Time-span Tree Leveled by Duration of Time-span

Masatoshi Hamanaka[1], Keiji Hirata[2] and Satoshi Tojo[3]

[1] RIKEN
[2] Future University Hakodate
[2] JAIST
masatoshi.hamanaka@riken.jp

Abstract. This paper describes a time-span tree leveled by the length of the time span. Using the time-span tree of the Generative Theory of Tonal Music, it is possible to reduce notes in a melody, but it is difficult to automate because the priority order of the branches to be reduced is not defined. A similar problem arises in the automation of time-span analysis and melodic morphing. Therefore, we propose a method for defining the priority order in total order in accordance with the length of the time span of each branch in a time-span tree. In the experiment, we confirmed that melodic morphing and deep learning of time-span tree analysis can be carried out automatically using the proposed method.

Keywords: Generative theory of tonal music (GTTM), time-span tree, time-span reduction, melodic morphing, Transformer.

1 Introduction

Our goal is to automate the system using a time-span tree of the Generative Theory of Tonal Music (GTTM) [1]. GTTM consists of grouping structure analysis, metrical structure analysis, time-span tree analysis, and prolongational tree analysis. a time-span tree is a binary tree with a hierarchical structure that describes the relative structural importance of notes that differentiate the essential parts of the melody from the ornamentation.

The time-span tree in Fig. 1 is the result of analyzing a melody (a) on the basis of GTTM. Reduced melodies can be extracted by cutting this time-span tree with a horizontal line and omitting the notes connected below the line (Fig. 1 (b)–(f)). Melody reduction with GTTM is the absorption of notes by structurally important notes.

The problem with previous systems using time-span trees is that the priority order of branches of a time-span tree is not defined. The GTTM-based melodic-morphing algorithm we previously proposed was difficult to automate because it included a time-span reduction process [2, 3]. We have been developing a GTTM analyzer using deep learning and have been able to automate grouping structure analysis and metrical structure analysis using deep leaning [4, 5]. How-ever, deep learning of time-span tree analysis is difficult to automate due to the ambiguity of the reduction process.

Therefore, we propose a method for defining the priority order in total order in accordance with the length of the time span of each branch in the time-span tree, ena-

bling melodic morphing and time-span analysis to be automated. Sections 2 and 3 describe problems with implementing our melodic-morphing algorithm and time-span analysis. In Section 4 we present our proposed method for the solving the above-mentioned problems. The experiments in Section 5, we show that melodic morphing and time-span analysis can be automating by prioritizing the branches of the time-span tree. We conclude in Section 6 with a brief summary and mention of future work.

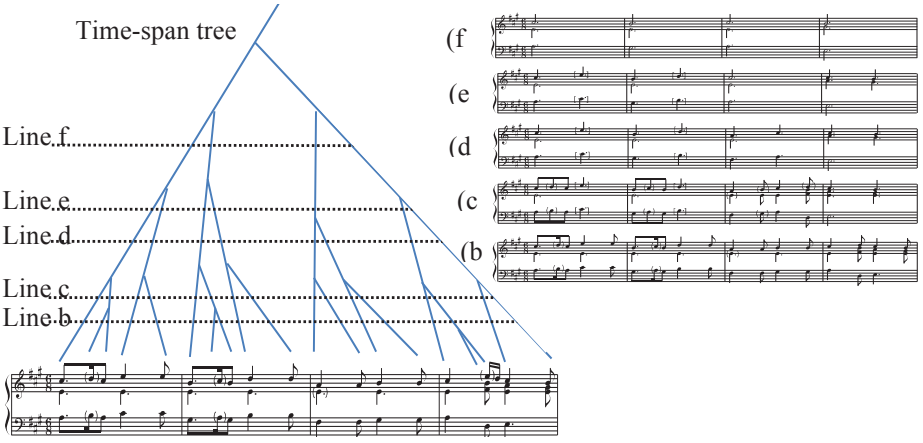

Fig. 1. Time-span tree and melody reduction

2 Implementation Problems of Melodic-Morphing Algorithm

The meaning of morphing is to change something, such as an image, into another through a seamless transition. For example, a method of morphing one face picture into another creates intermediate pictures through the following operations.
(a1) Link characteristic points such as eyes and nose, in the two pictures (Fig. 2a).
(a2) Rate the intensities of the shape (position), color, etc. in each picture.
(a3) Combine the pictures.

