Multiresolution Approaches to Audio Structure Analysis
–
Identifying Hierarchical Structures in Music

Diplomarbeit

Daniel Appelt

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Bonn, den 16. Juli 2008

Daniel Appelt

Acknowledgements

Many mentors, fellow students, colleagues, and friends have contributed in one way or another to the development of this diploma thesis.

First, special thanks go to the working group of Prof. Dr. Clausen at the University of Bonn, and, in particular, to my supervisor Dr. habil. Meinard Müller whose suggestions and constructive criticism had a major influence on this work.

Furthermore, I would like to express my gratitude to Gabriel Klein and Dr. Basim Al-Shaikhli who never hesitated to do another proof-reading, and whose comments and suggestions certainly improved this thesis.

Most part of this work came into existence in the branch library for medicine, science and agriculture at the University of Bonn. Special thanks go to all the people I met there, who cheered me up during the breaks and enjoyed lunch with me, in particular, Yvonne, Marieke, Julia, Katrin, Boris, Jan, Marcel, and Sebastian.

The development process as well as the writing of this thesis were greatly simplified by such sublime open source software projects as Linux, Octave, jEdit, Eclipse, LyX, and LaTeX. I am deeply indebted to their developers and supportive communities.

Finally, heartfelt thanks go to my family and all my friends who certainly suffered, time and again, from my commitment to this thesis, but, who, nonetheless, supported me all the time. Without you this work would not have been possible!

Contents

Contents

List of Figures

List of Tables

x

List of Algorithms

1 Introduction

The human cognitive system tends to organise perceived information into hierarchies and structures, a principle that also applies to music. Even listeners who are not musically trained unconsciously analyse music with regard to several structural aspects. These include, for example, parsing a musical piece into distinct sections, identifying main musical themes, and metric as well as tonal components such as tempo variations, or modulations.

The first-mentioned aspect is often referred to as the *grouping structure* of a musical piece which generally also implies a hierarchical segmentation. According to Marvin Minsky, one of the early pioneers in artificial intelligence, human listeners enjoy a "childlike fascination" [19] in constructing the grouping structure while listening. Nevertheless, given the vast amount of digital recordings available today and still being created, it is certainly unrealistic to expect that the world's music will be available along with suitable human annotations regarding its structure anytime in the future. On the other hand, any navigational aid to browse especially new and unknown music in large databases would be clearly appreciated by end-users. Being able to listen to the most representative sections in a piece of music as well as semantic intra-document browsing provides much more intuitive access to music than trial-listening methods such as intro scan mechanisms found in CD players, restricted playback access to randomly cut-out parts, or manually seeking interesting sections [12]. This is a prominent example where an automatic music structure analysis would be of great help. Other applications include the use in musical instruction or scientific analysis of music, as well as improving music audio data compression.

However, the automatic structural analysis of music is a hard problem. This is especially true if a musical piece is only given as digital audio recording, for example on an audio CD or as a file in MP3 format. Here, no structural information is explicitly provided. This problem, together with the broad range of structural aspects in music, led to the development of a multitude of techniques to reveal some of these components in digital audio recordings. Tempo and beat estimation, instrument identification, source separation, and classification are only a few of them.

In this diploma thesis, methodologies to improve the automatic analysis of grouping structure in music are examined. They are based largely on the foundations of an analysis system presented in [23] and [21]. Here, the approach is to first identify repeating sections in a piece of music and then, on the basis of this information, to compute an estimation of the global repetitive structure. In contrast to most other research in this field, the original system is able to correctly deal with a broad range of musical variations, for example in dynamics, timbre, execution of note groups,

and tempo progression. Therefore, it is especially suited to Western classical music where the aforementioned variations are commonly found. However, this specialisation expresses itself in a specifically chosen, fixed set of algorithm parameters, which do not cover popular music to the same extent. Likewise, and even more challengingly, the original system is confined to using a single given analysis resolution, that is, the degree of granularity used to mathematically approximate the audio signal under examination. In [23] and [21], it was already discovered that varying resolutions reveal distinct structural aspects in music. Therefore, the development of a single hierarchical model capable of incorporating several analysis resolutions was stated as the most significant open problem in both publications. This is one of the major challenges that are addressed in this thesis.

To this end, in the original algorithm, an appropriate place to combine intermediate results obtained from parallel analysis using different resolutions has to be identified. In order to investigate this problem, a closer familiarity with the original algorithm has to be acquired, which is the purpose of chapter 2. Afterwards, chapter 3 investigates the above-mentioned problem, and as a main result, introduces a *multiresolution approach* to discover repetitions in a digital audio recording, with the aim to combine the strengths of each separate analysis resolution while avoiding their weaknesses. Furthermore, a method to automatically set the parameters that are relevant in this part of the original algorithm is proposed, improving on the analysis of popular music.

Along with an improved range of identified repetitions, the main goal of this thesis is the development of methodologies that employ this gain in information to better estimate the global grouping structure of a piece of music. With respect to this, a clustering algorithm capable of computing a hierarchical structural overview from the set of repetitions is presented (chapter 4).

In order to quantify the improvements gained by both described techniques, experiments on a larger set of musical pieces are needed. Chapter 5 describes common approaches to automatic performance evaluation of music structure analysis systems. On the basis of this, a performance measure to compare a computed hierarchical structure against a manually annotated ground truth is presented along with its algorithmic implementation. Finally, corresponding evaluation results for the algorithms presented in chapters 3 and 4 are discussed.

Besides the enhanced audio player applications mentioned at the beginning, music structure analysis systems may be used to investigate the temporal relations between different interpretations of the same musical piece. This concept, also known as *partial audio synchronisation*, is introduced in chapter 6 as a further prominent application profiting heavily from an effective structural analysis.

Finally, chapter 7 summarises the presented work and gives an outlook to prospective future development.

2 Concepts and Fundamentals

This chapter introduces concepts and fundamentals relevant to automatic music structure analysis with special emphasis on the methods presented in [23] and [21]. Based on a given digital audio recording, the structural analysis of a musical piece generally proceeds in four steps.

First, as known from general information retrieval, the audio signal is transformed into a sequence of features, a concept that is introduced in section 2.1. In order to find repetitions in this sequence, every two audio features have to be compared. Therefore, in a second step, a suitable measure of similarity has to be chosen, on whose basis a similarity matrix is calculated (section 2.2). Successive components of strong similarity in this matrix reveal repetitions in the underlying feature sequence. In section 2.3 the notion of a path is introduced as a formal definition for this and a methodology to extract paths from similarity matrices is presented. Finally, section 2.4 illustrates a clustering method which derives an estimation of the global grouping structure from the set of extracted paths.

2.1 Audio Features

Often, a piece of music is only available as a digital audio recording. While generally providing great sound reproduction, this format may very well be the lowest possible level of semantic representation. Furthermore, it is extremely sensitive to the slightest variations, so that in most cases it is not suitable for musical analysis. A higher level of abstraction is needed, which in turn would induce a substantially lower data rate, allowing a much more efficient analysis.

For this purpose, in music retrieval an audio signal is transformed into a sequence $V := (v_1, v_2, \ldots, v_N)$ of feature vectors $v_n \in \mathcal{F}$, $1 \leq n \leq N$, each representing certain information inherent in the audio signal during a short time frame. Here, $\mathcal{F}$ denotes a suitable feature space, specifying the kind of information to be extracted.

Obviously, in this transformation some information is lost. Therefore, the feature vector space $\mathcal{F}$ has to be chosen carefully. Ideally, it should extract exactly the information which is relevant to the desired application or analysis domain. Additionally, it must allow the definition of meaningful measures of similarity, as most applications rely on comparisons between feature vectors.

In the case of music structure analysis, extracting information about played notes would be ideal. While reacquiring the original musical score from an audio signal is not possible in general, drawing conclusions about dominant musical notes and harmonies from an energy distribution across the audible frequency range may serve

as an approximation. In audio feature design, the most prominent technique to compute this energy distribution is known as short-time Fourier transform (STFT) [11, 4, 16].

The notion of a note is closely related to musical tuning systems, which define the mapping of notes to frequencies [2]. To allow for an efficient as well as intuitive musical analysis, the restriction of the feature space $\mathcal{F}$ to a certain tuning system is necessary. In this feature space, every dimension corresponds to a note in the respective musical tuning system. Thus, only those frequencies which are mapped to notes will be of interest in this constellation, and an analysis on the basis of notes becomes possible. Of course, a certain amount of energy in a note generally does not mean that this note was actually played by an instrument in the recorded audio. Accumulated overtone energies might also be responsible for this phenomenon. Nonetheless, in most cases, high energy in a note's frequency is in close relation to played musical notes.

This is especially true for Western music and its prominent twelve-tone equal temperament, where a close relationship between melody and harmony exerts this effect. Here, a further reduction to chroma is applied in most research [3]. Chroma is the set of 12 half tones making up an octave, i.e. $\left\{ \mathrm{C}, \mathrm{C}^\sharp, \mathrm{D}, \ldots, \mathrm{B} \right\}$. For chroma audio features, the energies of all notes belonging to the same chroma are summed up over all octaves. This yields a high degree of robustness to variations in timbre and articulation.

This thesis builds on *CENS* (chroma energy distribution normalised statistics) features, a special implementation of chroma features. They are calculated in a two-stage process. First, a filter bank is used to split the audio signal into 88 frequency subbands corresponding to notes A0 to C8 in the twelve-tone equal temperament. Next, every 100 ms an 88-dimensional feature vector spanning 200 ms of time is computed by convolving each of these subbands with a suitable window function. These are called short-time mean-square power (STMSP) features. Then, STMSPs for the set of chromas are computed by adding up corresponding STMSPs of respective notes. This yields real 12-dimensional vectors $v = (v_1, v_2, \ldots, v_{12}) \in \mathbb{R}^{12}$, where v_1 corresponds to chroma C, v_2 to chroma $\mathrm{C}^\sharp$, and so on. Finally, each vector v is normalised to $v/(\sum_{i=1}^{12} |v_i|)$ which evens out most dynamic variations. While these features already utilise the desired chroma representation, they are still too sensitive to most musical variations because of their fine temporal resolution of 10 features per second.

In the first step of the second stage, feature vector components are quantised causing another data reduction as well as a further invariance towards dynamic variations. To obtain the desired reduction in the temporal resolution, the resulting feature sequence is convolved using a Hann window over $\mathbf{w}$ consecutive vectors, whose outcome is downsampled with factor $\mathbf{d}$. Finally, after normalising resulting vectors with respect to the Euclidean norm, $\mathrm{CENS}_{\mathbf{d}}^{\mathbf{w}}$ features are obtained.
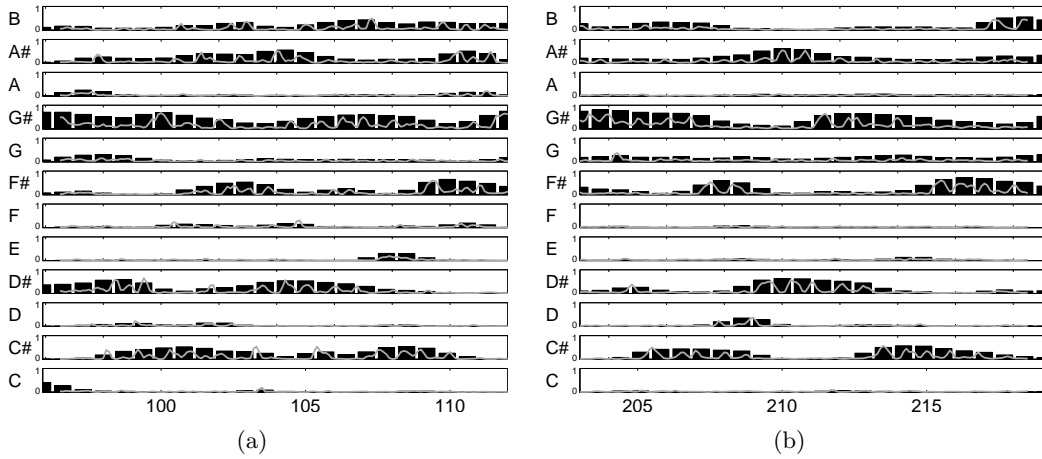
Figure 2.1: STMSP chroma (grey curves) and CENS_{10}^{41} (black bars) feature sequence of the segment $[96.5 : 112]$ seconds ((a) corresponding to the beginning of the first chorus section) and $[203 : 218.5]$ seconds ((b) corresponding to the beginning of the outro section) of *"Can't Take My Eyes Off You"* by *Barry Manilow*. Although this part of the outro section is a musical variation of the chorus which misses the vocals, for example, the harmonic similarity between both segments is clearly visible on the CENS feature level.

These features are elements of the set

$$\mathcal{F} := \left\{ v = (v_1, \ldots, v_{12}) \in [0,1]^{12} \mid \sqrt{\sum_{i=1}^{12} v_i^2} = 1 \right\}. \qquad (2.1)$$

In order to refer explicitly to values $\mathbf{w}$ and $\mathbf{d}$, a $\text{CENS}_{\mathbf{d}}^{\mathbf{w}}$ feature sequence will be denoted by $V[\mathbf{w}, \mathbf{d}] = (v[\mathbf{w}, \mathbf{d}]_1, v[\mathbf{w}, \mathbf{d}]_2, \ldots)$ in the following sections.

In general, $\mathbf{w}$ is chosen to be much larger than $\mathbf{d}$. This compensates for local time deviations as well as interpretative variations in the note execution of repetitions. At the same time, with larger $\mathbf{w}$, music characterisation is shifted from single notes to harmonies. Depending on the piece of music as well as the desired application, the possibility to vary both these parameters is essential. The described two-stage process perfectly serves this purpose.

Furthermore, only the first stage is computationally intensive because of the decomposition into the frequency subbands. Therefore, in practice, the first stage features are often pre-calculated, and, depending on the application and musical content, the second stage is computed as necessary with suitable parameters.

Figure 2.1 illustrates the differences between STMSP chroma and CENS features on an exemplary basis. The visible coarser temporal and dynamic resolution of the CENS_{10}^{41} features simplifies the identification of musically similar sections.

2.2 Similarity Measure and Similarity Matrix

Thanks to the Euclidean normalisation employed in the calculation of $\mathrm{CENS}_{\mathbf{d}}^{\mathbf{w}}$-features, meaningful similarity measures based on the inner product may be defined. In this case, the inner product coincides with the cosine of the angle between two vectors, yielding 1 if they are similar and 0 in the opposite case. Therefore, a *cost measure* $c : \mathcal{F} \times \mathcal{F} \to [0,1]$ may be defined by

$$c(v, w) := 1 - \langle v, w \rangle \tag{2.2}$$

for $\mathrm{CENS}_{\mathbf{d}}^{\mathbf{w}}$-vectors $v, w \in \mathcal{F}$.

In order to obtain the complete information about a cost measure with respect to a specific sequence of features, it needs to be evaluated for every pair of feature vectors in that sequence. An intuitive representation serving this purpose is the *self-similarity matrix*[1][10] which describes the complete cost measure map. It will be denoted by $\mathcal{S}[\mathbf{w}, \mathbf{d}]$, or simply by $\mathcal{S}$, if $\mathbf{w}$ and $\mathbf{d}$ are clear from the context, and is defined by its entries $\mathcal{S}(n, m) := c(v_n, v_m)$, $1 \leq n, m \leq N$. Two illustrations of similarity matrices are given in figure 2.2. Similarity matrices are quadratic and symmetric because of the symmetric cost measure. Additionally, they contain zeros on the diagonal as $c(v, v) = 0$ for every feature vector v.

In this thesis, repetitions in the feature sequence $V := (v_1, v_2, \ldots, v_N)$ are of particular interest. In $\mathcal{S}$, they are visible as lines or curves of entries exhibiting low cost. Usually, many additional matrix entries will have low values too, representing micro-term similarities which are not of interest in a global structural analysis. Additionally, these entries complicate the extraction of repetitions from $\mathcal{S}$. Therefore, improved cost measures enhance upon c by combining the costs of consecutive pairs of features that are necessary to form an acceptable repetition. Assuming constant tempo throughout a musical piece, filtering $\mathcal{S}$ along diagonals would be a solution.

But, tempo is, in general, not constant over time in music. Filtering along diagonals would have adverse effects on the similarity matrix in this case. A more flexible approach incorporates the efficient access to different temporal resolutions inherent in $\mathrm{CENS}_{\mathbf{d}}^{\mathbf{w}}$ features.

At the feature level, tempo changes may be locally simulated by modifying $\mathbf{w}$ and $\mathbf{d}$. For example, feature sequence indices of a faster repetition may be matched to a sequence of indices in the reference tempo by simultaneously lowering $\mathbf{w}$ and $\mathbf{d}$.

Therefore, a set of values $(\mathbf{w}_j, \mathbf{d}_j)$, $j \in [1 : T]$ along with reference tempo parameters $(\mathbf{w}, \mathbf{d})$ contained in this set are used to handle tempo variations. Then, a joint cost measure $c_L^{\min}$ may be defined by

$$c_L^{\min}(n, m) := \min_{j \in [1:T]} \frac{1}{L} \sum_{\ell=0}^{L-1} c(v[\mathbf{w}, \mathbf{d}]_{n+\ell}, v[\mathbf{w}_j, \mathbf{d}_j]_{m_j+\ell}), \tag{2.3}$$

---

[1]Strictly speaking this sections should be called *Cost Measure and Cost Matrix*. However, the terms *similarity measure* and *similarity matrix* were established in previous research on music retrieval. As these are exactly inverse terms, the desired meaning should be easy to grasp.
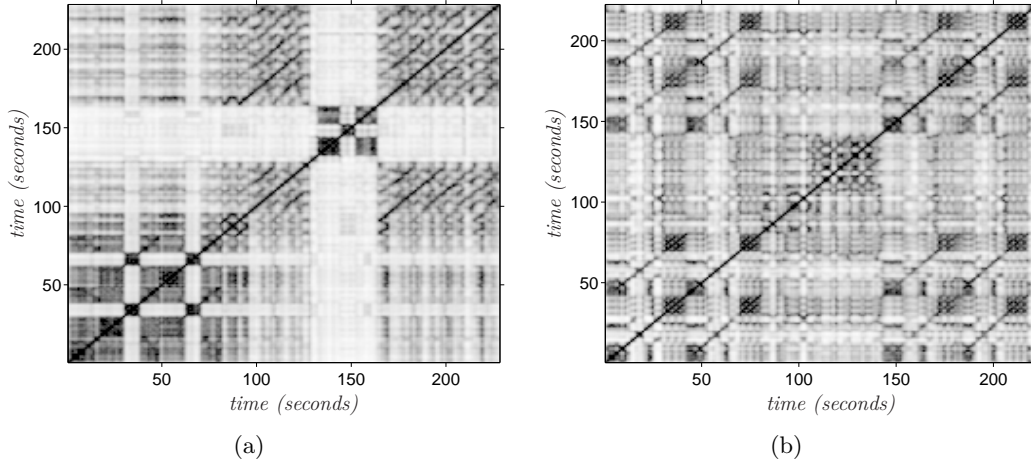
Figure 2.2: Self-similarity matrix $\mathcal{S}[41, 10]$ of (a) *"Can't Take My Eyes Off You"* by *Barry Manilow* and (b) a *Chailly* interpretation of *"Waltz 2"* from *"Suite for Variety Stage Orchestra"* by *Dimitri Shostakovich*. A strong similarity between two features is represented by a dark colour in the corresponding matrix entry. Furthermore, consecutive entries of strong similarity are visible which characterise musically similar sections.

where $m_j := \lceil m \cdot \mathbf{d}/\mathbf{d}_j \rceil$ and $1 \leq n, m \leq N$, after the involved sequences $V[\mathbf{w}_j, \mathbf{d}_j]$ were suitably extended with zero-padding.

Assuming a reference tempo of 1 feature per second, indices n and m in equation (2.3) correspond to absolute time positions in seconds in the audio signal. In order to calculate $c_L^{\min}(n, m)$, the subsequence of $V[\mathbf{w}, \mathbf{d}]$ of length L starting at absolute time n is compared to the subsequence of $V[\mathbf{w}_j, \mathbf{d}_j]$ of the same length starting at absolute time m, which corresponds to position m_j in the corresponding feature sequence, for every $j \in [1 : T]$. Consequently, the resulting similarity matrix is denoted by $\mathcal{S}_L^{\min}$.

Table 2.1 on the following page shows the three tempo parameter sets used in this thesis. Based on STMSP chroma features having a resolution of 10 Hz, each set includes possible tempo variations around a $\text{CENS}_{\mathbf{d}}^{\mathbf{w}}$ target feature resolution of 1, 2 and 5 Hz, respectively.

Note that for higher feature resolutions, less tempo variations are supported. One reason for this is that higher resolutions reveal other repetitive aspects. For example, the corresponding features are less smoothed, because they represent a smaller time frame of music. The same is true in the contextual cost measure calculation, where for constant L, a shorter amount of time is taken into consideration. Finally, a higher resolution is naturally computationally more intensive.

Therefore, higher feature resolutions are mainly used to extract short, exact repetitions more accurately, whereas lower resolutions like 1 Hz are employed to extract repetitions on a large scale where more musical variations need to be tolerated.

Table 2.1: Tempo changes (tc) simulated by changing the window size **w** and the downsampling factor **d**.

				1 Hz						2 Hz		5 Hz
j	1	2	3	**4**	5	6	7	8	1	**2**	3	**1**
$\mathbf{w}_j$	29	33	37	**41**	45	49	53	57	17	**21**	25	**9**
$\mathbf{d}_j$	7	8	9	**10**	11	12	13	14	4	**5**	6	**2**
tc	1.43	1.25	1.1	**1.0**	0.9	0.83	0.77	0.7	1.25	**1.0**	0.83	**1.0**

As a last note, the row labelled *tc* in table 2.1 contains the tempo change factor $\mathbf{d}/\mathbf{d}_j$ with respect to the reference tempo whose parameters are set in bold font.

Figure 2.3 shows on an exemplary basis, that the repetitive structure is greatly enhanced in $\mathcal{S}_L^{\min}$ even in the presence of tempo variations in the musical piece. Simply speaking, entries belonging to lines and curves in $\mathcal{S}$ remain so in $\mathcal{S}_L^{\min}$ while the rest of the matrix is smoothed. On the other hand, a blurring effect occurs around those lines and curves, introducing some uncertainty, which is especially true at their end, as corresponding entries now incorporate higher costs of following entries.

2.3 Identifying Repetitions

As noted above, in a similarity matrix, repetitions are represented by lines or curves of entries exhibiting low values. This section presents a greedy strategy to efficiently identify repetitions in similarity matrices. In the following, c will denote a contextually enhanced cost measure like $c_L^{\min}$ and $\mathcal{S}$ a similarity matrix based on this concept.

To encode repetitions mathematically, the notion of a *path* is used. It is defined as a sequence $P = (p_1, p_2, ..., p_K)$ of pairs of feature indices $p_k = (n_k, m_k) \in [1 : N]^2$, $1 \leq k \leq K$, and may be associated to corresponding entries in the similarity matrix, that is, it consists of certain preimages of the map $\mathcal{S}$. In the following, p_k may also be denoted as a *path link*. Of course, further requirements need to be specified in order to construct paths that may stand for real repetitions. As, generally, exact repetitions in a feature sequence are the rare exception due to musical variations in the underlying piece of music, acceptable variations need to be modelled, too.

A *valid path* is a path adhering to a *step-size constraint* as well as certain constraints regarding its path links' cost measure values. In [23] and [21], the *step-size constraint*

$$p_{k+1} = p_k + \delta \text{ for some } \delta \in \Delta, \tag{2.4}$$

where $\Delta := \{(1,1),(1,2),(2,1)\}$ and $1 \leq k \leq K - 1$, is employed. Regarding the *projections* $\pi_1(P) := (n_1, \ldots, n_K)$ and $\pi_2(P) := (m_1, \ldots, m_K)$ of a valid path onto the feature index sequence, this means, on the one hand, that real segments in a piece of music may be described, and, on the other hand, that repetitions of varying

(a)   (b)
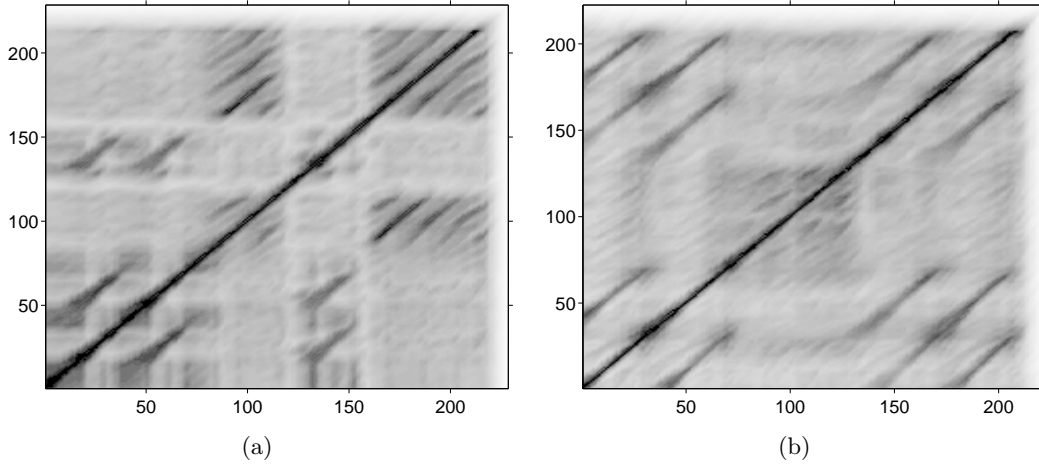
Figure 2.3: Enhanced self-similarity matrix $\mathcal{S}_{16}^{\min}[41, 10]$ of (a) the *Barry Manilow* and (b) the *Dimitri Shostakovich* example. In this representation, consecutive entries of strong similarity are accentuated which greatly simplifies the identification of musically similar sections. For example, the thick dark stripe from entry $(16, 48)$ to $(48, 80)$ in the *Barry Manilow* matrix corresponds to the repetition of the first verse, and the nearly identical stripe in the *Dimitri Shostakovich* example represents the repetition of the first A part in this musical piece.

tempo may be modelled. Regarding the first statement, $\pi_1(P)$ and $\pi_2(P)$ may be identified with the segments $\sigma_1(P) := [n_1 : n_K]$ and $\sigma_2(P) := [m_1 : m_K]$.

Furthermore, path links of a valid path are constrained by several *cost measure thresholds* as presented in table 2.2 on the next page. In [23] and [21], fixed values are chosen for these thresholds, mainly satisfying the demands of classical music.

Using these constraints, the objective of the greedy algorithm is to extract long valid paths. It proceeds in three stages. First, in an initialisation step, entries irrelevant to the path extraction are excluded from $\mathcal{S}$. As $\mathcal{S}$ is symmetric this includes everything below the diagonal. Additionally, the diagonal itself and a suitable surrounding are excluded. This is motivated by the fact that the diagonal contains solely the information that the whole feature sequence is similar to itself ($c(n, n) = 0$), and that this information was blurred into its surrounding by the contextually enhanced cost measure.

Next, a greedy strategy loop is used to extract paths. Here, the entry p of minimum cost is first sought in $\mathcal{S}$. If $c(p) < C_{in}$, a new path P is constructed starting from p. In this construction, all path links adhering to the path constraint are inspected at both end points of P, and the one with minimum cost is added to P as long as it has admissible cost (C_{ad} in table 2.2). Finally, all extracted path links, again, together with a suitable surrounding, are excluded from further inspection and the next loop iteration is started.

Table 2.2: Cost measure thresholds.

Threshold	Description	Constraint on a valid path $P = (p_1, p_2, ..., p_K)$
C_{in}	initial cost	$\exists k \in \{1, \ldots, K\} : c(p_k) < C_{in}$
C_{ad}	admissible cost	$\forall k \in \{1, \ldots, K\} : c(p_k) < C_{ad}$
C_{pr}	prune cost	$c(p_1) < C_{pr} \wedge c(p_K) < C_{pr}$
C_{av}	average cost	$\frac{1}{K} \sum_{k=1}^{K} c(p_k) < C_{av}$

In the post-processing stage, extracted paths are shortened at their end points until they fulfil the *prune cost constraint*. Next, all those paths which violate the *average cost constraint* or which are shorter than a length threshold K_0 are removed. Finally, all remaining paths $P = (p_1, p_2, ..., p_K)$ are extended again at their last link using a heuristic which creates path links that were lost due to the blurring introduced by the contextually enhanced cost measure (see end of section 2.2).

This leads to a set $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$ of valid paths, each corresponding to a pair of similar segments in the underlying audio recording. Furthermore, the tempo progression between two similar segments is encoded by means of the bijective map between the corresponding path's projections.

An illustration of the outlined procedure is given in figure 2.4 on the basis of the two running examples of this chapter.

2.4 Deriving a Global Structural Overview from Repetitions

In this section, an algorithm is presented that derives a global structural overview of a musical piece from the set of extracted valid paths $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$. Although the resulting structure will only be based on repetitions, it should still reveal underlying musical conceptions in many cases.

Before introducing the algorithm, some terms and notational conventions need to be established.

In the following, $\alpha = [s : t]$ denotes a *segment* consisting of indices in the feature sequence $V = (v_1, v_2, \ldots, v_N)$. Furthermore, a path $P = (p_1, p_2, ..., p_K)$ of pairs of feature indices $p_k = (n_k, m_k) \in [1 : N]^2$, $1 \leq k \leq K$ may be characterised by means of its *projections* $\pi_1(P) = (n_1, \ldots, n_K)$ and $\pi_2(P) = (m_1, \ldots, m_K)$. According to the path constraint $\Delta := \{(1, 1), (1, 2), (2, 1)\}$ introduced in last section, P encodes a bijective map between $\pi_1(P)$ and $\pi_2(P)$. Both projections correspond to the segments $\sigma_1(P) := [n_1 : n_K]$ and $\sigma_2(P) := [m_1 : m_K]$, respectively. On the basis of the bijective map, it is possible to calculate for every subsegment of $\sigma_1(P)$ a *suitable counterpart* in $\sigma_2(P)$ and vice versa.

The *support* of a set of segments $\alpha_m = [s_m : t_m]$, $1 \leq m \leq M$ is defined to be the subset

$$\text{supp}(\{\alpha_1, \ldots, \alpha_M\}) := \bigcup_{m=1}^{M} [s_m : t_m] \subset [1 : N]. \tag{2.5}$$

Figure 2.4: Illustration of the path extraction algorithm for the *Barry Manilow* (top row) and *Dimitri Shostakovich* (bottom row) example. The first figure shows all *admissible* entries that are originally contained in $\mathcal{S}_{16}^{\min}[41, 10]$. The second figure depicts the remaining admissible entries after the extraction of the first path, and the last figure finally shows all extracted paths.

Finally, a *similarity cluster* $\mathcal{A} = \{\alpha_1, \ldots, \alpha_M\}$ *of size* $M \in \mathbb{N}$ is a set of segments containing solely mutually similar segments. The following clustering algorithm aims to compute a *global structure*, that is, a complete set of relevant similarity clusters of maximal size.

Fundamentally, the algorithm is based around the notion of the *transitive closure*. That is, for a segment α which is similar to segment β which itself is similar to segment γ, it should be expected that all three segments are mutually similar and are, therefore, put into the same cluster. Unfortunately, as the information contained in the set of extracted paths incorporates some uncertainty, simply calculating the transitive closure will, in general, lead to unsatisfying results. For example, a cluster containing lots of slightly shifted elements might be a possible outcome.

The algorithm suggested in [23] and [21] tries to circumvent such inconsistencies by using a tolerance threshold when comparing cluster segments. It proceeds in three stages.

First, in the transitivity stage, a new cluster $\mathcal{C}_{P_i}^{\alpha}$ is constructed for every segment α in $P_i \in \mathcal{P}$. To this end, α is compared to every segment of paths $P_j \in \mathcal{P}$, $i \neq j$. If the intersection of both segments is sufficiently large compared to α, it is put into $\mathcal{C}_{P_i}^{\alpha}$ along with a suitable counterpart in P_j.

Then, in the merge step, every two clusters corresponding to a path's segments are merged into a single cluster $\mathcal{C}_{P_i}$. For that, all segments of both respective clusters

11

(a) (b)
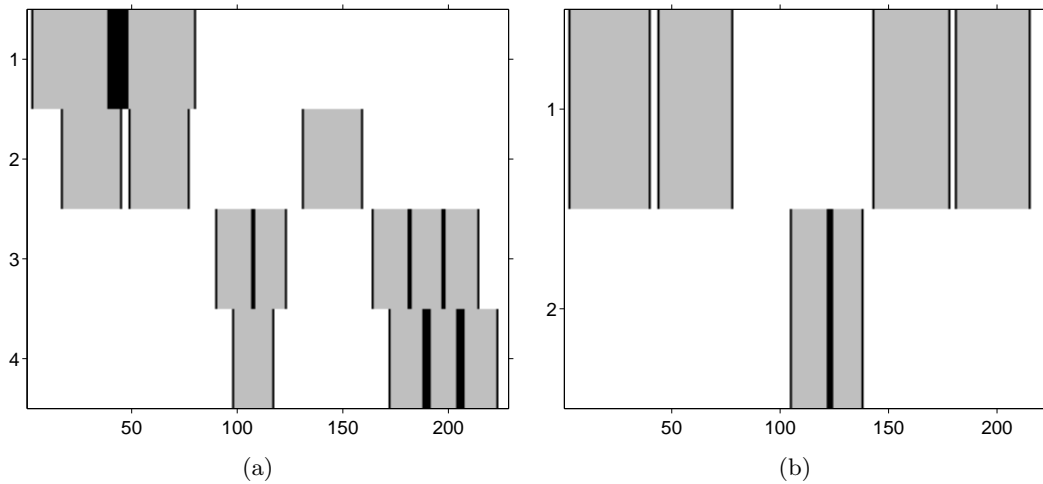
Figure 2.5: Final result of the clustering algorithm obtained for the (a) *Barry Manilow* and (b) *Dimitri Shostakovich* example. The second and third similarity cluster of the the *Barry Manilow* example correspond to the verse and chorus sections of the musical piece. The other two clusters complicate the structural overview a little bit, as they cover parts of the intro and outro section and additionally contain overlapping segments. The result for the *Dimitri Shostakovich* example correctly reflects the $A_1A_2BCA_3A_4$ structure inherent in this musical piece.

are compared in a pairwise fashion. Here, those segments are kept which are for the most part not covered by other segments. Additionally, if a segment is covered by other segments yielding a sufficiently large intersection, this intersection is kept too.

