Chapter 9
Implementing Methods for Analysing Music Based on Lerdahl and Jackendoff's *Generative Theory of Tonal Music*

Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo

Abstract We describe and discuss our computer implementations of Lerdahl and Jackendoff's (1983) *Generative Theory of Tonal Music* (GTTM). We consider this theory to be one of the most relevant music theories with regard to formalization because it captures aspects of musical phenomena based on the Gestalts perceived in music and presents these aspects with relatively rigid rules. However, the theory has several problems in terms of computer implementation. To overcome these problems, we have proposed four different kinds of analyser: an automatic time-span tree analyser (ATTA); a fully automatic time-span tree analyser (FATTA); the σGTTM analyser, which detects local grouping boundaries by combining GTTM with statistical learning using a decision tree; and the σGTTM-II analyser, with which we introduce full parameterization and statistical learning.

9.1 Introduction

Over the past ten years, we have developed several music analysers, based on Lerdahl and Jackendoff's (1983) *Generative Theory of Tonal Music* (GTTM), which provide us with abstracted structures from scores (Hamanaka et al., 2006, 2007; Kanamori and Hamanaka, 2014; Miura et al., 2009). When implementing music-theory-based methods for analysing music on a computer, we have to consider several problems,

Masatoshi Hamanaka
Clinical Research Center, Kyoto University, Kyoto, Japan
e-mail: masatosh@kuhp.kyoto-u.ac.jp

Keiji Hirata
Future University Hakodate, Hakodate, Hokkaido, Japan
e-mail: hirata@fun.ac.jp

Satoshi Tojo
Japan Advanced Institute of Science and Technology (JAIST), Nomi, Ishikawa, Japan
e-mail: tojo@jaist.ac.jp

including ambiguity, dependence on context and the trade-off between automation and variation in analysis results. Each of these problems will now be briefly introduced.

Ambiguity in music analysis A piece of music will typically have more than one interpretation, and dealing with such ambiguity is a major obstacle when implementing a music theory on a computer. We have to consider two types of ambiguity in music analysis, one involving human understanding of music and the other concerning the representation of music theory. The former stems from subjective interpretation and the latter from the incompleteness of formal theory. GTTM is no exception. Therefore, due to the presence of ambiguity, we assume that there is always more than one correct result.

Context dependence in music analysis Even if the same musicologist analyses the same note sequence, the analysis results will not always be the same. This is because the results depend on so many different factors, such as rhythm, chord progression, melody of the other parts, and the historical period in which the piece of music was composed. Moreover, a musicologist might take into account other unknown factors.

Trade-off relationship in music analysis There is a trade-off relationship between the automation of the analysis process and variation in the analysis results. Since an analysis program outputs only one interpretation for a given score, different ways of interpreting that score are ignored.

These problems are not specific to music analysis but arise whenever we try to build a computer model of an ability that, in a human, would require intelligence. This includes most abilities that involve recognition or understanding. Therefore, implementing a music theory on a computer can be considered an artificial intelligence problem.

The rest of this chapter is organized as follows. In Sects. 9.2 and 9.3, we give a brief overview of GTTM and present related work. In Sect. 9.4, we discuss the difficulty of implementing the theory on a computer. In Sects. 9.5 to 9.8, we go into depth about the four proposed analysers: the automatic time-span tree analyser (ATTA), the fully automatic time-span tree analyser (FATTA), the σGTTM analyser, and the σGTTM-II analyser. In Sect. 9.9, we propose our interactive analyser for the theory, and in Sect. 9.10, we describe a GTTM database we have constructed. Some experimental results are presented in Sect. 9.11.

9.2 Lerdahl and Jackendoff's (1983) *Generative Theory of Tonal Music*

Lerdahl and Jackendoff's (1983) *Generative Theory of Tonal Music* (henceforth, GTTM) generates four different types of structural description for a piece of music, each one intended to represent a separate aspect of the way that a listener understands

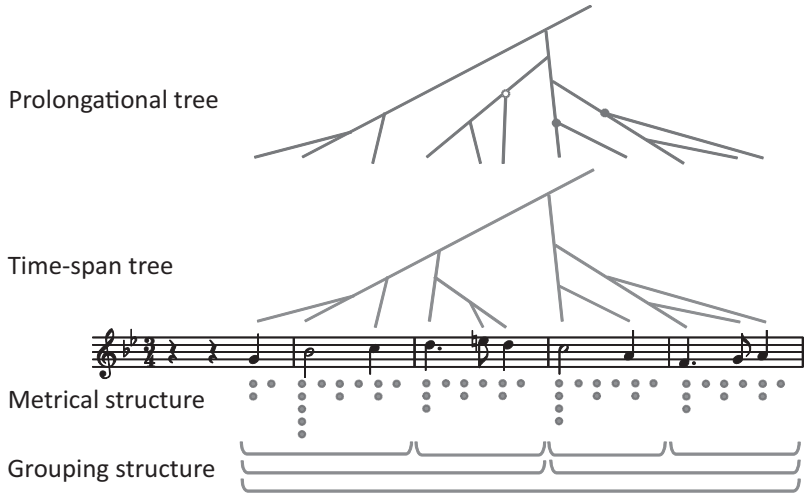

Fig. 9.1 Grouping structure, metrical structure, time-span tree, and prolongational tree

the piece. The respective outputs are a grouping structure, a metrical structure, a time-span tree, and a prolongational tree (see Fig. 9.1).

The grouping structure is intended to formalize the intuition that tonal music is organized into groups composed of subgroups (see bottom row in Fig. 9.1). The metrical structure describes the rhythmic hierarchy of a piece by identifying the positions of beats at different metrical levels. Metrical levels are represented as rows of dots below the staff. For example, in Fig. 9.1, the strongest beats occur at the beginning of every second bar, the next strongest at the beginning of each bar, the next strongest at the quarter note level and so on. The time-span tree is a binary tree having a hierarchical structure that describes the relative structural importance of notes that differentiate the essential parts of the melody from the ornamentation. The prolongational tree is a binary tree that expresses the structure of tension and relaxation in a piece of music.

A GTTM analysis has four processes: grouping structure analysis, metrical structure analysis, time-span reduction analysis, and prolongational reduction analysis. Each process has two types of rule: *well-formedness rules* (WFRs) and *preference rules* (PRs). Well-formedness rules are necessary conditions on assigning the structure and restrictions on these structures. For example, the GWFRs (grouping WFRs) are defined as follows (Lerdahl and Jackendoff, 1983, p. 345):

GWFR1: Any contiguous sequence of pitch events, drum beats, or the like can constitute a group, and only contiguous sequences can constitute a group.

GWFR2: A piece constitutes a group.

GWFR3: A group may contain smaller groups.

GWFR4: If group G_1 contains part of group G_2, it must contain all of G_2.

GWFR5: If group G_1 contains a smaller group G_2, then G_1 must be exhaustively partitioned into smaller groups.

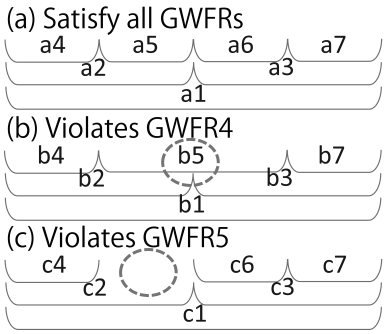

Fig. 9.2 Examples of GWFRs being satisfied or violated

The grouping structure in Fig. 9.2(a) satisfies all the GWFRs. In contrast, the grouping structure in Fig. 9.2(b) violates GWFR4 because a segment boundary at the second level of the grouping structure occurs in the middle of a group at the lowest level. The grouping structure in Fig. 9.2(c) violates GWFR5 because group c2 is not exhaustively partitioned into smaller groups.

When more than one structure can satisfy the WFRs, the PRs indicate the superiority of one structure over another. In the PRs, some rules are for the local level and others for the hierarchical level. For example, GPR2 is a local rule that is applied to four consecutive notes n_1, n_2, n_3, n_4 as follows (Lerdahl and Jackendoff, 1983, p. 345):

> **GPR 2 (Proximity)** Consider a sequence of four notes n_1, n_2, n_3, n_4. All else being equal, the transition n_2–n_3 may be heard as a group boundary if
>
> a. (Slur/Rest) the interval of time from the end of n_2 to the beginning of n_3 is greater than that from the end of n_1 to the beginning of n_2 and that from the end of n_3 to the beginning of n_4, or if
> b. (Attack-point) the interval of time between the attack points of n_2 and n_3 is greater than that between the attack points of n_1 and n_2 and that between the attack points of n_3 and n_4.

GPR2b is applied after a note that has a long duration (Fig. 9.3(a)), while GPR2a is applied after a note that has a long gap, even if the inter-onset interval (IOI) of each note is equal (Fig. 9.3(b)).

