6. CONCLUSION

Determining the "true" theme of a musical piece is a difficult and, in general, subjective process, so completely automated analysis of melodic patterns is impossible. The best we can hope for is a computer tool to assist the analyst in the process of sorting out the frequently occurring interesting patterns.

We presented a data structure that can be used to discover, store and explore interesting melodic patterns in music, and a heuristic function for ordering the patterns by their supposed interestingness. Although we empirically showed that proposed heuristics work well, the true contribution of our work is the working application for interactive analysis of patterns, implemented as a part of *Harmonia* suite.

We believe that the proposed technique can (and needs to) be extended even further. Perhaps the most obvious direction is to improve the evaluation heuristic by incorporating some aspects of aesthetics and, even more importantly, to let the tool recognize variations of a theme as the same theme.

7. REFERENCES

[1] E. Ferkova, M. Zdimal, and P. Sidlik, "Chordal Evaluation in MIDI-Based Harmonic Analysis: Mozart, Schubert, and Brahms," *Computing in musicology*, vol. 15, p. 172, 2007.

[2] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, 1960.

[3] J. Hsu and C. Liu, "Discovering nontrivial repeating patterns in music data," *Multimedia, IEEE Transactions on*, vol. 3, no. 3, pp. 311–325, 2001.

[4] M. Jekovec, "Računalniška analiza tem v skladbah," Bachelor's Thesis (in Slovenian), 2011.

[5] A. Kornstädt, "The JRing system for computer-assisted musicological analysis," in *International Symposium on Music Information Retrieval, Bloomington, IN, USA*. Citeseer, 2001, pp. 93–98.

[6] D. R. Morrison, "PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric," *Journal of the ACM*, vol. 15, no. 4, pp. 514–534, 1968.

[7] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 1, pp. 379–423, 1948.

[8] A. Takasu, T. Yanase, T. Kanazawa, and J. Adachi, "Music structure analysis and its application to theme phrase extraction," *Research and Advanced Technology for Digital Libraries*, pp. 854–854, 1999.

[9] D. Temperley, *Music and probability*. The MIT Press, 2006.

[10] P. Weiner, "Linear pattern matching algorithms," in *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*. IEEE, 1973, pp. 1–11.

[11] T. Weyde, "Integrating segmentation and similarity in melodic analysis," in *Proceedings of the International Conference on Music Perception and Cognition*. Citeseer, 2002, pp. 240–243.