In the last stage, the created clusters are compared to each other and only the most representative clusters are kept. That is, all clusters are discarded that are *covered* by another cluster containing a larger number of segments. Here, given a threshold T_{dc}, cluster $\mathcal{A}$ is said to be a T_{dc}-*cover* of cluster $\mathcal{B}$ if $\text{supp}(\mathcal{A}) \cap \text{supp}(\mathcal{B})$ contains more than T_{dc} percent of $\text{supp}(\mathcal{B})$. In the case of a tie-break, clusters with smaller support are discarded.

Finally, figure 2.5 shows the clustering results that were obtained on the basis of this method for the two exemplary musical pieces used throughout this chapter. Here, a similarity cluster is represented as a row of segments. A segment is coloured in grey ●, and its start and end are highlighted by black lines. An overlap between segments is also coloured in black.

3 Identifying Repetitions in Music - Multiresolution Approaches

> Nothing contributes more to a melody's being "memorable"
> than incorporating notes, rhythms, and phrases that repeat.
>
> Jason Blume in *6 Steps to Songwriting Success*

Repetition is a key element in writing popular music with chart hit potential, but, although not featured that prominently, it is also an integral part in most other musical genres. As was pointed out in chapter 2, knowledge about repetitions in the sequence of notes or harmonies of a musical piece may be utilised to estimate its global grouping structure.

The repetitions that a listener identifies in a piece of music depend on his level of abstraction towards musical details. For example, an increased tolerance towards variations in dynamics, articulation, or the execution of note groups will allow for longer repetitions. Of course, there are certain musical schemes, like verse-chorus lineups in popular music, or traditional approaches to music analysis, which lead to consistent results if two musically trained people are asked to analyse the repetitive structure in a certain piece of music. However, this situation changes for music that does not adhere to traditional structural schemes. Also, especially with regard to a hierarchical structural overview, it is sometimes favourable to find all repetitions inherent in a musical piece.

The music structure analysis system presented in chapter 2 supports different levels of abstraction by means of varying the analysis resolution. The two-stage design of the $\mathrm{CENS}_{\mathrm{d}}^{\mathrm{w}}$ features together with the resolution-specific sets of predefined tempo change parameters (cf. table 2.1), provide the foundations for this behaviour. For example, increasing the feature resolution has two effects. On the one hand, finer repetitive structures are revealed. On the other hand, repetitions found at lower resolutions will be split into smaller pieces, because of an increased sensitivity towards musical variations. Regarding the two running examples from chapter 2, the effects of different feature resolutions on the analysis result may be examined in figure 3.1.

The major shortcoming in the design of the original analysis system is its restriction to using a single given analysis resolution at a time. This chapter presents three so-called *multiresolution* approaches to repeal this constraint. Here, the original algorithm is, to some degree, run in parallel using multiple resolutions. Obtained intermediate results (e.g. sets of identified repetitions) are subsequently merged into
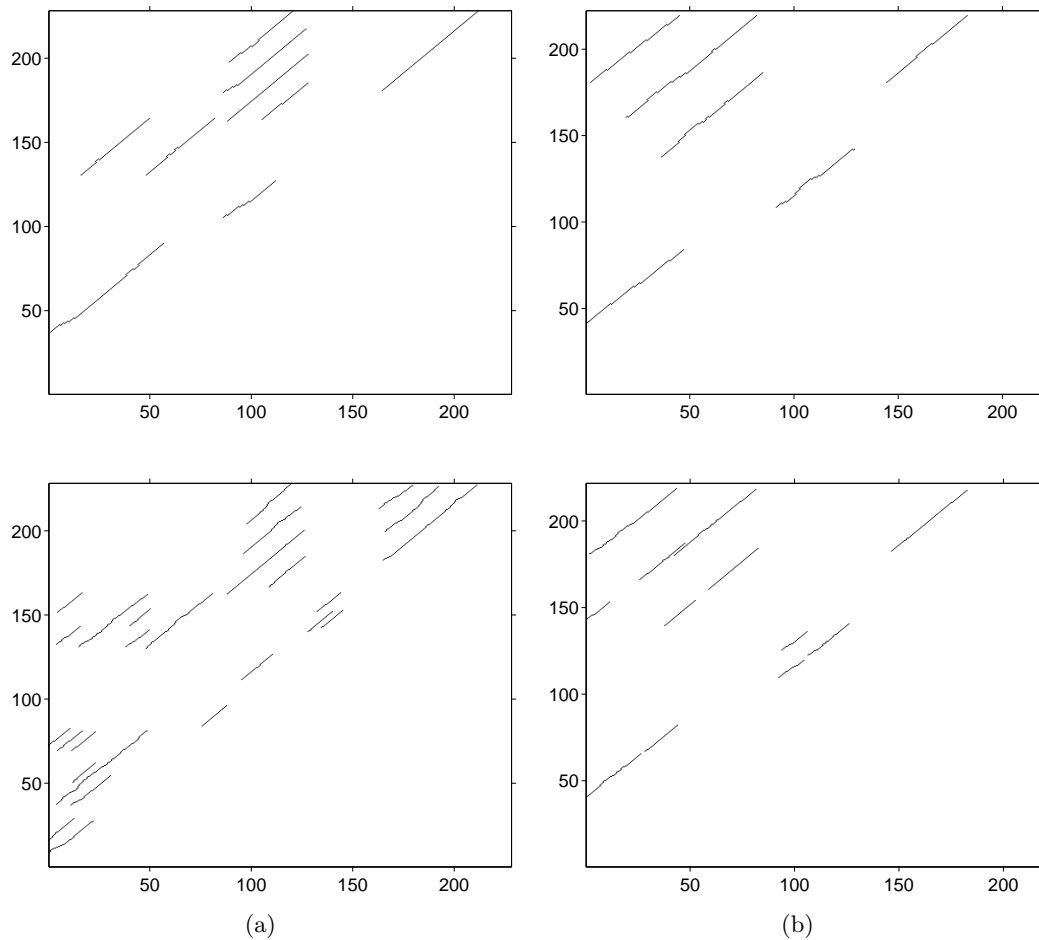
Figure 3.1: Path extraction results for (a) *"Can't Take My Eyes Off You"* by *Barry Manilow* and (b) a *Chailly* interpretation of *"Waltz 2"* from *"Suite for Variety Stage Orchestra"* by *Dimitri Shostakovich*. Results in the top row were obtained using a feature resolution of 1 Hz. For the bottom row 2 Hz was used. As is easily observable, the path structure at 2 Hz contains more details while some paths that are already present at 1 Hz are more fragmented.

a single result set. From there, a single instance of the algorithm computes the final results. The major difference between the techniques described in this chapter, is the point in the algorithm, where intermediate results from parallel analysis are combined. As a key requirement for all approaches, the final set of repetitions obtained in a multiresolution analysis should ideally be able to consistently describe all repetitions which would result from an individual analysis using any single resolution.

It is intentional that in the following sections, as few assumptions as possible are made about the underlying analysis system. There are several publications in which algorithms similar to the one presented in chapter 2 are described. In particular, only the following notions and concepts introduced in chapter 2 must be present.

Given a specific analysis resolution, repetitions are to be identified in a *contextually enhanced similarity matrix* $\mathcal{S}$ representing the pairwise similarity between every two features in a feature sequence. For a set of analysis resolutions Γ and every $\gamma \in \Gamma$, a corresponding similarity matrix $\mathcal{S}^\gamma$ may be computed by means of the analysis system under examination. Here, each resolution is given in features per second. In the case of the analysis system presented in chapter 2, these concepts are provided by $\mathcal{S}_L^{\min}$ and the set $\{1, 2, 5\}$. To formally refer to a repetition, the notion of a *path* is used, as introduced in section 2.3. Regarding the extraction of paths from similarity matrices, an *admissible cost threshold* C_{ad} equivalent to the one described in table 2.2 must exist, distinguishing *acceptable matrix entries* from entries that are not acceptable in the construction of a path. Of course, this threshold may change from resolution to resolution. Finally, in order to discriminate between objects specific to one of two distinct analysis resolutions, the terms *lower* and *higher* will be used, respectively.

In the first examined approach, sets of paths extracted using different resolutions are merged at the target resolution (section 3.1). While, essentially, this uses only valid information regarding repetitions in a piece of music, it is probably the most complicated method. As a possible simplification to this approach, the incorporation of lower resolution paths into higher-resolution similarity matrices is discussed in section 3.2. Finally, as the main result of this chapter, section 3.3 presents a methodology which performs the combination already on the similarity matrix level. Here, the admissible cost threshold is used to select entries in a lower-resolution similarity matrix, which should also be acceptable at a higher resolution, elevating musical robustness to this resolution. Furthermore, a similar technique may be used to eliminate a great amount of uncertainty in rearward path links introduced by smooth contextually enhanced similarity matrices (section 3.4).

Still preventing the universal applicability of the described techniques is the fact that the admissible cost thresholds are fixated beforehand. Section 3.5 presents a threshold selection algorithm which countervails even this last constraint. To illustrate all these techniques, the two examples presented in figure 3.1 will be used. Finally, a recapitulation of what has been achieved in this chapter and possible future research directions are pointed out in section 3.6.

<div align="center">(a)   (b)   (c)</div>
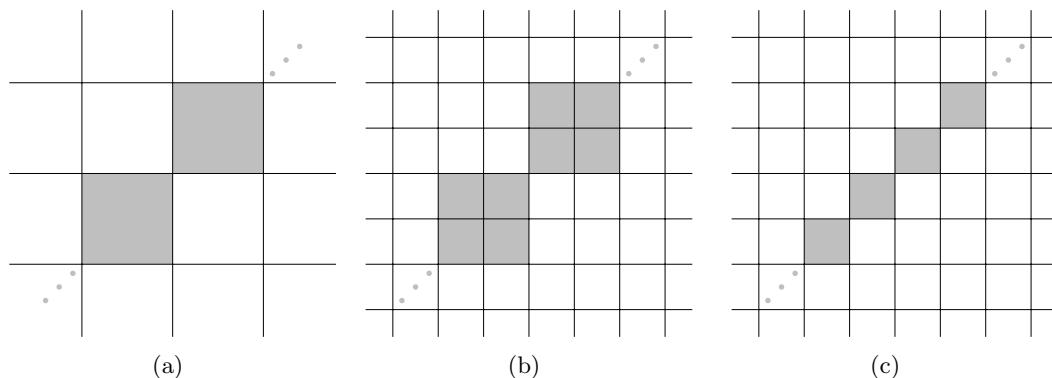
Figure 3.2: Upsampling a path from resolution $1\,\mathrm{Hz}$ (a) to resolution $2\,\mathrm{Hz}$ (b, c). The method depicted in (b) preserves all original path link information while the one illustrated in (c) is in conformity with the step-size constraint at a higher feature resolution.

3.1 Combining Paths of Different Resolutions

The multiresolution approach, which is probably the most straight-forward, is to join the sets of paths which resulted from parallel analysis at different feature resolutions. Here, two challenges have to be faced. First, a target resolution has to be chosen, to which all paths need to be resampled. Second, it has to be decided, what to do with conflicting paths from different resolutions.

Regarding the first question, as mentioned earlier, higher-resolution paths generally have better, more exact, end points, while at the same time a greater number of short paths is extracted. Additionally, a lot of those paths will generally be covered by lower resolution paths. Thus, they might be regarded as a refinement. In order to maintain this information, the target resolution needs to be at least as fine as the finest analysis resolution.

Next, a method to upsample paths of a lower resolution to the target resolution has to be chosen. Provided that the target resolution is a multiple of the analysis resolution, a method preserving all information needs to map a low resolution path link to all matrix entries that it covers in the target resolution. Figure 3.2 illustrates this on an exemplary basis. Here, as a major disadvantage, upsampled paths will presumably violate step-size constraint for paths like the one given in equation (2.4). In this case, further stages of the analysis system need to be modified. In order to circumvent this problem, upsampling methods are forced to incorporate heuristics (see figure 3.2(c) for an example), which, in general, will lead to weighted results. Finally, to obtain the best upsampling results possible, the target resolution needs to be a common multiple of all resolutions in Γ, which usually implies higher computational costs.

An even more delicate question is how to deal with conflicting paths. In practice, path extraction algorithms will compute consistent results, that is, the intersec-

tion of two paths will always be empty with regard to path links. Of course, this is not guaranteed, if two sets of paths are merged coming from different analysis resolutions. As mentioned above, higher-resolution paths should, ideally, be used to improve the temporal exactness of lower resolution paths *covering* them, which, unfortunately, is not a well defined relation due to uncertainties inherent in path extraction algorithms. First, it is possible that the distance between two paths is smaller than would be allowed in a direct path extraction. Furthermore, upsampled paths might cover two or more shifted repetitions from higher resolutions. Additionally, heuristics are, in general, already used to extend path ends in order to compensate for blurring effects in contextually enhanced similarity matrices. Upsampling them generally introduces even more inaccuracy. Consequently, it is a hard problem to decide which higher-resolution paths to select for a refinement, which path links or paths to dismiss, and how to bridge possible gaps between nearby paths.

In the course of this thesis, a pragmatic approach avoiding most of those issues was implemented. Here, the upsampling method first described is used, and path conflicts are simply ignored in the hope that the following algorithm stages are still well defined for the obtained results. With regard to both of the running examples introduced at the beginning this chapter, this approach resulted in extracted paths as illustrated in the first row of figure 3.3 on page 20. Most of the problems described in this section may be retrieved therein.

Another semantically simple approach is to put all resampled paths back into an empty similarity matrix, and to extract the final set of paths from there. This way, the obligation to resolve all conflicts is shifted to the path extraction algorithm. However, path extraction algorithms like the one described in chapter 2 are generally built with respect to particularities commonly found in contextually enhanced similarity matrices, that is, its values are expected to fall and rise smoothly. This is not the case here, as only those entries covered by paths will be set in this scenario. That is why smoothing this matrix would also be needed.

3.2 Incorporating Pre-Calculated Paths into Cost Matrices

Continuing from the reflections at the end of the previous section, the approach described now brings about a great simplification towards resolving conflicting paths. Here, paths extracted at a lower resolution are suitably upsampled and then incorporated into the similarity matrix at the next higher resolution.

This approach is outlined in algorithm 3.1, where set of multiresolution paths $\mathcal{P}$ is computed by iterating over the set of feature resolutions. In every iteration, previously computed paths are upsampled and then incorporated into the current similarity matrix. From this matrix, the next set of paths is extracted, and finally a new iteration is started.

Regarding the given algorithm, it has to be noted that lines 5 and 8 should not change anything in the first iteration.

Algorithm 3.1 Incorporating pre-calculated paths into cost matrices

---

 1: $\mathcal{P} \leftarrow \{\}$
 2: $\gamma \leftarrow \gamma_1$
 3:
 4: **for** $i \leftarrow 1 \ldots |\Gamma|$ **do**
 5: $\quad$ $\mathcal{P} \leftarrow$ upsample every $p \in \mathcal{P}$ from resolution γ to γ_i
 6: $\quad$ $\gamma \leftarrow \gamma_i$
 7: $\quad$ $\mathcal{S}^\gamma \leftarrow$ compute similarity matrix for resolution γ
 8: $\quad$ $\mathcal{S}^\gamma \leftarrow$ INCORPORATEPATHS($\mathcal{S}^\gamma, \mathcal{P}$)
 9: $\quad$ $\mathcal{P} \leftarrow$ extract paths from $\mathcal{S}^\gamma$
10: **end for**

Algorithm 3.2 INCORPORATEPATHS($\mathcal{S}, \mathcal{P}$)

Require: The set $\mathcal{P}$ of upsampled paths and matrix $\mathcal{S}$ of the next higher resolution.
Ensure: $\mathcal{S}$ includes information from upsampled paths.

 1: **for** $p \in \mathcal{P}$ **do**
 2: $\quad$ **for** $p_k = (n_k, m_k) \in p$ **do**
 3: $\quad\quad$ **if** $\mathcal{S}(n_k, m_k) \geq C_{ad}$ **then**
 4: $\quad\quad\quad$ $\mathcal{S}(n_k, m_k) \leftarrow C_{ad} - \varepsilon$
 5: $\quad\quad$ **end if**
 6: $\quad$ **end for**
 7: **end for**
 8:
 9: **return** $\mathcal{S}$

Similar to last section, a suitable upsampling method has to be chosen. But, the more important question is how to incorporate upsampled paths into the similarity matrix at the next higher resolution (line 8). As the main advantage of lower resolutions is their inherent stronger robustness towards musical variations, corresponding paths will, in general, encode longer repetitions than may be found at a higher resolution. Therefore, upsampled paths should be used to make the corresponding matrix entries acceptable for the next path extraction, if this is not yet the case. An implementation serving this purpose is presented in algorithm 3.2. Note that in line 4, $\mathcal{S}(n_k, m_k)$ needs to be set slightly lower than C_{ad}, as C_{ad} by definition (see table 2.2) contains the first non-acceptable cost value. Figure 3.3 (second row) illustrates this approach, again, using the two running examples of this chapter. Here, the same upsampling technique as in the last section was utilised.

As a main advantage, in this scenario path conflicts are avoided, as the last path extraction is predestined to compute the final result. Furthermore, the chosen upsampling method does not have as much influence on the final results as in the approach described in section 3.1, as here, upsampled paths are only used to make certain similarity matrix entries acceptable for the following path extraction. As, in

---

Algorithm 3.3 Combining cost matrices of different resolutions

---

1: $\mathcal{S} \leftarrow \{\}$
2: $\gamma \leftarrow \gamma_1$
3:
4: **for** $i \leftarrow 1 \dots |\Gamma|$ **do**
5:     $\mathcal{S} \leftarrow \max(\mathcal{S}, C_{ad}^{\gamma} - \varepsilon) * C_{ad}^{\gamma_i} / C_{ad}^{\gamma}$
6:     $\mathcal{S}_{up} \leftarrow$ upsample $\mathcal{S}$ from resolution γ to γ_i
7:     $\gamma \leftarrow \gamma_i$
8:     $\mathcal{S} \leftarrow$ compute similarity matrix for resolution γ
9:     $\mathcal{S} \leftarrow \min(\mathcal{S}, \mathcal{S}_{up})$
10: **end for**
11:
12: $\mathcal{P} \leftarrow$ extract paths from $\mathcal{S}$

---

general, further cost thresholds are used to decide where to start the construction of a path, and to determine the end points of a path, the modifications ideally only serve the desired purpose of reducing the fragmentation of path structures at higher resolutions. Therefore, this technique seems to be much more practicable.

Nevertheless, there are still some open problems. First, after the incorporation of upsampled paths, the smoothness inherent in the original similarity matrix might be compromised, as only those matrix entries covered by upsampled paths are modified. Second and more eminently, this approach relies heavily on the correctness of the path extraction, which has to be performed at every feature resolution. Here, the greater tolerance towards musical variations at lower resolutions may lead to slight shifts in the extracted paths, so that the entries that are modified in the higher-resolution matrix may not be acceptable as an extension in the following path constructions.

3.3 Combining Cost Matrices of Different Resolutions

The methodology presented in this section further improves on the problematic aspects identified at the end of section 3.2. Here, the combination is already performed on the similarity matrix level, leading to a simple and efficient multiresolution path extraction.

As may be seen in algorithm 3.3, this approach is very similar to the one presented in section 3.2. Only this time, the path extraction is not part of every iteration. On the contrary, the whole similarity matrix from a previous iteration is upsampled and then suitably incorporated into the similarity matrix at the current analysis resolution. This incorporation is realised by lines 5 and 9 in the algorithm.

First, the costs in the similarity matrix at a given resolution γ are adjusted so that all entries relevant to a path extraction become also admissible in a path extraction at the next higher resolution.
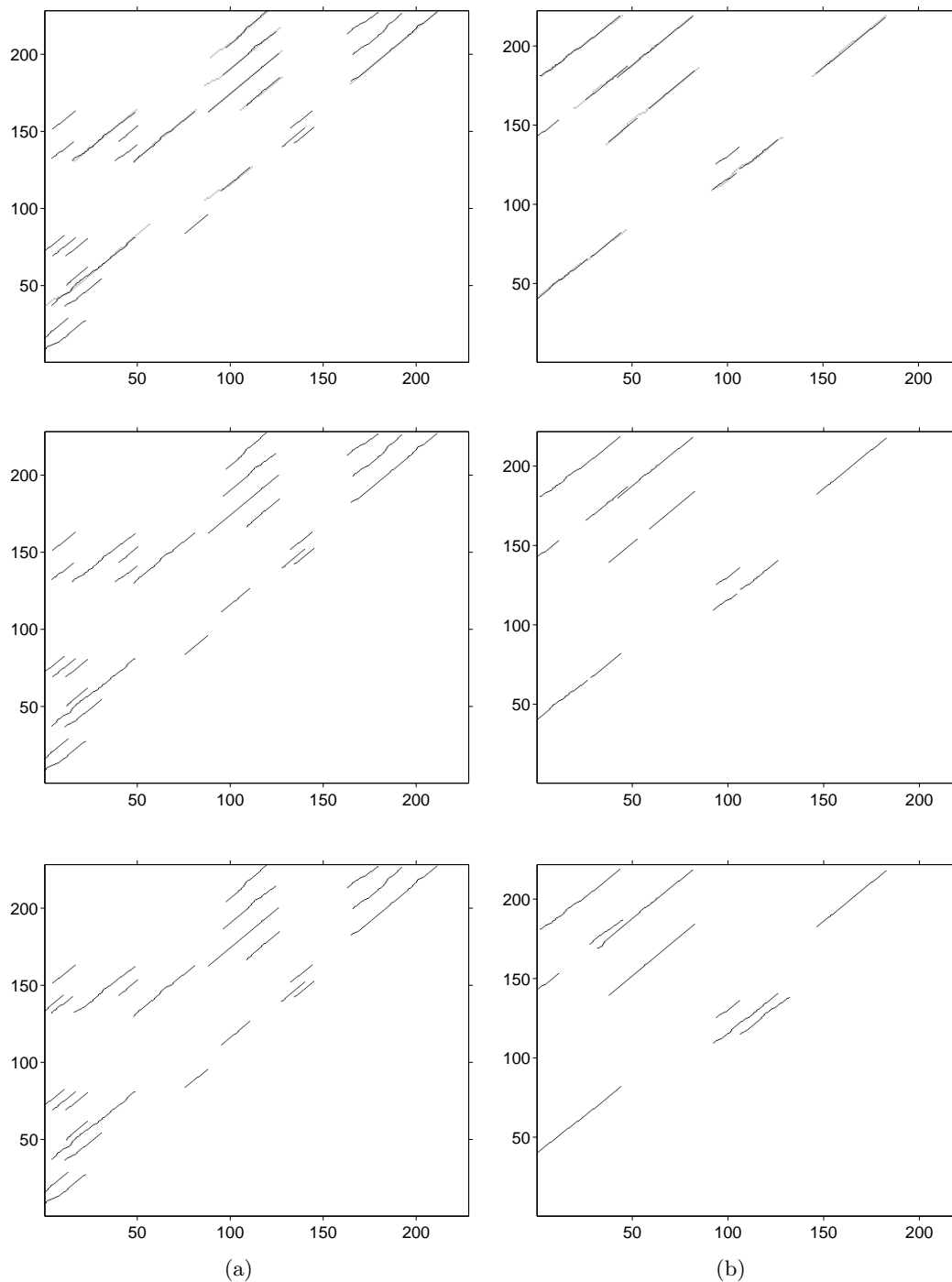
Figure 3.3: Multiresolution path extraction results (1 and 2 Hz) for (a) the *Barry Manilow* and (b) the *Dimitri Shostakovich* example. From top to bottom the following approaches were used: combining paths directly (section 3.1), incorporating extracted paths into similarity matrices (3.2), and combining similarity matrices (3.3). Different shades of grey are used in the first row to illustrate conflicting paths from both resolutions. Obviously, the last approach resolves most fragmentation effects.
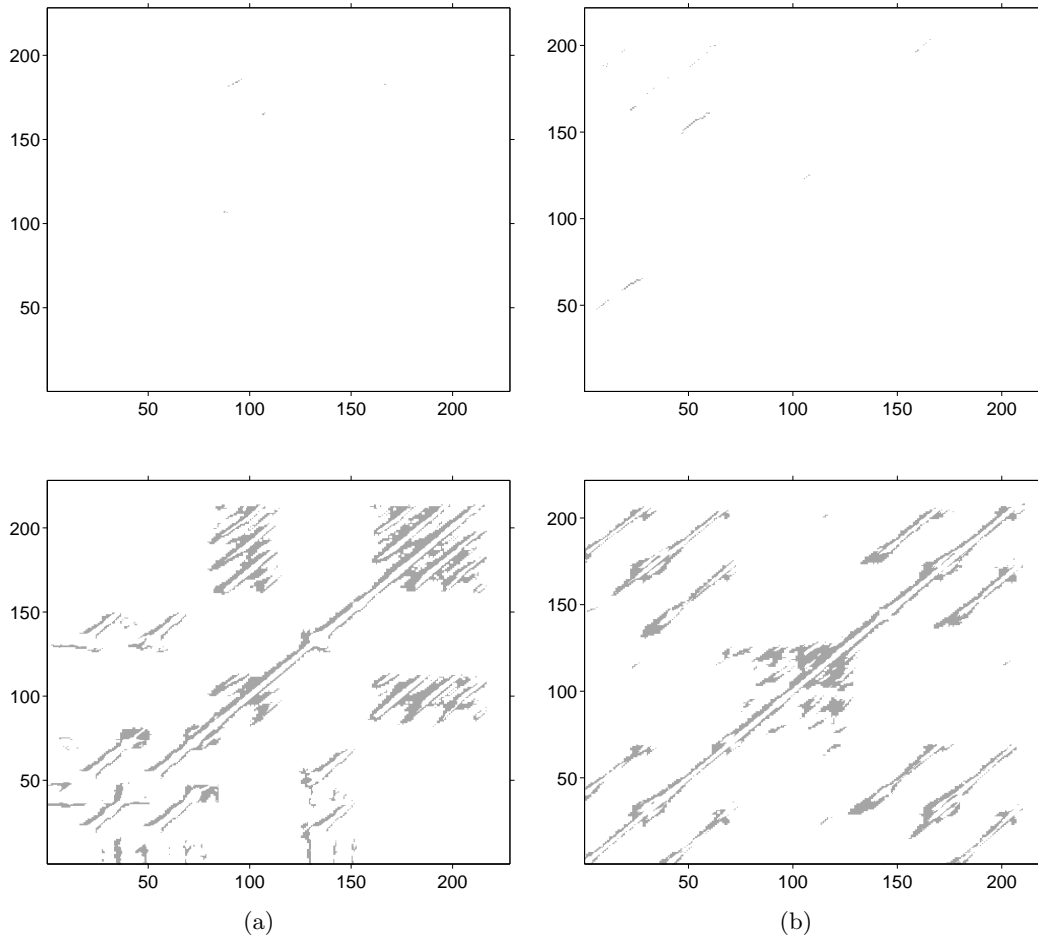
Figure 3.4: Adjusted entries in the 2 Hz similarity matrix of (a) the *Barry Manilow* example and (b) the *Dimitri Shostakovich* example. The non-white entries were set to admissible costs in the 2 Hz similarity matrix on the basis of the paths that were extracted at resolution 1 Hz (first row, section 3.2), or on the basis of the whole 1 Hz similarity matrix (section 3.3), respectively.

To this end, in line 5 of the algorithm, the values of the given matrix are first bounded by a maximum function and then adjusted to the admissible cost threshold at the next higher resolution. Here, the employed maximum function and the multiplication work on an per-entry basis.

This is the key element in allowing upsampled matrix entries to counteract fragmentation effects at higher resolutions. In this sense, it is similar to algorithm 3.2, only that here, every matrix entry is considered that would be acceptable in a potential path construction, whereas, in section 3.2, this is only the case for those entries that belong to extracted paths. Thus, one of the major concerns mentioned in that section is resolved here.

Furthermore, the statement in line 9 tries to maintain the smoothness that is typical for contextually enhanced similarity matrices. This is achieved, because all entries that are of lower cost in the upsampled matrix are transferred into the current similarity matrix. Consequently, whole regions of low costs are carried over into the fragmented areas. Additionally, a more sophisticated upsampling method may be employed, as now sufficient information for interpolation algorithms is available. To this end, *linear interpolation from nearest neighbours* proved to be most suitable.

The difference between the approach presented in this section and the one given in section 3.2 is illustrated in figure 3.4 on the basis of the two examples that were previously used in this chapter. Here, it is clearly visible that this section's method generally affects more matrix entries. The corresponding path extraction results given in the last row of figure 3.3 indicate that this effectively compensates the fragmentation effects that are introduced at higher resolutions.

3.4 Improving Path End Points

In order to make the path extraction more robust and efficient, often contextual information is added to similarity measures [3, 18, 21, 23]. This was also demonstrated in section 2.2. The only real problem introduced with these techniques is that, in general, heuristics are employed to estimate the exact end of a path, as it is generally blurred in the enhancement procedure.

A simple adaption of the technique presented in section 3.3 resolves this problem. It is outlined in algorithm 3.4, and depends on another cost threshold, C_{pr}, having the same meaning as in table 2.2. That is, the first and the last link of a valid path have to exhibit costs that are smaller than this threshold.

The key element here, is to incorporate information from the similarity matrix that is obtained for the reversed audio signal. For this matrix, a path extraction will compute *similar* repetitions to the standard case, but here, the beginning of a potential path corresponds to a blurred path end in the standard similarity matrix, and vice versa.

A combination of both matrices that replaces all blurred entries should ideally recover the temporal exactness inherent in similarity matrices that do not incorporate contextual information.

Algorithm 3.4 Improving path end points

1: $\mathcal{S} \leftarrow$ compute similarity matrix for feature sequence
2:
3: $\mathcal{S}_{re} \leftarrow$ compute similarity matrix for feature sequence of reversed audio signal
4: $\mathcal{S}_{re} \leftarrow$ flip $\mathcal{S}_{re}$ vertically and horizontally
5: $\mathcal{S}_{re} \leftarrow \max(\mathcal{S}_{re}, C_{pr} - \epsilon)$
6:
7: $\mathcal{S} \leftarrow \min(\mathcal{S}, \mathcal{S}_{re})$
8: $\mathcal{P} \leftarrow$ extract paths from $\mathcal{S}$

In order for this to work, the reversed similarity matrix first needs to be flipped (line 4) so that corresponding entries in both matrices represent the similarity between corresponding time frames.

In most cases the employed feature design will not allow to recreate the standard feature sequence from the feature sequence of the reversed audio signal. This is especially true in the case of $\mathrm{CENS}_{\mathrm{d}}^{\mathrm{w}}$ features. Here, the reversed sequence may be obtained from the reversed STMSP chroma feature sequence. Afterwards, the default parameters (see table 2.1) for the second stage of the feature computation cause each feature to contain a veritable amount of future information which introduces the desired robustness towards musical variations. Starting from that time, the values in normal and reversed feature sequences are no longer identical.

Because of this effect, paths that are extracted separately from both matrices will very likely be subtly shifted. Therefore, a straight combination of the normal and the reversed similarity matrix using the minimum will not lead to the desired result.

That is why, similar to the approach described in section 3.3, costs lower than the path pruning threshold are first adjusted in the reversed matrix (line 5), which generally affects those entries that would be used to start the creation of a new path. Then, after calculating the minimum of both matrices, information from the normal similarity matrix will largely determine the extracted path links, and information from the reversed matrix will help to extract more exact path endings.

Of course, this method may be easily integrated into algorithm 3.3. To this end, the reversed similarity matrix needs only to be computed for the highest feature resolution $\gamma_{|\Gamma|}$, and may then be incorporated into the final similarity matrix, just before the path extraction. Figure 3.5 illustrates the results that were obtained with this approach for the two running examples of this chapter.

3.5 An Automatic Threshold Selection Method

Experiments with popular music examples revealed that the techniques described in previous sections still suffer a limitation with regard to the employed cost thresholds.

In order to identify all relevant repetitions in a musical piece, these thresholds should be chosen in subjection to the distribution of cost values throughout the
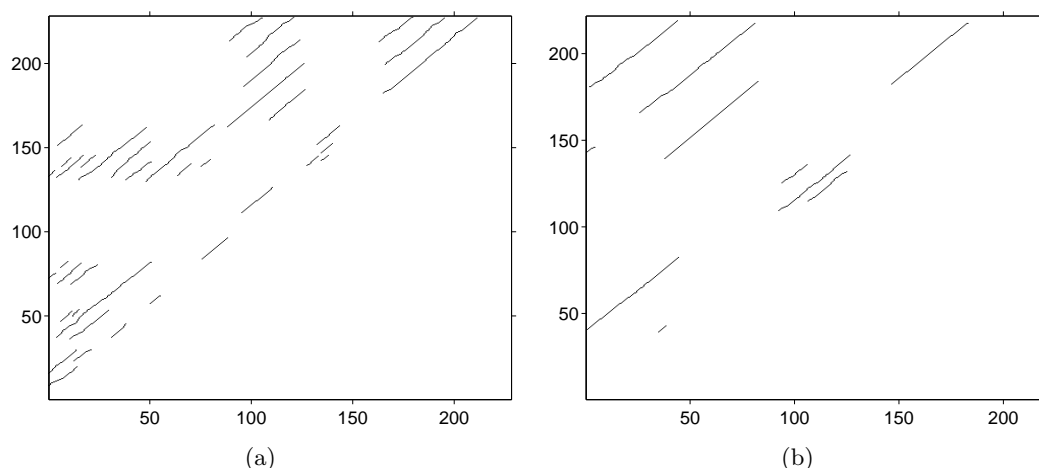
Figure 3.5: Multiresolution path extraction with improved path ends for (a) the *Barry Manilow* example and (b) the *Dimitri Shostakovich* example. Here, the 1 Hz and 2 Hz similarity matrices (section 3.3) were combined with the similarity matrix of the reversed audio signal at resolution 2 Hz.

similarity matrix. As mentioned at the beginning of this chapter, this is mostly due to musical uncleanness which may be seen as an integral part of popular music.