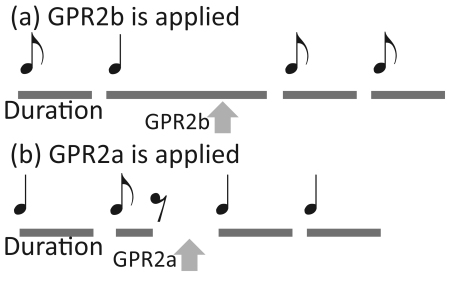

Fig. 9.3 Application of GPR2b and GPR2a

9.3 Related Work

Here, we briefly take a look at the history of cognitive music theory. The implication–realization model (IRM) proposed by Eugene Narmour abstracts and represents music according to changes in a melody, expressed symbolically (Narmour, 1990, 1992). The IRM has recently been implemented on computers, which can acquire the chain structures of IRM from a score (Yazawa et al., 2014). *Schenkerian* analysis acquires a deeper structure, called the *Urlinie* and *Ursatz*, from the musical surface (Schenker, 1935). Short segments of music can be analysed through Schenkerian analysis on a computer (Marsden, 2011). Other examples of music theories that lend themselves to computer implementation include that of Lerdahl (2001). The preference rule approach, pioneered by Lerdahl and Jackendoff (1983), was also adopted by Temperley (2001) and Daniel Sleator in their *Melisma Music Analyzer*.[1]

The main advantage of analysis by GTTM is that it can acquire tree structures (specifically, time-span and prolongational trees). These trees provide a summarization of a piece of music, which can then be used as the representation of an abstraction, resulting in a music retrieval system (Hirata and Matsuda, 2003). It can also be used for performance rendering to generate expressive musical performances (Hirata and Hiraga, 2003) and to reproduce music (Hirata and Matsuda, 2004). Moreover, the time-span tree can be used for melody prediction (Hamanaka et al., 2008a) and melody morphing (Hamanaka et al., 2008b). Figure 9.4 shows an iOS application that implements this melody morphing method and changes the degree of morphing of each half bar by using the values from the device's accelerometer (Hamanaka et al., 2011). When the user stops moving the device, the unit plays the backing melody of "The Other Day, I Met a Bear (The Bear Song)". When the user shakes it vigorously, it plays heavy soloing. When the user shakes it slowly, it plays a morphed melody ranging in morphing degree from copying the backing to heavy soloing.

The grouping structure analysis generated by GTTM is a type of melody segmentation. Previous segmentation methods have been unable to construct hierarchical grouping structures because they have been focused on detecting the local bound-

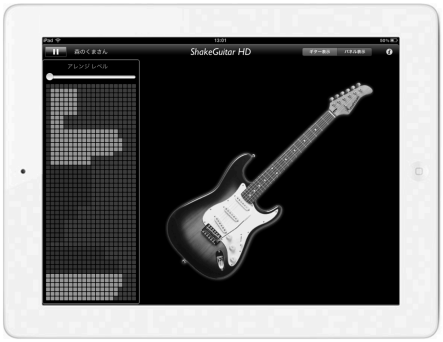

Fig. 9.4 ShakeGuitar

[1] Available at http://www.link.cs.cmu.edu/melisma/

aries of a melody (Cambouropoulos, 2006; Rodriguez-Lopez et al., 2014; Stammen and Pennycook, 1994; Temperley, 2001). A metrical structure analysis generated by GTTM, in contrast, is a kind of beat tracking. Current methods based on beat tracking (Davies and Bock, 2014; Dixon, 2001; Goto, 2001; Rosenthal, 1992) are only able to acquire the hierarchical metrical structure up to the bar level but not above that (e.g., at the two- and four-bar level).

9.4 Problems with Implementing GTTM

There are a number of features of GTTM that make it difficult to implement as a computer program. The main problems with implementation are discussed in this section.

9.4.1 Ambiguous Rule Definition

Some rules in GTTM are expressed ambiguously. For example, GPR4 is defined as follows (Lerdahl and Jackendoff, 1983, p. 346):

> **GPR4 (Intensification)** Where the effects picked out by GPRs 2 and 3 are relatively more pronounced, a larger-level group boundary may be placed.

The words "relatively" and "may be" in this sentence are ambiguous. The sentence also contains the phrase "more pronounced", but the comparison is unclear. Another example is that GTTM has rules for selecting proper structures when discovering similar melodies (called parallelism), but does not define similarity. To implement such ambiguous rules on a computer, we have to formalize the criteria for deciding whether each rule is applicable.

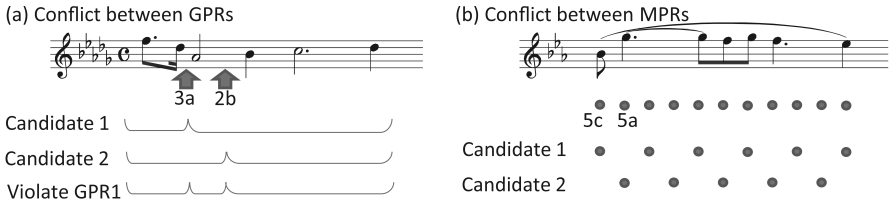

Fig. 9.5 Example of conflict between PRs

9.4.2 Conflict Among Preference Rules

Because there is no strict order for applying the GTTM rules, conflict between rules often occurs when applying them, resulting in ambiguities in analysis. Figure 9.5(a) shows a simple example of a conflict between GPR2b (Attack-Point) and GPR3a (Register). GPR2b states that a relatively greater interval of time between attack points initiates a grouping boundary, while GPR3a states that relatively greater pitch differences between smaller neighbouring intervals initiates a grouping boundary. Because GPR1 (alternative form) strongly prefers that note 3 in Fig. 9.5(a) should not form a group on its own, placing boundaries between notes 2 and 3 *and* between notes 3 and 4 is discouraged.

Figure 9.5(b) shows an example of conflict between MPRs 5c and 5a. MPR5c states that a relatively long slur results in a strong beat, and MPR5a states that a relatively long pitch event results in a strong beat. Because metrical well-formedness rule 3 (MWFR3) states that strong beats are spaced either two or three beats apart, a strong beat cannot be perceived at the onset of both the first and second notes.

To solve these problems, we have introduced the notion of *parameterization* in Sect. 9.5. Each rule in the theory should be given a weight, allowing the strength of its effect to be compared with that of other rules; this weight can be regarded as a parameter of the analysis process. In addition, we have externalized the hidden alternatives in the theory. This externalization in mechanizing GTTM includes introducing an algorithm for generating a hierarchical structure of the time-span tree. We call such weighting and externalization *full parameterization*. Employing these parameters together with statistical learning, we obtain a methodology to control the strength of each rule (described in Sect. 9.7).

9.4.3 Lack of Working Algorithm

Lerdahl and Jackendoff (1983) do not specify an algorithm for constructing a hierarchical structure because the preference rules only indicate preferred structures. For example, no algorithm is provided for acquiring a hierarchical grouping structure after acquiring local grouping boundaries in the grouping structure analysis. Also, there are many time-span reduction preference rules (TSRPRs) for selecting the head of a time-span, and there are various examples of analysis. However, no algorithm is presented for acquiring the hierarchical time-span tree. It is not realistic to first generate every structure that satisfies the WFRs and then select the optimal structure. For example, even for a musical fragment containing just 10 notes, there are 185,794,560 ($= 9^2 * 9!$) possible time-span trees.

To solve this problem, in Sect. 9.5 we present an algorithm for acquiring the hierarchical structure, taking into consideration some of the examples in GTTM (Hamanaka et al., 2006).

9.4.4 Less Precise Explanation of Feedback Link

GTTM has various feedback links from higher-level structures to lower-level ones, e.g., GPR7 (time-span and prolongational stability) requires a grouping structure that results in a more stable time-span and/or prolongational reduction. However, no detailed description and only a few examples are given.

Other feedback links in the GTTM rules are not explicit. For example, analysing the results of a time-span tree strongly affects the interpretation of chord progression, and various rules are related to chord progression, e.g., MPR7 (Cadence) requires a metrical structure in which cadences are metrically stable.

To solve this problem, in Sect. 9.9, we propose a tool that allows a user to modify the automatic analysis process and manually edit the structures generated. A user can thus acquire a target analysis that reflects his or her interpretation of a piece of music by iterating the automatic and manual processes interactively and easily.