2.1 Ideas of Melodic Morphing

Similarly, our melodic-morphing algorithm creates intermediate melo-dies with the following operations.
(b1) Link the common pitch events of the time-span trees of two melodies (Fig. 2b).

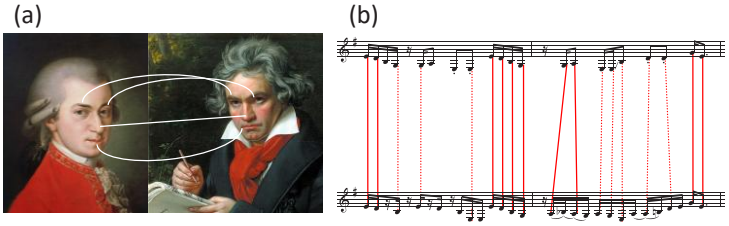

Fig. 2. Examples of linking two pictures/melodies

(b2) Remove those notes that do not reside in the common part by using partial melody reduction, which is explained in the next subsection.
(b3) Combine both melodies.

By using the time-span trees σ_A and σ_B from melodies A and B, respectively, we can calculate the common events of $\sigma_A \sqcap \sigma_B$, which includes not only the essential parts of melody A but also those of melody B (Fig. 3 (b1)). The *meet* operation $\sigma_A \sqcap \sigma_B$ is abstracted from σ_A and σ_B, and those abstracted notes that are not included in $\sigma_A \sqcap \sigma_B$ are regarded to be the difference between σ_A and σ_B.

2.2 Partial Melody Reduction

Music features contained in σ_A and σ_B should exist even in what is not included in the common part. To retrieve these characteristics, we need a method of smoothly increasing or decreasing the number of features. Partial melody reduction abstracts the notes of a melody by using reduction.

With partial melody reduction, we can first acquire melodies α_i (i = 1, 2, ···, n) from σ_A and $\sigma_A \sqcap \sigma_B$ with the following algorithm. The subscript i of α_i indicates the number of notes that are included in σ_A but not in $\sigma_A \sqcap \sigma_B$.

Step 1: Determine the level of abstraction The user determines the parameter L that determines the level of melody abstraction. Parameter L is from 1 to the number of notes that are included in σ_A but not included in $\sigma_A \sqcap \sigma_B$.

Step 2: Abstraction of notes This step involves selecting and abstracting a note that has the fewest dots, obtained from metrical analysis, in the difference of σ_A and σ_B. The numbers of dots can be acquired from the analysis results. If two or more notes have the fewest dots, we select the first one.

Step 3: Iteration Iterate step 2 L times.

Subsumption relations hold as follows for the time-span trees σ_{α_m} constructed with the above algorithm.