Threshold selection techniques are used to transfer grey-scale images into black-and-white images with the intention of separating darker foreground objects from a lighter background [29]. This description exactly corresponds to the meaning behind the admissible cost threshold C_{ad} used throughout this chapter. Here, the sought-after foreground objects are those regions that should be acceptable in the construction of valid paths, that is, lines or curves that are visible to the human eye in similarity matrix figures.

A threshold selection method serving this purpose was already suggested in [12] and refined in [18]. In both papers a method introduced by Otsu [24] is utilised to select a threshold which closely corresponds to C_{ad}. Therefore, similar approaches were investigated in the course of this thesis. To this end, standard thresholding techniques were applied to similarity matrix figures exported from Matlab using the open source image editor ImageJ[1].

The following observations are based on an empirical comparison of the analysis results with the personal perception of foreground and background objects. On the one hand, binarisation results depend on the mapping from cost values into the grey-scale spectrum in Matlab. A mapping on the basis of the double-logarithm improves the visibility of interesting regions in the similarity matrix both to the human eye as well as to threshold selection methods. On the other hand, the Isodata [28] and the aforementioned Otsu thresholding methods yield the best results.

---

[1]ImageJ `http://rsb.info.nih.gov/ij/`
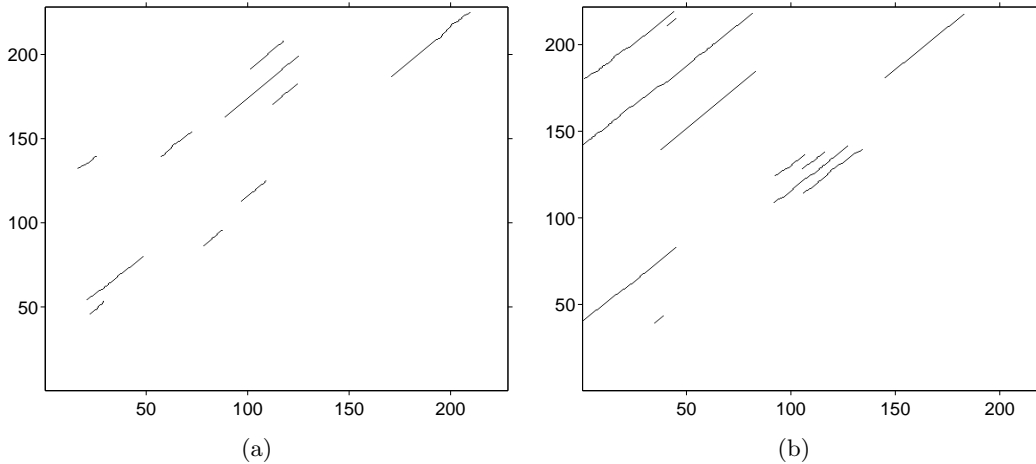
Figure 3.6: Multiresolution path extraction using an automatic threshold selection
for (a) the *Barry Manilow* example and (b) the *Dimitri Shostakovich*
example. Again, the 1 Hz and 2 Hz similarity matrices were combined
with the similarity matrix of the reversed audio signal at resolution 2 Hz
(cf. figure 3.5), but, this time, all thresholds relevant to the path extrac-
tion were chosen automatically. This method yields a clearly improved
result for the *Dimitri Shostakovich* example. On the one hand, the long
repetition from $(142, 5)$ to $(217, 81)$ is correctly detected, while, on the
other hand, not too much irrelevant information is added to the result.

With respect to these observations, a Matlab function was developed that should
be capable of deriving a suitable admissible cost threshold from a given similarity
matrix. To this end, the similarity matrix is first rearranged in order to restrict the
threshold selection algorithm to the part that is relevant for the path extraction.
First, the area above the diagonal and its blurred surroundings is extracted and
transformed into a rectangular matrix which generally is a requirement for thresh-
old selection algorithms. Then, the double-logarithmic mapping is applied to the
cost values of the resulting matrix. Finally, one of the above-mentioned threshold
selection methods is applied yielding the threshold value C_{ad}.

Other cost thresholds which are relevant to the path extraction (cf. table 2.2) are
derived from this value. To this end, C_{in} is first set to $C_{ad}/2$ and then iteratively
increased until at least an area of 2/3 of the transformed similarity matrix is neces-
sary to cover all entries fulfilling the initial cost constraint. Finally, C_{pr} is also set
to this value and C_{av} is set to be halfway between C_{in} and C_{ad}.

Experiments on the basis of this function mostly lead to superior results if the
Isodata algorithm is employed. Therefore, this method is preferably used in the
following.

The Otsu method often suffers from the phenomenon that the resulting threshold
causes a nearly equal number of entries to belong to the foreground and background,

respectively, which was also reported in [29]. In this case, C_{ad} is set inappropriately high which results in too many extracted paths.

In comparison to the previously employed fixed set of threshold values, the described selection method leads to comparable results for classical music while mostly improving the results for popular and jazz music. This should encourage a further development in this direction. With regard to the two running examples of this chapter, figure 3.6 depicts the result that were obtained using the described threshold selection method.

Nonetheless, there is still room for improvements as the thresholds are not always suitably chosen. For some examples, the chosen C_{ad} value diverges in both directions from the perfect value, but a clear pattern towards these effects could not yet be identified. Therefore, a theoretical justification for the general applicability of the chosen threshold selection method should be given.

Furthermore, the improvement towards the Otsu thresholding method described in [18] may be evaluated. Here, the threshold selection is confined to a defined amount of the similarity matrix entries emitting the highest values. This is substantiated by the fact that the Otsu method often classifies approximately half of all entries to be admissible.

3.6 Final Notes and Future Work

This chapter surveyed several techniques to improve the path extraction performance of the music structure analysis system introduced in chapter 2. To this end, three different basic approaches were developed.

First, a simple and efficient method for a multiresolution analysis of a musical piece was contrived (section 3.3). It is based on the observation that a higher analysis resolution generally improves the exactness of the extracted paths, while at the same time, the tolerance towards musical variation is reduced. Here, the start and end of repetitions are more accurately identified, whereas the extracted paths get easily too fragmented, which means that several short paths cover only the most prominent parts of a repetition. To this end, the suggested approach allows the incorporation of the robustness to musical variation that is present at a lower analysis resolution into the similarity matrix at a higher resolution.

This approach is nearly independent from the selected path extraction algorithm. Only a basic threshold needs to be available at each analysis resolution controlling which similarity matrix entries are acceptable in the path extraction and which are not. With respect to this, the employed heuristic implies that acceptable similarity matrix entries at a lower resolution should also be acceptable at the next higher resolution. This goal is achieved by algorithm 3.3. Experimental results indicate that the above-mentioned fragmentation phenomenon may be relieved in most cases. However, the presented approach relies on the validity of the above-mentioned heuristic which must not always be true, for example, if the threshold at the lower resolution is not correctly chosen.

Furthermore, a technique to improve the quality at the end of the extracted paths was introduced. It is virtually impossible to directly extract an exact ending of a path from an enhanced similarity matrix, as its entries generally represent the similarity between two time frames of veritable length in the underlying musical piece. That is why the path extraction method described in section 2.3 employs a simple heuristic to estimate a path's end.

In section 3.4, a technique resembling the previously described multiresolution approach is employed to improve upon this situation. To this end, the audio feature sequence and the corresponding similarity matrix of the reversed musical piece are computed.

As a more sophisticated heuristic, it is assumed that entries that fulfil the path pruning constraint with respect to the threshold C_{pr} in the reversed similarity matrix should also fulfil this constraint in the actual similarity matrix. Again, this clearly improves the path extraction results in the case of complicated repetitions. However, two corresponding entries from the regular and the reversed similarity matrix generally cover different portions of the underlying musical piece. Moreover, the manipulated similarity matrix entries are more relevant to the path extraction than in the previously described approach, as generally $C_{pr} < C_{ad}$. Thus, they may have too much influence on the path extraction results. For example, this may lead to paths that look more uneven, that is, the corresponding repetitions are seen to employ an unnatural number of tempo variations.

Finally, in section 3.5, automatic threshold selection methods were evaluated, as experiments with many popular and jazz pieces showed that the set of preselected threshold values that was employed so far is not appropriate for every musical piece. To this end, a given similarity matrix is reduced to the components that are relevant in the path extraction, and a threshold selection method is applied which originally allows the separation of a given picture into foreground objects and a background. On this basis, the admissible cost threshold for the path extraction is selected, and other relevant thresholds are derived from there.

Although this was only intended as a first test towards the applicability of automatic threshold selection methods, the results are mostly competitive in the cases where the preselected threshold values worked well, while superior results were obtained for popular and jazz music. Nonetheless, there is still room for improvement. Some possible future development directions were already discussed at the end of section 3.5, in order to give space to more global considerations in this section.

With respect to the multiresolution approach, a significant improvement in the accuracy of extracted paths may be expected if the window length of the employed audio features could be automatically adapted to the underlying musical material [31]. On the basis of onset detection or rhythmical and tempo analysis, it may be possible to calculate audio features that contain information corresponding to meaningful musical measures such as a complete musical bar.

Furthermore, a universally applicable automatic threshold selection method may attenuate the need to employ a multiresolution approach, as suitably chosen thresholds often avoid the fragmentation problem normally present at a higher resolution.

Therefore, a combination of both last-mentioned suggestions may yield superior results while still being computationally efficient.

In order to further improve the extraction of a path's end, the combination of the normal and reversed similarity matrix should be revised so that fluttering effects are avoided. To this end, a path could be extracted from the normal similarity matrix without directly applying a path end heuristic. Then, the entry corresponding to the last path link could be selected as starting point for a path extraction in the reversed matrix, only that here, new path links are confined to extend the path at its end.

Finally, it may even be possible to completely dispense with the kind of thresholds used so far. On the one hand, the first entry chosen for a new path is always the similarity matrix entry of minimum cost. Here, no threshold is required at all. On the other hand, a further entry should only be accepted as a new path link if its value is relatively high compared to the entries in a suitably chosen surrounding. In fact, this problem might be conveniently solved with an artificial neural network.

4 Clustering - from Repetitions to Musical Structure

The human cognitive system tends to organise perceived information into hierarchies and structures. According to the theory of cognition by Jackendoff and Lerdahl [14], listeners unconsciously organise music along three different dimensions. First, there is a grouping, or hierarchical structure, where smaller sections such as motifs or phrases are grouped into larger sections, ultimately ending up in a hierarchical tree of sections representing a musical piece in its entirety. Second, rhythmical emphasis in music is translated into a metric structure, which, in its most prominent form, would be annotated as time signatures in the score notation of a musical piece. Finally, there is a reductional structure which allows listeners to recognise main musical themes such as the chorus melody of a pop song independently of its surrounding arrangement.

As long as artificial intelligence is not on par with that of human beings, an automatic structural analysis which is based on the last-mentioned aspect is infeasible. Beyond that, an isolated analysis of the metric structure only offers a semantically poor explanation of a musical piece. However, the tree-like grouping structure of a musical piece is probably most often associated with musical structure. Therefore, the problem of computationally approximating the grouping structure is, in most cases, the first attempt to an automatic understanding of music. Accordingly, this chapter's main emphasis is placed on this subject.

Repetitions are one of the corner stones of grouping structures. In most cases, they are used intentionally by composers to trigger attention and to make a musical piece more "memorable" [5]. They may be present on a large scale such as repeating expositions in sonatas, as well as on a smaller scale, say repeating motifs. As shown in the last chapter, sophisticated methods to identify repetitions in digital audio recordings already exist. Therefore, in the majority of the research publications on automatic music structure analysis, the global structure inherent in a musical piece is approximated from the set of repetitions that were identified in it. This is also the basic approach employed in this chapter.

Methods that reveal inherent relations in a given set of objects are generally referred to as clustering techniques. A compact introduction to the above-mentioned clustering approach is given in [8]. A corresponding algorithm was already presented in section 2.4. Here, two starting points for major improvements were quickly discovered. On the one hand, the computed structural overview is, in general, not easy to interpret, as computed similarity clusters may contain overlaps in their own segments and with segments of other clusters (cf. figure 2.5). On the other hand,

the corresponding prototypical Matlab implementation contains numerous nested iterations over the set of identified repetitions, which seems to be a rather inefficient solution.

In most cases, manually resolving the above-mentioned conflicts in the computed similarity clusters leads to a hierarchical arrangement of clusters resembling the grouping structure introduced at the beginning of this chapter. The methodologies presented in this chapter may be used to directly estimate this grouping structure. In order to be computationally efficient, they are based on the *plane sweep* or *scan line* paradigm, a technique best known from computational geometry. In general, its use is geometrically motivated, in which case, a static n-dimensional problem is transformed into a dynamic $(n-1)$-dimensional problem. That is, the remaining dimension is used to iterate over the set of input objects. In the following, these objects are repeating segments in the analysed piece of music and the iteration works along the time domain. Thus, the sweep paradigm also designates a canonical approach to handle the clustering task.

In order to formalise the outlined approach and the desired outcome, section 4.1 establishes a hierarchy of mathematical terms and definitions. At the top of this hierarchy is the clustering result, a set of similarity clusters approximating the grouping structure of the musical piece. The input data for the clustering, that is, the repetitions that were identified in a piece of music constitute its foundations.

Nevertheless, all the clustering algorithms developed in this chapter already expect a set of clusters as their direct input. To this end, section 4.2 explains how to transform the given repetitions into a corresponding set of clusters. These clusters generally do not constitute the desired approximation of the grouping structure. For this, the actual clustering algorithms, which are all based on the plane sweep paradigm, are employed.

In order to familiarise the reader with this paradigm and its application to the above-mentioned clustering problem, section 4.3 first discusses a simpler, more classical clustering problem. In general, further transformations to the original set of clusters are necessary to obtain the desired result. A basic procedure supporting most of these transformations is to suitably split a given cluster into two disjoint subclusters. Section 4.4 introduces an algorithm for this task.

On the basis of this operation, a refinement of the first sweep clustering algorithm is possible which allows the computation of a hierarchical structural overview for a piece of music (section 4.5). Further techniques that provide a more efficient representation of the obtained result are presented in section 4.6. An insight into implementation details of the described clustering algorithms is given in section 4.7. Finally, section 4.8 presents related work on the topic including techniques specific to music structure analysis as well as general approaches to data clustering. It also highlights the achievements of the presented methodologies as well as directions for future development.
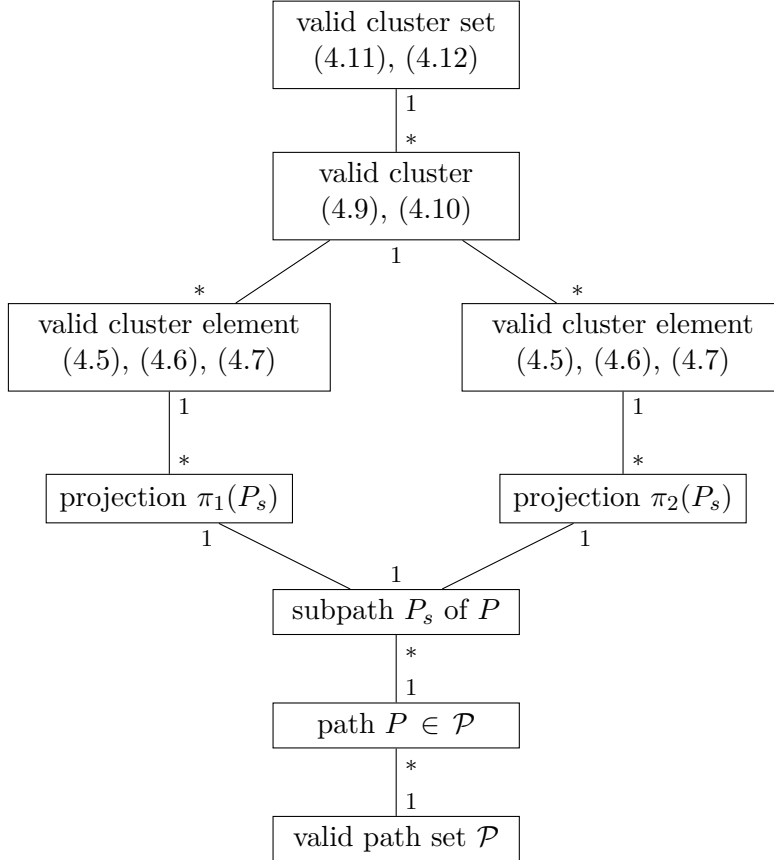
Figure 4.1: Overview of clustering data structures. The input of the clustering is the valid path set $\mathcal{P}$ at the bottom. The valid cluster set at the top is used to encode the requested approximation of the grouping structure. References to defining equations are given in braces and the multiplicity of instances of the depicted entities are annotated at the edges.

4.1 Formalisation of the Clustering Problem

In this section, the desired outcome of the clustering is formally specified. To this end, a hierarchy of definitions is introduced, which explains the connection between the repetitions that are used as input data, and the set of similarity clusters that constitute the requested approximation of the grouping structure. This connection is essential, as all the algorithms introduced in this chapter commonly need to access the provided input data in order to perform the data transformations that are necessary for the clustering. Furthermore, notational conventions are provided which allow for a cleaner and easier transcription of the algorithms.

Figure 4.1 provides an overview of the terms and definitions and shows their place in the aforementioned hierarchy. Additionally, this hierarchy characterises the associations between the corresponding data structures in the clustering algorithms.

In order to formalise the following concepts, the vocabulary that was established in chapter 2 is used.

As mentioned earlier, the clustering approach presented in this chapter is based on repetitions. Therefore, the notions *path*, *valid path*, and *projections of a path* onto the feature sequence are the most fundamental concepts.

The *valid path set* $\mathcal{P} = \{P_1, \ldots, P_M\}$ is the set of valid paths corresponding to the repetitions that were identified in a piece of music. These paths are assumed to be in compliance with the *step-size constraint* that was formulated in equation (2.4).

Then, given the feature sequence $V := (v_1, v_2, \ldots, v_N)$, a bijective map

$$\Phi_1^P : \pi_1(P) \quad \rightarrow \quad \pi_2(P) \tag{4.1}$$
$$n_k \quad \mapsto \quad m_k$$

may be defined between the projections of a valid path $P = (p_1, p_2, ..., p_K) \in \mathcal{P}$, with $p_k = (n_k, m_k) \in [1 : N]^2$, $1 \leq k \leq K$. Similarly, one defines $\Phi_2^P := (\Phi_1^P)^{-1}$.

As both projections encode subsequences in V, the order of the projections in P generally does not matter. Therefore, the identifiers accompanying Φ may be omitted if they are not explicitly necessary for the understanding. The same is true for the projections π_1 and π_2 if their corresponding path is evident from the context. Additionally, in order to refer to the matching projection inherent in P, $\Phi[\pi_1] = \pi_2$, and vice versa, may be used.

A valid path $P = (p_1, p_2, ..., p_K)$ may be seen as a map $P : [1 : K] \rightarrow [1 : N]^2$ with $P(k) = p_k$, $1 \leq k \leq K$. Similarly, one may use

$$\pi_1, \pi_2 : [1 : K] \rightarrow [1 : N].$$

Now, every restriction $[i : j] \subseteq [1 : K]$ of the index sequence induces a *subpath* $P_s = (p_i, p_{i+1}, \ldots, p_j)$ of P. In the following, subpaths will generally be constructed implicitly by restricting a single projection of a path to $[i : j]$. This is possible, as a single projection together with the corresponding map Φ suffices to completely describe a path. Moreover,

$$\Pi(P) := \{\pi_i(P_s), i \in [1 : 2] \,|\, P_s \text{ is subpath of } P\} \tag{4.2}$$

designates the set containing all projections onto the feature sequence of any subpath of P.

To obtain a clear and precise notation, the following operations are defined for projections π in the valid path P:

$$\begin{aligned} \text{START}(\pi) &:= \pi(1) \\ \text{END}(\pi) &:= \pi(K) \\ \text{LENGTH}(\pi) &:= |\pi(K) - \pi(1)|. \end{aligned}$$

Furthermore, π corresponds to the *segment* $[\text{START}(\pi) : \text{END}(\pi)]$.
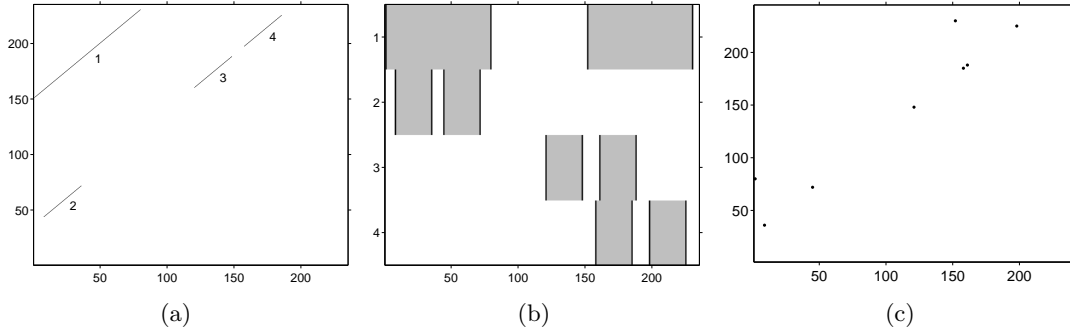
Figure 4.2: Three possibilities to visualise a set of paths. (a) Integration into the similarity matrix representation including a path index annotation. (b) Canonical representation of a path's projections as segments in the time domain. (c) A projection π of a path is depicted as point $(\text{START}(\pi), \text{END}(\pi)) \in \mathbb{R}^2$.

There are various possibilities to visualise a set of paths. For example, the three representations depicted in figure 4.2 all have advantages and disadvantages.

The representation used in previous chapters, given in figure 4.2(a), shows the direct correspondence to lines or curves of entries in the similarity matrix. It is very detailed, as even the point-wise relation between a path's projections is visible. The second illustration is a more canonical representation, where START, END, and LENGTH of a projection are easily discernible, but the point-wise relations are missing. Finally, figure 4.2(c) is the most compact representation of the set of paths. Here, for example, the projections of the first path are given by the points $(2, 80)$ and $(152, 230)$, respectively. While this representation excludes the information which projections are musically similar, it is, among others, simple to identify nearly *identical* projections. This representation more closely resembles classical clustering problems where points that lie close to one another have to be collected.

A *cluster element* containing L projections, is defined as $E = \{\pi_1, \ldots, \pi_L\}$ with $\pi_l = (n_{l_1}, \ldots, n_{l_{K_l}})$, $1 \leq l \leq L$, where

$$\exists P \in \mathcal{P} : \pi_l \in \Pi(P), \ 1 \leq l \leq L. \tag{4.3}$$

In the following, a cluster element will be used to model a *segment* of a *similarity cluster* as introduced in section 2.4. To this end, E is mapped to the segment $\alpha := [\text{START}(E) : \text{END}(E)]$ using its *average start* and *end points* defined by

$$\text{START}(E) := \text{round}\left(\frac{1}{L}\sum_{l=1}^{L} n_{l_1}\right) \text{ and } \text{END}(E) := \text{round}\left(\frac{1}{L}\sum_{l=1}^{L} n_{l_{K_l}}\right), \tag{4.4}$$

where mathematical rounding is used to regain valid feature indices. In this case, all of E's projections should substantially be identical.

Given an *identity threshold* $T_{id} \in \mathbb{N}_0$, E is said to fulfil the *identity constraint*, if

$$\forall i, j \in [1 : L] : |n_{i_1} - n_{j_1}| \leq T_{id} \wedge |n_{i_{K_i}} - n_{j_{K_j}}| \leq T_{id}, \tag{4.5}$$

that is, its projections may only differ by T_{id} indices at their start and end, respectively. Here, $T_{id} = 0$ is only useful under idealised conditions, as the given set of paths $\mathcal{P}$ generally contains uncertain information.

Furthermore, T_{id} induces a lower limit for the length of meaningful cluster elements. Given a *minimum length threshold* $T_{len} \in \mathbb{N}_0$, $T_{len} \geq 2T_{id}$, E is said to fulfil the *length constraint,* if

$$\text{LENGTH}(E) := |\text{END}(E) - \text{START}(E)| \geq T_{len}. \tag{4.6}$$

This constraint guarantees that start and end points of arbitrary projections contained in E are distinct.

Now, a *valid cluster element* is a cluster element fulfilling the identity and length constraint and, additionally,

$$\exists m : [1 : L] \to [1 : M], \text{ injective } : \pi_l \in \Pi(P_{m(l)}), \ 1 \leq l \leq L. \tag{4.7}$$

Here, the map m guarantees that every valid cluster element contains at most a single projection per valid path in the given set $\mathcal{P}$. There are two reasons why this constraint is useful. On the one hand, it serves efficiency, as the number of projections in a valid cluster element is bounded by the number of paths in $\mathcal{P}$, that is, by the size of the input to the clustering algorithm. On the other hand, multiple projections of the same path would need to be almost identical due to the identity constraint 4.5 and, therefore, would provide nearly no gain in information.

All clustering algorithms presented in this chapter work directly on valid cluster elements. Therefore, the given set of valid paths $\mathcal{P}$ must be mapped to valid cluster elements beforehand (cf. section 4.2). Here, at least equation (4.7) prevents that both projections associated with a path are put into the same cluster element. This induces that for every projection π in a cluster element there is a corresponding cluster element containing $\Phi[\pi]$, consequently, providing a connection between these both cluster elements.

Because of the similarity relation between the projections of a path, all elements in the set

$$\Omega(E) := \{\text{cluster element } E' \,|\, \exists \pi \in E \text{ with } \Phi[\pi] \in E'\} \tag{4.8}$$

are regarded as being *similar* to E. This constellation has a direct parallel in graph theory [1]. Here, it corresponds to a connected, undirected graph, where similar cluster elements stand for vertices, and the edges between them are given by Ω.

While the definition of a cluster element is directed towards storing a number of projections from different paths, the previous statements generally imply that each projection is put into a separate cluster element. That is why the question may arise, why cluster elements need to be defined in the first place.

With regard to the identity threshold T_{id}, it might be possible to merge valid cluster elements that encode nearly identical segments in the underlying piece of music. To this end, all involved projections are conflated in a single valid cluster element. Here, the challenge is to remain compliant with the previously established constraints, especially (4.7). With regard to this, two valid cluster elements E_1 and E_2 are called *mergeable*, if the cluster element $E = \{\pi | \pi \in E_1 \lor \pi \in E_2\}$ fulfils the identity constraint (4.5). A suitable merge operation for two mergeable, valid cluster elements that also enforces the other constraints is presented in section 4.3.

The fact that all cluster elements in the set $\Omega(E)$ of a merged valid cluster element E are seen to be similar to E implies a form of transitive closure. The intuition behind this is that if a section A in a piece of music is similar to a section B which itself is similar to a section C, then A should also be regarded as being similar to C. Ideally, the repetitions in the valid path set $\mathcal{P}$ exactly reflect this situation. In reality, there are often some paths that are not completely extracted because of a missing tolerance towards specific musical variations. For example, the build-up of tension in many musical pieces leads to a varying amount of musical variation in repetitions. By exploiting the above-mentioned transitive relations, it may be possible to repair these incomplete repetitive relations.

A *cluster* $\mathcal{C} = \{E_1, \ldots, E_K\}$ is a set of K valid cluster elements all being *mutually similar*, that is

$$\forall E \in \mathcal{C} \; : \; \Omega(E) \subseteq \mathcal{C}. \tag{4.9}$$

Regarding the above-mentioned parallel to graph theory, $\mathcal{C}$ corresponds to a connected graph. Furthermore, this definition implies that after merging two cluster elements, their corresponding clusters need to be merged too.

A cluster $\mathcal{C}$ may be mapped to a similarity cluster by using the segment representation for each of its contained cluster elements. $\mathcal{C}$ is a *valid cluster* if it fulfils

$$\forall i, j \in [1 : K], i \neq j : \alpha_i \cap \alpha_j = \emptyset, \tag{4.10}$$

that is, its corresponding similarity cluster segments do not overlap.

In its simplest form, a valid cluster consists solely of two cluster elements each corresponding to one of the projections of a path $P \in \mathcal{P}$, or a corresponding subpath P_s of P if this is necessary to fulfil the valid cluster constraints. This exactly corresponds to the situation outlined in figure 4.1. Furthermore, each of the previously used exemplary paths given in figure 4.2 may be transformed into a valid cluster this way. Then, figure 4.2(b) may also be seen as an illustration of these clusters where each cluster element E is represented by its corresponding segment $\alpha = [\text{START}(E) : \text{END}(E)]$, and each row represents a valid cluster.

Given an identity threshold $T_{id} = 3$, two mergeable cluster elements may be identified in this set of clusters. After merging both these elements the situation depicted in figure 4.3(a) is obtained. With respect to figure 4.2(b) it may also be seen that the two corresponding clusters were merged.
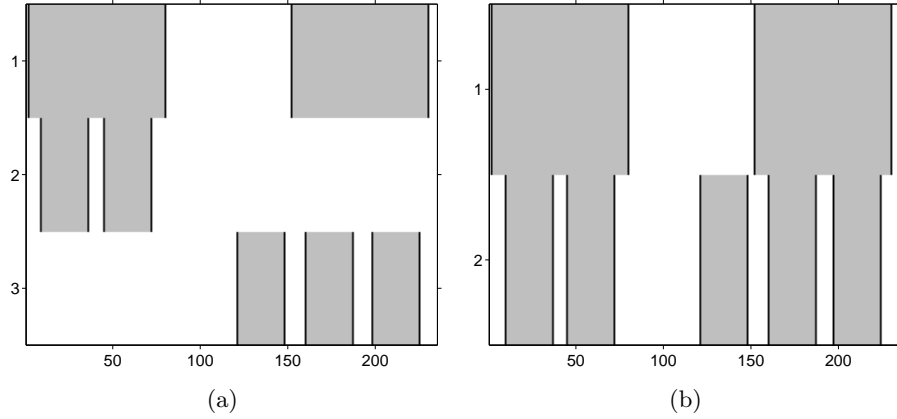
Figure 4.3: Transformation results for the previously depicted paths, (a) after conflating mergeable cluster elements, and (b) after a further transformation into a valid cluster set. A cluster element E is represented by the segment $\alpha := [\text{START}(E) : \text{END}(E)]$, and each row represents a cluster of similar cluster elements.

Finally, a *valid cluster set* $\{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ is a set of valid clusters $\mathcal{C}_l = \left\{ E_{l_1}, \ldots, E_{l_{K_l}} \right\}$, $1 \leq l \leq L$, if without loss of generality[1],

$$\forall (i,j) \in [1:L]^2, i < j, \forall (k,l) \in [1:K_i] \times [1:K_j]$$

either

$$\alpha_{i_k} \cap \alpha_{j_l} = \emptyset, \tag{4.11}$$

or

$$\alpha_{i_k} \cap \alpha_{j_l} = \alpha_{j_l}, \text{ and } \forall \pi_i \in E_{i_k} \, \exists \pi_j \in E_{j_l} \, \exists P \in \mathcal{P} : \pi_i, \pi_j \in \Pi(P) \tag{4.12}$$

holds.

In the latter case, E_{i_k} is said to *contain* cluster element E_{j_l}. As E_{j_l} is bound to contain a suitable restriction of every projection contained in E_{i_k}, a similar relation must be present for every other cluster element in $\mathcal{C}_i$ in order for $\mathcal{C}_j$ to be valid. This may be seen as a *consistency constraint* which guarantees that clusters with shorter, contained elements at least reproduce the information that is provided by the clusters that are associated with their containing elements.

A valid cluster set that is free of mergeable cluster elements while still retaining as much information as possible from the given set of paths $\mathcal{P}$ should be capable of providing an estimation of the grouping structure inherent in a piece of music. This is the desired result of the clustering approach introduced in this chapter. However, a result which complies with all the described constraints generally implies numerous manipulations to the originally given input data.

---

[1]That is, there is a suitable permutation of the set $[1:L]$.
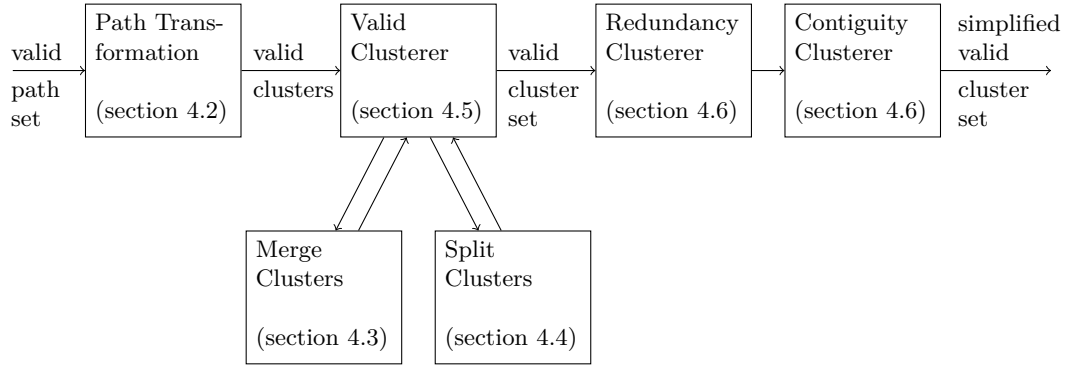
Figure 4.4: A block diagram of this chapter's clustering approach. The input of the clustering is the valid path set at the left. The simplified valid cluster set at the right encodes the requested approximation of the grouping structure. References to the respective sections are provided for each depicted algorithm.

A valid cluster set that may be obtained for the set of paths introduced in figure 4.2 is shown in figure 4.3(b).

In the following sections, several algorithms will be introduced which are capable of performing the necessary data transformations. In figure 4.4, a block diagram is given which visualises the sequence of these algorithms.

