technische universität
dortmund

Department of Computer Science

Master's Thesis

Constraint-based rearrangement of music

Daniel Stoller

August 31, 2015

Supervisors:

Prof. Dr. Heinrich Müller, Chair of Computer Graphics

Dr. Igor Vatolkin, Chair of Algorithm Engineering

Department of Computer Science
TU Dortmund

Contents

Mathematical Notation

Notation	**Meaning**
$\mathbb{N}$	Set of natural numbers $1, 2, 3, \ldots$
$\mathbb{N}_0$	Set of natural numbers $0, 1, 2, 3, \ldots$
$\mathbb{R}$	Set of real numbers
$\mathcal{M} = \{m_1, \ldots, m_N\}$	Set $\mathcal{M}$ of N elements m_i
$\mathbf{p}$	Vector
p_i	Element i of the vector
$\mathbf{p}^{\text{name}}$	Vector identified by ”name”
p_i^{name}	Element i of the vector identified by ”name”
$\mathbf{p}_k$	k-th vector of a series of vectors
$p_{k,i}$	Element i of the k-th vector
$\mathbf{A}$	Matrix
$A_{i,j}$	Element in row i and column j of matrix $\mathbf{A}$
$\mathbf{A}^{\text{name}}$	Matrix identified by ”name”
$A_{i,j}^{\text{name}}$	Element in row i and column j of matrix $\mathbf{A}^{\text{name}}$

1. Introduction

This introductory section will provide the context necessary for understanding the relevance of the contributions of this thesis. The importance of music and its rearrangement in particular will be discussed and the structure and main results of the thesis will be presented.
At first, we will begin by introducing the task of rearranging music and its potential applications in the following section 1.1. Afterwards, the scope of this thesis along with the main contributions to the field of music rearrangement and the most important results of our work will be presented in section 1.2. The third section 1.3 will provide a brief overview of this thesis by summarising its main sections.

1.1. Music and music rearrangement

Music constitutes an integral part of our cultural life and is nearly omnipresent. It pervades many areas of life, ranging from the use in advertisements and films to concerts and in general any form of personal consumption. The dawn of the digital era rapidly increased the accessibility of music and also made sharing and redistributing existing and new musical pieces significantly easier. Additionally, it allowed for computer-assisted editing of music, increasing the efficiency of many music editing tasks. A few examples of many include adding reverberation effects by simulating certain types of rooms or automatically correcting the sections of a vocal recording where the vocalist did not quite hit the right note.
In recent years, the ever increasing processing power of computers in combination with research in the field of music informatics steadily opened up new possibilities, allowing for the accomplishment of more and more sophisticated tasks. Most of these tasks are traditionally performed manually by a professional audio engineer using an expert audio editing software and often require a large amount of time. Automatic or at least semi-automatic solutions for these tasks would not only save time, but also potentially allow less experienced users to perform them.
Automatically adapting the length of a musical piece to a specific desired duration

is one of these tasks and presents an active topic in research. A possible application scenario is concerned with editing a music piece so that it has the same duration as an existing video and can be used as musical accompaniment. Another relevant scenario involves creating a short preview of a music piece for customers to listen to before possibly committing to a purchase of the whole piece on a commercial website. In video game design, dynamically adapting the duration of a music piece would be useful – sometimes, the background music of a game level should last as long as the player stays in this level, which could be for an arbitrarily long time that is not known in advance.

Rearranging an existing piece of music by changing its high-level structure constitutes another task that could also be combined with the previously mentioned change of duration. A system capable of executing such a task would allow even non-professional users to ”remix” an arbitrary music piece in a short amount of time. For example, a typical Western pop song consisting of segments that each serve the function of either a verse or a chorus could be converted into a song that only incorporates elements from the chorus segments. Additionally, the possibility of adjusting the duration of these segments would make it possible to increase the length of the verses while not making changes to the rest of the structure.

A whole range of additional requirements for the resulting track could arise in a real-world setting, for example selecting which parts of an original piece have to be used or not used for the rearranged piece. Excluding different parts from the produced results would allow for the efficient creation of instrumental pieces from tracks containing vocals. Another variant of this requirement would involve ”protecting” some parts, thereby either completely removing them from the result or forcing them to be played without modification. Such a concept would be useful when using music as an accompaniment for a film, because some parts of the audio may need to stay synchronised with the video despite the rearrangement. These parts could be marked as protected in order to prevent them from being modified.

Many more constraints for the produced result could be conceived, of which some could be very useful for a range of different scenarios. However, implementing each of these possibly contradicting constraints into a system requires a large amount of time. In this thesis, we will therefore focus on a certain subset of all theoretically possible constraints and implement them in a new system for music rearrangement.

The following section will outline the scope of the thesis, detailing the goal and the aspects that received particular attention. It will also briefly summarise the results achieved with the proposed system for music rearrangement.

1.2. Scope and results of the thesis

This thesis aims to address the problem of automatically rearranging any given music piece based on user-defined constraints, which is explicitly stated in section 3.1, by proposing a novel music rearrangement system. We did not focus on synthesising music from basic elements of sound, but instead on using certain pieces of the original to assemble a new song in such way that it fulfils the given constraints. These constraints include a target duration set by the user in order to specify the desired length of the new music piece. We integrated algorithms capable of scaling the duration of a music piece (creating the impression it is played in a faster or slower tempo than in the original music piece) in order to produce music pieces that fulfil this target duration very accurately, if desired by the user. Another important user constraint supported by our music rearrangement system defines a requirement for the structure of the generated output track.

Many of the recent methods for music rearrangement concatenate sections of the original song to form the new track while trying to minimise the perceptibility of the resulting transitions from one section to the next. Instead of restricting the range of possible inputs to music tracks of a certain genre like dance music in the approach from Wenger [42], the goal of this thesis was to develop a system suitable for any music piece regardless of genre. Our music rearrangement system is based on the already existing method from Wenner [48], but improves it in several areas. With an extensive evaluation of Wenner's method in section 2.4, we contributed a detailed examination of the state of the art regarding music rearrangement, revealing a range of different problems. Some of these problems were solved by our proposed system, while others remain as guidance for the direction of possible future work.

The system uses algorithms to extract the positions of the *beats*, which represent the underlying pulse of the track, and the structure of the original piece. Improving these methods was not within the scope of this thesis, as they are subject to other areas of research and implementations from the best available methods developed by other authors can be integrated into the system.

In contrast, solving the frequent occurrence of irritating changes in loudness at the transitions between two concatenated sections received particular attention. Our solution tackles the problem from two different angles. At first, we aimed to avoid concatenating sections exhibiting large loudness differences in order to prevent such problematic transitions from emerging in the result. Secondly, in case no suitable alternative transitions can be found or other transitions do not offer the same musical quality concerning other aspects, these problematic transitions can also be processed with the loudness equalisation method from section 4.4.2 targeted at smoothing the loudness change so it does not occur instantly, but over a time frame of up to a few seconds.

To find the optimal arrangement of sections of the original piece fulfilling the user constraints, an algorithm based on *dynamic programming* is employed by Wenner [48] with an asymptotic runtime complexity that is quadratically dependent on the length of the input track as well as linearly dependent on the duration of the output track. Consequently, rearranging longer tracks often leaves the user waiting for more than ten seconds to see the assembled result. A typical workflow is comprised of repeatedly changing the user constraints and listening to the result after each change in order to receive the best possible music piece. As a result, such long waiting times are severely detrimental to the user's productivity and the interactivity of the interaction with the system. In an effort to reduce these runtimes, we adapted the well-known *A* algorithm* [20] to the problem of finding an optimal arrangement of excerpts of the original piece in section 4.3.5 and also developed specific heuristics intended to provide an additional speed-up.

Because rhythm is a very important aspect of music and concatenating two sections of a music piece can result in rhythmical discontinuities, an approach to synchronise both excerpts to their common underlying rhythmical pulse is presented in section 4.4.1.

An important change to how a specific structure is enforced on the result was made to the original method from Wenner [48]. The approach in this thesis presented in section 4.3.6 is configurable by the user through tolerance parameters controlling how strictly the structure is enforced. Our algorithm trades the accurate fulfilment of the structural constraint for a potentially better sounding result. It allows a certain deviation of the resulting structure from the desired one, but in return also extends the space of possible solutions for assembling the piece, which often contains a significantly better solution with transitions that are rated as less perceptible.

The developed components of the music rearrangement system are evaluated in section 5, either by automatically executing a large number of rearrangements and analysing relevant metrics or with a listening study, where excerpts of the produced music tracks are rated by participants. Comparing the performance of the proposed variant of the A* algorithm designed to find the optimal arrangement to the performance of the dynamic programming approach from Wenner [48], we conclude that our algorithm outperformed the dynamic programming approach with regards to the average runtime in almost all cases. Using input tracks with durations of under three minutes presented the only exception, where the dynamic programming approach was slightly faster on average, but this difference is negligible for the user as both algorithms typically required less than one second. On the other hand, our modified A* algorithm scaled better with the duration of both the input and the output tracks, completing its computation significantly earlier than the dynamic approach especially for long input tracks lasting seven minutes and more.

The method proposed for enforcing a specific structure on the result with tolerances introduced in section 4.3.6 also proved to be a success – it lead to the selection of better solutions involving transitions estimated as less perceptible, while the structure of the produced results deviated only slightly from the desired structure defined by the user. Under the assumption that the measures used for estimating the transition quality accurately reflect the perception of the listener, increasing the segmentation tolerances consequently leads to better sounding output tracks.

In the conducted listening study, the participants were asked to rate audio snippets containing transitions from one section of the original piece to another regarding their *loudness continuity*, that is, how smooth the loudness changes over time and if any irritating loudness changes occur. These ratings significantly correlate with the estimations of the transition quality regarding loudness provided by our music rearrangement system, as shown in section 5.2.1. This correlation demonstrates that transitions lacking in loudness continuity can often be automatically identified and subsequently avoided when searching for the optimal solution. Additionally, section 5.2.2 confirms the ability of the proposed loudness equalisation method to improve the quality of the transitions on average. On the other hand, the synchronisation method also devised with the intention of improving an aspect of the transition quality, namely rhythm, did not have a consistently positive effect when applied to the audio excerpts. This is shown by statistically analysing the average ratings of the participants in section 5.2.2.

The user interface of our system shown in section 4.6 represents another contribution to the field of music rearrangement. It enables the user to rearrange the original music piece in a very intuitive manner, as most constraints can be efficiently entered using only a mouse without the need of typing in complicated sequences of commands.

In the following section, we will outline the structure of this thesis by summarising all of the upcoming sections. Some of these introduce system components that were not yet discussed in this section, but were also targeted at improving the performance of the music rearrangement system.

1.3. Structure of the thesis

At first, we will lay the foundations required for this thesis in section 2, beginning with the relevant aspects of music theory in section 2.1. Afterwards, we will present the current state of the art in the field of music rearrangement in section 2.2 and review the currently existing approaches. Following the decision to base our method on one of these approaches, we will extensively evaluate the selected approach with a preliminary study presented in section 2.3 to ascertain possible areas of improvement, which will be explored in the subsequent section 2.4.
Before discussing details of the proposed music rearrangement system, we will provide an overview in section 3, comprised of a more formal description of music arrangement as a problem in section 3.1 and an introduction to our system designed to solve this problem in section 3.2.
Because the system components are only briefly explained in section 3.2, the following section 4 describes them more thoroughly. Each of the subsections is dedicated to one system component and their order corresponds to the position of the respective component in the system's processing pipeline. The first part of this pipeline constitutes a beat tracking system presented in section 4.1 to detect the locations of beats in the original music piece. Afterwards, the preprocessing stage described in section 4.2 involves extracting musical information based on the previously detected beats and has to be performed only once for an input track. Using this information, the path optimisation stage described in section 4.3 computes the optimal solution from all available solutions that fulfil the user-defined constraints. This optimal solution is equivalent to a description of how the output track should be assembled, that is, which sections from the original piece are used and in which order. In the jump optimisation stage introduced in section 4.4, this description is refined and certain audio processing steps are performed on the output signal, both in order to achieve a higher transition quality. If desired by the user, the final output signal is scaled so it has exactly the desired duration in section 4.5. Interaction with the music rearrangement system is made possible by the user interface presented in section 4.6.
Subsequent to the explanation of the proposed music rearrangement system in section 4, its evaluation will take place in section 5. Its goal was to test whether the implementations of all major components actually provide the intended improvements and it employed both an automatic evaluation procedure in section 5.1 and a listening study in section 5.2.
Finally, we will draw conclusions from our work in section 6, summarising the capa-

bilities, but also the limitations of our system and possible approaches for improvement as part of future work.

The appendix A provides an overview of the parameters introduced throughout the thesis and also describes the databases containing music pieces that were used for evaluation purposes. Additionally, it features details of experimental results obtained during the evaluation.

Important concepts occurring on multiple occasions throughout this thesis will be introduced with their names written in **bold**. They are defined as precisely as possible and are generally included in the index as a means of reference. Other concepts from external sources not explicitly defined in this thesis will be written in *italic* and feature references to the exact definitions, if necessary. Italics will also be used for mentioning concepts that are not yet explicitly defined, but will be at a later point.

2. Background and related work

This section will present the knowledge required for understanding the proposed music rearrangement system in section 4. At first, the relevant aspects and definitions related to music theory will be presented in section 2.1. Afterwards, section 2.2 will give an overview of the existing approaches to rearranging music that leverage some of these aspects. We will select one of these approaches as a starting point for our own system in section 2.3 and describe a preliminary study conducted to extensively evaluate it. Finally, the results of this preliminary study uncovering different areas of improvement are presented in section 2.4.

2.1. Music theory

In order to understand the currently existing approaches outlined in section 2.2 and the method developed in this thesis, some concepts from music theory will be presented in this section providing the basis on which these methods work on.

Musical structure In Western popular music, a song can be structurally divided into a number of *modules* [31]. These modules are categorised according to the function they fulfil as follows. A *verse* module primarily features a unique lyric content, while *chorus* modules often contain the same lyrics and exhibit a higher musical intensity relative to the verse modules. *Bridge* modules on the other hand present a contrast to these modules and serve as a transition from one module to another, making the listener anticipate the following module. The overall structure then emerges from the temporal arrangement of the different modules. For example, a common arrangement found in modern pop music is to use a verse and a chorus twice in this order, followed by a bridge module that lets the listener anticipate the final chorus module.
In classical music, musical structure is a much more complex phenomenon [4] and will not be explored in greater detail. In this thesis, it is sufficient that the concept of different musical parts with specific functions occurring in some order, which was

applied to Western pop music above, can also be used to describe classical music on a general level.

The more technical term **segmentation** is defined as a description or annotation of the musical structure present in a song and represents a core concept in this thesis. It consists of a sequence of **segments**, corresponding to the notion of modules, each with a point in time where the segment begins (when it ends, the next segment begins). A **segment transition** from one segment to the next is the point in time where the first segment ends and the next one begins. Furthermore, every segment is assigned to a **cluster** in such a way that all segments serving the same function (for example, all verse modules) belong to the same cluster. This simplified concept of musical structure can be applied to a wide range of musical genres and is still powerful enough to provide many opportunities for rearranging music, of which some are mentioned in section 1.1.

Tempo and rhythm Music is a time-dependant phenomenon, where the time is often divided into short periods of the same length, indicated by audible pulses called **beats**, allowing musicians to synchronise their play [38]. The **tempo** of a piece is described by the amount of beats per minute (bpm). Musical events like the beginning of a note or a snare drum hit tend to occur together at beat positions, but they are also allowed to take place in between the beats. Depending on some characteristics of these events, e.g. how loud or how powerful they are perceived, the corresponding beats can be categorised into **strong beats** and **weak beats**.

A pattern of strong and weak beats is called **meter** and very often repeats itself throughout Western music pieces, so beats can be subdivided into groups called **measures**, where one measure contains a number of those repetitions. In pop music for example, a meter with a strong beat followed by a weak beat is common along with measures containing four repetitions of this meter, resulting in a measure consisting of four strong and four weak beats. Although tapping to the beat usually occurs along the strong beats of a measure, it can also be performed on other **metric levels** by including the weak beats in between the strong beats or skipping a constant number of strong beats after every tap.

Time signatures are noted down as a fraction describing in the numerator how many beats fit into measure and in the denominator what type of note gets a beat, e.g. a $\frac{3}{4}$ time signature contains three beats per measure where each beat is fully occupied by a quarter note. Western music and in particular Western pop music very often uses a $\frac{4}{4}$ time signature.

Instrumentation Music employs a wide variety of different instruments to produce musical tones. These tones can vary in their **pitch**. We conceptualise pitches as ”distinct sonic entities” and ”we mentally represent them as a series of points occupying higher or lower, intervallically defined positions on an imaginary, quasi-spatial, vertically aligned two-dimensional continuum” [5]. Every instrument has a specific range of pitches it is capable of producing – while a flute covers high-pitched notes, a bass guitar does not, but can instead generate very low-pitched notes.

Even if two different instruments cover a similar range of pitches, an experienced musician can distinguish the sound of both instruments even when they play the exact same note due to a psychoacoustical effect named **timbre** [5], often described as the quality of a tone. Timbre emerges in part from the presence of additional frequencies (*harmonics*) in the sound whose distribution varies depending on the instrument used.

Loudness A note that is played can not only vary in pitch and in timbre, but also in **loudness** that can be defined as follows: ”When a sound or noise of any quality or structure impinges upon the human ear, the magnitude of the resultant sensation is termed the loudness.” [30] Therefore, loudness is a psychoacoustical unit and should not be confused with the term **sound pressure** as an objective, physical measure of sound strength. Although loudness primarily depends on the sound pressure, it is also influenced by the frequencies the sound is composed of. *Equal-loudness contours* demonstrate how the loudness of a tone with a specific, constant sound pressure varies depending on its frequency [21].

Loudness plays a very important role in music. A slow increase in loudness creates excitement, anticipation or tension, while a slow decrease can indicate the end of a musical idea or a whole song. Differences in loudness like a high loudness in chorus modules and a low loudness in verse modules can help to structure the musical piece by emphasising specific parts.

Consequently, the loudness changes in a music piece are often expected by the user or occur over a longer period of time so they are not irritating to the listener. We introduce the concept of **loudness continuity** for this case and this continuity is violated whenever a sudden, unexpected change in loudness occurs while listening to a song. This happens particularly often in the area of digital music editing, when parts of a song with different loudness are cut together without an appropriate transition.

2.2. Existing approaches

Several approaches have been proposed to automatically change or rearrange a music piece according to a given set of user constraints. Because automatically rearranging music is still a relatively new research topic, there is a limited amount of related work available. We will present an overview of this related work in this section.

When only a change in duration of a music piece is required, then this can in principle be achieved very simply by adjusting the playback rate, simultaneously causing an undesired change in pitch. *Timescale-pitch modifications* like WSOLA [43] can be employed to change tempo and pitch independently, so the scaling of duration can be performed without changing the pitch. Although yielding good results for small duration changes, higher scaling factors not only produce sound artefacts, but also alienate the original piece, drastically changing the listening experience to the point where the result has little resemblance with the original. Additionally, one would like to influence the segmentation of the produced result in some way, for example to create ”remixes”, which is not supported by this approach.

While the application of rearrangement methods found in computer graphics dealing with multi-dimensional data like pictures to music as a one-dimensional signal is theoretically possible, it turned out to not work well in practice [48]. Treating the music piece as a sound texture analogous to graphical textures and applying for example *tiling* and *stitching* produces good results for ambient noises like wind and rain, but fails for structured music, often repeating the same short section [33]. *Seam-carving* [2] is another popular approach often used to intelligently retarget images and works by finding and expanding regions of low interest to the viewer and is theoretically also applicable to music. To find the correct points in time to expand or contract the signal a saliency measure is needed, but for music these measures are not robust enough yet [48].

Consequently, research in the last few years focused on another, more promising approach based on concatenating parts of the musical piece in question while minimising the perceptibility of cuts between the sequentially arranged parts by searching for sections with a similar hearing impression. Using an equivalent description from another point of view, this type of approach plays back the original and occasionally ”jumps” from the current point in time to another.

Researchers at TU Braunschweig incrementally built upon such a retargeting approach [46], developing a genetic algorithm [47] and reducing the search space for cut positions to whole measures as identified by a beat tracker [42]. While yielding good results for dance music often exhibiting a strong, regular beat, it does

not necessarily generalise well when applied across many different genres of which some feature changes in tempo, loudness and other musical properties. Furthermore, when entering the desired segmentation of the result, the user is not assisted by an automatic segmentation method and instead has to build it from the ground up on a measure-by-measure basis.
Such an automatic segmentation is instead employed by S. Wenner [48], whose work is similar, but not only targeted at dance music and also contains a wide range of additional features useful for different video editing tasks as well as singing removal or creation of infinite music. Because it uses ”bixels” (a contraction of the words beat and pixel), the musical content between two consecutive beats, for self-similarity calculation and not whole measures, cuts from one count of a measure to a different count can occur. These cuts lead to a measure with an unusual number of beats that sounds rhythmically irritating.
With a user-defined segmentation both approaches suffer from a significantly reduced quality of cuts, as many potentially less perceptible cuts do not lead to a solution satisfying the segmentation constraints.
In summary, the previously proposed methods lack the musical understanding in one or several areas in order to always be able to successfully generate a music piece of high quality. Methods based on synthesis largely fail for structured music and are not able to authentically reproduce the musical style and sounds of different artists, while methods based on concatenating sections of the original piece sometimes suffer from the suboptimal selection of cut positions and from the lack of suitable cut positions in case a music piece progressively changes and does not repeat itself.

2.3. Preliminary study

The two ”jump-based” approaches outlined in the previous section 2.2 presented the most promising basis for our own algorithm. Although the method from Wenger [46] often provides high quality outputs for dance music, the goal of this thesis is to build an all-purpose music rearrangement system without such a genre restriction. In contrast, the algorithm developed by S. Wenner [48] appeared to fit better to the scope of this thesis and was therefore selected as a reference method. In a preliminary study to investigate possible areas of improvement, we reimplemented and intensively evaluated Wenner’s method with a self-built database called ”CC1” (see appendix A.3). It contains 42 Creative Commons songs sourced from the ”Free Music Archive” [49]. The genre and in some cases the subgenre for every song was provided in addition to the audio file and allowed for a selection covering a wide range of different genres. Because the output quality of the method depends on the accuracy of the employed beat tracking system, ground truth data for the beat positions was created for every song by "tapping" to the beat while listening to it in order to eliminate incorrectly detected beats as the cause for suboptimal results.

Every experiment was conducted with the automatically detected beats by the beat tracker from Davis [9] (see section 4.1) and with the ground truth beat data as input. To cover both shortening and lengthening the original piece, all songs were retargeted to 60 and 600 seconds with varying amounts of seconds as tolerance, meaning the resulting length could deviate from the target duration more or less. This leads to the set of configurations listed in table A.8, where the "Setting" column describes the target duration, the tolerance in seconds and whether the ground truth (GT) beat data was used. The resulting cuts in the output were graded with regards to different musical aspects using marks from one (low quality) to ten (high quality). Marks for the tonal quality (T) can be found in the first column, focusing on how continuous the instrumentation sounds at the cut. The second column deals with the rhythmical continuity (R), demanding that the beats are evenly spaced across time so the listener is able to intuitively tap to the beat without issues. A correct preservation of measures (M) is the concern of the third column, penalising "broken measures" containing fewer or more beats than normally expected. In the fourth column, a compositional aspect (C) describes how well the piece summarizes or respectively extends its original and makes deductions for frequent re-occurrences of the same segment. Additional remarks about the jump quality are listed in the last column.

In the following section, the results of this preliminary study are discussed.

2.4. Issues of the selected approach

The results of the preliminary study of Wenner's method in the previous section 2.3 revealed a number of unknown problems (even when using the ground truth beat positions), which will be listed along with problems already mentioned by the author [48]. We will reference the sections in this thesis dealing with some of these problems, while others will remain for potential future work discussed in section 6 due to the time constraints in the context of this thesis.

Frequent repetition of short sections When extending music to 600 seconds, sometimes the algorithm generates a solution that contains the same short excerpt of the original song many times in direct succession. Although this often produces imperceptible cuts, a lot of repetitions are detrimental to a pleasing experience for the listener. In addition, a large part of the original is not used in the end result, so it does not "summarise" the original piece very well. As seen in table A.8 and heard in audio example 2.1, extending the song "Barbarian" from "Pierlo" for example caused many repetitions of the same short section. How to penalise the jumps responsible for this effect so that jumps not producing this problem are used instead is discussed in section 4.3.4.

Audio 2.1.: Audio excerpt of "Barbarian" from "Pierlo" that was repeated over 40 times after extending it to 600 seconds during evaluation.

Bixel-based segmentation enforcement The implementation of *structure-aware retargeting* in the original work [48], where the currently desired segment is enforced on a bixel-by-bixel basis, severely restricts the space of solutions, because there is a fixed position in the path at which the cluster of the current segment has to change. Although this results in pieces with segmentations that match the desired segmentation with a high precision of up to a bixel's length, the cuts tend to sound significantly worse than without the segmentation constraint. In section 4.3.6, we propose a method to enforce a segmentation with a user-defined level of tolerance, so that slight deviations of the resulting segmentation from the desired segmentation are allowed and also exploited, if they lead to a result of higher quality.

Disregard for time signatures As already mentioned in section 2.2, the algorithm does not always respect the measure, resulting in a different amount of beats in the measure containing the cut. This issue is especially noticeable with music genres featuring a constant time signature and a clear indication of the current position in the measure, e.g. pop music. Taking the standard $\frac{4}{4}$ time signature found in most Western music as an example, jumps leading to measures with an uneven number of beats like seven are particularly jarring, whereas extending the measure by two beats is less striking. "Gonna Make It Through This Year" from "Great Lake Swimmers" from the database CC1 listed in table A.8 also features this time signature and is a good example for this type of problem. An excerpt of the result of retargeting the song to 60 seconds in audio example 2.2 demonstrates a disregard for the time signature despite an almost perfect rhythmical aspect. Although tapping along to the result works fine, counting the beats according to a $\frac{4}{4}$ time signature does not end with a count of one for the final beat, as one would expect. A potential solution

Audio 2.2.: Excerpt of ”Gonna Make It Through This Year” after retargeting to 60 seconds using the ground truth beats. The normal rule of four beats per measure is broken at the end by finishing the piece not on the first, but on what would normally be the second beat of the measure.

involves detecting the time signature of the piece and avoiding jumps between beats with different respective positions in their measures.

Ignorance of melody Melody represents another important part of music and plays in an important role when searching for self-similarity, as it often contains repetitions of the same melodic elements. Currently, the measure used for computing the similarity of two bixels is computed with a feature describing the timbre (see section 4.2.1 for more details), but melody is not taken into account. Although not many jumps produced irritating changes in melody, additional features could be added to further increase the robustness of the similarity measure and thus overall jump quality.

Cutting off vocals Another issue noted by both Wenger [42] and Wenner [48], which also occurred for every experiment annotated with ”Vocals” in table A.8, concerns tracks with vocals. Audio example 2.3 demonstrates this problem with the song ”Amazing Grace” when shortening it to 60 seconds with a tolerance of three seconds. Because humans are especially sensitive to the human voice, jumping

Audio 2.3.: Beginning of the short version of ”Amazing Grace” with three seconds tolerance. Just as the male singer begins with his verse, the jump to a choral part interrupts it.

from or to a point in time where vocals are present produces an imperceptible cut only in rare cases. Even when the jump successfully transitions between two identically sung notes, it sometimes combines the beginning of a verse with the end of another, alienating the original lyrics and producing meaningless sentences. The original work by Wenner provided a work-around by manually marking the parts with vocals and forcing the algorithm to use them without any modification or not at all, but an automatic solution that first detects the presence of vocals and then avoids jumps occurring mid-sentence would be significantly less time-consuming.

Changes in tempo Although the database CC1 contains mostly music with a constant tempo, there are also a few songs with either sudden tempo changes or with gradually changing tempo like the punk song ”Dead Elements” from ”Angstbreaker”, of which an excerpt is presented in audio example 2.4. Additionally, some

Audio 2.4.: Excerpt of ”Dead Elements” exhibiting a slight, gradual tempo increase in the first part and a sudden tempo change to a faster tempo at the beginning of the guitar solo.

pieces like ”Campaign Speech” from ”Convey” slow down when nearing the end. The currently proposed methods assume a regular beat with an equal duration occupied by every “basic building block“ for synthesis (whole measures in case of Wenger, bixels in case of Wenner), leading to issues when tempo changes are present. Unpredictable, very noticeable jumps from slow parts to fast parts and vice versa are especially common for such input tracks and significantly degrade the output quality. Taking the tempo difference between the segments to be concatenated into account when considering which jumps to include in the result could be a solution. Moreover, if such a jump is used despite this adjustment, a timescale-pitch modification with a varying tempo parameter may be used around the resulting cut position to smoothen the sudden tempo change or fully correct smaller tempo differences.

Inaccuracies caused by differently sized bixels Different distances between neighbouring beats also call for a modification of the mentioned state-of-the-art algorithms as they implement most of their functions on a measure-by-measure or bixel-by-bixel basis, including the implementation of segmentation constraints. The length of a music piece containing a specific number of such building blocks can vary depending on which blocks are selected from the original, especially when tempo changes lead to building blocks of very different length. Consequently, algorithms enforcing a change in segment clusters after a specified number of blocks can produce results with transition times that deviate from the user-specification. This is solved as our implementation of segmentation enforcement in section 4.3.6 dynamically computes the current length of the considered solution and determines which segment cluster should be enforced at the moment based on this information.

Additionally, the number of building blocks required for an output with a certain length can only be estimated. Instead of computing the optimal solution only for the best estimate like in Wenner [48] and risking a deviation of the resulting music piece from the target duration, we determine a possible range for the number of building blocks that is likely to be sufficient to produce a solution with the desired duration in section 4.3.2.

Changes in loudness A problem often encountered in the retargeted results pertains to cuts connecting music excerpts differing in loudness. Investigating the cause revealed the very low cost of including those jumps into the solution, leading to many solutions with irritating loudness changes at the cuts. Such cases are annotated in the comments section of table A.8 and one of them is demonstrated in audio example 2.5. We tackle this problem from two different angles as follows. A separate

♫ **Audio 2.5.:** Increasing the duration of "Simple" from "Orbique" without tolerance leads to a sudden decrease in loudness.

feature for loudness is extracted in section 4.2.2 and combined in section 4.3.3 with the original timbre-related feature to also take the loudness differences induced by jumps into account, so that jumps producing an irritating change in loudness are avoided. The other approach outlined in section 4.4.2 does not avoid these problematic jumps in the first place, but modifies the signal itself around the resulting cut in such a way that sudden loudness changes instead occur over a longer period of time to improve the hearing impression.

Sound artefacts due to inaccurate beat positions Even though beat tracking itself is not the focus of attention in this thesis, sometimes small timing inaccuracies in the returned beat positions lead to sound artefacts at jump positions in spite of an almost perfectly tracked beat. Audio example 2.6 demonstrates this issue and features two hi-hat sounds occurring in rapid succession, although only one should be present. The original algorithm already tries to find the most optimal alignment

♫ **Audio 2.6.:** Retargeting "Stormy Blues" from "Arne Bang Huseby" to 600 seconds without tolerance leads to a subtle sound artefact: Experienced listeners may notice the moment where two hi-hat sounds occur almost simultaneously.

of the source and target signal by positioning one signal up to 0.02 seconds earlier or later and computing the sum of squared differences, but fails for timing errors larger than these 0.02 seconds. Therefore, a more general solution is developed in section 4.4.1 to find the best cut position given the jump times, addressing timing inaccuracies around the cut position.

Shifts of sounds in the stereo field A few songs like "Blackroad" from "Tryannic Toy" in audio example 2.7 made heavy use of stereo effects, for example changing the perceived location of a guitar in the stereo field during the song. However, analysis

Audio 2.7.: Excerpt of "Blackroad" after extension to 600 seconds without tolerance. At first, one guitar on the left and on the right side can be heard. Then the song transitions to a single guitar located in the center of the stereo field. Near the end, a jump suddenly cuts this guitar off and instead the two guitars on both sides return.

is performed after converting the input signal to mono and is therefore unaware of any stereo effects, leading to jumps with sudden changes in the perceived sound location of some instruments. For instance, cuts from a guitar heard on the right ear to a guitar on the left ear or a guitar heard from both sides to a guitar heard "inside one's head" can be noticed with a good stereo loudspeaker setup and occur in the song "Blackroad". One approach to alleviate this problem could consist of computing the similarity between different bixels for every channel separately and then combining the resulting measures into one.

Overall, the preliminary study revealed a range of different issues and consequently many areas of improvement.

In the next section 3, we will state the problem of rearranging music more thoroughly and present an overview of a system proposed to solve this problem, while tackling some of the issues mentioned above.

3. Overview

This section will describe the problem of music rearrangement in detail and will introduce a system proposed to solve this problem. Beginning with the following section 3.1, music rearrangement will be formulated as a problem by describing the required and optional inputs along with the desired output. Afterwards, section 3.2 will present an overview of the music rearrangement system we propose to solve this problem.

3.1. Problem statement

Because music rearrangement is a very broad term, we will introduce a more detailed definition of music rearrangement as a problem in the context of this thesis.
First and foremost, music rearrangement always requires an already existing piece of music as one part of the input. The overall goal is to assemble a new music piece from this input track that fulfils a set of user-defined constraints, which represent the other part of the input.

Principle of assembly We restrict the range of possibilities for solving the music rearrangement problem as follows. The output track should generally not be synthesised independently from the input music, but rather contain a series of different sections of the input track to sound as similar to the original song as possible, while still fulfilling the user constraints presented later in this section. However, we also allow for the subsequent processing of a signal constructed in such a manner to further increase its quality.
Consequently, a solution generated by this approach of concatenation is approximately equivalent to playing the original song, but with a list of **jumps** each with a **jump origin** indicating the time at which to change the current play position in the original song and with a **jump destination** defining the position in the original song to which the playback position is changed. A jump whose origin is located before its destination in the original song is called a **forward jump** and decreases

the duration of the output, while a **backward jump** features a destination located before the origin and increases the duration of the final piece. Both types of jumps change the duration by the amount of time between jump origin and destination, which is our definition for the term **jump distance**. The resulting song consequently contains a **cut** for every jump at which the music signal before the jump origin and the signal after the jump destination is joined together.

User constraints Besides the input track, a set of user constraints serve as input when considering the problem of music rearrangement. They pose a restriction to the space of possible outputs and enable the user to design the new music piece to their liking. In the following, we will list a portion of all possible constraints.

The **target duration** constraint is mandatory as input and defines the desired duration along with a minimum and maximum duration the new music piece is allowed to have. Additionally, two parameters define the time points in seconds in the original track at which the output track should start and end, respectively. In the context of the assembly procedure mentioned previously, these start and end positions determine where the playback initially starts and where it ends after performing all jumps.

Optionally, the user should also be able to define the structure of the output, that is, where different parts of the original song are allowed to occur and for how long. When using this constraint, it is necessary for the user to first define a **ground truth segmentation** of the input track by annotating its structure. For example, a typical pop song could be divided into segments each belonging to either the verse or the chorus module. Afterwards, the desired segmentation, called **target segmentation**, is designed by the user. For every point in time, it enforces the usage of audio material from segments that are assigned to a particular cluster in the ground truth segmentation. Following our example regarding a typical pop song, the user would be able to enforce an output containing audio material from the chorus segments in the first thirty seconds and from the verse segments for the remainder of the output track.

Additionally, the optional **importance** constraint allows the user to specify which parts of the input track should be included in the output. More specifically, a degree of importance is defined for every time point in the original song, incentivising the use of some parts of the input track over others.

In principle, the set of user constraints can be extended with many more, mostly optional, constraints to increase the level of control the user has over the output track. The proposed system introduced in section 3.2 implements a number of such constraints in addition to the previously mentioned constraints. These implemented user constraints are listed in table A.1 and will be explained in greater detail in the respective subsections of section 4.

In summary, a system for music rearrangement has to produce a new track using sections of the original piece so that the resulting cuts are as imperceptible to the listener as possible, while fulfilling the user-defined constraints.

3.2. System overview

The music rearrangement system proposed in this thesis is partly based on the approach from Wenner [48] selected in section 2.4, building on some of its core ideas while modifying and extending it to remedy its shortcomings.

One of these core ideas, which also plays a central role in this thesis, is the concept of a **bixel**. It is defined as the musical content located between two consecutive beats. As a result, every song can be divided into a sequence of bixels according to its beat positions. We use bixels as the building blocks of our music rearrangement system after identifying the positions of the beats, that is, the output is in principle assembled by concatenating a series of bixels. In other words, the (infinitely large) set of possible jumps that is considered for a solution is restricted to jumps whose origins and destinations each align with one of the detected beat positions.

In the remainder of this section, we will provide an overview of the proposed music rearrangement system, which can be represented as a processing pipeline, by briefly going over all the stages in this pipeline.

The implementation of the system employs a combination of C++ and MATLAB [25] as programming languages. C++ was selected due to its efficiency and controllability of low-level operations vital for quickly solving the optimisation problem in section 4.3, while MATLAB enables a fast creation of prototypes using statistical and audio processing tools.

The system allows for a wide range of user-defined constraints for the resulting music piece such as duration, segmentation and the usage of specific parts of the original according to an *importance function*. In addition, results with potentially annoying repetitions or cuts with a sudden change in loudness can be avoided with respective parameters. Other features include a fast path optimisation algorithm to find the optimal solution fulfilling the user constraints, a loudness equalisation method designed to restore loudness continuity at cuts and the employment of timescale-pitch modification algorithms for results that fulfil the target duration very accurately.

The general structure of the proposed system is shown in figure 3.1. On an abstract level, it is comprised of two different phases explained in the following. The **analysis** phase is concerned with extracting relevant features from the input song that are used in the subsequent **synthesis** phase to generate a new music piece according to user-defined constraints. The analysis has to be done only once for every input track, while the synthesis is performed on this constant set of data whenever the user changes the constraints, allowing for an interactive rearrangement due to the synthesis phase being completed in a few seconds in most cases (see section 5.1.2 for details on runtimes).

After providing the original music piece, the analysis begins by identifying the locations of the beats with a **beat tracking** system detailed in section 4.1. Alternatively, the beat data can be manually imported from a user-defined external source in the form of a file containing the beat positions with comma-separated values.

The beat positions obtained from the previous stage are then used along with the music piece in the **preprocessing** stage presented in section 4.2 for the extraction of musical features describing the timbre and the loudness of every bixel. To assist the user in defining a ground truth segmentation for the original piece, an automatic segmentation is performed according to the method in section 4.2.3. The computed segmentation can be either recalculated with different parameters or manually refined and serves to assign every bixel to a cluster to facilitate the segmentation enforcement in section 4.3.6. For every music piece, the aforementioned analysis steps have to be done only once and the output can be used to rearrange the piece as often as desired, as it does not depend on the user constraints (except for the ground truth segmentation which is treated as constant after user confirmation).

After confirming the ground truth segmentation, the system enters the synthesis phase that is executed whenever the user decides to generate a new music piece with the current constraints. At first, the **path optimisation** presented in section 4.3 searches a path through the original piece leading to an optimal solution with the least cost, where costs for including different jumps in the solution are contained in the *unified cost matrix* defined in section 4.3.3. This matrix takes into account the penalties imposed by several user constraints and is afterwards extended to include penalties for annoying repetitions in section 4.3.4. The path optimisation process itself is accelerated with a *multiple goal A* algorithm* introduced in section 4.3.5. In order to support rearrangement with a user-defined target segmentation, section 4.3.6 presents a method that modifies the costs for jumps dynamically during

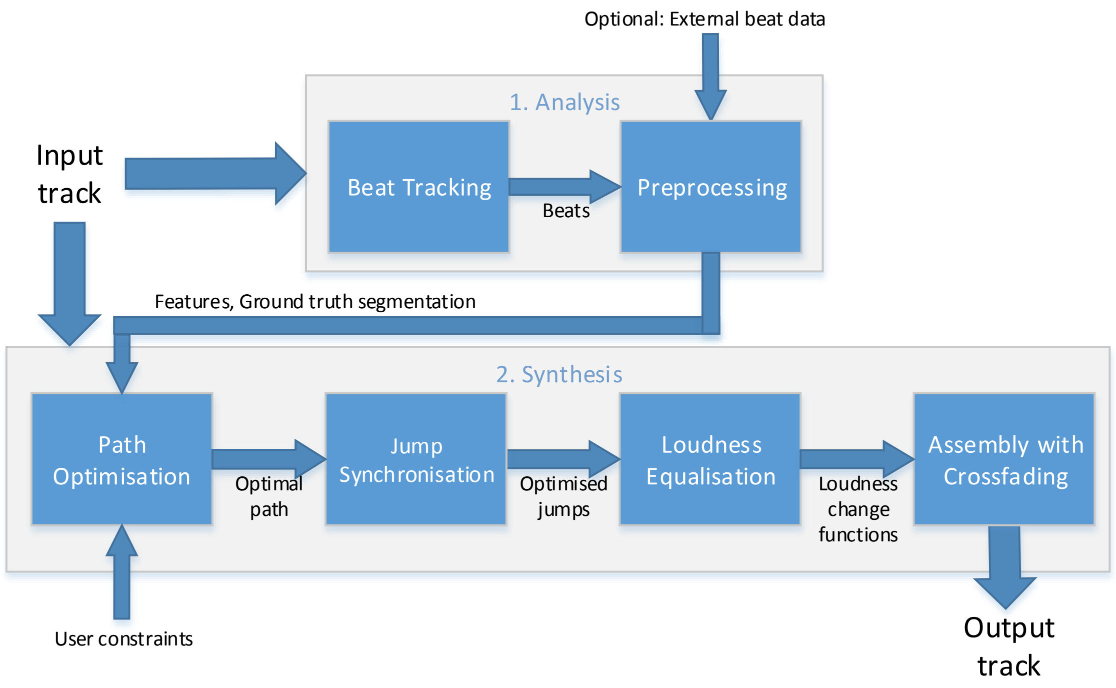


Figure 3.1.: Overview of the proposed music restructuring system.

path optimisation so that the result approximately fulfils the target segmentation. The optimal path output returned by the path optimisation stage consists of jumps from one point of time to another in the original signal that often need to be postprocessed to achieve high transition qualities. Therefore, a series of stages concerning the optimisation of these jumps follows, starting with the **jump synchronisation** that aligns every jump origin and destination as described in section 4.4.1 to account for potential inaccuracies of the detected beat positions. To improve loudness continuity at the resulting cuts, the **loudness equalisation** stage in section 4.4.2 builds a model describing how to amplify or attenuate the audio signal around the jump origin and destination of every jump, before both of the corresponding excerpts are crossfaded in the final **assembly with crossfading** presented in section 4.4.3 to eliminate sound artefacts. After assembling the new music piece from the input track according to the given jump positions and the optimised audio material near the cut positions, it is optionally exactly scaled to the target duration with a timescale-pitch modification method from section 4.5.

With the exception of the final assembly stage, the input signal is always assumed to be in a mono format (with an implied conversion to mono for multi-channel signals) before any actions related to the signal are performed, unless otherwise noted.

In the following, the music rearrangement system will be explained in detail.

4. System components

The components of the proposed system for constraint-based rearrangement of music will be presented in this section.
As already stated in the previous section 3.2, the system consists of several stages arranged in a pipeline. The next sections will each explain a stage of the system in greater detail and their order of occurrence corresponds to to their position in the pipeline. For better comprehensibility, the explanations of the steps performed during these stages will be supplemented with illustrations and audio examples generated with tracks from the database CC1 (see table A.3), where appropriate. Finally, the user interface of the system will be presented in section 4.6.

4.1. Beat tracking system

Because our system relies on the concept of using sections of the original track to assemble a new music piece, we need to divide the original music into a discrete number of slices in a musically meaningful way so the transitions between the slices sound as good as possible when concatenating them. A promising approach also used by Wenner [48] is to use the beat positions as boundaries for these slices, producing one bixel per pair of neighbouring beat positions.
Formally, beat tracking yields a vector $\mathbf{b}^{\mathrm{pos}}$ containing the $N+1$ detected beat positions in seconds, resulting in N bixels $b_1, b_2, \ldots, b_N$, where the i-th bixel b_i represents the musical content between b_i^{pos} and b_{i+1}^{pos} seconds. The length of every bixel can be described as an N-dimensional vector $\mathbf{b}^{\mathrm{len}}$ with $b_i^{\mathrm{len}} = b_{i+1}^{\mathrm{pos}} - b_i^{\mathrm{pos}}$.

Selection process Because beat tracking as a problem in itself is out of the scope of this thesis, we assessed existing beat tracking systems and decided which one to use based on their suitability for our application. A recent evaluation of beat tracking systems can be found in [50], where four metrics indicating performance with a percentage value are deployed. Two metrics *CMLc* and *CMLt* require the correct metric level and two metrics *AMLc* and *AMLt* do not, where the suffix

"c" in the name of the metric indicates the requirement of *continuity.* Considering our application, the detected beats do not have to represent a specific metric level, as long as they all share the same metric level or at least exhibit the same delay compared to a metric level (for example detecting only the weak beats after the strong beats), because such a delay does not significantly affect the output when concatenating the bixels. Continuity decreases with the number of shifts between metric levels occurring and thus should ideally be high to avoid pairs of bixels containing differently sized portions of a measure. Consequently, the *AMLc* metric led to the selection of the beat tracker from Davies [9], as it provides not only a good performance, but also a freely available implementation [6] that can be easily integrated into our system, because it is written in MATLAB.
An example of the beat tracking system applied to the song "El barzón" from "Los Amparito" contained in our database CC1 can be heard in audio excerpt 4.1, where beeping sounds in the song mark the positions of the detected beats. Note that

Audio 4.1.: Extract of the song "El barzón" performed by "Los Amparito" from database CC1 in table A.3 with beep sounds at the detected beat positions

the detected beats are the weak beats exactly halfway between the strong beats that one would usually tap to. Because over the whole song just the weak beats are consistently detected, this does not present a problem - on the contrary, it provides a good basis for the rest of the music rearrangement system, as the resulting bixels represent the same time frame from a musical perspective.