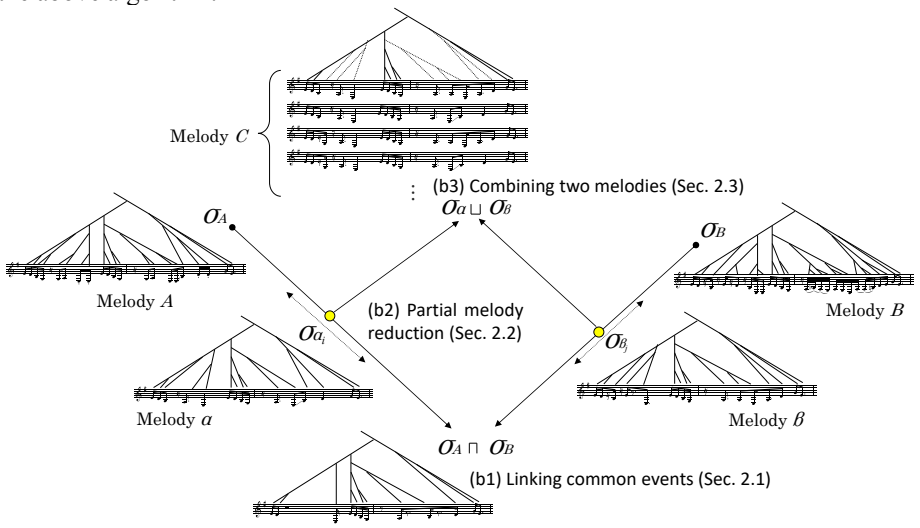

Fig. 3. Overview of melodic-morphing algorithm

$$\sigma_A \sqcap \sigma_B \sqsubsetneq \sigma_{\alpha_n} \sqsubsetneq \sigma_{\alpha_{n-1}} \sqsubsetneq \cdots \sqsubsetneq \sigma_{\alpha_2} \sqsubsetneq \sigma_{\alpha_1} \sqsubsetneq \sigma_A \quad (1)$$

In Fig. 3 (b2), there are nine notes included in σ_A but not included in $\sigma_A \sqcap \sigma_B$. Therefore, the value of n is 8, and we can acquire eight types of melody α_i (i = 1, 2, ···, n) between σ_A and $\sigma_A \sqcap \sigma_B$. Hence, melody αi attenuates features that exist only in melody A.

In the same manner, we can acquire melody β from σ_B and $\sigma_A \sqcap \sigma_B$ as follows.

$$\sigma_A \sqcap \sigma_B \sqsubsetneq \sigma_{\beta_j} \sqsubsetneq \sigma_B \quad (2)$$

2.3 Combining Two Melodies

We use the *join* operator ⊔ to combine melodies σ_{α_i} and σ_{β_j}, which are the results of the partial reduction done using the time-span tree of melodies σ_A and σ_B (Fig. 3 (b3)).

The simple *join* operator is not sufficient for combining σ_{α_i} and σ_{β_j}, because $\sigma_{\alpha_i} \sqcup \sigma_{\beta_j}$ is not always a monophony nevertheless σ_{α_i} and σ_{β_j} are monophonies. In other words, the result of the operation may become polyphony (chords) when the time-span structures overlap and the pitches of the notes differ.

To solve this problem, we introduce a special notation, [n_1, n_2], which indicates note n_1 or note n_2, as a result of $n_1 \sqcup n_2$. Accordingly, the result of $\sigma_\alpha \sqcup \sigma_\beta$ is all possible combinations of monophony.

2.4 Implementation Problems of Melodic-morphing Algorithm

Although we have given priority to automating the morphing process, our melodic-morphing algorithm has the following two problems.

Problem 1: No order of abstract notes. The first problem has to do with the order of abstract notes in partial melody reduction. In Step 2 of Section 2.2, an abstraction is made from the notes with the fewest dots, but this is not always the case, for example, in a time-span tree where there is a structurally salient note on a weak beat. In addition, we have to consider whether it is appropriate to uniquely determine the partial reduction path, as in Equation 1 in Step 3. If there are multiple paths for partial reduction, there is a possibility that more diverse melodies can be output.

Problem 2: Notes with overlapping times occur. The second problem is that the two notes overlapped temporally that may occur in the *join* of two time-span trees. In such cases, it is necessary to manually select one melody from among multiple generated melodies, and it is difficult to completely automate the morphing method. Further, the user remains in the dark as to the morphing process. In particular, it is difficult for the user to understand that the number of melodies output as a result of a number of melodic morphing changes. Even if the user understands the outline of the morphing method in Section 2, the outputs of multiple melodies may not match his or her expectations.

Our approach for automating melodic morphing is to define the order of notes abstracted by partial reduction and the order of notes selected by *join*. That is, when the time-span trees σ_A and σ_B of melodies A and B and the number of notes to be abstracted for each are determined, a unique melody, C, is obtained.

3 Implementation Problems of Deep-learning-based Time-span Tree Analyzer

There are three problems in the deep learning of time-span tree analysis, as follows.

Problem 3: Low Number of Ground Truth Data Sets. As ground truth data of the time-span tree, 300 melodies and their time-span trees are published in the GTTM database [6]. However, the number of data sets (300) is extremely small for learning deep neural networks (DNNs). For a small amount of learning data, over-fitting is inevitable, and an appropriate value cannot be out-put when unknown data are input.