In order to foster a clear and easily readable notation in the the coming sections, the following definitions will be used with respect to cluster element $E = \{\pi_1, \ldots, \pi_L\}$ with $\pi_l = (n_{l_1}, \ldots, n_{l_{K_l}})$, $1 \leq l \leq L$:

$$
\begin{aligned}
\text{MINIMUMSTART}(E) &:= \min_{l=1,\ldots,L} (n_{l_1}) \\
\text{MAXIMUMSTART}(E) &:= \max_{l=1,\ldots,L} (n_{l_1}) \\
\text{MINIMUMEND}(E) &:= \min_{l=1,\ldots,L} (n_{l_{K_l}}) \\
\text{MAXIMUMEND}(E) &:= \max_{l=1,\ldots,L} (n_{l_{K_l}}).
\end{aligned}
$$

4.2 Transforming a Valid Path Set into a Set of Valid Clusters

All clustering algorithms that are introduced later in this chapter work directly on a given set of valid clusters. However, the actual input to the clustering is the result of a path extraction, that is, a valid path set.

A valid path encodes the temporal relation between two similar sections in the analysed piece of music. Therefore, the easiest way to obtain a usable input, is to transform each path into a valid cluster of two cluster elements, each containing one of the path's projections (cf. section 4.1).
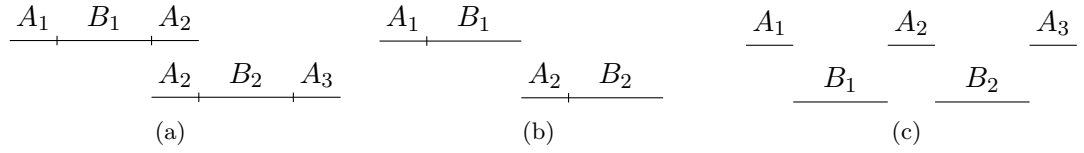
Figure 4.5: Resolving short to intermediate overlaps. (a) Original overlapping projections. (b) Both projections are shortened at their end if LENGTH(A_2) < T_{len}. (c) Otherwise, two valid clusters are necessary to retain all the information given by both projections.

As elements in a valid cluster need to fulfil equation (4.10), it is reasonable to first adjust those paths that contain overlapping projections. Here, the aim is to create a valid cluster that reproduces as much information as possible inherent in a given path. Additionally, requirements like equation (4.6) have to be respected.

Algorithm 4.1 on the facing page shows one possibility to transform a valid path into a set of valid clusters[2]. As mentioned above, its main task is to resolve a possible overlap given by

$$\pi_{1_2} = \pi_1 \cap \pi_2 = [\text{START}(\pi_2) : \text{END}(\pi_1)] \tag{4.13}$$

in line 2 of the algorithm. Depending on LENGTH(π_{1_2}), three different techniques are used to obtain valid clusters.

In the first two cases, the projections π_1 and π_2 may be identified with musical sections $A_1 B_1 A_2$ and $A_2 B_2 A_3$, exhibiting an overlap corresponding to section A_2. Both scenarios are outlined in figure 4.5. In the algorithm, this corresponds to lines 1 to 12. After the first split operation in line 2, one obtains $\pi_{1_1} = A_1 B_1$, $\pi_{1_2} = A_2$, $\pi_{2_1} = A_2 B_2$ and $\pi_{2_2} = A_3$, respectively.

If the overlap A_2 is of negligible length, that is LENGTH(A_2) < T_{len}, both projections are shortened at their end (line 5ff.) leading to a cluster with elements corresponding to $A_1 B_1$ and $A_2 B_2$, respectively. Here, the intuition is that paths extracted from contextually enhanced similarity matrices exhibit the most uncertain information at their end (cf. end of section 2.3). Additionally, in some applications, for example music players, it is desirable to be able to jump to the exact beginning of a section so that here no information should be discarded on the basis of simple heuristics.

In the second case, the overlap is of *intermediate* length so that two clusters are necessary to retain all the information inherent in the given path. To this end, another split operation is performed (line 10), leading to π_{3_1} and π_{3_2} which represent the musical sections A_2 and B_2, respectively. Then, the subprojections π_{1_2} and π_{3_1} corresponding to the overlap section A_2 are put into a single cluster element. Together with its similar sections A_1 and A_3 this leads to a cluster consisting of three elements. Both remaining B sections constitute a second, individual cluster.

---

[2]Note that cluster elements are simply represented as sets of projections in the algorithm.

Algorithm 4.1 Transforming a valid path set into a set of valid clusters

---

Require: A valid path $P = (p_1, \ldots, p_K)$ with $\text{START}(\pi_1) < \text{START}(\pi_2)$.
Ensure: A set of valid clusters $\mathcal{V}$ representing P.

1: **if** $\text{END}(\pi_1) \geq \text{START}(\pi_2)$ **then** ▷ resolve overlap
2:     $(\pi_{1_1}, \pi_{1_2}) \leftarrow$ split π_1 at value $\text{START}(\pi_2) - 1$
3:     $(\pi_{2_1}, \pi_{2_2}) \leftarrow (\Phi[\pi_{1_1}], \Phi[\pi_{1_2}])$
4:
5:     **if** $\text{LENGTH}(\pi_{1_2}) < T_{len}$ **then**
6:        $\mathcal{C} \leftarrow \{\{\pi_{1_1}\}, \{\pi_{2_1}]\}\}$
7:        $\mathcal{V} \leftarrow \{\mathcal{C}\}$
8:
9:     **else if** $\text{LENGTH}(\pi_{1_2}) < \text{LENGTH}(\pi_{2_1})$ **then** ▷ $\text{END}(\pi_{1_2}) < \text{START}(\pi_{2_2})$
10:        $(\pi_{3_1}, \pi_{3_2}) \leftarrow$ split π_{2_1} at value $\text{END}(\pi_{1_2})$
11:        $\mathcal{C}_1 \leftarrow \{\{\pi_{1_2}, \pi_{3_1}\}, \{\pi_{2_2}\}, \{\Phi[\pi_{3_1}]\}\}$
12:        $\mathcal{C}_2 \leftarrow \{\{\pi_{3_2}\}, \{\Phi[\pi_{3_2}]\}\}$
13:        $\mathcal{V} \leftarrow \{\mathcal{C}_1, \mathcal{C}_2\}$
14:
15:     **else**
16:        $s \leftarrow 0$
17:        **while** $\text{LENGTH}(\pi_{1_1}) < T_{len}$ **do** ▷ determine necessary index shift
18:           $s \leftarrow$ number of indices in π_{2_1}
19:           $(\pi_{1_1}, \pi_{1_2}) \leftarrow$ split π_1 at value $\text{END}(\pi_{2_1})$
20:           $(\pi_{2_1}, \pi_{2_2}) \leftarrow (\Phi[\pi_{1_1}], \Phi[\pi_{1_2}])$
21:        **end while**
22:
23:        **if** $s > 0$ **then** ▷ redefine P, π_1, π_2
24:           $P \leftarrow (\ (\pi_1(1), \ldots, \pi_1(K-s)), (\pi_2(1+s), \ldots, \pi_2(K))\ )$
25:           $(\pi_{1_1}, \pi_{1_2}) \leftarrow$ split π_1 at value $\text{START}(\pi_2) - 1$
26:           $(\pi_{2_1}, \pi_{2_2}) \leftarrow (\Phi[\pi_{1_1}], \Phi[\pi_{1_2}])$
27:        **end if**
28:
29:        $\mathcal{C} \leftarrow \{\{\pi_{1_1}\}\}$ ▷ cluster of shorter elements
30:        **while** $\text{END}(\pi_{1_2}) \geq \text{END}(\pi_{2_1})$ **do**
31:           $(\pi_{1_1}, \pi_{1_2}) \leftarrow$ split π_{1_2} at value $\text{END}(\pi_{2_1})$
32:           $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{\pi_{1_1}, \pi_{2_1}\}\}$
33:           $(\pi_{2_1}, \pi_{2_2}) \leftarrow (\Phi[\pi_{1_1}], \Phi[\pi_{1_2}])$
34:        **end while**
35:        $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{\pi_{2_1}\}\}$
36:        $\mathcal{V} \leftarrow \{\mathcal{C}\}$
37:     **end if**
38: **else** ▷ no overlap
39:     $\mathcal{C} \leftarrow \{\{\pi_1\}, \{\pi_2\}\}$
40:     $\mathcal{V} \leftarrow \{\mathcal{C}\}$
41: **end if**

$$
\begin{array}{c}
A_1 \quad\quad A_2 \\
\overline{} \\[-4pt]
\quad A_2 \quad\quad A_3 \\
\overline{} \\
\text{(a)}
\end{array}
\qquad
\begin{array}{c}
A_1 \;\; A_2 \;\; A_3 \;\; A_4 \;\; A_5 \\
\overline{} \\[-4pt]
A_2 \;\; A_3 \;\; A_4 \;\; A_5 \;\; A_6 \\
\overline{} \\
\text{(b)}
\end{array}
\qquad
\begin{array}{c}
A_1 A_2 \quad A_3 A_4 \\
\overline{} \\[-4pt]
A_3 A_4 \quad A_5 A_6 \\
\overline{} \\
\text{(c)}
\end{array}
$$
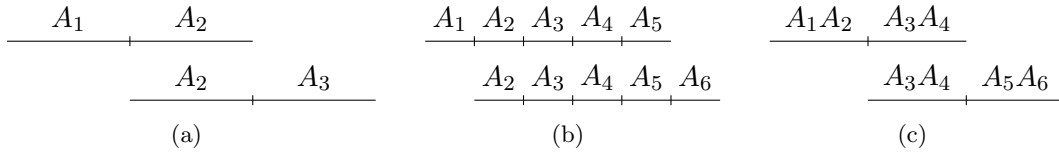
Figure 4.6: Resolving overlaps that span at least half of each projection with the ideal situation that $\varepsilon = \emptyset$. (a) In the simplest case, the overlap covers half of each projection. (b) If $\textsc{Length}(A_1) \geq T_{len}$, each A section is transformed into an individual cluster element. (c) If $\textsc{Length}(A_1) < T_{len}$, the original path first needs to be manipulated. Here, $\textsc{Length}(A_1 A_2) \geq T_{len}$.

The decision, whether the overlap A_2 encodes a valid musical section or not, is based solely on the minimum length threshold T_{len}. Thus, it has to be chosen with great care. More often than not, overlaps are caused by path extraction parameters that were chosen to be too tolerant towards musical variations. Setting T_{len} too low, in this case, may have bad effects on the overall clustering, as the resulting short and musically irrelevant cluster elements have to be integrated into a structural hierarchy which itself generally induces further disadvantageous transformations.

The third of the mentioned techniques is used if the overlap spans at least half of each projection. In this case, the path may be seen to encode numerous repetitions of a single shorter musical section A.

Here, the simplest example is projections that correspond to $A_1 A_2 \varepsilon$ and $A_2 A_3 \varepsilon$, where ε is a negligible portion at the start of an A section. If, additionally, $\varepsilon = \emptyset$, this corresponds to $\pi_{1_2} = \pi_{2_1}$ in the algorithm, or empty B sections in the previous cases. This situation is outlined in figure 4.6(a).

The larger the amount of overlap, the higher the number of A sections will be in each projection (cf. figure 4.6(b)). Here, each A section is created by consecutive split operations on the projection π_1, where new split values are provided by corresponding split results of projection π_2 (line 31ff.).

In extreme situations, the overlap may induce $\textsc{Length}(A_1) < T_{len}$, that is, A_1 cannot be represented by a valid cluster element. In this case, a transformation into a valid cluster is only possible if the original path is manipulated.

To this end, $k \in \mathbb{N}$ is sought, so that $A_1 \ldots A_k$ is of acceptable length (see line 17ff.). Then, π_2 is shortened at its start by the amount of feature indices spanned by the sections $A_2 \ldots A_k$ and π_1 is shortened at its end by a similar number of features indices (line 24). This leads to a new path with projections corresponding to $A_1 \ldots A_k \ldots A_K \varepsilon$ and $A_{k+1} \ldots A_{K+k} \varepsilon$ with a suitable $K \in \mathbb{N}$.

This manipulation is motivated by the fact that all of the A sections are seen to be mutually similar, so that the direct relation between the two sections A_l and A_{l+1} given by the original path P may be replaced by a relation between A_l and A_{l+k}, $1 \leq l \leq K$. Figure 4.6(c) illustrates this solution for the case $k = 2$.
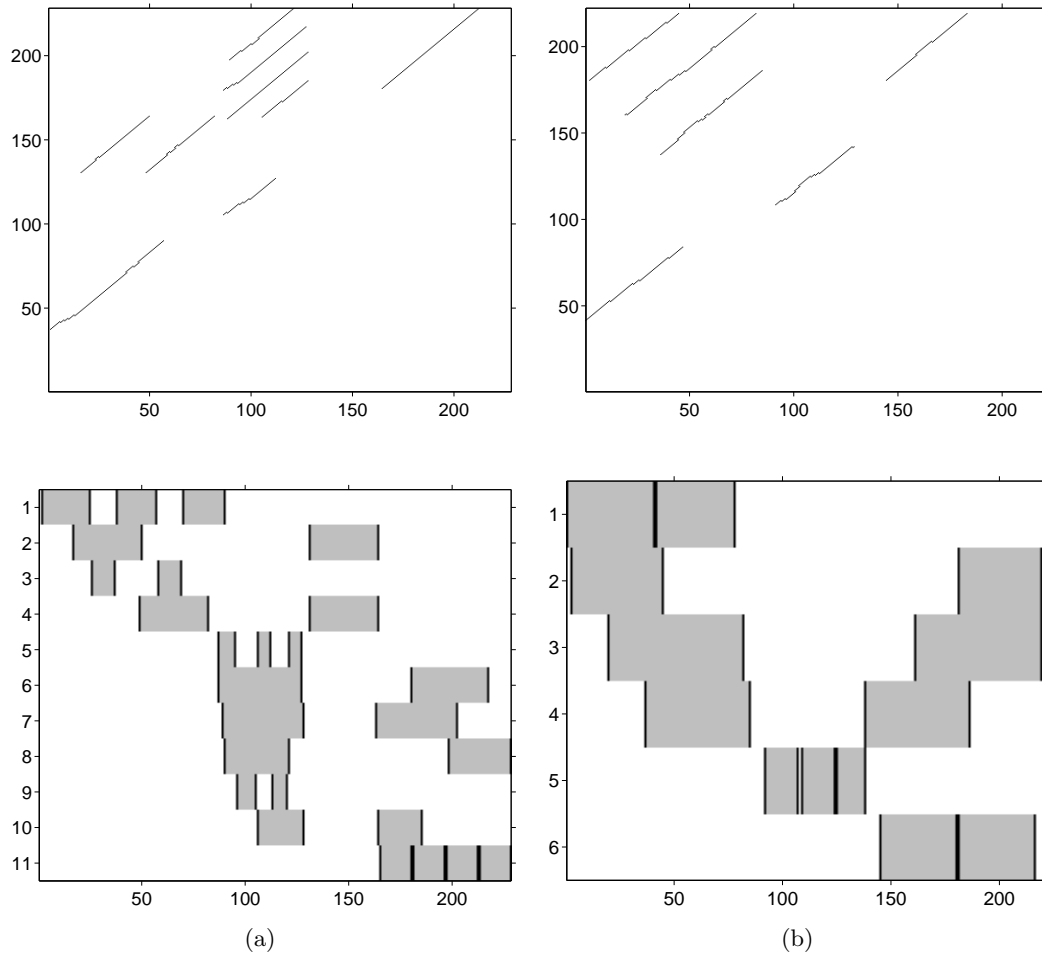
Figure 4.7: Originally extracted paths (top row) and their corresponding transformation into valid clusters (bottom row) for (a) *"Can't Take My Eyes Off You"* by *Barry Manilow* and (b) a *Chailly* interpretation of *"Waltz 2"* from the *"Suite for Variety Stage Orchestra"* by *Dimitri Shostakovich.* Clusters that contain more than two elements originate from overlapping paths. The results were obtained at a feature resolution of 1 Hz, using the clustering parameters $T_{id} = 3$ and $T_{len} = 6$.

However, this manipulation only works well if all A sections are of similar length. Otherwise, the temporal relation between the features in the resulting projections will not be accurate. Fortunately, this manipulation is only necessary if the original path is located near the diagonal of the underlying similarity matrix. Therefore, this situation may be circumvented by excluding a suitable area surrounding the diagonal from path extraction which is generally the case for the approach presented in section 2.3.

As stated at the beginning of this section, the outlined algorithm designates only one possibility to transform the given paths into valid clusters. One of its major advantages is that it generally conserves a large amount of information inherent in a path. Figure 4.7 illustrates the results of the algorithm on the basis of the two running examples introduced in chapter 2.

The split operation on projections, as seen in line 2 of the algorithm, warrants a more detailed examination. In order to split a projection π at a value v, first, a binary search for v in π is performed. The split itself is performed in a way that afterwards

$$\forall n \in \pi, n \le v : n \in \pi_1 \text{ and } \pi_2 = \pi \setminus \pi_1 \tag{4.14}$$

holds for the resulting subprojections π_1 and π_2. As described in section 4.1, splitting projection π induces the creation of two new subpaths which use the index sets that are created for π_1 and π_2. Therefore, the projections $\Phi[\pi_1]$ and $\Phi[\pi_2]$ are directly accessible as seen in line 3.

There is one problematic aspect to this split operation. Because of the step-size constraint for paths, it is possible that $v \notin \pi$, and consequently $v \notin \pi_1$. That is why, the last part of equation (4.13) only holds in an ideal scenario. This fact is especially delicate if the split operation is used to cut cluster elements to specific end points, for example, to enforce the structural hierarchy.

Furthermore, only a minimum number of length constraint tests is performed in algorithm 4.1. Thus, in an implementation all created clusters need to be checked for compliance with equation (4.6) before a further use.

The transformations outlined in this section are also typical for the following algorithms, where equivalent operations are defined for clusters and cluster elements.

4.3 A Sweep Approach to a Classical Clustering Problem

This section describes a first, highly simplified clustering method. Its main purpose is to introduce the sweep paradigm and to outline its applicability to this chapter's problem domain. The presented algorithm may be seen as a classical clustering procedure in $\mathbb{R}^2$, where points that lie close to one another are to be collected into *identity clusters*. In general, squares of a given side length are used to define an identity cluster. Then, the clustering algorithm has to find as small a number of disjoint and non-empty squares as possible that cover all the given points (cf. figure 4.8).

Figure 4.8: A sweep approach to find nearly identical points in $\mathbb{R}^2$. Only those points that are crossed in short succession by the sweep line are tried to be fitted into an identity cluster. Here, intermediate clustering results at time 30 (a) and 170 seconds (b) are illustrated.

In the more general case described in this section, a set of given cluster elements is transformed into subsets of $\mathbb{R}^2$ and the squares are of side length T_{id}, so that identified identity clusters characterise mergeable cluster elements (cf. section 4.1). To this end, the transformation of a cluster element E is given by

$$\begin{aligned}
[\text{MINIMUMSTART}(E) : \text{MAXIMUMSTART}(E)] \times & \quad (4.15) \\
[\text{MINIMUMEND}(E) : \text{MAXIMUMEND}(E)] \quad \subset \quad \mathbb{R}^2.
\end{aligned}$$

Instead of identity clusters, the desired outcome of this chapter are clusters that encode disjoint, musically similar sections as defined in section 4.1. Therefore, in order to avoid confusion, the result of this section's clustering will be referred to as identified mergeable cluster elements in the following.

The plane sweep paradigm was first introduced in [30]. An excellent application-oriented introduction is given in [9].

The sweep algorithms developed in this chapter share some basic concepts. First, they all are seen to employ a so-called *sweep line* that iterates over the set of cluster elements handling a number of discrete *events* for every element.

All operations relevant to a sweep algorithm are partitioned into such events. Thus, they must be chosen with regard to the desired application. Furthermore, all events are associated with an *event time*, that is, a specific, discrete point in time determining when an event should be processed by the algorithm. For example, the *start event* of a cluster element could be associated with the element's average start time (4.4).

In the following, the cluster element belonging to the event that is currently processed will be referred to as *current element*.

In a static scenario, all events are known beforehand and inserted into the *event data structure* $\mathcal{E}$ in the order of ascending event time. Thus, an iteration over the event structure corresponds to the motion of the sweep line across the plane from left to right, see figure 4.8. The processing of an event often depends on information about other cluster elements which are located in the proximity of the sweep line. A further data structure, the *status structure* $\mathcal{S}$, is used to store these *active* objects. Again, events are used to define the period of an element's activity, by means of adding it to or removing it from $\mathcal{S}$, respectively.

The main characteristic of sweep algorithms is that geometric properties inherent in the problem domain are exploited in order to confine comparisons between objects to relevant cases. For problems that might be modelled in a suitable fashion, the sweep paradigm often leads to time optimal algorithms. Here, the major challenge is to find an appropriate geometric model. This includes the definition of events together with a suitable processing of involved objects, specifying when objects are active, and how to efficiently access them.

The sweep clustering algorithm introduced now, is capable of identifying mergeable cluster elements in a given set of valid clusters. A naive approach to this problem generally involves the inspection of every pair of cluster elements. This number may be strongly reduced if the two data dimensions given by equation (4.15) are processed independently.

As outlined in section 4.1, two mergeable, valid cluster elements E_1 and E_2 are mainly constrained by equation (4.5). This implies that a conjunction of two separate tests concerning the start and end of two cluster elements may be employed in order to determine if they are mergeable.

Here, the dimension corresponding to the start of an element is used for the sweep iteration, and the second dimension is only accessed if the mergeability test in the first dimension is successful. More precisely, the aim is to define sweep events that implicitly guarantee that all cluster elements present in the status structure $\mathcal{S}$ fulfil this requirement with respect to a currently processed element. Thus, a sophisticated criterion characterising the start of two mergeable cluster elements is necessary.

Two mergeable, valid cluster elements E_1 and E_2 with

$$\text{MINIMUMSTART}(E_1) < \text{MINIMUMSTART}(E_2), \tag{4.16}$$

also need to fulfil

$$\text{MAXIMUMSTART}(E_2) \leq \text{MINIMUMSTART}(E_1) + T_{id}. \tag{4.17}$$

Geometrically, this simply means that the START points of E_1 and E_2 need to lie close to one another.

In the following algorithm, sweep events corresponding to $\text{MAXIMUMSTART}(E)$ and $\text{MINIMUMSTART}(E) + T_{id}$ are used for every cluster element E. These two events also designate the start and end of its activity, that is, the start and end of its presence in the status structure $\mathcal{S}$. This is well-defined, as according to equation (4.5),

$$\text{MAXIMUMSTART}(E) \leq \text{MINIMUMSTART}(E) + T_{id}.$$

Then, a cluster element E_2 fulfils (4.16) as well as (4.17) with respect to an arbitrary cluster element E_1, if and only if it is present[3] in the status structure $\mathcal{S}$ at time MINIMUMSTART(E_1) + T_{id}. In order to be mergeable, E_1 and E_2 further need to fulfil a similar constraint at their end. This conforms to the above-mentioned requirements and finally yields the sweep algorithm 4.2 on the next page.

As mergeable cluster elements encode nearly identical segments in the underlying piece of music, they should be merged into a single cluster element, which implies that their associated clusters must be merged as well (cf. section 4.1). These operations are directly triggered if mergeable cluster elements are identified in the sweep iteration (line 19ff.).

To this end, algorithm 4.3 on page 47 presents a suitable merge operation for mergeable, valid cluster elements. Here, special attention is required if both elements contain projections π_1 and π_2 that originate from the same path P (line 3). In general, π_1 and π_2 will not be identical, but, in order to regain a valid cluster element, only a single projection per P is allowed (cf. equation (4.7)).

Therefore, π_1 and π_2 are replaced by a single projection π which might, for example, use the average of both projections' start and end points (line 4). In order to maintain a connection between the cluster element resulting from the merge operation and the elements containing $\Phi[\pi_1]$ and $\Phi[\pi_2]$, both these projections need to be replaced by $\Phi[\pi]$ (cf. line 9 in the algorithm and equation (4.9)).

Then, both of these elements may be seen to represent an identical section and, therefore, should also be merged, which itself might result in further merge operations. However, in this case, special treatment may be necessary. As the respective elements may contain further projections, they generally do not need to be mergeable.

Nevertheless, these problems lie outside the scope of this section. In any case, the clustering algorithm should detect all mergeable cluster elements, so that it is sufficient to trigger all merge operations from there. The only facility necessary, in this case, is a means to associate a single projection with multiple cluster elements.

In contrast to this, it is relatively easy to implement a merge operation for clusters. Here, the respective sets of cluster elements are simply joined. Consequently, after merging two cluster elements, it is guaranteed that all elements containing the projections of a specific subpath belong to the same cluster.

Given N valid cluster elements as input for the clustering algorithm, a run time in $O(N \log N + NM)$ is obtained, where M denotes an upper bound for the number of elements E_2 fulfilling (4.16) and (4.17) with respect to an arbitrary cluster element E_1. Here, the first term in the addition corresponds to the sort operation which is necessary for the set of $2N$ sweep events, and the second term estimates the time required to process these events in the sweep iteration.

In the worst case, this also yields a run time in $O(N^2)$ which was previously considered unacceptable. But, in contrast to a naive approach, this bound is only

---

[3]Note that MINIMUMSTART(E_1) + T_{id} < MINIMUMSTART(E_2) + T_{id} ⇔ MINIMUMSTART(E_1) < MINIMUMSTART(E_2).

Algorithm 4.2 Simple sweep clustering

Require: A set of clusters $\mathcal{V} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ with $\mathcal{C}_l = \left\{ E_{l_1}, \ldots, E_{l_{K_l}} \right\}$, $1 \leq l \leq L$.
Ensure: The set of clusters $\mathcal{V}$, now, free of mergeable cluster elements.

1: ▷ Initialisation
2: $\mathcal{E} \leftarrow$ sorted set of MAXIMUMSTART and MINIMUMSTART $+ T_{id}$ events for given cluster elements
3: $\mathcal{S} \leftarrow \{\}$
4:
5: **for all** $e \in \mathcal{E}$ **do**   ▷ Sweep
6:         $E_{cur} \leftarrow$ element associated with e
7:
8:         **switch** type of e
9:
10:            **case** MAXIMUMSTART
11:                $\mathcal{S} \leftarrow \mathcal{S} \cup \{E_{cur}\}$
12:            **end case**
13:
14:            **case** MINIMUMSTART $+ T_{id}$
15:                $\mathcal{S} \leftarrow \mathcal{S} \setminus \{E_{cur}\}$
16:                $\mathcal{C}_{cur} \leftarrow$ cluster associated with E_{cur}
17:
18:                **for all** $E \in \mathcal{S}$ **do**
19:                    **if** E and E_{cur} are mergeable **then**
20:                        $\mathcal{C} \leftarrow$ cluster associated with E
21:                        **if** $\mathcal{C} \neq \mathcal{C}_{cur}$ **then**
22:                            $\mathcal{V} \leftarrow \mathcal{V} \setminus \{\mathcal{C}\}$
23:                            $\mathcal{C}_{cur} \leftarrow$ MERGE($\mathcal{C}_{cur}, \mathcal{C}$)
24:                        **end if**
25:
26:                        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{E\}$
27:                        $E_{cur} \leftarrow$ MERGE(E_{cur}, E)
28:                    **end if**
29:                **end for**
30:            **end case**
31:
32:        **end switch**
33:
34: **end for**

Algorithm 4.3 MERGE(E_1, E_2)

Require: Mergeable valid cluster elements E_1 and E_2.

Ensure: A valid cluster element containing the information inherent in both elements.

1: **for all** $\pi_1 \in E_1$ **do**

2:

3:     **if** $\exists \pi_2 \in E_2 \; \exists P \in \mathcal{P} : \pi_1, \pi_2 \in \Pi(P) \wedge \pi_1 \neq \pi_2$ **then**

4:        $\pi \leftarrow$ merge π_1 and π_2

5:        $E_2 \leftarrow (E_2 \setminus \{\pi_2\}) \cup \{\pi\}$

6:

7:        **for all** $i \in \{1, 2\}$ **do**

8:           $E \leftarrow$ element associated with $\Phi[\pi_i]$

9:           $E \leftarrow (E \setminus \{\Phi[\pi_i]\}) \cup \{\Phi[\pi]\}$

10:        **end for**

11:     **else**

12:        $E_2 \leftarrow E_2 \cup \{\pi_1\}$

13:     **end if**

14:

15: **end for**

16:

17: $E_1 \leftarrow \{\}$

18: remove E_1 from its associated cluster

19:

20: **return** E_2

---

reached if $M = N$. Thus, the run time of the sweep iteration depends on the configuration present in the input data which is, of course, preferable to the case where N^2 operations are always required.

In order to improve the run time even more, the status structure $\mathcal{S}$ may be sorted by the average end time of the cluster elements. Then, an iteration over $\mathcal{S}$ in the order of ascending end time could be stopped prematurely at the first element E with MINIMUMEND(E) > MINIMUMEND(E_{cur}) + T_{id}. This leads to a run time in $O(N \log N + K)$ if K pairs of mergeable cluster elements are present in the input and insert, remove, and search operations on $\mathcal{S}$ are bounded by $O(\log N)$.

An essential characteristic of the presented sweep clustering procedure is its first-fit heuristic which implies that mergeable cluster elements are identified in the order of the sweep iteration. With regard to the sweep paradigm, this is the most natural approach. However, this method generally does not identify the largest sets of mergeable cluster elements possible.

Finally, it has to be noted that the presented clustering algorithm on its own is nothing more than an educational example. The resulting clusters are, in general, not valid as they may contain overlapping cluster elements.
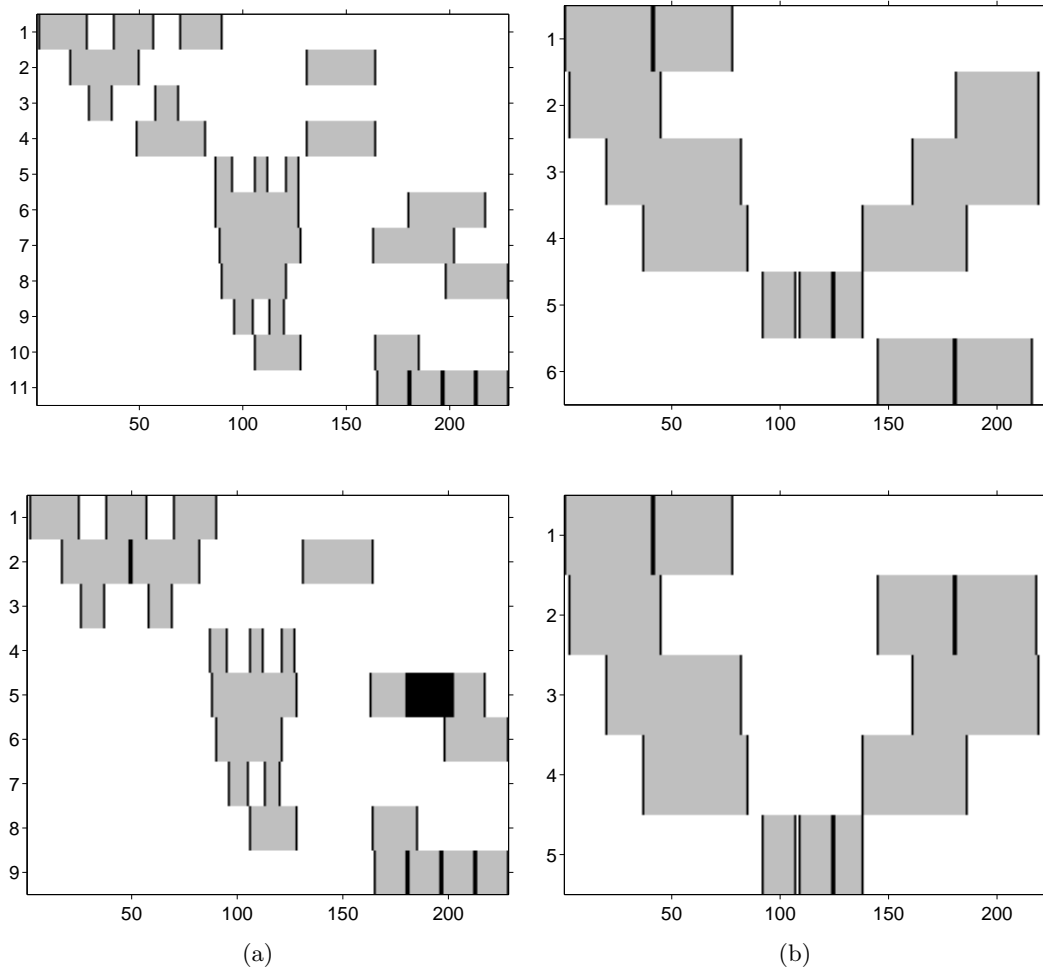
Figure 4.9: Valid clusters obtained from the path transformation (top row) and the result of the simple clustering (bottom row) for (a) the *Barry Manilow* and (b) the *Dimitri Shostakovich* example. The clustering parameters $T_{id} = 3$ and $T_{len} = 6$ were used.

4.4 Splitting a Valid Cluster

The desired result of this chapter is a valid cluster set that provides a suitable approximation of the grouping structure of the underlying musical piece. The clustering algorithms portrayed in this chapter work directly on a set of valid clusters, and a transformation of the original input data into this form is given in section 4.2.

However, a valid cluster set is subject to a number of constraints, for example equations (4.11) and (4.12). Thus, the clusters obtained from the input data generally need to be transformed heavily in order to obtain a corresponding valid cluster set. A basic approach supporting most of these transformations is to split a given cluster into two disjoint subclusters which is discussed in this section.

The source of a split operation on a cluster is, in general, a cluster element E that needs to be split into two disjoint parts in order to fit into a hierarchical structure. In order to integrate the result of this operation into a set of valid clusters, it is required by equation (4.9) that all of the cluster's elements are split accordingly. This effectively corresponds to a split operation on the cluster itself. To this end, the point-wise temporal relations between the elements of a cluster need to be exploited. A suitable approach to implement this operation is outlined in algorithm 4.4.

As stated in section 4.1, the temporal point-wise relation between cluster elements is based on the projections they contain (cf. line 9ff.). Consequently, every cluster element E that is split is at least connected to another element E_{asc} in the same cluster whose split value is determined accordingly. This induces a traversal over all elements in the given cluster.