Evaluation on database CC1 As mentioned in section 2.3, ground truth beat data was created for the music database CC1 from table A.3 by manually tapping to the beat when listening to the songs. Besides the AMLc metric, another especially useful metric in the context of our application is the *information gain metric* [8]. Generally speaking, it considers the relative beat errors for the detected beat positions and summarises this error distribution using a histogram. Afterwards, the information gain of this error distribution in comparison to a uniform distribution that equals randomly selecting positions as beats is calculated. Consequently, the metric does not significantly punish beat detections on another metric level or beats being detected with a constant delay, making it a suitable indicator for how well the result produced by a restructuring system based on bixels can at most turn out to be.
Table A.7 shows the beat tracking performance for every song in the database CC1

with the metrics CMLc, CMLt, AMLc, AMLt and the information gain generated with the "Beat Tracking Evaluation Toolbox" [7] written in MATLAB. The standard configuration of the toolbox was used, leading to information gain values that can range between zero and about five.

For some songs like "Lullaby", the metrics CMLc and CMLt requiring the correct metric level are zero, although AMLc and AMLt are almost at the maximum value of 100. This is due to the the beat tracker detecting another metric level almost perfectly. The excellent results of the Wenner method for this song in table A.8 prove that the correct detection of a specific metric level is indeed not an issue for a bixel-based restructuring system, making the AMLc and AMLt metrics more useful when choosing a beat tracker for such a system than the CMLc and CMLt metrics. The detected beats for the song "Stormy Blues" are rated with zeros for every metric except the information gain metric, as they mark every second beat perfectly, but the meter is based on a strong beat followed by two weak beats. However, this incorrect meter does not greatly affect the output quality, as table A.8 shows, demonstrating the suitability of the information gain metric for our purposes. In general, the results of the preliminary study in table A.8 have to be analysed with the beat tracking evaluation scores from table A.7 in mind.

Overall, the selected beat tracker performs acceptably well for most songs, reaching an average information gain of 2.34, This is a good value considering that even an almost perfectly tracked beat results in values around 3.5 due to minimal timing inaccuracies, as the results for the track "Barbarian" in table A.7 demonstrate.

While beat tracking as a problem of its own is neither the focus of this thesis nor intended to be improved, the results can be used to discern whether music retargeting has failed because of incorrect beat detection or because of some deficiency of the proposed algorithm itself by comparing the results when using the beat tracker to those produced with the manually generated ground truth beat data.

4.2. Preprocessing

The goal of the preprocessing stage is to extract musical information about the input song and every one of its bixels in particular needed for the subsequent path optimisation and thus requires the beat positions from the beat tracking system as additional input.

More specifically, a measure is needed to estimate the perceptual quality of a **bixel transition**, that is, selecting a bixel b_j as a successor to some bixel b_i so that b_j will

be played directly after b_i in the result (we call b_i the **origin** and b_j the **destination** of the bixel transition). This measure yields a **bixel transition cost** for every bixel transition representing its sound quality, using low numbers for a high transition quality and vice versa. Transitions from a bixel b_i to its natural successor b_{i+1} mean the original signal can be used without alterations. All other bixel transitions lead to a cut in the final output and are also called **bixel jumps**. Similar to the definitions of jumps in section 2.2, we categorise bixel jumps from any bixel b_i to another non-successive bixel b_j according to their direction. **Forward bixel jumps** fulfil $j > i + 1$ and **backward bixel jumps** require $j < i + 1$. Note that under this definition, the case $j = i + 1$ can not occur for a bixel jump, but only for a simple bixel transition that does not produce a cut in the result. The **distance** of a bixel jump is defined as $|j - (i + 1)|$ bixels, while the **bixel jump shift** is equal to $j - (i + 1)$ and takes the direction of the jump into account.
These bixel transition costs will be used in section 4.3 to determine the optimal order of bixels to use for assembling the solution.
MATLAB along with the "Signal Processing Toolbox" [26] and the "MIR Toolbox" [22], which was specifically designed for music information retrieval tasks, offers a suitable environment for extracting information about the bixels in order to create a measure for the bixel transition costs. A large portion of the functionality required is already implemented and suboptimal runtimes due to MATLAB being an interpreted language are not problematic as the preprocessing stage has to be performed only once for every music piece.
In the first section 4.2.1, the timbre of all bixels will be analysed to estimate the transition quality regarding timbre when concatenating two bixels, while the second section 4.2.2 aims to do the same for the loudness continuity concept introduced in section 2.1.
Finally, an automatic segmentation method will be presented in section 4.2.3 to help the user in creating the ground truth segmentation for the music piece that can later be used in combination with a target segmentation to enforce the resulting song to have a specific structure.

4.2.1. Transition costs regarding timbre

In this section, we will obtain a measure $D'(i, j)$ for the transition quality regarding timbre when selecting bixel b_j as a successor of bixel b_i in the result and represent it as a $N \times N$ matrix $\mathbf{D}'$. We call such a square matrix containing the bixel transition costs for all pairs of bixels a **transition cost matrix**.

At first, a $N \times N$ **self-similarity matrix S** is computed according to [48] that describes the perceptual similarity of any bixel b_i to any other bixel b_j with a value $S(i,j) \in [0,1]$ as follows. For every bixel, a 40-dimensional vector is extracted, containing the first 40 *mel-frequency cepstral coefficients (MFCCs)* often used in the context of speech recognition, which are "the results of a cosine transform of the real logarithm of the short-term energy spectrum expressed on a mel-frequency scale" [52]. The resulting vectors describe the timbre of every bixel. In our implementation, this feature extraction is done with the MIR Toolbox [22] and executed in parallel for every bixel by using the "Parallel Computing Toolbox" [24] to speed up the process.

Afterwards, the *Spearman rank correlation* is used to calculate the distance between the feature vector of every bixel, yielding the self-similarity matrix **S** after normalisation to the range $[0,1]$.

An example is presented for the song "El barzón" from "Los Amparito" from the database CC1 in figure 4.2 (a). The diagonal contains ones, because every bixel is perfectly similar to itself. Orange and yellow areas in the form of squares reveal groups of bixels which are similar to each other. Figure 4.2 (b) shows the lower right part of the matrix **S** from figure 4.2 (a) in greater detail.

In order to extend this bixel-based notion of similarity to include temporally adjacent bixels, consider the $N \times N$ matrix $\mathbf{S}'$ defined in [48]:

$$S'_{i,j} = \sum_{t=-m}^{m} w_t S_{i+t,j+t}. \tag{4.1}$$

Like in [48], the values $m = 2$ and $w_t \in \{w_{-m}, \dots, w_m\}$ as the normalised binomial coefficients are left unchanged, thereby taking into account how similar the m bixels before and after bixel b_i and b_j are to each other. However, the authors do not mention how to proceed near the edges of the matrix, where the formula refers to non-existent entries in **S** with invalid indices. This is resolved when a more refined formulation is used instead, which only sums up valid entries and divides this weighted sum by the sum of all used weights:

$$\begin{aligned} S''_{i,j} &= \frac{\sum_{t=m_{start}}^{m_{end}} w_t S_{i+t,j+t}}{\sum_{t=m_{start}}^{m_{end}} w_t} && \text{with} \\ m_{start} &= \max\{-m, 1 - \min\{i,j\}\} && \text{and} \\ m_{end} &= \min\{m, N - \max\{i,j\}\}. \end{aligned} \tag{4.2}$$

This results in a matrix $\mathbf{S}''$, where bixels near the start and end of the song can still exhibit just as high similarity values as other bixels and keeps the diagonal values equal to one, which is the logical consequence of every bixel being perfectly similar to itself.

The next step consists of calculating the transition cost matrix containing the costs for concatenating two bixels. Again, the definition from Wenner [48]

$$D_{i,j} = 1 - S'_{i+1,j} \tag{4.3}$$

for the transition cost matrix $\mathbf{D}$ does not take edge cases into account. The entries in the last row $D_{N,j}$ of the matrix can not be computed, as $S'_{N+1,j}$ is out of bounds and consequently undefined. To obtain appropriate values for this row, we redefine $\mathbf{S}'$ to be a $(N+2m) \times (N+2m)$ matrix that results from zero-padding $\mathbf{S}$ before applying a two-dimensional *convolution* on $\mathbf{S}$ with the $(2m+1) \times (2m+1)$ kernel matrix

$$\mathbf{K} = \begin{pmatrix} w_{-m} & 0 & \dots & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & w_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & w_m \end{pmatrix}, \tag{4.4}$$

where the weights $w_t \in \{w_{-m}, \dots, w_m\}$ again represent the normalised binomial coefficients used in Wenner [48]. With this definition for the matrix $\mathbf{S}'$, the row $N+m+1$ of $\mathbf{S}'$ corresponds to the undefined row $N+1$ of Wenner's matrix $\mathbf{S}'$ and can be used instead to calculate the last row of the transition cost matrix. It contains values obtained by applying the kernel that overlapped the zero-padded region, which are therefore low. This new version of the matrix $\mathbf{S}'$ is illustrated for our example song in figure 4.2 (c) and (d). With $N = 405$ bixels and $m = 2$, the row in question is number $N+m+1 = 408$ and reveals its low values with a blue colour.

Using the matrix $\mathbf{S}''$ from equation (4.2) whenever applicable and the above definition of the $(2m+1) \times (2m+1)$ matrix $\mathbf{S}'$ for computing the last row, we obtain our improved transition cost matrix $\mathbf{D}'$:

$$D'_{i,j} = \begin{cases} 1 - S'_{i+1+m,j+m} & \text{if } i = N \\ 1 - S''_{i+1,j} & \text{else.} \end{cases} \tag{4.5}$$

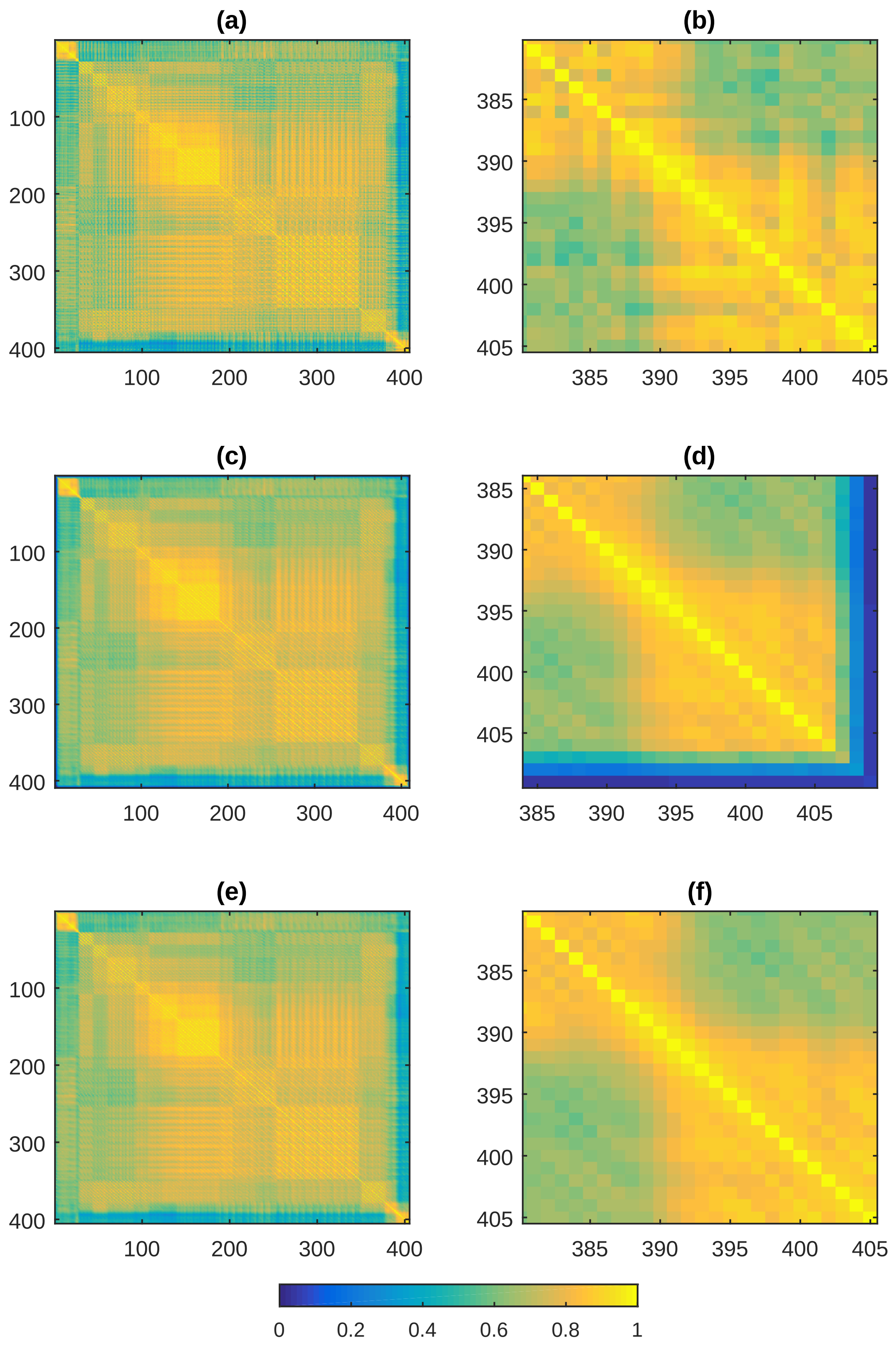


Figure 4.2.: Self-similarity matrices for the song ”El barzón” from ”Los Amparito”, where both dimensions describe the bixel index. (a) shows $\mathbf{S}$ and (b) a zoomed version detailing the lower right part. (c) and (d) analogously represent $\mathbf{S}'$ and (e) and (f) the matrix $\mathbf{S}''$. All values lie in the $[0, 1]$ range and are coloured according to the colour bar at the bottom.

Due to the low values in the row $N + m + 1$ of $\mathbf{S}'$, the last row of $\mathbf{D}'$ contains high costs. This is desirable, as it penalises bixel jumps originating from the very last bixel of the song, which often produce jarring transitions from the silence present at the end of the music piece to the remaining louder sections.

In contrast to Wenner [48], the diagonal elements representing bixel jumps with a shift of $\delta = -1$ are not reset to one to avoid the repetition of a single bixel, because the concept explained in section 4.3.4 is a more general and flexible approach for avoiding repetitions.

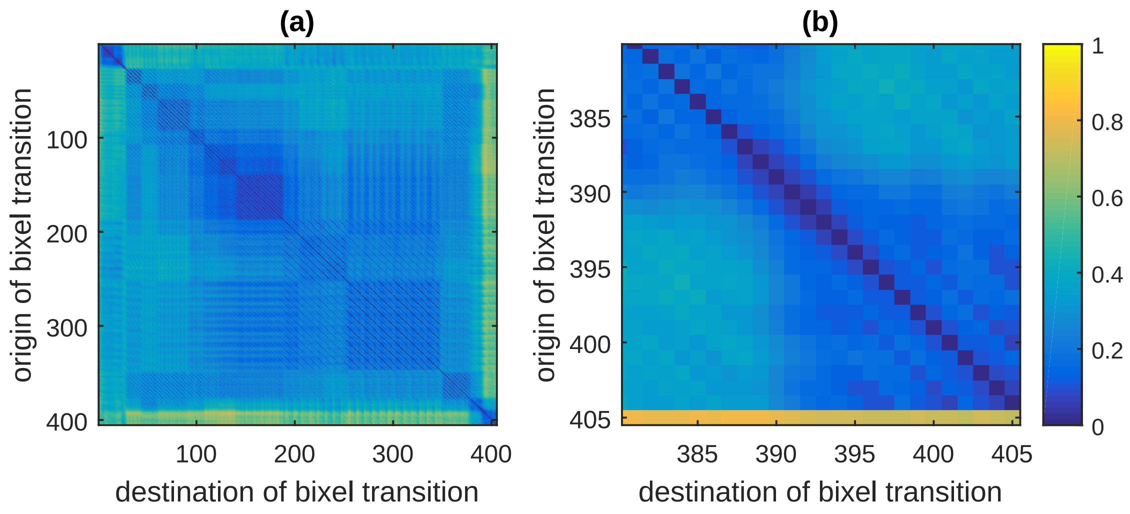


Figure 4.3.: Transition cost matrix $\mathbf{D}'$ for the song ”El barzón” from ”Los Amparito”. (a) shows the full matrix while (b) shows the lower right part in greater detail.

4.2.2. Transition costs regarding loudness

In the previously constructed transition cost matrix $\mathbf{D}'$ from section 4.2.1, timbre was taken into account, but loudness plays a very insignificant role. Looking at the extracted 40-dimensional MFCC vectors used for the calculation, only the first coefficient, representing a ”collection of average energies of all frequency bands” [52], provides a rough estimation of the loudness. As a result, loudness has little influence on the similarity measure obtained by the Spearman rank correlation, leading to many jumps with extreme loudness differences as discussed in section 2.4. Therefore, we propose an additional transition cost matrix $\mathbf{L}$, designed to supplement the transition cost matrix $\mathbf{D}'$. This matrix $\mathbf{L}$ aims to describe how jarring the loudness change between any two bixels would be perceived if played in succession.

Loudness modelling In contrast to simply calculating the average *energy* for every bixel, a loudness model also takes human perception into account: The perceived loudness of a tone is dependant on its frequency [30], and loud sounds can mask quieter sounds occurring shortly after [15].

Several loudness models have been proposed, differing in whether they are designed to be applied to *time-varying* or *stationary* sounds [19]. Music falls into the category of time-varying sounds, as it exhibits a time-dependant behaviour in contrast to stationary sounds produced by noise-polluting sources like static noise from a TV.

Two widely used models for time-varying sounds have been developed by Zwicker [14] based on previous work for steady sounds [53] and by Moore and Glasberg [19], again based on a previous publication dealing with steady sounds [29]. Both are implemented in the openly available "Loudness Toolbox" [18] for MATLAB. Our tests with the Loudness Toolbox on the first ten songs of the CC1 database have shown the Moore and Glasberg model to take approximately 45 to 115 minutes to process one music piece while the Zwicker model was successfully computed in about one minute (see table A.6 for details). Even when considering the relaxed time constraints in the preprocessing stage, the high average runtime of the Moore and Glasberg implementation makes its usage infeasible in the context of a music rearrangement software. As the second column in table A.6 shows, we also managed to further reduce the average runtime of the Zwicker model to approximately 20 seconds by parallelising the computation with the Parallel Computing Toolbox. Although we are not aware of publications containing an extensive, qualitative comparison between both models especially for music signals, the results of both models also obtained with the implementation from the Loudness Toolbox were found to be very similar when analysing music in "A Brief Comparison of Loudness Evaluation Methods" [44].

Therefore, the Zwicker model is computed and its *instantaneous loudness* extracted, which represents the loudness in *sone* over time in the input signal as a discrete function $l(t)$.

Sone is a psychoacoustic unit measuring how loud a sound is perceived and scales linearly, that is, a doubled sone value represents a sound that is perceived twice as loud.

In figure 4.4, the instantaneous loudness function according to Zwicker is plotted. The loudness first tends to increase steadily, displaying a repetitive pattern due to the musical structure, and reaches a climax near the end of the piece, before finally subsiding in the outro.

In the remainder of this section, the loudness function obtained from the loudness

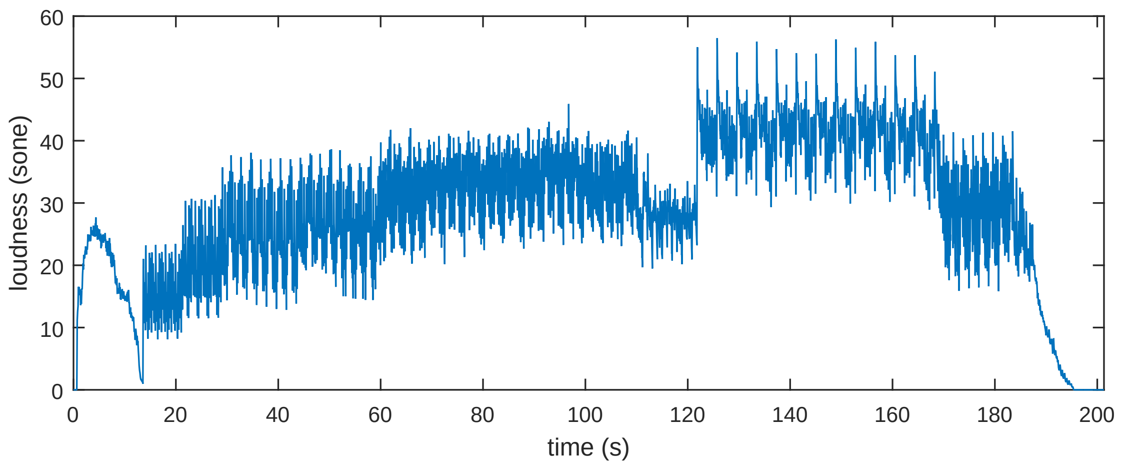


Figure 4.4.: The instantaneous loudness function $l(t)$ according to the Zwicker model extracted for the example song "El barzón" from "Los Amparito".

model will be used to generate a measure for the loudness continuity introduced in section 2.1. The general idea is to calculate the average loudness near the start and end of the bixels and then compare these measurements for a given pair of bixels to estimate the loudness continuity when concatenating them.

Assigning loudness to bixels At first, the average loudness at the beginning and at the end of every bixel is calculated as follows. A parameter t_{mean} defines the desired amount of seconds at the start and at the beginning of each bixel to take into account. For this thesis, $t_{\text{mean}} = 0.2$ is used, as it is large enough to be robust against fluctuations in the loudness curve and short enough to capture the start and end loudness in a meaningful way. Future work should include optimising this parameter based on listening experiments.
If the length of the considered bixel b_i is shorter than this duration, the window size t_i^{lim} over which the average is calculated has to be restricted to the length of the bixel to only include loudness values from the bixel itself and none from of its neighbouring bixels:

$$t_i^{\text{lim}} = \min\{t_{\text{mean}}, b_i^{\text{len}}\}. \tag{4.6}$$

As a result, the vector $\mathbf{t}^{\text{lim}}$ describes how many seconds are used in the average calculation performed for every bixel.
In figure 4.5, the part of the loudness function displayed in figure 4.4 corresponding to the bixel 13 of the song "El barzón" from "Los Amparito" is presented. The green and red areas illustrate the time intervals used for the upcoming calculation

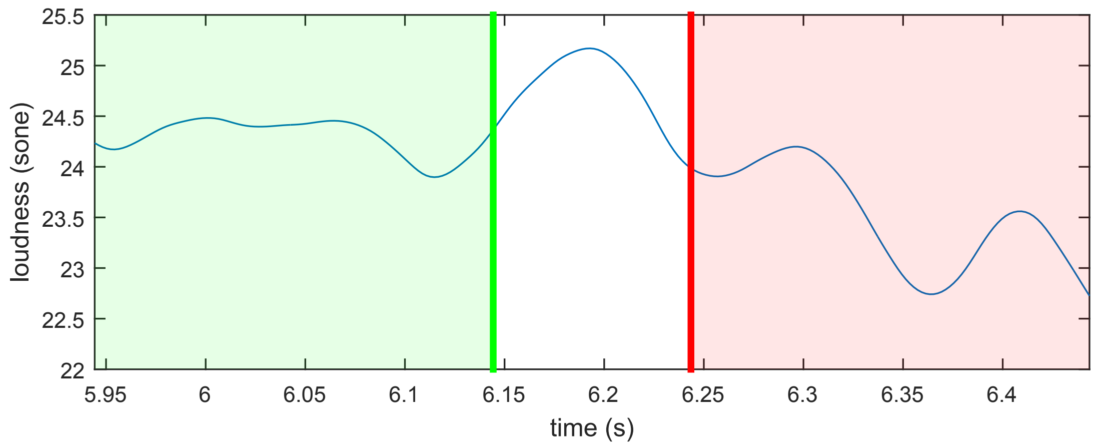


Figure 4.5.: The part of the loudness function that corresponds to the bixel b_{13} of the song "El barzón" from "Los Amparito". The green area is used for the start loudness and the red area for the end loudness calculation.

of the loudness at the start and at the end of the bixel, respectively. Note that both intervals are always of the same length and also overlap if the bixel is shorter than $2 \cdot t_{\text{mean}}$ seconds.

Instead of calculating a simple average of all loudness values within these intervals, we put more weight on loudness values located near the start and end of the bixel, because they lie closer to a potential jump and therefore have a greater influence on how the bixel transition is perceived. Mathematically, the weights used for calculating the loudness averages for a bixel b_i are modelled as a linear function $w_i(t)$ that returns zero for $t = 0$ and has a positive gradient. This gradient is defined in such a way that the area enclosed between the function and the t axis between $t = 0$ and $t = t_i^{\text{lim}}$ is exactly one, ensuring that the sum of weights used for the subsequent weighted average calculation is one and no retroactive normalisation is required:

$$w_i(t) = \frac{2t}{(t_i^{\text{lim}})^2}. \tag{4.7}$$

Based on the weighting functions w_i, we compute the vectors $\mathbf{l}^{\text{start}}$ and $\mathbf{l}^{\text{end}}$ containing a weighted estimate of the loudness at the start and at the end of every bixel:

$$\begin{aligned} l_i^{\text{start}} &= \int_{b_i^{\text{pos}}}^{b_i^{\text{pos}}+t_i^{\text{lim}}} l(t) w_i(-t + b_i^{\text{pos}} + t_i^{\text{lim}}) \, \mathrm{d}t \qquad \textit{and} \\ l_i^{\text{end}} &= \int_{b_{i+1}^{\text{pos}}-t_i^{\text{lim}}}^{b_{i+1}^{\text{pos}}} l(t) w_i(t - b_{i+1}^{\text{pos}} + t_i^{\text{lim}}) \, \mathrm{d}t. \end{aligned} \tag{4.8}$$

The above definition results in a strong influence of the loudness values near the beginning b_i^{pos} of a bixel b_i when calculating its start loudness l_i^{start} as well as the values near the end b_{i+1}^{pos} of the bixel when calculating its end loudness l_i^{end}.

Loudness continuity measures Given the start and end loudness for every bixel, two measures of loudness continuity are introduced:
Transitional loudness is based on the idea that a transition from a bixel b_i to a bixel b_j is perceived as smooth with regards to loudness if the end loudness l_i^{end} of bixel b_i is approximately equal to the start loudness l_j^{start} of bixel b_j. We shall derive a $N \times N$ transition cost matrix $\mathbf{L}^{\text{trans}}$ from this idea by calculating the Euclidean distance between the respective loudness values:

$$L_{i,j}^{\text{trans}} = |l_i^{\text{end}} - l_j^{\text{start}}|. \tag{4.9}$$

All entries in the matrix $\mathbf{L}^{\text{trans}}$ are then divided by the maximum entry to normalise all values to the range $[0, 1]$.
Comparative loudness is based on the concept that a jump from a bixel b_i to another bixel b_j does not produce an irritating loudness change if the musical content that was expected after bixel b_i, namely bixel b_{i+1}, is approximately as loud at the beginning as the start of the bixel b_j which is played instead:

$$L_{i,j}^{\text{comp}} = |l_{i+1}^{\text{start}} - l_j^{\text{start}}|. \tag{4.10}$$

We set $l_{N+1}^{\text{start}} = 0$ for the purpose of this calculation, because the listener expects silence after the last bixel and thus a loudness of 0 sone. Again we normalise the entries of the matrix to the $[0, 1]$ range.
Both concepts can arguably describe a condition for the loudness continuity explained in section 2.1, so we finally combine both loudness matrices into one loudness matrix $\mathbf{L}$ by selecting the pair-wise minimum from both matrices:

$$L_{i,j} = \min\{L_{i,j}^{\text{comp}}, L_{i,j}^{\text{trans}}\}. \tag{4.11}$$

This results in a transition cost matrix $\mathbf{L}$ with high costs exclusively for bixel jumps that fulfil neither of both requirements for loudness continuity, i.e., only unexpected and large changes in loudness lead to a high transition cost. An example for the matrix $\mathbf{L}$ can be seen in figure 4.6. Structures in the form of rectangles indicate bixels with similar loudness, whereas the yellow colours visible at the borders demonstrate the drastic difference in loudness at the beginning and end of the piece compared to

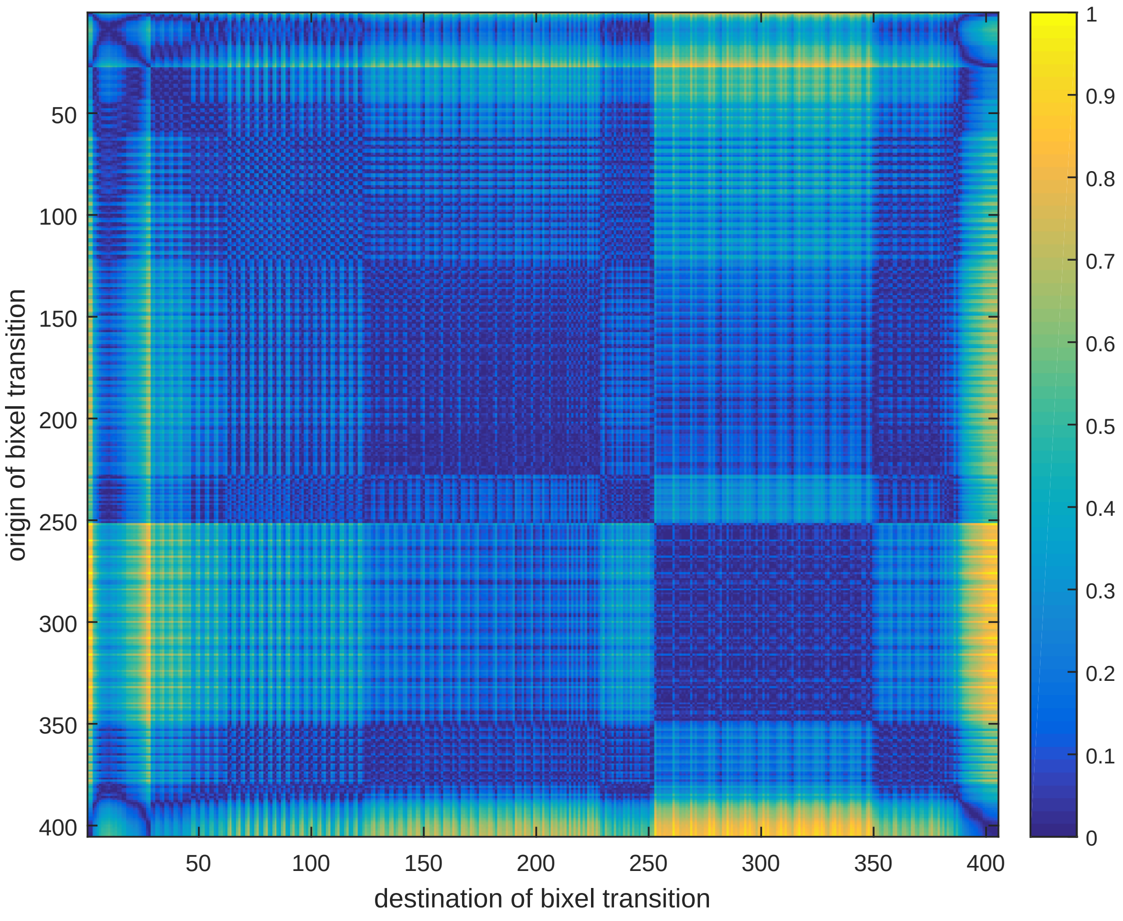


Figure 4.6.: Loudness matrix **L** computed according to equation (4.11) for the example song ”El barzón” from ”Los Amparito”.

the rest. Also note that selecting the successor b_{i+1} of a bixel b_i, that is, not causing a jump but leaving the original piece as is, incurs no cost: Similar to the transition cost matrix $\mathbf{D}'$ in section 4.2.1, $L_{i,i+1} = 0$ because using equation (4.11) and (4.10), $L_{i,i+1} = \min\{|l^{\text{start}}_{i+1} - l^{\text{start}}_{i+1}|, L^{\text{trans}}_{i,i+1}\} = \min\{0, L^{\text{trans}}_{i,i+1}\} = 0$. As a result, only playing back the original piece and not jumping does not incur any cost.

4.2.3. Automatic segmentation

The proposed restructuring system supports the enforcement of a user-specified target segmentation (explained in section 4.3.6) for the produced song based on a ground truth segmentation. Creating such a ground truth segmentation for the input track can be time-intensive, especially when it is unknown to the user. To accelerate the process, an automatic segmentation method is employed whose result provides a basis that can be corrected by the user, if necessary.

Mathematically, we represent the term ”segmentation” introduced in section 2.1

with C clusters identified by the cluster indices $\mathcal{C} = \{1, \ldots, C\}$ and a function $s(t)$ returning the cluster index of the cluster that the current segment at t seconds in the music piece is assigned to. In particular, we define the notation $s_{\text{ground}}(t)$ for the ground truth segmentation.

Proposed method The method already presented by Wenner [48] will be adopted in this thesis and further refined. In contrast to Wenner's method, the self-similarity matrix $\mathbf{S}''$ from equation (4.2) is not smoothed with a Gaussian kernel in our implementation, because irregularities in the data are filtered out later in the *peak picking* stage. A *novelty score* representing the likelihood of a segment transition at a specific time is extracted by convolving the self-similarity matrix $\mathbf{S}''$ along the diagonal with an $M \times M$ checkerboard kernel [16]. The values in range of $[0, 1]$ indicate how much the upcoming part of the song is different to the previous one, combined with how similar those parts are to each other. Although the last step is also done in the work of Wenner [48], we use $M = 128$ instead of $M = 96$ to detect segment transitions, because our beat tracker tends to produce shorter bixels and therefore a larger kernel is needed to cover the same amount of seconds of musical material on average.

An exemplary novelty curve as a function of time is presented in figure 4.7. Over the whole duration of 200 seconds, large values correctly indicate a segment transition at the end of the intro and the beginning of the outro.

In general, local maxima of the novelty curve tend to indicate transitions from one segment to another and are selected by a process called peak picking implemented

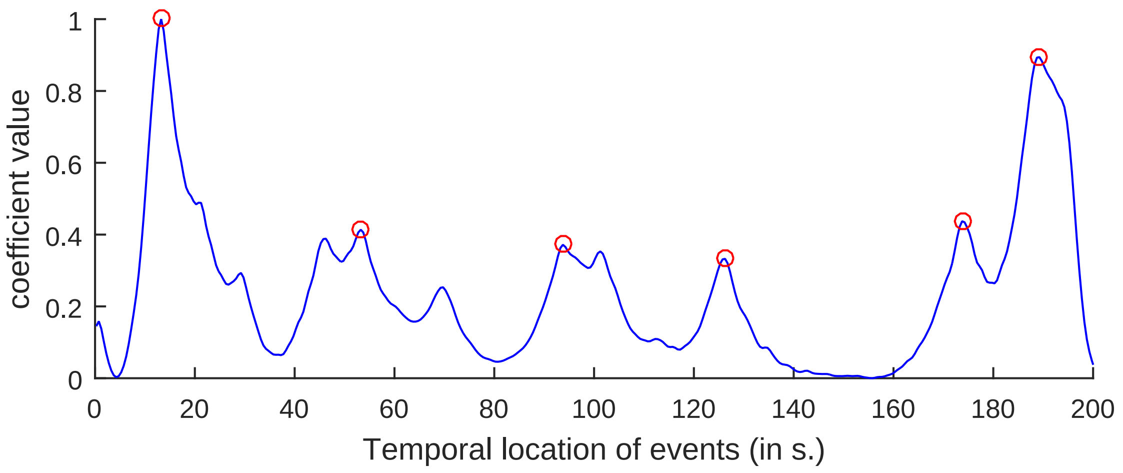


Figure 4.7.: Novelty curve extracted for the song "El barzón" from "Los Amparito" with local maxima identified by the peak picking process marked with red circles.

in the MIR Toolbox [22] for MATLAB with the following settings: The first and last sample of the novelty curve is not considered as a candidate for a peak, because the song start and end do not represent transitions from one segment to the next. Additionally, on a scale where one represents the distance between the maximum and the minimum of the input signal, only local maxima whose neighbouring local minima both differ in their coefficient value more than $c_{\text{peak}} = 0.1$ are selected. This value is the default in MIR Toolbox, but as part of future work, it could be dynamically changed depending on how many segments should be identified. The process above effectively filters the local maxima by marking only a subset of them as relevant and removes the need for smoothing the similarity matrix as mentioned at the beginning of this paragraph. For our example in figure 4.7, peak picking returns the local maxima marked with red circles.

After retrieving the segment transitions, the resulting segments are clustered into groups according to their musical similarity with the goal of identifying groups of segments serving the same function. Like in [48], the bixels are assigned to the different segments according to their position and every segment is assigned a feature vector calculated as the average of the 40-dimensional MFCC vectors of all its bixels. Afterwards, the segments along with their MFCC descriptors, which can be viewed as multi-dimensional points with each point representing a segment, are clustered with *spectral clustering by normalised cuts* [40] using the MATLAB implementation from the "Graph Demo" package [27].

The primary limitation of this clustering algorithm (shared with many other clustering algorithms) is that the number of clusters C has to be known in advance and specified as input. This is reflected in the previous work from Wenner [48], in which the user has to select a specific value for C in the hope of obtaining a meaningful segmentation. In this thesis, we will go one step further towards fully automating and thereby accelerating the whole process of creating a ground truth segmentation by automatically selecting the optimal number of clusters C within a given range $C_{\text{range}} = [C_{\text{min}}, C_{\text{max}}]$, where C_{min} and C_{max} are user-specified positive integers and $2 \leq C_{\text{min}} \leq C_{\text{max}}$. Additionally, the maximum number of clusters C_{max} can be at most equal to the number of detected segments, because every segment can only be assigned to exactly one cluster. The optimal number of clusters C_{opt} is then defined as the value for $C \in C_{\text{range}}$ whose associated clustering result fits the given segment features best, according to some evaluation metric $c_{\text{eval}}(C)$:

$$C_{\text{opt}} = \underset{C \in C_{\text{range}}}{\arg\max} \{c_{\text{eval}}(C)\}. \tag{4.12}$$

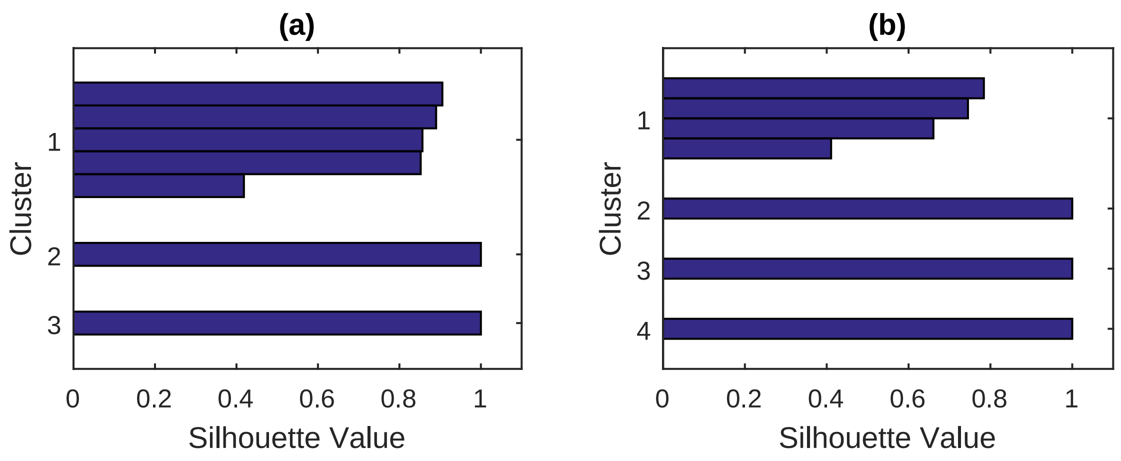


Figure 4.8.: Spectral clustering results of the song ”El barzón” from ”Los Amparito” with (a) $C = 3$ and (b) $C = 4$ clusters evaluated using a silhouette plot. Each bar represents the silhouette value of a segment.

In order to evaluate the function $c_{\text{eval}}(C)$ for a specific C, we calculate the *silhouette* value $c_{\text{sil}}(i) \in [-1, 1]$ for every segment that was clustered [36]. The silhouette of a point (equivalent to a segment in our context) is high, if the other points in the same cluster are close and there is a large distance to the points in the other clusters, indicating that it is clustered correctly. Therefore, we use the mean of the silhouette of all segments as metric $c_{\text{eval}}(C)$ and select the clustering that produces the maximum average silhouette value. Note that the user can still enforce a specific number of clusters C to be used and avoid this automatic selection by setting $C_{\text{min}} = C_{\text{max}}$. The procedure is visualised using our example song and $C_{\text{range}} = [3, 4]$ in figure 4.8. When clustering with $C = 3$, every segment except the intro and outro (clusters two and three) is assigned to the first cluster. As figure 4.8 (a) shows, this is an acceptable clustering since the intro and outro sound very different from the rest of the piece and differences between segments in the first cluster are comparatively low. The segment with the largest difference in this first cluster is assigned to an extra cluster when setting $C = 4$. However, this visibly lowers the silhouette values of the

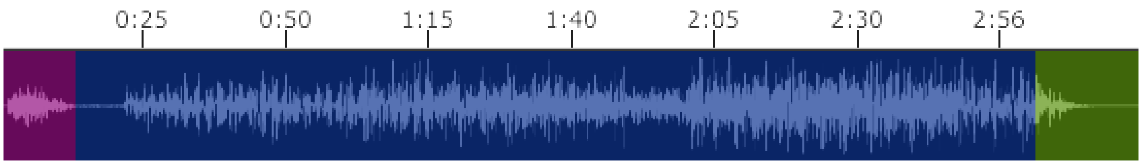


Figure 4.9.: Clustering of the song ”El barzón” from ”Los Amparito” with $C = 3$ clusters chosen from the possible values in $C_{\text{range}} = [3, 4]$. The red section marks the intro and the green section the outro, while the segments in between were merged to one large cluster.

remaining segments seen in figure 4.8 (b), because their distance to this new cluster is quite low. Taking the average over the silhouette values of every segment for both settings, we obtain $c_{\text{eval}}(3) = 0.846$ and $c_{\text{eval}}(4) = 0.8$. As a result, the clustering with $C = 3$ clusters is chosen as the final output, because it exhibits a higher mean silhouette value than all other clusterings in the C_{range} interval.
Figure 4.9 displays the resulting segmentation for the example song by highlighting the waveform with a colour corresponding to the assigned cluster.
In the following section, we will explain how the path optimisation stage utilises the musical information about the original track obtained by the methods described in this section in order to find the optimal arrangement that fulfils the given user constraints.

4.3. Path optimisation

This section is concerned with finding an optimal solution given a set of user constraints. At first, path optimisation will be formally introduced as a problem in the upcoming section 4.3.1. The following section 4.3.2 deals with estimating the length of the optimal path a priori, that is, before an optimisation algorithm is executed. Section 4.3.3 will provide the basis for a cost function that incorporates some user constraints and is subsequently extended with the repetition avoidance concept in section 4.3.4. After the necessary problem and cost function definitions, a version of the A* algorithm supporting multiple goals and employing specially designed heuristics will be presented in section 4.3.5. Finally, modifying the cost function during the execution of the path optimisation algorithm will be introduced in section 4.3.6 as a method to ensure that the segmentation of the resulting song matches the user-defined target segmentation.