9.5 ATTA: Automatic Time-Span Tree Analyser

We extended the original theory of GTTM with full externalization and parameterization and proposed a machine-executable extension of GTTM called exGTTM (Hamanaka et al., 2006). The externalization includes introducing an algorithm to generate the hierarchical structure of a time-span tree using a combination of top-down and bottom-up processes. The parameterization includes introducing a parameter for controlling the priorities of rules, in order to avoid conflict among rules, as well as parameters for controlling the shape of the hierarchical time-span tree. We developed an automatic time-span tree analyser (ATTA) to implement exGTTM. The user can manually configure parameters and thus alter the analysis results generated by the program. An example of constructing a time-span tree is given in the following subsections.

9.5.1 Time-Span Segmentation

In the procedure of time-span reduction, we divide the entire piece into hierarchical time-spans. The division procedure, shown in Fig. 9.6, is as follows.

1. Regard all of the resultant groups in a grouping analysis as time-spans.
2. Divide a time-span into two at the strongest beat when a time-span in the lowest level includes more than one note.
3. Repeat 2 recursively.

Steps 1 and 2 correspond, respectively, to Lerdahl and Jackendoff's (1983, pp. 146–147) time-span reduction Segmentation Rules 1 and 2.

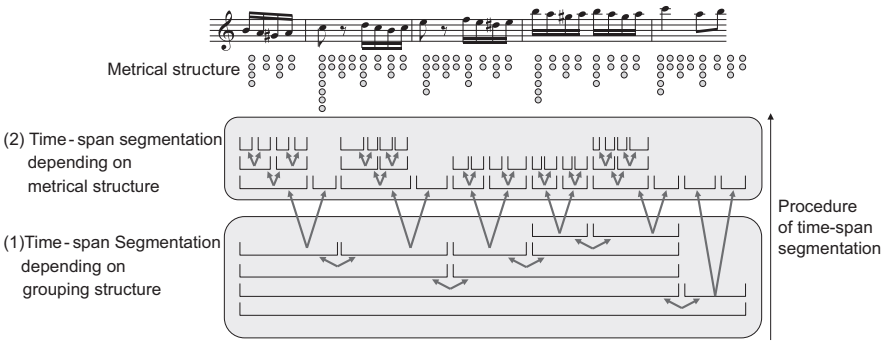

Fig. 9.6 Time-span segmentation

9.5.2 Implementation of Time-Span Reduction Preference Rules

A piece of music is formed into a binary tree where, at each node, the more important branch extends upward as a head node. The selection of a head at each node is hierarchically computed from the lower level. Therefore, heads are selected from leaves to root branches. At the lowest level, every note is selected as a head for its time-span and the next level head is selected repetitively. In this section, we explain our implementations of time-span reduction preference rules (TSRPRs) 1, 3, 4, 8 and 9.

9.5.2.1 Calculation of Basic Parameters

For each level in the hierarchy, we provide the following basic parameters and rule-application principles.

We calculate four basic parameters:

ϕ_i Offset-to-onset interval (OOI) of ith gap between heads
ψ_i Inter-onset interval (IOI) of ith gap between heads
ξ_i Difference in pitch in semitones of ith gap between heads
μ_i Number of metrical levels in which ith head is a beat

The term i indicates the order of heads at the current level of time-span. For example, at the lowest level, the head order is the same as the note order. In the first three parameters, i represents the ith gap between heads, that is, the gap between the ith and $(i+1)$th head, while μ_i is the number of metrical dots for the ith head (i.e., the number of metrical levels in which the ith head is a beat). The probability that the ith head becomes a next-level head by the kth rule is denoted by $D_{\mathrm{TSRPR}k}(i)$ where $0 \leq D_{\mathrm{TSRPR}k}(i) \leq 1$ and $k \in \{1,3,4,8,9\}$. The basic parameters and $D_{\mathrm{TSRPR}k}(i)$ are renewed at each level in the time-span tree, because the number of heads changes as a result of selecting heads at each hierarchical level.

9.5.2.2 Implementation of TSRPR1 (Metrical Position)

TSRPR1 prefers heads of time-spans to occur in relatively strong metrical positions. We normalize the strength between 0 and 1 and define the likelihood of an event being a head by the number of metrical dots divided by the maximum number of metrical dots, thus

$$D_{\mathrm{TSRPR1}}(i) = \mu_i / \max_j \mu_j \ . \tag{9.1}$$

9.5.2.3 Implementation of TSRPR3 (Registral Extremes)

TSRPR3 is concerned with the pitch of a head. TSRPR3a weakly prefers an event to be a head if its melodic pitch is higher; while TSRPR3b weakly prefers an event to be a head if its bass pitch is lower. Thus, $D_{\mathrm{TSRPR3a}}(i)$ returns a higher value if ξ_i is higher:[2]

$$D_{\mathrm{TSRPR3a}}(i) = \xi_i / \max_j \xi_j \ . \tag{9.2}$$

Conversely, $D_{\mathrm{TSRPR3b}}(i)$ returns a higher value if ξ_i is lower:

$$D_{\mathrm{TSRPR3b}}(i) = 1 - \xi_i / \max_j \xi_j \ . \tag{9.3}$$

9.5.2.4 Implementation of TSRPR4 (Parallelism)

TSRPR4 involves parallelism and prefers heads to be in parallel positions in time-spans that are construed to be parallel. The parallelism between heads in the current hierarchical level's time-spans is evaluated using the same method as is used in the grouping and metrical analysis.

$D_{\mathrm{TSRPR4}}(i)$ is calculated as follows. First, we calculate the similarity between the interval from head i with length r and the one from j with the same length. Next, for each i, we calculate the similarity for all js with the same length r. As Lerdahl and Jackendoff (1983) do not define an effective melodic similarity measure, we define our own. This similarity measure does not affect any other parts of the system, so we can substitute other methods such as that proposed by Hewlett and Selfridge-Field (1998).

Let us first consider the example in Fig. 9.7. In the figure, three notes out of four coincide with respect to onset time. Two notes out of the three also coincide with respect to pitch. In our implementation, we regard a greater number of notes having the same onset time as indicating greater similarity of melodies. Furthermore, the greater the number of instances where notes with the same onset time have the same pitch, the more similar the melodies are considered to be. In the example in

[2] There are several ways in which registral extremity could be explicated. We define $D_{\mathrm{TSRPR}k}(i)$ as simply as possible in order to allow the user who is manipulating the parameters of ATTA to understand it easily.

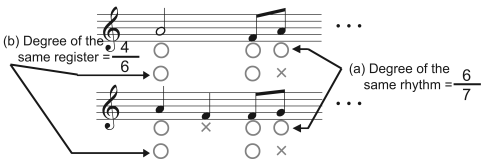

Fig. 9.7 Similarity of parallel phrases

Fig. 9.7, we use *notes* and their *durations* for explanation. However, when calculating similarity of time-spans, we use *heads* and the *lengths of their time-spans*.

We formalize the above discussion as follows. We assume that the beginning and ending heads possess beats and that the length of an interval r is a multiple of a beat. Also, we assume that parallelism cannot occur between time-spans separated by distances less than, say, a quarter note of a beat. Given a beat number m (≥ 1), we write the interval of r beats from m as $[m, m+r)$, which does not include the $(m+r)$th beat. We define the basic parameters as follows:

$N(m,r)$ the number of heads in $[m, m+r)$.
$O(m,n,r)$ the number of heads with the same time-span onset in $[m, m+r)$ and $[n, n+r)$.
$P(m,n,r)$ the number of heads with the same pitch, as well as the same time-span onset.

We define the similarity between intervals $[m, m+r)$ and $[n, n+r)$ with these parameters:

$$G(m,n,r) = \left\{ \frac{O(m,n,r)}{N(m,r)+N(n,r)} \times (1-W_m) + \frac{P(m,n,r)}{O(m,n,r)} \times W_m \right\} \times r^{W_l}, \quad (9.4)$$

where

W_m $(0 \leq W_m \leq 1)$ For each head, gives more weight to similarity of time-span onset than similarity of pitch.
W_l $(0 \leq W_l \leq 1)$ Gives more weight to longer intervals, r, than shorter ones, when parallel intervals overlap each other.

In the above expressions, $1 \leq m,n \leq L-r+1$ and $1 \leq r \leq L$, where L is the total number of beats. Beyond this domain, we regard $G(m,n,r) = 0$. Note that r^{W_l} becomes 1 when $W_l = 0$, and as r increases, r^{W_l} also increases provided $W_l > 0$. Thus, as W_l approaches 1, the similarity of longer intervals becomes more significant.