In the time-span analysis by musicologists, the entire time-span tree cannot be acquired at once but gradually analyzed from the bottom up. Therefore, the minimum process of analysis is set as one data set, then the number of data sets is increased. For example, if the DNN [7] directly learns the relationship between a four-note melody and its time-span tree, the number of data sets is only one. If we consider the process of reducing one note to one data set, the number of data sets will be three, as shown in Fig. 4a.

The trained DNN estimates the melody consisting of n-1 notes that is reduced to one note when a melody consisting of n notes is input. A time-span tree for a melody consisting of four notes can be constructed by estimating four to three notes, three to two notes, and two to one note, and combining the results (Fig. 4b).

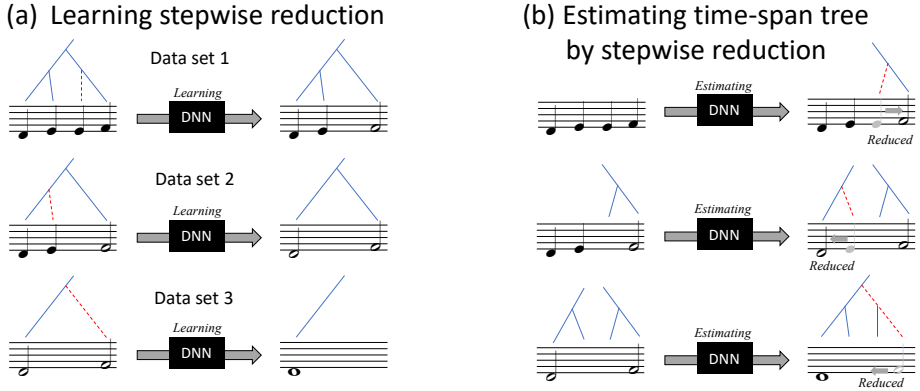

Fig. 4. Learning and estimating by stepwise reduction

Problem 4: Ambiguity of Reduction Process.

Time-span reduction removes decorative notes by pruning from the leaves at the tip of the tree, leaving only structurally important notes in the melody. To implement the stepwise reduction described above, the priority of branches must be obtained in a total order.

However, when it comes to GTTM, there are only a few examples of reduction using a time-span tree, and there is no detailed explanation on the reduction procedure [1]. For example, in Fig. 1, we can see five levels of reduction results, but it is not clear how many levels are necessary.

Marsden *et al*. [8] suggested a means of determining the salience of two note events a and b, neither of which are descendants of the other. They proposed defining the salience of an event as the duration of the maximum of the time spans of the two children at the branching point when the event is generated or where it is reduced.
To automate stepwise reduction, it is more important for the DNN to learn the relationship before and after the reduction than it is to reduce the order of the notes to close to that of human cognition. In Section 4, we propose a time-span tree leveled by the duration of the time span for a simple reduction order that is easy for the DNN to learn.

4 Solution: Time-span Tree Leveled by Duration of Time Span

The *head* in a time-span tree is the top-most pitch event, that is, the most salient in the tree. When two adjacent subtrees are combined, one of the two heads of the subtrees becomes the head of the whole. This indicates that the head of a tree is most salient in the time interval the tree occupies. Since a tree is a hierarchical combination of subtrees, the longest interval of each event in the tree is the most salient as the head of a subtree. Accordingly, we define the base case, when a subtree consists of a single pitch event, to be the duration of the event.

> **Maximum time span:** We call the longest temporal interval when a given pitch event becomes most salient as the maximum time span for the event. In other words, the maximum time span of a pitch event coincides with the temporal duration of the subtree of which the event becomes the head, as a result of the time-span analysis.

The priority of each branch of the time-span tree is determined with a time-span tree drawn with the maximum time span used in the time-span segmentation carried out as the first step of the analysis of time-span reduction. The branch priority is determined in accordance with the following rules.

- Priorities are assigned to each level from the top of the time-span tree drawn with the duration of the time span.
- At the top level, the main branches take precedence.
- At the second and subsequent levels, the higher the priority of a branch X is, the higher the priority of the branch off of X becomes.