4.3.1. Problem formulation

Formally, a path through the original music piece, called **bixel path**, can be defined as a vector containing the indices of the k bixels visited along the path in their order of occurrence:

$$\mathbf{p} = (p_1, p_2, \dots, p_k). \tag{4.13}$$

Given a term $T_{i,j}$ describing the cost of selecting bixel b_j as successor to bixel b_i in such a path, for which the basis will be established in section 4.3.3 and which will be explicitly defined in section 4.3.4, the total **cost of a bixel path $\mathbf{p}$** is given by

$$c_{\text{path}}(\mathbf{p}) = \sum_{i=1}^{k-1} T_{p_i,p_{i+1}}. \tag{4.14}$$

Not all possible paths are **valid** as they are constrained by various user-defined parameters: By default, the start time t_{start} and end time t_{end} are set to the beginning and the end of the original music piece, respectively, to ensure that the resulting music piece starts and ends in the same manner as the original. However, they can be changed to arbitrary positions in the music piece by the user. Assuming these positions correspond to the bixels $b_{i_{\text{start}}}$ and $b_{i_{\text{end}}}$, only paths containing at least $k \geq 2$ bixels with $p_1 = i_{\text{start}}$ and $p_k = i_{\text{end}}$ are considered valid in the optimisation problem.
Additionally, a positive target duration t_{target} in seconds in conjunction with a non-negative tolerance value $t_{\text{tol}} < t_{\text{target}}$ in seconds are given by the user to constraint the duration of the final music piece to the interval

$$t_{\text{range}} = [t_{\text{min}}, t_{\text{max}}] = [t_{\text{target}} - t_{\text{tol}}, t_{\text{target}} + t_{\text{tol}}]. \tag{4.15}$$

These two parameters enable the user to control the trade-off between the accuracy of the produced result regarding the resulting duration and the quality of the discovered path: Higher values for t_{tol} lead to more valid solutions of which the best one is determined, but the duration of the resulting music piece can deviate more from the target duration.
The **length of a valid bixel path** is equivalent to the duration of a music piece assembled according to this bixel path, containing the first bixel in the path beginning at t_{start} and all remaining bixels in this path including the last bixel up to the end time t_{end}. More formally, the length can be expressed as a function $d(\mathbf{p})$ calculating the sum of the length of the bixels used, subtracted by the portions of the start and end bixels that are not included in the result because they lie before or after the specified start time t_{start} or end time t_{end}, respectively:

$$d(\mathbf{p}) = \sum_{i=1}^{k} b_{p_i}^{\text{len}} - (t_{\text{start}} - b_{i_{\text{start}}}^{\text{pos}}) - (b_{i_{\text{end}}+1}^{\text{pos}} - t_{\text{end}}). \tag{4.16}$$

Suppose we know the optimal bixel path $\mathbf{o}_k$ of all valid bixel paths with k bixels for every value of $k \geq 2$. Given the resulting length of a bixel path $d(\mathbf{p})$, we define the space of possible solutions as all possible values for k whose associated bixel path $\mathbf{o}_k$ leads to a result with a duration $d(\mathbf{o}_k)$ in the t_{range} interval. The optimisation problem then consists of selecting the value for k from this solution space for which the optimal bixel path $\mathbf{o}_k$ has the least cost $c_{\text{path}}(\mathbf{o}_k)$:

$$k_{opt} = \arg\min_{k\geq 2}\{c_{\text{path}}(\mathbf{o}_k) \mid d(\mathbf{o}_k) \in t_{\text{range}}\}. \tag{4.17}$$

In case no value for k fulfils this requirement, that is, there is no k for which $\mathbf{o}_k$ represents a path with a length within the defined interval t_{range}, the k value leading to the path deviating the least from the target duration t_{range} is selected instead:

$$k_{opt} = \arg\min_{k\geq 2}\{\min\{|d(\mathbf{o}_k) - t_{\min}|, |d(\mathbf{o}_k) - t_{\max}|\}\}. \tag{4.18}$$

However, the optimal path $\mathbf{o}_k$ can not be computed for every possible k as it can be arbitrarily large. For this reason, a range of possible values is estimated in the next section 4.3.2 by considering how long a bixel path with a specific amount of bixels can be. The corresponding optimal bixel paths $\mathbf{o}_k$ are then obtained by executing the path optimisation algorithm outlined in section 4.3.5.

4.3.2. Estimating the path length a priori

Unfortunately, the number of bixels needed to produce an output with a specific length can only be estimated, if bixel lengths are not completely equal, because selecting shorter bixels leads to a shorter output than concatenating the same number of longer bixels.

At first, we develop an upper limit for the number of bixels in a valid bixel path. The first bixel of a valid bixel path is always $b_{i_{\text{start}}}$, the last bixel is always $b_{i_{\text{end}}}$ and both contribute a constant amount of seconds $t_{\text{con}} \geq 0$ to the total length of the path, depending on the start and end times:

$$t_{\text{con}} = (b^{\text{pos}}_{i_{\text{start}}+1} - t_{\text{start}}) + (t_{\text{end}} - b^{\text{pos}}_{i_{\text{end}}}). \tag{4.19}$$

So all valid bixel paths only vary in their selection of the remaining $k' = k-2$ bixels. Because a valid bixel path's length must fulfil the user-specified target duration range t_{range} and the first and last bixel account for t_{con} seconds, these remaining k' bixels must have a length in the range

$$t'_{\text{range}} = [t'_{\min}, t'_{\max}] = [t_{\min} - t_{\text{con}}, t_{\max} - t_{\text{con}}]. \tag{4.20}$$

Because every one of these k' bixels contributes to the total duration with at least the length of the shortest bixel available and the maximum length specified is $t'_{\max}$, a theoretical upper limit of k' for this target duration is

$$k'_{\text{limit}} = \lceil \frac{t'_{\max}}{\min_i\{b_i^{\text{len}}\}} \rceil. \tag{4.21}$$

Although optimising over the whole possible range $k \in [2, k'_{\text{limit}} + 2]$ covers all valid solutions, it is also computationally expensive, because k'_{limit} is typically large when outliers in the form of particularly short bixels occur. Therefore, we propose an estimation method to significantly reduce the runtime by calculating the narrowest range $k_{\text{range}} = \{k_{\min}, ..., k_{\max}\}$ of values for k whose associated paths still contain an optimal solution with a probability higher than a threshold $p_{\min} \in [0, 1)$. This minimum success probability $p_{\min}$ is intended to provide a trade-off between path optimisation accuracy and runtime. Setting a suitable default value for the minimum probability $p_{\min}$ is discussed in the evaluation section 5.1.1.

In the remainder of this section, a statistical description of the potential length of a bixel by representing it as a *random variable* and based on its combinations an estimate of a bixel path's length will be obtained in the form of another random variable. The values of random variables are dependant on chance and are typically described by *probability distributions* (noted by the symbol $\sim$) that describe this relationship between probability and value. Additionally, let $P(X = a)$ denote the probability of a random variable X exhibiting a value equal to a.

We assume the length of the n-th randomly selected bixel from the available bixels $b_1, \ldots, b_N$ to be a *normally distributed* random variable

$$\begin{aligned} B_n &\sim \mathcal{N}(\mu_{\text{b}}, \sigma_{\text{b}}^2) && \text{where} \\ \mu_{\text{b}} &= \frac{1}{N}\sum_{i=1}^{N} b_i^{\text{len}} && \text{and} \\ \sigma_{\text{b}}^2 &= \frac{1}{N}\sum_{i=1}^{N} (b_i^{\text{len}} - \mu_{\text{b}})^2. \end{aligned} \tag{4.22}$$

The normal distribution in this case has an average equal to the average bixel length and the variance is calculated as the sum of quadratic distances of the bixel lengths from this average. For our example song, the probability distribution for B_n, which is the same for every n, is plotted in figure 4.10 (a) and exhibits a very narrow peak around the mean bixel length of 0.48, because the bixel lengths do not deviate much from this average.

Based on this statistical model of the length of a beat, consider then the random

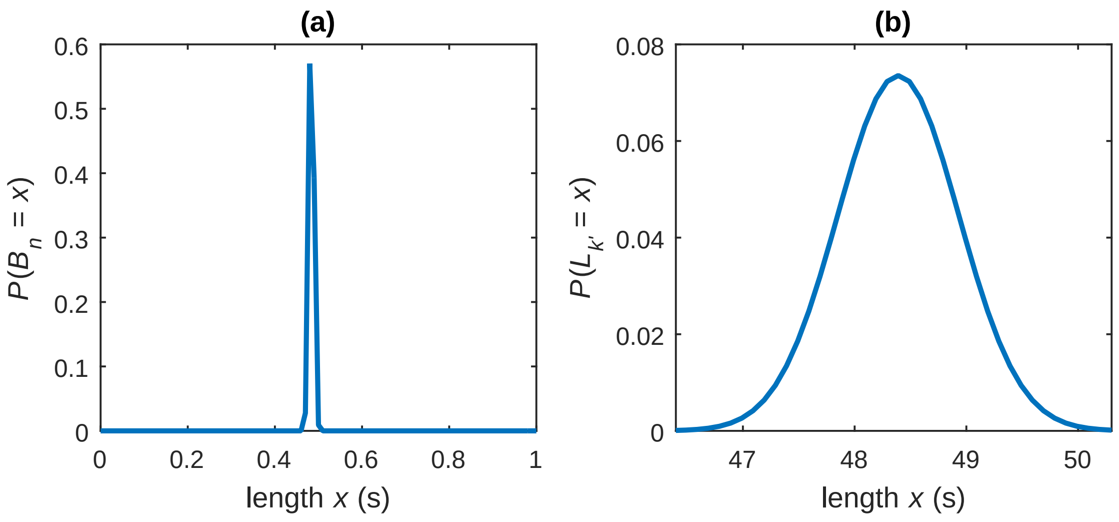


Figure 4.10.: Probability distribution of every random variable B_n representing the length of the n-th randomly selected bixel for the song ”El barzón” from ”Los Amparito” plotted in (a) and the probability distribution of the length of a path with k' bixels $L_{k'}$ for $k' = 100$ in (b).

variable $L_{k'}$ describing the length of a path containing k' bixels, consisting of the sum of all its bixel's lengths:

$$L_{k'} = \sum_{i=1}^{k'} B_i. \tag{4.23}$$

Thus, $L_{k'}$ describes the result of an experiment in which a random bixel is selected k' times and whose lengths are summed. In other words, $L_{k'}$ represents the length of a valid bixel path containing $k = k' + 2$ bixels, beginning with the start bixel $b_{i_{\text{start}}}$, followed by the k' randomly selected bixels and ending with the end bixel $b_{i_{\text{end}}}$, but subtracted by t_{con} seconds.

As the next step, we statistically represent the outcome of this experiment again using a normal distribution: Because the sum $Z = X + Y$ of two normally distributed random variables $X \sim N(\mu_1, \sigma_1^2)$ and $Y \sim N(\mu_2, \sigma_2^2)$ is itself a random variable $Z \sim N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$ following a normal distribution, $L_{k'}$ is also normally distributed:

$$L_{k'} \sim \mathcal{N}(k'\mu_{\text{b}}, k'\sigma_{\text{b}}^2). \tag{4.24}$$

The probability of a bixel path with k' bixels having a total length of l seconds is thus equal to $P(L_{k'} = l)$ according to our model. For $k' = 100$ bixels, the resulting probability distribution for L_{100} is plotted in the range where it deviates significantly from zero in figure 4.10 (b). With a very high probability, a random selection of $k' = 100$ bixels will amount to musical material with a total duration between 47 and 50 seconds.

Using this statistical basis, we estimate the probability that a valid bixel path with k bixels, which contains $k' = k - 2$ freely selectable bixels, has a length within the user-specified target duration range t_{range}. This probability is equal to the probability of k' randomly selected bixels having a total length in the range t'_{range} and is calculated for every $k \in [2, k'_{\text{limit}} + 2]$:

$$P_{\text{len}}(k) = P(t'_{\text{min}} \leq L_{k-2} \leq t'_{\text{max}}). \tag{4.25}$$

Note that beginning with this function $P_{\text{len}}(k)$, the estimation will from now on not be based on k' but rather k, which includes the first and last bixel of a valid bixel path and can be used by the path optimisation process described in section 4.3.5. This is achieved by referencing the random variable L_{k-2} as well as the corrected duration boundaries t'_{min} and t'_{max}. For visualisation purposes, setting $t'_{\text{range}} = [120, 130]$ for our example song results in a high probability for bixel paths with about 240 to 275 bixels to achieve a length between 120 and 130 seconds, as shown in figure 4.11 (a).

Under the assumption that optimal solutions are uniformly randomly distributed across the valid solutions, this probability distribution for valid solutions is now divided by its total probability to obtain the probability of an optimal solution at a

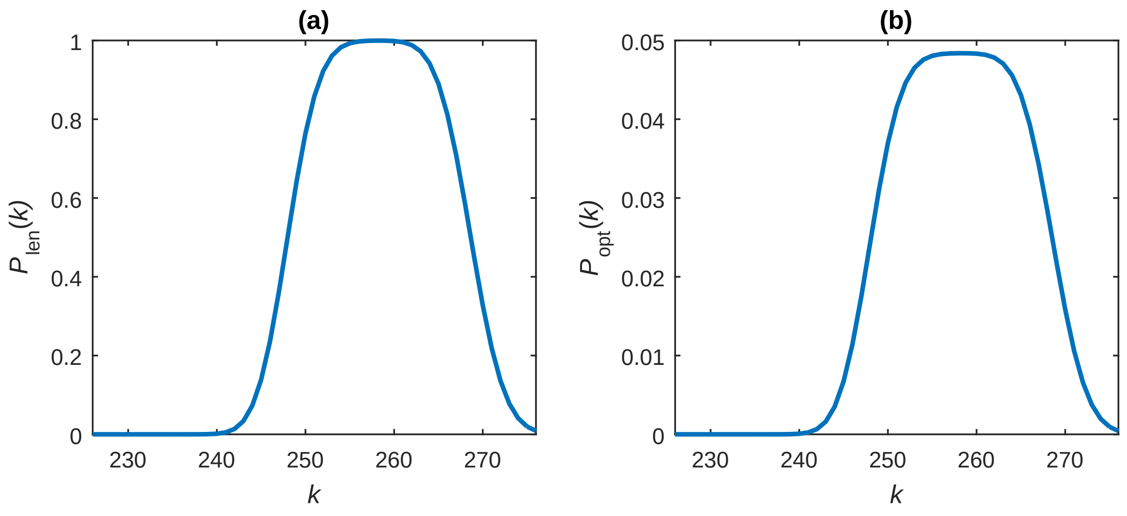


Figure 4.11.: Probability $P_{\text{len}}(k)$ of $k' = k - 2$ bixels having a length between 120 and 130 seconds plotted in (a) over the relevant k range for the song ”El barzón” from ”Los Amparito”. Corresponding probability distribution for P_{opt} for the optimal solution plotted in (b).

specific value of $k \in [2, k'_{\text{limit}} + 2]$:

$$P_{\text{opt}}(k) = \frac{P_{\text{len}}(k)}{\sum_{i=2}^{k'_{limit+2}} P_{\text{len}}(i)}. \tag{4.26}$$

In figure 4.11 (b), the resulting probabilities of $P_{\text{opt}}(k)$ are plotted for our exemplary settings.

The final task of choosing the k_{range} so that $k_{\max} - k_{\min}$ is minimised to reduce computational costs, while ensuring that the success probability

$$p_{\text{succ}} = \sum_{k=k_{\min}}^{k_{\max}} P_{\text{opt}}(k) \tag{4.27}$$

is greater than the parameter $p_{\min}$, is solved optimally by the algorithm 4.1.

To informally prove the optimality of this algorithm, we need to make the following observation about the function $P_{\text{opt}}(k)$. It has exactly one maximum and the function is monotonically increasing before and decreasing after this maximum. This is reasoned by its construction with the function $P_{\text{len}}(k)$, which calculates the area of the probability distribution functions corresponding to $L_{k'}$ that increase in their average using the constant target duration interval t'_{range} as bounds.

A key statement is that for every $d = k_{\max} - k_{\min} \geq 0$, the algorithm selects the $k_{\text{range}} = \{k_{\min}, \ldots, k_{\min} + d\}$ leading to the highest possible success probability p_{succ}. Beginning with $d = 0$, this can be proven by induction: In the first three lines of the algorithm, $k_{\min}$ and $k_{\max}$ are both set to the position of the maximum of

```
Input: $P_{\text{opt}}(k)$, $p_{\min}$
Output: $k_{\min}$, $k_{\max}$
 1: $k_{\min} = \arg\max_k \{P_{\text{opt}}(k)\}$
 2: $k_{\max} = k_{\min}$
 3: $p_{\text{curr}} = P_{\text{opt}}(k_{\min})$
 4: while $p_{\text{curr}} < p_{\min}$ do
 5:    if $P_{\text{opt}}(k_{\min} - 1) > P_{\text{opt}}(k_{\max} + 1)$ then
 6:       $k_{\min} = k_{\min} - 1$
 7:       $p_{\text{curr}} = p_{\text{curr}} + P_{\text{opt}}(k_{\min})$
 8:    else
 9:       $k_{\max} = k_{\max} + 1$
10:       $p_{\text{curr}} = p_{\text{curr}} + P_{\text{opt}}(k_{\max})$
11:    end if
12: end while
```

Algorithm 4.1: Selection of the optimal k_{range}

$P_{\text{opt}}(k)$, so no other k_{range} can provide a higher success probability. Assume that for an arbitrary, but constant value of d, the optimal k_{range} equals $\{\hat{k}_{\min}, \ldots, \hat{k}_{\max}\}$. Note that the range $\{\hat{k}_{\min}, \ldots, \hat{k}_{\max}\}$ has to contain the positions of all the d highest values of the function $P_{\text{opt}}(k)$ in order to optimal, because due to the mentioned property of $P_{\text{opt}}(k)$, they are always directly adjacent to each other and lie in one continuous interval. As a result, the optimal k_{range} for $d+1$ has to contain the $d+1$ highest values, so it must cover the whole previous k_{range} and additionally include the $d+1$-highest value, which has to be located at either $\hat{k}_{\min} - 1$ or $\hat{k}_{\max} + 1$. The algorithm executes this induction step in the loop starting at line four and exactly selects this required position after deciding between both candidates in line five. When constructing the k_{range} in this manner, the optimal k_{range} is found as soon as its corresponding success probability p_{succ} is at least $p_{\min}$, because the success probabilities for any k_{range} with a lower value of d were already computed and did not exceed the required threshold given by $p_{\min}$.
This concludes our informal proof of the optimality of the algorithm 4.1 and the description of how a range for the number of required bixels for a valid bixel path is calculated. In the next section, we will introduce a unified cost matrix designed to combine multiple user constraints into one, uniform representation.

4.3.3. Unified cost matrix

The following section defines the overall costs of transitioning from one bixel to another, unifying multiple constraints into a $N \times N$ unified cost matrix $\mathbf{T}'$. This allows not only for a compact description of constraints but also accelerates implementations of path optimisation, because instead of computing bixel transition costs by interpolating between multiple values from different matrices and vectors possibly multiple times for the same pair of bixels, just one access to retrieve the final value from the unified cost matrix is required.
One of the constraints represented in this unified cost matrix is called **importance** and is defined as a function $I(i) \in [0, 1]$ returning for all $i \in \{1, \ldots, N\}$ a value influencing the probability of the bixel b_i being included in the output [48]. The perception-based transition costs precomputed in the preprocessing stage for timbre in the matrix $\mathbf{D}'$ from section 4.2.1 and for loudness described by the matrix $\mathbf{L}$ in section 4.2.2 also have to be integrated into the music rearrangement system.
An observation already made in [48] is that the importance constraint can be merged with the perception-based transition costs. In a similar fashion, we merge the importance and both transition costs according to timbre and loudness mentioned above

into one unified cost matrix $\mathbf{T}'$ with

$$\begin{aligned} T'_{i,j} &= w_\mathrm{d} D'_{i,j} + w_\mathrm{l} L_{i,j} + w_\mathrm{p} I(i) \qquad &&\text{with} \\ w_\mathrm{d} + w_\mathrm{l} + w_\mathrm{p} &= 1 &&\text{and} \\ w_\mathrm{d}, w_\mathrm{l}, w_\mathrm{p} &\in [0,1], \end{aligned} \tag{4.28}$$

where the weight variables allow the user to control the influence of the different aspects on the selected jumps: In case the user does not want to work with the importance concept, $w_\mathrm{p} = 0$ prevents the importance function from having any influence on the result. If the produced cuts exhibit irritating changes in loudness, the user can increase the weight w_l of the loudness matrix $\mathbf{L}$, so the costs of jumps with these loudness differences increase and will be avoided by the subsequent path optimisation. Alternatively, the user can enable the loudness equalisation method in section 4.4.2 to not avoid these jumps, but equalise the produced loudness differences present in the constructed signal.

To prevent the bixels near the end of the song ”El barzón” from being used in the result for example, one can define an importance function as illustrated in figure 4.12 (a) to penalise the use of such bixels with a cost of one. For the user, the domain of this function is based on all time points in the song to allow for an intuitive design of the function independent of the notion of bixels. In our re-

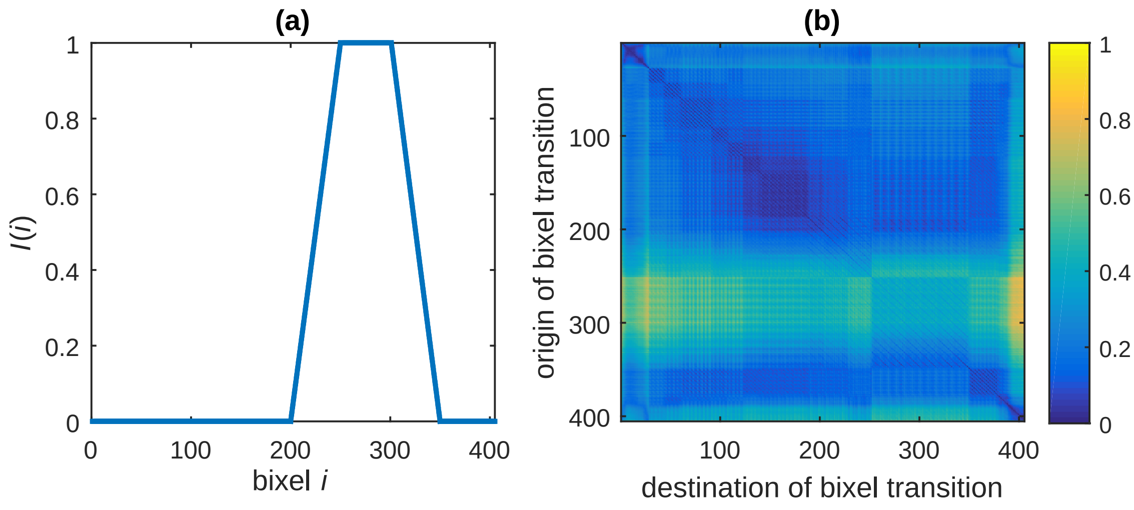


Figure 4.12.: Exemplary importance function for the song ”El barzón” from ”Los Amparito” in (a) describing the cost of including a specific bixel in the resulting song, making it very costly to use the bixels near the end of the song. Unified cost matrix $\mathbf{T}'$ displayed in (b) generated with the parameters $w_\mathrm{d} = 0.4$, $w_\mathrm{l} = 0.3$ and $w_\mathrm{p} = 0.3$, using the importance curve from (a).

structuring system, it is converted to the bixel-wise representation $I(i)$ by sampling the user-specified function at the center of every bixel. Afterwards, the importance function $I(i)$ is integrated according to the w_p parameter into the unified cost matrix $\mathbf{T}'$, making it more costly to include jumps originating from the bixels near the end, seen in figure 4.12 (b) as a light green, horizontal colouration of the matrix. As a result, solutions involving these bixels increase in cost, because any transition originating from such a bixel is costly - even selecting the successive bixel, which normally does not cost anything as mentioned in sections 4.2.1 and 4.2.2, now has a cost of $T'_{i,i+1} = w_\mathrm{p} I(i)$.
Overall and most importantly, the matrices $\mathbf{D}'$ and $\mathbf{L}$ describing the respective transition costs regarding timbre and loudness were merged into one unified transition cost matrix $\mathbf{T}'$. Please refer to section 5.2.1 for an investigation into suitable settings for the weights w_d and w_l of the transition cost matrices so that the perceived transition quality is as good as possible.

4.3.4. Repetition avoidance

We introduce a novel constraint to avoid highly repetitive results that occurred often during the evaluation of the method by Wenner [48] in section 2.4. These results use the same few bixels very often in direct succession and tend to not only be annoying for the user, but also misrepresent the original piece, as many of its sections are left out in the result. This is caused during the path optimisation stage by backward bixel jumps with a short distance and a low cost, which are utilised very often and still lead to a bixel path with a low cost that is finally selected as the optimal bixel path.
One possible approach to this problem is to modify transitions costs when considering the shortest path for every bixel visited during the path optimisation by calculating penalties for the transitions to all successor bixels based on how often and where they occur in this path. Although this is a very exact method to avoid the too frequent usage of a few bixels, it is also very computationally expensive as potentially millions of bixel transitions are considered that each require their own calculation.
Instead, we employ a simpler concept called *repetition avoidance*. The idea is to impose a high cost on backward bixel jumps, particularly those with a short distance that consequently are likely to cause a repetition of only a few bixels, to avoid an excessive usage of these jumps in the results. The cost is dependant on how many bixels the jump probably causes to be replayed. A bixel jump with a shift of $\delta = -x$

bixels is penalised with double the cost of a bixel jump with a shift equal to $\delta = -2x$. This works under the assumption that the influence of every backward jump in the solution on the length of the produced music piece is independant of all other jumps in the solution. More accurately, a bixel jump following a backward bixel jump is presumed to occur at the origin of this backward jump at the earliest, so that the number of repeated bixels is equal to its distance. If the next jump is a forward bixel jump occurring earlier than the origin of the backward bixel jump, the repeated segment is shorter than estimated. As a consequence, repetition avoidance is an approximative approach and does not guarantee the complete elimination of short repetitions.

Again, we aim to integrate constraints into one unified cost matrix, whenever possible: The matrix $\mathbf{T}'$ from the previous section 4.3.3 is modified along the diagonal and below to form a new $N \times N$ unified cost matrix $\mathbf{T}$:

$$T_{i,j} = \begin{cases} T'_{i,j} & \text{if } i < j \\ \min\{T'_{i,j} + c_{\text{rep}}(i - j + 1), 1\} & \text{else,} \end{cases} \tag{4.29}$$

where $c_{\text{rep}}(i)$ represents the penalty for a backward bixel jump with a distance of i that leads to an extension of the piece by i bixels:

$$c_{\text{rep}}(i) = \frac{w_{\text{r}}}{i}. \tag{4.30}$$

The variable $w_{\text{r}} \geq 0$ is a user-defined value controlling how strongly repetition avoidance is applied. Setting it to zero will lead to $\mathbf{T} = \mathbf{T}'$ and therefore disable repetition avoidance, while increasing values will penalise backward bixel jumps with a larger distance and to a greater extent.

A value of $w_{\text{r}} = 4$ results in a matrix depicted in figure 4.13 for our example, exhibiting costs near one along the diagonal and below, which decrease in the lower left direction (orthogonally to the diagonal). This makes solutions involving short backward bixel jumps, which are located along this diagonal, costly and thus less likely to be selected as the optimal path.

A suitable default value for w_{r} will be determined in the corresponding evaluation section 5.1.3.

The resulting unified cost matrix $\mathbf{T}$ is used for the path optimisation process as it combines importance, timbre, loudness perception and repetition avoidance with respective weighting parameters. It is recomputed every time a new song should be created with different constraint settings or weights and significantly reduces

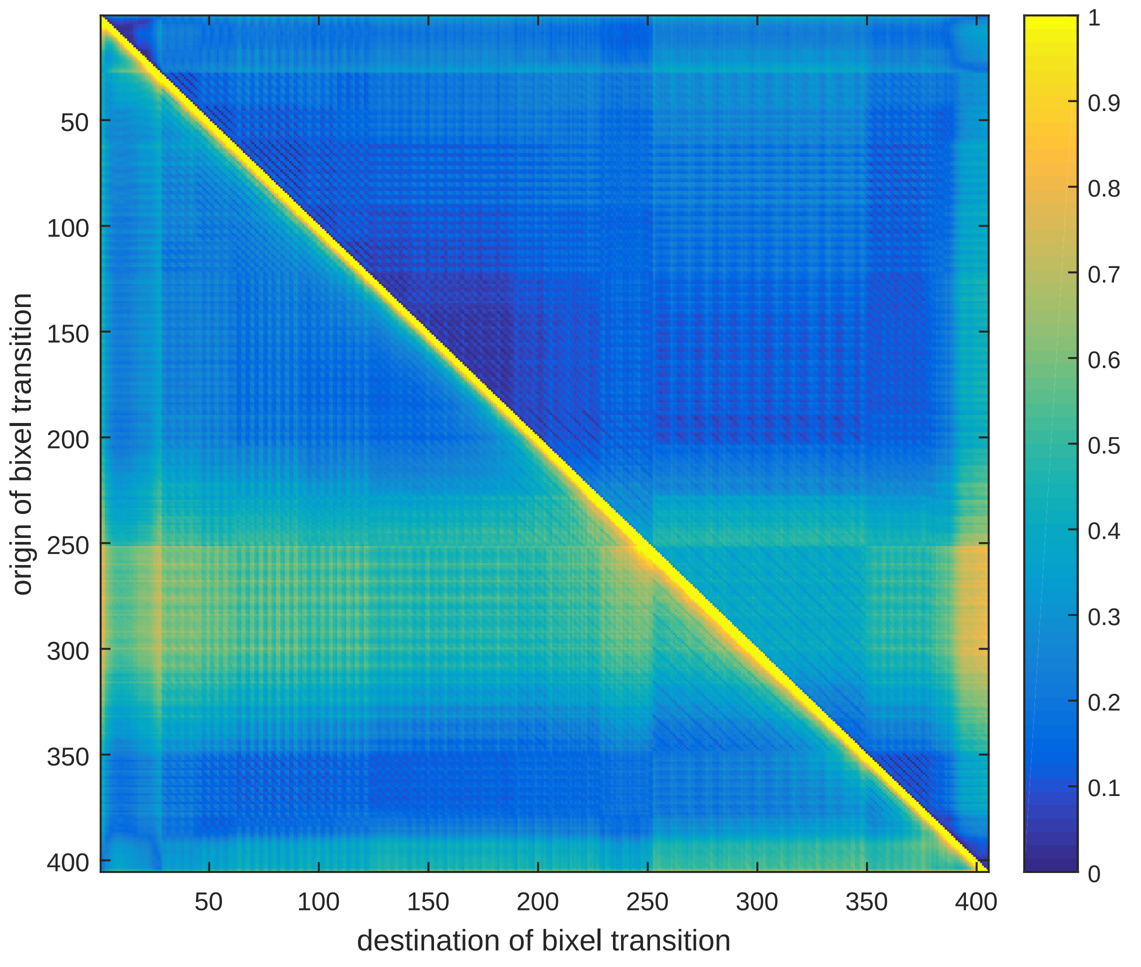


Figure 4.13.: Unified cost matrix **T** for the song "El barzón" from "Los Amparito" with a repetition avoidance setting of $w_\mathrm{r} = 4$, using the matrix $\mathbf{T}'$ from figure 4.12

the computation time of path optimisation, because every possible bixel transition cost is only calculated once and then stored for repeated access instead of being calculated multiple times.

4.3.5. Multiple goal A* algorithm

This section is concerned with obtaining the optimal bixel path based on the definitions from section 4.3.1. A novel algorithm will be proposed to accelerate this pathfinding process compared to the previous work in [48]. Both the proposed algorithm as well as the dynamic programming approach used in the previous work is implemented in C++ in our music rearrangement system, as it offers high performance and introduces low overhead that can be reduced even further by optimising the performed instructions on a low level.

The dynamic programming approach by Wenner [48] is based on a recursive formulation for the cost of the minimum path from bixel $b_{i_{\text{start}}}$ to any bixel b_i with k-stops:

$$C_{\text{start}}(i,k) = \min_{j \in \{1,\dots,N\}} \{C_{\text{start}}(j,k-1) + T_{j,i}\}. \tag{4.31}$$

After initialising all values $C_{\text{start}}(i,2)$ with the cost of all transitions originating from the start bixel $T_{i_{\text{start}},i}$, dynamic programming is used to iteratively compute $C_{\text{start}}(i,k)$ for each k, beginning with $k=3$ and ending with $k=k_{\max}$. The runtime complexity is $\Theta(N^2\ k_{\max})$ and needs $\Theta(N\ k_{\max})$ of space for the array holding the $C_{start}(i,k)$ values. Because the runtime grows quadratically with the amount of bixels in the original piece and therefore tendentially with the length of the original music piece, longer music causes long waiting times when restructuring. Additionally, all possible transitions at all positions in the bixel path are considered in the process, despite the possibility that some of these transitions make a bixel path too costly to be optimal and therefore do not have to be traversed.

In the remainder of this section, we will represent the problem as a *single-source shortest-paths problem* in a graph and propose a more efficient algorithm on this basis. Although the asymptotic complexity is still $O(N^2\ k_{\max})$ in the worst case, the runtime required in a real-world setting is often considerably lower than the dynamic programming approach from Wenner [48] as the evaluation section 5.1.2 shows, because fewer bixel transitions have to be taken into account while searching for the optimal bixel path.

Formulation as a graph problem We will represent every possible bixel path starting with the bixel $b_{i_{\text{start}}}$ with an equivalent path in a graph, enabling a graph algorithm that is presented later to traverse the graph in search of the optimal path. Consider a directed, weighted graph $G = (\mathcal{V}, \mathcal{E}, c_{\text{t}})$, where every vertex $v_{i,k}$ represents a bixel b_i in combination with the position k it is used in a bixel path and where every edge represents the act of transitioning from one bixel to another in such a bixel path along with its associated transition cost. If the edge costs are then set according to the transition costs contained in the unified cost matrix $\mathbf{T}$, the resulting graph G exactly represents our path optimisation problem.

More specifically, the vertices and edges as well as their costs are defined as

$$\begin{aligned}\mathcal{V} =&\{v_{i_{\text{start}},1}\} \cup \{v_{i,k} \mid i \in \{1,\ldots,N\} \wedge k \in \{2,\ldots,k_{\max}\}\},\\ \mathcal{E} =&\{(v_{i_{\text{start}},1}, v_{i,2}) \mid i \in \{1,\ldots,N\} \cup\\ &\{(v_{i,k}, v_{j,k+1}) \mid i,j \in \{1,\ldots,N\} \wedge k \in \{2,\ldots,k_{\max}-1\}\},\\ c_{\text{t}}(v_{i,k}, v_{j,k+1}) =&T_{i,j}.\end{aligned} \tag{4.32}$$

In this graph, every possible bixel path is represented by exactly one path originating from the node $v_{i_{\text{start}},1}$. All other nodes can be visually aligned on a grid as shown in figure 4.14 with the horizontal axis representing the number of bixels k and the vertical axis representing the original bixels.
Therefore, visiting a node $v_{i,k}$ is equivalent to selecting bixel b_i as the k-th bixel in a bixel path. Because only paths with $b_{i_{\text{end}}}$ as the last bixel are valid and the exact k value is not known beforehand, but rather a range of solutions across the k_{range} determined in section 4.3.2 should be computed, a set of goal nodes, which are highlighted in red in figure 4.14, is defined:

$$\mathcal{G}_{\text{end}} = \{v_{i_{\text{end}},k} \mid k \in k_{\text{range}}\}. \tag{4.33}$$

Note that the dynamic programming approach from Wenner [48] is equivalent to a

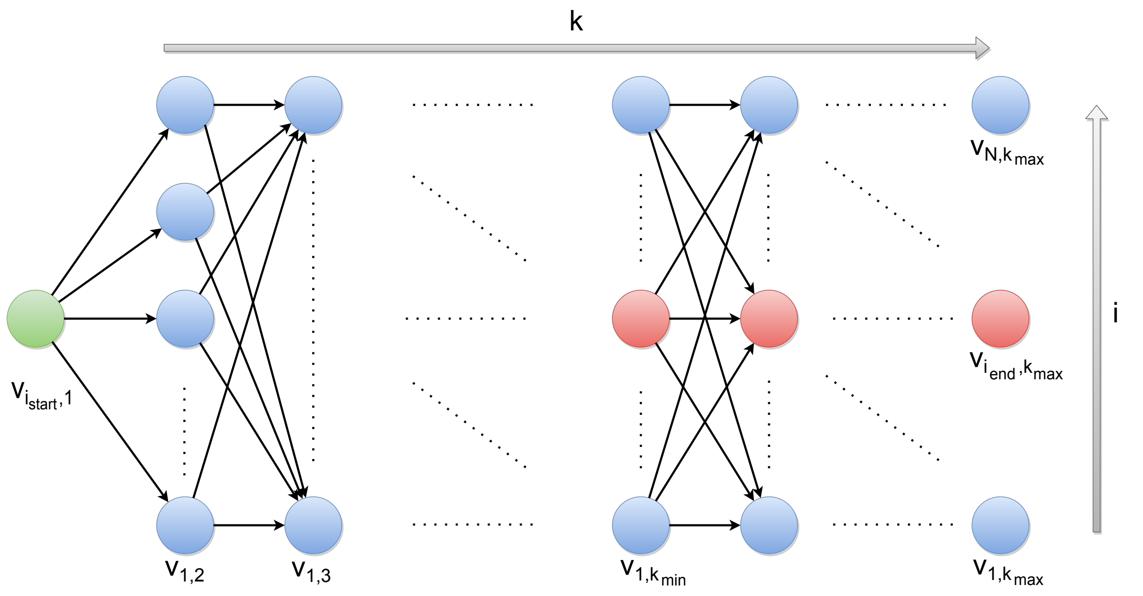


Figure 4.14.: The graph G representing the problem space with the current position in the considered bixel path k on the horizontal and the bixel number i of the included bixel on the vertical axis. The start node $v_{i_{\text{start}},1}$ is coloured in green, while the set of goal nodes $\mathcal{G}_{\text{end}}$ is coloured red.

graph algorithm calculating the shortest path to every node in the graph G in a column-wise manner by considering every one of its predecessor nodes. This results in $|\mathcal{E}| = N^2(k_{\max} - 2) + N$ operations, as every edge in the graph is used to check for a potentially better path. But in reality, some of these transitions are not used in the final solution due to their high cost. As implied in section 4.2.1, this is often the case for bixel jumps originating from the last bixel. Remember that bixel jumps in either direction always incur some cost, so a bixel path with a large amount of these jumps is not likely to be optimal - despite that, paths like these are taken into consideration and slow down the computation. This motivates the idea of minimising the amount of calculated transitions.

Graph algorithm selection and adaption With the given graph, the optimisation problem is equivalent to a single-source shortest paths problem with multiple goals, because the shortest path to any node contained in the set of goals has to be found from a given start node.

For this type of problem, *Dijkstra's algorithm* [11] is typically employed, which processes nodes in the order of their minimum distance to the start node. Consequently, edges leading to an already processed destination node do not have to be taken into account, because the shortest path from the start node to these processed nodes are already known and only worse paths can be found afterwards.

Another solution is known as the *A* algorithm* [20] and manages two sets of nodes, the *closed set* containing already evaluated notes to which the shortest path is known and the *open set* containing the set of tentative nodes still requiring an evaluation. It is a generalisation of Dijkstra's algorithm as it employs a *heuristic*, which estimates the minimum remaining distance from a node to the goal in order to process more "promising" nodes first, and behaves exactly like Dijkstra's algorithm if this heuristic function is set to zero. Assuming the heuristic function $\hat{h}$ used is *monotonic* (which is defined later in this section), the pseudo code 4.2 together with the termination condition check in pseudo code 4.3 describes the A* algorithm.

In the first four lines, the open and closed sets are initialised and the function g representing the distance from the start node for any given node as well as the estimate f returning an estimate for the total cost of a path from the start to the end goal through a given node are both set to return zero for the start node. Afterwards, a loop starts and first determines the most promising node with the lowest f value of all nodes from the open set (line six), then removes it from the open set (line seven) and afterwards adds it to the closed set (line eight), because the optimal path to

```
Input: Graph $G$, heuristic $\hat{h}$, start node $v_\mathrm{s}$, goal node $v_\mathrm{g}$
Output: Backtrackable shortest path given by $p_\mathrm{pred}$
 1: $\mathcal{V}_\mathrm{closed} = \emptyset$
 2: $\mathcal{V}_\mathrm{open} = \{v_s\}$
 3: $g(v_\mathrm{s}) = 0$
 4: $f(v_\mathrm{s}) = 0$
 5: while $\mathcal{V}_\mathrm{open} \neq \emptyset$ do
 6:   $v = \arg\min_{v \in \mathcal{V}_\mathrm{open}} \{f(v)\}$
 7:   $\mathcal{V}_\mathrm{open} = \mathcal{V}_\mathrm{open} \setminus v$
 8:   $\mathcal{V}_\mathrm{closed} = \mathcal{V}_\mathrm{closed} \cup v$
 9:   if goalCheck($\mathcal{V}_\mathrm{closed}$) then
10:     return $p_\mathrm{pred}$
11:   end if
12:   for each $e = (v, v') \in \mathcal{E}$ do
13:     if $v' \in \mathcal{V}_\mathrm{closed}$ then
14:       continue
15:     end if
16:     $g_\mathrm{curr} = g(v) + c_\mathrm{t}(e)$
17:     if $v' \notin \mathcal{V}_\mathrm{open} \vee g_\mathrm{curr} < g(v')$ then
18:       $\mathcal{V}_\mathrm{open} = \mathcal{V}_\mathrm{open} \cup v'$
19:       $p_\mathrm{pred}(v') = v$
20:       $g(v') = g_\mathrm{curr}$
21:       $f(v') = g_\mathrm{curr} + \hat{h}(v', v_\mathrm{g})$
22:     end if
23:   end for
24: end while
```

Algorithm 4.2: A* algorithm for a single goal with a monotonic heuristic $\hat{h}$

this node from the start node is known at this point. Then the function described by the pseudo code 4.3 is executed, checking whether the goal node v_g is now in the closed set, at which point the algorithm can stop and return the function p_pred that allows the optimal path to be retrieved via backtracking, starting at the goal node and retrieving the next node with this function until the start node is reached. Beginning in line twelve, all neighbours of the current node v not in the closed set are investigated in regards to a shorter path to this node through the current one: The length of the best possible path to the neighbour v' through v is computed in line 16 and line 17 checks if this solution is better than any previously calculated ones: If the neighbour is not in the open set or the current length g_curr is lower than the best found so far, we make sure it is now in the open set and assign v as the predecessor of v' in the optimal path and update the distance functions f and g with this new cost, additionally including the heuristic estimate for g.

```
Input: Current set of closed nodes $\mathcal{V}_{\text{closed}}$, goal node $v_{\text{g}}$
Output: Boolean indicating termination
1: function goalCheck
2: if $v_{\text{g}} \in \mathcal{V}_{\text{closed}}$ then
3:     return true
4: else
5:     return false
6: end if
7: end function
```

Algorithm 4.3: Function goalCheck as a a subroutine of the A* algorithm for a single goal, checking the termination condition

Note that reducing the function values of f in line 21 implies a different order of nodes in the open set $\mathcal{V}_{\text{open}}$ in which they are extracted in line six. A fast implementation of both of these instructions, especially the change of node value in line 21 as it is potentially executed very often, is a crucial factor for the runtime of the algorithm. Therefore, we chose the Fibonacci heap [17] as a data structure for maintaining the open set. It offers a constant *amortised* runtime for the *decreaseKey* operation that is executed in line 21 when the associated value of a node is reduced and the data structure holding the nodes therefore needs to be updated to maintain the order of nodes internally, so it can still provide a constant runtime for the minimum node extraction in line six. We use the ”Boost” C++ libraries [3], because they feature an efficient implementation of the Fibonacci heap.

While the A* algorithm is designed to find the shortest path to a single goal node, we defined a set of goals $\mathcal{G}_{\text{end}}$ in equation (4.33), for which we need to determine the optimal path. Consequently, we adapt the A* algorithm to a **multiple goal A* algorithm** able to compute the shortest path from a start node to *any* of the goal nodes. Instead of a single goal node v_{g}, the input consists of the set of goals defined in equation (4.33) and the goalCheck function is replaced by an altered version outlined in pseudo code 4.4. The termination condition now requires all goals in the goal set to be an element of the closed set, i.e. the optimal path to all goals has to be known. At that point, there exists no undiscovered, shorter path to any of the goal nodes and the shortest paths can thus be recovered by backtracking the paths starting from every goal node with the predecessor function p_{pred}. Finally, from all these shortest paths, the optimal bixel path is selected according to the equations (4.17) and (4.18).

As an additional modification, the heuristic function needs to be adapted to accommodate multiple goals. In the remainder of this section, this issue will be handled

```
Input: Current set of closed nodes $\mathcal{V}_{\text{closed}}$, set of goal nodes $\mathcal{G}_{\text{end}}$
Output: Boolean indicating termination
1: function goalCheck
2: if $\mathcal{G}_{\text{end}} \subseteq \mathcal{V}_{\text{closed}}$ then
3:     return true
4: else
5:     return false
6: end if
7: end function
```

Algorithm 4.4: Function goalCheck as a a subroutine of the multiple goal A* algorithm, checking the termination condition

after explaining the concept of a heuristic function in greater detail. Afterwards, several specific definitions of heuristics will be presented for the problem at hand.

Heuristics design

A **heuristic function** $\hat{h}(v, g)$ for the A* algorithm returns an estimate of the distance between a node v and a goal node g. This allows the algorithm to consider "promising" nodes first, while nodes with a presumably long distance to the goal are visited later in the process, or not at all. Because the heuristic must be evaluated for every visited node, its computational complexity should ideally be in $O(1)$, as otherwise the time required for computing the heuristic estimates in practice outweighs the time saved by visiting fewer nodes.

Formal requirements for heuristics For the correctness of the algorithm the heuristic used has to be **admissible**, that is, it never overestimates the actual distance h to the goal but rather provides a lower bound:

$$\forall v \in \mathcal{V} : \hat{h}(v, g) \leq h(v, g). \tag{4.34}$$

If the heuristic is not admissible, the A* algorithm could overlook the optimal solution due to overestimating the total cost of a path.
As already stated, the outlined pseudo code 4.2 only works for a *monotonic* heuristic. While an alternative implementation without this condition could be used, it is desirable to fulfil the monotonicity requirement, because then the paths calculated to nodes in the closed set are guaranteed to be optimal. Consequently, the evaluation of other paths leading to those nodes can be omitted during the remainder of the pathfinding process.