The similarity of head i in $[m, m+r)$ and j in $[m, m+s)$ is expressed by

$$A(i,j) = G(timespan_{\mathrm{s}}(i), timespan_{\mathrm{s}}(j), timespansize(i)), \quad (9.5)$$

where $timespan_{\mathrm{s}}(i)$ is the first beat of the time-span including i, and $timespansize(i)$ is the length (the number of beats) of the time-span including i. We define $D_{\mathrm{TSRPR4}}(i,j)$, by normalizing $A(i,j)$, as

$$D_{\mathrm{TSRPR4}}(i,j) = \begin{cases} A(i,j)/A_{\max}, & \text{if } timespanpos(i) = timespanpos(j); \\ 0, & \text{otherwise.} \end{cases} \quad (9.6)$$

where $A_{\max} = \max(A(i,1), A(i,2), \ldots, A(i,L))$ and *timespanpos*(i) is the interval from the beginning of the time-span to i. Note that i indicates the order of heads at the current time-span level, after which $D_{\text{TSRPR4}}(i,j)$ renews at each level of time-span.

9.5.2.5 Implementation of TSRPR8 (Structural Beginning)

TSRPR8 prefers heads to be nearer the beginnings of their time spans. $D_{\text{TSRPR8}}(i)$ returns 1 if the head is at the beginning position; otherwise, 0:

$$D_{\text{TSRPR8}}(i) = \begin{cases} 1, & \text{if } i = i_{\text{start}}, \\ 0, & \text{otherwise}, \end{cases} \tag{9.7}$$

where i_{start} is the head at the beginning of the time-span.

9.5.2.6 Implementation of TSRPR9 (Structural Ending)

TSRPR9 prefers heads to be nearer the tails of their time-spans. $D_{\text{TSRPR9}}(i)$ returns 1 if the head is at the tail position, otherwise, 0:

$$D_{\text{TSRPR9}}(i) = \begin{cases} 1, & \text{if } i = i_{end}, \\ 0, & \text{otherwise}, \end{cases} \tag{9.8}$$

where i_{end} is the head at the tail of the time-span.

9.5.3 Generation of Time-Span Tree

We calculate the plausibility of head $D^{\text{timespan}}(i)$ using $D_{\text{TSRPR}k}(i)$ where $k \in \{1, 3\text{a}, 3\text{b}, 4, 8, 9\}$, as follows:

$$D^{\text{timespan}}(i) = B^{\text{timespan}}(i) + \sum_k \begin{cases} B^{\text{timespan}}(k) \times S_{\text{TSRPR4}}, & \text{if } D_{\text{TSRPR4}}(i,k) = 1, \\ 0, & \text{if } D_{\text{TSRPR4}}(i,k) = 0, \end{cases} \tag{9.9}$$

where

$$B^{\text{timespan}}(i) = \sum_k D_{\text{TSRPR}k}(i) \times S_{\text{TSRPR}k} \tag{9.10}$$

where $k \in \{1, 3\text{a}, 3\text{b}, 8, 9\}$. $S_{\text{TSRPR}k}$ indicates the relative weighting associated with each rule. The larger this value is, the more strongly the rule acts. $B^{\text{timespan}}(i)$ represents the weighted summation of $S_{\text{TSRPR}k}$ and $D_{\text{TSRPR}k}(i)$ where k is the rule number and $D^{\text{timespan}}(i)$ represents the sum of $B^{\text{timespan}}(i)$ and the summation of $B^{\text{timespan}}(k)$, where the ith head and kth head are parallel and consequently $D_{\text{TSRPR4}}(i,k) = 1$. A

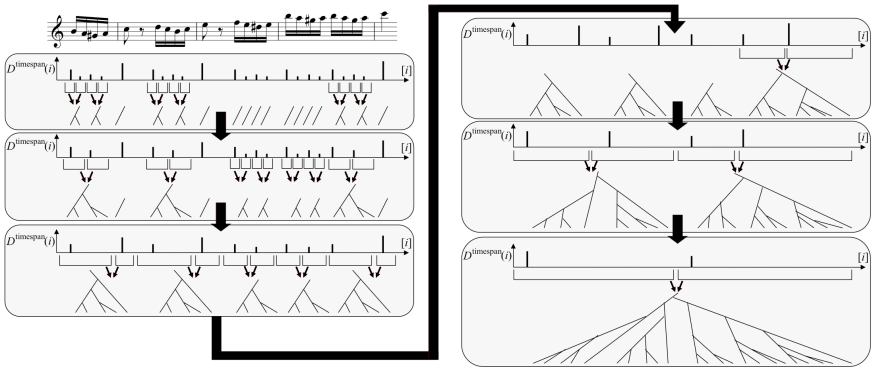

Fig. 9.8 Selecting next-level heads

hierarchical time-span tree is constructed by iterating the calculation of the plausibility of head $D^{\text{timespan}}(i)$ for the current heads and choosing the next-level heads (Fig. 9.8). If there are two candidate heads in the current hierarchical level, we choose the next-level head $\hat{h}$, as expressed by

$$\hat{h} = \begin{cases} i, & \text{if } D^{\text{timespan}}(i) \leq D^{\text{timespan}}(j); \\ j, & \text{otherwise.} \end{cases} \tag{9.11}$$

The order of choosing the next-level head is the reverse of that of constructing time-spans in the time-span segmentation, as described in Sect. 9.5.2. $D_{\text{TSRPR}k}(i)$ and $D^{\text{timespan}}(i)$ are renewed at each level of the hierarchy, since i indicates the order of heads in the current level and changes at each level of time-spans.

9.6 FATTA: Fully Automatic Time-Span Tree Analyser

Although the ATTA has adjustable parameters for controlling the weight or priority of each rule, these parameters have to be set manually (Hamanaka et al., 2006). This takes a long time, because finding the optimal values of the settings themselves takes a long time. Therefore, we also developed a fully automatic time-span tree analyser (FATTA), which can automatically estimate the optimal parameters by introducing a feedback loop from higher-level structures to lower-level structures on the basis of the stability defined in GPR7 and TSRPR5 (Hamanaka et al., 2007):

> **GPR7 (Time-Span and Prolongational Stability)** Prefer a grouping structure that results in more stable time-span and/or prolongational reductions.
>
> (Lerdahl and Jackendoff, 1983, p. 52)

and

> **TSRPR5 (Metrical Stability):** In choosing the head of a time-span T, prefer a choice that results in more stable choice of metrical structure
>
> (Lerdahl and Jackendoff, 1983, p. 165)

These rules require information from later processes, such as time-span/prolongational reductions, to be sent back to the earlier processes. To automatically estimate the optimal parameters, we have to evaluate the level of time-span tree stability derived using the ATTA. We use GPR7 and TSRPR5 for calculating the level of stability. Figure 9.9 shows the process flow of the FATTA, which consists of the ATTA and a feedback loop by the GPR7 and TSRPR5.

9.6.1 Implementation of GPR7 with Tonal Pitch Space

GPR7 is the rule applied to the feedback loop between the time-span/prolongational reduction and grouping structure analysis. This rule leads to a preference for a grouping structure that results in more stable time-span and/or prolongational reductions. The term D_{GPR7} indicates the degree of being a head by GPR7, which varies continuously between 0 and 1. We define D_{GPR7} as

$$D_{\mathrm{GPR7}} = \sum_i distance(p(i), s(i)) \times size(i)^2 \Big/ \sum_i size(i)^2 \,, \tag{9.12}$$

where i indicates the head of the time-span, which has primary and secondary branches denoted by $p(i)$ and $s(i)$, respectively. The $distance(x, y)$ indicates the distance between notes x and y in the tonality of the piece, which is defined according to Lerdahl's (2001) theory of tonal pitch space. We normalized the distance from 0 to 1. The $size(i)$ indicates the length of the time-span that has head i. When calculating D_{GPR7}, we use the square of $size(i)$ for weightings for empirical reasons.

9.6.2 Implementation of TSRPR5

TSRPR5 is the rule applied to the feedback loop between the time-span reduction and the metrical structure analyser. This rule leads to a preference that results in a more stable choice of metrical structure in choosing the head of a time-span. The term D_{TSRPR5} indicates the strength of the rule in a given instance, which varies continuously between 0 and 1. We define D_{TSRPR5} as

$$D_{\mathrm{TSRPR5}} = \frac{1}{\sum_i size(i)^2} \begin{cases} size(i)^2, & \text{if } dot(p(i)) \geq dot(s(i)) \,, \\ 0, & \text{otherwise} \,, \end{cases} \tag{9.13}$$

where $dot(x)$ indicates the number of metrical dots for note x.