Figure 5 shows a time-span tree drawn with the duration of the time span. The branch priority is determined in order from the top in accordance with the first rule. Then, in accordance with the second rule, branch 1 has the highest priority in this time-span tree, and branch 2 has the second-highest priority. The second level in this tree is the double-note level. In accordance with the second rule, the branch off from 1 becomes 3, and that from 2 becomes 4. In the same manner, the priority is determined up to the 16th note level.

4.1 Automatic Melodic Morphing

For automatic partial reduction, we determine how much each melody is to be reduced and reduce the branches of the non-common part of the two melodies. If the

non-common part of the melody of *A* is reduced by 30%, the reduction ratio of melody *B* is determined to be 70%, so that the total is 100%. Then, in the non-common part of each melody, the branches are reduced in order from the branch with the lowest priority. The number of notes is finite, so reducing them in accordance with a set reduction ratio is often impossible. In such cases, the branches are reduced to be closest to the reduction ratio.

As described in Section 2.3, when a melody is synthesized by a *join* operation, the branches of the time-span tree may overlap at the same time. For example, if the branches and notes overlap at the same time due to the *join* operation of melody *A* and *B*, the note with the lower reduction ratio is left. If both reduction ratios are 50%, the note of *A* is left.

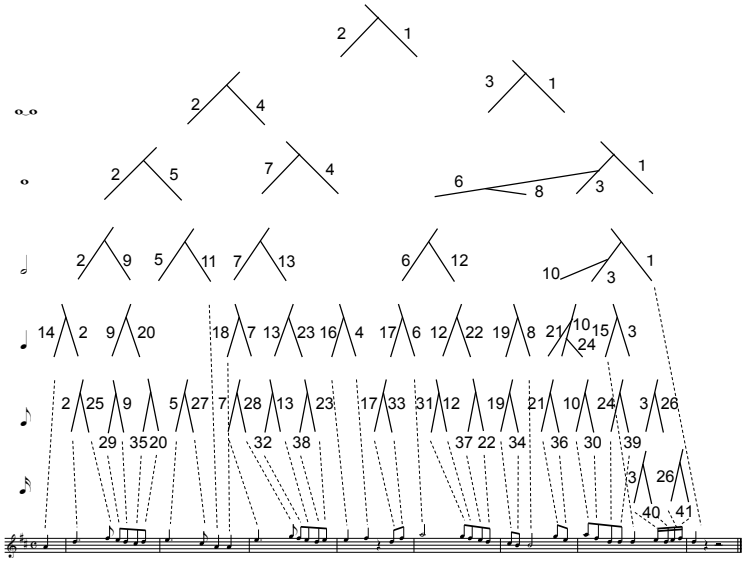

Fig. 5. Time-span tree leveled by duration of time span.

4.2 Automatic Time-span Tree Analysis by Deep Learning

The melody is leveled by the duration of the time span, then it is reduced one note at a time from the lowest level. In the following explanation, when there is a branch, the child branch is called a "sub-branch," and the parent branch is called the "main branch". Since the ground truth data of the time-span tree are mono-phonic, the target is monophony in this paper.

In the time-span tree leveled by the duration of time span, the level of the main branch is always higher than that of a sub-branch. Therefore, if the reduction is carried out in order from the lowest level, the reduction process will proceed without contradiction. It is also important that the reduction process be simple when learning stepwise reduction with a DNN.

Previous time-span tree analyzers (ATTA [9] and sigmaGTTMIII [10]) had low performance because they analyzed in a bottom-up manner using only local information. In contrast, we propose using the entire note sequence before and after stepwise reduction for learning the DNN.

When a recurrent neural network (RNN) [11] or long short-term memory (LSTM) [12] is used as the DNN, the DNN can learn using note sequence, but when a long note sequence is input, the DNN forgets the beginning of it, thus it cannot make use of all the information of the note sequence.

Seq2Seq [13] and Transformer [14] can learn and predict using the information of the entire note sequence. The difference between Seq2Seq and Transformer is the representation of position in the note sequence: Seq2seq uses relative positions by sequentially inputting sequence data into the RNN, while Transformer has an independent additional layer of position information and uses the absolute position.