A heuristic is **monotonic**, if

$$\forall\ (v_1, v_2) \in \mathcal{E} : \hat{h}(v_1, g) \leq c_{\mathrm{t}}(v_1, v_2) + \hat{h}(v_2, g). \tag{4.35}$$

This means that for every node along a path, the total estimated cost consisting of the shortest distance to this node plus the heuristically estimated remaining distance to the goal can not decrease when travelling along this path.
Due to not knowing the optimal k value a priori, $\mathcal{G}_{\mathrm{end}}$ was defined in equation (4.33), representing the possible final nodes of a valid bixel path. As a consequence, the heuristic has to be adapted to be able to handle multiple goals. A **multiple goal heuristic function** $\hat{h}'$ for a given set of goals $\mathcal{G}$ describes the minimum distance from a node v to *any* of the goal nodes and can be constructed from a single goal heuristic $\hat{h}$ by selecting the minimum of $\hat{h}$ evaluated for every goal in $\mathcal{G}$ separately:

$$\hat{h}'(v, \mathcal{G}) = \min_{g \in \mathcal{G}} \{\hat{h}(v, g)\}. \tag{4.36}$$

After this theoretical introduction, we will present several concrete heuristics specifically tailored to our path optimisation problem.

Designing a problem-specific heuristic In many practical applications of the A* algorithm like real-world navigation, the nodes on the graph can be positioned in Euclidean space so that their Euclidean distances between each other in this space can be used as a simple and meaningful lower bound for the distances in the graph. Unfortunately, the costs for jumping between different bixels follow no such obvious rule, as the costs for bixel jumps in either direction are not only dependant on their distance – for example, jumping a few bixels ahead may incur a greater cost than jumping from the beginning to the end of the song - and vice versa. However, bixel jumps always cause *some* cost and therefore skipping a specific amount of bixels in total has some associated minimum cost, which will motivate the first of the three heuristics outlined in this section.

Travel distance heuristic The basic idea of the travel distance heuristic is to determine the minimum cost per bixel skipped in the forward or backward direction to estimate how ”efficient” any jump can be at most in terms of cost per distance. At runtime for a given node, the minimum amount of bixels the piece has to be lengthened or shortened to reach a goal node is determined and multiplied with the minimum cost per bixel skipped in the respective direction to yield a lower bound for the remaining cost.

First, we will introduce the heuristic for the case in which only a single goal exists for better comprehensibility before deriving the final, similar multi-goal heuristic.

In a preprocessing step taking place directly before the path optimisation is performed, the minimum costs for a bixel jump with a shift of $\delta \in [-N, N-2]$ are computed:

$$c_{\text{jump}}(\delta) = \min_{i,j \in \{1,\dots,N\}} \{T_{i,j} \mid j - (i+1) = \delta\}. \tag{4.37}$$

Figure 4.15 (a) shows the minimum costs $c_{\text{jump}}(\delta)$ for bixel jumps with a shift equal to a specific amount of bixels δ for our example song. In this case, the cost function is almost axisymmetrical, because the matrix $\mathbf{D}'$ used is derived from the symmetrical self-similarity matrices $\mathbf{S}'$ and $\mathbf{S}''$, as presented in section 4.2.1. The costs tend to increase as the distance increases, but fluctuate periodically, because jumping a whole measure (which is often four beats as discussed in section 2.1) often produces less perceptible cuts and has a lower associated cost.

In the next preprocessing step, the minimum cost of any bixel jump with a shift of $\delta \in [-N, N-2]$ per distance travelled is obtained after dividing these costs by the

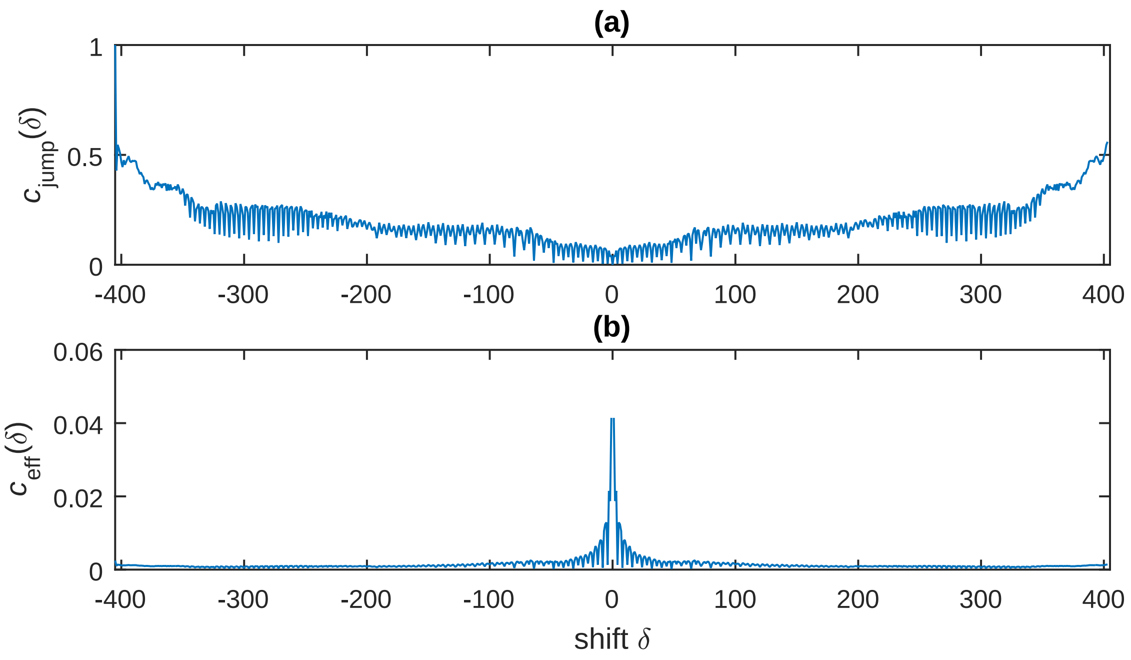


Figure 4.15.: Values of (a) $c_{\text{jump}}(\delta)$ and (b) $c_{\text{eff}}(\delta)$ both plotted for all possible shifts δ for the example song "El barzón" from "Los Amparito". The matrix $\mathbf{T}$ was generated with $w_{\text{l}} = w_{\text{p}} = w_{\text{r}} = 0$ and $w_{\text{d}} = 1$, leading to $\mathbf{T} = \mathbf{D}'$. $\delta = 0$ is calculated in (a), but not in (b) and is not required for the subsequent operations.

respective distance $|\delta|$ (except for $\delta = 0$ which stays undefined):

$$c_{\text{eff}}(\delta) = \frac{c_{\text{jump}}(\delta)}{|\delta|}. \tag{4.38}$$

In general, these minimum costs tend to decrease with increasing distance, which can be seen in figure 4.15 (b).

To complete the preprocessing phase, we determine how large the cost per distance in bixels has to be for any combination of jumps, or in other words, how efficiently "travel" between bixels is possible at most. This leads to c_{fwd} and c_{bwd} representing the minimum costs per bixel distance when jumping forwards or backwards:

$$\begin{aligned} c_{\text{fwd}} &= \min_{\delta \in [1, N-2]} \{c_{\text{eff}}(\delta)\} \qquad \text{and} \\ c_{\text{bwd}} &= \min_{\delta \in [-N, -1]} \{c_{\text{eff}}(\delta)\}. \end{aligned} \tag{4.39}$$

Based on these "maximum efficiency indicators", the travel distance heuristic is able to return a minimum cost given a node representing a part of a solution: We determine the amount of bixels this current solution needs to be shortened or lengthened to eventually reach the goal node without jumping and multiply this with the aforementioned costs per bixel c_{fwd} and c_{bwd}. For the example illustrated in figure 4.15, both equal to a minimum cost of about 0.000184 per "travelled" bixel. At runtime, these values are required for algorithm 4.5 to return a lower bound for the remaining distance from any node to a single goal node and constitutes our first problem-specific single-goal heuristic function $\hat{h}_1$ outlined in pseudo code 4.5.

```
Input: Considered node $v_{i,k}$ and goal node $g_{i',k'}$
Output: Lower bound $\hat{h}_1(v_{i,k}, g_{i',k'})$ for the distance between $v_{i,k}$ and $g_{i',k'}$
 1: $i_{\text{proj}} = i' - k' + k$
 2: if $k \leq k' \wedge i_{\text{proj}} - i = 0$ then
 3:     return 0
 4: else if $k < k' \wedge i_{\text{proj}} - i > 0$ then
 5:     return $(i_{\text{proj}} - i)\ c_{\text{fwd}}$
 6: else if $k < k' \wedge i_{\text{proj}} - i < 0$ then
 7:     return $(i - i_{\text{proj}})\ c_{\text{bwd}}$
 8: else
 9:     return $\infty$
10: end if
```

Algorithm 4.5: Computing the travel distance heuristic $\hat{h}_1$

In the first line of the algorithm, the goal is "projected" diagonally to obtain the position i_{proj} at the current k that would lead to the goal node without jumps. This theoretical position (which can be outside of this graph) can be compared to the current node in regards to its bixel number i to determine the amount of bixels that need to be skipped to reach the goal. In case this number of bixels is zero (line two) and the k position of the current node is at most the goal's k' position, zero is returned in line three, because not jumping at all and simply selecting the successive bixel leads to the goal does not cost anything in almost all cases (except when the importance function has a non-zero penalty for all bixels). Visually, this case occurs for all nodes along the diagonal aligned to the goal node with a k position at most that of the goal, which is illustrated in figure 4.16 in green. Otherwise, if the node is at a k position where jumping to the goal is still possible, the difference in bixel positions $i - i_{\text{proj}}$ is used to distinguish whether forward bixel jumps (lines four and five) or backward bixel jumps (lines six and seven) are necessary to reach the goal. Independant of the direction, the distance that has to be covered by these jumps is multiplied with the respective minimum costs per distance in bixels c_{fwd} or c_{bwd} to yield a minimum for the remaining cost. The dark blue area in figure 4.16 covers nodes for which jumping backwards is possible and necessary to reach the goal, while nodes in the light blue area require a forward jump. If $k > k'$ or $k = k'$ and the considered node is not the goal node, the goal was "missed" and can not be reached any more since one can only move "to the right" along the k axis in the graph, which is why an infinitely large distance is returned in line 9. This is illustrated in figure 4.16 by the yellow area.

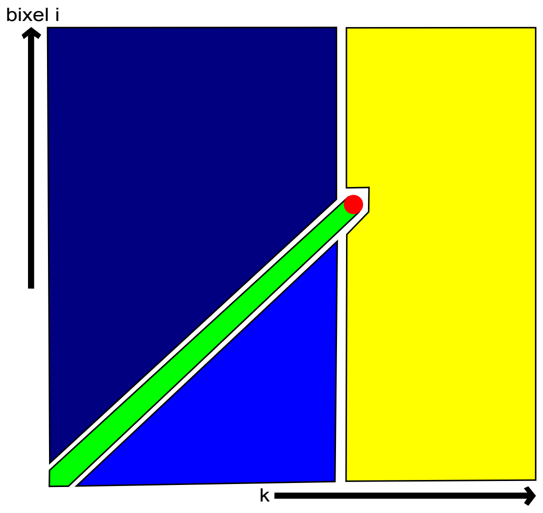


Figure 4.16.: The cases of the travel distance heuristic $\hat{h}_1$ illustrated with differently coloured areas of the graph G for the case of only one goal marked in red.

Due to our problem definition containing a set of goals, we need to extend this single goal heuristic to a multi-goal heuristic. As can be seen in the graph illustration in figure 4.14, the set of goals now forms a horizontal line that is projected to the current k value in the same manner as in the single goal heuristic, leading to two positions $i_{\min}$ and $i_{\max}$. Note that $i_{\min} \geq i_{\max}$, because a higher bixel index i leads to the goal line at an earlier k value. Algorithm 4.6 calculates the heuristic $\hat{h}'_1(v, \mathcal{G}_{\text{end}})$ as the first of our three multi-goal heuristics.

Analogously to the single goal heuristic, we illustrate the different cases of the multi-goal version with differently coloured areas in figure 4.17. Similar to the last case in the single-goal heuristic, no goal can be reached any more if we are not at $b_{i_{\text{end}}}$ at the last possible k value $k_{\max}$ (yellow areas), so an infinitely high distance is returned in line two. The following line three checks for the case in which the goal lies before the current bixel and the current k value falls into the k_{range} (pink rectangle) and therefore a backward jump with a distance equal to the difference between the current bixel and the goal bixel $b_{i_{\text{end}}}$ leads to the goal. If the current node does not fall into these two categories, $i_{\max}$ is computed in line six and used to check if a forward jump is needed in line seven (light blue area). Should no forward jump be needed, $i_{\min}$ is calculated to distinguish the case where a backward jump is needed (dark blue area) in lines eleven and twelve from the case where no jump is necessary at all and zero can be returned in line 14. The two cases for backward jumps in lines four and twelve differ in that up to $k_{\min}$, estimates of nodes along a diagonal in the dark blue area are equal due to the comparison based on the "diagonal projection", while nodes representing bixels behind $b_{i_{\text{end}}}$ in the pink rectangle have equal estimates when located on the same horizontal line parallel to the goal nodes.

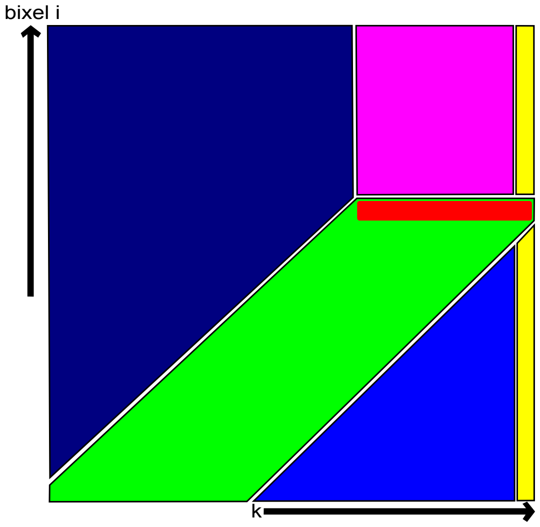


Figure 4.17.: The cases of the travel distance heuristic h_1 illustrated with differently coloured areas of the graph G with the set of goals $\mathcal{G}_{\text{end}}$ in red.

All in all, the travel distance heuristic avoids the consideration of bixel paths containing **wrong jumps**, i.e., forward bixel jumps when the song's duration should be increased or vice versa, because the heuristic estimate increases for the nodes to which those jumps lead to. However, the heuristic offers almost no information about *when* to consider a jump when traversing nodes diagonally along the graph. Both the blue and the green area exhibit equal estimates along the diagonals – this will motivate the second heuristic of this thesis.

```
Input: Considered node $v_{i,k}$ and set of goals $\mathcal{G}_{\text{end}}$ defined in equation (4.33)
Output: Minimum distance $\hat{h}'_1(v_{i,k}, \mathcal{G}_{\text{end}})$ from $v_{i,k}$ to any goal in $\mathcal{G}_{\text{end}}$
 1: if $k = k_{\max} \wedge i \neq i_{\text{end}}$ then
 2:    return $\infty$
 3: else if $i > i_{\text{end}} \wedge k \geq k_{\min}$ then
 4:    return $(i - i_{\text{end}})\ c_{\text{bwd}}$
 5: else
 6:    $i_{\max} = i_{\text{end}} - k_{\max} + k$
 7:    if $i < i_{\max}$ then
 8:       return $(i_{\max} - i)\ c_{\text{fwd}}$
 9:    else
10:       $i_{\min} = i_{\text{end}} - k_{\min} + k$
11:       if $i > i_{\min}$ then
12:          return $(i - i_{\min})\ c_{\text{bwd}}$
13:       else
14:          return $0$
15:       end if
16:    end if
17: end if
```

Algorithm 4.6: Computing the travel distance heuristic $\hat{h}'_1$

Jump necessity heuristic While the travel distance heuristic is concerned with avoid forward jumps when the input track should be extended and vice versa, it does not specify where along a diagonal of nodes the jump should occur. The jump necessity heuristic is added as a complement in case a jump is necessary, which does not change in its estimate across different k with the same bixel, but rather across a diagonal – this should prevent the A* algorithm from traversing a whole diagonal of the graph despite the possible necessity to jump earlier due to the potentially high cost of jumps with origins near the end.

In a preprocessing step, the minimum cost of jumping from any bixel $b_i, i \in \{1, \ldots, N\}$ to any other bixel (except the successor as this is not considered jumping) is calculated:

$$c_{\text{orig}}(i) = \min_{j \in \{1, \ldots, N\}} \{T_{i,j} \mid j \neq i + 1\}. \tag{4.40}$$

For the song "El barzón" from "Los Amparito", the values of this function are plotted in figure 4.18 (a). Large values at the end of the song are especially noticeable. The jump necessity heuristic exploits this fact to prevent the pathfinding process from including such bixels near the end as jumping away from this region is costly.

In the next step, the minimum costs for jumping with a varying interval $[i, j]$ of

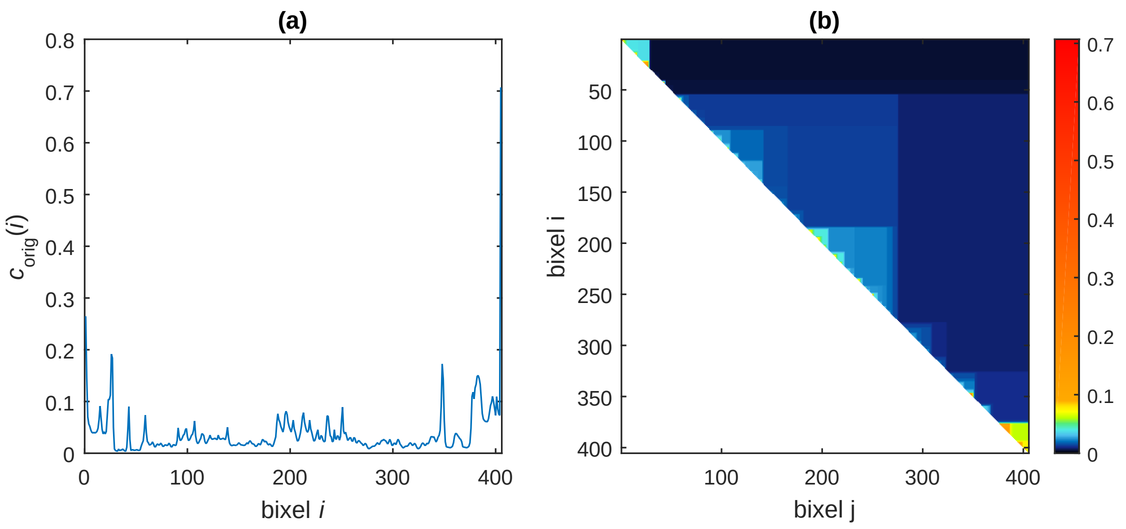


Figure 4.18.: Values of (a) $c_{\text{orig}}(i)$ plotted for every bixel b_i and (b) $c_{\text{inter}}(i, j)$ visualised with a heat map that focuses on the values between 0 and 0.1 (the white area is outside of the function's domain) and assigns i to the vertical and j to the horizontal axis. In both illustrations, the input song is "El barzón" from "Los Amparito". The required matrix $\mathbf{T}$ was generated with $w_{\text{l}} = w_{\text{p}} = w_{\text{r}} = 0$ and $w_{\text{d}} = 1$, leading to $\mathbf{T} = \mathbf{D}'$.

possible bixel origins are stored for all $i, j \in \{1, \ldots, N\}$ with $i \leq j$ after calculating them according to the following formula:

$$c_{\text{inter}}(i, j) = \min_{i \leq b \leq j} \{c_{\text{orig}}(b)\}. \tag{4.41}$$

The result of $c_{\text{inter}}(i, j)$ describes the minimum cost of all bixel jumps whose origin bixel's index is in the $[i, j]$ interval. It can be accessed in constant time when executing the A* algorithm, because it was precomputed and saved. For our example song, the function is illustrated with a heat map in figure 4.18 (b). To make the generally low values more distinguishable, a large number of different colours is used for the range $[0, 0.1]$, while the remaining values are coloured with a mix of orange and red. The white area does not belong to the domain of the function c_{inter} due to the restriction $i \leq j$.

At runtime, the jump necessity heuristic can be evaluated for the case of multiple goals by executing algorithm 4.7, making use of the function c_{inter} to obtain the minimum costs of jumping out of the considered bixel range.

At first, the projected bixel positions $i_{\max}$ and $i_{\min}$ are calculated in lines one and two. The following conditional statements in lines three to six cover the cases in which the cost is either zero or infinitely high and are semantically equivalent to

```
Input: Considered node $v_{i,k}$ and set of goals $\mathcal{G}_{\text{end}}$ defined in equation (4.33)
Output: Minimum distance $\hat{h}'_2(v_{i,k}, \mathcal{G}_{\text{end}})$ from $v_{i,k}$ to any goal in $\mathcal{G}_{\text{end}}$
 1: $i_{\max} = i_{\text{end}} - k_{\max} + k$
 2: $i_{\min} = i_{\text{end}} - k_{\min} + k$
 3: if $i_{\max} \leq i \leq i_{\min} \wedge i \leq i_{\text{end}}$ then
 4:     return $0$
 5: else if $k = k_{\max} \wedge i \neq i_{\text{end}}$ then
 6:     return $\infty$
 7: else
 8:     $k_{\text{last}} = k + N - i$
 9:     $i_{\text{last}} = i + \min\{k_{\text{last}}, k_{\max} - 1\} - k$
10:     return $c_{\text{inter}}(i, i_{\text{last}})$
11: end if
```

Algorithm 4.7: Algorithm computing the jump necessity heuristic $\hat{h}'_2(v, \mathcal{G}_{\text{end}})$

those used in the travel distance heuristic. Should the considered node not belong to one of these cases, a path to a goal exists, but a jump is necessary to reach that goal. To determine the minimum costs of jumping in this case, the interval of possible jump origins is computed in lines eight and nine: In line eight, the k value at which the last bixel would be reached when starting from the considered node $v_{i,k}$ and making no jumps called k_{last} is determined. With this information, the range of k values at which we have to jump to reach the goal starts at the current position k and ends at either k_{last}, because jumping is inevitable at the last bixel, or at $k_{\max} - 1$, because there are no goals reachable any more after that. Whichever possibility arises first determines the last position at which jumping allows us to reach the goal. Therefore, we take the minimum of these positions and use it to compute i_{last} in line nine representing the corresponding bixel at this position that would be reached if no jumping had yet occurred (again a "diagonal projection"). Finally, equipped with the knowledge that at least one jump has to be performed at $k' \in [k, \min\{k_{\text{last}}, k_{\max} - 1\}]$ whose origin is a bixel with index between i and i_{last}, inclusively, we return the minimum cost for such a jump in line ten, which is possible in constant time as the function c_{inter} was precomputed.

Jump combination heuristic The last heuristic of the total of three developed in this thesis is called jump combination heuristic. Generally speaking, it extends the travel distance heuristic to yield a more powerful heuristic with distance estimates that are at least as high, if not higher. Again the total shift required to reach the goal and the minimum associated cost $c_{\text{jump}}(\delta)$ for a jump with a specific shift δ from equation (4.37) is used, but this time the possible combinations of jumps with

different shifts that sum up exactly to the required total shift are analysed and their minimum cost determined. As an example for one of these combinations, jumping $\delta = 10$ bixels forward can be achieved by jumping five bixels forward two times, or by jumping 15 bixels forward and then jumping backward with a shift of -5. Over all possible combinations, the optimal combination of jumps with least total cost must be computed.
A way to phrase and solve this problem is by representing it as an *integer linear program* (ILP) [32]. Let $x_i \in \mathbb{N}_0$ with $i \in \{-N, \ldots, N-2\}$ be the decision variable indicating how often a bixel jump with shift i is selected to achieve the total shift δ. Then the integer linear program minimising the total cost of all selected bixel jumps can be written as

$$\begin{aligned} \text{minimise} \quad & \sum_{i=-N}^{N-2} c_{\text{jump}}(i) x_i \\ \text{subject to} \quad & \sum_{i=-N}^{N-2} i \cdot x_i = \delta, \\ & \text{and } \forall\ i \in \{-N, \ldots, N-2\} : x_i \geq 0. \end{aligned} \tag{4.42}$$

The summation constraint ensures that the resulting position in the graph is changed exactly δ bixels after performing all the jumps encoded in the x_i variables. Finding the optimal solution to this ILP for a specific δ yields a lower bound for changing the length of the solution by δ bixels. Theoretically, this ILP can be solved for every value of δ that is needed while path optimisation is performed and the jump combination heuristic can use its solution to provide a lower bound for the distance to a goal in a similar fashion to the travel distance heuristic by determining the interval of possible δ values that would lead to a goal without any further jumps and selecting the minimum associated cost of them. In practice, solving an ILP takes far too much time to be feasible to use as part of a heuristic – the time needed to determine the optimal solution for all considered nodes exceeds the time saved by not having to visit some nodes and thus actually increases the overall runtime.

Selecting and combining heuristics The three heuristics outlined above were each conceived in an attempt to provide an admissible and consistent heuristic that provides a good estimate to reduce the total amount of considered nodes as much as possible. Although the jump combination heuristic is better in that regard compared to the travel distance heuristic, yielding an estimate at least as high for all possible inputs and sometimes an even higher one, the complexity of calculating the heuristic itself has to be taken into account. To avoid causing a high overhead to the A* algorithm, the heuristic should be computable in $O(1)$ at runtime. This renders the powerful jump combination heuristic unusable in practice, as the ILP defined in (4.42) has to be solved for every occurring value of δ, causing a higher overall computation time than without a heuristic. Additionally, this computation can not be executed during the preprocessing stage in section 4.2 to retrieve the ILP solutions in constant time during path optimisation, as the range of potentially required δ values depends on the theoretically unbounded $k_{\max}$, which can not be evaluated before the user selects a target length specification t_{range}.

The travel distance and jump necessity heuristics are not only computable in constant time for every node thanks to the employed preprocessing, but also complement each other and are consequently combined into one heuristic $\hat{h}'$, which will be used in conjunction with the multiple goal A* algorithm in our music rearrangement system:

$$\hat{h}'(v_{i,k}, \mathcal{G}_{\text{end}}) = \max\{\hat{h}'_1(v_{i,k}, \mathcal{G}_{\text{end}}), \hat{h}'_2(v_{i,k}, \mathcal{G}_{\text{end}})\}. \tag{4.43}$$

The maximum operator is applied to both heuristics to select the highest and therefore most meaningful estimate of the minimum distance between the two nodes.

4.3.6. Segmentation enforcement with tolerances

The user can define a ground truth segmentation in combination with a target segmentation to enforce a specific structural constraint on the resulting song. For example, annotating the occurrences of the verses and choruses in a song by assigning them to different clusters and then only allowing the use of the choruses in the target segmentation results in a new song exclusively containing parts from the choruses of the original song. This section focuses on how the target segmentation is enforced in the context of the music restructuring system.

At first, the ground truth segmentation constructed in section 4.2.3 in the form of the function $s_{\text{ground}}(t)$ is converted into a discretely parametrised function $s_{\text{b}}(i)$ assigning a cluster to every bixel b_i:

$$s_{\text{b}}(i) = s_{\text{ground}}(b_i^{\text{pos}} + \frac{b_i^{\text{len}}}{2}), \quad i \in \{1, \dots, N\}. \tag{4.44}$$

As a result, the cluster of every bixel is equal to the cluster enforced at its centre.

In the previous work from Wenner [48], not only the ground truth segmentation is defined on a bixel-wise basis, but also the target segmentation: A bixel path $\mathbf{p}$ as defined in equation (4.13) is only valid, if it additionally has a specific amount of bixels k and the segment of every bixel b_{p_i} used belongs exactly to the cluster specified by the target segmentation. Depending on the currently required cluster, the used transition cost matrix is manipulated to only allow for transitions to bixels belonging to this cluster. This formulation severely limits the amount of valid paths, because the transition from one cluster of segments to another is forced to occur exactly at one specific position in the bixel path. Restricting the amount of valid paths also tends to produce solutions with higher costs and therefore theoretically with more perceptible transitions, because paths with a lower cost $c_{\text{path}}(\mathbf{p})$ that even slightly deviate from the segmentation constraint are excluded.

Additionally, defining the target segmentation as a list of cluster indices constraining the cluster of every bixel in the resulting bixel path is unintuitive for the user. Thus, we allow the user to set points in time as segment boundaries in the exact same way the boundaries of the ground truth segmentation can be set. The resulting target segmentation $s_{\text{target}}(t)$ denotes the cluster that should be enforced at every point in time t. Note that t is unbounded, as the first or last cluster is returned for the infinitely large regions located before the first and after the last segment transition, respectively.

For many applications, e.g. when creating a remix of an existing song, the positions of the segment transitions do not need to be very exact. Consequently, we introduce a concept for segmentation enforcement with the tolerance variables t_{low} and t_{high}, where both represent non-negative, real numbers and $t_{\text{low}} \leq t_{\text{high}}$. These allow the resulting song to have segment transitions that deviate up to t_{high} seconds from the target positions. A deviation of t_{low} seconds is not penalised by an increased transition cost between the involved segments, whereas a deviation between t_{low} and t_{high} seconds is penalised according to a linearly increasing function starting at zero for t_{low} and ending at one for t_{high}.

Similar to the approach from Wenner [48], the segmentation enforcement is achieved by manipulating the transition cost matrix. However, we do not make this manipulation dependant on the current position in the bixel path, but rather on the current length consisting of all previously selected bixels during path optimisation: Mathematically, we model the transition costs ensuring the segmentation constraint as a function $T(i, j, t)$, where $i, j \in \{1, \dots, N\}$ identify the origin and destination bixels of the considered bixel transition and t in seconds describes the current position in the resulting song to calculate which cluster should be enforced. Suppose t' is the current position in the new song during path optimisation, calculated by summing the lengths of the previously chosen bixels, including the last bixel b_i. Additionally, let the nearest required segment transition be at position t_{near} from segments belonging to the cluster c_x to segments assigned to cluster c_y. In the special case that no segment transition exists at all, t_{near} is set to ∞, c_x to the currently enforced cluster $s_{\text{target}}(t')$ and c_y stays undefined, as it is not needed. Also assume that the next segment transition has to take place in t_{next} seconds, which is set to ∞ in case no such transition exists.

We define a piecewise linear fading function $f_{\text{fade}}(t) \in [0, 1]$ that returns zero representing a forbidden change of clusters and one if transitioning should be allowed without an additional penalty. Values in between should result in an additional cost that linearly depends on the distance to the segment transition under consideration:

$$f_{\text{fade}}(t) = \begin{cases} 1 & \text{if } |t| \leq t_{\text{low}} \\ \frac{t_{\text{high}} - |t|}{t_{\text{high}} - t_{\text{low}}} & \text{if } t_{\text{low}} < |t| \leq t_{\text{high}} \\ 0 & \text{else} \end{cases} \tag{4.45}$$

In figure 4.19, the fade function is plotted with the default tolerance settings used in the restructuring system. During path optimisation, the function is aligned horizontally so that $t = 0$ represents the location of the nearest segment transition in the target segmentation. In the near vicinity of t_{low} seconds, the function returns one to indicate that no penalty is applied for jumps resulting in a change of clusters. Deviations between t_{low} and t_{high} seconds from the specified transition incur a penalty depending on their distance to the segment transition. If the current position and the nearest segment transition are more than t_{high} seconds apart, the function returns zero to only allow bixel transitions whose destinations belong to the currently enforced cluster.

Note that setting both t_{low} and t_{high} to zero essentially simulates the method proposed by Wenner [48] by not allowing any tolerance regarding deviations from the

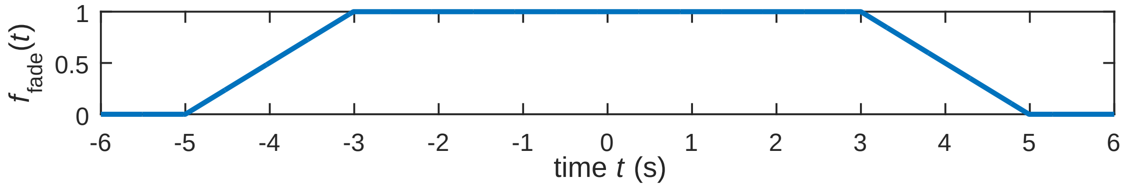


Figure 4.19.: Values of the fade function $f_{\text{fade}}(t)$ on the vertical axis plotted against time t on the horizontal axis with $t_{\text{low}} = 3$ and $t_{\text{high}} = 5$ seconds.

target segmentation aside from the inevitable inaccuracies caused by working on a bixel basis.

Mathematically, horizontally aligning the fade function f_{fade} with respect to the nearest segment transition at t_{near} is realised by using $t = t' - t_{\text{near}}$ as input.

After computing all of the variables mentioned above for the current length t' of the bixel path during path optimisation, the algorithm 4.8 computes the new transition cost $d = T(i, j, t')$ for a bixel transition from b_i to b_j. Line one checks whether the considered successor bixel b_j does not overlap with the following segment, but rather completely lies in the current segment. If this is the case, line two determines if the current position is near enough to a segment transition that the fading concept needs to be applied. In this case, lines three and four as well as lines seven and eight return the normal values from the transition cost matrix $\mathbf{T}$ if the bixels b_i and b_j are both assigned to the cluster before or after the transition. Jumping from the cluster before to the cluster after the transition is handled in lines five and six, using the fade function f_{fade} with the distance to the segment transition as input to linearly interpolate between the respective transition cost $T_{i,j}$ and one to incur additional cost, if necessary. Transitions with bixels involving any other combination of clusters are assigned an infinitely high cost in line ten to enforce the usage of bixels whose segments belong to the desired clusters.

If the current bixel lies completely inside the current segment and the current position is not near a segment transition, we simply need to allow transitions involving only bixels from the currently enforced cluster $s_{\text{target}}(t')$ in lines 13 and 14 and prevent all other bixels being used by assigning infinite costs in line 16.

In case the bixel b_j would overlap with one or more of the following segment transitions when appending it to the current bixel path, the enforced cluster s_{end} at the position $t' + b_j^{\text{len}}$ where the bixel path would end after appending this bixel b_j is calculated in line 20. All bixel transitions within this cluster s_{end} or the current cluster as well as transitions from the current to this cluster s_{end} are left unchanged in their costs and the fade concept is not used, so a segment change at exactly this position in the bixel path can be performed without penalties in lines 21 and 22. Every other bixel transition is forbidden in line 25. The additional exception handling beginning in line 20 covers cases where a large candidate bixel b_j overlaps with the upcoming segment transition and therefore the next cluster needs to be allowed, but the nearest segment transition is positioned before the current position t' and consequently the algorithm would not take this upcoming cluster into consideration.

Applying this algorithm during path optimisation every time a transition cost needs

```
Input: Considered bixel transition from bixel $b_i$ to bixel $b_j$, current time $t'$, time of
    the nearest segment transition $t_{\text{near}}$, enforced clusters $c_x$ and $c_y$ before and after
    the nearest segment transition, time of next segment transition $t_{\text{next}}$
Output: Transition cost $d$
 1: if $t' + b_j^{\text{len}} < t_{\text{next}}$ then
 2:   if $|t' - t_{\text{near}}| \leq t_{\text{high}}$ then
 3:     if $s_{\text{b}}(i) = c_x \wedge s_{\text{b}}(j) = c_x$ then
 4:       return $T_{i,j}$
 5:     else if $s_{\text{b}}(i) = c_x \wedge s_{\text{b}}(j) = c_y$ then
 6:       return $f_{\text{fade}}(t' - t_{\text{near}})\ T_{i,j} + (1 - f_{\text{fade}}(t' - t_{\text{near}}))$
 7:     else if $s_{\text{b}}(i) = c_y \wedge s_{\text{b}}(j) = c_y$ then
 8:       return $T_{i,j}$
 9:     else
10:       return $\infty$
11:     end if
12:   else
13:     if $s_{\text{b}}(i) = s_{\text{target}}(t') \wedge s_{\text{b}}(j) = s_{\text{target}}(t')$ then
14:       return $T_{i,j}$
15:     else
16:       return $\infty$
17:     end if
18:   end if
19: else
20:   $s_{\text{end}} = s_{\text{target}}(t' + b_j^{\text{len}})$
21:   if $(s_{\text{target}}(t') = s_{\text{b}}(i) \wedge s_{\text{end}} = s_{\text{b}}(i)) \vee (s_{\text{target}}(t') = s_{\text{b}}(i) \wedge s_{\text{end}} = s_{\text{b}}(j)) \vee$
      $(s_{\text{target}}(t') = s_{\text{b}}(j) \wedge s_{\text{end}} = s_{\text{b}}(j))$ then
22:     return $T_{i,j}$
23:   else
24:     return $\infty$
25:   end if
26: end if
```

Algorithm 4.8: Segmentation enforcement by manipulating transition costs

to be calculated leads to paths which always fulfil the given segmentation constraints, if possible: Setting the start bixel $b_{i_{\text{start}}}$ to a bixel belonging to a different cluster than the one enforced at the beginning of the target segmentation is an example for conflicting constraints resulting in a solution with a high cost because of inevitable constraint violation.

An application example in figure 4.20 for this segmentation enforcement demonstrates the effect of the tolerances t_{low} and t_{high}. When comparing the positions of the segment transitions, the segmentation of the result produced with three and five seconds as segmentation tolerances deviates slightly from the target segmenta-

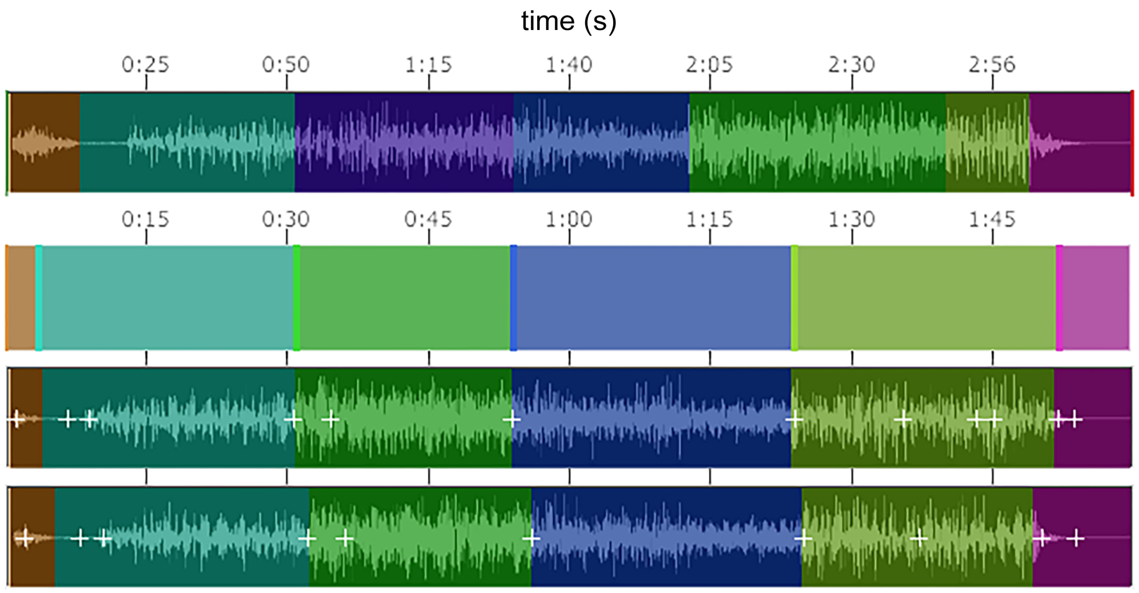


Figure 4.20.: Remixing of the song ”El barzón” from ”Los Amparito” using segmentation enforcement and a target duration of 120 seconds without tolerance. The horizontal axes represent time. From top to bottom: Ground truth segmentation containing seven segments each assigned to a different cluster, target segmentation specifying the cluster to enforce in the result at each point in time, resulting song and segmentation with $t_{\text{low}} = t_{\text{high}} = 0$, resulting song and segmentation with $t_{\text{low}} = 3$ and $t_{\text{high}} = 5$ seconds. White crosses indicate the positions of cuts resulting from bixel jumps in the generated solution.

tion, while the result produced with zero seconds as tolerances, which is functionally equivalent to the method from Wenner [48], fulfils the segmentation constraint almost perfectly. However, the increase in segmentation tolerances also leads to one less jump and therefore possibly to a better sounding final music piece – this effect is evaluated more comprehensively in section 5.1.4, where suitable default values for the tolerance parameters t_{low} and t_{high} will be determined.
In the next section, we propose methods for optimising the cuts caused by assembling the music piece according to the bixel path computed in this section.

4.4. Jump optimisation

The jump optimisation stage is concerned with assembling the final music signal using the bixel path determined by the path optimisation stage in the previous section 4.3. Each bixel jump in the bixel path results in a cut, whose perceived transition quality may be improved by applying signal processing techniques in its vicinity.

One of the issues we aim to resolve in this stage are sound artefacts at the cuts that occur when the beat positions returned by the beat tracker or imported from a file may not be exactly accurate, deviating from the actual beat positions in the song by fractions of a second, as discussed in section 2.4 and shown in audio example 2.6. An approach to prevent such artefacts by correcting the inaccuracy of the detected beat positions will be discussed in the following section 4.4.1.
Another issue often encountered during the evaluation of Wenner's method [48] in section 2.4 is the occurrence of loudness changes at the cut positions. The bixel jumps responsible for these problematic cuts can be avoided using a high weight w_l for the loudness matrix $\mathbf{L}$ described in section 4.2.2, but that also excludes jumps with an otherwise high quality and could thus lead to an overall reduction in output quality. In section 4.4.2, we do not try to avoid these jumps, but instead aim to equalise the loudness difference present at the resulting cuts by modifying the signal itself to facilitate a smooth loudness change.
Finally, the output signal is assembled by concatenating the different sections of the original music piece using crossfading as described in section 4.4.3 around the cut positions to avoid cracking and popping effects and to generally smoothen the transition even further.

4.4.1. Synchronisation

The bixel jumps generated by the path optimisation stage have to be translated from bixel indices to precise time designations describing the exact position in the music signal for the final assembly. Without jump synchronisation, the bixel jumps are converted to time designations using the respective beat positions detected by the beat tracking system: A bixel jump from bixel b_i to bixel b_j is translated to a jump on the signal level from b_{i+1}^{pos} seconds to the destination at b_j^{pos} seconds. This section will present a method designed to synchronise these jump positions with the goal of removing potentially occurring sound artefacts.
According to an evaluation of beat tracking systems [28], the beat tracker used in this thesis [9] features a time resolution of 11.6 milliseconds. Combined with variations in the tracking accuracy, a sound artefact sometimes occurs at a cut position when using the detected beat positions for the assembling the output signal.
Such a sound artefact occurs for example in the song "Lullaby" from "Ghost" contained in the database CC1 in table A.3, of which a short excerpt is given in audio example 4.21 as a reference.
The audio around the cut position resulting from a specific bixel jump is given in

Audio 4.21.: Excerpt of the song "Lullaby" from "Ghost"

audio example 4.22. Notice how near the end a guitar note starts to sound and almost immediately after starts to play again, interrupting the previous guitar note abruptly.

Audio 4.22.: Audio produced by jumping according to the beat positions from 213.74 to 13.4 seconds in "Lullaby" from "Ghost"

This is one exemplary result of the beat positions being so inaccurate that the beginning of the original note is mistakenly included in the output signal before cutting to the target material. Visualising the audio example by plotting the **amplitude**, which describes the sound pressure and is represented as a value in the range $[-1, 1]$ in this thesis, over time in figure 4.23 confirms this to be the issue: The resulting audio is a combination of the audio content at the jump origin in figure 4.23 (a) located before zero on the time axis and the audio content at the jump destination in figure 4.23 (b) beginning at zero on the time axis. Around the jump origin, the start of the new guitar note appears as a sudden peak in the waveform at approximately -0.015 seconds. Consequently, about 0.015 seconds of audio containing the beginning of this guitar sound are included in the result before the jump is performed. After the jump destination however, this guitar note does not sound until about 0.015 seconds after the cut, a fact again revealed by the appearance of the waveform. This creates the undesired hearing impression of two guitar notes in rapid succession which was not present in the original.
Short and loud, non-harmonic sounds like snare drum hits are especially problematic as the final crossfading executed in section 4.4.3 is not able to create the impression of one continuous sound in contrast to many other cases when transitioning between harmonic notes.
To avoid these problems, a jump synchronisation method will be developed in this section to correct the jump times. The correction does not necessarily have to align the used jump times perfectly to the actual beat positions, but only needs to synchronise both music signals in the vicinity of the jump with regards to their timing, that is, the jump origin and destination should have the same position relative to their respective actual beat position.