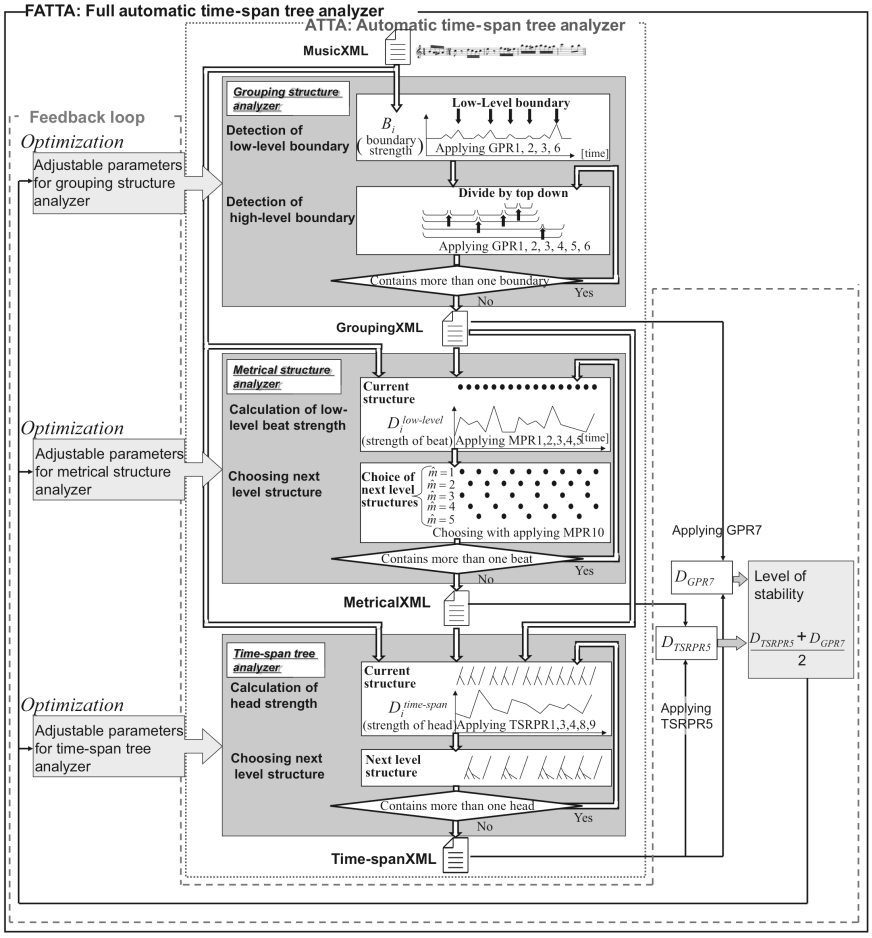

Fig. 9.9 Processing flow of fully automatic time-span tree analyser

9.6.3 Optimization of Adjustable Parameters

The optimal parameter sets of the ATTA can be obtained by maximizing the average of $D_{\mathrm{GPR7}}(0 \leq D_{\mathrm{GPR7}} \leq 1)$ and $D_{\mathrm{TSRPR5}}(0 \leq D_{\mathrm{TSRPR5}} \leq 1)$. Because there are 46 adjustable parameters, e.g., S_{rules} or W_m, it takes a long time to calculate all the combinations of parameter sets. To decrease the calculation time, we constructed the following algorithm:

1. Maximize the average of D_{GPR7} and D_{TSRPR5} by changing a parameter from minimum to maximum.
2. Repeat 1 for all parameters.

3. Iterate 1 and 2, as long as the average of D_{GPR7} and D_{TSRPR5} is increased from the previous iteration.

Finally, the FATTA can output only one analysis result without manual configuration. The computation time depends on the piece. In our experiment, described in Sect. 9.11, the shortest piece took about 5 minutes to analyse and the longest took about one week.

9.7 σGTTM Analyser

Our σGTTM system can detect the local grouping boundaries in a GTTM analysis by combining GTTM and statistical learning with a decision tree (Miura et al., 2009). A decision tree is a statistical learning method in which decisions are made by considering the value of each ramification. When learning the decision tree, bigger ramifications have a greater influence on the decision-making process, which causes it to be closer to the root position.

9.7.1 Abstraction of Training Data

As training data, we selected 100 MusicXML files that were then manually analysed by a musicologist and checked by GTTM experts. The value we want to know is the existence of a local grouping boundary (denoted as b), so that the value can be represented as 1 or 0 (boundary exists or not). A local GPR such as GPR 2 or 3 should also be abstracted because whether there is a boundary or not is determined by the local GPR. Considering that there is a local GPR for avoiding groups consisting of single notes, not only interval n (between note n and note $n+1$) but also the neighbouring intervals (interval $n-1$ and interval $n+1$) should be checked. Therefore, the data were abstracted in the form, B^n_{GPR}, where the superscript n refers to the nth interval and the subscript *GPR* means the type of local GPR, of which there are six (2a, 2b, 3a, 3b, 3c, 3d). The abstracted data for interval n can thus be denoted by B^n_{2a}, B^n_{2b}, B^n_{3a}, B^n_{3b}, B^n_{3c}, B^n_{3d}. Considering the neighbouring intervals, the total abstracted data can be represented by 18 $(= 6\ rules \times 3\ (n-1, n, n+1))$ elements. Each element has a value of 1 or 0 (rules exist or not). The existence of a local grouping boundary (b) is determined on the basis of these 18 elements.

9.7.2 Detecting the Priority of Local GPRs Using a Decision Tree

We chose C4.5, an algorithm developed by Quinlan (1993), to construct the decision tree. Figure 9.10 shows an example of the constructed decision tree. From the training data, we can obtain the conditional probability of local grouping boundaries for each

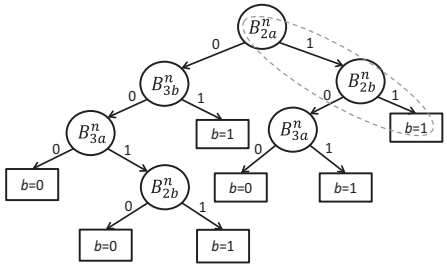

Fig. 9.10 Example of constructed decision tree

combination of local GPRs. When this conditional probability is 0.5 or more, GTTM detects the existence of a local grouping boundary ($b = 1$), and when it is less than 0.5, no boundary is detected ($b = 0$). For the example in Fig. 9.10, we detect a local grouping boundary when $B^n_{2a} = 1$ and $B^n_{2b} = 1$.

Unfortunately, the performance of the σGTTM analyser is not good enough because it can construct only one decision tree from 100 fragments of a piece and grouping analysis data, and it sometimes outputs irrelevant results.

9.8 σGTTM-II Analyser

We therefore developed the σGTTM-II analyser, based on the assumption that a piece of music has multiple interpretations; thus, it constructs multiple decision trees (each corresponding to a different interpretation) (Kanamori and Hamanaka, 2014).

The main idea with the σGTTM-II analyser is to reiterate clustering and statistical learning to classify each piece of music on the basis of the priority of the local GPR and to detect the local grouping structure more appropriately and easily. This analyser classifies a set of piece of music into clusters and outputs one detector of local grouping structure per cluster. We can detect a local grouping boundary more easily by choosing the most favourable detector from among various candidates.

First, we randomly classify training data into clusters. The training data of each cluster is then trained by a decision tree. After this training, a decision tree of GPR priority is constructed. We refer to this constructed decision tree as the "detector". In Fig. 9.11, clusters and detectors *A* and *B* mean that detector *A* is constructed in cluster *A*, detector *B* is constructed in cluster *B*, and so on. However, this part is problematic because an irrelevant analysed music structure might exist in a cluster. This is due to the detectors of each cluster representing the features of the entire music structure as the same for each cluster.

To solve this problem, the analyser evaluates the performance of each detector as it is constructed and then reclassifies the training data into clusters that generate the best performing detector. In Fig. 9.11, the clusters after reclassification are represented as A', B', and so on. The analyser then compares the training data of each cluster before $(A, B, \ldots)$ and after $(A', B', \ldots)$ reclassification. The less the training data in

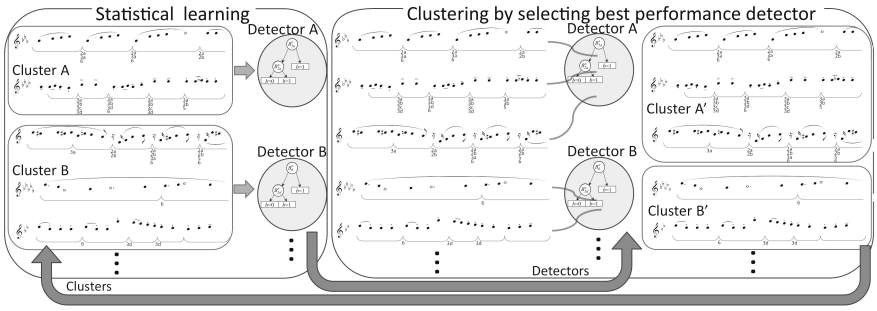

Fig. 9.11 Iterations of clustering and statistical learning

the cluster change, the more the constructed detectors cover the features such as priority of the local GPRs of all training data in the cluster.