Therefore, if the absolute position is important for stepwise time-span reduction, Transformer will have high performance, and if the relative position is important, Seq2Seq will have high performance. We evaluated which of the two has the higher performance, as described in Section 5.2.

5 Experiment and Results

As a verification of the usefulness of our proposed time-span tree leveled by the duration of time span, we conducted an experiment to confirm whether melodic morphing and time-span tree analysis can be carried out automatically.

5.1 Automating Melodic Morphing by Prioritization of Branches

After acquiring the time-span tree, there was no arbitrariness in the prioritization of the branches, partial reduction, and combination of melodies. Therefore, when the reduction ratio was determined, the morphed melody could be deterministically obtained. In Figure 6, the notes included in melody *A* are displayed with stems up, and those included in melody *B* are displayed with stems down.

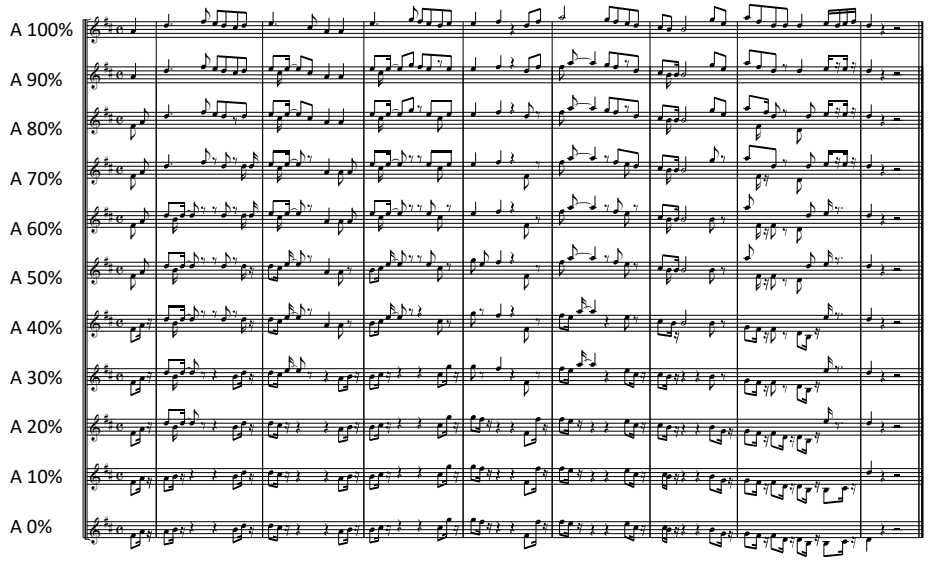

Fig. 6. Results of automatic morphing.

5.2 Comparison of Seq2Seq and Transformer in Stepwise Time-span Reduction

Learning data and evaluation data were created from MusicXML, which are score data, and time-spanXML, which is the ground truth of a time-span tree by the following procedure. The proposed method was first used to reduce (in a stepwise manner) each of the 300 melodies by using the time-span tree, and data before and after the reduction were generated.

Next, the notes in the melodies were made into a one-character string with the pitch and duration concatenated. The pitch was represented as 12 types: C, C#, D, D#, E, F, F#, G, G#, A, A#, and B (excluding octave information). A key that was a major or minor key was then changed to C major or A minor. The duration was represented by multiplying the duration elements of MusicXML by 4. By multi-plying by 4, the duration of most notes became an integer, but since there were melodies containing only a few triplets, quintuplets, sextuplets, and septuplets, the duration was rounded up to an integer. Then, a space was inserted between the strings to represent notes. Finally, in the note sequence after the reduction, "r" was inserted at the position of the reduced note so that we would know which note had been reduced.

The Seq2Seq and Transformer models were both trained with 7362 stepwise time-span-reduction training data sets generated from 270 songs from a GTTM database consisting of 300 pieces, and 849 evaluating data sets were generated from the remaining 30 pieces. Table 1 shows the accuracy of matching the evaluation data and prediction data after 20,000 epochs of training. We can see that Transformer outperformed Seq2Seq in stepwise time-span reduction. Learning was carried out using Nvidia Quadro RTX5000 for laptops [15], and the learning time of Seq2Seq was six days, which is much longer than the seven hours taken by Transformer.