In the algorithm, the split operation for every such element E_{asc} at value $\Phi(v)$ is queued into the data structure $\mathcal{Q}$ and, a corresponding entry in the set $\mathcal{T}$ is made to guarantee that this happens only once. In graph theory, this corresponds to the creation of a minimum spanning tree for the connected graph corresponding to the cluster.

The split algorithm may be seen to employ a first-fit heuristic, as the split value for all elements except for the one provided as input is determined on the basis of the first suitable projection that is split, respectively (cf. line 14ff.). In fact, this helps to circumvent inconsistencies inherent in the original valid path set $\mathcal{P}$.

Regarding the parallel to a connected graph, the traversal algorithm generally has more than one possibility to get from an element E_1 to element E_2 in the given cluster. This implies that a split value in E_1 may entail different split values in E_2. The reason for this is that the underlying set of paths and their corresponding maps Φ, in general, do not contain consistent information. As a heuristic solution, the presented algorithm ensures that a shortest possible route is taken from the cluster element given as input to every other element in the cluster. This implies that a minimum number of evaluations of the different Φ incarnations is necessary.

The split operation on projections as seen in line 10 of the algorithm was already discussed in section 4.2. Similar to algorithm 4.3, the connection between elements in the original cluster needs to be recreated in both clusters resulting from the split operation. In the implementation of the presented algorithm, two subpaths are

---

Algorithm 4.4 SPLITVALIDCLUSTER($\mathcal{C}, E, v$)

Require: Valid cluster element E of cluster $\mathcal{C}$ and split value v.
Ensure: $\mathcal{C}$ split at value v in E into clusters $\mathcal{C}_1$ and $\mathcal{C}_2$.

1: $\mathcal{C}_1 \leftarrow \mathcal{C}_2 \leftarrow \{\}$
2: $\mathcal{Q} \leftarrow \{(E, v)\}$ ▷ queue data structure
3: $\mathcal{T} \leftarrow \{E\}$ ▷ traversed elements
4:
5: **while** $\mathcal{Q} \neq \{\}$ **do**
6:   $E_1 \leftarrow E_2 \leftarrow \{\}$
7:   $(E, v) \leftarrow$ DEQUEUE($\mathcal{Q}$)
8:
9:   **for all** $\pi \in E$ **do**
10:    $(\pi_1, \pi_2) \leftarrow$ split π at value v
11:    $E_1 \leftarrow E_1 \cup \{\pi_1\}$
12:    $E_2 \leftarrow E_2 \cup \{\pi_2\}$
13:
14:    $E_{asc} \leftarrow$ element associated with $\Phi[\pi]$
15:    **if** $E_{asc} \notin \mathcal{T}$ **then**
16:     ENQUEUE($\mathcal{Q}, (E_{asc}, \Phi(v))$)
17:     $\mathcal{T} \leftarrow \mathcal{T} \cup \{E_{asc}\}$
18:    **end if**
19:   **end for**
20:
21:   $\mathcal{C}_1 \leftarrow \mathcal{C}_1 \cup \{E_1\}$
22:   $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup \{E_2\}$
23: **end while**
24:
25: **return** $(\mathcal{C}_1, \mathcal{C}_2)$

---

created instead of simple projections when π is split (cf. line 10), and instead of splitting $\Phi[\pi]$, the corresponding projections of the previously created subpaths are used if the respective element is traversed.

Furthermore, it is desirable that splitting a cluster element at value v results in a cluster element that has an average end point of v and the second element having its average start point at value $v + 1$. However, as shown at the end of section 4.2, this is generally not the case, as it is possible that v is not contained in every projection due to the step-size constraint for paths. Here, the next lower value will be used for splitting the affected projections. This fact demands attention, especially, if the split operation is employed to retain the hierarchical integrity of a valid cluster set.

If a suitable hash map is employed for the data structure $\mathcal{T}$, the time complexity of the outlined algorithm is bounded by $O(|\mathcal{C}|\,|\mathcal{P}|)$ as every cluster element in $\mathcal{C}$ may contain up to $|\mathcal{P}|$ projections (cf. equation 4.7).

Finally, it is useful to keep track of the relation between the cluster $\mathcal{C}$ and the clusters resulting from the split operation, for example, to avoid similar operations at a later point in a sweep clustering algorithm. In fact, this information is generally needed on an per element basis. Therefore, in the following sections, it is assumed that an operation

$$(E_1, E_2) \leftarrow \text{GetDescendants}(E)$$

is defined that returns the cluster elements of similar name in algorithm 4.4 for every element $E \in \mathcal{C}$ after a split operation was performed on cluster $\mathcal{C}$. For simplicity reasons, the implementation of this technicality is not shown explicitly in the presented algorithm.

4.5 Computing a Valid Cluster Set

This section presents a sweep clustering algorithm that is capable of transforming a given set of valid clusters into a valid cluster set. Therefore, it may be seen as an enhanced version of the clustering algorithm presented in section 4.3 which employs the splitting technique introduced in section 4.4 to enforce the requested hierarchical structure (cf. figure 4.4). Additionally, a refined sweep iteration is employed which allows dynamic changes to the set of clusters during its progression over time.

As mentioned earlier, computing a valid cluster set is generally a challenging task due to the tight constraints this implies. To this end, the solution presented in algorithm 4.5 on the next page looks surprisingly simple, but, as with most simple solutions, "the devil is in the details".

The employed sweep iteration is based on events corresponding to $\text{Start}(E)$ and $\text{End}(E)$ for a given cluster element E which also define the duration of its activity. Similar to algorithm 4.2, data structures $\mathcal{V}$, $\mathcal{S}$, and $\mathcal{E}$ are utilised to store the set of clusters, active elements and sweep events, respectively.

The sweep iteration across the plane is used to find constellations of cluster elements that conflict with valid cluster set constraints. In this case, the cluster of a conflicting element is split into two new, disjoint clusters that resolve the conflict. However, as is typical for the sweep paradigm, this only constitutes a local solution. In general, the newly created clusters still contain conflicting cluster elements that start later in time. Therefore, a *dynamisation* of the event data structure $\mathcal{E}$ is required, which allows the incorporation of new clusters into the current sweep iteration. Similarly, the originally conflicting cluster should be discarded.

The above-mentioned conflicts are identified during the processing of Start events, where newly created and discardable clusters are temporarily stored in the data structures $\mathcal{V}_{old}$ and $\mathcal{V}_{new}$ (line 11). These data structures are then used to update the set of clusters $\mathcal{V}$, the status structure $\mathcal{S}$, and the event structure $\mathcal{E}$ (line 13ff.).

Regarding the latter two, special care has to be taken. First, only those cluster elements in $\mathcal{V}_{new}$ which are active in the current sweep context should be added to $\mathcal{S}$.

Algorithm 4.5 Computing a valid cluster set

Require: A set of clusters $\mathcal{V} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ with $\mathcal{C}_l = \left\{E_{l_1}, \ldots, E_{l_{K_l}}\right\}$, $1 \leq l \leq L$.
Ensure: The valid cluster set $\mathcal{V}$.

1: ▷ Initialisation
2: $\mathcal{E} \leftarrow$ sorted set of START and END events for given cluster elements
3: $\mathcal{S} \leftarrow \{\}$
4:
5: **for all** $e \in \mathcal{E}$ **do**   ▷ Sweep
6:      $E_{cur} \leftarrow$ element associated with e
7:
8:      **switch** type of e
9:
10:          **case** START
11:              $(\mathcal{V}_{old}, \mathcal{V}_{new}) \leftarrow$ HANDLESTART(E_{cur})
12:
13:              $\mathcal{V} \leftarrow (\mathcal{V} \setminus \mathcal{V}_{old}) \cup \mathcal{V}_{new}$
14:              $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{\text{elements} \in \mathcal{V}_{old}\}) \cup \{\text{active elements} \in \mathcal{V}_{new}\}$
15:              $\mathcal{E} \leftarrow (\mathcal{E} \setminus \{\text{events} \in \mathcal{V}_{old}\}) \cup \{\text{relevant events} \in \mathcal{V}_{new}\}$
16:          **end case**
17:
18:          **case** END
19:              $\mathcal{S} \leftarrow \mathcal{S} \setminus \{E_{cur}\}$
20:          **end case**
21:
22:      **end switch**
23: **end for**

Second, it is advisable to enhance $\mathcal{E}$ only with events which are relevant at the current event time or later. Everything before that is no longer of interest to the sweep iteration.

Furthermore, according to the motivation at the end of section 4.3, the employed status structure $\mathcal{S}$ allows the traversal of the set of contained active elements in ascending or descending order of their average end points. To this end, the implementation of the algorithm employs a *doubly-linked* sorted data structure that supports efficient updates and queries.

The main part of the START event handling is outsourced into algorithm 4.6. Here, the principal data structures $\mathcal{S}$, $\mathcal{E}$, and $\mathcal{V}$ defined in the main algorithm are expected to be freely accessible for simplicity reasons. Only the cluster element E_{cur} is provided as input parameter in order to indicate whose START event is about to be processed.

All constellations that are incompatible with valid cluster set constraints at the given event time are resolved in this algorithm. In general, this concerns all elements

---

Algorithm 4.6 HANDLESTART(E_{cur})

1: $\mathcal{V}_{old} \leftarrow \mathcal{V}_{new} \leftarrow \{\}$
2: $\mathcal{C}_{cur} \leftarrow$ cluster associated with E_{cur}
3: $isUpdated \leftarrow$ FALSE
4:
5: **for all** $E \in \mathcal{S}$ in descending order of END(E) **do**
6:     **if** MAXIMUMEND(E_{cur}) $\leq$ MINIMUMEND(E) $+ T_{id}$ **then**   ▷ containment
7:         $E_{cut} \leftarrow$ CUTELEMENT(E_{cur}, E)
8:         **if** $E_{cut} = E$ **then**
9:             $\mathcal{V} \leftarrow \mathcal{V} \setminus \{C\}$
10:            $\mathcal{S} \leftarrow \mathcal{S} \setminus \{E\}$
11:            $\mathcal{E} \leftarrow \mathcal{E} \setminus \{\text{events of } E\}$
12:         **end if**
13:
14:         $\mathcal{E} \leftarrow \mathcal{E} \setminus \{\text{events of } E_{cur}\}$
15:         $\mathcal{C}_{cur} \leftarrow$ MERGE($\mathcal{C}_{cur}$, cluster associated with E_{cut})
16:         $E_{cur} \leftarrow$ MERGE(E_{cur}, E_{cut})
17:         $isUpdated \leftarrow$ TRUE
18:     **else**   ▷ overlap
19:         $\mathcal{C} \leftarrow$ cluster associated with E
20:         $(\mathcal{C}_1, \mathcal{C}_2) \leftarrow$ SPLIT($\mathcal{C}, E,$ START(E_{cur}) $- 1$)
21:         $(E_1, E_2) \leftarrow$ GETDESCENDANTS(E)
22:         $\mathcal{V}_{old} \leftarrow \mathcal{V}_{old} \cup \{\mathcal{C}\}$
23:         $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \cup \{\mathcal{C}_1\}$
24:
25:         **if** $\forall E \in \mathcal{C}_2 :$ LENGTH(E) $\geq T_{len}$ **then**   ▷ cf. (4.6)
26:             $(\mathcal{C}'_2, \mathcal{C}_{tmp}) \leftarrow$ SPLIT($\mathcal{C}_{cur}, E_{cur},$ END(E))
27:             $(E'_2, E_{tmp}) \leftarrow$ GETDESCENDANTS(E_{cur})
28:
29:             **if** $\forall E \in \mathcal{C}'_2 :$ LENGTH(E) $\geq T_{len}$ **then**   ▷ cf. (4.6)
30:                 $\mathcal{C}_2 \leftarrow$ MERGE($\mathcal{C}_2, \mathcal{C}'_2$)
31:                 $E_2 \leftarrow$ MERGE(E_2, E'_2)
32:                 $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \cup$ CLEAN($\mathcal{C}_2$)
33:             **end if**
34:         **end if**
35:     **end if**
36: **end for**
37:
38: **if** $isUpdated$ **then**
39:     $\mathcal{V}_{old} \leftarrow \mathcal{V}_{old} \cup \{\mathcal{C}_{cur}\}$
40:     $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \cup$ CLEAN($\mathcal{C}_{cur}$)
41: **else**
42:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{E_{cur}\}$
43: **end if**
44:
45: **return** $(\mathcal{V}_{old}, \mathcal{V}_{new})$

E that have a non-empty intersection with E_{cur}, which is the case, if and only if E is contained in the status structure $\mathcal{S}$. This is evident, as elements are active between their START and END events and thus

$$\text{START}(E) \leq \text{START}(E_{cur}) \leq \text{END}(E), \tag{4.18}$$

while $\text{START}(E) < \text{END}(E)$ holds for the valid cluster element E.

The algorithm enforces the valid cluster set constraints given by equation (4.11) and (4.12). Two different types of conflicts between E_{cur} and an active element are possible. If E_{cur} is contained in element E this implies that E_{cur} must be enhanced with information from the valid path set $\mathcal{P}$ that it lacks compared to E, that is, for all $P \in \mathcal{P}$ where $\exists \pi \in E \; \nexists \pi' \in E_{cur} : \pi, \pi' \in \Pi(P)$, a suitable subprojection of π has to be added to E_{cur} (line 6ff.). Otherwise, both elements overlap, and the cluster containing E is split which results in two elements, each conforming to one of the equations cited above (line 18ff.).

With respect to the first case, it is assumed that E contains E_{cur}, if

$$\text{MAXIMUMEND}(E_{cur}) \leq \text{MINIMUMEND}(E) + T_{id}, \tag{4.19}$$

as this implies that both elements are at least mergeable at their end (line 6). This corresponds to equation (4.17) for the start of cluster elements. In order to achieve the compliance with equation (4.12) a suitably shortened copy of E is computed, which is then merged into E_{cur}.

The shortened copy is obtained using the function CUTELEMENT which is outlined in algorithm 4.7 on the facing page. Given two cluster elements E_1 and E_2, it computes a cluster element that is mergeable with E_1 while containing a relevant part of every projection in E_2.

Here, two aspects are especially notable. First, it is checked, whether the given element E_2 may directly be used for merging (line 1). This is generally desirable as it retains as much information as possible from the two elements. Second, if E_2 is not mergeable, a shortened copy is created on the basis of the split operation on clusters which was introduced in section 4.4 (line 6ff.). Generally speaking, element E_2 is cut at both ends in order to obtain an element whose START and END correspond to E_1.

Nevertheless, both alternatives imply that START and END of E_{cur} may change subtly after merging it with the resulting element (line 16 in HANDLESTART). The reason for this was already discussed at the end of sections 4.4 and 4.2. Additionally, this may be promoted by the rounding function that is used to compute START and END in equation (4.4).

This fact might at first seem problematic given that, in general, there are numerous elements in $\mathcal{S}$ which have to be compared to E_{cur}.

Regarding the end of E_{cur}, this is unsubstantiated, as always the most recent version is used to determine the type of conflict with an active element (line 6). In contrast to this, cluster elements are only implicitly compared at their start by means of the sweep iteration. As START events of cluster elements are processed

---

Algorithm 4.7 CUTELEMENT(E_1, E_2)

1: **if** E_1, E_2 are mergeable **then**
2:     **return** E_2
3: **end if**
4:
5: $\mathcal{C}_2 \leftarrow$ cluster associated with E_2
6: **if** START(E_2) < START(E_1) **then**
7:     $(\mathcal{C}_{tmp}, \mathcal{C}_2) \leftarrow$ SPLIT($\mathcal{C}_2, E_2,$ START(E_1) $- 1$)
8:     $(E_{tmp}, E_2) \leftarrow$ GETDESCENDANTS(E_2)
9: **end if**
10:
11: **if** END(E_2) > END(E_1) **then**
12:     $(\mathcal{C}_2, \mathcal{C}_{tmp}) \leftarrow$ SPLIT($\mathcal{C}_2, E_2,$ END(E_1))
13:     $(E_2, E_{tmp}) \leftarrow$ GETDESCENDANTS(E_2)
14: **end if**
15:
16: **return** E_2

---

in ascending order, the corresponding conflicts of an element E with START(E) < START(E_{cur}) are resolved, before processing the START event of E_{cur}. Furthermore, algorithm 4.3 guarantees that

$$\text{START}(\text{MERGE}(E_1, E_2)) \geq \min(\text{START}(E_1), \text{START}(E_2))$$

for two arbitrary valid cluster elements E_1 and E_2 which induces that $\forall E \in \mathcal{S}$: START(E_{cur}) $\geq$ START(E) even after possible merge operations. Because of these two facts, no new conflicts with non-active elements occur.

In the case that the conflict between E_{cur} and E is of the overlap type, the cluster associated with E is manipulated. To this end, the sort order inherent in the status structure $\mathcal{S}$ is an important factor. The iteration over the elements in $\mathcal{S}$ in descending order of their average end implies that all conflicts of this type are handled after the above-mentioned merge operations that might change E_{cur}. This fact is eminent, as, here, the manipulations to an active element's cluster are based on E_{cur}. To this end, cluster $\mathcal{C}$ of active element E is split in a way that the resulting element E_1 does not overlap with E_{cur} and E_2 is completely contained in it (line 20ff.).

The cluster associated with E_1 is directly added to $\mathcal{V}_{new}$. Corresponding to section 4.2, it is assumed that a test against the length constraint (4.6) is performed implicitly when trying to add newly created clusters to $\mathcal{V}$. In the case that a cluster fails this test, it simply should be discarded.

An explicit version of this test is performed for the cluster of element E_2 (line 25). If it fails, the overlap with E_{cur} is of negligible length, and the cluster is discarded. Hence, the performed manipulation corresponds to a shortening of cluster $\mathcal{C}$ at its end. The motivation behind this is similar to the one given in section 4.2 for the corresponding case of overlapping projections.

If the test is successful, a containment conflict is obtained which is similar to the one discussed above. This time, E_2 and its corresponding cluster $\mathcal{C}_2$ have to be enhanced with information from element E_{cur} and cluster $\mathcal{C}_{cur}$, respectively (line 26ff.).

It has to be noted that the resolution of overlap conflicts is based on a rather simple heuristic. The motivation to split clusters of active elements and not the one of E_{cur} is solely based on the fact that this fits best into the sweep iteration.

In order to complete the analysis of algorithm 4.6, the CLEAN operation on clusters needs to be discussed (lines 32 and 40).

Clusters obtained from previously described manipulations fulfil the valid cluster set constraints at least locally. The main reason for avoiding a direct inclusion into the set $\mathcal{V}$, is that they generally are not valid, that is, the constraint given by (4.10) is violated. In order to resolve overlap conflicts inherent in a single cluster, the CLEAN operation is used.

Similar to the algorithm presented in this section, it is implemented using the plane sweep paradigm, but its manipulations are based on the transformations described in section 4.2. This is due to the fact that, here, all involved elements are seen to be repetitions of the same musical section. In general, it is necessary to transform a given cluster into numerous clusters of shorter elements in order to resolve all conflicts. That is why a set of clusters is returned from CLEAN. As essential property this operation preserves the locally calculated hierarchical structure. This is evident, as all cluster elements that are created during the processing of a START event include at least all relevant information from their containing elements. Due to the similarity to techniques already described in this chapter, no algorithm is provided explicitly for the CLEAN operation.

The presented clustering algorithm successfully transforms a given set of valid clusters into a valid cluster set. As outlined in figure 4.10, it enforces the valid cluster set constraints locally in the proximity of the sweep line while the structure prior to this point is refined or enhanced with consistent information.

The final clustering result for the two running examples of this chapter is depicted in figure 4.11. This indicates that the algorithm is capable of computing an easily comprehensible structural overview of a musical piece. Still, some possible improvements are visible. On the one hand, the results contain some redundancy, that is, clusters whose information is completely described by another cluster. On the other hand, there are corresponding sequences of cluster elements only separated by very short gaps which may be seen as a sort of over-fragmentation. These problems are addressed in a post-processing step which is introduced in section 4.6.

The presented algorithm adopts best practices regarding the sweep paradigm as the two dimensions inherent in a cluster element are processed separately (cf. equation (4.15)). Corresponding to section 4.3, the employed START and END events and the processing in ascending order of the START event time implicitly confine the elements in the status structure to conflicting elements. Furthermore, the design of the status structure $\mathcal{S}$ allows the resolution of conflicts with active elements in order of their end points which implies that different conflict types are processed independently.

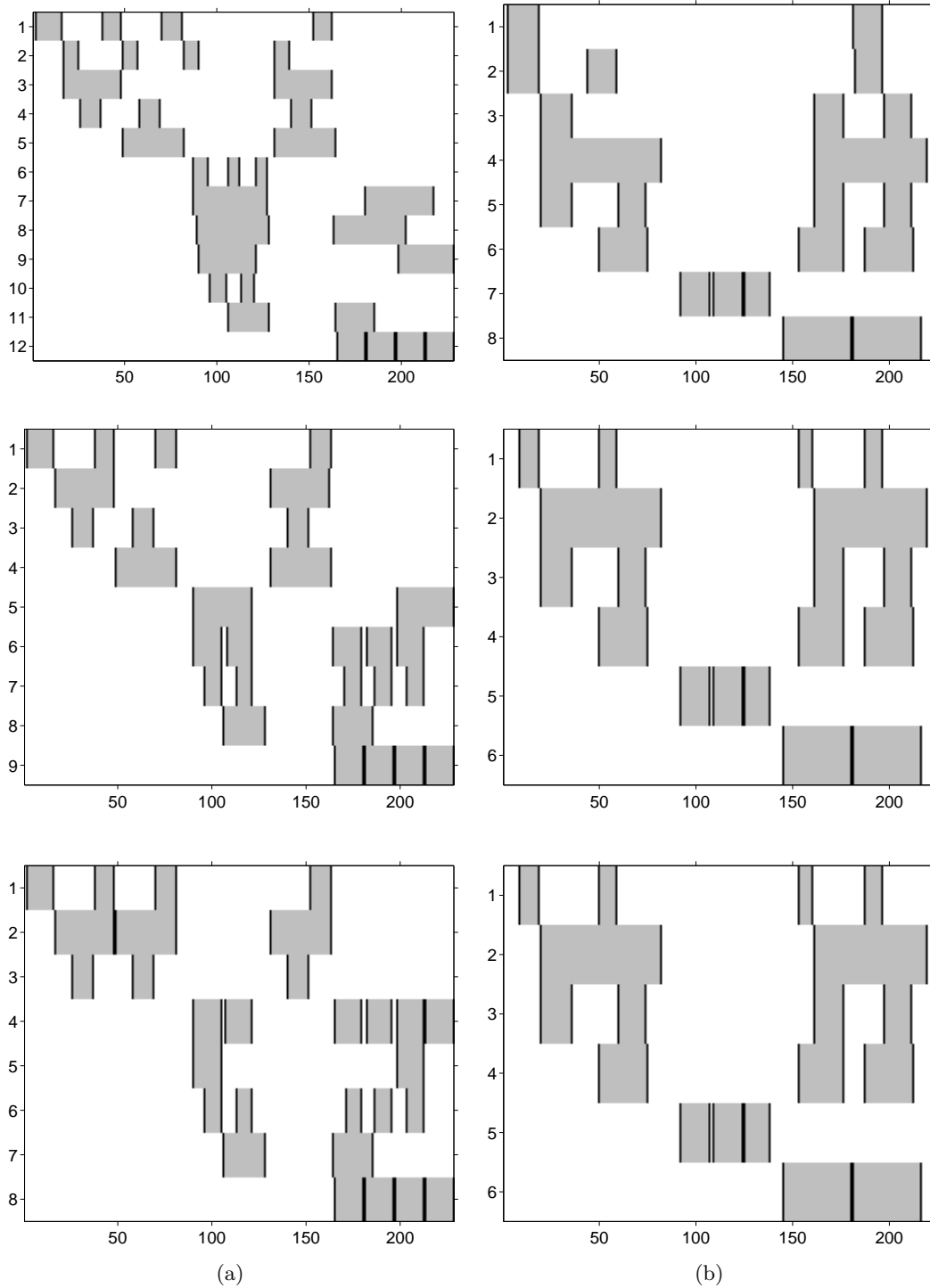Figure 4.10: Intermediate clustering results obtained at 50 (top row), 100 (middle), and 150 seconds (bottom) during the sweep iteration for (a) the *Barry Manilow* example and (b) the *Dimitri Shostakovich* example. It is easily visible that conflicts with valid cluster set constraints are resolved locally at the current event time while preserving and refining the hierarchical structure prior to this point.
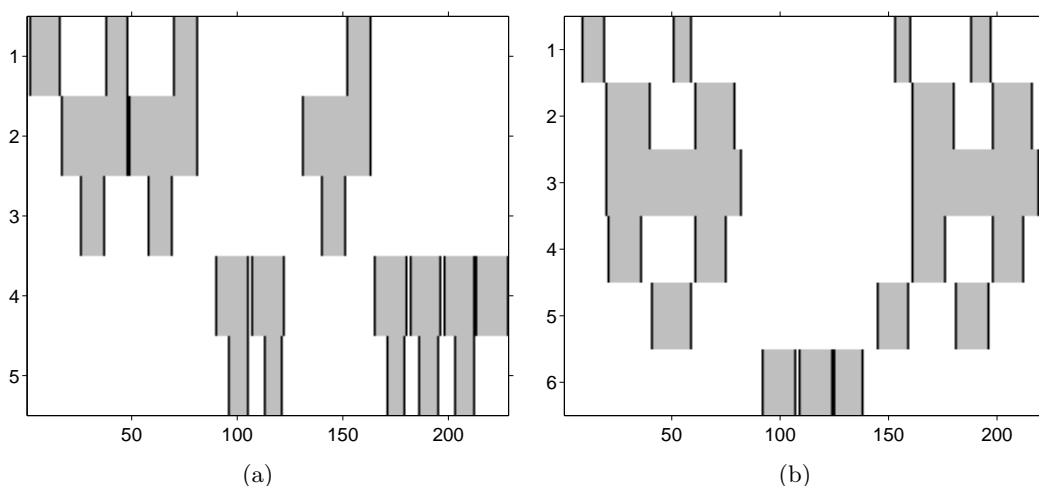
57

Figure 4.11: Valid cluster sets computed by the clustering algorithm for (a) the *Barry Manilow* and (b) the *Dimitri Shostakovich* example. With respect to the valid clusters depicted in figure 4.7, the presented structural overview is more easily comprehensible while, still, some redundancy and a possible over-fragmentation are present.

Finally, the time complexity of the given algorithm depends on several factors. Given a suitable implementation of the dynamic event data structure $\mathcal{E}$, it may be bounded by $O(N \log N + NM)$, where M is an upper bound for the time complexity of the HANDLESTART algorithm. This algorithm has major impact on the overall run time, and depends itself on the complexity of the SPLITVALIDCLUSTER and CLEAN algorithms. While the time complexity of the former was examined at the end of section 4.4, the CLEAN operation was said to be virtually similar to this section's sweep algorithm, only that it operates solely on a single cluster. Overall, it seems to be difficult to avoid at least a quadratic time complexity.

4.6 Further Improvements

Although the clustering algorithm outlined in section 4.5 computes valid cluster sets, in general, the obtained representation of the grouping structure of a musical piece is not as efficient as possible. On the one hand, some of the resulting information may be redundant, and, on the other hand, a subtle fragmentation between clusters may be seen as the consequence of computational deficiencies rather than being inherent in the musical material. This section introduces two further clustering algorithms which were developed to alleviate this situation.

Regarding the first aspect, all clusters whose information may be *completely described* by another cluster should be discarded. To this end, two different scenarios are possible.
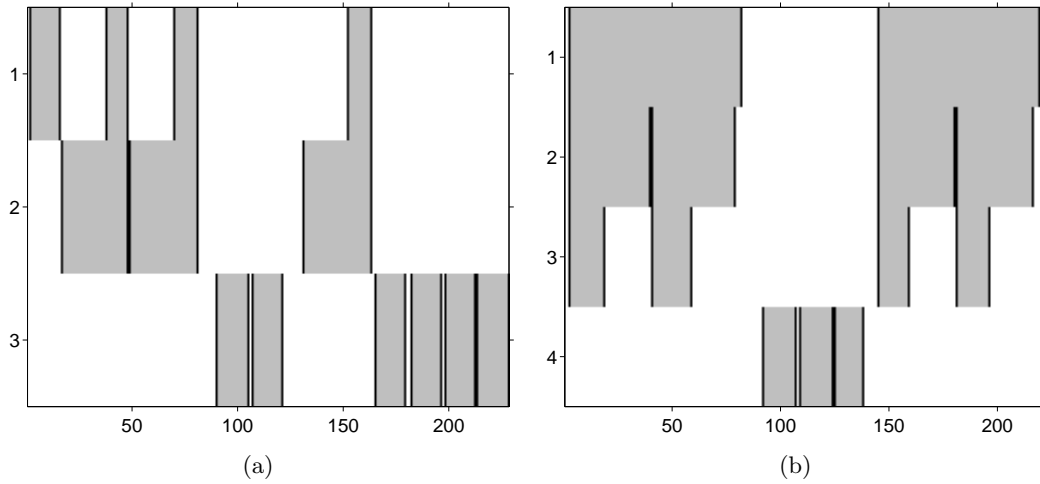
Figure 4.12: Improved clustering results for (a) the *Barry Manilow* and (b) the *Dimitri Shostakovich* example. Most of the artifacts that were present in the regular valid cluster sets are resolved. Only the third cluster in the *Dimitri Shostakovich* example is redundant. Furthermore, for both pieces the relevant musical sections were correctly identified. From top to bottom, the clusters obtained for the *Barry Manilow* example correspond to the intro, verse, and chorus / outro sections. The second cluster of the *Dimitri Shostakovich* example correctly reflects the repeating *A* section of this musical piece.

First, if there are two clusters $\mathcal{C}_1 = \{E_{1_1}, \ldots, E_{1_K}\}$ and $\mathcal{C}_2 = \{E_{2_1}, \ldots, E_{2_L}\}$, $K \leq L$, and

$$\exists m : [1 : K] \to [1 : L], injective : \alpha_{1_k} \cap \alpha_{2_{m(k)}} = \alpha_{1_k}, \qquad (4.20)$$

then, all information provided by cluster $\mathcal{C}_1$ is already present in $\mathcal{C}_2$.

In a more specific situation, each E_{1_k} has an inverse relation to $E_{2_{m(k)}}$, that is, $\alpha_{1_k} \cap \alpha_{2_{m(k)}} = \alpha_{2_{m(k)}}$, and the elements contained in $\mathcal{C}_1$ are only *slightly larger* than their counterparts in $\mathcal{C}_2$. In this case, the additional information provided by $\mathcal{C}_1$ is merely negligible. Consequently, one may argue that $\mathcal{C}_1$ should be discarded in order to obtain a cleaner structural overview.

This may be seen as a softening of the identity constraint given by equation (4.5). To this end, threshold T_{id} could be re-used for the definition: E_{1_k} is slightly larger than $E_{2_{m(k)}}$, if and only if $\alpha_{1_k} \cap \alpha_{2_{m(k)}} = \alpha_{2_{m(k)}}$, and

$$\text{START}(E_{2_{m(k)}}) \leq \text{START}(E_{1_k}) + T_{id} \wedge \text{END}(E_{1_k}) \leq \text{END}(E_{2_{m(k)}}) + T_{id}. \qquad (4.21)$$

The main challenge of this approach is to find an efficient implementation that avoids the discarding of both versions of a cluster that have a one-to-one relation in the last-mentioned scenario.

With regard to the fragmentation problem, it may be assumed that the elements in cluster $\mathcal{C}_1$ are *contiguously followed* by elements in cluster $\mathcal{C}_2$, if and only if

$$\forall E_1 \in \mathcal{C}_1 \, \exists E_2 \in \mathcal{C}_2 \, : \, \text{MINIMUMSTART}(E_2) - \text{MAXIMUMEND}(E_1) \leq T_{id}.$$

In this case, a similarity cluster consisting of the segments $[\text{START}(E_1) : \text{END}(E_2)]$ may be seen as a valid replacement for $\mathcal{C}_1$. If the relation between the elements in $\mathcal{C}_1$ and $\mathcal{C}_2$ additionally exists on a one-to-one basis, $\mathcal{C}_2$ should be discarded too.

Of course, this transformation should only be employed in situations where a valid cluster set can be regained. Furthermore, it should be directly applied in a way that the remaining set of clusters is free of contiguously followed clusters.

Both of these clustering approaches were implemented in a similar manner to the sweep algorithm presented in section 4.5. Due to this fact, corresponding pseudo-code algorithms are not provided.

The serial combination of the three clustering algorithms constitutes the final algorithmic result of this chapter (cf. figure 4.4). As may be seen in figure 4.12, an easily comprehensible overview of the hierarchical structure inherent in a musical piece may be obtained using this approach.

4.7 Implementation Details

All of the above-mentioned algorithms were implemented using the JAVA programming language. First, a framework of classes was developed that is capable of representing the objects introduced in section 4.1, such as paths, subpaths, projections, cluster elements, and clusters, together with elementary manipulation operations like splitting and merging. On this basis, each of the presented sweep clustering algorithms was implemented in a separate class. This allows, for example, nearly arbitrary serial combinations of the clustering algorithms.

JAVA was chosen as implementation language, as it offers some important advantages over other alternatives. First, as all of the previous analysis steps which compute the valid path set $\mathcal{P}$ are based on Matlab, it was attempted to also employ the Matlab language for this clustering approach. This soon proved to be an infeasible solution, as Matlab generally is not suitable for the type of data manipulations which are required to efficiently implement the various types of objects and data structures mentioned above. However, Matlab features a tight integration of JAVA. In JAVA all sorts of pre-built data structures are directly available from the accompanying class library, which allowed an efficient and more meaningful object-oriented modelling of the problem domain.

This resulted in a framework of around twenty JAVA classes which are bundled into a single JAVA archive for inclusion in Matlab. All clustering classes employ a common interface allowing the formation of arbitrary serial combinations of clusters and a unified input and output. Moreover, this generally allows an easy integration into other audio structure analysis systems which need not be based on Matlab.

4.8 Final Notes and Future Work

The methodologies presented in this chapter are all referred to as clustering techniques. To this end, clustering applications in other sciences were studied with the aim of finding similar approaches.