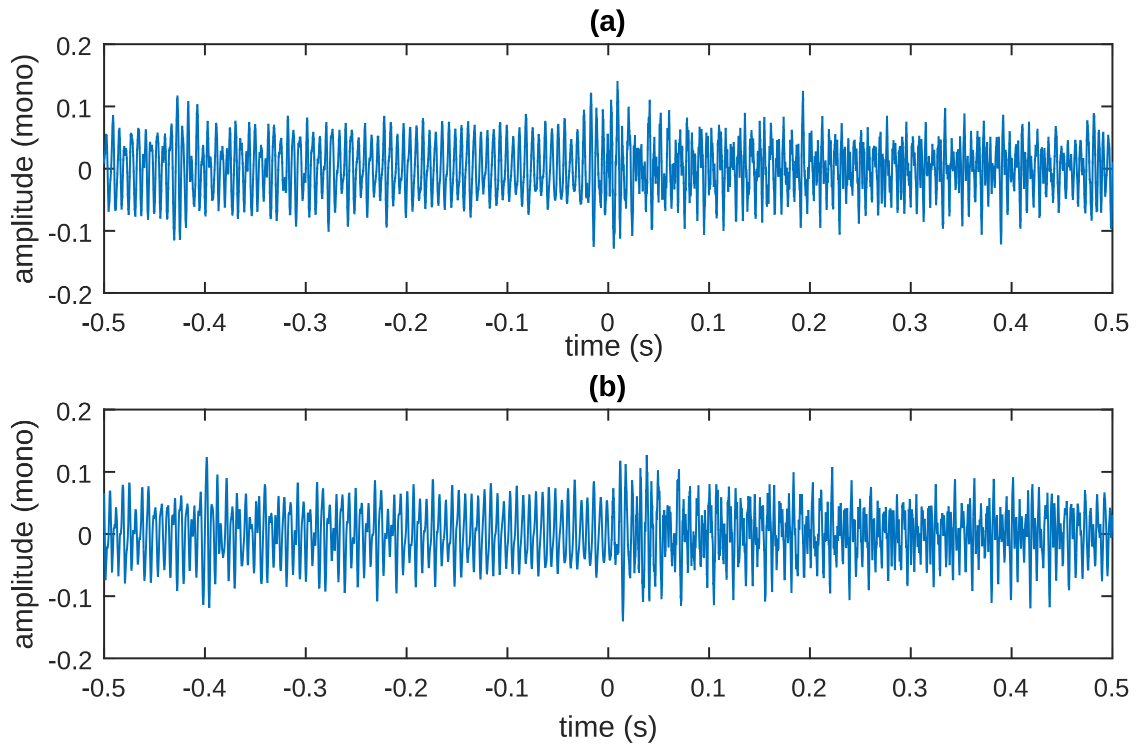


Figure 4.23.: Two waveforms each visualising the amplitudes of one second of audio of the song "Lullaby" from "Ghost" exactly centered on the detected beat positions. Zero on the time axis represents the beat positions at (a) the jump origin at 213.74 seconds and (b) the jump destination at 13.4 seconds.

The delay between a detected beat position b_i^{pos} and the actual beat position in seconds can be modelled as a normally distributed random variable D_i. Positive values indicate a detected beat is located after the corresponding actual beat and vice versa. Unfortunately, the deviation of this random variable cannot be determined empirically by comparing the detected beat positions with the ground truth beat data due to its slight timing inaccuracies caused by the human listener not always perfectly tapping to the beat. However, we can expect an average delay of $\mu_\text{e} = 0$ seconds, because detections located before and after the actual beat should occur equally often and set the standard deviation to $\sigma_\text{e} = 0.1$ seconds. This value was chosen experimentally, as it was just high enough for the synchronisation method to be able to correct every problematic example found during the evaluation of Wenner's method, but not higher to avoid cases where a correction is unnecessary but mistakenly applied regardless. The influence of σ_e on the behaviour of this synchronisation method will be explained in detail later in this section.

Based on these statistical models D_i of the delays, we model the **synchronisation error** for a cut produced by a jump from any beat position b_i^{pos} to any other beat position b_j^{pos} as the sum of both delays at the involved pair of beat positions with a random variable S, where $P(S = x) = P(D_i - D_j = x)$. Positive values can be interpreted as the amount of seconds that are unnecessarily included into the final music piece around the cut position and negative values indicate missing audio material, resulting in a slight perceived "jump" as the next beat starts earlier than expected. Considering that the probability distribution for D_i is axisymmetrical, fulfilling $P(D_i = x) = P(D_i = -x)$, and applying the rule that the sum of two normally distributed random variables is again normally distributed with the total sums of the averages and variances as parameters leads to

$$S \sim \mathcal{N}(0, 2\sigma_{\text{e}}^2). \tag{4.46}$$

For every given jump (t_1, t_2) from t_1 to t_2 seconds, t_{sync} seconds of audio before and after the jump origin and jump destination respectively are extracted from the original music piece. We set $t_{\text{sync}} = 0.5$ seconds, so that even for very slow songs with only 60 beats per minute both audio excerpts cover at least the length equivalent to the distance between two consecutive beats in a measure, enabling a musically meaningful comparison. Assuming a degree of similarity between both signals at the jump position, we detect the relative position of both signals to each other at which the signals align best using *cross-correlation* [45], where values outside the signal's domain are padded with zeros. The result can be represented as a function $r(l)$ of the correlation between the two signals depending on the lag $l \in [-2 \cdot t_{\text{sync}}, 2 \cdot t_{\text{sync}}]$ describing the temporal translation in seconds which was applied to one signal. We use the unbiased definition of cross-correlation [23] to not skew the results towards near-zero lag values due to having more samples available for comparison.

For our example song "Lullaby" from "Ghost", the unbiased cross-correlation for the problematic jump is plotted in figure 4.24 (a). In contrast to the normal cross-correlation, the values do not depend on the amount of lag considered and therefore form a "band" across the whole possible lag range. The fluctuations are a result of the similar frequencies contained in both signals, so that moving one signal relative to the other also produces periodicities when determining their correlation.

Afterwards, the correlation function $r(l)$ is weighted by element-wise multiplication with the synchronisation error model S:

$$r_{\text{w}}(l) = P(S = l) \cdot r(l). \tag{4.47}$$

This step prevents lag values from being selected as reference for synchronisation

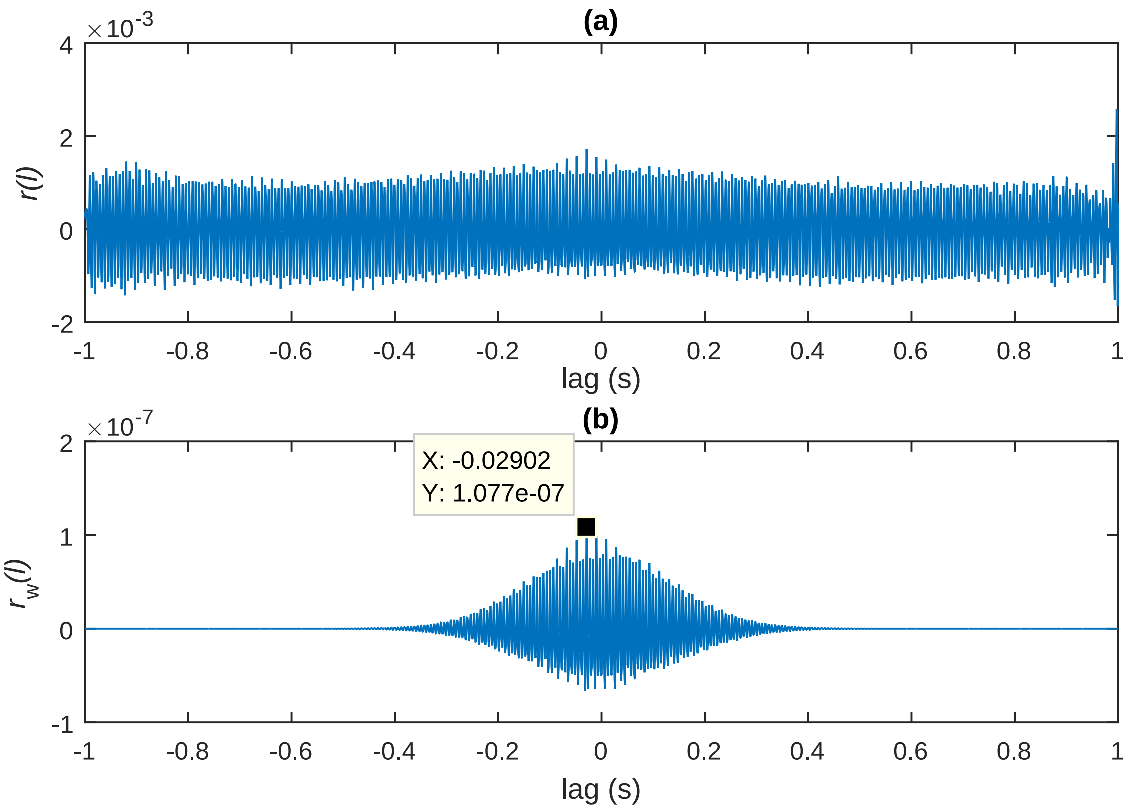


Figure 4.24.: Optimising the jump from 213.74 to 13.4 seconds in "Lullaby" from "Ghost" by calculating the unbiased cross-correlation function $r(l)$ (a) and subsequent weighting resulting in the weighted cross-correlation function $r_{\mathrm{w}}(l)$ with a certain maximum $r_{\max}$ (b). The horizontal axes represent the lag distance in seconds.

which are too high considering the average synchronisation error. For our example, figure 4.24 (b) displays this weighted cross-correlation function $r_{\mathrm{w}}(l)$ and exhibits near-zero values for absolute lags $|l|$ greater than 0.4 seconds.

The position of maximum correlation in the weighted cross-correlation function

$$r_{\max} = \underset{l \in [-2 \cdot t_{\mathrm{sync}}, 2 \cdot t_{\mathrm{sync}}]}{\arg\max} \{r_{\mathrm{w}}(l)\} \tag{4.48}$$

is determined and represents the amount of time that one signal needs to be translated to achieve the maximum signal similarity. This emphasises the importance of the parameter σ_{e}, because large values increase the risk of selecting an extreme value for $r_{\max}$ that actually introduces instead of removes timing errors and consequently sound artefacts, while low values near zero prohibit larger timing corrections to the point where the original jump positions from the path optimisation stage are always left virtually unchanged. In figure 4.24 (b), the point on the function corresponding

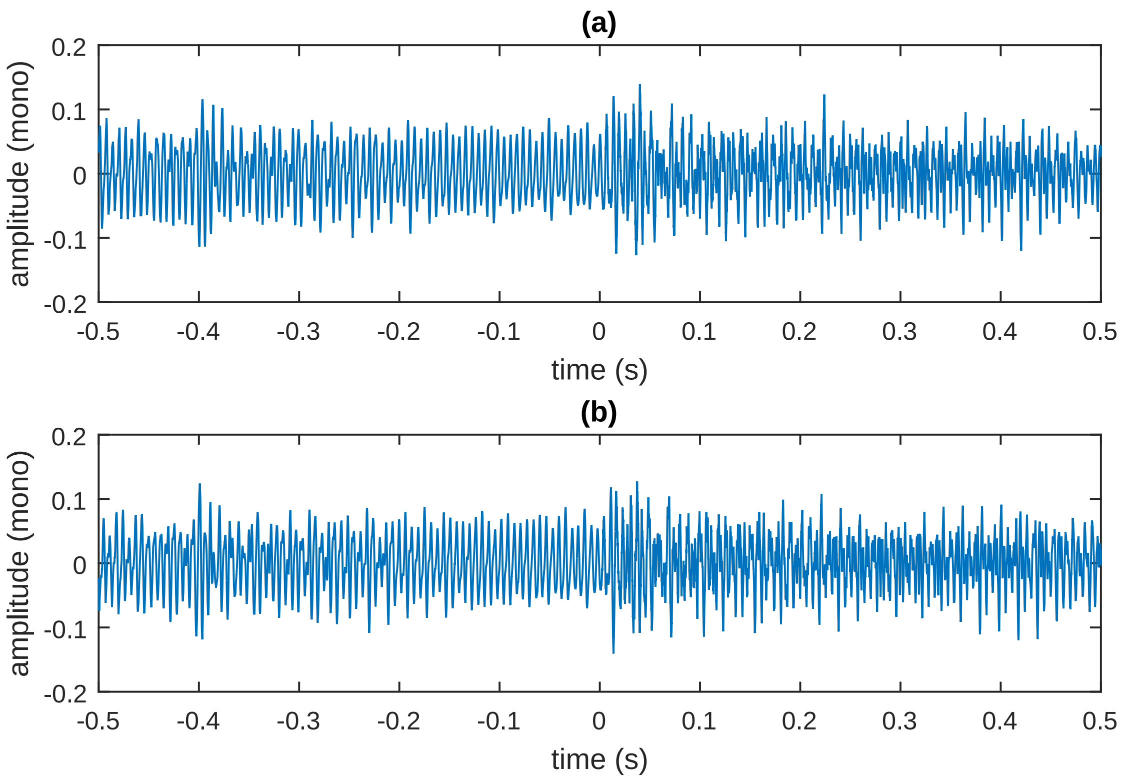


Figure 4.25.: Result of the jump optimisation applied to the jump from 213.74 to 13.4 seconds in "Lullaby" from "Ghost": (a) shows the waveform centered around the corrected jump origin of 213.71 seconds, while (b) displays the same waveform shown in figure 4.23 (b) centered around the jump destination. Horizontal axes represent time relative to the optimised jump positions.

to $r_{\max}$ is marked with a black square and the "X" value indicates the associated lag in seconds.

Taking the case of a negative detected delay $r_{\max} < 0$ as an example, either the jump origin has to be positioned earlier or the destination later in order to correct this kind of timing inaccuracy. Based on insights from music theory, setting the jump origin or destination earlier should theoretically be preferable to setting it later, as notes beginning at the beat positions will be fully included and not interrupted by the cut after already starting to sound.

Consequently, we define the final jump times in seconds as

$$(t_1', t_2') = \begin{cases} (t_1 + r_{\max}, t_2) & \text{if } r_{\max} \leq 0 \\ (t_1, t_2 - r_{\max}) & \text{else.} \end{cases} \tag{4.49}$$

In our example, the delay is negative ($r_{\max} \approx -0.029$ seconds) and thus the jump origin is shifted to an earlier position, thereby leaving out the beginning of the guitar sound so that only the guitar note after the jump destination is included in the result. Figure 4.25 displays the audio waveforms in the vicinity of the synchronised jump. They not only look very similar, but are also very well aligned with regards to the temporal dimension due to the jump synchronisation and lead to an imperceptible cut even without applying the crossfading from section 4.4.3, as demonstrated in audio example 4.26. The guitar note at the fourth beat in the second measure

Audio 4.26.: Audio produced by jumping according to the synchronised jump positions from 213.71 to 13.4 seconds in "Lullaby" from "Ghost".

near the end of the audio example now sounds exactly the same as all the other times when this note is played, in contrast to the sound artefact at the same position in the audio example 4.22.
In the next section, we will present the loudness equalisation method also targeted at the sound quality in the vicinity of cuts, but focusing specifically on sudden changes in loudness.

4.4.2. Loudness equalisation

Complementing the loudness avoidance outlined in section 4.2.2, the loudness equalisation stage as part of the jump optimisation process analyses the audio at the cut position and attempts to smooth out sudden changes in loudness potentially violating the loudness continuity: If a significant loudness change is detected after analysing the loudness function precomputed in section 4.2.2, the quieter signal is amplified and the louder signal is attenuated to provide a smoother transition. This is achieved by multiplying the relevant part of the signal with the appropriate **amplitude change function**, which contains real, non-negative values leading to higher amplitudes for values greater than one and to lower amplitudes for values lower than one. Amplification of the signal is done carefully to avoid causing sound artefacts when amplitudes are outside the supported range and are consequently "cut off" and set to the nearest possible value, a process called *clipping* [10]. This is achieved by subjecting all factors in the amplitude change function to an upper bound depending on the maximum amplitude of the signal so the resulting amplitudes never exceed the valid range.
The maximum duration over which a loudness adjustment can occur is influenced by a parameter t_{lim} and the actual duration is also dependant on the ratio obtained

by dividing the loudness of the quieter signal by the loudness of the louder signal, creating longer transitions for large loudness changes and vice versa.

Relation of loudness and amplitude The method developed in this section estimates the loudness before and after the cut and then constructs a model based on this information, describing how the loudness near the cut has to be changed in order to obtain a slowly changing loudness with an imperceptible loudness difference directly at the cut position. Mathematically, this ”loudness change instruction” is represented as a **loudness change function**, containing values indicating the factor needed for multiplication with the estimated current loudness to achieve the desired loudness.

Because the signal itself describes the amplitude over time, the relationship between loudness and the signal amplitude has to be established to be able to apply a loudness change function to the signal by converting it to an amplitude change function. As discussed in section 2.1, loudness is a complex psychoacoustic phenomenon and therefore modifying the signal amplitudes to precisely achieve a certain loudness is no simple task.

Thus an approximative rule [39] will be used to convert any loudness change indicated by a factor l_c contained in a loudness change function to the corresponding amplitude change factor a_c: At first, the loudness change factor l_c is transferred to the logarithmic *decibel* scale by computing

$$d_\text{c} = 10\ \log_2 l_\text{c}, \tag{4.50}$$

which is then used to approximately calculate the desired amplitude change factor a_c as a relative change in sound pressure:

$$a_\text{c} = 10^{\frac{d_\text{c}}{20}}. \tag{4.51}$$

Applying this factor a_c to the signal by multiplication serves to roughly achieve the desired loudness change indicated by l_c.

Reversing the process allows us to estimate the loudness change factor l_c when applying a given amplitude change factor a_c by first inverting equation (4.51), yielding

$$d_\text{c} = 20\ \log_{10} a_\text{c}, \tag{4.52}$$

and finally inverting equation (4.50) to convert the decibel gain d_c to the loudness change factor

$$l_c = 2^{\frac{d_c}{10}}. \tag{4.53}$$

We will refer to this approximative relationship between amplitude and loudness whenever it is used in the remainder of this section.

Extracting loudness functions around the jump positions The loudness equalisation procedure explained in the following is executed for every jump (t'_1, t'_2) belonging to the generated solution, retrieved from the previous jump synchronisation step in section 4.4.1.

At first, the points on the loudness function $l(t)$ from section 4.2.2 corresponding to the jump times are identified. The two time intervals containing t_{lim} seconds before and after these points in time are identified and the associated parts of the original loudness function used for the subsequent operations. We found $t_{\text{lim}} = 3$ to be a suitable value high enough to give the impression of a smooth loudness change even for jumps with large loudness differences. If there is less audio material available around the jump origin and destination than required by the t_{lim} parameter due to the jumps being too close to the beginning or end of the music piece, the jump is skipped and the next one is processed.

For illustration purposes, we use the song "Amsterdam" by "LASERS" from our database CC1 as an example throughout this section to present intermediate results of our method after applying it to this song as well as the final result. In particular, the jump from 213.06 to 173.9 seconds derived from a specific combination of detected beat positions will serve as an example of a bixel jump producing a large, irritating change in loudness, which can be heard in audio example 4.27. The

Audio 4.27.: Audio produced by jumping from 213.06 to 173.9 seconds in "Amsterdam" from "LASERS".

corresponding loudness functions that were extracted according to the method explained above are visualised in figure 4.28 and reveal large differences measured in sone. On the other hand, both functions fluctuate in a similar manner, exhibiting local maxima and minima at mostly the same positions. This is due to the fact that the outro features a fade-out, indicated by the decline of the green graph, and the jump occurs a few seconds after the beginning of the outro to a position earlier in the song that sounds exactly the same, but louder.

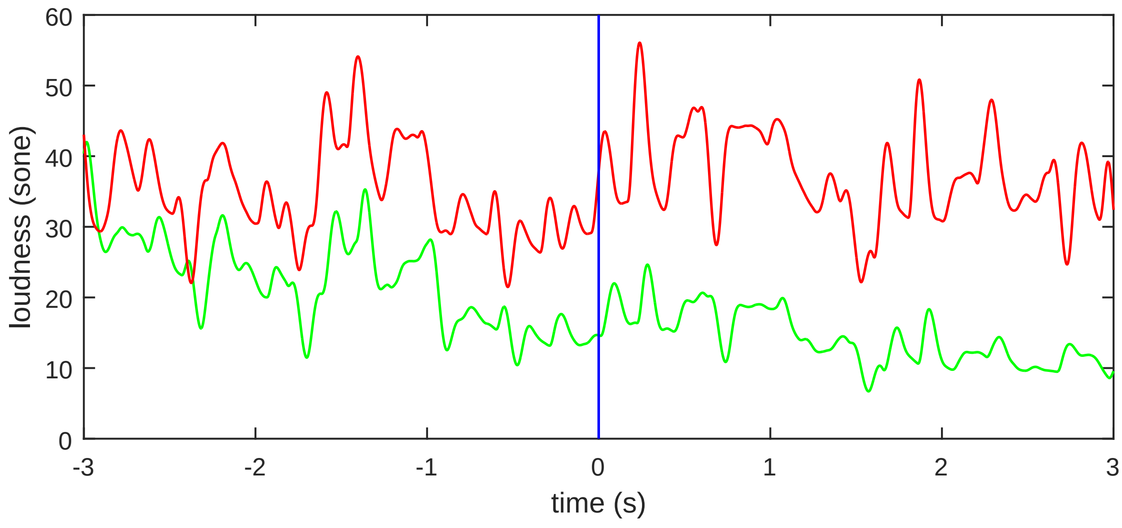


Figure 4.28.: Relevant parts of the loudness function for the song ”Amsterdam” by ”LASERS” extracted around the jump origin at 213.06 seconds (green) and around the jump destination at 173.9 seconds (red). Both functions are aligned along the horizontal axis so that the zero position indicated by the blue vertical line represents the jump origin and destination, respectively.

Comparing loudness functions We will compare the given excerpts of the loudness function around the jump origin and destination in order to detect sudden changes in loudness at the resulting cut by analysing the loudness function before the jump origin and after the jump destination. When aligning the two parts of the loudness function with respect to the jump times as seen in figure 4.28, intersections of both parts are particularly interesting as they represent points in time where the loudness is equal and therefore no loudness equalisation should theoretically be necessary. Additionally, at these moments the loudness function that exhibits greater values than the other one changes, which is an important information used later in the procedure.

Therefore, we detect intersections of the loudness function excerpts and abort the process for the current jump, if an intersection is closer than t_{thresh} seconds to the cut position, because then no significant loudness difference can exist when jumping. The parameter t_{thresh} is set to t_{loud}, which will be introduced and explained later.

If one or more intersections before the cut position exist, the nearest one is used to restrict the considered domain by excluding the region before this intersection, because it contains points in time where the order of the signals in terms of their loudness was reversed. The same is done analogously with the nearest intersection after the cut position, if it exists. When applying this step to the loudness function excerpts depicted in figure 4.28, the intersection at about 2.4 seconds before the cut is detected and consequently used to only take the loudness values after this intersection into account, leading to the restricted time domain visible in figure 4.29. After this step, it is guaranteed that over the remaining time domain, one of the loudness functions always returns values at least as high as the other one. Our example follows this rule and for every point in time in the reduced domain exhibits a loudness near the jump origin that is not higher than the corresponding loudness near the jump destination.

In order to measure how jarring the loudness change is perceived at the cut, we simplify this representation to just an average loudness l_{before} present before and an average loudness l_{after} present after the jump. To estimate these two averages, we compute the mean of the values in the loudness function representing t_{loud} seconds before and after the cut position, because a single point on the loudness function only represents the instantaneous loudness at a specific moment.

Determining the best value for t_{loud} is critical, as it severely affects the results. Averaging over a too large time frame may cause perceptible, but short loudness changes right at the cut position to not be detected and thus not corrected, but a too small value can lead to an excessively extreme loudness adjustment over a long duration for such local loudness differences at the cut position. The underlying question to answer is how many seconds into the past the hearing impression is "integrated" by the human ear when perceiving loudness – a short sound from a

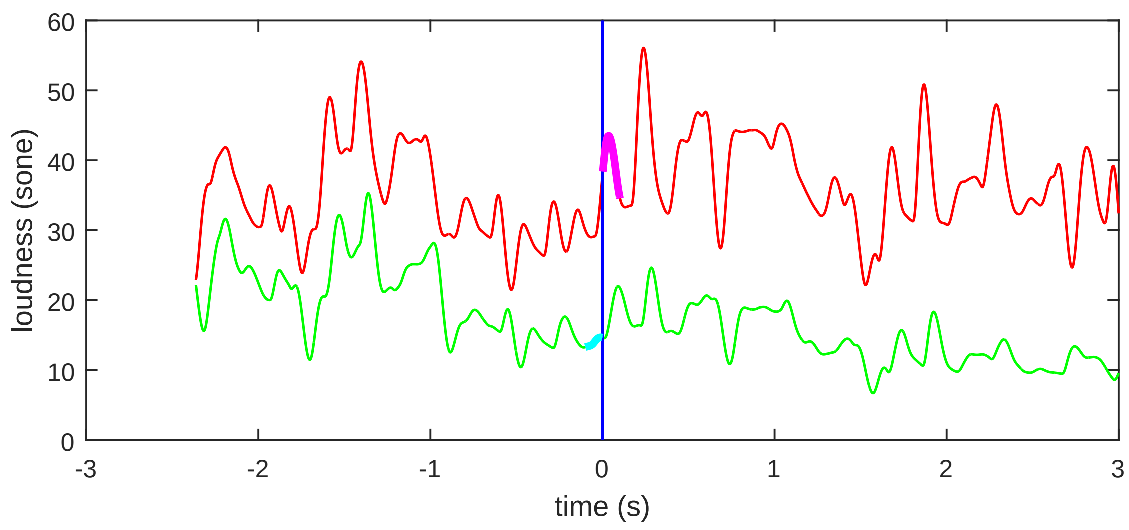


Figure 4.29.: Loudness functions from figure 4.28 after restricting the considered range based on their intersections. The section of the loudness function used for calculating l_{before} is highlighted in cyan and the section used for calculating l_{after} is highlighted in magenta.

snare drum not being perceived as loud as a long flute note in spite of an equal sound pressure level is an example for this integration effect. Chapter four of the "Master Handbook of Acoustics" [13] provides an answer: "100 msec appears to be the integrating time or the time constant of the human ear". Based on this information, we set t_{loud} to 0.1 seconds and therefore set the threshold t_{thresh} for the minimum amount of time available before and after the jump to the same value to ensure the presence of a sufficiently large time frame required for the averaging operation.
In figure 4.29, the parts of the loudness functions used for calculating the average loudnesses are highlighted. This yields $l_{\text{before}} = 14$ and $l_{after} = 40.4$ sone and appropriately describes the loudness change audible in audio example 4.27.
We aim to construct a loudness change function that gradually changes the loudness around the cut position so that the loudnesses before and after the cut ideally meet "half way" at a loudness of

$$l_{\text{mean}} = \frac{l_{\text{after}} + l_{\text{before}}}{2}. \tag{4.54}$$

For our example, we obtain $l_{\text{mean}} = 27.2$ sone as the optimal target loudness to achieve at the cut position.
If the user does not want to correct the whole detected loudness difference, but only a portion $l_{\text{factor}} \in [0, 1]$, the loudnesses l_{before} and l_{after} are adjusted so that only a fraction of the detected loudness difference is actually corrected later:

$$\begin{aligned} l'_{\text{before}} &= l_{\text{factor}} \cdot l_{\text{before}} + (1 - l_{\text{factor}}) \cdot l_{\text{mean}} \qquad \text{and} \\ l'_{\text{after}} &= l_{\text{factor}} \cdot l_{\text{after}} + (1 - l_{\text{factor}}) \cdot l_{\text{mean}}. \end{aligned} \tag{4.55}$$

The remaining steps in the loudness equalisation method will use these adjusted values and interpret them as the actual difference that needs to be corrected. Lower values for l_{factor} consequently lead to lower differences and therefore the loudness is adjusted less. By default and in our example, loudness equalisation is fully activated in our music rearrangement system with $l_{\text{factor}} = 1$, resulting in no adjustment to the detected loudnesses.

Constructing a loudness change function Given the parts of the loudness functions around the cut, we construct two loudness change functions aimed at gradually changing the loudness around the jump origin and destination so that ideally no change in loudness can be noticed at the cut position after assembling the music signal using these modified parts of the original audio. Additionally, the duration over which this gradual loudness change is performed should depend on the relative loudness change from the louder to the quieter signal

$$l_{\text{rel}} = \frac{\min\{l'_{\text{before}}, l'_{\text{after}}\}}{\max\{l'_{\text{before}}, l'_{\text{after}}\}} \tag{4.56}$$

in such a way that the duration of the loudness change in both directions t_{trans} is proportional to the relative loudness change:

$$t_{\text{trans}} = (1 - l_{\text{rel}}) \cdot f_{\text{trans}}. \tag{4.57}$$

The parameter $f_{\text{trans}} > 0$ linearly scales this duration. Additionally, t_{trans} is restricted to be at most half as long as the lowest distance from the cut position to either boundary of the considered time domain, which was potentially reduced due to intersections between the two parts of the loudness function. Through subjective listening tests we found $f_{\text{trans}} = 3$ to be a suitable setting providing loudness transitions smooth enough to not be irritating, but also not unnecessarily long. For example, a sudden doubling in loudness at a cut ($l_{\text{rel}} = 0.5$) is smoothed with a loudness change over a total of $2 \cdot t_{\text{trans}} = 3$ seconds. In the example presented in figure 4.29, we obtain $l_{\text{rel}} = \frac{14}{40.4} \approx 0.35$, meaning the quieter signal has 35% of the loudness of the louder signal at the cut position, and $t_{\text{trans}} = 1.185$, restricted by the distance to the nearest intersection illustrated in figure 4.29 located about 2.4 seconds before the cut position. If the planned length of the transition t_{trans} is lower than $t_{\text{trash}} = 0.1$ seconds, the corresponding jump is skipped and the next jump is processed.

After the duration of the loudness change was determined, the next phase of the loudness equalisation procedure is concerned with the possible amplification of the quieter signal and setting the center of the loudness change accordingly. The loudness change should not necessarily be performed both before and after the jump to an equal degree due to potential clipping when amplifying the quieter signal. Instead, the center position of the intended loudness change is moved to incorporate more of the louder signal and less of the quieter signal, if more attenuation can be applied than amplification. This technique aims to ensure the loudness is changed equally fast over time.

Because the quieter signal has to be amplified, clipping can occur when the resulting amplitude is out of the supported range and is consequently set to the closest supported value. Clipping introduces unpleasant and distortions in the signal and should consequently be avoided. In this thesis, the amplitudes of the signal are represented as real numbers in the range $[-1, 1]$ and for the upcoming calculation, we assume a stereo input music signal that is defined by two functions $a_1(t)$ and $a_2(t)$ returning the amplitude over time for both the left and the right channel, respectively. For any other number of channels, the principle shown below works analogously. To prevent clipping, the possible amplification of the quieter signal at the cut position is limited to a maximum value $a_{\text{max}} \geq 1$ calculated by identifying the highest absolute amplitude in the quieter part that is potentially amplified:

$$a_{\text{max}} = \begin{cases} \min_{t \in [t'_1 - t_{\text{trans}}, t'_1]} \left\{ \frac{1}{|a_1(t)|}, \frac{1}{|a_2(t)|} \right\} & \text{if } l'_{\text{before}} \leq l'_{\text{after}} \\ \min_{t \in [t'_2, t'_2 + t_{\text{trans}}]} \left\{ \frac{1}{|a_1(t)|}, \frac{1}{|a_2(t)|} \right\} & \text{else.} \end{cases} \tag{4.58}$$

This restriction ensures that amplifying the signal does not result in amplitudes outside of the valid range. Note that a_{max} represents an amplitude change factor and not any kind of loudness unit. Consequently, we convert a_{max} to a loudness change factor $l_{\text{max}} \geq 1$ according to the equations (4.52) and (4.53). For the problematic jump in the song "Amsterdam" visualised in figure 4.29, the maximum absolute amplitude is 0.3715 and leads to $a_{\text{max}} \approx 2.69$ and $l_{\text{max}} \approx 1.82$, meaning the loudness of the quieter signal can approximately increased by a factor of 1.82 without causing clipping.

Afterwards, the loudness at which both signals should theoretically converge to at the cut position is calculated, depending on how much amplification is possible:

$$l_{\text{conv}} = \min\{l_{\text{mean}}, \min\{l'_{\text{before}}, l'_{\text{after}}\} \cdot l_{\text{max}}\}. \tag{4.59}$$

Ideally, this "convergent loudness" should be exactly the mean l_{mean} of the loudnesses before and after the jump – this is always the case when the quieter signal can be amplified as much as required without clipping. Otherwise, the loudness equalisation method takes into account the fact that the loudness can not be equally corrected both before and after the jump and adapts by attenuating the louder signal more to make up for the limited amount of possible amplification of the quieter signal. Both loudness change functions will be constructed so that the loudness of the part before the jump and the part after the jump meet at the loudness l_{conv}, which equals to ca. 25.38 sone for our example.

To ensure the loudness is changed more or less equally fast across the whole transition, its position is moved t_{corr} seconds away from the cut position in the direction of the louder signal in case less than the required ideal amount of amplification can be realised, i.e., if $l_{\text{conv}} < l_{\text{mean}}$:

$$t_{\text{corr}} = \frac{l_{\text{mean}} - l_{\text{conv}}}{l_{\text{mean}} - \min\{l'_{\text{before}}, l'_{\text{after}}\}} \cdot t_{\text{trans}}. \tag{4.60}$$

Amplification is not possible as much as ideally required in our example with a jump from a quiet to a loud part (or mathematically formulated, $l_{\text{conv}} \approx 25.38 < l_{\text{mean}} = 27.2$), so the loudness change functions are not centered on the jump times but rather at $t_{\text{corr}} \approx 0.16$ seconds after the cut – because more attenuation is performed than amplification, the attenuation after the cut is distributed over a longer period of time than the amplification before the cut.

As a result, the time domains of the loudness change functions around the jump origin and destination are given by

$$\begin{aligned}
[t_{\text{before}}, t'_1] &= [t'_1 + t'_{corr} - t_{\text{trans}}, t'_1] && \text{and} \\
[t'_2, t_{\text{after}}] &= [t'_2, t'_2 + t'_{corr} + t_{\text{trans}}] && \text{where} \\
t'_{corr} &= \begin{cases} t_{\text{corr}} & \text{if } l'_{\text{before}} \le l'_{\text{after}} \\ -t_{\text{corr}} & \text{else.} \end{cases}
\end{aligned} \tag{4.61}$$

Within these intervals, both loudness change functions are constructed according to the following steps: At the jump origin and destination, the loudness change factor returned by the loudness change function should be $\frac{l_{\text{conv}}}{l'_{\text{before}}}$ and $\frac{l_{\text{conv}}}{l'_{\text{after}}}$ respectively, because it indicates the factor required to obtain the "convergent loudness" l_{conv} at the cut when given the respective estimated loudness before and after the jump. We interpolate between these factors and one (representing no change) according to interpolation functions calculating the area enclosed by both parts of the loudness function around the cut position based on the difference function

$$l_{\text{diff}}(t) = |l(t'_1 + t) - l(t'_2 + t)|, \; t \in [t'_{\text{corr}} - t_{\text{trans}}, t'_{\text{corr}} + t_{\text{trans}}], \tag{4.62}$$

for which $t = 0$ is aligned at the cut position. This difference function is used to define the interpolation functions within the time domains from equation (4.61):

$$\begin{aligned} i_{\text{out}}(t') &= \frac{\int_{t_{\text{before}}}^{t'} l_{\text{diff}}(t - t_1')\,\mathrm{d}t}{\int_{t_{\text{before}}}^{t_1'} l_{\text{diff}}(t - t_1')\,\mathrm{d}t}, \qquad t' \in [t_{\text{before}}, t_1'] \qquad \text{and} \\ i_{\text{in}}(t') &= \frac{\int_{t_2'}^{t_{\text{after}}+t_2'-t'} l_{\text{diff}}(t - t_2')\,\mathrm{d}t}{\int_{t_2'}^{t_{\text{after}}} l_{\text{diff}}(t - t_2')\,\mathrm{d}t}, \qquad t' \in [t_2', t_{\text{after}}]. \end{aligned} \tag{4.63}$$

The functions return values in the range of $[0, 1]$ and describe the weight of the respective loudness change factors $\frac{l_{\text{conv}}}{l'_{\text{before}}}$ and $\frac{l_{\text{conv}}}{l'_{\text{after}}}$ at every point during the loudness change before and after the cut. In contrast to a simple linear function, the loudness change is applied more intensively in regions where the loudness differs more and vice versa. This design should theoretically cause the loudnesses around the cut position to align more closely and produce a smoother transition. We consider a cut from a quiet to a loud part in the following example. If the music directly after the jump is significantly louder than half a second later, the interpolation function i_{in} will stay near one after the cut position for longer than a simple linear interpolation. Consequently, a high attenuation is retained longer, and in exchange returns to zero faster as soon as the music is quieter again and therefore similar to the corresponding audio at the jump origin. For our exemplary jump contained

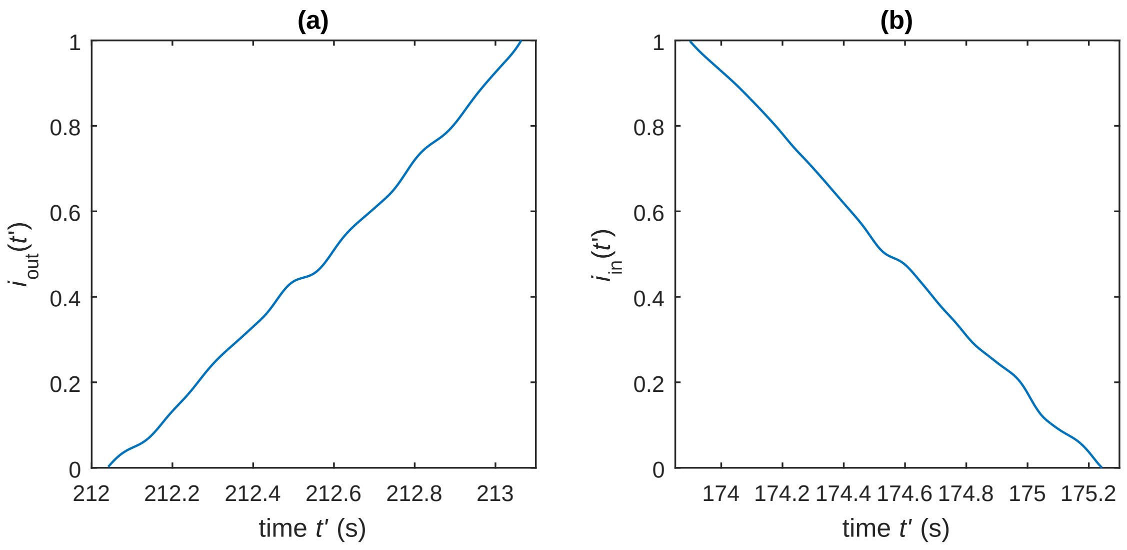


Figure 4.30.: (a) Interpolation function $i_{\text{out}}(t')$ assigning a weight to the loudness change factor $\frac{l_{\text{conv}}}{l'_{\text{before}}}$ for every time point t' within the interval before the jump from equation (4.61) and (b) interpolation function $l_{\text{in}}(t')$ behaving analogously with the loudness change factor $\frac{l_{\text{conv}}}{l'_{\text{after}}}$ for every time point t' within the interval after the jump destination from equation (4.61).

in audio example 4.27, the interpolation function i_{out} designed for the correction before the jump is visualised in figure 4.30 (a) and i_{in} responsible for the loudness equalisation after the jump is plotted in figure 4.30 (b). Small non-linearities are visible and originate from the enclosed area between both parts of the loudness function, leading to fluctuations in the difference function l_{diff} in equation (4.62).

The loudness change functions for the jump origin and destination are defined as

$$\begin{aligned} l_{\text{out}}(t) &= i_{\text{out}}(t) \cdot \frac{l_{\text{conv}}}{l'_{\text{before}}} + (1 - i_{\text{out}}(t)), \quad t \in [t_{\text{before}}, t'_1] \qquad \text{and} \\ l_{\text{in}}(t) &= i_{\text{in}}(t) \cdot \frac{l_{\text{conv}}}{l'_{\text{after}}} + (1 - i_{\text{in}}(t)), \qquad t \in [t_{t'_2}, t_{\text{after}}]. \end{aligned} \tag{4.64}$$

Both functions interpolate between the respective loudness change factor required at the cut position, which either indicates amplification or attenuation, and the value one, which implies not modifying the signal. Their domains are equal to the time intervals computed previously in equation (4.61). Based on the equations above and the interpolation functions visualised in figure 4.30 for our example, figure 4.31 contains the plots of both loudness change functions. According to the function l_{out}, the loudness of the signal right before the jump origin should be gradually increased over about one second, until it reaches a maximum intended increase of approximately 82% (limited by the maximum clipping-free loudness change factor of $l_{\max} \approx 1.82$).

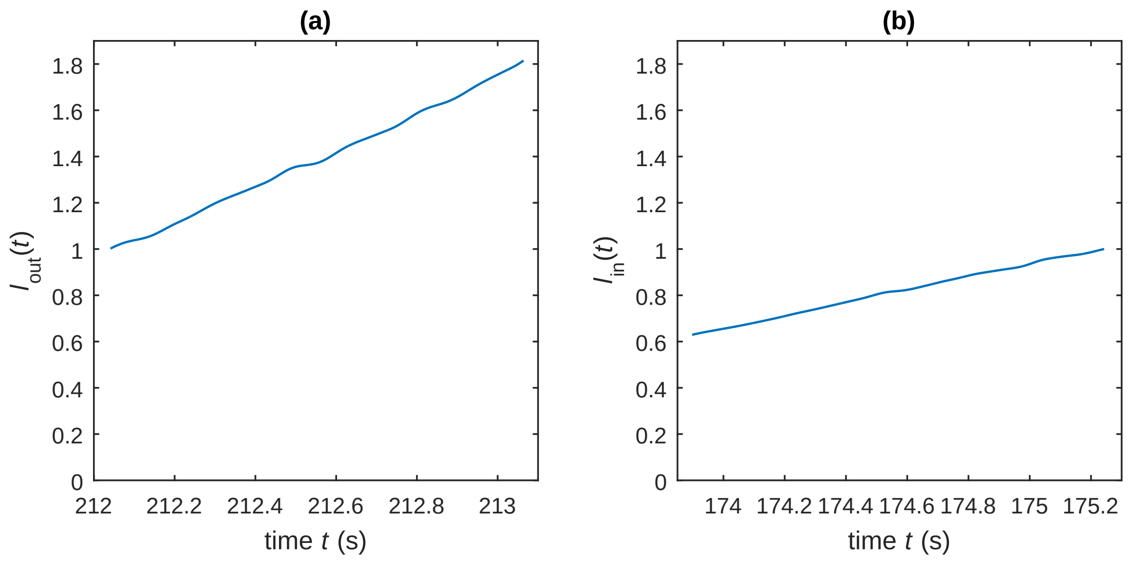


Figure 4.31.: (a) Loudness change function $l_{\text{out}}(t)$ returning loudness change factors for the time before the jump origin and (b) the analogous loudness change function $l_{\text{in}}(t)$ concerning the signal directly after the jump destination.

In contrast, the function l_{in} indicates the necessity of reducing the loudness in the signal directly after the jump destination to about 60% of the original loudness before slowly returning the loudness to original levels.

In the next step, the loudness change functions are extended to also modify the parts of the audio signal that are only audible due to the crossfading employed in section 4.4.3, namely the sections directly after the jump origin and directly before the jump destination. Simply replicating the value present at the border of the respective loudness change function is sufficient for this extension process.

Applying the loudness change function to the signal The loudness change function from equation (4.64) can not yet be applied to the audio signal by multiplication, because the change is given as a relative loudness factor and not an amplitude change factor. Therefore, the loudness change functions are converted to functions approximating the required decibel gain using the formula in 4.50. From these decibel gain functions, the amplitude gain functions can be derived according to the formula 4.51.

Afterwards, both loudness changes are applied by multiplying the corresponding amplitude gain function with all channels of the audio signal at he jump origin and the jump destination, respectively. The two resulting audio snippets are used to assemble the output song by crossfading them in the following section 4.4.3.

Finally, applying the loudness equalisation method from this section in its entirety to the problematic jump in our introductory audio example 4.27 leads to audio example 4.32.

Audio 4.32.: Audio containing the jump from 213.06 to 173.9 seconds in "Amsterdam" from "LASERS" after applying loudness equalisation with $l_{\text{factor}} = 1$ and the default parameters from table A.2.

In comparison, the sudden and irritating loudness change present in the untreated audio example is gone and instead only a slow and gradual loudness increase over a time frame of $2 \cdot t_{\text{trans}} = 2.37$ seconds is present in the audio signal, making the music around this cut position more comfortable to listen to.

To generally increase the quality of the produced cuts even further, the next section deals with crossfading as a means to remove short sound artefacts at the cuts.

4.4.3. Crossfading

When assembling the final song by concatenating different pieces from the original according to the calculated jumps, the amplitudes present before and after a cut originate from different parts of the music signal and therefore can be very different from each other, creating a discontinuity in the produced signal. While the jump optimisation discussed in section 4.4.1 indirectly increases the chance of continuous amplitudes at cut positions due to the use of cross-correlation, the signal can still contain an irregular change in amplitude exactly at the cut position, creating a ”crackling” or ”popping” sound. For suboptimal jumps introducing changes in instrumentation, crossfading is also useful as it smoothly fades between different instruments to make the change less noticeable.
Again we use the song ”Amsterdam” from our database CC1, which already served as an example in the previous section 4.4.2, for demonstration purposes. In audio example 4.33, the effects of such problematic jumps that persist after jump synchronisation can be heard in the form of a short clicking noise at the cut.