After this comparison between clusters, if the total difference of training data in clusters before and after reclassification is more than two, the analyser returns to constructing detectors again, and if the total difference is less than one, or if reclassification has been performed 150 times, it outputs the training data and detectors of each cluster. Finally, we construct the most appropriate detector on the basis of the priority of the local GPRs of the entire training data in a cluster.

In our experiment in Sect. 9.11, we changed the initial number of clusters from 1 to 100 in order to compare the performance.

9.9 Interactive GTTM Analyser

We propose an interactive GTTM analyser that can use either the ATTA or σGTTM-II analyser (Fig. 9.12). We should point out that there is a trade-off relationship between the automation of the analysis process and variation in the analysis results (Fig. 9.13). Figure 9.14 shows an overview of our interactive GTTM analyser consisting of the ATTA/σGTTM-II analyser, GTTM manual editor, and GTTM process editor.

9.9.1 Manual Editor for GTTM

In some cases, ATTA may produce an acceptable result that reflects the user's interpretation, but in other cases it may not. A user who wants to change the analysis result according to his or her interpretation can use the GTTM manual editor. This editor has numerous functions including loading and saving the analysis results, calling the ATTA or σGTTM-II analyser, recording the editing history, undoing the editing, and autocorrecting incorrect structures.

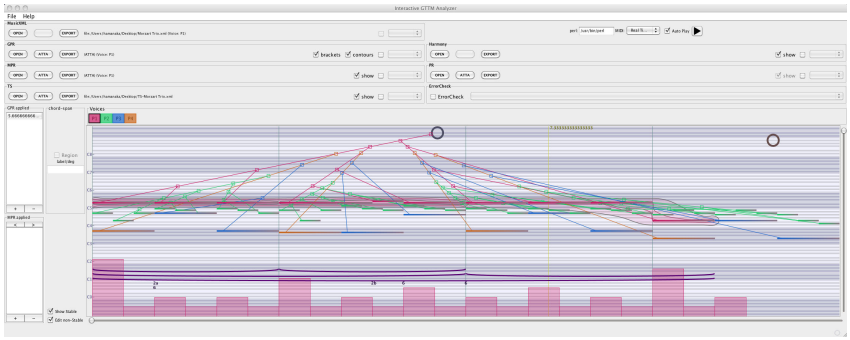

Fig. 9.12 Interactive GTTM analyser

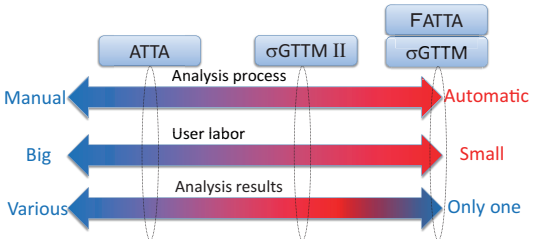

Fig. 9.13 Trade-off between automation of analysis process and variation of analysis results

9.9.2 Process Editor for GTTM

The analysing process with the ATTA and GTTM manual editor is complicated, and sometimes a user may become confused as to what he or she should do next, since there are three analysing processes in the ATTA and five editing processes in the GTTM manual editor. A user may iterate the ATTA and manual edit processes multiple times.

To solve this problem, we propose a GTTM process editor that presents candidates for the next process of analysis. A user can change the process simply by selecting the next process. The process editor enables seamless change in the analysis process by using the ATTA and, in the manual edit process, by using the GTTM manual editor, representing candidates for the next process of analysis. The representation method differs depending on the number of candidates for the next process.

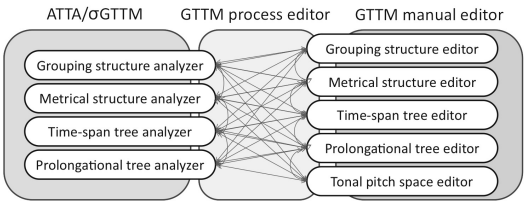

Fig. 9.14 Overview of interactive GTTM analyser

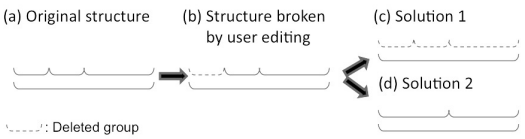

Fig. 9.15 Two types of solution for broken grouping structure

When there are multiple candidates, the process controlling function automatically opens the popup menu and shows the candidates. For example, if there is a grouping structure, as shown Fig. 9.15(a), and a user deletes a group at the upper left (Fig. 9.15(b)), the grouping structure of Fig. 9.15(b) is broken because GWFR3 does not hold. The GWFR3 has the constraint that a group must contain smaller groups. There are only two processes for solving this problem:

- Delete all the groups at the same level of the deleted group (Fig. 9.15(c)).
- Extend the group following the deleted group to the left (Fig. 9.15(d)).

The next process can then be executed depending on which of the two processes displayed in the popup menu the user selects.

9.9.3 Implementation on Client-Server System

The ATTA and σGTTM-II are updated frequently, and sometimes it is a little difficult for users to download an updated program. We therefore implement our interactive GTTM analyser on a client-server system. The graphic user interface on the client side runs as a Web application written in Java, while the analyser on the server side runs as a program written in Perl. This enables us to update the analyser frequently while allowing users to access the most recent version automatically.

9.10 GTTM Database

In constructing a musical analyser, test data from musical databases are useful for evaluating and improving the performance of the analyser. At present, we have a database of 300 analyses that are being used for researching music structural analysis (Hamanaka et al., 2014). At this stage, several rules in the theory allow only monophony, so we restrict the target analysis data to monophonic music in the GTTM database.

9.10.1 XML-Based Data Structure

We use an XML format for all analysis data. MusicXML was chosen as the primary input format because it provides a common exchange format for music notation, analysis, retrieval, and other applications (Recordare, 2011). We designed GroupingXML (XML format for grouping structure), MetricalXML (XML format for metrical structure), TimespanXML (XML format for time-span tree), and ProlongationalXML (XML format for prolongational tree) as the export formats for our four proposed analysers. We also designed HarmonicXML to express the chord progressions. The XML format is suitable for expressing the hierarchical grouping structures, metrical structures, time-span trees, and prolongational trees.

9.10.2 Score Data

The database should contain a variety of different musical pieces. When constructing it, we used 8-bar excerpts from whole pieces of music because the time required for analysing and editing by a musicology expert would be too long if whole pieces had to be analysed. We collected 300 monophonic 8-bar excerpts from classical music that included notes, rests, slurs, accents, and articulations entered manually with the *Finale* music notation software (MakeMusic, 2015). We exported the MusicXML using the *Dolet* plug-in.[3] The 300 whole pieces and the 8-bar segments were selected by a musicologist.

9.10.3 Analysis Data

We asked an expert musicologist to analyse manually the score data in a way that was faithful to GTTM by using the manual editor in the GTTM analysis tool to assist in editing the grouping structure, metrical structure, time-span tree, and prolongational tree. She also analysed the chord progression. Three other experts cross-checked these manually produced results.

9.11 Experimental Results

In this section, we compare the performance of our four analysers and show examples of the analysis results.

[3] http://www.musicxml.com/dolet-plugin/dolet-6-plugin-for-finale/

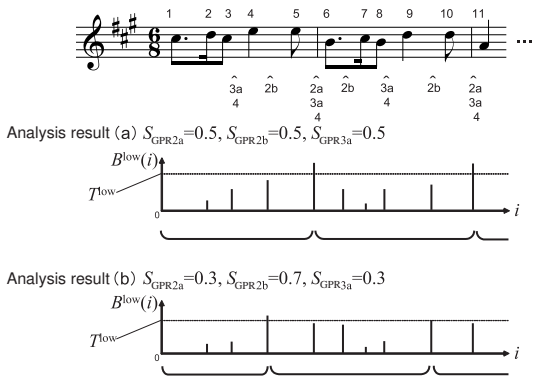

Fig. 9.16 Analysis of Mozart's Sonata K 331

9.11.1 Analysis by ATTA

There are at least two plausible grouping structures for the main theme from Mozart's Sonata K 331: a structure that has a boundary between notes 4 and 5 (Fig. 9.16(a)); or one in which there is a boundary between notes 5 and 6 (Fig. 9.16(b)). The analyser can output either of these grouping structures by using exGTTM with appropriate values for the strengths of grouping preference rules such as S_{GPR2a}, S_{GPR2b}, and S_{GPR3a}. In Fig. 9.16, T^{low}, where $0 \leq T^{\mathrm{low}} \leq 1$, is an adjustable parameter for the threshold in the low-level grouping boundary and $B^{\mathrm{low}}(i)$ is a local strength, represented by a real number between 0 and 1, defined so that the larger this value is, the more likely the boundary is.