According to [15] and [33], the clustering method developed in this chapter may be described as transitive, agglomerative clustering. In order to take advantage of these techniques, an exact representation of this chapter's problem domain in the terms of other clustering approaches has yet to be developed. The most challenging aspect which has to be modelled is the desired consistency in the output objects as well as the tolerance towards the threshold T_{id} described by equations (4.12) and (4.5), respectively. Corresponding to section 4.1, this model surely needs to based on the point-wise temporal relations between the projections inherent in a path.

The exemplary results obtained in this chapter indicate that the presented formalisation of the clustering problem and its desired outcome (cf. figure 4.1) allows the modelling of an approximation of the hierarchical structure inherent in a musical piece. Moreover, the introduced framework of clustering algorithms (cf. figure 4.4) is capable of computing a corresponding result, generally providing a compact and easily comprehensible representation of the structure.

To this end, one of the major design principles is to preserve as much information as possible from the set of paths given as input to the clustering. Another important aspect is the utilisation of the plane sweep paradigm which minimises the necessary number of comparisons between the projections of the given paths. This allows an efficient implementation which is one of the cornerstones facilitating the data transformations that are required to obtain a hierarchical structure.

Apart from these qualities, the developed approach also features an easily understandable parameter set whose main purpose is to specify the expected amount of inconsistencies in the input data. On the one hand, the threshold T_{id} defines a tolerance for identical segments (cf. equation (4.5)), and, on the other hand, the threshold T_{len} designates the minimum length of acceptable segments (cf. equation (4.6)).

Of course, there are still some open problems as well as further possible improvements. First, while as much information as possible is preserved from the given set of paths, this generally does not imply that always the most relevant data is kept.

With respect to this, a possible improvement is directly evident from the results obtained in section 3.4. As, in this case, the extraction quality at both ends of a path is assumed to be similar, it is no longer substantiated to shorten paths or cluster elements only at their end in order to resolve inconsistencies. On the contrary, this should be based on the quality of the involved path links. To this end, each path link needs to be enhanced with its corresponding similarity measure.

For example, the shortening of a path employed in algorithm 4.1 (line 6) could be implemented as outlined in algorithm 4.8. Here, START and END operations on paths are defined similarly to the respective operations for projections given in section 4.1.

Algorithm 4.8 Qualitative shortening of a path

1: $start \leftarrow 1$
2: $end \leftarrow K$
3:
4: **while** $K - end + start - 1 < \text{LENGTH}(\pi_{1_2})$ **do**
5:      **if** $\mathcal{S}(\text{END}(P)) < \mathcal{S}(\text{START}(P))$ **then**   ▷ evaluate similarity matrix
6:          $start \leftarrow start + 1$
7:      **else**
8:          $end \leftarrow end - 1$
9:      **end if**
10:     $P \leftarrow P|_{[start:end]}$ ▷ redefine P, π_1, π_2
11: **end while**
12:
13: $\mathcal{C} \leftarrow \{\{\pi_1\}, \{\pi_2\}\}$

Another problem accompanying the split operations described in section 4.2 and 4.4 is due to the step-size constraint for paths. Possible feature index gaps inherent in projections generally prevent exact split results although projections and cluster elements are seen to represent continuous segments in a musical piece. This may affect the run time of the clustering as more CLEAN operations may be necessary in order to obtain a consistent result. To this end, the definitions of START and END of cluster elements (cf. equation (4.4)) should be revised.

A more architectural problem is the complexity of the the developed hierarchy of definitions and data structures (section 4.1). This is caused by the fact that the desired hierarchical result needs to be encoded while still preserving the temporal point-wise relations inherent in the input data, as these are necessary for transformations that are invariant to tempo variations. Nevertheless, the above-mentioned qualitative shortening of overlapping paths or cluster elements is also based on this point-wise information.

Another relevant factor is the required tolerance towards inconsistencies in the given set of paths. A simplification of the formal model may be possible if inconsistencies in the input data are resolved before applying the clustering algorithm.

Furthermore, a continuous model of the problem domain might at least clarify the formalisation given in section 4.1.

A more general question is whether the first-fit heuristic induced by the sweep paradigm is always a good choice. This heuristic implies, for example, that the order of the average start points of cluster elements controls to a great extent the way in which conflicts are resolved.

Finally, there is one problem which is not solvable by improving the presented algorithms alone. As the whole analysis system is based on repetitions, it is not possible to identify musical structure which goes beyond that. Apart from heuristic approaches like the ones presented in [12] and [31], the metric structure of a musical piece could be analysed to further subdivide identified musical sections.

5 Evaluation

The techniques and algorithms that were introduced in chapters 3 and 4 were only evaluated on an exemplary basis. In order to gain more confidence in their performance, an automatic evaluation on a larger data set of musical pieces is needed.

In general, the evaluation of the results obtained from an automatic structural analysis of a musical piece depends on a *ground truth* reflecting the desired outcome. Therefore, manual annotations need to be provided for every piece of music in the chosen test data set. In this context, the manual annotation of a musical piece is defined as the process of labelling events or scenes with correct descriptions by an expert listener [13]. Unfortunately, many aspects in music may be described in several ways. For example, there may be several structural descriptions for a single musical piece each characterising a different level of abstraction.

Section 5.1 provides insight into these problems and introduces the annotation method used in this thesis. Correspondingly annotated test data sets are presented in section 5.2. In section 5.3, evaluation measures that are generally employed by other researchers are introduced and analysed with respect to their applicability to this thesis' problem domain. This results in two performance evaluation procedures which are capable to quantify the quality of a computed grouping structure of a musical piece (section 5.4). Finally, the evaluation results for the chosen test data sets are given and discussed in section 5.5.

5.1 Manual Music Annotation

Manual music annotation is a straining task. With regard to structural analysis, *similar* sections in a piece of music have to be identified. Here, every section may be described by its start and end time (in seconds) together with a label where similar labels are used to identify musically similar sections. To this end, the similarity of label strings still has to be defined. Table 5.1 on the next page gives a simple annotation example for the pop song *"Should I Stay or Should I Go"* by *The Clash* as used in [17].

It can be easily seen that this song consists of two repeating sections, namely verse and chorus, as well as an introductory section which is not repeated. Unfortunately, labelling is not always unambiguous [7, 25]. In general, a single manual annotation of a musical piece solely reflects one level of abstraction, eventually providing only a single label per instant of time. In fact, the same song might be annotated in several ways (see figure 5.1).

Sometimes it is not even clear whether two sections should be treated as being similar or not.

Table 5.1: Manual annotation of *"Should I Stay or Should I Go"* by *The Clash.*

Label	Start time	End time
intro	0.356009	15.166168
verse	15.166168	40.615873
verse	40.615873	66.448163
chorus	66.448163	89.742857
verse	89.742857	117.315374
verse_instru	117.315374	140.886531
chorus	140.886531	185.213968

For example, the section labelled "verse_instru" in the *"Should I Stay or Should I Go"* annotation is an instrumental solo interpretation of the verse and might as well be identified as an individual instrumental section (see figure 5.1(d)). Thus, the underscore could be used to delimit alternative section labels, which may all be allowed as possible solutions in the computed result [25].

Further problems might occur if repeating sections expose large musical variations such as the introduction of extra measures. Again, multiple annotation possibilities exist in this case, for example splitting sections into several sub-sections or treating sections as being similar nonetheless.

The clustering technique presented in chapter 4 computes a hierarchical structural overview of a musical piece. For a complete evaluation of these structures, manual annotations reflecting several levels of abstraction are necessary. This could also alleviate some of last-mentioned problems. However, manually annotating a number of musical pieces to the minutest detail would mean an enormous effort.

Often, annotations would either end up micro-sectioning a piece if finer musical variations are respected, or multiple alternative hierarchical structures would need to be accepted, and annotated, of course.

Nevertheless, many common variations mentioned at the beginning of this section may easily be computed from simple annotations reflecting only a single hierarchical level. Annotations available from other sources almost always are of this form too, as most applications only require a coarse structural overview.

For these reasons, the evaluation method used in this thesis is based on simple annotations, and common annotation variations need to be computed automatically during the evaluation process. Furthermore, the annotated sections reflect structuring schemes that are commonly found in literature on music analysis, and generally represent clearly comprehensible structural elements. Musical variations that are not sufficiently long to form independent sections are ignored, so that an annotated section generally captures between 10 to 30 seconds of music. To further simplify the annotation and evaluation process, alternative section labels, as suggested in [25], are not employed. To this end, sections are regarded being musically similar, if and only if their section labels are identical except for any numerical characters.

(a)	intro	verse	verse	chorus	verse	verse	chorus
(b)	intro	verse		chorus	verse		chorus
(c)	intro	vc			vc		
(d)	intro	verse	verse	chorus	verse	instru	chorus

Figure 5.1: Visualisation of alternative annotations for *"Should I Stay or Should I Go"* by *The Clash*. (a) Annotation from table 5.1 where "verse_instru" is interpreted as a regular verse section. (b) Concatenation of two consecutive verse sections. (c) Concatenation of two verses and a chorus. (d) Alternative label for the instrumental interpretation of the last verse.

Due to these simplifications, slightly inferior results may be obtained as computed hierarchical structures may reflect details not available in annotations. Still, a satisfying approximation of the grouping structure of a musical piece should be able to explain the previously mentioned annotation variants to a large extent. In turn, this creates the opportunity to easily re-use the annotations that were produced for this thesis in other applications, as well as re-using third party annotations in this evaluation. Consequently, a performance comparison with other music structure analysis systems should be easier.

In order to produce manual annotations for this thesis, the freely available audio editor *Audacity*[1] was employed. Here, *text marker track*s offer the possibility to define and label musical sections. This annotation may then be exported to a text file. Most other popular audio editors should offer similar capabilities. For example, *Wavesurfer*[2] was used to create the test data set annotations in [17]. In order to make the annotation text files available to the music structure analysis system, corresponding import and export functions were developed in Matlab.

5.2 Test Data Sets

In music retrieval, an evaluation test data set consists of a number of digitally recorded musical pieces along with corresponding annotations. Due to the fact that most commercially released music may not be shared freely, there is no music retrieval test data set that has become as established as similar sets like for example TREC[3] in the text retrieval domain. At most, plain annotations are made available to the public and the respective audio recordings have to be tracked down individually by each researcher.

Here, the challenge is to obtain the exact version which is demanded by the annotation. One of the major disadvantages is, for example, that an audio track

---

[1] Audacity audio editor `http://audacity.sourceforge.net`
[2] WaveSurfer audio editor `http://www.speech.kth.se/wavesurfer/`
[3] Text REtrieval Conference (TREC) `http://trec.nist.gov`

that was ripped from a CD using different equipment mostly exhibits a variable amount of silence at its beginning [13].

In an attempt to relieve this situation the Japanese National Institute of Advanced Industrial Science and Technology (AIST) has built the RWC (Real World Computing) Music Database[4], which is probably the best known test data set for music retrieval. It consists of six collections including, among others, several pieces of popular, jazz, and classical music. Each piece of music is available as a digital audio recording along with a corresponding MIDI version and a lyrics text file if applicable. However, this database is only accessible after purchasing a research license. Therefore, evaluation results which are based on this database are not very helpful for people that are not directly involved into this research domain. Additionally, the popular music collection is strongly targeted towards Japanese popular music. Regarding this thesis, the most prevalent disadvantage is that further annotations as announced in [13] are not yet available, and that they will merely feature annotations of chorus sections.

An alternative may be the MPEG-7 test set developed by Geoffroy Peeters [26], which already provides a large collection of pop song annotations. However, this collection also suffers from the fact that it is exclusively targeted towards commercial music which cannot be shared along with the annotations. A further alternative overlooked in research until today might be the steadily growing amount of music released under a Creative Commons license[5]. As the smallest common denominator all available incarnations of this license allow the free distribution of the licensed work. The largest source of Creative Commons licensed music today is the Internet Archive[6] listing over 10.000 releases of so-called netlabels, that is, record labels that distribute their music primarily over the Internet. Because of its free nature, it is the rare exception that commercial music is made available under this license. Nonetheless, the amount of commercial quality netlabel music is steadily growing as the netlabel scene tries to establish itself as an alternative to traditional record labels. Data sets that are based on this license might therefore become a viable alternative as they circumvent the above mentioned problems.

The performance evaluation in this thesis is based on two test data sets. The SYNCHRONISATION data set contains 56 pieces of classical as well as pop and jazz music and was previously used in [22]. The annotations for this data set were created by the author of this thesis following the principles outlined in section 5.1.

Furthermore, another 60 musical pieces constitute the POP data set which is based on the annotations used in [17][7] incorporating 14 songs of the above-mentioned MPEG-7 test suite. While, generally adhering to the established annotation principles, this data set suffers from some inconveniences. First, there is no explicit definition available regarding the similarity of section labels. For example, it is not clear whether sections labelled with "verse", "verse_instru", "verse1" or, "verse2"

---

[4]RWC Music Database `http://staff.aist.go.jp/m.goto/RWC-MDB/`

[5]Creative Commons `http://creativecommons.org`

[6]Internet Archive `http://www.archive.org`

[7]Pop data set annotations `http://www.elec.qmul.ac.uk/digitalmusic/downloads/#segment`

Table 5.2: Statistics about the number of musical pieces and their annotated structure in both employed test data sets. The last three columns show the average number of annotated clusters per musical piece, the average number of segments in a cluster and the average length of an annotated segment (in seconds).

Data set	Pieces	Clusters	Segments	Length
Pop	60	2.90	3.14	17.36
Synchronisation	56	2.11	3.21	25.33

all should be expected to be musically similar. Furthermore, the analysis system described in the respective paper identifies sections on the basis of changing states so that it is not guaranteed that direct repetitions of a similar sections are annotated separately (cf. the chorus sections given in table 5.1). Nonetheless, the original annotations are used as is which possibly induces slightly worse evaluation results.

Table 5.2 provides some general statistics on the employed test data sets. The individual musical pieces contained in both sets are listed in appendix A and B.

5.3 Performance Measures

In order to compare computed analysis results to manual annotations, some kind of performance measure has to be employed. This measure needs to quantify the quality of the approximation of the annotated structure that is provided by the computed results. As the automatic music structure analysis system described in this thesis is based on repetitions, only repeated sections in a piece of music may be expected to be identified. Therefore, it is only measured how well annotated repeating sections are *explained* by the computed results. Here, for every set of annotated similar sections, a *corresponding* set of computed similar sections is identified, and its approximation performance is measured.

In music structure analysis, three different measures of goodness are widely used. First, musical structure may be evaluated on the basis of label sequences obtained from computed and annotated structures (see for example [6], [18], and [27]). Here, repeated similar sections are mapped to similar labels, and an edit distance is used to express the difference between both resulting label sequences. For example, an annotated label sequence of $ABAB$ and a computed structure of $ABBAB$ would lead to an edit distance of 1, because of an extra insertion of B in the second sequence. In order to use this method, a common level of abstraction must be found which allows a meaningful mapping between computed and annotated sections. In this thesis, this is especially critical, as a computed hierarchical structure must be compared to simple annotations (cf. section 5.1). As a further disadvantage it is not possible to quantify the approximation quality of computed sections solely on the basis of the edit distance.

Boundary measures are often used additionally in order to circumvent the last-mentioned problem. In its simplest form, statistics on distances between computed sections' end points and closest annotated end points are produced. In a more sophisticated approach, only those end points of matching sections that were identified during the edit distance calculation are compared. It should be obvious that both measures on their own generally do not allow a meaningful conclusion about the performance of a music structure analysis system.

A performance measure that is rather popular in information retrieval is the F-measure [32]. It is defined as the harmonic mean of precision and recall rate, where the precision rate may be seen as a measure for the exactness, and the recall rate as a measure for the completeness of the computed result. F-measure performance evaluations of music structure analysis systems were employed, for example, in [12], [31], and [18].

In its most basic form it may be used to measure the quality of the approximation of an annotated section α_a by a computed section α_c. Here, similar to section 2.4, segment $\alpha = [s : t]$ denotes a sequence of indices in the feature sequence of an analysed piece of music. In accordance with the definitions given in chapter 4,

$$\text{START}(\alpha) \quad := \quad s \tag{5.1}$$

$$\text{END}(\alpha) \quad := \quad t \tag{5.2}$$

$$\text{LENGTH}(\alpha) \quad := \quad t - s \tag{5.3}$$

will be used in the following. Then, for an annotated section α_a and a computed section α_c, precision and recall rates are defined as

$$\text{PRECISION}(\alpha_a, \alpha_c) := \frac{\text{LENGTH}(\alpha_a \cap \alpha_c)}{\text{LENGTH}(\alpha_c)} \tag{5.4}$$

and

$$\text{RECALL}(\alpha_a, \alpha_c) := \frac{\text{LENGTH}(\alpha_a \cap \alpha_c)}{\text{LENGTH}(\alpha_a)}, \tag{5.5}$$

respectively. Here, the intersection $\alpha_a \cap \alpha_c$ is said to be the *correctly detected* segment in the computed result. Finally, the F-measure is defined by

$$\text{F-MEASURE} := \frac{2 \cdot \text{RECALL} \cdot \text{PRECISION}}{\text{RECALL} + \text{PRECISION}}. \tag{5.6}$$

The presented concepts may be easily generalised to annotated and computed similarity clusters, that is sets of similar segments (cf. section 2.4). In this case, some kind of map between both sets of segments must be available, in order to decompose the evaluation into a corresponding set of segment evaluations.

A carefully chosen F-measure evaluation is capable of combining aspects of exactness and structural completeness found in both measures that were first introduced. Consequently, the performance evaluation in this thesis is based on this concept.

5.4 Performance Evaluation Procedure

In general, there is more structure inherent in a musical piece than is evident from a provided ground truth. In particular, this applies to the performance evaluation introduced in this chapter. The hierarchical structural overview that is computed by the clustering algorithm described in chapter 4 is meant to reveal the underlying musical structure to a large extent. Nevertheless, only simple annotations were established for the test data sets. In general, evaluating the quality of every computed cluster will therefore be unrealistic. However, as explained in section 5.1, computed clusters should be able to describe the provided manual annotations to a large extent.

The aim of this performance evaluation is to find the best matching set of computed clusters for every annotated cluster of a musical piece and to calculate an F-measure for this constellation. On this basis, it is possible to obtain an overall F-measure for a musical piece and for a whole data set of musical pieces.

As outlined at the end of section 5.3, the F-measure is computed on the basis of similarity cluster segments. However, the input for the evaluation is a computed valid cluster set as defined in section 4.1 and a manual annotation given as tabular data similar to the example in table 5.1. A suitable transformation of a computed cluster into a corresponding similarity cluster was already presented in section 4.1. The transformation of a given annotation is also quite simple. The provided time values are converted into corresponding feature indices, and the segments that constitute a similarity cluster are identified on the basis of the section labels. Here, two labels are said to denote similar sections if they are equal except for any numerical characters.

The evaluation procedure is outlined in algorithm 5.1 on page 71. Similar to the clustering algorithms that were presented in chapter 4, it is based on the plane sweep paradigm. Thus, the terms and definitions that were introduced there also apply in this case. The input of the algorithm are the sets $\mathcal{V}$ and $\mathcal{W}$ of computed and annotated clusters, respectively. Here, the most conspicuous difference to the original clustering algorithms is that the involved comparisons are solely based on similarity cluster segments. Therefore, the implementation does not depend on the rather complex cluster element data structure.

The results of the evaluation procedure are the two maps BestApproximation and F-Measure. For a given annotated cluster $\mathcal{A}$, BestApproximation($\mathcal{A}$) specifies the set of computed clusters that best match $\mathcal{A}$, and F-Measure($\mathcal{A}$) gives the corresponding F-measure. Here, the best matching set of computed clusters is the set that maximises the F-measure.

To this end, for two similarity clusters $\mathcal{A}$ and $\mathcal{C}$, precision and recall rate are defined as

$$\text{Precision}(\mathcal{A}, \mathcal{C}) := \frac{\text{Length}(\mathcal{A} \cap \mathcal{C})}{\text{Length}(\mathcal{C})} \qquad (5.7)$$

and

$$\text{RECALL}(\mathcal{A}, \mathcal{C}) := \frac{\text{LENGTH}(\mathcal{A} \cap \mathcal{C})}{\text{LENGTH}(\mathcal{A})}, \qquad (5.8)$$

respectively. In this case, for an arbitrary similarity cluster $\mathcal{C}$,

$$\text{LENGTH}(\mathcal{C}) := \sum_{\alpha \in \mathcal{C}} \text{LENGTH}(\alpha) \qquad (5.9)$$

yields a reasonable result, as the segments in the computed clusters and annotated clusters are all disjoint. The first assertion follows directly from equation (4.10), and the second holds due to the fact that only simple annotations are used.

This leaves the question, how $\text{LENGTH}(\mathcal{A} \cap \mathcal{C})$ should be defined. In other words, it has to be specified what *correctly detected* means given the two similarity clusters $\mathcal{A}$ and $\mathcal{C}$. Of course, this definition should be based on the segments that are contained in $\mathcal{A}$ and $\mathcal{C}$. To this end,

$$\text{LENGTH}(\mathcal{A} \cap \mathcal{C}) := \sum_{\alpha_a \in \mathcal{A}} \sum_{\alpha_c \in \mathcal{C}} \text{LENGTH}(\alpha_a \cap \alpha_c)$$

would be the simplest solution. However, in this case, similarity cluster $\mathcal{C}$ does not really need to approximate $\mathcal{A}$ to yield a high F-measure. For example, a single segment $\alpha_c \in \mathcal{C}$ could have a non-empty intersection with multiple segments in $\mathcal{A}$. As the annotation of a musical piece should be reflected by the computed analysis result, the segmentation given by cluster $\mathcal{C}$ should be at least as detailed as the one given by $\mathcal{A}$. Therefore, in the outlined example, only the intersection with a single annotated segment should be included in the F-measure calculation. Intuitively, the annotated segment that has the maximum intersection with α_c is the segment that is approximated by α_c. This leads to the following equation

$$\text{LENGTH}(\mathcal{A} \cap \mathcal{C}) := \sum_{\alpha_c \in \mathcal{C}} \max_{\alpha_a \in \mathcal{A}} \left(\text{LENGTH}(\alpha_a \cap \alpha_c) \right). \qquad (5.10)$$

From equation (5.10) follows that

$$\text{F-MEASURE}(\mathcal{A}, \mathcal{C}) > 0 \Leftrightarrow \exists \alpha_a \in \mathcal{A} \; \exists \alpha_c \in \mathcal{C} \, : \, \alpha_a \cap \alpha_c \neq \emptyset. \qquad (5.11)$$

In a naive approach, each $\alpha_a \in \mathcal{A}$ needs to be compared to every computed segment in order to determine the computed cluster $\mathcal{C}$ maximising $\text{F-MEASURE}(\mathcal{A}, \mathcal{C})$. In algorithm 5.1, the sweep iteration is used to solve this problem more efficiently.

To this end, a map $\mathcal{H} : \{\alpha \in \mathcal{C} : \mathcal{C} \in \mathcal{V}\} \times \mathcal{W} \to \{\alpha \in \mathcal{A} : \mathcal{A} \in \mathcal{W}\}$ is iteratively constructed that yields

$$\mathcal{H}(\alpha_c, \mathcal{A}) := \begin{cases} \underset{\alpha_a \in \mathcal{A}}{\text{argmax}} \left(\text{LENGTH}(\alpha_a \cap \alpha_c) \right) & \exists \alpha_a \in \mathcal{A} : \text{LENGTH}(\alpha_a \cap \alpha_c) > 0 \\ \varepsilon & \text{otherwise} \end{cases}$$

$$(5.12)$$

after the sweep. In the given algorithms, $\mathcal{H}$ actually contains a second component that holds the length of the respective intersection.

Algorithm 5.1 F-measure evaluation

Require: A set of computed clusters $\mathcal{V} = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$ and a set of annotated clusters $\mathcal{W} = \{\mathcal{A}_1, \ldots, \mathcal{A}_L\}$.

Ensure: BESTAPPROXIMATION : $\mathcal{W} \rightarrow \mathscr{P}(\mathcal{V})$, F-MEASURE : $\mathcal{W} \times \{\mathcal{W}\} \rightarrow [0,1]$ which give the best matching set of computed clusters for every annotated cluster and the corresponding F-measure.

1: $(\mathcal{E}, \mathcal{S}_a, \mathcal{S}_c, \mathcal{H}, \mathcal{I}) \leftarrow \text{INITEVALUATION}(\mathcal{V}, \mathcal{W})$
2:
3: **for all** $e \in \mathcal{E}$ **do** $\triangleright$ Sweep
4:     $\alpha_{cur} \leftarrow$ segment associated with e
5:
6:     **switch** type of e and α_{cur}
7:
8:         **case** START of a computed cluster segment
9:             **for all** $\alpha \in \mathcal{S}_a$ **do**
10:                 $\mathcal{A} \leftarrow$ cluster associated with α
11:                 $\mathcal{H}(\alpha_{cur}, \mathcal{A}) \leftarrow (\alpha, \text{LENGTH}(\alpha \cap \alpha_{cur}))$
12:                 $\mathcal{I}(\mathcal{A}) \leftarrow \mathcal{I}(\mathcal{A}) \cup \{\text{cluster associated with } \alpha_{cur}\}$
13:             **end for**
14:             $\mathcal{S}_c \leftarrow \mathcal{S}_c \cup \{\alpha_{cur}\}$
15:         **end case**
16:
17:         **case** START of an annotated cluster segment
18:             $\mathcal{A} \leftarrow$ cluster associated with α_{cur}
19:             **for all** $\alpha \in \mathcal{S}_c$ **do**
20:                 **if** $\mathcal{H}_2(\alpha, \mathcal{A}) < \text{LENGTH}(\alpha \cap \alpha_{cur})$ **then** $\triangleright$ second component
21:                     $\mathcal{H}(\alpha, \mathcal{A}) \leftarrow (\alpha_{cur}, \text{LENGTH}(\alpha \cap \alpha_{cur}))$
22:                     $\mathcal{I}(\mathcal{A}) \leftarrow \mathcal{I}(\mathcal{A}) \cup \{\text{cluster associated with } \alpha\}$
23:                 **end if**
24:             **end for**
25:             $\mathcal{S}_a \leftarrow \mathcal{S}_a \cup \{\alpha_{cur}\}$
26:         **end case**
27:
28:         **case** END of a computed cluster segment
29:             $\mathcal{S}_c \leftarrow \mathcal{S}_c \setminus \{\alpha_{cur}\}$
30:         **end case**
31:
32:         **case** END of an annotated cluster segment
33:             $\mathcal{S}_a \leftarrow \mathcal{S}_a \setminus \{\alpha_{cur}\}$
34:         **end case**
35:
36:     **end switch**
37: **end for**
38:
39: (F-MEASURE, BESTAPPROXIMATION) $\leftarrow$ EVALUATE$(\mathcal{H}, \mathcal{I})$

Algorithm 5.2 $\textsc{InitEvaluation}(\mathcal{V}, \mathcal{W})$

1: $\mathcal{E} \leftarrow$ sorted set of $\textsc{Start}$ and $\textsc{End}$ events for segments in $\mathcal{V}, \mathcal{W}$
2: $\mathcal{S}_a \leftarrow \mathcal{S}_c \leftarrow \{\}$
3:
4: $\mathcal{H} : \{\alpha \in \mathcal{C} : \mathcal{C} \in \mathcal{V}\} \times \mathcal{W} \rightarrow \varepsilon \cup \{\alpha \in \mathcal{A} : \mathcal{A} \in \mathcal{W}\} \times \mathbb{N}_0$ with
   $\forall (\alpha, \mathcal{A}) \in \{\alpha \in \mathcal{C} : \mathcal{C} \in \mathcal{V}\} \times \mathcal{W} \ : \ \mathcal{H}(\alpha, \mathcal{A}) \leftarrow (\varepsilon, 0)$
5:
6: $\mathcal{I} : \mathcal{W} \rightarrow \mathscr{P}(\mathcal{V})$ with $\forall \mathcal{A} \in \mathcal{W} \ : \ \mathcal{I}(\mathcal{A}) \leftarrow \emptyset$
7:
8: **return** $(\mathcal{E}, \mathcal{S}_a, \mathcal{S}_c, \mathcal{H}, \mathcal{I})$

Furthermore, a map $\mathcal{I} : \mathcal{W} \rightarrow \mathscr{P}(\mathcal{V})$ is constructed that specifies for every annotated cluster $\mathcal{A}$ the candidates for the best matching computed cluster using

$$\mathcal{I}(\mathcal{A}) := \{\mathcal{C} \in \mathcal{V} : \textsc{F-Measure}(\mathcal{A}, \mathcal{C}) > 0\} \, .$$

Both these maps and the necessary sweep data structures are initialised in the function $\textsc{InitEvaluation}$ which is given in algorithm 5.2.

In the sweep iteration in algorithm 5.1, different $\textsc{Start}$ and $\textsc{End}$ events are used for annotated and computed segments, and segments are active between their respective events. Also, two distinct status structures $\mathcal{S}_a$ and $\mathcal{S}_c$ are utilised to store annotated and computed segments that are active. On the basis of this, all intersections may be easily identified in the $\textsc{Start}$ events.

At the start of a computed segment α, only a single annotated segment may be active. This one is used to create the first real entry for α in $\mathcal{H}$ (cf. line 11). If further annotated segments start while α is active, $\mathcal{H}$ is updated in accordance with equation (5.12) (cf. line 20ff.). Here, the second component of $\mathcal{H}$ is used in order to determine whether an update is necessary. Finally, the construction of $\mathcal{I}$ in both these events is a trivial task.

The function $\textsc{Evaluate}$ which is presented in algorithm 5.3 computes for every annotated cluster the best matching set of computed clusters and the corresponding F-measure. This calculation is based on the previously obtained maps $\mathcal{H}$ and $\mathcal{I}$.

With respect to this, it has to be explained why a number of computed clusters should be accepted as an approximation of a single annotated cluster.

Frequently, the computed clusters represent a more fragmented structure than the one that is given by the annotation. In general, incomplete extracted paths or inconsistencies in the extracted paths produce this effect. This may lead to a number of clusters that gain a high precision rate with respect to the same annotated cluster. On the one hand, several computed clusters may encode different parts of an annotated cluster, and, on the other hand, there may be clusters with many short segments of which some are contained in a cluster with fewer, but longer segments.

Consequently, it would not be reasonable to accept only a single computed cluster as the calculated approximation of an annotated cluster.

Algorithm 5.3 Evaluate($\mathcal{H}, \mathcal{I}$)

1: $c_{all} \leftarrow d_{all} \leftarrow 0$ ▷ overall measures
2:
3: **for all** $\mathcal{A} \in \mathcal{W}$ **do**
4:
5:      **for all** $\mathcal{U} \in \mathscr{P}\left(\mathcal{I}(\mathcal{A})\right)$ **do**
6:        $c \leftarrow d \leftarrow 0$ ▷ measures per annoted cluster
7:
8:        **for all** $\alpha \in \mathcal{C} : \mathcal{C} \in \mathcal{U}$ **do**
9:          **if** $\forall \mathcal{C}' \in \mathcal{U} \setminus \{\mathcal{C}\} \ \forall \alpha' \in \mathcal{C}' \ : \ \alpha' \cap \alpha \neq \alpha$ **then**
10:            $c \leftarrow c + \textsc{Length}(\alpha)$
11:            $d \leftarrow d + \mathcal{H}(\alpha, \mathcal{A})$
12:          **end if**
13:        **end for**
14:
15:        $f \leftarrow \textsc{CalculateFMeasure}(\textsc{Length}(\mathcal{A}), c, d)$
16:
17:        **if** $f > \textsc{F-Measure}(\mathcal{A})$ **then**
18:          $\textsc{F-Measure}(\mathcal{A}) \leftarrow f$
19:          $\textsc{Computed}(\mathcal{A}) \leftarrow c$
20:          $\textsc{Detected}(\mathcal{A}) \leftarrow d$
21:          $\textsc{BestApproximation}(\mathcal{A}) \leftarrow \mathcal{U}$
22:        **end if**
23:      **end for**
24:
25:      $c_{all} \leftarrow c_{all} + \textsc{Computed}(\mathcal{A})$
26:      $d_{all} \leftarrow d_{all} + \textsc{Detected}(\mathcal{A})$
27: **end for**
28:
29: $\textsc{F-Measure}(\mathcal{W}) \leftarrow \textsc{CalculateFMeasure}(\textsc{Length}(\mathcal{W}), c_{all}, d_{all})$
30:
31: **return** $(\textsc{F-Measure}, \textsc{BestApproximation})$

Algorithm 5.4 CalculateFMeasure(a, c, d)

Require: Length of annotated (a), computed (c), and correctly detected (d) segments.
Ensure: F-Measure for given values.

1: $p \leftarrow d \ / \ c$
2: $r \leftarrow d \ / \ a$
3: **return** $2rp \ / \ (r + p)$

Therefore, all combinations of computed clusters in $\mathcal{I}(\mathcal{A})$ are regarded as best approximation candidates for the annotated cluster $\mathcal{A}$ (cf. line 5). Here, the notation $\mathscr{P}\left(\mathcal{I}(\mathcal{A})\right)$ is used to denote the power set of $\mathcal{I}(\mathcal{A})$.

However, it is not a good idea to calculate the F-measure on the basis of all the segments that are contained in a candidate set of clusters. For example, a segment that is covered by a longer segment of another cluster in the same candidate set would yield a too high recall rate. Therefore, a computed segment is only included in the F-measure calculation if this is not the case (cf. line 9).

The test for the containment relation should, of course, not be completely performed at this point as this would be a rather inefficient solution. For example, in the implementation of the outlined algorithm, the containment relations are already determined during the sweep iteration.

Consequently, this evaluation procedure computes the most meaningful approximation of a given annotated cluster by computed clusters, and the approximation quality is provided on the basis of the F-measure. Additionally, the lengths of the evaluated computed segments and of their correctly detected parts are summed up separately over all annotated clusters in order to compute the overall F-measure for the musical piece (cf. line 25ff.). The F-measure for a complete data set of musical pieces may be obtained in a similar fashion.