Audio 4.33.: Audio produced by the jump from approximately 24.9 to 181.6 seconds in ”Amsterdam” from ”LASERS”.

In figure 4.34, the amplitudes of the audio signal around the jump origin (a) and the jump destination (b) are plotted and visualise the different hearing impressions of both parts with the appearances of their waveforms. The assembly consists of concatenating the signal before the jump origin plotted in blue and the signal after the jump destination plotted in purple (ignoring the loudness equalisation in this case) and produces the waveform in figure 4.34 (c), which corresponds to the audio example 4.33. The amplitudes in the immediate vicinity of the cut position are coloured in red and exhibit a visible ”step” or ”jump” responsible for the clicking noise heard in audio example 4.33.
Crossfading is employed in our music rearrangement system to interpolate between amplitudes in the vicinity of the resulting cut, effectively fading the audio located at the jump origin out while simultaneously fading in the audio at the jump destination. The duration t_{cross} in seconds over which the crossfading is executed is critical, as sounds from both signals are mixed together and can be heard simultaneously: If the duration is too high, undesirable results sounding like two different music pieces played simultaneously can occur while too small durations do not lead to a correction of the sound artefacts mentioned above. We determined a total duration

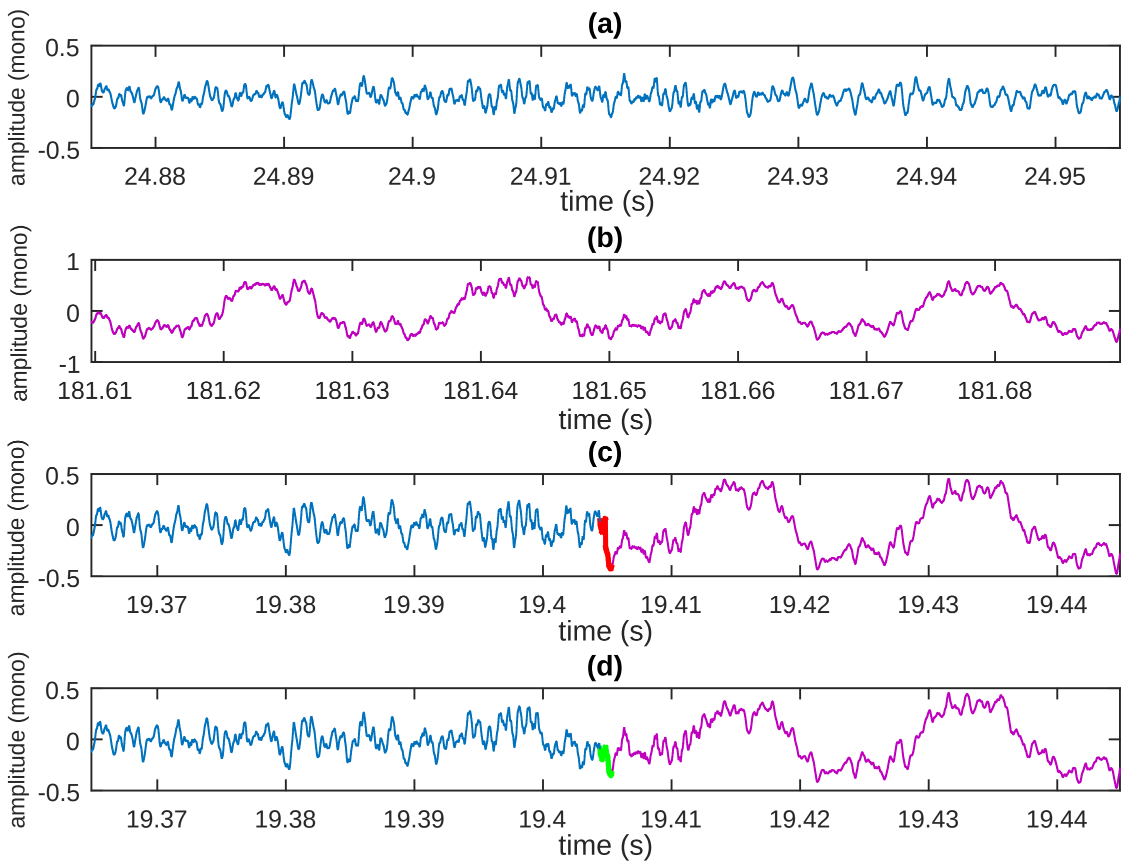


Figure 4.34.: Plots of the amplitudes in the vicinity of a jump from 24.9 to 181.6 seconds in the song ”Amsterdam” from ”LASERS” (converted to mono for illustration purposes). (a) Audio centered on the jump origin, (b) audio centered on the jump destination, (c) audio produced without crossfading and (d) audio produced with crossfading.

of $t_{\text{cross}} = 0.04$ seconds to be suitable, because it was just high enough to correct all sound artefacts and any larger values would often lead to more perceptible cuts due to the listener being able to hear the ”mixing process” as it happens.

Additionally, the weight functions used for interpolation can vary and influence the resulting hearing impression. The simplest type of crossfade is called *constant gain crossfade* [1] and uses two linear interpolation functions containing the required amplitude change factors for fading in and out, where any two corresponding factors responsible for the same point in time always add up to one. This property guarantees the resulting amplitudes to always be in the supported range $[-1, 1]$ and consequently no clipping can occur. On the other hand, this type of crossfade leads to a ”dip” in loudness near its center [1] at ca. $\frac{t_{\text{cross}}}{2} = 0.02$ seconds, which is especially noticeable with longer crossfade durations t_{cross}. *Constant power crossfades* eliminate this effect by using interpolation functions whose value pairs can sum up

to more than one. However, this can lead to clipping problems causing clicking and popping noises for recordings of songs with high absolute amplitudes, which we aimed to prevent in the first place.
Because this section is concerned with solving amplitude continuity problems in the range of a few samples constituting only a few milliseconds of audio, the slight decrease in loudness when using a constant gain crossfade is not audible and employing constant power crossfading would needlessly increase the complexity of the system, additionally forcing us to deal with potential clipping issues.
Consequently, we use a simple constant gain crossfade over a total duration of $t_{\text{cross}} = 0.04$ starting at 0.02 seconds before and ending 0.02 seconds after every cut. We apply this procedure separately to all channels of the original music piece. For our example song, the result of this crossfading stage is demonstrated in audio example 4.35 and does not contain any sound artefacts near the cut. Plotting the

Audio 4.35.: Audio produced by the jump from approximately 24.9 to 181.6 seconds in "Amsterdam" from "LASERS" after crossfading over $t_{\text{cross}} = 0.04$ seconds at the cut position.

amplitudes of the produced signal in figure 4.34 (d) confirms the sudden jump in amplitude highlighted in red in figure 4.34 (c) to be the cause of the sound artefact, because the corresponding amplitudes in figure 4.34 (d), highlighted in green and interpolated by crossfading, do not exhibit this irregularity.
Finally, the assembled output track can be optionally scaled to a specific, exact duration using the time-scale modification explained in the next section.

4.5. Time-scale modification

From the previous section 4.4.3, we obtain a new output track in the form of an audio signal after concatenating and crossfading different parts of the original track. This section is concerned with changing the duration of the whole output track after all other stages of the music rearrangement have been completed so it fulfils the user-specified target duration with high precision.
The restructuring system proposed in this thesis is based on the concept of bixels as the elementary building block used for output generation. As a result, the target duration specified by the user can not be exactly fulfilled when no sufficient tolerance is set, but only approximately achieved with deviations of up to a bixel's length. In some usage scenarios, like creating background music for a film or an advertisement,

more precision is desirable, if not mandatory. Therefore, the user can optionally enable the "accurate mode", which activates the application of a time-scale modification to the output track to scale it exactly to the desired length.

Time-scale modification was already presented as a related approach in section 2.2 and constitutes an optional addition to the end of the system's processing pipeline. The input to this stage consists of the generated music signal of length t_{res} along with the target duration t_{target} given by the user.

A positive *scaling factor* needed for a time-scale modification algorithm is calculated, representing the relative change in duration to achieve the target duration:

$$f_{\text{scale}} = \frac{t_{\text{target}}}{t_{\text{res}}}. \tag{4.65}$$

Values greater than one indicate the song has to be lengthened, while values less than one imply the song has to be shortened to fulfil the duration requirement.

A plethora of algorithms for time-scale modification exist, of which the authors in [12] give an overview and also implement some of the well-known ones in their "TSM Toolbox" for MATLAB that we also employ in our music rearrangement system. By default, the *WSOLA* algorithm [43] is executed with the scaling factor from equation (4.65) as input if the user requires the song to fulfil the target duration with high precision. Although there is currently no option in the user interface of the music rearrangement system to select between different algorithms, as time-scale modification is not the focus of this thesis but only a small addition, this could be easily added if different algorithms are desired for different songs.

The upcoming section presents the user interface of the developed music rearrangement system.

4.6. User interface

The user interface of our music rearrangement application has to provide access to all features presented in the previous sections and also needs to be able to visualise and play the input and output songs. Therefore we decided to use the cross-platform C++ framework "JUCE" [35] for the implementation of the user interface, because it provides many functions related to handling audio input and output needed in the context of a music rearrangement system. Additionally, the "dRowAudio" module for JUCE [37] was utilised, as it allows to display waveforms for audio signals and interact with them through an audio player.

We will outline a typical workflow with our application in the remainder of this section and present the involved interface elements.
At first, the user selects the input track and then decides between importing external beat data from a file or using the automatic beat tracker from section 4.1 to detect the beat positions. Afterwards, the preprocessing stage is executed to generate the required loudness function, the transition cost matrices and the first automatic segmentation according to section 4.2. Upon completion, the application enters its first stage, where the ground truth segmentation can be edited.

Original segmentation stage A screenshot of the user interface in this stage with the song "Amsterdam" from "LASERS" opened can be found in figure 4.36. On the upper left hand side, the permanently available standard controls are shown. A track can be loaded and a produced music piece can be exported to an audio file. The current settings can be saved in the form of a project file, containing all necessary information to describe the current state of the application. Saved project files can

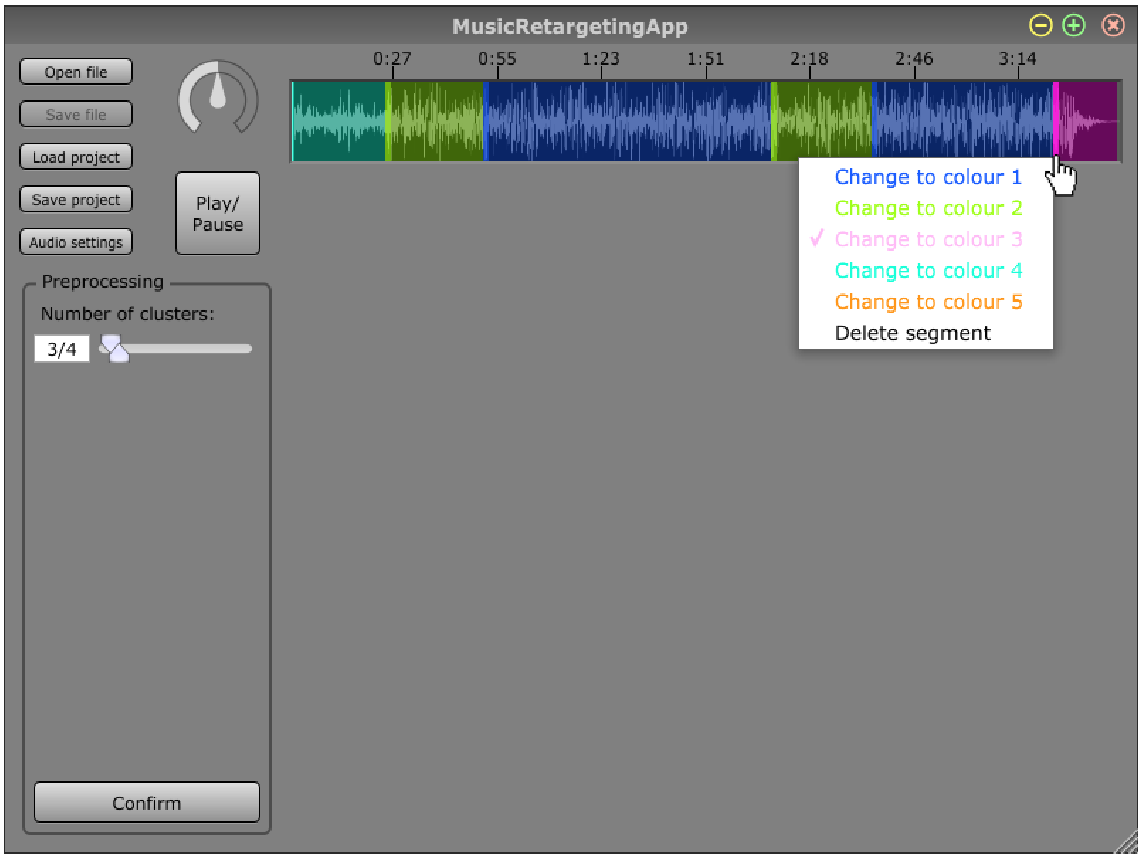


Figure 4.36.: Screenshot of the user interface for the music rearrangement system in the segmentation stage, allowing the user to edit the ground truth segmentation.

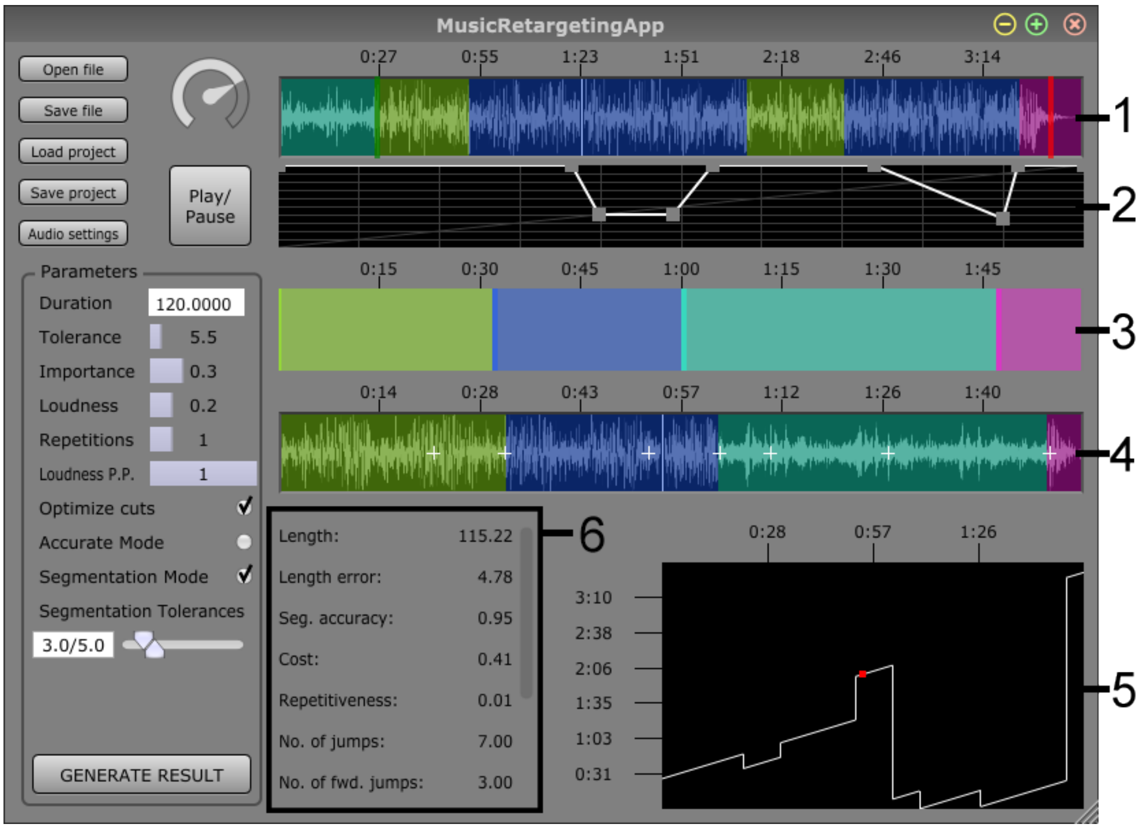
be loaded at a later point in time to continue working on a track without losing any progress. In order to play back audio, it can be necessary on some computers to configure the audio devices in the audio settings menu, which opens after clicking the "Audio settings" button. The large "Play/Pause" button starts and pauses the current audio playback, while the slider above controls the volume.

During the original segmentation stage, the user can manipulate the slider with two knobs in the area titled "Preprocessing" to change the possible range C_{range} for the number of clusters C introduced in section 4.2.3. Upon releasing the mouse after dragging the slider, a new segmentation for the original song is calculated immediately and then displayed on the upper right hand side: The original audio is illustrated by a waveform and each segment is highlighted using colours that each represent their cluster membership. Should a correction of this automatic segmentation be necessary, the user can drag the segment transitions to change the location of the segments, or alternatively right click the transition for a context menu, which is opened for the last segment in the screenshot as an example. The user is then able to assign the selected segment to a different cluster (represented by the colour) and also delete the segment completely – in this case, the previous segment is automatically extended to cover the time interval of the removed segment. After clicking the "Confirm" button in the lower left corner, the current segmentation is confirmed as the ground truth segmentation s_{ground} and considered as constant for the following rearrangement stage.

Rearrangement stage After completing the original segmentation stage, the user can edit all constraints in real-time and generate a new solution based on these constraints with the path optimisation algorithm from section 4.3, as illustrated in figure 4.37. The constraints and additional parameters for jump optimisation and time-scale modification are listed on the left hand side in the area titled "Parameters". From top to bottom, the first settings define the t_{range} with the target duration t_{target} and the duration tolerance t_{tol} in seconds.

Next, the weights for the importance function $I(i)$ and the loudness matrix $\mathbf{L}$ can be set. If their sum is below one, the difference to one is used as the weight w_{d} for the transition cost matrix $\mathbf{D}'$ regarding timbre. Otherwise, both values are summed and normalised to one before they are used as weights w_{p} for the importance function and w_{l} for the loudness matrix, while the weight w_{d} is set to zero. This method enables the user to intuitively set three weights with just two sliders.

Continuing with the list of parameters, the repetition avoidance parameter w_{r} is



Parameters

Duration	120.0000
Tolerance	5.5
Importance	0.3
Loudness	0.2
Repetitions	1
Loudness P.P.	1
Optimize cuts	✓
Accurate Mode	●
Segmentation Mode	✓
Segmentation Tolerances	3.0/5.0

Length:	115.22
Length error:	4.78
Seg. accuracy:	0.95
Cost:	0.41
Repetitiveness:	0.01
No. of jumps:	7.00
No. of fwd. jumps:	3.00

Figure 4.37.: Screenshot of the user interface for the music rearrangement system in the rearrangement stage, enabling the user to generate a new music piece from the original piece based on the different constraints.

followed by the intensity factor l_{factor} of the loudness equalisation method. All sliders mentioned above support the direct input of exact values after clicking on them in addition to the expected dragging functionality. The jump synchronisation, the time-scale modification to fulfil the target duration with high precision and the segmentation enforcement can each be toggled on or off with radio buttons below. Finally, the segmentation tolerances t_{low} and t_{high} can be set at the bottom.

For the main screen space, we will again present each interface element from top to bottom and refer to them using the numbers on the right hand side. The original audio with the fixed ground truth segmentation is displayed at the top (1) as well as a green and a red bar that can be dragged to the desired start and end position t_{start} and t_{end}. Directly below, points in the black area (2) can be created and moved around to model the importance curve that uses the same time axis as the display of the original audio and is translated into the bixel-wise importance function $I(i)$ for the path optimisation stage. The higher the importance curve is located at a specific point in time, the less penalty is induced by including the corresponding bixel in

the solution. In our example, the audio between approximately the one minute and 25 seconds mark and one minute and 50 seconds is less likely to be included in the final result. The same principle applies to the section assigned to the blue cluster near the end of the music piece. Under the importance curve, we placed the display of the editable target segmentation $s_{\text{target}}(t)$ (3), where new segments can be freely created and existing ones moved, modified and deleted similar to the ground truth segmentation in the previous original segmentation stage, except for being restricted to the already defined set of clusters. Its timeline always covers the time interval between zero and the current target duration and updates whenever the target duration is changed. The fourth horizontally aligned display (4) depicts the waveform of the generated music piece along with the resulting segmentation. Playing back the result starting at any point in time only requires a click at the corresponding position in the display followed by the activation of the "Play/Pause" button. Additionally, white crosses on the waveform indicate positions of cuts to make them easy to locate and play back.
Notice that the active segmentation enforcement with tolerances causes the target segmentation above (3) to roughly match the result segmentation (4), visible by the matching pattern of colours in both displays.
The visualisation in the lower right (5) contains a white graph describing the path that the result takes through the original song: The horizontal axis corresponds to the time axis of the result and the vertical axis describes the position in the original song, causing unaltered playback of original pieces to form a diagonal line and jumps to manifest itself as discontinuities in the plot in the form of vertical lines. When playing back the result, the red dot continually moves along this graph to the right on the horizontal axis and indicates the current positions in the result and in the original music piece.
Finally, a list of metrics to evaluate the output quality is displayed at the bottom (6), of which some will be introduced and used in the following evaluation section 5.
Users can adjust the parameters to their liking and generate a new result at any time by clicking on the button in the lower left corner.
This concludes the presentation of the proposed music rearrangement system. In the following section, the system will be evaluated concerning a range of different aspects.

5. Evaluation

In this section, we will evaluate the music rearrangement system proposed in the previous section 4 regarding a range of different aspects.
The evaluation consists of an automatic evaluation as well as a listening study. In the automatic evaluation presented in section 5.1, several components of the system are tested using automatic procedures. Because running these tests does not require user interaction and can therefore efficiently cover a wide range of test cases, an effort was made to evaluate as many features of the system as possible with this method.
On the other hand, one can argue that the main criterion for evaluating a music rearrangement system is the perceived sound quality of the produced music pieces. Due to the jump-based approach, the output tracks consist of unmodified sections of the original tracks except in the vicinity of the cut positions, where sometimes the audio signal is changed by the jump optimisation stage from section 4.4. As a result, the sound quality of the cuts represents the critical factor for determining the overall output quality. Consequently, we conducted a listening study that presented a series of music excerpts containing cuts generated by the system and asked the participants to rate their quality according to different aspects. The results of this listening study are discussed in section 5.2.
At first, we will begin with describing the automatic evaluation.

5.1. Automatic evaluation

Many components of our music rearrangement system are suited for an individual automatic evaluation, because appropriate metrics can be defined to evaluate the quality of their generated results.
One of these automatically evaluated components is the estimation of the optimal path's length from section 4.3.2: In section 5.1.1, we automatically determine if a better solution outside of the estimated k_{range} exists by computing the solutions associated to all theoretically possible values of k. If that is the case, our estimation

method failed to find the optimal solution, otherwise it succeeded. For a large number of test cases, we compute the percentage of successes and use it to evaluate the estimation algorithm, as this percentage should be similar to the percentage indicated by the given parameter p_{min}.

The multiple goal A* algorithm presented in section 4.3.5 is evaluated with and without employing heuristics in section 5.1.2 regarding its average runtime and how it depends on different parameters. Additionally, we compare it to our implementation of the dynamic programming approach from Wenner [48].

Repetition avoidance was introduced as a user constraint in section 4.3.4, designed to avoid the repetition of short segments in the output track. We will propose a metric to estimate the level of annoyance that a generated solution would evoke in a listener to allow for the automatic evaluation in section 5.1.3.

Finally, the segmentation enforcement with tolerances presented in section 4.3.6 will be automatically evaluated in section 5.1.4. As part of the evaluation, the parameter space defined by the two tolerance parameters t_{low} and t_{high} is explored using a range of different settings between zero and ten seconds and their effects on the generated solution are discussed.

Database and setup For the automatic evaluation procedures, a set of test cases, where each case specifies certain inputs to the music rearrangement system, needs to be defined and subsequently executed. These inputs consist of a music piece as well as a set of user constraints. Because the automatic evaluation can process a large number of test cases in little time, a large set of test cases was created in an effort to cover a wide range of application scenarios by selecting music from many different genres as well as many different settings for the user constraints.

We extended the music database CC1 listed in table A.3 by including particularly long tracks in order to measure the runtimes of the multiple goal A* algorithm in section 5.1.2 more extensively, resulting in the database CC2 with 59 tracks listed in table A.4. The additional music pieces have an average duration of 496 seconds, while the average length of all music pieces in the CC1 database is about 215 seconds. Again, the tracks from this database and their genres are sourced from the Free Music Archive [49] and belong to different musical styles.

The settings for the user constraints cover a range of target durations $t_{\text{target}} \in \{60, 120, \dots, 1200\}$ without any additional tolerance ($t_{\text{tol}} = 0$), where all values represent seconds. We did not employ tolerance to avoid cases in which the original track does not have to be modified at all, because its duration already fulfils the desired target duration t_{range}. For each of those 20 desired durations, we create one test case that features segmentation enforcement and one test that does not. When enforcing a segmentation, the ground truth segmentation used is the result of the automatic segmentation with the default range of clusters $C_{\text{range}} = [3, 5]$ and the target segmentation is equivalent to the ground truth segmentation, but scaled on the time-axis to the respective target duration t_{target}. The remaining user constraints are set to their default values.

All in all, the 59 tracks in the database CC2 in combination with the 20 target durations and the activation or deactivation of segmentation enforcement yield a total of $59 \cdot 20 \cdot 2 = 2360$ test cases covering a wide range of application scenarios. We will refer to this set of test cases as the **standard test set** in the following sections, beginning with the automatic evaluation of the path length estimation.

5.1.1. Estimating the path length a priori

In section 4.3.2, a method to estimate the path length before executing the path optimisation procedure was developed that determines the range k_{range} in which the number of bixels of an optimal bixel path is likely to fall into. Setting the parameter $p_{\min} \in [0, 1)$ to a specific probability should guarantee the probability of finding an optimal solution within the resulting k_{range} to be at least as high as $p_{\min}$. This guarantee will be put to the test in this section.

For every test case in the standard test set, we first determine the optimal bixel path by setting $k_{\text{range}} = [2, k'_{\text{limit}} + 2]$ and thus taking every possible number of bixels into account. Afterwards, we employ the path length estimation multiple times using 0.5, 0.6, 0.7, 0.8, 0.9, 0.95 and 0.99 as probabilities for $p_{\min}$ (with the intention of exploring the most promising probabilities near one more extensively and leaving out the lower range) and determine the best solution within the respective resulting range k_{range}. If this solution has a higher associated cost than the optimal solution, it is suboptimal and we represent this failure with a value of 0. Otherwise, we assign the value 1 to indicate a success. Averaging these values over all experiments conducted with the same specific value of $p_{\min}$ results in an approximation of the actual probability of finding the optimal solution, which we call **empirical success probability**.

The results for our standard test set are shown in figure 5.1. In particular, figure 5.1 (a) visualises the empirical success probability for a given setting of $p_{\min}$ and demonstrates that the previously mentioned guarantee is satisfied by our system: The empirical success probability is always as least as high as the probability defined

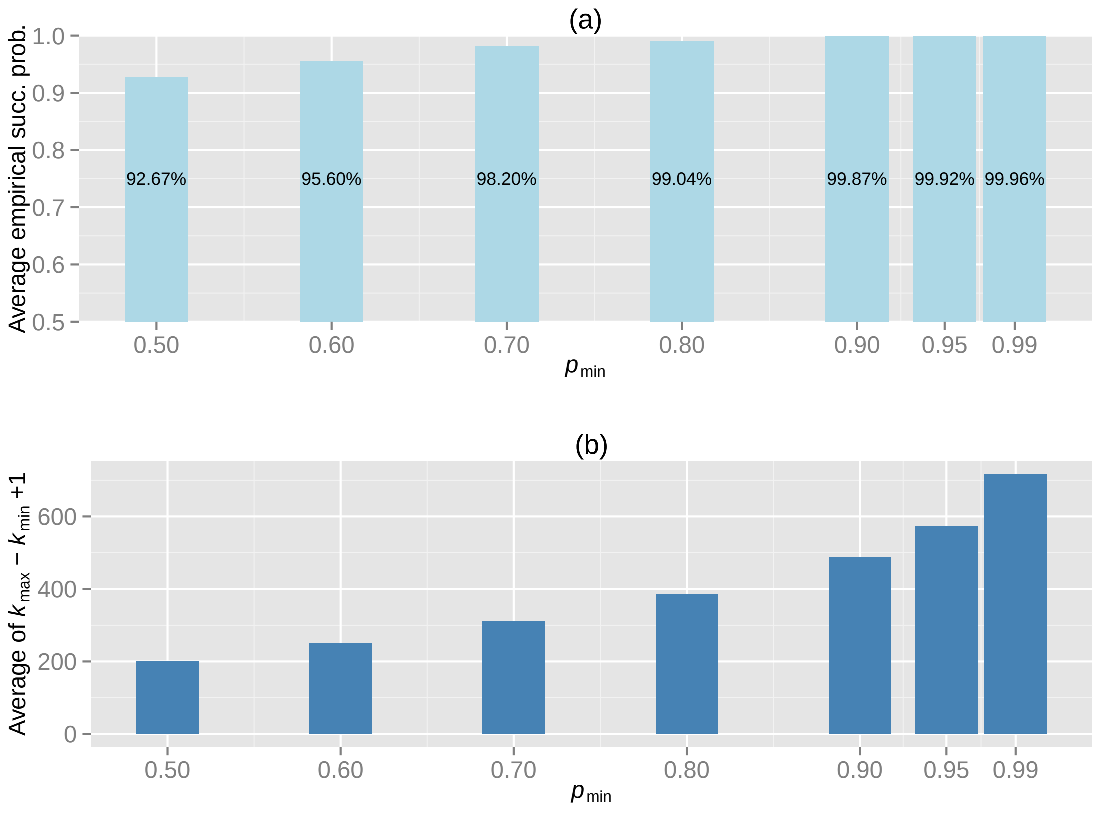


Figure 5.1.: Averages of (a) the empirical success probabilities and (b) the mean of $k_{\text{max}} - k_{\text{min}} + 1$ representing the number of goal nodes $|\mathcal{G}_{\text{end}}|$ plotted for a range of values for p_{min}.

by the parameter p_{min}. One would ideally expect the empirical success probabilities to be only marginally larger than this minimum threshold, if the statistical models of the lengths of bixels B_n in equation (4.22) and bixel paths $L_{k'}$ in equation (4.23) are correct, because algorithm 4.1 provably computes a k_{range} that is as narrow as possible while still fulfilling the minimum probability p_{min}. This approximate equality between the empirical success probability and the probability described by the parameter p_{min} would also allow for a more intuitive selection of a suitable value for p_{min}, as such a value would then resemble the actual chance of finding the optimal solution. However, figure 5.1 (a) clearly depicts very high success probabilities even for low settings of p_{min}. A reason for this could be that modelling the bixel length B_n as a normal distribution does not provide a very accurate description of the actual distribution of bixel lengths: Often, the first and last bixel of a track are significantly longer or shorter, caused by not detecting any beats at the beginning or at the end, for example due to silence. In such a case, the deviation μ_{b} of the normal distribution is significantly higher than for example in figure 4.10 merely on the basis of these two bixels. Consequently, the model implies that *all* bixels deviate from the mean much more than they do in reality. The error in this inaccurate model propagates through every subsequent calculation in the proposed algorithm, until the final step returns a k_{range} that is unnecessarily "wide" and covers the optimal solution with a much higher probability than estimated. Future work could include constructing a model that represents the distribution of beat lengths more accurately and integrating it into our own method.

The estimated range of possible lengths for the optimal bixel path k_{range} is important for the path optimisation process: Calculating $k_{\max} - k_{\min} + 1$ yields the number of goal nodes, that is, the cardinality $|\mathcal{G}_{\text{end}}|$ of the set of goal nodes $\mathcal{G}_{\text{end}}$ to which the shortest path must be found. As we will show in section 5.1.2, more goal nodes lead to higher average computation times for the path optimisation algorithm. In figure 5.1 (b), this number of goal nodes $|\mathcal{G}_{\text{end}}| = k_{\max} - k_{\min} + 1$ is plotted for the various values of $p_{\min}$ and increases more rapidly as $p_{\min}$ approaches one.

In conclusion, setting $p_{\min}$ to some value always implies a trade-off: As the probability of finding the optimal solution increases for larger values of $p_{\min}$, the number of goal nodes and consequently the average runtime of the path optimisation procedure also increases, as shown in the following section. We set a default value of $p_{\min} = 0.6$ for our music rearrangement system, as it offers an empirical success probability of over 95% despite the relatively low number of associated goal nodes $|\mathcal{G}_{\text{end}}|$.

5.1.2. Path optimisation algorithms

This section is concerned with evaluating the runtime of the path optimisation algorithms discussed in section 4.3.5. We compare the original approach using dynamic programming (DP) from Wenner [48] and the proposed multiple goal A* algorithm based on the formulation of the optimisation problem as a graph problem. The multiple goal A* algorithm was tested in two different versions. One version (A*) does not use a heuristic (or in other words, it uses a heuristic that always returns zero) and the other version (A*-H) employs the problem-specific heuristic $\hat{h}'$ from equation (4.43) also developed in that section.

All experiments were performed on a computer equipped with an "Intel i7" quad-core processor with a frequency of 3.4 GHz and 16 GB of RAM. For every test case in the standard test set, the optimal bixel path was determined by executing the three different implementations of path optimisation mentioned above while measuring their wall-clock runtimes.

Analysing the recorded runtimes leads to the conclusion that the multiple goal A* algorithm presents a successful improvement of the dynamic programming approach in general: As an approximate average over all test cases, the multiple goal A* algorithm without heuristics requires a computation time of 2.63 seconds, while the dynamic programming approach needs 4.68 seconds and therefore significantly more time. The use of heuristics for the multiple goal A* algorithm slightly speeds up the calculation and leads to an average runtime of 2.48 seconds. Although this result could possibly be improved even further using more powerful heuristics, it proves the usefulness of the specific heuristics proposed in section 4.3.5 of this thesis. We also recorded the amount of calculated transitions for every test case and found this number to be proportional to the required runtime of both A* algorithm variants, exhibiting a linear relationship with a very high, significant correlation of over 0.98. On average, the heuristics reduced the amount of considered bixel transitions by about 9%. Combining both of the aforementioned results, this reduction in the amount of considered bixel transitions explains the reduction in average runtime when enabling heuristics of ca. $1 - \frac{2.48}{2.63} \approx 0.057 \equiv 5.7\%$. This percentage describing the measured speed-up is slightly lower than the theoretically possible decrease in runtime by 9%, as it is counteracted to a certain degree by the overhead induced by the additional time required for computing the heuristic function $\hat{h}'$.

Referring to the estimation of k_{range} in the previous section 5.1.1, figure 5.2 illustrates the relationship between the minimum success probability $p_{\min}$ and the average runtime of all test cases with a certain setting for $p_{\min}$. Because the set of goal nodes grows in size with increasing values for $p_{\min}$, as shown in figure 5.1, all three tested algorithms tend to require increasingly more time. However, independent of $p_{\min}$, there is always a large difference between the dynamic programming approach coloured in red and both variants of the multiple goal A* algorithm. Additionally, enabling the heuristics consistently leads to a slightly lower computation time.

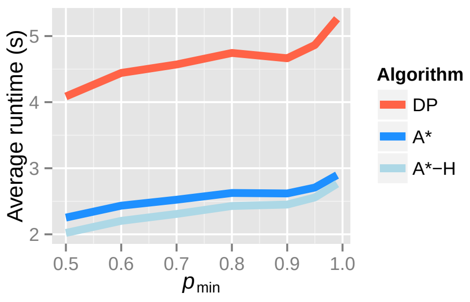


Figure 5.2.: Average runtimes in seconds for the three path optimisation algorithms presented in section 4.3.5, for different minimum success probabilities $p_{\min}$.

The runtime is not only affected by the minimum success probability $p_{\min}$, but also by the number of bixels N in the original track and the maximum number of bixels $k_{\max}$ that is considered as a length of a possible bixel path. N and $k_{\max}$

are both approximately proportional to the duration of the original track and the maximum target duration t_{max}, respectively. Variations in these variables for the same number of bixels occur due to bixels with different lengths. Two songs of identical length for example have a different amount of bixels N, if the beat tracker detects a higher tempo and therefore more beats in one song than in the other. In addition, we examined whether segmentation enforcement has a significant influence on the average runtimes.

In the following, we analyse the runtime as a function of the length of the original piece as well as the target duration t_{target} both with and without using segmentation enforcement. Note that the resulting range of target durations t_{range} is equal to $[t_{\text{target}}, t_{\text{target}}]$ in this evaluation, because we disabled the tolerance setting with $t_{\text{tol}} = 0$, leading to $t_{\text{max}} = t_{\text{target}}$ and $t_{\text{min}} = t_{\text{target}}$. Due to the variances in performance for different tracks making the resulting plots of the runtime difficult to interpret, we smoothly interpolated each function using *locally weighted regression* with the function *loess* provided by R [34]. The resulting curves are shown in figure 5.3 and describe these interpolated, average runtimes. As mentioned in section 4.3.5, all three path optimisation procedures have the same asymptotic complexity of $O(N^2\ k_{\text{max}})$. This asymptotic complexity manifests itself in the behaviour of these plotted functions: For various durations of the input track in figures 5.3 (a) and (b), the runtimes of all three algorithms increase approximately quadratically with the length of the original track, caused by the N^2 component of the complexity. This was confirmed after fitting a quadratic function $y = ax^2 + c$ to the three series of smoothed data points, because every fitted function exhibited a *coefficient of determination* of over $R^2 = 98.9\%$, indicating it explains almost all of the variance in the set of data points. Apart from almost equal runtimes for tracks shorter than two minutes, both variants of the A* algorithm quickly set themselves apart from the dynamic programming approach as the duration of the input tracks increases. Overall, using the multiple goal A* algorithm with heuristics appears to be the best of the three available solutions. Input tracks lasting 300 seconds or less are the exception, where all algorithms finished in a fraction of a second, but sometimes the dynamic programming approach was marginally faster than the variants of the multiple goal A* algorithm. This is due to less overhead when setting up the data structures holding the open and the closed set (like the Fibonacci heap) in comparison to the simple two-dimensional array needed to store the $C(i, k)$ values of the dynamic programming approach.

As the target duration t_{target} is proportional to the maximum considered length k_{max}

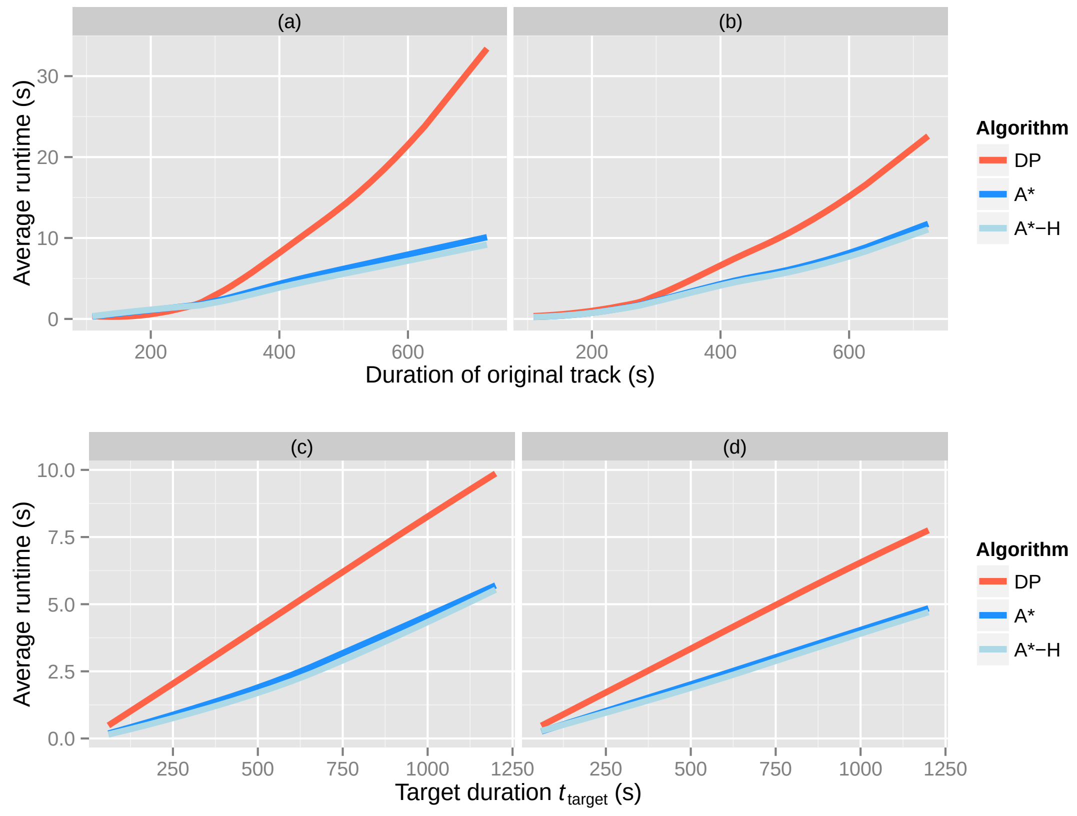


Figure 5.3.: Average runtimes in seconds for the three path optimisation algorithms presented in section 4.3.5 as a function of ((a) and (b)) the duration of the original track and ((c) and (d)) the maximum target duration t_{max}. Segmentation enforcement is disabled for the plots in (a) and (c) and enabled for the plots in (b) and (d).

for a bixel path and the asymptotic complexity is $O(N^2\, k_{\text{max}})$, one would expect the runtimes to increase linearly with the target duration t_{target}. Figures 5.3 (c) and (d) depict the average runtime of the algorithms for different target durations t_{target} and the mostly straight lines confirm this expected linear relationship. However, the lines of all three algorithm variants feature a different gradient, implying the runtime of the approach based on dynamic programming increases at a higher rate than the runtimes of both variants of the multiple goal A* algorithm for increasing target durations.

In conclusion, both variants of the multiple goal A* algorithm offer a reduced runtime in comparison to the dynamic programming approach for longer input and output tracks. This difference grows even further when increasing either the length of the input track or the target duration.

When activating the segmentation enforcement, theoretically one would expect both variants of the A* algorithm to become faster, because many nodes do not have to be considered as they are assigned an infinitely high cost due to violating the segmentation constraint. However, comparing the performance between disabled segmentation enforcement in figures 5.3 (a) and (c) and enabled segmentation enforcement in figures 5.3 (b) and (d) leads to the conclusion that the runtimes of both multiple goal A* algorithms are roughly equal. Perhaps, the supposed speed-up is cancelled out by the increased overhead from having to look up the cluster of the bixel corresponding to a successor node, but further investigation would be required to confirm this as the cause. Additionally, the dynamic programming approach is faster with segmentation enforcement enabled, although the runtime should theoretically be independent of this factor: All of the $|\mathcal{E}| = N^2(k_{\max} - 2) + N$ edges each representing a bixel transition at a specific position in the bixel path are always taken into consideration, regardless of how costly the optimal path to a node turns out to be. A potential cause could be related to the computation of the optimal path to the current node based on all predecessor nodes: In our implementation, the optimum cost for every predecessor node is added to the cost of the respective bixel transition leading to the current node. Iterating over all predecessor nodes, this cost is computed for every node and a conditional statement checks whether it is lower than the least known cost until this point. In that case, this cost is set as the new minimum cost. After the iteration, the optimum cost to the current node was determined as the minimum of the mentioned costs associated with all predecessor nodes. *Branch prediction* as a processor feature may correctly assume that none of the predecessor nodes with an infinitely high cost due to violating the segmentation constraints are involved in the optimal path to the current bixel and therefore evaluating the conditional statement checking for a path with less cost is significantly faster. Further investigations should be undertaken in order to confirm the validity of the above explanations or to uncover the actual causes.

In summary, the proposed multiple goal A* algorithm achieved lower runtimes for longer input or output tracks than the dynamic programming approach from [48] and almost equal runtimes for shorter input or output tracks. However, it still has the same asymptotic complexity of $O(N^2\ k_{\max})$: The runtime increases quadratically with the duration of the input track and linearly with the target duration, only with a smaller factor. Using the heuristic designed specifically for the search of the optimal bixel path provided an additional, albeit slight, decrease in average runtime.

In the following section, repetition avoidance as a user constraint is evaluated by observing its effects on the generated output.

5.1.3. Repetition avoidance

Repetition avoidance was introduced as a user constraint in section 4.3.4 and is ideally evaluated with a study using multiple output tracks generated with the same input track and constraints, except different settings for the repetition avoidance parameter w_{r}. It would then involve participants listening to these output tracks and reporting their level of annoyance caused by the potentially occurring repetitions. Because this would pose a large time demand on the participants and would likely reduce the number of people willing to take part in such a study significantly, we instead employ an automatic evaluation.