Figure 9.17 shows the analyses of two pieces, Beethoven's "Turkish March" and the traditional English folk song "Greensleeves", that were set with the same parameters. The numbers at the nodes in the tree in Fig. 9.17 indicate the applicable rules. The parameters of ATTA are configured by hand, because the optimal values of the parameters depend on a piece of music.

9.11.2 Comparison of ATTA and FATTA

We evaluated the performance of ATTA and FATTA using an F-measure given by the weighted harmonic mean of precision P (proportion of selected groupings/dots/heads that are correct) and recall R (proportion of correct groupings/dots/heads that are identified). In calculating the F-measure of the grouping analyser and time-span tree analyser, we did not consider the possibility that a low-level error is propagated up to a higher level; we counted wrong answers without regard to the differences in grouping levels and time-span levels.

The grouping, metrical, and time-span tree structures will change depending on the adjustable parameters. To evaluate the baseline performance of ATTA, we used the following default parameters: S_{rules} = 0.5, T_{rules} = 0.5, W_s= 0.5, W_r = 0.5, W_l =

0.5, and σ= 0.05. The parameter range of T_{rules}, W_s, W_r, and W_l was 0 to 1.0 and the resolution was 0.1. The parameter range of σ was 0 to 0.1 and the resolution was 0.01 (see Table 9.1).

On average, it took about 10 minutes per piece to find a tuning for the set of parameters (Table 9.1). As a result of configuring the parameters, each F-measure of ATTA outperformed the baseline. After automatic parameter optimization, FATTA achieved average F-measures of 0.48 for grouping structure, 0.89 for metrical structure and 0.49 for time-span tree (see Table 9.2). FATTA thus outperformed the baseline of ATTA, but did not perform as well as ATTA when the latter's parameter values were configured by hand.

9.11.3 Number of Clusters in σGTTM-II Analyser

When we first classify each piece of music into clusters, we do not know the optimum number of clusters for producing the best performance of the σGTTM-II analyser.

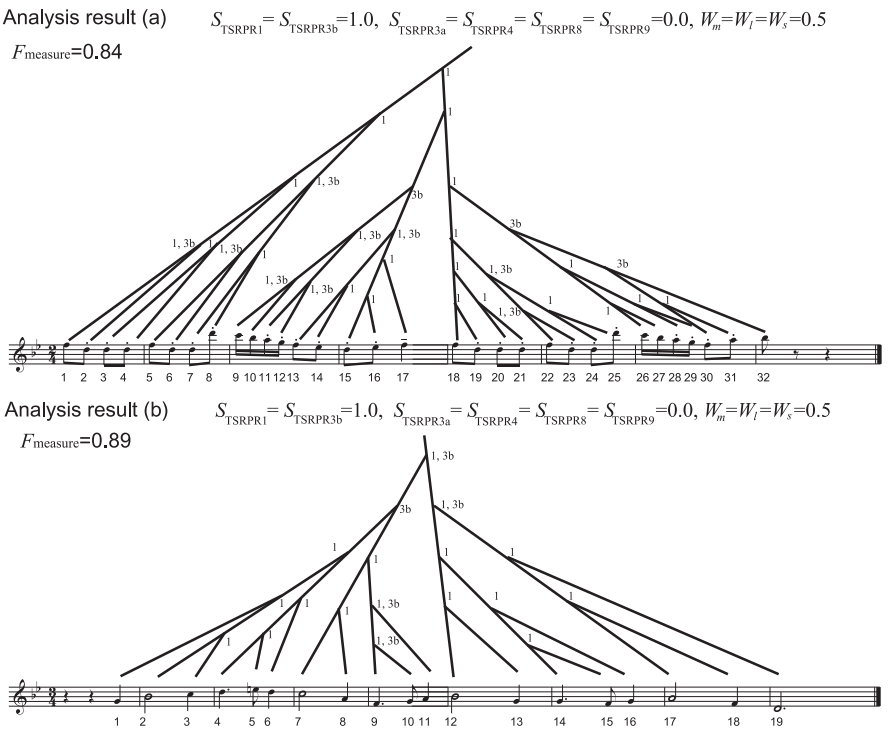

Fig. 9.17 Analysis of two pieces having same parameter sets: (a) Beethoven, "Turkish March", (b) English Traditional, "Greensleeves"

Table 9.1 Adjustable parameters

Structure	Parameter	Description
Grouping structure	$S_{\mathrm{GPR}j}$	The strength of each grouping preference rule. $j \in \{2a, 2b, 3a, 3b, 3c, 3d, 4, 5, 6\}$
	σ	The standard deviation of a normal distribution for GPR5.
	W_s	Preference weighting for end of a parallel segment in favour of the start of a parallel segment.
	W_r	Preference weighting of same rhythm in favour of the same register in parallel segments.
	W_l	Preference weighting for larger parallel segments.
	T^{GPR4}	The value of the threshold that decides whether GPRs 2 and 3 are relatively pronounced or not.
	$T^{low-level}$	The value of the threshold that decides whether transition i is a low-level boundary or not.
Metrical structure	$S_{\mathrm{MPR}j}$	The strength of each metrical preference rule. $j \in \{1, 2, 3, 4, 5a, 5b, 5c, 5d, 5e, 10\}$
	W_r	Preference weighting for same rhythm in favour of same register in parallel groups.
	$T^{\mathrm{MPR}j}$	The value of the threshold that decides whether or not each rule is applicable. $j \in \{4, 5a, 5b, 5c\}$
Time-span tree	$S_{\mathrm{TSRPR}j}$	The strength of each time-span tree preference rule. $j \in \{1, 3a, 3b, 4, 8, 9\}$

Therefore, we first tested the system with the number of clusters ranging from 1 to 100. This means the number of input clusters of the σGTTM-II analyser is one and the analyser outputs one detector, and then the number of input clusters is two and the analyser outputs two detectors, and so on. Thus, the analyser runs 100 times through the input and output. On each run, the analyser reiterates clustering and statistical learning multiple times until it is ready to output detectors. The results of this experiment are shown in Fig. 9.18.

Table 9.2 F-measures of ATTA and FATTA

	Baseline of ATTA	ATTA	FATTA
Grouping structure	0.46	0.77	0.48
Metrical structure	0.84	0.90	0.89
Time-span tree	0.44	0.60	0.49

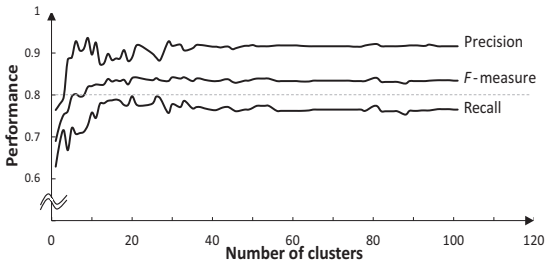

Fig. 9.18 Performance of σGTTM-II analyser

9.11.4 Comparison of ATTA, σGTTM Analyser, and σGTTM-II Analyser

We compared σGTTM-II with the ATTA and σGTTM analysers. The performance of the σGTTM-II analyser was highest when the number of clusters was 10. The σGTTM-II analyser outperformed the ATTA with adjusted parameters and the σGTTM analyser when it came to selecting the optimum detector (Table 9.3).

9.11.5 Comparison of Analysis Results from Two Musicologists with GTTM Database

Another musicologist who had not been involved in the construction of the GTTM database was asked to manually analyse the 300 scores in the database in a way that was faithful to GTTM. We provided her with only the 8-bar-long monophonic pieces but allowed her to refer to the original score as needed. When analysing pieces of music, she could not see the analysis results already in the GTTM database. She was told to take as much time as she needed. The time needed for analysing one song ranged from fifteen minutes to six hours.

The analysis results for 267 of the 300 pieces were the same as the original results in the GTTM database. The remaining 33 pieces had different interpretations, so we added 33 new analysis results to the GTTM database after they were cross-checked by three other experts.

For those 33 pieces with different interpretations, we found the grouping structure in the database to be the same as that obtained by the musicologist. For all 33 pieces, in the time-span tree, the root branch and branches directly connected to the root

Table 9.3 Performance comparison of ATTA, σGTTM analyser, and σGTTM-II analyser

	Precision	Recall	F-measure
ATTA	0.78	0.79	0.77
σGTTM	0.76	0.63	0.69
σGTTM-II	0.91	0.73	0.81

branch in the database were the same as those in the musicologist's results. In other words, only some branches were different in both analyses.

Of the pieces analysed, one of Johann Pachelbel's fugues in C major had the most unmatched time-spans when the analysis results in the GTTM database (Fig. 9.19(a)) were compared with those from the musicologist (Fig. 9.19(b)). From yet another musicologist, we obtained the following comments about different analysis results for this piece of music.