In many publications, a further structural variation is allowed in the computed results. As the analysis system described in this thesis depends solely on repetitions, it is possible that repeating sequences of sections are not correctly recognised. This corresponds to the situation illustrated in figure 5.1. Without employing further analysis techniques, it is generally not possible to compute the annotated structure given in figure 5.1(a). Given correctly extracted repetitions, it is more likely that a structure corresponding to figure 5.1(c) will be computed.

That is why a second evaluation procedure was developed where certain combinations of annotated clusters are allowed to replace the original clusters. To this end, the set $\mathcal{W}$ is replaced by sets of the form

$$\mathcal{U} \subset \mathscr{P}(\mathcal{W}) : \forall \mathcal{A} \in \mathcal{W} \; \exists_1 \mathcal{A}' \in \mathcal{U} : \mathcal{A} \in \mathcal{A}', \tag{5.13}$$

where,

$$\forall \mathcal{A}' \in \mathcal{U} \; \exists \mathcal{C} \in \mathcal{V} \; \exists \alpha_c \in \mathcal{C} \; \forall \mathcal{A} \in \mathcal{A}' \; \exists \alpha_a \in \mathcal{A} : \alpha_a \cap \alpha_c \neq \emptyset. \tag{5.14}$$

The union of segments contained in the annotated clusters of $\mathcal{A}' \in \mathcal{U}$ is treated as a single annotated cluster. On the basis of this, the previously described evaluation is performed for each possible annotation variant $\mathcal{U}$, and the variant yielding the highest overall F-measure is taken as the evaluation result.

The difference between both evaluation procedures is illustrated on an exemplary basis in figure 5.2. Here, an annotated structure of $A_1 B_1 A_2 B_2$ is given, and segments corresponding to $A_1 B_1$ and $A_2 B_2$ constitute the only computed cluster (figure 5.2(a)).

The first evaluation procedure rates the annotated clusters $\mathcal{A} = \{A_1, A_2\}$ and $\mathcal{B} = \{B_1, B_2\}$ separately, yielding a low precision rate for the computed segments
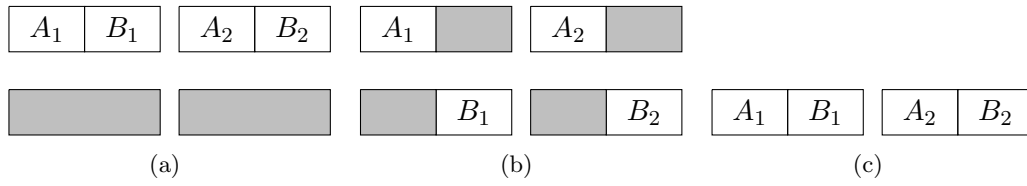
Figure 5.2: Evaluation procedures. (a) The top row contains the annotated structure and the bottom row two segments of a computed cluster. (b) First evaluation procedure: The computed cluster is seen to approximate each annotated cluster individually. (c) Second evaluation procedure: Approximation of an annotation variant by the computed cluster.

in both cases (figure 5.2(b)). In the second evaluation procedure, the combined annotated cluster $\mathcal{C} = \{A_1, A_2, B_1, B_2\}$ is a possible annotation variant, as there is a computed segment that has a non-empty intersection with segments in $\mathcal{A}$ and $\mathcal{B}$. In this case, a perfect F-measure is obtained, as for every computed segment the correctly detected portions in $\mathcal{A}$ and $\mathcal{B}$ are taken into account (figure 5.2(c)).

While the first described evaluation procedure often leads to unjustifiably low F-measure values, especially in the case of simply structured pop music, the second procedure may cause too positive results in some marginal cases where the analysis system completely fails to compute a meaningful result.

If the previously depicted example had been annotated as $A_1 B_1 B_2 A_2$, the computed cluster would still yield the same evaluation result, while according to the annotation $A_1 B_1$ clearly should not be regarded as being similar to $B_2 A_2$. The reason for this deficiency is that the originally annotated segments are evaluated independently in the combined annotated cluster $\mathcal{C} = \{A_1, A_2, B_1, B_2\}$. Because of this, they are also allowed to appear in arbitrary order in the computed segments. In reality, this failure should not occur that often, as in this case, differently annotated musical sections need to be identified as being musically similar nonetheless by the analysis system.

A possible alternative evaluation procedure might employ the split and roll-up process described in [25]. This process is capable of creating structurally corresponding representations of simple, non-hierarchical annotations and computed results which allow a direct comparison.

Here, the annotated and computed segments are first split and re-labelled to a common time-base. Then, consecutive segments in both resulting sequences that have equivalent labels are conflated again into larger segments leading to the simplest possible common structural representation of both sequences. The F-measure evaluation is then performed on the basis of this representation.

The main challenge in implementing this process is the necessary adaption to a hierarchical computed structure. Every abstraction level present in the computed hierarchy would need to be evaluated on its own, as only a single section label per instant of time is allowed in the final representation. This also complicates
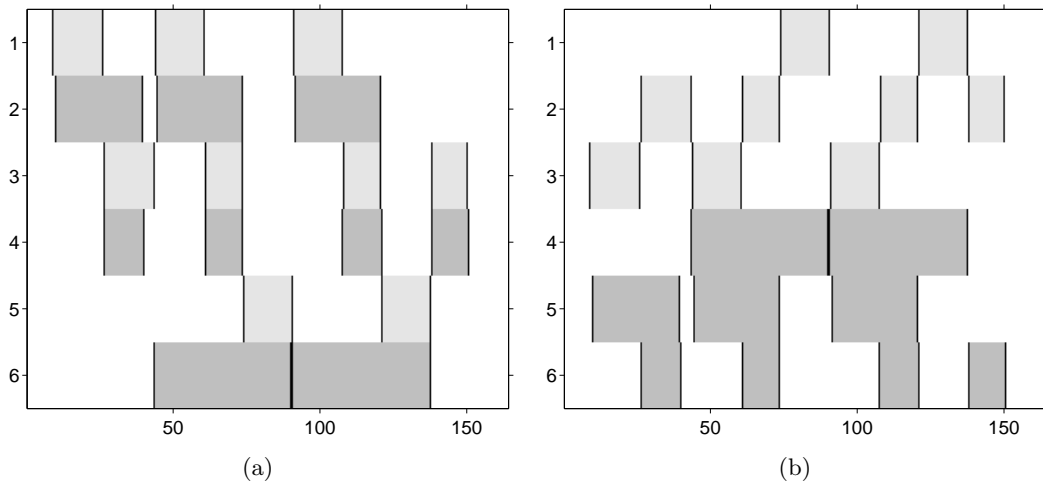
Figure 5.3: Visualisation of the results from the (a) first and (b) second evaluation procedure for *"With A Little Help From My Friends"* by *The Beatles*. Rows that depict annotated clusters (light grey ⬤) are directly followed by rows of the computed clusters (darker grey ⬤) that constitute the identified best matching constellation. An overall F-measure of (a) 0.71 and (b) 0.97 was obtained, respectively. As all annotated sections were correctly identified except for repeating sequences of sections that could not be separated, an F-measure of 0.97 seems to be reasonable.

the evaluation of combinations of computed clusters that may be easily achieved using the previously described evaluation procedures. Therefore, a split and roll-up evaluation procedure was not implemented for this thesis, as the necessary effort did not seem to justify the prospective improvement in the evaluation precision.

5.5 Results

This sections outlines the performance evaluation results that were obtained for the two data sets introduced in section 5.2. The aim was to investigate two aspects, namely the applicability of the music structure analysis system to a broader selection of musical pieces, and the effects of the methodologies that were introduced in chapter 3.

To this end, an analysis of both data sets of musical pieces was performed for all useful combinations of the developed methodologies. The analysis results were then rated on the basis of the two performance evaluation procedures that were described in section 5.4.

This yielded precision and recall rates, along with a corresponding F-measure for each musical piece, as well as over the entire number of pieces in the POP and SYNCHRONISATION data set, respectively.

The analysis of each musical piece was performed with feature resolutions of 1 Hz and 2 Hz, and with the combination of 1 and 2 Hz on the basis of the multiresolution approach introduced in section 3.3. Additionally, the path end improvement (section 3.4) as well as the automatic threshold selection method (section 3.5) were optionally employed. The obtained paths were then transformed into a hierarchical structure using the clustering approach that was presented in chapter 4. As clustering parameters (specified in number of features) $T_{id} = 2$, $T_{len} = 6$ and $T_{id} = 3$, $T_{len} = 12$ were used for feature resolutions 1 Hz and 2 Hz, respectively. Therefore, only those extracted musical sections having a length of at least 6 seconds were allowed in the results.

Overall values for both data sets are given in table 5.3 for nearly all combinations of these techniques. Further results for the analysis resolution of 1 Hz were omitted as their effect on the analysis is comparable to the situation at a feature resolution of 2 Hz. Moreover, it was decided that a comparison of the clustering approach from chapter 4 with the original clustering approach of the analysis system (section 2.4) is not particularly useful on the basis of this evaluation, as the results obtained using both methods are too different with respect to their expressiveness and comprehensibility.

With regard to both evaluation procedures, it is visible that the second evaluation procedure leads to significantly higher precision rates which was to be expected given the fact that it evaluates more possible mappings between annotated and computed clusters. As stated in section 5.4, this increase is mostly seen to be substantiated. The different outcome of both procedures is depicted in figure 5.3 that gives both evaluation results for a musical piece from the POP data set. As a similar behaviour was visible for many other musical pieces, the second evaluation method was chosen for a further analysis. Moreover, the increase in the precision rate is always lower for the SYNCHRONISATION data set. This is due to the fact that it contains numerous classical pieces whose musical structure is often more complex than is the case for pop music.

Unfortunately, the results do not differ significantly between the employed analysis techniques. Therefore, it is not possible to make a substantiated statement about the advancements featured by the techniques described in chapter 3. One reason for this is surely that the data sets were not explicitly developed with respect to this aim, that is, the musical pieces as well as their annotations do not seem to cover problematic cases where the newly developed techniques can show their potential. Furthermore, as mentioned at the end of section 5.2, a number of the employed annotations do not exactly represent the underlying musical structure which may cause further random effects on the evaluation measures.

Nonetheless, the obtained results at least support the assumption that the three techniques developed in chapter 3 by themselves do not have negative effects on the analysis results. In fact, the combination of a feature resolution of 2 Hz and the automatic threshold selection method gained best results for both data sets.

With respect to the overall performance and the applicability to a broader range of musical pieces, the results are of an acceptable quality.

Table 5.3: Overall evaluation results for each data set and a selected number of analysis configurations. Recall (R) and precision (P) rate as well as the corresponding F-measure (F) are provided on the basis of both outlined evaluation procedures. For the analysis system, selected combinations of a feature resolution of $1\,\mathrm{Hz}$ (1) and $2\,\mathrm{Hz}$ (2), and the path end improvement (E) as well as the threshold selection method (T) could be chosen. In the case of multiple analysis resolutions, the multiresolution approach was employed.

| Data	1	2	T	E	Evaluation 1			Evaluation 2		
					R	P	F	R	P	F
Pop	x				**0.71**	0.40	0.51	0.74	0.74	0.74
		x			0.68	0.47	0.55	0.71	0.76	0.73
		x	x		0.68	**0.50**	0.57	0.71	**0.83**	**0.77**
		x		x	0.70	0.45	0.55	0.73	0.77	0.75
		x	x	x	**0.71**	**0.50**	**0.58**	0.73	0.82	**0.77**
	x	x			0.69	0.43	0.53	**0.75**	0.72	0.74
	x	x	x		0.69	0.46	0.55	0.72	0.82	0.76
	x	x		x	0.68	0.43	0.53	0.71	0.78	0.74
	x	x	x	x	0.69	0.46	0.55	0.72	0.77	0.74
Synchronisation	x				0.74	0.56	0.64	0.75	0.86	0.80
		x			0.73	**0.68**	**0.70**	0.75	**0.89**	0.81
		x	x		0.74	0.66	**0.70**	**0.76**	**0.89**	**0.82**
		x		x	**0.75**	0.65	**0.70**	**0.76**	**0.89**	**0.82**
		x	x	x	0.72	0.62	0.67	0.73	0.85	0.79
	x	x			**0.75**	0.63	0.68	**0.76**	**0.89**	**0.82**
	x	x	x		0.73	0.64	0.68	0.75	0.87	0.80
	x	x		x	**0.75**	0.63	0.68	**0.76**	0.88	**0.82**
	x	x	x	x	0.73	0.59	0.65	0.74	0.84	0.79
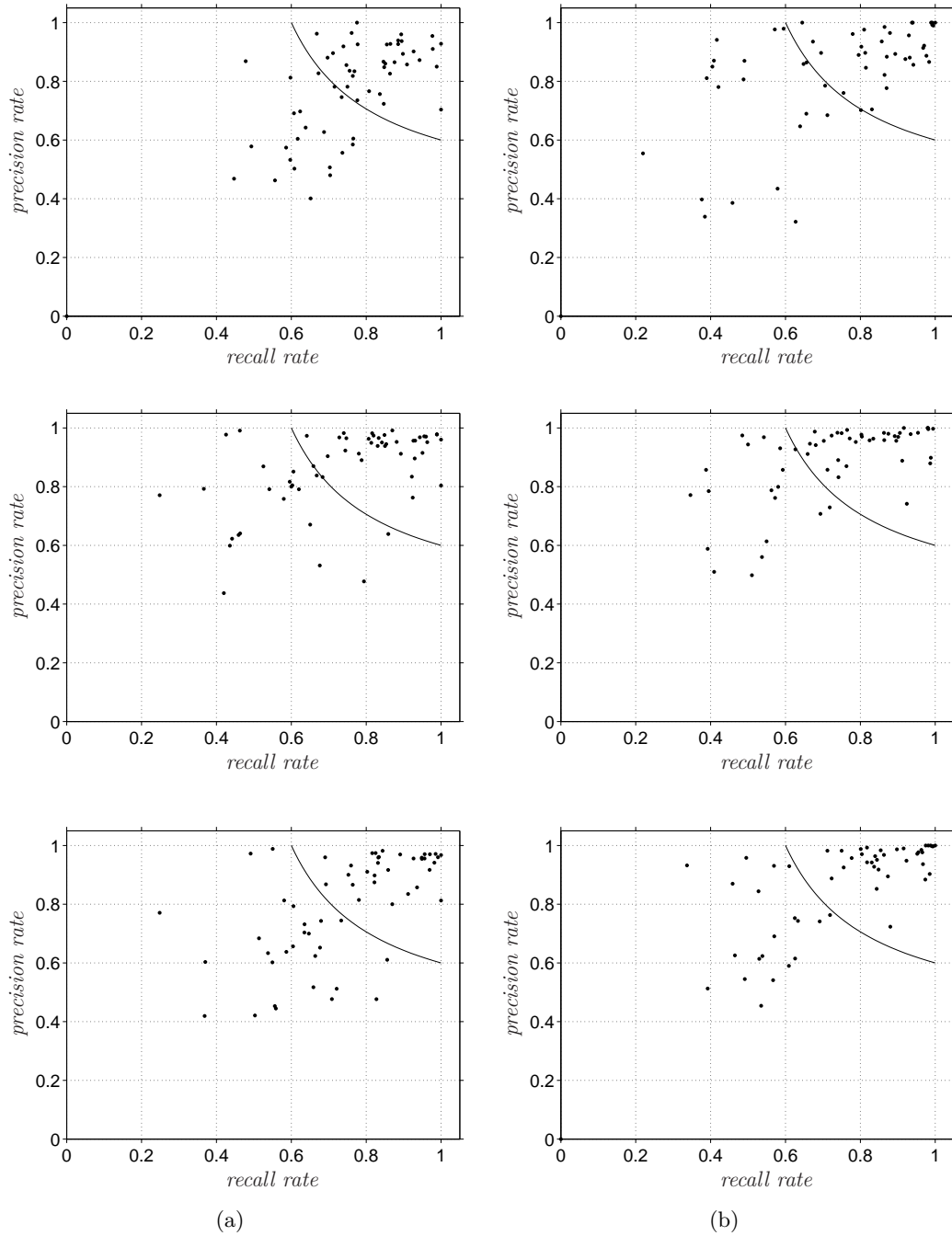
Figure 5.4: Precision and recall rate for each musical piece in the (a) POP and (b) SYNCHRONISATION data set obtained from the second evaluation. The three tested configurations are: 1 Hz (top), 2 Hz with automatic threshold selection (middle), and the multiresolution approach with 1 and 2 Hz (bottom). As a reference point, an F-measure value of 0.75 is given as a solid line.

79

In music structure analysis research, an F-measure of 0.75 is often regarded as a lower bound for correctly identified musical structures [12, 25]. Although the evaluation procedure outlined in this chapter is special in some aspects, an empirical analysis of the evaluation results confirmed that this value is a reasonable choice. To this end, the overall values for both data sets support the above-mentioned statement.

The individual results obtained for the musical pieces are depicted in figure 5.4 for three of the tested analysis configurations. Additionally, the exact values may be retrieved from the tables given in appendices A and B. As is easily visible, the results differ severely. In fact, F-measures covering the complete range from 0 to 1 were obtained. The first, rather unusual case happened if the path extraction thresholds were set too rigidly for the musical material. In this case, the automatic threshold selection method presented in section 3.3 clearly showed its strengths. An F-measure close to 1 was often obtained for classical music. Pieces of pop music that employ clear patterns of changing harmonies and whose annotations preferably consist of longer sections also obtained very high F-measures. Among others, this was often the case for *The Beatles*.

Some selected evaluation results are provided in figure 5.5. Note that the given figures are probably not didactically ideal as the represented mapping from computed clusters to annotated clusters is based on the first evaluation procedure while the annotated F-measure is obtained from the second evaluation procedure. The reason for this is that the given figures are generally easier to comprehend than those obtained by the second evaluation procedure.

The examples by *Prince* and *Abbey Lincoln* are representative for musical pieces where too many or too few paths were extracted. Here, the second case at least represents a viable improvement with respect to the analysis using the pre-selected path extraction thresholds where not a single path was extracted. The *Björk* and the second *Brahms* example also suffer somewhat from a too tolerant path extraction. But in this case, it is hard even for most human listeners to decide whether two musical sections should be considered similar or not. In this case, slightly more rigid thresholds could also have had a more negative effect.

Finally, in order to illustrate that no superior analysis combination could be deduced from this evaluation, two figures are given where analysis results obtained using 2 Hz and the automatic threshold selection method are contrasted to the results obtained using the multiresolution approach with 1 and 2 Hz. Figure 5.6 illustrates an example where the first method outperformed the multiresolution approach, whereas figure 5.7 depicts an example of the contrary case. Both examples are representative for the most extreme differences that were obtained using the various analysis combinations. In contrast to this, most musical pieces did not exhibit significant evaluation performance differences throughout the employed analysis configurations.

Finally, the number of musical pieces combined with the number of varying test configurations show that the complete music structure analysis system including the newly developed methodologies is reliably implemented.
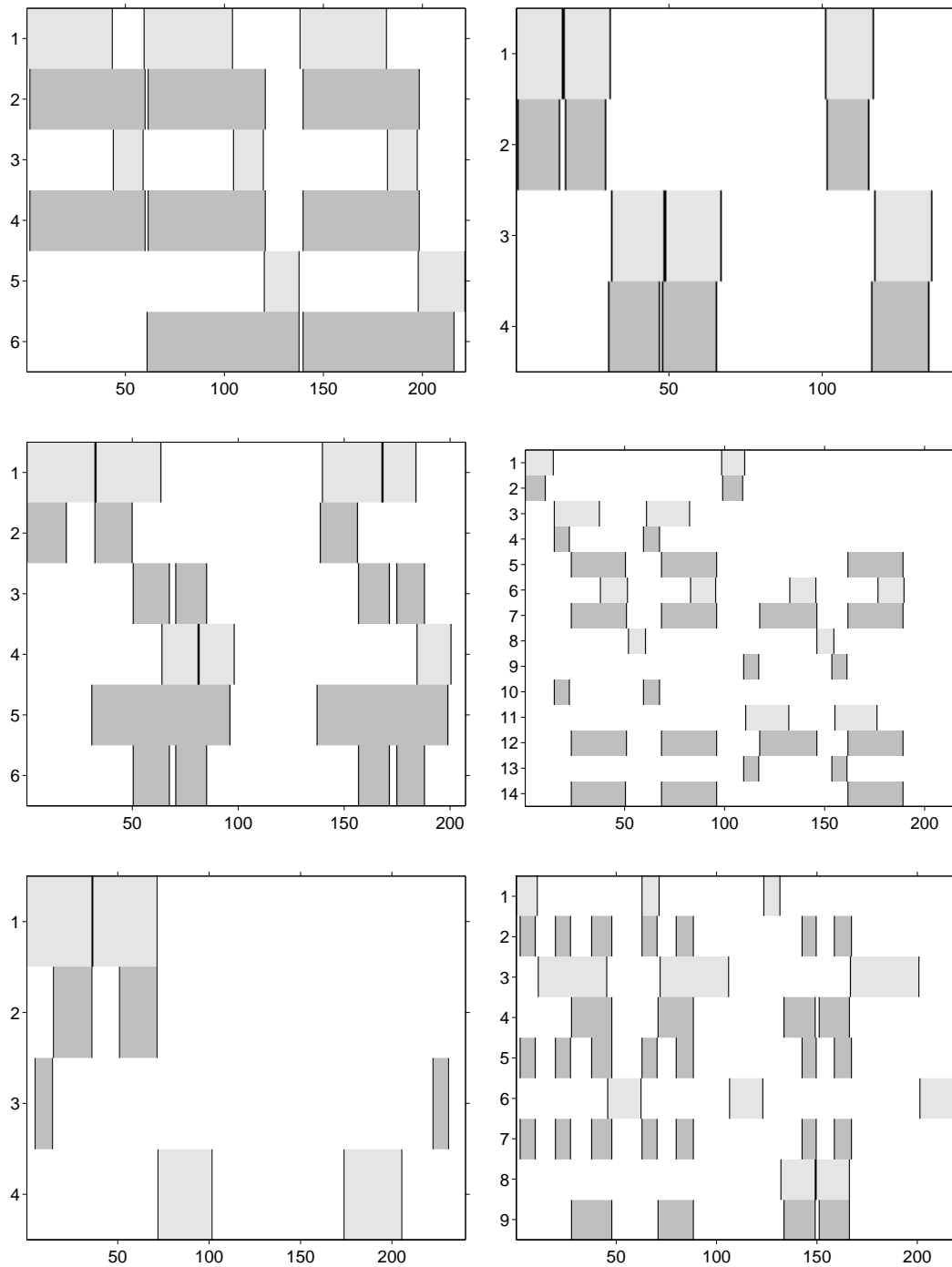
Figure 5.5: Examples ranging from excellent (top left) to not so good (bottom right) evaluation results. These results were obtained at 2 Hz including the automatic threshold selection method. *"The Look Of Love"* by *Dusty Springfield* (F-measure 0.97), *"Hungarian Dances No, 5 in G minor (Allegro)"* (0.96) and *"Hungarian Dances No, 6 in D major (Vivace)"* by *Johannes Brahms* as interpreted by *Scholz* (0.76), *"It's Oh So Quiet"* by *Björk* (0.73), *"You And I"* by *Abbey Lincoln* (0.53), *and "Kiss"* by *Prince* (0.43).
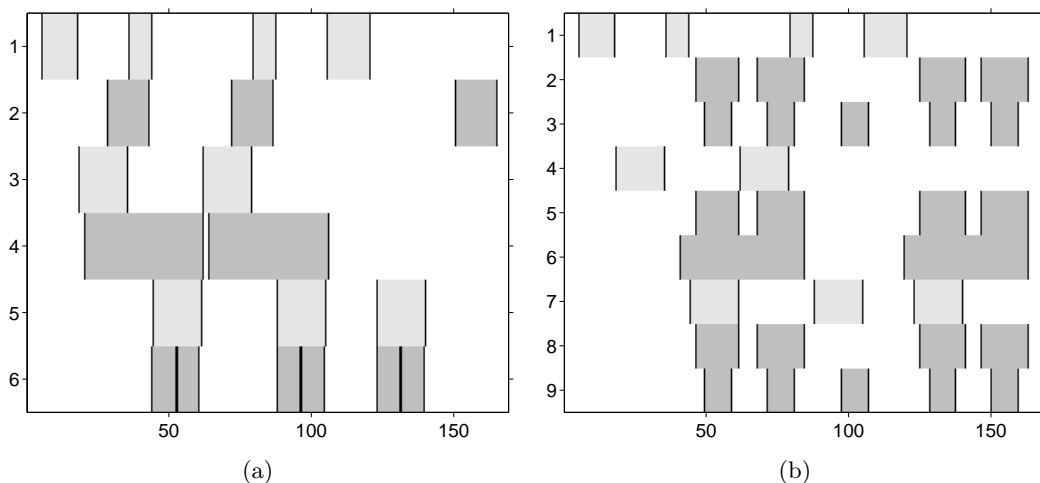
Figure 5.6: Evaluation results for *"Wannabe"* by the *Spice Girls*. An F-measure of (a) 0.84 was obtained using the analysis at 2 Hz together with the automatic threshold selection method, and (b) 0.59 was obtained for the multiresolution approach with 1 Hz and 2 Hz. The first analysis method even revealed the finer substructure inherent in the chorus.
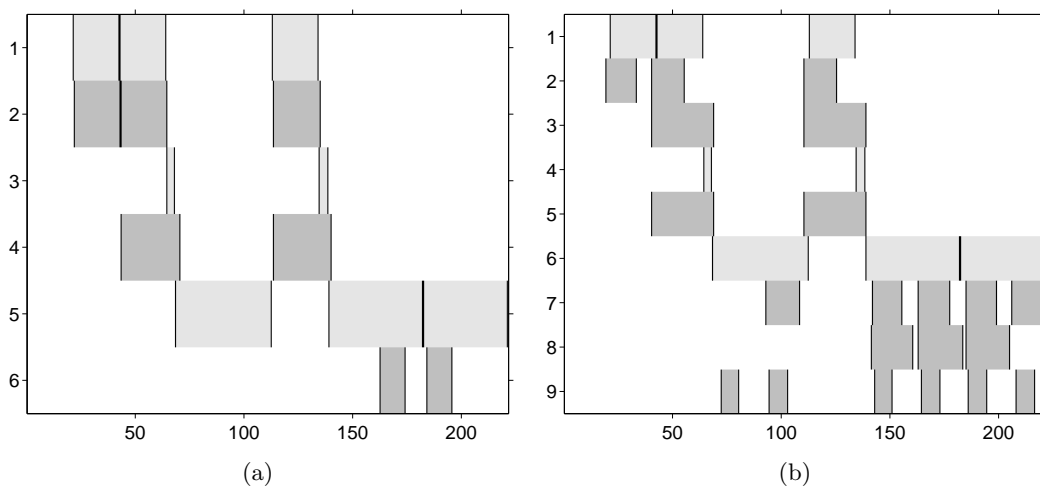
Figure 5.7: Evaluation results for *"Can't Take My Eyes Off You"* by *Lauryn Hill*. An F-measure of (a) 0.65 was obtained using the analysis at 2 Hz together with the automatic threshold selection method, and (b) 0.89 was obtained for the multiresolution approach with 1 Hz and 2 Hz. In this case, the threshold selection method chose too rigid path extraction thresholds, so that the chorus sections of the piece were only marginally identified.

6 Application: Path-Constrained Partial Music Synchronisation

Digital music collections often contain different versions of a single musical work. This includes recordings of interpretations by varying artists, different edits of a single recording, as well as various representation formats such as digital audio, MIDI, or score material. With respect to music retrieval in general and browsing applications in particular, one important task is to find a temporal alignment of semantically corresponding events in two versions of the same underlying musical piece. In particular, *audio synchronisation*, where this task is constrained to digital audio recordings, has seen a lot of research in recent years [21].

Although current synchronisation techniques are capable of coping with significant variations regarding tempo, dynamics, or instrumentation, they mostly rely on the assumption that the two recordings to synchronise exhibit the same global structure. However, in real-world scenarios, structural differences are not that uncommon. A pop song, for example, is generally released in structurally different versions which are meant to serve the varying needs of radio stations, dance clubs, or home listeners. In classical music, different conductors often vary the number of repetitions, and solo parts may contain significant interpretative differences. Consequently, audio synchronisation techniques should also be robust towards this end.

In this chapter, a combined approach to audio synchronisation is presented which incorporates further knowledge obtained through structural analysis in order to circumvent the above-mentioned constraint. This technique, called *path-constrained partial music synchronisation*, was first introduced in [22].

Most approaches to audio synchronisation share a great number of analogies with audio structure analysis algorithms. With respect to this, section 6.1 first presents a general approach to audio synchronisation and introduces the compound structural analysis of two audio recordings. On this basis, a so-called *path-constrained similarity matrix* for two audio recordings is developed which incorporates various structural information (section 6.2). This matrix together with a procedure to extract *partial matches* which is presented in section 6.3 encompasses the main result of this chapter. Furthermore, section 6.4 presents a methodology which may be used to evaluate the performance of this approach together with corresponding results which were obtained for one of the test sets that was introduced in chapter 5. Finally, section 6.5 gives a summary of what has been achieved in this chapter, together with possible future developments in this field.

6.1 Analogies with Audio Structure Analysis

The most common approach to time-align two digital audio recordings is known as dynamic time warping (DTW). Similar to audio structure analysis, this technique employs a transformation of the audio signals into audio feature sequences, a similarity measure to compare these sequences, and an algorithm to extract an alignment path from a similarity matrix (cf. chapter 2).

In contrast to audio structure analysis, here the similarity measure is used to compare features from the sequence $V := (v_1, v_2, \ldots, v_N)$ with features from the sequence $W := (w_1, w_2, \ldots, w_M)$, corresponding to both audio recordings, respectively. In general, the employed measure expresses real similarity, as for example

$$c(v, w) := \langle v, w \rangle \tag{6.1}$$

for $(v, w) \in V \times W$, which may also be seen as a *score* value (cf. equation (2.2)). This results in an $(N \times M)$ similarity matrix, which is defined by $\mathcal{S}(n, m) := c(v_n, v_m)$, $(n, m) \in [1 : N] \times [1 : M]$. The aim of classical DTW is to extract a single path which starts at matrix entry $(1, 1)$, ends at (N, M), and adheres to a step-size constraint with $\Delta := \{(1, 0), (0, 1), (1, 1)\}$. To this end, dynamic programming is used to compute a corresponding *score maximising path*. The temporal relation between feature indices in both sequences which is induced by the obtained path, then, is seen as the final alignment result. Furthermore, most of the enhancement approaches introduced in chapters 2 and 3 may be applied to improve the result.

Figure 6.1 depicts two alignment results that were obtained on the basis of a classical DTW approach. Both alignments are not satisfying as the DTW method is based on the assumption that the audio recordings do not exhibit structural differences.

The analysis of structural commonalities and differences between two audio signals may be easily modelled as a special case of the general music structure analysis of a single musical piece. To this end, a *compound* or *joint* audio structure analysis may be obtained by treating the concatenation of both feature sequences V and W as a representation of a single musical piece. Now, if both audio recordings exhibit an identical musical structure, the resulting structural overview should consist of two possibly temporally distorted copies of the same structure.

6.2 Path-Constrained Similarity Matrix

In order to obtain an acceptable audio synchronisation between structurally different recordings, a similarity matrix is needed that allows the computation of a time-alignment which is constrained to semantically meaningful sections. In substance, this is achieved by incorporating information obtained from a compound structural analysis as well as information of an ordinary path extraction into the normal similarity matrix $\mathcal{S}$.
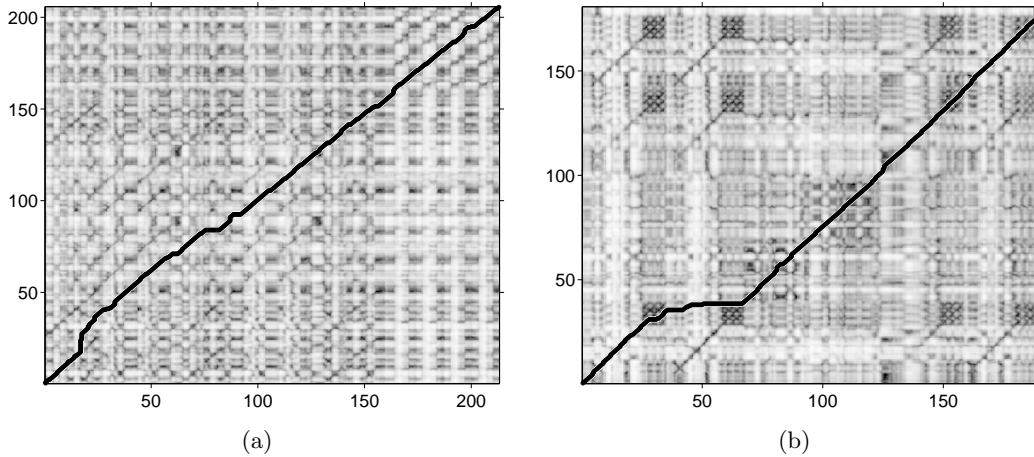
Figure 6.1: Classical DTW alignment of (a) a studio take (vertical) and the single version (horizontal) of *"Hey Jude"* by *The Beatles* and (b) a *Chailly* and *Yablonsky* interpretation of *"Waltz 2"* from *"Suite for Variety Stage Orchestra"* by *Dimitri Shostakovich*. The audio recordings were modified in order to highlight the synchronisation problems that may occur in case of structural differences. The obtained alignment is represented as black path in the corresponding DTW matrices.