In order to rate the output quality, we design a metric estimating the expected level of annoyance a listener would report when listening to the output track. The metric is formulated as a function r_{path} measuring the repetitiveness of any bixel path $\mathbf{p}$ containing $k \geq 2$ indices of bixels:

$$\begin{aligned} r_{\text{path}}(\mathbf{p}) &= \frac{1}{k-1} \cdot \sum_{i=2}^{k} r_{\text{bixel}}(\mathbf{p}, i) \qquad \text{where} \\ r_{\text{bixel}}(\mathbf{p}, i) &= \sum_{1 \leq j < i, p_j = p_i} \frac{1}{i-j}. \end{aligned} \tag{5.1}$$

The function $r_{\text{bixel}}(\mathbf{p}, i)$ models the annoyance that a specific bixel at a position i in the bixel path $\mathbf{p}$ causes by computing a sum over all previous positions at which this bixel already occurred. These positions of repetition j are measured in their level of annoyance according to their distance in bixels $i - j$ to the current position i. The idea behind this computation is that repeating a bixel after a certain number of other bixels x have been played is presumed to be approximately twice as annoying as repeating a bixel after $2x$ other bixels have been played.

Overall, the metric r_{path} modelling the annoyance of a bixel path is calculated as the average of the annoyance levels r_{bixel} of all contained bixels (except the first one, as no repetition can occur at the beginning).

Equipped with this metric, we executed the standard test set with a range of different settings for the repetition avoidance parameter $w_r \in \{0, 0.5, 1, 1.5, 2\}$, aiming to explore the lower possible values starting with zero as the least possible value, and analysed the generated bixel paths regarding their repetitiveness. Grouping all test cases according to their value of the repetition avoidance parameter w_r and computing the average for every group yields the bar plot in figure 5.4 illustrating the average values of the metric r_{path} for all output tracks generated with a certain setting for w_r. For increasing values of w_r, the estimated repetitiveness of the results strictly decreased on average, demonstrating the success of the proposed implementation of repetition avoidance at least in the context of the proposed metric. Future work could include a listening study in order to investigate the ability of our metric to accurately model the listener's experience by comparing the reported levels of annoyance to the values of the metric. Although the average value of the metric significantly dropped when setting the repetition avoidance parameter from 0 to 0.5, there were diminishing returns for larger values of w_r, yielding only small improvements. One possible reason could be that the penalty function in equation (4.30) does not penalise backward jumps of slightly larger distance enough. Another possibility is that the remaining short backward jumps are unavoidable and are therefore included in the output despite their high cost, because they are strictly required for an output that fulfils the user constraints: Due to the missing tolerance t_{tol} regarding the duration of the output, a specific backward jump of short distance may be necessary to reach the optimal solution that minimally deviates in length from the target duration t_{target}.

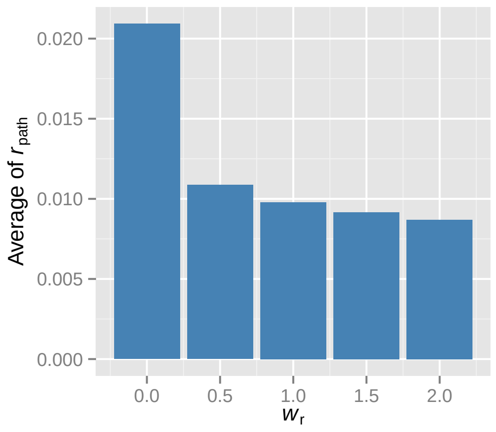


Figure 5.4.: Average values of the metric r_{path} for output tracks generated with different settings for w_r.

In conclusion, we set $w_{\text{r}} = 0.5$ as the default value in our music rearrangement system, as it offers a significant reduction in the estimated level of annoyance, while losing as little information as possible due to the minimisation operation in equation (4.29), where too large values are limited to the maximum value of one.

In the next section, we demonstrate the effects of the segmentation tolerances t_{low} and t_{high} from section 4.3.6 on the output quality.

5.1.4. Segmentation tolerance

The segmentation enforcement in section 4.3.6 includes two tolerance parameters t_{low} and t_{high} describing the amount of seconds the segment transitions of the result are allowed to deviate from the segment transitions in the target segmentation. A deviation of t_{low} seconds does not lead to a higher cost of the solution, while deviations between t_{low} and t_{high} seconds are penalised with a higher cost depending on their amount. Utilising tolerances is intended as a trade-off between the output quality and how accurately the target segmentation is enforced, allowing for an output track with less perceptible cuts whose segmentation slightly deviates from the target segmentation. In this section, we will investigate some effects of these tolerances on the resulting music pieces to determine whether this concept succeeds in offering the aforementioned trade-off.

Because the two tolerance parameters t_{low} and t_{high} both represent an amount of seconds as a non-negative, real number, the space of possible configurations is infinitely large. Therefore, we define a set of reasonable values $\mathcal{P} = \{0, 2, 4, 6, 8, 10\}$ for both parameters and derive the set of considered configurations by selecting all combinations from $\mathcal{P} \times \mathcal{P}$ which are valid, that is, where $t_{\text{low}} \leq t_{\text{high}}$. Note that the configuration with $t_{\text{low}} = 0$ and $t_{\text{high}} = 0$ represents the segmentation enforcement without any tolerance employed in [48]. For every one of these configurations, the standard test set was executed.

With the generated results, we explore the influence of the tolerances t_{low} and t_{high} on the accuracy of the segmentation enforcement and the quality of the output. We define the measure **segmentation accuracy** as a metric to measure the similarity between the target and the result segmentation, which is a real number in the range $[0, 1]$ computed as follows: The amount of time in which the resulting track contains audio material from the correct cluster defined in the target segmentation is divided by its total duration. This ratio represents the percentage of time where the segmentation of the result matches the target segmentation. Therefore, the value 1.0 implies a perfect segmentation of the result without any deviations from the target segmentation and larger values are generally better. The overall audio quality of the output track is estimated with the total cost $c_{\text{path}}(\mathbf{p})$ of the optimal bixel path, which consists of the sum of all bixel transition costs, and the number of jumps in the solution. This evaluation approach relies on two assumptions to produce meaningful results. First, the costs of the jumps as indicated by the unified transition cost matrix $\mathbf{T}$ must correlate to how perceptible or irritating the resulting cut would be for a listener. We will examine the validity of this assumption in the listening study in section 5.2.1. Second, we presume that using less jumps tends to produce a better track overall than a result with more jumps, because every cut potentially leads to a degradation of the perceived quality.

Based on the above assumptions, the generated tracks can be automatically evaluated. The results are shown in figure 5.5 and visualise the average values of different metrics for every considered configuration of t_{low} and t_{high} with heatmaps employing a scale of colours for the occurring values.

Figure 5.5 (a) illustrates the average segmentation accuracy ("Seg. Acc.") for different tolerance configurations. Wenner's method [48] using no tolerance in the lower left achieved almost perfect segmentation accuracy, producing results whose segmentations match the target segmentation about 99% of the time. As expected, the segmentation accuracy declined for larger tolerance settings, but did so slowly, not dropping below 90% even when setting both parameters to ten seconds. Although both tolerance settings lead to a greater amount of feasible jumps, increasing t_{high} also penalises jumps producing a deviation from the target segmentation, in contrast to t_{low}. This explains why the parameter t_{high} had a smaller influence on the segmentation accuracy than the parameter t_{low} in the experiment.

In return for the lower average segmentation accuracy, the cost $c_{\text{path}}(\mathbf{o}_{k_{\text{opt}}})$ of the

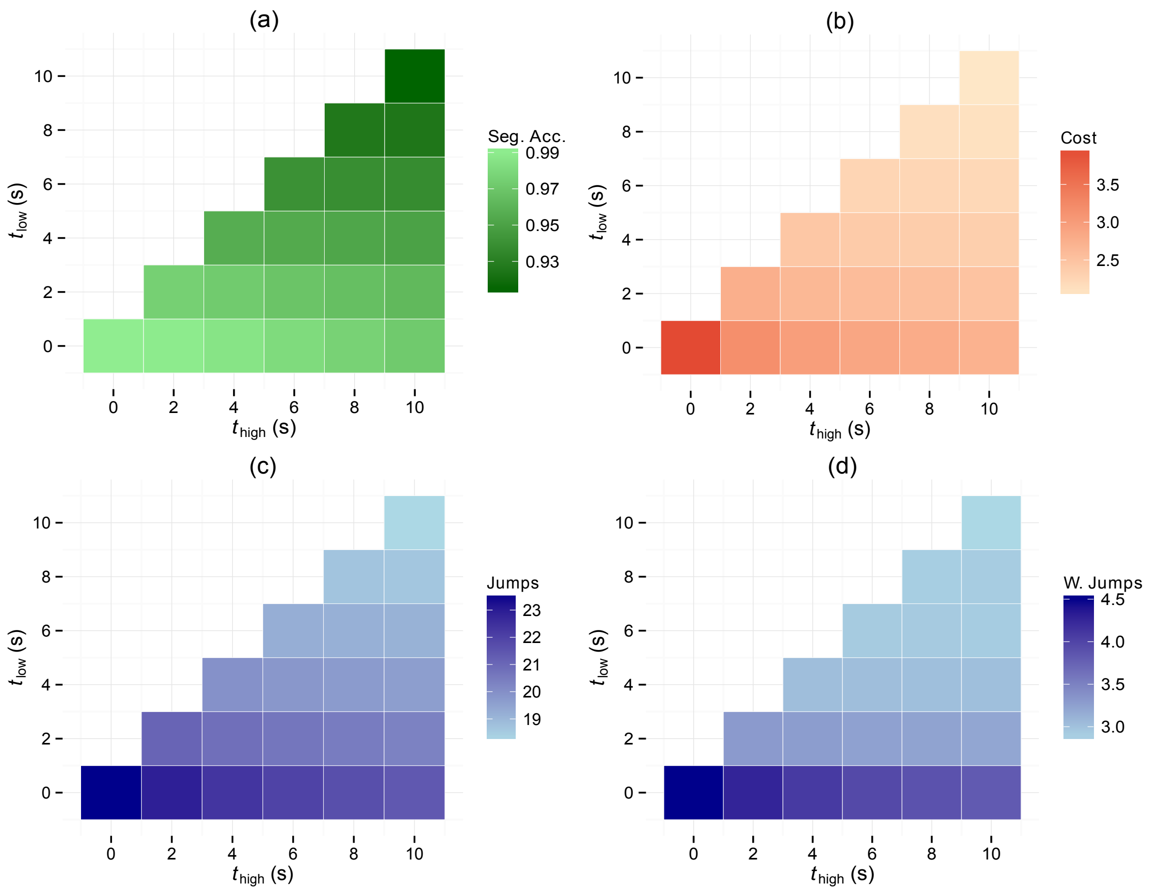


Figure 5.5.: Heatmaps visualising the average (a) segmentation accuracy, (b) cost of the bixel path, (c) number of jumps in the bixel path and (d) number of wrong jumps in the bixel path for various configurations of the segmentation tolerances t_{low} and t_{high}.

optimal bixel path $\mathbf{o}_{k_{\text{opt}}}$ averaged over all experiments in the standard test set was considerably lower when employing tolerances with $t_{\text{high}} \geq 2$ compared to the configuration without tolerance, as figure 5.5 (b) demonstrates. Increasing tolerance values further decreased the total average cost slightly and steadily. Assuming the cost $T_{i,j}$ of a bixel jump from any bixel b_i to any other bixel b_j relates to the perceived quality of the cut when concatenating both bixels, this result proves a higher average quality of the output tracks generated with larger segmentation tolerances compared to tracks generated using lower values for the tolerance parameters.

In addition to the estimated quality of the selected jumps, we measure the number of jumps being included in a result on average depending on the settings for segmentation tolerance and display the results in figure 5.5 (c). Without any tolerance, the generated bixel path of a solution contained more than 23 jumps on average. When increasing either t_{low} or t_{high}, this number decreased steadily to about 18.2 for the highest considered tolerance configuration. One reason is that fewer jumps with a small distance have to be included in the solution to correct the positions of the resulting segment transitions in order to align them precisely with the segment transitions in the target segmentation. In particular, sometimes a wrong jump causing an involuntary extension of the solution when trying to shorten the original piece or vice versa is necessary to enforce the segmentation accurately. The average number of these wrong jumps ("W. Jumps") serving no other purpose than fulfilling the segmentation constraint also decreased with higher tolerances, as shown in figure 5.5 (d). We presume the decrease in the average number of (wrong) jumps to be a positive influence on the output quality, because every jump potentially causes an irritating cut in the result.

Considering the results of the evaluation, we set $t_{\text{low}} = 2$ and $t_{\text{high}} = 4$ as default for our music rearrangement system, because it provided a significantly reduced cost, only slightly reduced segmentation accuracy and a fewer average number of wrong jumps. However, this represents only a general recommendation for the user and can be interactively changed.

In conclusion, the concept of segmentation tolerance was successful in providing a trade-off between segmentation accuracy and output quality, assuming the cost of a bixel path and the number of jumps provide an indication of how pleasing the resulting piece would be rated by a human listener. The proposed method of enforcing a segmentation with tolerances fulfilled its purpose in the context of path optimisation by reducing the cost of the optimal solution and the number of jumps. As a consequence, even if the aforementioned assumptions do not prove to be true, the unified cost matrix **T** would need an adjustment to better reflect how a human listener would rate the qualities of the bixel transitions, but not the method of segmentation enforcement. This relationship between the unified cost matrix **T** and the perceived quality of the respectively produced transitions will be explored in the listening study conducted in the following section 5.2.

5.2. Listening study

This section describes the listening study conducted to evaluate the proposed music rearrangement system regarding the quality of its output tracks. In particular, we focused on analysing the perceptibility of the produced cuts by presenting audio snippets containing three seconds of audio before and after a cut in an output track to the listener and asking different types of questions regarding their quality. We collected the required data from participants through two online surveys, where the audio snippets could be played as often as desired and an associated question had to be answered and submitted. In total, 28 people completed the first survey, while the second survey had 45 responses.
The listening study investigates two different aspects of the music rearrangement system. To evaluate the transition quality of the selected bixel jumps, the participant had to rate audio excerpts in the following section 5.2.1 according to different musical aspects. Each of these excerpts contained the result of a bixel jump and was not processed with the jump optimisation stage from section 4.4 to evaluate the ability of our system to find the bixel jumps leading to the least perceptible cuts. The second area of interest pertains to the jump optimisation stage and whether the application of this stage to the produced cuts has a positive influence on the perceived transition quality. This investigation is described in section 5.2.2 and used pairs of audio snippets each containing the same bixel jump, where only one of them was processed with our jump optimisation algorithms, and asked listeners to select which one of the two given audio snippets is preferred. Before discussing these areas of evaluation more thoroughly in the following sections, we will describe the general design of the listening study in the remainder of this section.

Used database In order to determine the possible influence that knowing a music piece can have on the response of a listener, we selected four rather popular tracks as well as four music pieces that turned out to be completely unknown to all of the participants and combined them, yielding the database EVAL presented in table A.5.

The first four entries in this table represent the better known tracks and were sourced from the private collection of the author, while the last four tracks are freely available on the Free Music Archive [49] and are also contained in the databases CC1 and CC2. To limit the time required for participating in the study to about fifteen minutes, only the eight tracks from the database EVAL were used and also split across two different online surveys, each including two of the more well-known tracks along with two of the unknown tracks. Additionally, we attempted to cover a wide range of genres despite the relatively low total number of tracks, as seen in the columns describing the genre and subgenre of every piece.

Data generation New music pieces are required as output from the music rearrangement system to create an audio snippet for every cut that can be presented to the participant of the listening study. To generate this output, we define the **typical test set** as the collection of inputs for the music rearrangement system. It consists of two settings for the target duration t_{target}: One requires extending the original piece to twice its length, while the other one demands an output half as long as the original. Both settings for the target duration employ no tolerance t_{tol} and are included in the typical test set with and without segmentation enforcement, leading to four different constraint configurations each with all other constraints set to their default values. Similar to the standard test set used for the automatic evaluation in section 5.1, the ground truth segmentation is automatically determined with $C_{\text{range}} = [3, 5]$ and the target segmentation is a scaled version of the ground truth segmentation. The database EVAL from table A.5 is selected as set of inputs for the original music piece, resulting in $4 \cdot 8 = 32$ total configurations in the typical test set. In comparison to the standard test set, the typical test set is considerably smaller, because only a relatively small amount of audio data can be manually evaluated by a human in a listening study. Despite that, the typical test set was designed in an effort to cover the main application scenarios, including extending and shortening a piece in combination with either using or not using segmentation enforcement.

Additional questions Complementing the main questions presented in the following sections, we explore the possible effects of the musical education of listeners and their knowledge of the tracks on their responses.

In both online surveys, participants had to classify their level of musical education according to four different categories, ranging from no musical knowledge and not playing any music instrument to being a professional musician. The following sections will refer to the levels of musical education with the numbers one to four, where

a higher number implies a higher level of musical education.
The online surveys were structured according to the four original music pieces from the EVAL database: At first, the listener was able to listen to a preview of the original track and had to indicate whether he knows the track. Afterwards, a set of audio snippets generated with this track as input was presented before proceeding analogously with the next original track.
The main questions regarding the audio snippets along with an analysis of the associated responses will take place in the following sections, starting with the evaluation of the transition quality.

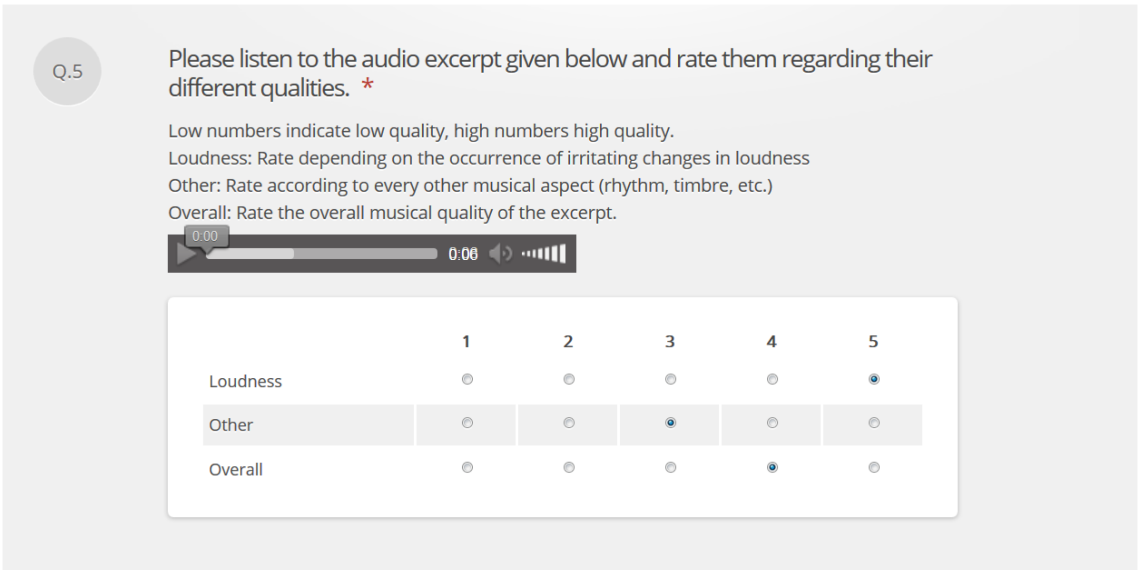
5.2.1. Transition quality

Arguably the most important criterion for evaluating a music rearrangement system is the quality of the produced tracks. The cuts as a result of the jump-based approach are particularly crucial, because they represent the only sections in the output that significantly differ from the original and where discontinuities can irritate a listener. The goal of this section is to yield insight into the following issues related to the transition quality of these cuts: Probably most importantly, we will evaluate the overall quality of the transitions at the cuts. Additionally, we will investigate the effect on transition quality when changing the influence of timbre and loudness on the matrix $\mathbf{T}$ used for path optimisation. This is achieved by experimenting with different weights w_d and w_l for the matrix $\mathbf{D}$ regarding timbre from section 4.2.1 and the matrix $\mathbf{L}$ regarding loudness from section 4.2.2, respectively. Furthermore, the potential effects of musical education and familiarity with the presented track on the responses will be analysed.

Data generation and selection Due to the large number of cuts produced when enforcing a segmentation, the configurations with active segmentation enforcement had to be excluded from the typical test set for the purpose of this section. For every remaining configuration in the typical test set, a set $\{0, 0.1, 0.2, 0.3, 0.4\}$ of values as weights w_l for the loudness matrix $\mathbf{L}$ was used to produce a total of five tracks, allowing us to analyse the effect of $\mathbf{L}$ on the quality of the selected jumps. No larger values than 0.4 were used for the weight w_l in order to keep the number of produced cuts low and because it was assumed that loudness continuity does not have a greater influence on the overall transition quality. We set $w_\mathrm{d} = 1 - w_\mathrm{l}$, so the transition cost matrix $\mathbf{T}$ included the matrices $\mathbf{L}$ and $\mathbf{D}$ to a different degree and all weights summed up to one, as required in equation (4.28). The loudness equalisation method from section 4.4.2 was not used, as the jumps were intended to be rated regarding their loudness continuity without any further modification in order to focus on the influence of the loudness weight w_l. From the results, an audio snippet for every cut was generated, containing six seconds of audio centered on the cut position.

For every track from the EVAL database, a random configuration from the test set described above that uses the respective track was selected. All cuts corresponding to these selected test cases were included in the listening study.

Study design The audio excerpts selected according to the previous paragraph were presented to each participant, who was asked to rate them according to three different aspects, as shown in figure 5.6. Every aspect had to be rated on a scale from one to five, where higher numbers indicate a higher perceived quality. The first aspect is the loudness continuity introduced in section 2.1, while the second aspect covers all other factors required for an imperceptible transition, such as timbre, rhythm and meter. Finally, the participant had to rate the overall quality of the audio snippet. We deliberately separated the loudness continuity from all other factors to focus on evaluating the usefulness of the proposed loudness matrix **L**.



Q.5

Please listen to the audio excerpt given below and rate them regarding their different qualities. *

Low numbers indicate low quality, high numbers high quality.
Loudness: Rate depending on the occurrence of irritating changes in loudness
Other: Rate according to every other musical aspect (rhythm, timbre, etc.)
Overall: Rate the overall musical quality of the excerpt.

	1	2	3	4	5
Loudness	○	○	○	○	●
Other	○	○	●	○	○
Overall	○	○	○	●	○

Figure 5.6.: One of the questions of the online survey designed to investigate the transition quality of the music rearrangement system. An audio excerpt could be played arbitrarily often and had to be rated according to three different aspects on a scale from one to five.

Discussion of results On a range from one to five, the average ratings for loudness continuity, all other musical aspects and the transition quality overall were 3.56, 3.35 and 3.27, respectively. Normally, one would expect the overall transition quality to be somewhere in between the ratings for the loudness continuity and the other musical aspects, because both represent subcategories and in combination should lead to a rating of the overall transition quality. However, this was not the case – one reason could be a misunderstanding of the category ”Other”, because some of the corresponding aspects like timbre were perhaps unknown.

Averaging these ratings over all cuts associated with a specific setting of the loudness weight w_l results in the plot in figure 5.7. For increasing weights w_l starting with zero, the produced cuts rapidly increased in their loudness continuity until $w_\text{l} = 0.2$ and afterwards stayed at a relatively high level, as the red curve shows. This proves the ability of the proposed loudness matrix $\mathbf{L}$ to capture the concept of loudness continuity, because it leads to the selection of jumps with higher loudness continuity when its influence w_l is increased. The ratings concerning all other musical aspects (”Other”) first decreased in the process as expected, but suddenly increased slightly for $w_\text{l} > 0.2$. A possible explanation is that the loudness matrix $\mathbf{L}$ unintentionally encodes some information about these aspects in addition to loudness continuity, but further investigation would be necessary to determine the cause.

Overall, the transition quality first declined due to the decreased quality in other

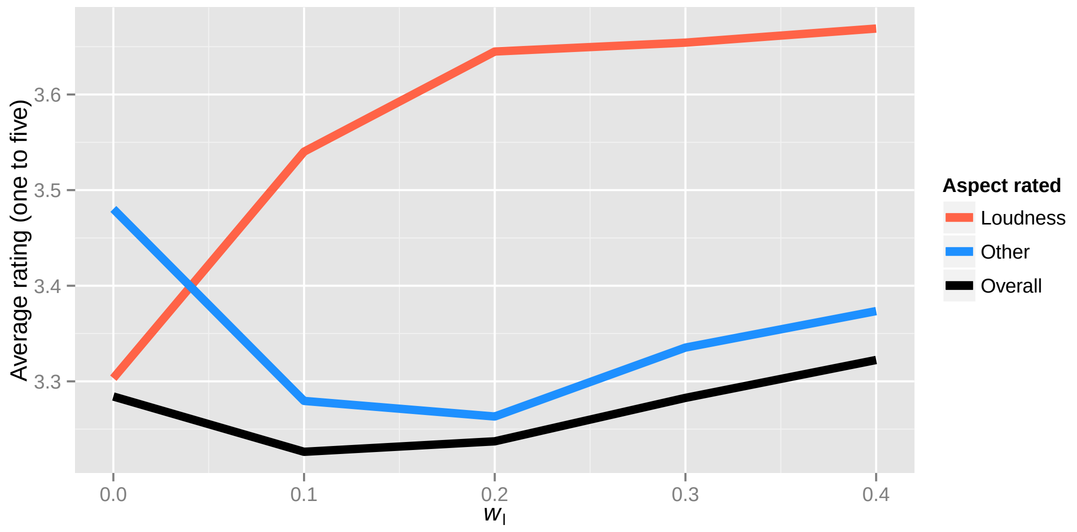


Figure 5.7.: Average ratings of the loudness continuity (”Loudness”), all other musical aspects (”Other”) and of the transition quality as a whole (”Overall”) for different values of the loudness weight w_l.

musical aspects, but steadily increased for larger values for w_l, leading to the conclusion that the loudness matrix **L** could be assigned a larger influence than previously expected when choosing the considered range of values for w_l. Therefore, we set the default weights for our music rearrangement system to $w_\mathrm{l} = 0.4$ and $w_\mathrm{d} = 1 - w_\mathrm{l} = 0.6$, but a further investigation concerning larger values for w_l should be conducted, as the overall transition quality should reach a maximum and then decline at some point when increasing w_l further, as the selected jumps tend to ignore all other musical aspects except loudness.

Instead of analysing the average ratings of the selected jumps when changing the weights for the different matrices regarding timbre and loudness, we can also skip the path optimisation step and directly explore the relationship between the bixel transition costs contained in these matrices and the perceived transition quality of the respectively produced cuts. Ideally, these costs from the cost matrices **D** and **L** should correlate very strongly with the ratings of the participants, because the path optimisation process relies on the accuracy of these transition costs to be able to select the best sounding bixel transitions.

Plotting the bixel transition costs concerning timbre and loudness against the average rating regarding loudness and other musical aspects for every audio snippet in the listening study results in figure 5.8. In particular, figure 5.8 (a) visualises the relationship between the cost $D_{i,j}$ of a bixel jump and how highly it was rated in the

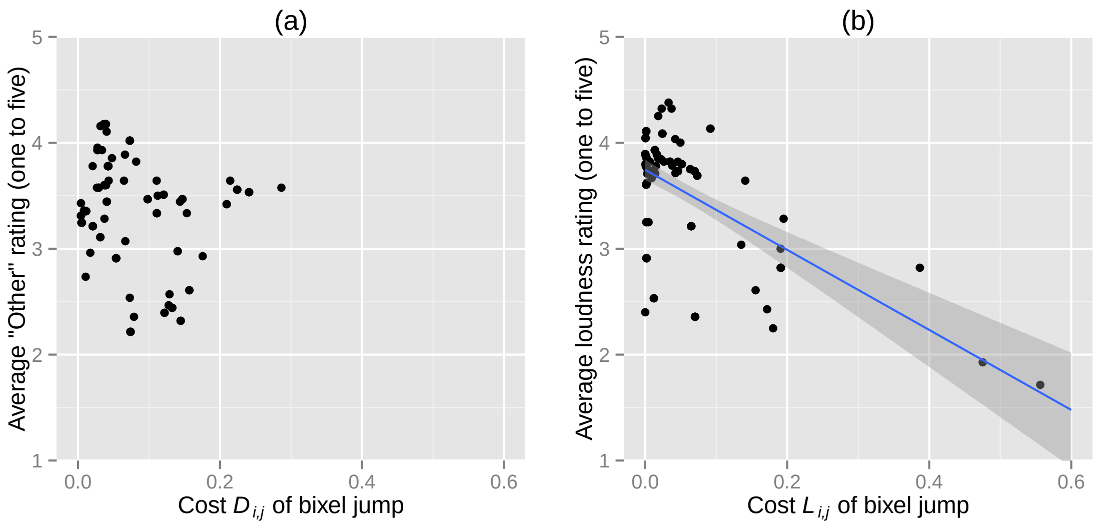


Figure 5.8.: (a) Average rating of all other musical aspects except loudness for bixel jumps depending on their costs $D_{i,j}$ concerning timbre. (b) Average loudness ratings for bixel jumps depending on their costs $L_{i,j}$ regarding loudness.

category ”Other” on average. A perfect result would consist of points scattered across a straight line, beginning with a high average rating for the least costly bixel jump and ending with a low average rating for the most expensive bixel jump. Unfortunately, the result has not only a very different appearance, but also does not exhibit any significant correlation between the two variables at all. Although the matrix **D** focuses on timbre and is not meant to capture other musical aspects, future work could include integrating more music features into its computation. Alternatively, separate matrices with their respective influences on the transition cost matrix **T** could be developed, analogously to the loudness matrix **L**. Figure 5.8 (b) on the other hand demonstrates the relation between the cost $L_{i,j}$ of a bixel jump and its average loudness rating with a *regression line.* The grey area demonstrates where the true regression line is located with a confidence of 95%, obtained by calculating the variance of its estimated slope and intersection. We compute the Spearman rank correlation and test for its *significance* [51] to prove the existence of the desired monotonic relationship between these variables. The resulting correlation coefficient is $\rho \approx -0.35$ and was found to be significant, as the true correlation coefficient is in the range $[-0.5, -0.18]$ with a probability of 95%. This proves the ability of our proposed loudness matrix **L** to at least partially reflect the actual transition quality regarding loudness.
No significant relationships were found between the musical education of the listener and the ratings when using $p = 0.05$ as level of significance. An unexpected connection between knowing a track and the respective ratings was discovered: The result of a *paired Wilcoxon signed-rank test* [41] employed to compare the overall average ratings between the participants that did and the participants that did not know the track showed that with a probability of 95%, knowing the track led to overall ratings that are between 0.2 and 0.48 points higher on average.
Following the evaluation of the transition quality of the selected bixel jumps without applying jump optimisation, the next section investigates whether the jump optimisation stage improves these results even further.

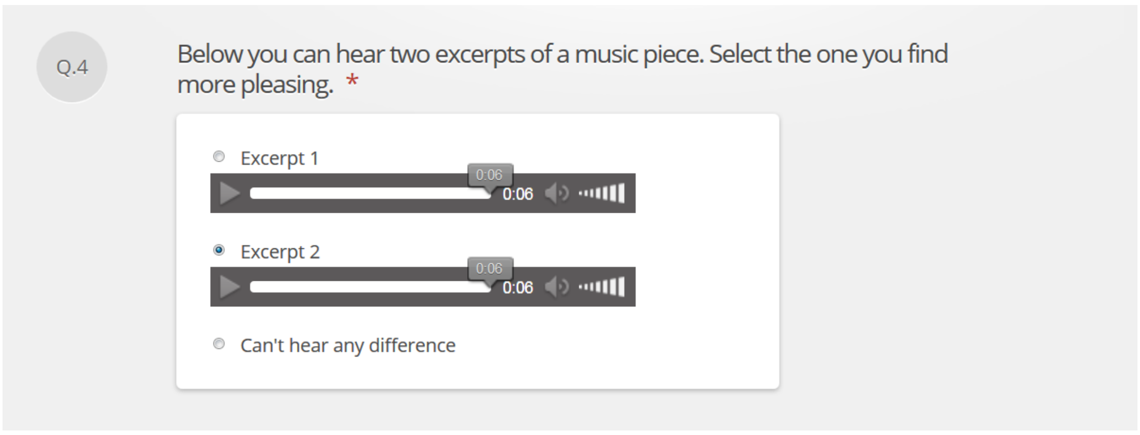
5.2.2. Jump optimisation

This section specifically focuses on the evaluation of the jump optimisation in the proposed music rearrangement system described in section 4.4.
Because the goal of the jump optimisation stage is to enhance the quality of the signal around a cut, this part of the listening study involved participants listening to two versions containing the same bixel jump, but only one of them was post-

processed with jump optimisation, and then determines if the jump optimisation makes this version preferable. Additionally, as not every jump required such post-processing, we compute how often the jump optimisation significantly changed the audio signal around the cut. Similar to the previous section, possible effects of the musical education and track knowledge of the participants on their responses will be considered.

Data generation and selection Again, the typical test set was executed to generate the required output tracks, this time once with jump optimisation and once without in order to analyse the differences. For every cut, an audio snippet containing six seconds of audio centered around the cut position was extracted. Afterwards, the resulting audio snippets were grouped into pairs, where a pair of audio snippets contained the same bixel jump, but only one of the two audio snippets was processed by the jump optimisation procedure.
For every track in the database EVAL, two pairs of audio snippets were randomly selected from all pairs of audio snippets belonging to the respective track, with the following exception: Only pairs of audio snippets containing a bixel jump that is considered to be **significantly altered** by the jump optimisation process were available for the random selection. We define a bixel jump to be significantly altered by the jump optimisation, if it exhibits an absolute loudness difference $|l_{\text{after}} - l_{\text{before}}|$ of at least 5 sone and/or an absolute detected synchronisation error $|r_{\max}|$ of at least 0.05 seconds. The goal of this threshold was to only include pairs of audio snippets in the surveys that were significantly changed by the jump optimisation procedure and to exclude pairs of audio snippets with imperceptible differences. From all 556 generated pairs, 511 were significantly altered, which leads to the conclusion that the jump optimisation procedure appears to change the signal of almost every produced cut significantly, thus emphasising the relevance of our investigation concerning the effects of these changes.
All in all, the random selection according to the above rules yielded 32 audio snippets in total, distributed across eight tracks each with two pairs of audio snippets.

Study design For each selected pair of audio snippets, we asked the participants to select the audio snippet that is perceived as more pleasing, as shown in figure 5.9. No further information was given, meaning the participant was not able to determine which audio snippet was additionally processed and therefore had to rely only on the presented musical content. Either the participant was able to hear a difference and selected one of the two audio snippets as the answer for the question, or reported not


Q.4

Below you can hear two excerpts of a music piece. Select the one you find more pleasing. *

Excerpt 1 [0:06]

[✓ Excerpt 2] [0:06]

Can't hear any difference

Figure 5.9.: One of the questions of the online survey designed to investigate the effects of jump optimisation on the quality of the produced cuts. Two audio excerpts containing the same cut could be played, but only one of them was additionally processed by jump optimisation.

being able to distinguish both audio snippets with a third option. This set-up of the study allows us to investigate how often the additionally processed audio snippets were preferred and how often a perceptible change was applied to the audio signal.

Discussion of results Figure 5.10 illustrates the distribution of the responses averaged over all pairs of audio snippets and also distinguishes between the different levels of musical education reported by the participants. The red segments should ideally appear small if the jump optimisation works as intended, as they illustrate the average percentage of cases in which the unedited audio snippet without any jump optimisation was preferred over the additionally processed audio snippet.
Overall, the jump optimisation stage appeared to worsen the perceived quality of the result in about 27% of all cases (”Preferred unedited”). An equally large portion, highlighted in blue, represents how often the participants were not able to distinguish the two respective audio snippets on average (”No difference”). On average, 47% of all answers indicated a preference of the processed audio snippet over the unedited one (”Preferred JO”). In summary, the audio snippets produced by the jump optimisation procedure were rated at least as pleasing as or more pleasing than the unedited version 73% of the time on average.
Particularly interesting is the effect of the musical education of listeners on their average responses, as the percentage of responses indicating the preference of the unedited audio snippet (”Preferred unedited”) tended to decrease with increasing reported levels of musical education. However, this effect is not necessarily significant, especially because only four of the 73 participants identified themselves as profes-

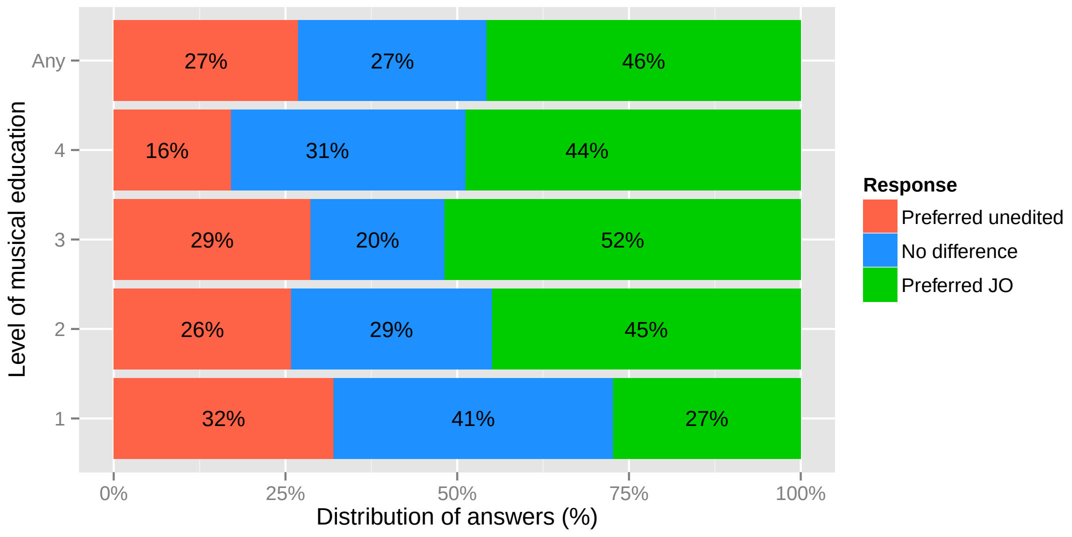


Figure 5.10.: The distribution of responses as an average over all questions each associated with a pair of audio snippets asking which snippet is preferred. In the top row, all responses regardless of the respondents' musical education are considered (Musical education: ”Any”). Additional distributions, which only take participants with a certain level of musical education ranging from no musical education (1) to professional musicians (4) into account, are presented below.

sional musicians (level of musical education: 4) and consequently, the corresponding results are subject to deviations caused by outliers due to the low sample size.

Considering the possible explanations for the cuts which actually worsened in quality after applying the jump optimisation procedure, either the loudness equalisation method from section 4.4.2 or the synchronisation algorithm from section 4.4.1 must have caused the issue. In an attempt to identify this cause, we analyse the extent to which both of these algorithms alter the signal and how it relates to the output quality as measured by the participant's responses.

Considering the loudness equalisation method, which was fully activated with $l_{\text{factor}} = 1$ when generating the output for this part of the study, the extent to which the signal is changed should depend on the absolute detected loudness difference $|l_{\text{after}} - l_{\text{before}}|$ measured in sone. When assigning the preference of the snippet with jump optimisation a value of 1, a preference of the unedited snippet a value of 0, and the statement of not hearing any difference a ”neutral” value of 0.5, the **average response score** is obtained by taking the mean of the responses for a pair of audio snippets. This numerical representation of the distribution of responses allows us to investigate the issue further: In figure 5.11 (a), the average response score is shown along with the absolute detected loudness difference for every bixel jump and demonstrates that a larger alteration of the signal's loudness tended to improve the resulting audio around the cut. The correlation with a coefficient of approximately 0.66 was significant: With a 95% probability, the true correlation coefficient was in the interval $[0.3, 1]$ and confirms the tendency of the loudness equalisation method to improve the signal quality, assuming it changes the signal at all. It could however still miss actually existing loudness differences and mistakenly not correct them, which is not accounted for in this observation. When only taking responses made with knowledge of the track into account, this relationship exhibits an even higher correlation, illustrated in figure 5.11 (c), with a correlation coefficient of 0.87 and a 95% probability for the true correlation coefficient to be in the range $[0.66, 1]$.

For all presented pairs of audio snippets, figure 5.11 (b) visualises the relationship between the absolute delay $|r_{\max}|$ in seconds that was detected and subsequently corrected by the synchronisation method and the average response score. Although we

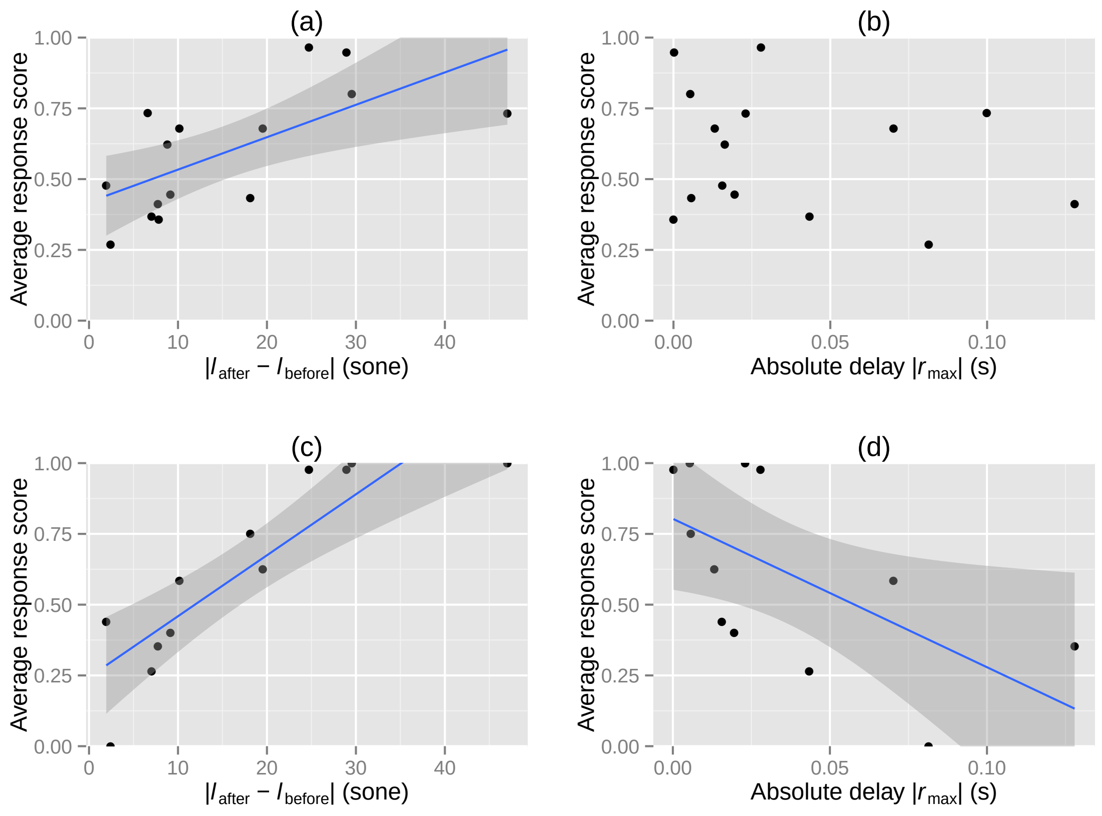


Figure 5.11.: The average response scores for every pair of audio snippets depending on the detected loudness difference $|l_{\text{after}} - l_{\text{before}}|$ in sone ((a) and (c)) and on the absolute delay $|r_{\max}|$ ((b) and (d)). Figures (a) and (b) consider all responses, while figures (c) and (d) only take responses from participants that knew the respective track into account.

did not find a significant correlation, this changed when only considering responses made with knowledge of the respective track: For this subset of responses, we found a significant negative correlation visible in figure 5.11 (d) with a coefficient of -0.6, and a 95% probability of the true correlation being between -1 and -0.15. This demonstrates that the synchronisation method tended to at least slightly decrease the output quality when rated by listeners who were familiar with the original track.
In conclusion, the decreased quality of some audio snippets after processing by the jump optimisation stage is probably caused by the synchronisation algorithm, not the loudness equalisation method. This problem could occur due to changing the jump times incorrectly, that is, determining a delay r_{max} that does not correspond to the synchronisation error present at the beat position. As a result, the audio around the cut contains an irritating rhythmic inconsistency, caused by the unusual amount of time between the beat directly before the cut and the beat directly after the cut that deviates from the normally present inter-beat interval.
We found no significant differences in the average response score when only considering the responses made with knowledge of the respective track used to generate the audio snippets.
This concludes the evaluation of the music rearrangement system. In the last section, we will summarise our work, the achieved advancements in the field of music rearrangement and possible areas of improvement.

6. Conclusions

The goal of this thesis was to propose a system capable of solving the music rearrangement problem presented in section 3.1: Given an already existing piece of music and a set of user constraints, a new music piece has to be produced that fulfils these constraints by concatenating different sections of the original and optionally applying postprocessing techniques in the vicinity of the resulting cuts. In other words, the new music piece is generated by playing back the original piece and jumping from one position in the original to another at specific times.

We decided to base our approach on the previous work from Wenner [48] and intensively evaluated it with a self-built database containing a wide variety of music pieces in section 2.3. The results in section 2.4 revealed a range of different problems occurring at the cut positions in the produced pieces and represent a detailed report on the state of the art in the field of music rearrangement. In particular, we focused on avoiding and correcting sudden changes in loudness at the cuts at different stages of the system.