Analysis results from GTTM database In analysis result (a), note 2 was interpreted as the start of the subject of the fugue (Fig. 9.19(a)). Note 3 is more salient than note 2 because note 2 is a non-chord tone. Note 5 is the most salient note in the time-span tree of the first bar because notes 4 to 7 are a fifth chord and note 5 is a tonic of the chord. The reason that note 2 was interpreted as the start of the subject of the fugue is uncertain, but a musicologist who is familiar with music before the Baroque era should be able to see that note 2 is the start of the subject of the fugue.

Analysis results from musicologist Analysis result (b) was a more simple interpretation than result (a), in which note 1 is the start of the subject of the fuga. However, it is interesting that the trees of the second and third beats of the third bar are separated because both are the fifth chord.

The musicologist who made this comment said that it is difficult to analyse a monophonic piece of music from a contrapuntal piece of music without seeing the other parts. Chord information is necessary for a GTTM analysis, and a musicologist who is using only a monophonic piece of music has to imagine the other parts, which can result in multiple interpretations.

9.12 Conclusion

We have described our efforts to develop computer implementations of analytical methods based on music theory. By introducing full parameterization and statistical learning, we were able to improve the performance of our analysers. However, a few problems still remain. For example, none of our analysers can automatically analyse polyphonic music, which consists of several independent parts (Hamanaka et al., 2013). We plan to address this problem in future versions of the analyser.

The GTTM database contains analysis data for 300 monophonic pieces. However, the manual editor in our interactive GTTM analyser is designed to deal with polyphonic pieces. Although the analyser works only on monophonic pieces, a user can analyse polyphonic pieces by using the analyser's manual editor to divide polyphonic pieces into monophonic parts. We also attempted to extend the GTTM framework to enable the analysis of polyphonic pieces. We plan to publicize 100 pairs of polyphonic scores and the analysis results of the musicologists consulted in this study. Although the 300 pieces in the current GTTM database are only eight bars long, we

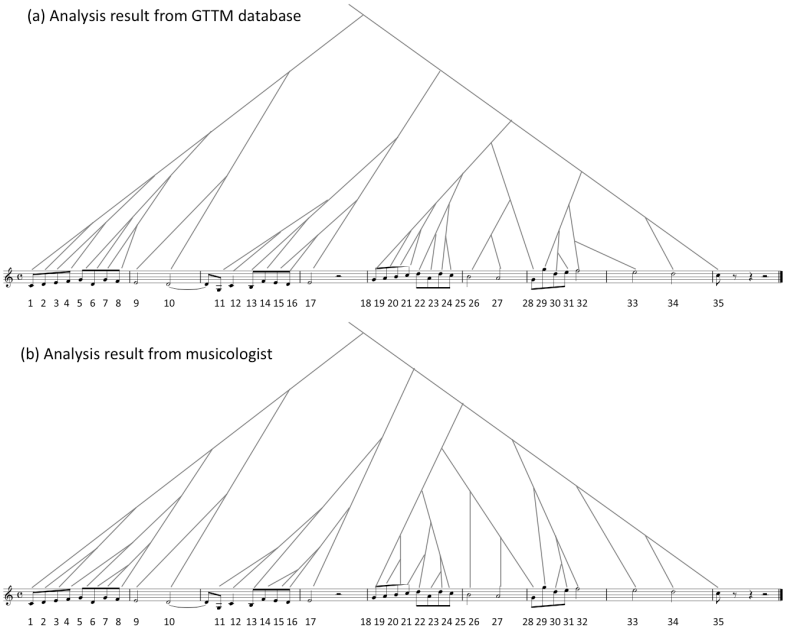

Fig. 9.19 Time-span trees of one part from a fugue in C major by Johann Pachelbel

plan to also analyse whole pieces of music by using the interactive GTTM analyser's slide bar for zooming piano-roll scores and GTTM structures.

Supplementary Material The interactive GTTM analyser and GTTM database are available online at http://www.gttm.jp/.

References

Cambouropoulos, E. (2006). Musical parallelism and melodic segmentation. *Music Perception*, 23(3):249–267.

Davies, M. and Bock, S. (2014). Evaluating the evaluation measures for beat tracking. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, pages 637–642, Taipei, Taiwan.

Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58.

Goto, M. (2001). An audio-based real-time beat tracking system for music with or without drum-sounds. *Journal of New Music Research*, 30(2):159–171.

Hamanaka, M., Hirata, K., and Tojo, S. (2006). Implementing "A Generative Theory of Tonal Music". *Journal of New Music Research*, 35(4):249–277.

Hamanaka, M., Hirata, K., and Tojo, S. (2007). FATTA: Full automatic time-span tree analyzer. In *Proceedings of the 2007 International Computer Music Conference (ICMC 2007)*, pages 153–156, Copenhagen, Denmark.

Hamanaka, M., Hirata, K., and Tojo, S. (2008a). Melody expectation method based on GTTM and TPS. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, pages 107–112, Philadelphia, PA.

Hamanaka, M., Hirata, K., and Tojo, S. (2008b). Melody morphing method based on GTTM. In *Proceedings of the 2008 International Computer Music Conference (ICMC 2008)*, pages 155–158, Belfast, UK.

Hamanaka, M., Hirata, K., and Tojo, S. (2013). Time-span tree analyzer for polyphonic music. In *10th International Symposium on Computer Music Multidisciplinary Research (CMMR 2013)*, pages 886–893, Marseille, France.

Hamanaka, M., Hirata, K., and Tojo, S. (2014). Music structural analysis database based on GTTM. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, pages 325–330, Taipei, Taiwan.

Hamanaka, M., Yoshiya, M., and Yoshida, S. (2011). Constructing music applications for smartphones. In *Proceedings of the 2011 International Computer Music Conference (ICMC 2011)*, pages 308–311, Huddersfield, UK.

Hewlett, W. and Selfridge-Field, E. (1998). *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11 of *Computing in Musicology*. MIT Press.

Hirata, K. and Hiraga, R. (2003). Ha-Hi-Hun plays Chopin's Etude. In *Working Notes of IJCAI-03 Workshop on Methods for Automatic Music Performance and their Applications in a Public Rendering Contest*, pages 72–73.

Hirata, K. and Matsuda, S. (2003). Interactive music summarization based on generative theory of tonal music. *Journal of New Music Research*, 5(2):165–177.

Hirata, K. and Matsuda, S. (2004). Annotated music for retrieval, reproduction. In *Proceedings of the 2004 International Computer Music Conference (ICMC 2004)*, pages 584–587, Miami, FL.

Kanamori, K. and Hamanaka, M. (2014). Method to detect GTTM local grouping boundaries based on clustering and statistical learning. In *Proceedings of the 2014 International Computer Music Conference (ICMC 2014)*, pages 1193–1197, Athens, Greece.

Lerdahl, F. (2001). *Tonal Pitch Space*. Oxford University Press.

Lerdahl, F. and Jackendoff, R. S. (1983). *A Generative Theory of Tonal Music*. MIT Press.

MakeMusic (2015). Finale. http://www.finalemusic.com/.

Marsden, A. (2011). Software for Schenkerian analysis. In *Proceedings of the 2011 International Computer Music Conference (ICMC2011)*, pages 673–676, Huddersfield, UK.

Miura, Y., Hamanaka, M., Hirata, K., and Tojo, S. (2009). Decision tree to detect GTTM group boundaries. In *Proceedings of the 2009 International Computer Music Conference (ICMC 2009)*, pages 125–128, Montreal, Canada.

Narmour, E. (1990). *The Analysis and Cognition of Basic Melodic Structures: The Implication–Realization Model*. University of Chicago Press.

Narmour, E. (1992). *The Analysis and Cognition of Melodic Complexity: The Implication–Realization Model*. University of Chicago Press.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Recordare (2011). MusicXML 3.0 tutorial. http://www.musicxml.com/wp-content/uploads/2012/12/musicxml-tutorial.pdf.

Rodriguez-Lopez, M., Volk, A., and Bountouridis, D. (2014). Multi-strategy segmentation of melodies. In *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, pages 207–212, Taipei, Taiwan.

Rosenthal, D. (1992). Emulation of human rhythm perception. *Computer Music Journal*, 16(1):64–76.

Schenker, H. (1935). *Der freie Satz*. Universal Edition. (Published in English as E. Oster (trans., ed.) *Free Composition*, Longman, New York, 1979.).

Stammen, D. R. and Pennycook, B. (1994). Real-time segmentation of music using an adaptation of Lerdahl and Jackendoff's grouping principles. In *Proceedings of the 3rd International Conference on Music Perception and Cognition (ICMPC 1994)*, pages 269–270, Liège, Belgium.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. MIT Press.

Yazawa, S., Hamanaka, M., and Utsuro, T. (2014). Melody generation system based on a theory of melody sequences. In *Proceedings of the International Conference on Advanced Informatics: Concepts, Theory and Applications*, pages 347–352.