Similar to section 3.3, a combination of several similarity matrices is calculated with the aim of regaining their strengths in the final result while hiding their weaknesses. To this end, four similarity matrices are constructed. First, the two similarity matrices $\mathcal{S}^{\mathrm{chroma}} := 1 - \mathcal{S}[\mathbf{w}, \mathbf{d}]$ and $\mathcal{S}^{\mathrm{enh}} := 1 - \mathcal{S}_L^{\min}$ are computed with identical reference parameters $\mathbf{w}$ and $\mathbf{d}$, as introduced in section 2.2. Furthermore, paths are extracted from $\mathcal{S}^{\mathrm{enh}}$ using the method described in section 2.3, and converted back into the similarity matrix $\mathcal{S}^{\mathrm{path}}$. Here, all entries corresponding to a path link are set to 1, and the remaining entries are set to 0. Finally, on the basis of the algorithm described in section 2.4, a global structural overview is constructed from the set of paths. Then, corresponding to $\mathcal{S}^{\mathrm{path}}$, the obtained similarity clusters are re-integrated into the similarity matrix $\mathcal{S}^{\mathrm{struct}}$. Finally, the *path-constrained similarity matrix* $\mathcal{S}^{\mathrm{pc}}$ is calculated as

$$\mathcal{S}^{\mathrm{pc}} := \frac{1}{6}(\mathcal{S}^{\mathrm{path}} + \mathcal{S}^{\mathrm{struct}}) * (\mathcal{S}^{\mathrm{chroma}} + \mathcal{S}^{\mathrm{enh}} + 1), \tag{6.2}$$

where $*$ denotes the point-wise multiplication of matrix entries.

The global structural overview which is incorporated into $\mathcal{S}^{\mathrm{struct}}$ is supposed to repair those relations which have only been partially extracted as paths. However, inconsistencies and inaccuracies inherent in the extracted paths may also have negative effects on the construction of the global structure. Consequently, a combination of both matrices is used in the factor $(\mathcal{S}^{\mathrm{path}} + \mathcal{S}^{\mathrm{struct}})$, which restricts the non-zero entries in $\mathcal{S}^{\mathrm{pc}}$ to entries that are supported by extracted structural aspects.
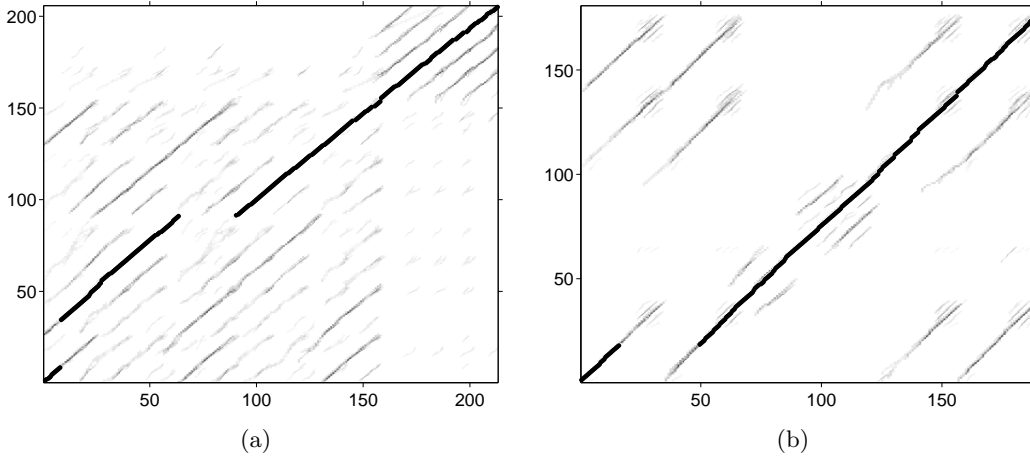
Figure 6.2: Alignments obtained using the partial matching procedure on the similarity matrix $\mathcal{S}^{\mathrm{pc}}$ for (a) *The Beatles* and (b) the *Dimitri Shostakovich* example. While the alginment is constrained to musically meaningful sections it still contains unnecessary jumps in between the repetitions of sections in both depicted examples.

The combination of $\mathcal{S}^{\mathrm{chroma}}$ and $\mathcal{S}^{\mathrm{enh}}$ offers a compromise between the temporal exactness and the robustness towards musical variations. In fact, $(\mathcal{S}^{\mathrm{chroma}} + \mathcal{S}^{\mathrm{enh}} + 1)$ is used, in order to ensure that all entries selected by structural features are non-zero and weighted by corresponding similarity measures. As the entries of all involved matrices lie in the range $[0, 1]$, the same applies to the matrix $\mathcal{S}^{\mathrm{pc}}$.

6.3 Partial Matching Procedure

The goal of this section is to extract a partial time-alignment between two audio recordings from the corresponding path-constrained similarity matrix. Here, the motivation is that if a musical section in one recording does not have a suitable counterpart in the other recording, it is preferable to have no alignment rather than having a bad alignment as is typical for classical DTW approaches. Thus, the notion of a path has to be generalised in order to allow for arbitrarily large gaps. A *match* is a path $\mu = (\mu_1, \ldots, \mu_L)$ with $\mu_l = (n_l, m_l) \in [1 : N] \times [1 : M]$, $l \in [1 : L]$ which adheres to a step-size constraint using $\Delta := \mathbb{N}^2$. Consequently, both sequences $(n_1, \ldots, n_L)$ and $(m_1, \ldots, m_L)$ are required to be strictly monotonic. Furthermore, a match should align similar and preferably long consecutive segments in the two given audio recordings. The *score* of match μ is then defined as $\sum_{l=1}^{L} \mathcal{S}^{\mathrm{pc}}(n_l, m_l)$.

Similar to DTW, dynamic programming is used to compute a score-maximising match. To this end, an accumulated similarity matrix D is defined by

$$D(n, m) := \max \left\{ D(n, m-1), D(n-1, m), D(n-1, m-1) + \mathcal{S}^{\mathrm{pc}}(n, m) \right\} \quad (6.3)$$

and $D(n, 0) := D(0, m) := 0$ for $(n, m) \in [0 : N] \times [0 : M]$. Here, the maximum score is given by $D(N, M)$ and an according score-maximising match may be computed by an ordinary *backtracking* algorithm.

Here, the path-constrained similarity matrix $\mathcal{S}^{\mathrm{pc}}$ introduced in section 6.2 plays an important role as it restricts a match to those matrix entries which correspond to structural aspects inherent in the musical material. Therefore, a semantically meaningful match aligning musically similar events may be obtained.

However, the resulting match generally still suffers from fragmentation, that is, the alignment may jump in between the repetitions of a section in one of the audio recordings. For example, given the musical sections $A_1^1 B^1$ in the first recording, and $A_1^2 A_2^2 B^2$ in the second recording, it may be possible that the first half of A_1^1 is aligned to the first half of A_1^2, whereas the second half of A_1^1 is aligned to the second half of A_2^2. However, it would be more semantically correct to align A_1^1 completely to A_1^2 or A_2^2.

Figure 6.2 illustrates this procedure on the basis of the two examples given in figure 6.1. Here, in the *Shostakovich* example an A section is missing at the start of the *Chailly* interpreation (horizontal). The only A section at the start is aligned in part to the first and second A section in the *Yablonsky* interpretation.

A cleaning step is used in order to circumvent such fragmentation and to obtain consecutive runs in the alignment that are as long as possible. To this end, first, match μ is decomposed into pairwise disjoint *path components* of maximum length. Here, two consecutive path links $\mu_l = (n_l, m_l)$ and $\mu_{l+1} = (n_{l+1}, m_{l+1})$ in μ are considered to belong to the same path component, if they fulfil

$$\max(n_{l+1} - n_l, m_{l+1} - m_l) \leq \tau \tag{6.4}$$

for a given threshold τ.

Then, the path component μ^1 containing the most path links is successively extended to the upper right and lower left. To this end, a similarity matrix entry $(n, m) \in \mathcal{S}^{\mathrm{pc}}$ fulfilling the step-size constraint is sought, for which a corresponding path link $\mu_k = (n_k, m_k) \in \mu \setminus \mu^1$ exists, so that

$$(|n - n_k| \leq \tau \lor |m - m_k| \leq \tau) \land \mathcal{S}^{\mathrm{pc}}(n, m) > \rho \mathcal{S}^{\mathrm{pc}}(n_k, m_k) \tag{6.5}$$

for some tolerance threshold $\rho \in [0, 1]$. Here, the first part of the disjunction removes a possibly unnecessary jump from μ^1 to another path component, and the second part guarantees that the new path link preserves a more or less comparable score. Path component μ^1 is extended iteratively by matrix entries identified in this fashion, while corresponding path links μ_k are removed from the original match. If no more extensions to μ^1 are possible, it is removed from the match, and the process is restarted for the next shorter path component and so on. The set of extended path components then constitutes the new match μ'. In the implementation of the described cleaning procedure the parameters are set to $\tau = 3$ and $\rho = 0.6$, respectively.
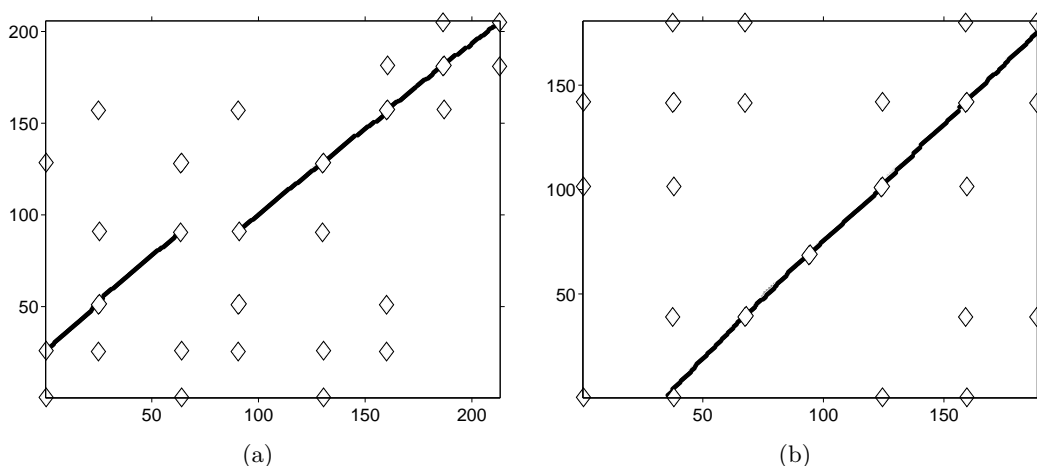
Figure 6.3: Final alignment result obtained after the cleaning step for (a) *The Beatles* and (b) the *Dimitri Shostakovich* example. In both cases musically similar sections of a maximum possible length are aligned. White diamonds indicate the start and end points of musically similar sections.

According to this procedure, the number of path components in the match is reduced or retained. However, the score of μ' generally is significantly lower than that of μ. To compensate for this effect, the positive entries in the similarity matrix $\mathcal{S}^{\mathrm{pc}}$ are restricted to connected regions of positive scores that contain at least one path link of μ'. Here, the connectivity between matrix entries is defined similarly to equation (6.4). Finally, a match μ'' is obtained from this matrix using the dynamic programming approach outlined above. This match represents the final alignment result.

The final alignments that were obtained for both previously employed examples are given in figure 6.3.

6.4 Experimental Results

Similar to chapter 5, the evaluation of the developed synchronisation procedure is based on manually annotated musical pieces. To this end, different interpretations, mixes, or cover versions of the same underlying musical piece are used as a starting point. In order to extend the number of possible test cases, structural manipulations to these original recordings are computed on the basis of the annotations. Here, insertions and deletions of possibly consecutive musical sections are used that have a duration of at least 20 seconds. Then, an audio synchronisation between two different and possibly modified audio recordings is computed by the algorithm outlined in this chapter.

The audio synchronisation performance is evaluated on the basis of the path components inherent in a computed match. To this end, the manual annotations of

both recordings are utilised in order to determine whether a path component aligns musically similar sections. Here, a section is said to be part of the match, if more than half of it is covered by the corresponding projection of the match. Furthermore, as a second criterion, the distance between an end point of a path component and the similarity matrix entry corresponding to the end of the outermost two musically similar sections covered by that component is measured with respect to the maximum norm. Then, a computed match is said to be correct, if all of its path components solely align musically similar sections. If, additionally, there is no end point distance that exceeds a given threshold T_{bias}, the match is said to be strongly correct.

On the basis of these performance measures, the described synchronisation algorithm was tested on 128 different synchronisation pairs which resulted in 318 path components [22]. Using $T_{bias} = 3$ as the bias threshold, 81% of all matches were correct, and 44% were even strongly correct. A detailed presentation of the obtained results together with sonifications of the computed alignments is available online[1].

6.5 Final Notes and Future Work

In this chapter a new audio synchronisation technique was introduced, which is capable of computing a meaningful alignment between two digital audio recordings even in the presence of global structural differences.

To this end, general music structure analysis is employed to identify musically similar parts in both audio recordings as well as musical sections that are missing from one of both recordings. In order to be robust towards a great a amount of musical variation, a combination of several similarity matrices is computed, the path-constrained similarity matrix $\mathcal{S}^{pc}$ (section 6.2).

Among others, it is based on two matrices that are obtained from a joint structural analysis of both audio recordings. On the one hand, a matrix representing identified repetitions which enforces direct musical similarities, and on the other hand a matrix representing the clustering result of the joint structural analysis which is supposed to repair similarity relations that are missing from the first matrix.

Furthermore, a flexible time-alignment procedure was developed in order to avoid bad alignments in the case of global structural differences which is typical for classical DTW methods. With regard to this, a technique based on dynamic programming is used to extract a match that aligns similar consecutive segments that are as long as possible in both audio recordings (section 6.3). One important aspect of this approach is the restriction of the extracted alignments to those entries in $\mathcal{S}^{pc}$ which correspond to common structural aspects inherent in both musical pieces. Moreover, a post-processing step is introduced to enforce alignments of possibly long consecutive segments.

---

[1]Path-constrained partial music synchronisation results `http://www-mmdb.iai.uni-bonn.de/projects/partialSync`

The performance of the suggested synchronisation technique was evaluated on a large test data set (section 6.4). The obtained results testify to a good synchronisation performance especially in the presence of global structural variations. Nonetheless, there are still some aspects in the described system that may need further development.

One improvement which is directly eminent, is to employ the advancements to general music structure analysis that were introduced in chapter 3 and 4. With regard to this, common structural patterns in audio recordings should be identified more accurately which should substantially improve the similarity matrix corresponding to the clustering result of the joint analysis.

Given a suitable music structure analysis system, it may even be possible to primarily base the synchronisation technique on the clustering result of the joint structural analysis. An approach based on the longest common subsequence problem (LCS) may be employed to identify a best matching sequence of common musical sections in two audio recordings. On the basis of this information, simple classical DTW techniques may be used to also obtain an exact alignment within these structural components. As a preliminary test, a naive implementation in Matlab showed promising results with simple audio recordings. In the future, a more sophisticated implementation may be based on the algorithm outlined in [20].

Finally, it is a remaining task to characterise the musical parts that could not be aligned, that is, musical sections that are missing in one of the recordings. To this end, the result of the joint structural analysis may be used. With a similar approach, it should even be possible to compute an alignment between audio recordings that employ temporally rearranged global structures.

7 Summary and Future Work

The topic of this thesis was the automatic structural analysis of music material. The aim of such an analysis is to automatically deduce structural information inherent in a musical piece from a corresponding digital audio recording.

The desired type of information may be described as a grouping structure, that is, a possibly hierarchical subdivision of the musical piece along the time domain into musical sections such as motifs, phrases, or musical themes.

The foundation of this thesis was an existing audio structure analysis system which was presented in [23] and [21]. In contrast to most other available systems, it was able to correctly deal with a broad range of musical variations, for example in dynamics, timbre, execution of note groups, and tempo progression.

Nonetheless, it lacked some desirable qualities. On the one hand, the analysis results were highly dependent on the chosen audio feature resolution. On the other hand, it was often difficult to interpret the finally obtained results because of unresolved overlaps between identified musical sections.

With regard to the first point, the aim of this thesis was the development of a multiresolution approach which allows a structural analysis that simultaneously incorporates audio features of multiple resolutions. Here, the hope was that this allows the identification of all possible structural aspects at the same time.

Furthermore, an improvement of the structural quality of the final result was aspired. To this end, a methodology that is capable of computing an easily comprehensible approximation of the hierarchical structure inherent in a musical piece was the desired result.

One of the cornerstones in the processing of the original analysis system was the computation of a matrix representing the amount of similarity between any two short frames of time in the underlying musical piece. During the course of this thesis it quickly became apparent that this was the key to a multiresolution analysis.

To this end, a methodology was developed that allows the incorporation of structurally relevant information from a lower resolution similarity matrix into a higher resolution matrix (chapter 3). On the basis of this approach, it was possible to combine the robustness towards musical variation that is typical for an analysis at a lower resolution with the exactness inherent in features of a higher resolution. This effect could also be verified on the basis of experimental results. Furthermore, this approach can be easily integrated into the original analysis system.

A further weakness in the original system concerned the exactness at the end of an identified repetition as it was only estimated on the basis of a simple heuristic. A technique similar to the above-mentioned multiresolution approach was developed to circumvent this deficiency. This time, information from the similarity matrix of

the reversed musical piece was incorporated into the matrix that was actually used for the identification of repetitions. In this case, the obtained exemplary results were also promising.

Moreover, automatic threshold selection methods were examined in order to allow an automatic adaption of the parameters that are relevant to the identification of repetitions in the actual musical piece. This proved to be necessary as the previously employed set of fixed parameters was not capable of covering all music material. Experiments indicated that in the cases where the predefined values worked well, the proposed method achieved comparable results. In most other cases, the automatic approach yielded much better results.

In this thesis, the main effort went into the improvement of the clustering step of the analysis system (chapter 4). To this end, a formalisation of the problem domain was introduced that defines the connection between the desired hierarchical structural result and the set of identified repetitions of a musical piece. With respect to this, a clustering approach on the basis of the sweep paradigm was developed that performs a corresponding transformation into an easily comprehensible, hierarchical structural overview. This is one of the most important achievements of this thesis as other analysis systems generally do not offer this feature.

In order to gain more confidence in the performance of the developed methodologies, an automatic evaluation on a larger data set of musical pieces was performed (chapter 5). It was based on a ground truth that only reflected a single level of structural abstraction per musical piece which is typically the case for musical structure test data sets. Here, a performance evaluation procedure needed to be established that was capable of judging the quality of a computed hierarchical structure with respect to this information. Because of these facts, the developed evaluation algorithm needed to be more powerful than would normally be expected, as a suitable mapping from the simple ground truth to the computed hierarchical structure needs to be calculated.

For the performance evaluation, two test data sets were employed containing over 100 musical pieces in total. The annotations for the SYNCHRONISATION data set were created by the author of this thesis for the performance evaluation employed in [22]. As an external resource, the POP data set was utilised which was established for another research publication that had a slightly different focus. Unfortunately, the obtained overall results did not indicate significant improvements for the approaches introduced in chapter 3. This is mainly due to fact that the employed musical pieces were not explicitly chosen with respect to this application. Moreover, for some musical pieces, the ground truth was not as detailed as would have been necessary to highlight the profits obtained by the above-mentioned improvements. Nonetheless, the obtained results are very promising, as, besides most classical pieces, a great number of pieces of pop music were analysed successfully.

As an application of the described audio structure analysis system, an audio synchronisation algorithm was introduced (chapter 6). While, generally, synchronisation algorithms are built on the assumption that the musical material to be synchronised employs a common structural form, this is often not the case in reality.

Music structure analysis may be applied to the concatenation of two digital audio recordings in order to detect common structural patterns. On the basis of this information, a synchronisation procedure was presented that aligns musically similar sections while omitting the alignment if one of the recordings misses a section. In this case, a performance evaluation on the basis of multiple structurally differing interpretations of a number of musical pieces yielded promising results.

Automatic music structure analysis is a vivid research field. Although the methodologies presented in this thesis improve the originally given analysis system with respect to its accuracy as well as the quality of the extracted musical structure, there are still several possible improvements.

First, many of the described techniques may be improved. With respect to the the ending of identified repetitions, some experiments showed that the incorporation of the reversed similarity matrix sometimes has too much influence on the overall quality of the extracted repetitions. A possible improvement was already outlined at the end of chapter 3. More effort seems to be necessary to obtain a threshold selection method that really works universally. Here, the presently favoured method should at least be justified on a more theoretical level. Furthermore, it may even be plausible to try a completely new approach to identify repetitions from a similarity matrix, for example, on the basis of an artificial neural network. Regarding the developed multiresolution approach, a further improvement may be achieved if the analysis resolution could be automatically chosen with respect to the underlying music material. To this end, onset, tempo, and rhythm detection systems may be employed.

With respect to the proposed clustering approach, improvements are possible in all fields. On the one hand, the efficiency and accuracy of the system may be greatly improved, if formalised constraints are more explicitly enforced and if information that is additionally available like the quality of the path links is exploited. On the other hand, the complexity of the developed methodology may be discouraging, so that it should possibly be detached from the task of coping with inconsistencies inherent in the input data. These inconsistencies could probably already be resolved in an earlier stage of the analysis system.

Apart from these direct improvements to the developed approaches, the analysis performance may be greatly enhanced by incorporating information on the basis of other audio features. This may even allow a characterisation of non-repeating musical sections or a more detailed characterisation of repeating sequences of musical sections. Furthermore, the analysis of music that does not exhibit clear harmonic or melodious patterns which are presently preferred by the analysis system would surely improve. To this end, onset and timbre features may be employed.

94

A Pop Data Set

Table A.1: Musical pieces in the POP data set. For each piece, the F-measure from the second evaluation procedure is provided for three tested configurations: 1 Hz (denoted as 1 in the column header), 2 Hz with automatic threshold selection (2T), and the multiresolution approach with 1 and 2 Hz (12) (cf. chapter 5).

Artist	Title	F-measure		
		1	2T	12
a-ha	Take On Me	0.87	0.94	0.95
Alanis Morissette	Head Over Feet	0.63	0.71	0.60
Alanis Morissette	Thank U	0.62	0.52	0.39
Beastie Boys	Intergalactic	0.74	0.50	0.69
Björk	It's Oh So Quiet	0.80	0.73	0.88
Black Eyed Peas	Cali To New York	0.90	0.77	0.82
Britney Spears	...Baby One More Time	0.55	0.75	0.64
Chicago	Old Days	0.96	0.98	0.98
Chumbawamba	Tubthumping	0.53	0.65	0.46
Deus	Suds And Soda	0.57	0.69	0.67
Eminem	Stan	0.78	0.50	0.58
Gloria Gaynor	I Will Survive	0.65	0.69	0.84
Madonna	Like A Virgin	0.91	0.90	0.89
Michael Jackson	Bad	0.80	0.88	0.71
Michael Jackson	Black Or White	0.58	0.59	0.67
Nick Drake	Northern Sky	0.64	0.74	0.80
Nirvana	Smells Like Teen Spirit	0.59	0.82	0.58
Norah Jones	Lonestar	0.87	0.90	0.85
Oasis	Wonderwall	0.85	0.75	0.68
Portishead	Wandering Star	0.78	0.88	0.85
Prince	Kiss	0.56	0.43	0.57
R.E.M.	Drive	0.66	0.66	0.61
Radiohead	Creep	0.76	0.60	0.60
Seal	Crazy	0.46	0.53	0.50
Simply Red	Stars	0.86	0.91	0.77

Artist	Title	F-measure		
		1	2T	12
Sinead O'Connor	Nothing Compares 2 U	0.69	0.54	0.63
Spice Girls	Wannabe	0.77	0.84	0.59
The Clash	Should I Stay Or Should I Go	0.61	0.70	0.80
The Cranberries	Zombie	0.50	0.60	0.57
The Monkees	Words	0.79	0.84	0.96
The Beatles	A Day In The Life	0.85	0.89	0.89
The Beatles	All I've Got To Do	0.79	0.84	0.86
The Beatles	All My Loving	0.86	0.96	0.95
The Beatles	Anna (Go To Him)	0.91	0.88	0.83
The Beatles	Being For The Benefit Of Mr. Kite	0.89	0.92	0.89
The Beatles	Devil In Her Heart	0.84	0.84	0.87
The Beatles	Don't Bother Me	0.79	0.88	0.89
The Beatles	Fixing A Hole	0.79	0.89	0.97
The Beatles	Getting Better	0.66	0.83	0.65
The Beatles	Good Morning, Good Morning	0.68	0.84	0.71
The Beatles	Hold Me Tight	0.88	0.93	0.89
The Beatles	I Saw Her Standing There	0.79	0.89	0.74
The Beatles	I Wanna Be Your Man	0.91	0.96	0.81
The Beatles	It Won't Be Long	0.91	0.89	0.94
The Beatles	Little Child	0.74	0.64	0.66
The Beatles	Lovely Rita	0.00	0.38	0.38
The Beatles	Lucy In The Sky With Diamonds	0.89	0.89	0.89
The Beatles	Misery	0.85	0.96	0.94
The Beatles	Money (That's What I Want)	0.75	0.66	0.68
The Beatles	Not A Second Time	0.97	0.98	0.98
The Beatles	Please Mister Postman	0.51	0.69	0.50
The Beatles	Roll Over Beethoven	0.66	0.79	0.46
The Beatles	Sgt. Pepper's Lonely Hearts Club Band	0.83	0.89	0.90
The Beatles	Sgt. Pepper's Lonely Hearts Club Band (Reprise)[1]	-	-	-
The Beatles	She's Leaving Home	0.93	0.91	0.91
The Beatles	Till There Was You	0.91	0.93	0.96
The Beatles	When I'm Sixty-Four	0.94	0.98	0.98
The Beatles	With A Little Help From My Friends	0.92	0.97	0.96
The Beatles	Within You Without You	0.79	0.63	0.71
The Beatles	You Really Got A Hold On Me	0.82	0.94	0.93

---

[1]This musical piece did not contain an annotated cluster.

B Synchronisation Data Set

Table B.1: Classical musical pieces in the SYNCHRONISATION data set. For each piece, the F-measure from the second evaluation procedure is provided for three tested configurations: 1 Hz (denoted as 1 in the column header), 2 Hz with automatic threshold selection (2T), and the multiresolution approach with 1 and 2 Hz (12) (cf. chapter 5).

Musical Piece	Interpretation	F-measure		
		1	2T	12
Ludwig van Beethoven, Piano Sonata No. 17 "Sturm-Sonate" (op. 31:2)	Barenboim	0.99	0.99	0.99
	Gilels	0.99	0.99	0.99
	Pollini	1.00	0.99	0.99
Ludwig van Beethoven, Symphony No. 5 in C minor (op. 67:1)	Bernstein	0.99	0.96	1.00
	Karajan	0.95	0.90	0.90
	Kegel	1.00	1.00	1.00
	Scherbakov [Liszt Version]	0.93	0.94	0.94
	Sawallisch	0.94	0.86	0.99
Johannes Brahms, Hungarian Dances No. 5 in G minor (Allegro)	Ormandy	0.72	0.89	0.88
	Scholz	0.84	0.96	0.88
Johannes Brahms, Hungarian Dances No. 6 in D major (Vivace)	Bernstein	0.83	0.86	0.83
	Scholz	0.78	0.76	0.72
Antonín Dvořák, Symphony No. 9 "From the New World" (op. 95)	Francis	0.97	0.93	0.97
	Maazel	0.97	0.82	0.96
Edvard Grieg, Peer Gynt Suite No. 1 (op. 46)	Beecham	0.55	0.65	0.52
	Gunzenhauser	0.55	0.48	0.60
	Karajan	0.53	0.53	0.49
Felix Mendelssohn, A Midsummer Night's Dream, Wedding March	Levine	0.82	0.86	0.68
	Tate	0.75	0.78	0.85

Musical Piece	Interpretation	F-measure		
		1	2T	12
Dimitri Shostakovich, Suite for Variety Stage Orchestra, Waltz 2	Chailly	0.92	0.93	0.91
	Yablonsky	0.86	0.79	0.71
Antonio Vivaldi, Concerto No. 1, "La primavera" (Spring) - Allegro (RV 269, op. 8:1)	Mae	0.42	0.47	0.44
	Nishizaki	0.50	0.55	0.74
	Perlman	0.36	0.66	0.68
	Zukerman	0.43	0.45	0.55

Table B.2: Popular musical pieces in the SYNCHRONISATION data set. For each piece, the F-measure from the second evaluation procedure is provided for three tested configurations: 1 Hz (denoted as 1 in the column header), 2 Hz with automatic threshold selection (2T), and the multiresolution approach with 1 and 2 Hz (12) (cf. chapter 5).

		F-measure		
Artist	Title	1	2T	12
The Beatles	Help!	0.89	0.84	0.88
The Beatles	Help! [live]	0.74	0.88	0.88
The Beatles	Hey Jude	0.75	0.75	0.65
The Beatles	Hey Jude [studio run-through]	0.86	0.92	0.94
The Beatles	Strawberry Fields Forever	0.78	0.81	0.83
The Beatles	Strawberry Fields Forever [Take 1]	0.89	0.83	0.88
The Beatles	Strawberry Fields Forever [Take 7 and Edit Piece]	0.84	0.92	0.97
Ben Harper	Strawberry Fields Forever	0.74	0.78	0.80
The Beatles	Yesterday	0.91	0.91	0.95
The Beatles	Yesterday [live]	0.85	0.85	0.85
Paul McCartney	Yesterday	0.89	0.88	0.95
Buffalo Springfield	For What It's Worth	0.90	0.70	0.53
Shantel & Sergio Mendes & Brasil' 66	For What It's Worth	0.31	0.50	0.60
Public Enemy	He Got Game	0.63	0.58	0.58
Nena	Irgendwie, Irgendwo, Irgendwann	0.78	0.89	0.90
Nena feat. Kim Wilde	Anyplace, Anywhere, Anytime	0.88	0.93	0.88
Jan Delay	Irgendwie, Irgendwo, Irgendwann	0.92	0.93	0.93
Dusty Springfield	The Look Of Love	0.94	0.97	0.97
Diana Krall	The Look Of Love [live]	0.56	0.65	0.50
Stevie Wonder	You And I	0.61	0.78	0.65
Abbey Lincoln	You And I	0.00	0.53	0.00
Gloria Gaynor	I Will Survive	0.64	0.80	0.86
Cake	I Will Survive	0.76	0.67	0.62
Carl Carlton	Everlasting Love	0.67	0.70	0.88
Jamie Cullum	Everlasting Love	0.70	0.72	0.74

Artist	Title	F-measure		
		1	2T	12
Radiohead	High & Dry	0.76	0.70	0.79
Jamie Cullum	High & Dry [live]	0.74	0.72	0.62
Kylie Minogue	Hand On Your Heart	0.39	0.81	0.57
José González	Hand On Your Heart	0.92	0.94	0.94
Barry Manilow	Can't Take My Eyes Off You	0.90	0.81	0.91
Lauryn Hill	Can't Take My Eyes Off You	0.58	0.65	0.89

Bibliography

[1] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Springer, 2000.

[2] J.M. Barbour. *Tuning and Temperament: A Historical Survey*. Dover Publications, 2004.

[3] M.A. Bartsch and G.H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *Multimedia, IEEE Transactions on*, 7(1):96–104, Feb. 2005.

[4] M.J. Bastiaans. Gabor's expansion of a signal into Gaussian elementary signals. *Proceedings of the IEEE*, 68(4):538–539, 1980.

[5] J. Blume. *6 Steps to Songwriting Success: The Comprehensive Guide to Writing and Marketing Hit Songs*. Billboard Books, 2004.

[6] W. Chai. Structural analysis of musical signals via pattern matching. *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, 5, 2003.

[7] W. Chai. Automated Analysis of Musical Structure. Master's thesis, Massachusetts Institute of Technology, 2005.

[8] R.B. Dannenberg and N. Hu. Discovering Musical Structure in Audio Recordings. *Music and Artificial Intelligence: Second International Conference, IC-MAI 2002, Edinburgh, Scotland, UK, September 12-14, 2002: Proceedings*, pages 43–57, 2002.

[9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[10] J. Foote. Visualizing music and audio using self-similarity. *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 77–80, 1999.

[11] D. Gabor et al. Theory of Communication. 1946.

[12] M. Goto. SmartMusicKIOSK: music listening station with chorus-search function. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 31–40, New York, NY, USA, 2003. ACM. ISBN 1-58113-636-6.

Bibliography

[13] M. Goto. AIST Annotation for the RWC Music Database. *7th International Conference on Music Information Retrieval (ISMIR 2006)*, October 2006.

[14] R. Jackendoff and F. Lerdahl. The capacity for music: What is it, and what's special about it? *Cognition*, 100(1):33–72, 2006.

[15] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[16] A. Klapuri and M. Davy. *Signal Processing Methods for Music Transcription.* Springer, 2006.

[17] M. Levy and M. Sandler. Structural Segmentation of Musical Audio by Constrained Clustering. *Audio, Speech, and Language Processing, IEEE Transactions on [see also Speech and Audio Processing, IEEE Transactions on]*, 16(2): 318–326, 2008.

[18] L. Lu, M. Wang, and H.J. Zhang. Repeating pattern discovery and structure analysis from acoustic music data. *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 275–282, 2004.

[19] M. Minsky. Music, Mind, and Meaning. *Computer Music Journal*, 5(3):28–44, 1981.

[20] E.W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.

[21] M. Müller. *Information Retrieval for Music and Motion.* Springer, Berlin, 2007.

[22] M. Müller and D. Appelt. Path-constrained Partial Music Synchronization. In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2008), to appear*, 2008.

[23] M. Müller and F. Kurth. Towards Structural Analysis of Audio Recordings in the Presence of Musical Variations. *EURASIP Journal on Applied Signal Processing*, 2007(Article ID 89686):18 pages, January 2007.

[24] N. Otsu et al. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[25] J. Paulus and A. Klapuri. Music structure analysis by finding repeated parts. *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 59–68, 2006.

[26] G. Peeters. Sequence Representation of Music Structure Using Higher-Order Similarity Matrix and Maximum-Likelihood Approach. pages 35–40, 2007.

[27] C. Rhodes and M. Casey. Algorithms for determining and labelling approximate hierarchical self-similarity.

[28] T.W. Ridler and S. Calvard. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man and Cybernetics*, 8(8):630–632, 1978.

[29] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13:146–165, 2004.

[30] M.I. Shamos and D. Hoey. Geometric intersection problems. *Proc. 17th Annu. IEEE Sympos. Found. Comput. Sci*, pages 208–215, 1976.

[31] Y. Shiu, H. Jeong, and C.C.J. Kuo. Similarity matrix processing for music structure analysis. *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 69–76, 2006.

[32] C.J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann Newton, MA, USA, 1979.

[33] Y. Zhao and G. Karypis. Clustering in Life Sciences. *Functional Genomics: Methods and Protocols*, 2003.