Table 1. Comparison of Seq2Seq and Transformer models.

	Seq2Seq	Transformer
Accuracy	0.90	0.99

6 Conclusion

We proposed the introduction of time-span tree leveled by the duration of time span to problems that are difficult to automate due to the lack of prioritization of time-span tree branches. Experimental results confirmed that melodic morphing and time span analysis based on deep learning can be automated.

We plan to develop various applications and content by using a time-span tree. Our morphing method has appeared in the smart-phone applications of Melody Slot Machine [16], which has a huge number of downloads. By using an automated morphing system, it is possible to build a system that facilitates the addition of content on Melody Slot Machine.

References

1. Lerdahl, F. and Jackendoff, R.: *A Generative Theory of Tonal Music*. MIT Press, Cambridge, 1985.
2. Hamanaka, M., Hirata, K., and Tojo, S.: Melody Morphing Method Based on GTTM. In: *Proceedings of International Computer Music Conference* (ICMC2008), pp. 155–158, 2008.
3. Hamanaka, M., Hirata, K., and Tojo, S.: Melody Extrapolation in GTTM Approach. In: *Proceedings of International Computer Music Conference* (ICMC2009), pp. 89–92, 2009.
4. Hamanaka, M., Hirata, K., and Tojo, S.: deepGTTM-I: Local Boundaries Analyzer Based on Deep Learning Technique. In: *Proceedings of International Symposium on Computer Music Multidisciplinary Research* (CMMR2016), pp. 8–20, 2016.
5. Hamanaka, M., Hirata, K., and Tojo, S.: deepGTTM-II: Automatic Generation of Metrical Structure based on Deep Learning Technique. In: *Proceedings of the 13th Sound and Music Conference* (SMC2016), pp. 221–249, 2016.
6. Hamanaka, M., Hirata, K., and Tojo, S.: "Musical Structural Analysis Database Based on GTTM. In: *Proceedings of the 15th International Conference on Music Information Retrieval Conference* (ISMIR2014), pp. 107–112, 2014.
7. Amari, S., Ozeki, T., Karakida, R., Yoshida, Y., and Okada, M. Dynamics of Learning in MLP: Natural Gradient and Singularity Revisited. *Neural Comput.*, vol. 30, issue 1, pp. 1–33, 2018.
8. Marsden, A., Tojo, S., and Hirata, K. No longer 'somewhat arbitrary': calculating salience in GTTM-style reduction. In: *Proceedings of the 5th International Conference on Digital Libraries for Musicology* (DLfM'18), pp. 26–33, 2018.
9. Hamanaka, M., Hirata, K., and Tojo, S.: ATTA: Automatic Time-Span Tree Analyzer Based on Extended GTTM. In: *Proceedings of the 6th International Conference on Music Information Retrieval Conference* (ISMIR 2005), pp. 358–365, 2005.
10. Hamanaka, M., Hirata, K., and Tojo, S.: σGTTM III: Learning-Based Time-Span Tree Generator Based on PCFG. In: *Proceedings of International Symposium on Computer Music Multidisciplinary Research* (CMMR2015), pp. 387–404, 2015.
11. Pineda, J. F. Generalization of Back-propagation to Recurrent Neural Networks. *Physical Review Letters*, 19(59): 2229–2232, 1987.
12. Hochreiter, S. and Schmidhuber, J. Long Short-term Memory. Neural *Comp.* 9(8): 1735–1780, 1997.
13. Sutskever, I., Vinyals, O. and Le, V. Q. Sequence to Sequence Learning with Neural Networks. In: *Advances in Neural Information Processing Systems 27*, pp. 3104–3112, 2014.
14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, N. A., Kaiser, L. and Polosukhin, I. Attention Is All You Need. In: *Proceedings of the 31st Conference on Neural Information Processing Systems* (NIPS2017), vol. 30, pp. 6000–6010, 2017.
15. Nvidia: Quadro for laptops, https://www.nvidia.com/en-us/design-visualization/quadro-in-laptops/, 2021.
16. Hamanaka, M. Melody Slot Machine. https://gttm.jp/hamanaka/en/melodyslotmachine/, 2021