Our music rearrangement system begins by identifying the locations of the beats in the original music piece and uses them to subdivide the piece into a series of bixels, where each bixel contains the musical content between two consecutive beats. In section 4.1, we presented suitable metrics for selecting a beat tracker in the context of a music rearrangement system and chose a beat tracker on this basis. As improving or developing a beat tracker was out of the scope of this thesis, future work could include conducting further research in the field of beat tracking to indirectly improve the performance of bixel-based music rearrangement systems.

In section 4.2.2, we proposed a model describing how jarring a transition between any two bixels of the input track would be perceived regarding loudness in order to avoid such problematic transitions from being included in the output track. The transition qualities predicted by this model were found to significantly correlate with the perceived transition quality regarding loudness reported by a human listener, as the evaluation of the listening study in section 5.2.1 showed, indicating that the model successfully captures the hearing impressions of the listeners. As a result, increasing the influence of this model on the selection of the optimal transitions to be included in the output track leads to cuts with a demonstrably higher average rating regarding loudness. In addition to this loudness model, we adapted the model concerning timbre from Wenner [48] and combined both models. Generalising the idea of the loudness model, it could be viable to design models for every relevant musical aspect that influences the transition quality and to subsequently combine them using weights optimised with the help of listening studies. This could potentially lead to less perceptible cuts, because the selection of the optimal transitions would then take all musically relevant aspects into account.

The automatic segmentation method in section 4.2.3 was adopted from Wenner [48] to help the user in defining a description of the musical structure of the input track, which is later used to enforce a specific structure of the output track. It divides the piece into different segments and also clusters them according to their similarity. In addition, the intuitive user interface allows for an efficient editing of the automatically generated segments to achieve the desired structure. We extended this automatic segmentation method to automatically find the optimal number of clusters within a given range of possible numbers, which makes it easier for the user to obtain a suitable segmentation of the input track. However, its robustness could be improved by incorporating more musical features (like a feature describing the melody) into the feature vector used for the clustering algorithm.

The path optimisation in section 4.3 calculates the optimal order of bixels to use for the production of the output track and received particular attentation in this thesis. At first, the number of bixels required to produce a music piece with a desired length is estimated in section 4.3.2. The estimation succeeds in covering the optimal solution with a high probability that is always greater than the probability given by an additional parameter $p_{\min}$, as shown in section 5.1.1. Future work could include improving the underlying statistical model of the length of a bixel so the probability of covering the optimal solution is very close to the additional parameter $p_{\min}$ to make it more intuitive to use.

Because the path optimisation process accounts for the vast majority of the overall runtime when generating a new music piece according to the given user constraints, we focused in particular on accelerating this process. The graph-based perspective on the problem of finding the optimal arrangement of bixels provided the basis for proposing a multiple goal A* algorithm and the associated heuristics in section 4.3.5. This algorithm required significantly less time to generate a new solution compared to the dynamic programming approach by Wenner [48] and enabling the proposed heuristics slightly decreased the required time even further, as shown in section 5.1.2. Especially for long input or output tracks, our music rearrangement system achieves a greatly reduced runtime in comparison to Wenner's algorithm.

When extending a music piece, sometimes a short section of the original music piece is repeated a large number of times in a row, as the evaluation of Wenner's method revealed in section 2.4. The ability to avoid these repetitions was integrated into the system in section 4.3.4 and successfully reduced the estimated level of annoyance a listener would report when hearing the output track, as section 5.1.3 demonstrated. Future work could pertain to investigating whether the employed metric accurately represents the perceived level of annoyance by conducting the appropriate listening tests, which were excluded in this thesis due to time constraints.

From the path optimisation stage, an optimal set of jumps is obtained, indicating when and how to change the playback position in the original piece. In order to make the cuts produced from these jumps less perceptible for the listener, two main methods were devised in section 4.4 to optimise this set of jumps and the resulting cuts. In the jump synchronisation stage as the first main part of jump optimisation, the provided jump times based on the detected beat positions are corrected in case the beat tracker did not accurately locate the actual beat position, which sometimes leads to sound artefacts at the resulting cuts. The evaluation of this jump synchronisation method in section 5.2.2 led to the conclusion that it failed to increase the quality of the cuts in the average case. Cross-correlation as employed by the jump synchronisation method appears to be an inadequate approach for correcting the jump times, as it sometimes worsens particularly the rhythmical quality of the produced cut. Further work could include exploring a more suitable method to robustly synchronise the jump times and account for inaccurate beat positions. A method to smooth jarring loudness changes at the cuts presented in section 4.4 constitutes the second part of the jump optimisation. This loudness equalisation method significantly improved the quality of the produced cuts regarding loudness by altering the loudness of the signal parts before and after every problematic cut according to the evaluation in section 5.2.2. When employing both the jump synchronisation and the loudness equalisation, the evaluation in section 5.2.2 also demonstrates the success of our proposed jump optimisation procedure as a whole – as an average over all participants and presented cuts, it provided a better or equally good sound quality 73% of the time in comparison to the unaltered version of the cut.

Overall, the music rearrangement succeeds in creating new music of decent quality in most cases for a large number of genres. It also improves on the selected approach from Wenner [48] in several areas, as the extensive evaluation in section 5 showed. However, its components often involve several parameters listed in table A.2, whose settings could be optimised to enhance the system's performance even further. In addition, some of the issues mentioned in section 2.4 could not be implemented due to time constraints and thus also constitute a part of possible future work. Among these problems, cutting off vocals is especially important as it occurred frequently for tracks with vocals. Possible research could be concerned with finding the parts of a music piece that contain vocals and prohibit jumps in these regions, or even with synthesising new vocals around the produced cuts to make the transition less noticeable. The change of tempo at cuts was another issue and could be resolved by avoiding jumps from sections with a low tempo to sections with a high tempo and vice versa. In addition, the disregard for time signatures, where measures with an unusual number of beats were created, could be solved by detecting the time signature of the original piece and penalising such problematic jumps whose origin and destination are not located at the same positions within their respective measures.

A. Additional information

A.1. Parameters

Table A.1.: List of user constraints

Name	Default	Domain	Description	Page
t_{start}	0	Time of orig. track	Start position of result (s)	44
t_{end}	End of orig. track	Time of orig. track	End position of result (s)	44
t_{target}	60	$(0, \infty)$	Target duration (s)	44
t_{tol}	60	$(0, \infty)$	Duration tolerance (s)	44
$I(i)$	0	$[0, 1]$	Importance of every bixel	50
w_{d}	0.6	$[0, 1]$	Weight of matrix for timbre $\mathbf{D}$	51
w_{l}	0.4	$[0, 1]$	Weight of loudness matrix $\mathbf{L}$	51
w_{p}	0	$[0, 1]$	Weight of loudness function $I(i)$	51
w_{r}	0.5	$[0, \infty)$	Intensity of repetition avoidance	53
$s_{\text{ground}}(t)$		$t \in \mathbb{R}$	Ground truth segmentation	40
$s_{\text{target}}(t)$		$t \in \mathbb{R}$	Target segmentation	71
t_{low}	2	$[0, \infty)$	Allowed segm. deviation (s)	71
t_{high}	4	$[0, \infty)$	Penalised segm. deviation (s)	71
l_{factor}	1	$[0, 1]$	Intensity of loudness equalisation	87

Table A.2.: List of internal parameters

Name	Value	Domain	Description	Page
m	2	$\mathbb{N}_0$	Number of norm. binom. coefficients	31
t_{mean}	0.2	$(0, \infty)$	Window size for bixel loudness averages (s)	36
M	128	$\mathbb{N}$	Size of checkerboard kernel	40
c_{peak}	0.1	$[0, 1)$	Threshold for novelty curve maxima	40
p_{min}	0.6	$[0, 1)$	Minimum success probability	46
μ_{e}	0	$[0, \infty)$	Mean beat error (s)	78

Table A.2.: List of internal parameters (continued)

Genre	Value	Domain	Description	Page
σ_e	0.1	$(0, \infty)$	Std. deviation of beat error (s)	78
t_{sync}	0.5	$(0, \infty)$	Window for cross-correlation (s)	79
t_{lim}	3	$(0, \infty)$	Window for loudness equalisation (s)	84
t_{thresh}	0.1	$(0, \infty)$	Min. distance to loudness intersection (s)	85
t_{loud}	0.1	$(0, \infty)$	Window size of loudness average (s)	86
f_{trans}	3	$(0, \infty)$	Scale factor for loudness change (s)	88
t_{cross}	0.04	$[0, \infty)$	Duration of crossfade (s)	94

A.2. Databases

Table A.3.: Database CC1

Genre	Subgenre	Artist	Title
Blues		Arne Bang Huseby	Stormy Blues
Blues		Julia Haltigan	All I Can Think Of Is You
Country		Randy Travis and Brandon	Amazing Grace
Electronic	Ambient	LASERS	Amsterdam
Electronic	Chip	an0va	The First Noel / O Come All Ye Faithful
Electronic	Dance	Broke For Free	Calm The Fuck Down
Electronic	Downtempo	Tours	Enthusiast
Electronic	Dubstep	LittleLight	Illumination
Electronic	Glitch	Oribque	Simple
Electronic	IDM	Pierlo	Barbarian
Electronic	Techno	Foniqz	Spectrum (Subdiffusion Mix)
Electronic	Trip-Hop	_ghost	Lullaby
Experimental	Avantgarde	Jared C. Balogh	Equal Value (Ode To A Squirrel)
Experimental	Drone	The Upsidedown	E-Love

Table A.3.: Database CC1 (continued)

Genre	Subgenre	Artist	Title
Experimental	Electroacoustic	Origamibiro	Flicker
Experimental	Field Recording	Aaron Ximm	Spring Rain
Experimental	Musique Concrete	Computer Truck	Euritmix Sux My Dix
Folk		Gaby Cardoso	Mujerzuela
Folk	Psych	The New Mystikal Troubadours	Tonight: A Lonely Century
Folk	Singer-Songwriter	Great Lake Swimmers	Gonna Make It Through This Year
Hip-Hop		Blackwell	I Neva
Instrumental		David Lohstana	Petit talible (instrumental version)
Instrumental		The Kyoto Connection	Hachiko The Faithful Dog (short)
International		Los Amparito	El barzón
Jazz		Kevin MacLeod	AcidJazz
Classical		Mozart by Columbia State University	Symphony No. 40 IV
Classical		Beethoven	Moonlight Sonata (short)
Classical		Beethoven	Für Elise
Pop		Fresh Body Shop	Fireballs
Pop		Lilly Wolf	Jealousy
Pop		On returning	Paris (energy of life)
Rock	Alternative	Blind Violet	Deep
Rock	Garage	Artistes d'Origine in Contrôlée	Reveille toi - Dewey
Rock	Indie	Dumbo Gets Mad	Plumy Tale
Rock	Industrial	Nine Inch Nails	7 Ghosts I
Rock	LoFi	Tyrannic Toy	Blackroad

Table A.3.: Database CC1 (continued)

Genre	Subgenre	Artist	Title
Rock	Metal	Insane Ride	Wrong or Right
Rock	Noise	Lately Kind Of Yeah	Where Is My Jaw?
Rock	Postpunk	Mules	Teenage Freakout
Rock	Postrock	et	Kopeika
Rock	Progressive	Convey	Campaign Speech
Rock	Psych	Flowerheads	06
Rock	Punk	Angstbreaker	Dead Elements
Soul/RnB		Juanitos	Hey

Table A.4.: Database CC2. Contains database CC1

Genre	Subgenre	Artist	Title
Folk		Labib	Saleh Bey
Folk	Singer-Songwriter	Cian Nugent	Double Horse
International		Caligine	Me Piánoune Zaládes
Jazz		Quantum Jazz	If I Can't Dance It's Not My Revolution
Rock	Psych	Noi	Everything Is Changing
Experimental	Avantgarde	Ellen Fullman	Never Gets Out Of Me
Classical		Gio Micheletti	Happy Sinners
Classical		Massimo Mastrangeli	Bagliori di tempesta
Classical		Maya Filipič	Anthos
Classical		Rob Costlow	Bliss
Electronic	Techno	Adrian Sanchez	ADRIAN SANCHEZ G PAL OPHRA
Electronic	Techno	Dj Rostej	Light Rays
Electronic	Techno	-mystery-	Decadence
Electronic	Techno	Saelynh	Summer in Paradise
Electronic	Techno	Synthager	The Way of Atlant

Table A.5.: Database EVAL

Genre	**Subgenre**	**Artist**	**Title**
Rock		Beatles	Hey Jude
Pop		Leann Rimes	How Do I Live
Rock	Hard Rock	Motörhead	Ace Of Spades
Classical		Columbia State University Orchestra	Mozart's Symphony No. 40 IV
Hip-Hop		Blackwell	I Neva
International		Los Amparito	El barzón
Folk	Singer-Songwriter	Great Lake Swimmers	Gonna Make It Through This Year
Electronic	Ambient	LASERS	Amsterdam

A.3. Experiment evaluations

Table A.6.: Runtimes in seconds of models in the Loudness Toolbox [18]

Zwicker [14]	**Zwicker [14] parallelised**	**Moore et al [19]**
45.7	11.6	3290.5
61.1	15.3	4371.4
66.7	17.5	4823.3
38.9	10.4	2747.7
79.2	21.5	5772.3
49.4	13.3	3492.3
81.5	23.3	5848
77.9	21.6	5586.8
93.4	25.6	6844.9
60.2	16.6	4253.4
	Averages:	
65.4	17.67	4703
	Std. deviations:	
17.6	5.2	1309.5

Table A.7.: Davies beat tracker [9] evaluated on database CC1

Title	**CMLc**	**CMLt**	**AMLc**	**AMLt**	**Inf. gain**
Stormy Blues	0	0	0	0	2.26
All I Can Think Of Is You	97.91	97.91	97.91	97.91	2.86
Amazing Grace	99.68	99.68	99.68	99.68	3.07
Amsterdam	0	0	91.94	91.94	2.2
The First Noel / O Come All Ye Faithful	60.54	87.57	60.54	87.57	2.15
Calm The Fuck Down	74.6	94.71	74.6	94.71	2.73
Enthusiast	51.81	93.26	51.81	93.26	3
Illumination	0	0	54.21	99.52	2.49
Simple	4.73	4.73	4.73	4.73	0.76
Barbarian	100	100	100	100	3.45
Spectrum (Subdiffusion Mix)	0	0	53.3	93.39	1.79
Lullaby	0	0	99.8	99.8	2.64
Equal Value (Ode To A Squirrel)	78.26	78.26	78.26	78.26	3.15
E-Love	77.51	86.8	77.51	86.8	2.25
Flicker	3.59	3.59	85.34	87.93	1.68
Spring Rain	55.03	87.83	55.03	87.83	1.64
Euritmix Sux My Dix	99.46	99.46	99.46	99.46	3.54
Mujerzuela	98.88	98.88	98.88	98.88	3.01
Tonight: A Lonely Century	0.85	1.13	31.6	31.6	0.64
Gonna Make It Through This Year	100	100	100	100	3.23
I Neva	2.49	12.77	2.49	12.77	3.42
Petit talible	99.71	99.71	99.71	99.71	3.15
Hachiko The Faithful Dog (short)	0	0	15.2	73.86	1.06
El barzón	0	0	100	100	3.06
AcidJazz	100	100	100	100	2.89
Symphony No. 40 IV	50.58	87.81	50.58	87.81	1.5
Moonlight Sonata (short)	0	0	0.12	0.12	0.8
Für Elise	0.82	3.28	1.57	3.54	0.31
Fireballs	0	0	99.05	99.05	2.26
Jealousy	99.29	99.29	99.29	99.29	3.13

Table A.7.: Davies beat tracker [9] evaluated on database CC1 (continued)

Title	CMLc	CMLt	AMLc	AMLt	Inf. gain
Paris (energy of life)	87.15	87.15	87.15	87.15	2.55
Deep	99.42	99.42	99.42	99.42	2.98
Reveille toi - Dewey	0	0	100	100	1.89
Plumy Tale	99.72	99.72	99.72	99.72	2.97
7 Ghosts I	98.91	98.91	98.91	98.91	3.5
Blackroad	62.58	96.13	62.58	96.13	2.72
Wrong or Right	12.03	18.9	60.69	60.69	1.46
Where Is My Jaw?	99.14	99.14	99.14	99.14	3.03
Teenage Freakout	6.32	6.32	48.88	61.43	1.36
Kopeika	80.21	95.95	80.21	95.95	2.4
Campaign Speech	23.54	32.26	23.54	32.26	0.65
06	0	0	100	100	3.06
Dead Elements	14.29	14.29	56.31	56.31	1.43
Hey	99.69	99.69	99.69	99.69	2.96
Average values:	48.61	54.19	70.43	79.46	2.34
Standard deviations:	43.93	46.12	34.12	31.96	0.89

Table A.8.: Evaluation of [48] on a subset of database CC1

Title	Setting	T	R	M	C	Comments
Stormy Blues	60 0	8	10	10	10	Stereo field location changes
Stormy Blues	60 3	7	2	1	5	
Stormy Blues	60 10	10	10	10	10	
Stormy Blues	600 0	8	7	6	10	Double Hi-Hat sound
Stormy Blues	600 30	10	8	10	10	
Stormy Blues	GT 60 0	9	10	1	10	
Stormy Blues	GT 60 3	10	10	10	10	
Stormy Blues	GT 60 10	8	9	5	7	
Stormy Blues	GT 600 0	1	6	6	1	Jump from mid-song to the silent beginning
Stormy Blues	GT 600 30	4	7	7	4	see above
All I Can Think Of Is You	60 0	7	10	10	8	Small loudness change
All I Can Think Of Is You	60 3	6	10	10	8	see above
All I Can Think Of Is You	60 10	10	10	10	10	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
All I Can Think Of Is You	600 0	9	10	10	10	
All I Can Think Of Is You	600 30	10	10	10	10	
All I Can Think Of Is You	GT 60 0	7	10	10	9	Small loudness change
All I Can Think Of Is You	GT 60 3	6	10	10	10	see above
All I Can Think Of Is You	GT 60 10	9	10	10	10	
All I Can Think Of Is You	GT 600 0	6	10	5	10	Large loudness change
All I Can Think Of Is You	GT 600 30	10	10	10	10	
Amazing Grace	60 0	4	10	10	5	Vocals
Amazing Grace	60 3	5	10	10	2	Vocals
Amazing Grace	60 10	9	10	10	6	
Amazing Grace	600 0	2	9	9	5	
Amazing Grace	600 30	5	9	9	7	
Amazing Grace	GT 60 0	2	10	3	3	Vocals
Amazing Grace	GT 60 3	1	9	2	1	Vocals
Amazing Grace	GT 60 10	9	10	10	6	
Amazing Grace	GT 600 0	2	9	9	5	
Amazing Grace	GT 600 30	5	9	9	7	
Amsterdam	60 0	10	8	6	9	
Amsterdam	60 3	10	8	6	9	
Amsterdam	60 10	10	10	10	10	
Amsterdam	600 0	10	9	8	10	
Amsterdam	600 30	8	10	10	10	
Amsterdam	GT 60 0	8	10	8	10	Snare sound artifact
Amsterdam	GT 60 3	8	10	8	10	see above
Amsterdam	GT 60 10	8	10	10	10	see above
Amsterdam	GT 600 0	7	10	10	10	Jump from fadeout to full loudness
Amsterdam	GT 600 30	7	10	10	10	see above
The First Noel / O Come All Ye Faithful	60 0	7	8	6	5	
see above	60 3	9	3	10	10	
see above	60 10	9	3	10	10	
see above	600 0	8	8	8	3	Many repetitions of the same short segment
see above	600 30	9	10	8	4	
see above	GT 60 0	7	9	6	6	
see above	GT 60 3	10	10	10	10	
see above	GT 60 10	10	10	10	10	
see above	GT 600 0	8	10	8	5	Many repetitions of the same short segment
see above	GT 600 30	8	10	8	5	see above

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Calm The Fuck Down	60 0	10	10	9	8	
Calm The Fuck Down	60 3	10	10	9	10	
Calm The Fuck Down	60 10	10	10	10	10	
Calm The Fuck Down	600 0	7	10	6	7	
Calm The Fuck Down	600 30	10	10	10	10	
Calm The Fuck Down	GT 60 0	10	10	10	10	
Calm The Fuck Down	GT 60 3	10	10	10	10	
Calm The Fuck Down	GT 60 10	10	10	10	10	
Calm The Fuck Down	GT 600 0	8	10	10	6	
Calm The Fuck Down	GT 600 30	9	10	10	8	
Enthusiast	60 0	8	10	10	8	
Enthusiast	60 3	9	10	10	9	
Enthusiast	60 10	10	10	10	10	
Enthusiast	600 0	9	10	10	10	
Enthusiast	600 30	10	10	10	10	
Enthusiast	GT 60 0	8	10	10	10	
Enthusiast	GT 60 3	10	10	5	10	
Enthusiast	GT 60 10	10	10	10	10	
Enthusiast	GT 600 0	5	10	10	10	
Enthusiast	GT 600 30	10	10	10	10	
Illumination	60 0	1	10	2	5	
Illumination	60 3	1	10	2	5	
Illumination	60 10	1	10	2	5	
Illumination	600 0	9	8	8	8	
Illumination	600 30	9	8	8	8	
Illumination	GT 60 0	10	10	8	6	
Illumination	GT 60 3	10	10	1	8	
Illumination	GT 60 10	8	10	1	5	
Illumination	GT 600 0	8	8	10	6	
Illumination	GT 600 30	10	10	10	6	
Simple	60 0	7	5	2	4	
Simple	60 3	7	10	10	8	
Simple	60 10	8	10	10	8	
Simple	600 0	6	10	10	6	Large loudness change
Simple	600 30	8	10	10	9	
Simple	GT 60 0	3	8	5	5	
Simple	GT 60 3	1	7	7	4	
Simple	GT 60 10	1	10	10	6	Large loudness change
Simple	GT 600 0	1	10	3	4	
Simple	GT 600 30	2	10	4	4	
Barbarian	60 0	8	10	7	3	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Barbarian	60 3	9	10	10	8	
Barbarian	60 10	10	10	10	10	
Barbarian	600 0	10	10	10	5	Many repetitions of the same short segment
Barbarian	600 30	10	10	10	5	see above
Barbarian	GT 60 0	10	10	10	8	
Barbarian	GT 60 3	10	10	10	7	
Barbarian	GT 60 10	10	10	10	10	
Barbarian	GT 600 0	10	10	10	5	Many repetitions of the same short segment
Barbarian	GT 600 30	10	10	10	5	see above
Spectrum (Subdiffusion Mix)	60 0	10	10	10	10	
see above	60 3	10	10	10	10	
see above	60 10	10	10	10	10	
see above	600 0	10	10	10	10	
see above	600 30	10	10	10	10	
see above	GT 60 0	10	10	3	8	
see above	GT 60 3	10	10	10	10	
see above	GT 60 10	10	10	10	10	
see above	GT 600 0	10	10	5	10	
see above	GT 600 30	10	9	10	10	
Lullaby	60 0	10	10	10	10	
Lullaby	60 3	10	10	10	10	
Lullaby	60 10	10	10	10	10	
Lullaby	600 0	8	9	10	10	
Lullaby	600 30	9	9	10	10	
Lullaby	GT 60 0	8	10	7	10	
Lullaby	GT 60 3	10	10	10	10	
Lullaby	GT 60 10	10	10	10	10	
Lullaby	GT 600 0	7	9	8	10	Inaccurate beats produce sound artifacts
Lullaby	GT 600 30	8	9	10	10	see above
Equal Value (Ode To A Squirrel)	60 0	10	10	8	7	
see above	60 3	10	10	10	8	
see above	60 10	9	10	10	7	
see above	600 0	10	10	10	4	Many repetitions of the same short segment
see above	600 30	10	10	10	4	see above
see above	GT 60 0	7	10	10	10	
see above	GT 60 3	10	10	10	10	
see above	GT 60 10	10	10	10	10	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
see above	GT 600 0	8	10	6	4	Many repetitions of the same short segment
see above	GT 600 30	10	10	10	4	see above
E-Love	60 0	4	10	10	8	Large loudness change
E-Love	60 3	4	10	10	8	see above
E-Love	60 10	5	10	10	8	see above
E-Love	600 0	8	10	10	3	Vocals
E-Love	600 30	9	10	10	8	
E-Love	GT 60 0	8	7	10	8	
E-Love	GT 60 3	10	10	10	8	
E-Love	GT 60 10	10	10	10	8	
E-Love	GT 600 0	8	10	10	3	Vocals
E-Love	GT 600 30	9	10	10	9	
Flicker	60 0	5	10	10	7	
Flicker	60 3	6	10	10	7	
Flicker	60 10	7	9	10	8	
Flicker	600 0	2	8	10	8	
Flicker	600 30	2	8	10	8	
Flicker	GT 60 0	2	10	10	5	
Flicker	GT 60 3	2	10	10	5	
Flicker	GT 60 10	2	10	10	5	
Flicker	GT 600 0	1	4	3	3	Many repetitions of the same short segment
Flicker	GT 600 30	2	4	3	3	see above
Spring Rain	60 0	1	4	1	6	
Spring Rain	60 3	10	10	10	10	
Spring Rain	60 10	8	10	10	10	
Spring Rain	600 0	8	10	8	10	
Spring Rain	600 30	10	10	10	10	
Spring Rain	GT 60 0	10	10	5	8	
Spring Rain	GT 60 3	10	10	5	8	
Spring Rain	GT 60 10	10	10	10	10	
Spring Rain	GT 600 0	9	10	10	10	
Spring Rain	GT 600 30	8	8	10	10	Inaccurate beats produce sound artifacts
Euritmix Sux My Dix	60 0	9	10	10	8	
Euritmix Sux My Dix	60 3	9	10	3	7	
Euritmix Sux My Dix	60 10	9	10	10	9	
Euritmix Sux My Dix	600 0	7	10	1	5	
Euritmix Sux My Dix	600 30	8	10	1	6	
Euritmix Sux My Dix	GT 60 0	8	9	10	8	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Euritmix Sux My Dix	GT 60 3	9	9	10	8	
Euritmix Sux My Dix	GT 60 10	10	10	10	9	
Euritmix Sux My Dix	GT 600 0	6	10	4	5	
Euritmix Sux My Dix	GT 600 30	7	10	4	7	
Mujerzuela	60 0	9	10	8	10	
Mujerzuela	60 3	10	10	8	10	
Mujerzuela	60 10	10	10	8	10	
Mujerzuela	600 0	10	10	10	10	
Mujerzuela	600 30	8	10	10	10	
Mujerzuela	GT 60 0	9	10	8	10	
Mujerzuela	GT 60 3	10	10	8	10	
Mujerzuela	GT 60 10	10	10	10	10	
Mujerzuela	GT 600 0	8	10	8	10	
Mujerzuela	GT 600 30	9	10	10	10	
Tonight: A Lonely Century	60 0	3	2	1	7	Tempo changes
Tonight: A Lonely Century	60 3	3	2	1	7	
Tonight: A Lonely Century	60 10	3	2	1	7	
Tonight: A Lonely Century	600 0	4	6	8	8	
Tonight: A Lonely Century	600 30	5	8	8	8	
Tonight: A Lonely Century	GT 60 0	3	8	2	1	
Tonight: A Lonely Century	GT 60 3	3	8	2	1	
Tonight: A Lonely Century	GT 60 10	3	8	2	1	
Tonight: A Lonely Century	GT 600 0	3	8	5	4	
Tonight: A Lonely Century	GT 600 30	5	8	5	4	
Gonna Make It Through This Year	60 0	4	9	3	8	
see above	60 3	4	9	5	8	
see above	60 10	4	9	5	8	
see above	600 0	8	10	10	10	
see above	600 30	10	10	10	10	
see above	GT 60 0	4	9	3	8	
see above	GT 60 3	4	9	5	8	
see above	GT 60 10	8	10	7	4	
see above	GT 600 0	10	10	10	10	
see above	GT 600 30	10	10	10	10	
I Neva	60 0	6	10	7	8	
I Neva	60 3	3	10	8	6	Large loudness change
I Neva	60 10	7	10	10	9	
I Neva	600 0	8	10	10	10	Jump from fadeout to full loudness
I Neva	600 30	9	10	10	10	see above

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
I Neva	GT 60 0	4	10	8	6	Large loudness change
I Neva	GT 60 3	6	10	8	6	
I Neva	GT 60 10	6	10	8	6	
I Neva	GT 600 0	6	10	8	6	Vocals
I Neva	GT 600 30	8	10	10	9	Jump from fadeout to full loudness
Petit talible (instrumental version)	60 0	8	10	10	10	
see above	60 3	10	10	10	10	
see above	60 10	10	10	10	10	
see above	600 0	9	10	10	10	
see above	600 30	10	10	10	10	
see above	GT 60 0	10	10	10	10	
see above	GT 60 3	10	10	10	10	
see above	GT 60 10	10	10	10	10	
see above	GT 600 0	10	10	6	10	
see above	GT 600 30	10	10	10	10	
Hachiko The Faithful Dog (short)	60 0	7	10	8	10	
see above	60 3	8	10	8	10	
see above	60 10	9	10	9	8	
see above	600 0	7	10	10	10	Loudness change
see above	600 30	8	10	10	10	see above
see above	GT 60 0	9	10	10	9	
see above	GT 60 3	9	10	10	9	
see above	GT 60 10	7	10	10	8	Loudness change
see above	GT 600 0	4	10	8	6	see above
see above	GT 600 30	7	10	10	8	see above
El barzón	60 0	10	10	10	10	
El barzón	60 3	10	10	10	10	
El barzón	60 10	10	10	10	10	
El barzón	600 0	10	10	10	6	Many repetitions of the same short segment
El barzón	600 30	10	10	10	6	Many repetitions of the same short segment
El barzón	GT 60 0	10	10	10	10	
El barzón	GT 60 3	10	10	10	10	
El barzón	GT 60 10	10	10	10	10	
El barzón	GT 600 0	7	10	8	6	Many repetitions of the same short segment

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	Setting	T	R	M	C	Comments
El barzón	GT 600 30	10	10	10	6	Many repetitions of the same short segment
AcidJazz	60 0	10	10	5	8	
AcidJazz	60 3	10	10	10	9	
AcidJazz	60 10	10	10	10	10	
AcidJazz	600 0	6	9	5	8	
AcidJazz	600 30	8	10	10	9	
AcidJazz	GT 60 0	10	10	5	8	
AcidJazz	GT 60 3	10	10	10	9	
AcidJazz	GT 60 10	10	10	10	10	
AcidJazz	GT 600 0	6	9	5	8	
AcidJazz	GT 600 30	8	10	10	9	
Symphony No. 40 IV	60 0	9	10	9	9	
Symphony No. 40 IV	60 3	10	10	10	10	
Symphony No. 40 IV	60 10	10	10	10	10	
Symphony No. 40 IV	600 0	4	8	6	7	
Symphony No. 40 IV	600 30	10	10	10	8	
Symphony No. 40 IV	GT 60 0	10	10	10	9	
Symphony No. 40 IV	GT 60 3	10	10	10	10	
Symphony No. 40 IV	GT 60 10	10	10	10	10	
Symphony No. 40 IV	GT 600 0	3	10	3	9	
Symphony No. 40 IV	GT 600 30	9	10	10	9	
Fireballs	60 0	8	10	10	6	
Fireballs	60 3	9	10	10	10	
Fireballs	60 10	10	10	10	10	
Fireballs	600 0	6	10	7	9	
Fireballs	600 30	10	10	10	10	
Fireballs	GT 60 0	6	10	6	9	
Fireballs	GT 60 3	10	10	10	9	
Fireballs	GT 60 10	10	10	10	10	
Fireballs	GT 600 0	9	10	8	9	
Fireballs	GT 600 30	10	10	10	10	
Deep	60 0	9	10	10	10	
Deep	60 3	10	10	10	10	
Deep	60 10	10	10	10	10	
Deep	600 0	9	10	10	8	
Deep	600 30	10	10	10	8	
Deep	GT 60 0	10	10	3	8	
Deep	GT 60 3	10	10	10	10	
Deep	GT 60 10	10	10	10	10	
Deep	GT 600 0	10	10	5	7	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Deep	GT 600 30	10	10	10	10	
Reveille toi - Dewey	60 0	8	10	7	6	
Reveille toi - Dewey	60 3	6	10	8	9	
Reveille toi - Dewey	60 10	6	10	8	10	
Reveille toi - Dewey	600 0	9	10	10	10	
Reveille toi - Dewey	600 30	10	10	10	10	
Reveille toi - Dewey	GT 60 0	6	10	6	9	
Reveille toi - Dewey	GT 60 3	8	10	6	7	
Reveille toi - Dewey	GT 60 10	9	10	6	6	
Reveille toi - Dewey	GT 600 0	7	10	10	9	
Reveille toi - Dewey	GT 600 30	8	10	10	10	
Plumy Tale	60 0	6	10	8	6	
Plumy Tale	60 3	4	10	10	5	
Plumy Tale	60 10	10	10	10	10	
Plumy Tale	600 0	9	10	10	10	
Plumy Tale	600 30	10	10	10	10	
Plumy Tale	GT 60 0	5	10	10	5	
Plumy Tale	GT 60 3	6	10	10	6	
Plumy Tale	GT 60 10	10	10	10	10	
Plumy Tale	GT 600 0	7	10	8	10	
Plumy Tale	GT 600 30	10	10	10	10	
7 Ghosts I	60 0	10	10	7	10	
7 Ghosts I	60 3	10	10	10	10	
7 Ghosts I	60 10	10	10	10	10	
7 Ghosts I	600 0	9	10	9	5	Many repetitions of the same short segment
7 Ghosts I	600 30	10	10	10	5	see above
7 Ghosts I	GT 60 0	8	10	3	10	
7 Ghosts I	GT 60 3	9	9	10	10	
7 Ghosts I	GT 60 10	9	9	10	10	
7 Ghosts I	GT 600 0	9	10	9	5	Many repetitions of the same short segment
7 Ghosts I	GT 600 30	10	10	10	5	see above
Blackroad	60 0	9	10	7	10	Loudness change
Blackroad	60 3	9	10	10	10	see above
Blackroad	60 10	10	10	10	10	see above
Blackroad	600 0	7	10	7	8	Stereo field location changes
Blackroad	600 30	8	10	10	9	see above
Blackroad	GT 60 0	7	10	3	10	Loudness change
Blackroad	GT 60 3	6	10	10	10	see above

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Blackroad	GT 60 10	7	10	10	10	see above
Blackroad	GT 600 0	7	10	8	10	Stereo field location changes
Blackroad	GT 600 30	8	10	10	10	see above
Wrong or Right	60 0	7	10	4	3	
Wrong or Right	60 3	8	4	8	9	
Wrong or Right	60 10	7	4	5	3	Very inaccurate beat positions
Wrong or Right	600 0	8	6	8	8	
Wrong or Right	600 30	8	6	10	9	
Wrong or Right	GT 60 0	10	10	8	6	
Wrong or Right	GT 60 3	10	10	8	6	
Wrong or Right	GT 60 10	10	10	10	10	
Wrong or Right	GT 600 0	10	10	10	10	
Wrong or Right	GT 600 30	10	10	10	10	
Where Is My Jaw?	60 0	4	10	8	6	Jumps only at the end
Where Is My Jaw?	60 3	6	10	10	7	see above
Where Is My Jaw?	60 10	6	10	10	7	see above
Where Is My Jaw?	600 0	9	10	8	4	Many repetitions of the same short segment
Where Is My Jaw?	600 30	10	10	10	4	see above
Where Is My Jaw?	GT 60 0	4	10	8	6	Jumps only at the end
Where Is My Jaw?	GT 60 3	6	10	10	7	see above
Where Is My Jaw?	GT 60 10	6	10	10	7	see above
Where Is My Jaw?	GT 600 0	8	10	8	4	Many repetitions of the same short segment
Where Is My Jaw?	GT 600 30	10	10	10	4	see above
Teenage Freakout	60 0	8	4	6	6	Tempo changes
Teenage Freakout	60 3	8	4	6	7	see above
Teenage Freakout	60 10	8	4	6	7	see above
Teenage Freakout	600 0	6	1	1	4	see above
Teenage Freakout	600 30	3	1	1	4	see above
Teenage Freakout	GT 60 0	8	4	6	6	see above
Teenage Freakout	GT 60 3	8	5	6	7	see above
Teenage Freakout	GT 60 10	10	7	10	10	Tempo slightly changes
Teenage Freakout	GT 600 0	7	9	6	5	
Teenage Freakout	GT 600 30	10	10	10	4	Many repetitions of the same short segment
Kopeika	60 0	1	6	1	5	
Kopeika	60 3	2	6	1	5	
Kopeika	60 10	4	6	1	7	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Kopeika	600 0	9	10	8	8	
Kopeika	600 30	9	10	10	8	
Kopeika	GT 60 0	7	10	5	8	
Kopeika	GT 60 3	5	10	4	7	
Kopeika	GT 60 10	5	10	10	7	
Kopeika	GT 600 0	10	10	7	10	
Kopeika	GT 600 30	8	10	10	10	Loudness change
Campaign Speech	60 0	7	1	1	7	
Campaign Speech	60 3	8	1	1	7	
Campaign Speech	60 10	9	1	1	8	Tempo changes
Campaign Speech	600 0	5	1	1	9	
Campaign Speech	600 30	10	10	10	10	
Campaign Speech	GT 60 0	2	9	3	8	Tempo changes
Campaign Speech	GT 60 3	1	9	3	9	see above
Campaign Speech	GT 60 10	1	9	3	9	see above
Campaign Speech	GT 600 0	9	10	10	8	Snare sound artifact
Campaign Speech	GT 600 30	9	10	10	9	
06	60 0	9	10	10	10	
06	60 3	10	10	10	10	
06	60 10	10	10	10	10	
06	600 0	10	10	10	8	
06	600 30	10	10	10	8	
06	GT 60 0	10	10	10	6	Repetitions of the same short segment
06	GT 60 3	10	10	10	10	
06	GT 60 10	10	10	10	10	
06	GT 600 0	10	10	8	9	
06	GT 600 30	10	10	10	9	
Dead Elements	60 0	2	1	1	6	
Dead Elements	60 3	1	1	1	6	
Dead Elements	60 10	1	1	1	6	
Dead Elements	600 0	1	1	1	3	
Dead Elements	600 30	1	1	1	3	
Dead Elements	GT 60 0	4	8	1	5	Tempo changes
Dead Elements	GT 60 3	6	10	4	4	
Dead Elements	GT 60 10	6	9	4	4	
Dead Elements	GT 600 0	2	10	2	6	
Dead Elements	GT 600 30	6	10	4	6	
Hey	60 0	10	10	10	10	
Hey	60 3	10	10	10	10	
Hey	60 10	10	10	10	10	

Table A.8.: Evaluation of [48] on a subset of database CC1 (continued)

Title	**Setting**	**T**	**R**	**M**	**C**	**Comments**
Hey	600 0	10	10	8	10	
Hey	600 30	10	10	10	10	
Hey	GT 60 0	10	10	10	10	
Hey	GT 60 3	10	10	10	10	
Hey	GT 60 10	10	10	10	10	
Hey	GT 600 0	10	10	8	10	
Hey	GT 600 30	10	10	10	10	
Average values:		7.8	9.2	8	8	
Standard deviation:		2.7	2	2.9	2.3	

List of Figures

List of Algorithms

Bibliography

[1] Audacity team: *Crossfade types.* `http://manual.audacityteam.org/o/man/crossfade_tracks.html`. – Accessed: 2015-08-31

[2] Avidan, S.; Shamir, A.: Seam carving for content-aware image resizing. In: *ACM Transactions on graphics (TOG)* Vol. 26, 2007, p. 10

[3] Boris, S.: *The Boost C++ Libraries.* XML Press, 2011

[4] Caplin, W. E.: *Classical form: A theory of formal functions for the instrumental music of Haydn, Mozart, and Beethoven.* Oxford University Press, 1998

[5] Christensen, T.: *The Cambridge history of Western music theory.* Cambridge University Press, 2002

[6] Davies, M. E. P.: *Beat Tracker Implementation in MATLAB.* `https://code.soundsoftware.ac.uk/projects/davies-beat-tracker`. – Accessed: 2015-08-31

[7] Davies, M. E. P.: *Beat Tracking Evaluation Toolbox in MATLAB.* `https://code.soundsoftware.ac.uk/projects/beat-evaluation`. – Accessed: 2015-08-31

[8] Davies, M. E. P.; Degara, N.; Plumbley, M. D.: Evaluation methods for musical audio beat tracking algorithms. In: *Queen Mary University of London, Centre for Digital Music, Tech. Rep. C4DM-TR-09-06* (2009)

[9] Davies, M. E. P.; Plumbley, M. D.: Context-Dependent Beat Tracking of Musical Audio. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* 15 (2007), No. 3, p. 1009–1020

[10] Davis, G.; Davis, G. D.: *The sound reinforcement handbook.* Ch. 12, p. 201–203, Hal Leonard Corporation, 1989

[11] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* 1 (1959), p. 269–271

[12] Driedger, J.; Müller, M.: TSM Toolbox: MATLAB Implementations of Time-Scale Modification Algorithms. In: *Proceedings of the International Conference on Digital Audio Effects (DAFx).* Erlangen, Germany, 2014, p. 249–256

[13] Everest, F. A.; Pohlmann, K.: *Master Handbook of Acoustics.* McGraw-Hill Education, 2009

[14] Fastl, H.; Zwicker, E.: *Psychoacoustics: Facts and models.* Vol. 22. Springer Science & Business Media, 2007

[15] Fletcher, H.; Munson, W. A.: Relation between loudness and masking. In: *The Journal of the Acoustical Society of America* 9 (1937), No. 1, p. 78

[16] Foote, J. T.; Cooper, M. L.: Media segmentation using self-similarity decomposition. In: *Electronic Imaging 2003*, International Society for Optics and Photonics, 2003, p. 167–175

[17] Fredman, M. L.; Tarjan, R. E.: Fibonacci heaps and their uses in improved network optimization algorithms. In: *Journal of the ACM (JACM)* 34 (1987), No. 3, p. 596–615

[18] GENESIS: *MATLAB Loudness Toolbox.* `http://genesis-acoustics.com/en/loudness_online-32.html`. – Accessed: 2015-08-31

[19] Glasberg, B. R.; Moore, B. C. J.: A model of loudness applicable to time-varying sounds. In: *Journal of the Audio Engineering Society* 50 (2002), No. 5, p. 331–342

[20] Hart, P. E.; Nilsson, N. J.; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems Science and Cybernetics* 4 (1968), No. 2, p. 100–107

[21] ISO: Acoustics - Normal equal-loudness-level contours / International Organization for Standardization. Geneva, Switzerland, 2003 (226:2003). – Research paper

[22] Lartillot, O.; Toiviainen, P.; Eerola, T.: A Matlab Toolbox for Music Information Retrieval. In: Preisach, C. (Ed.); Burkhardt, H. (Ed.);

Schmidt-Thieme, L. (Ed.); Decker, Reinhold (Ed.): *Data Analysis, Machine Learning and Applications*. Springer Berlin Heidelberg, 2008 (Studies in Classification, Data Analysis, and Knowledge Organization), p. 261–268

[23] Mathworks: *Cross-Correlation function "xcorr" in MATLAB*. `http://de.mathworks.com/help/signal/ref/xcorr.html#inputarg_scaleopt`. – Accessed: 2015-08-31

[24] Mathworks: *MATLAB Parallel Computing Toolbox*. `http://mathworks.com/products/parallel-computing/`. – Accessed: 2015-08-31

[25] Mathworks: *MATLAB programming language*. `http://www.mathworks.com/products/matlab/`. – Accessed: 2015-08-31

[26] Mathworks: *MATLAB Signal Processing Toolbox*. `http://mathworks.com/products/signal/`. – Accessed: 2015-08-31

[27] Matthias H., U.: *Graph Demo - a Matlab GUI to explore similarity graphs and their use in machine learning*. `http://www.ml.uni-saarland.de/code/GraphDemo/GraphDemo.htm`. – Accessed: 2015-08-31

[28] McKinney, M. F.; Moelants, D.; Davies, M. E. P.; Klapuri, A.: Evaluation of audio beat tracking and music tempo extraction algorithms. In: *Journal of New Music Research* 36 (2007), No. 1, p. 1–16

[29] Moore, B. C. J.; Glasberg, B. R.; Baer, T.: A model for the prediction of thresholds, loudness, and partial loudness. In: *Journal of the Audio Engineering Society* 45 (1997), No. 4, p. 224–240

[30] Olson, H. F.: The measurement of loudness. In: *Audio Magazine* (1972), February, p. 18–22

[31] Open Music Theory Textbook: *Form in pop/rock music*. `http://openmusictheory.com/popRockForm.html`. – Accessed: 2015-08-31

[32] Papadimitriou, C. H.; Steiglitz, K.: *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998

[33] Parker, J.R.; Behm, B.: Creating audio textures by example: tiling and stitching. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* Vol. 4, 2004, p. 317–320

[34] R Core Team: *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing (Session), 2015. – URL `http://www.R-project.org/`. – Accessed: 2015-08-31

[35] ROLI Ltd.: *JUCE cross-platform C++ library.* `http://www.juce.com/`. – Accessed: 2015-08-31

[36] Rousseeuw, P. J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. In: *Journal of computational and applied mathematics* 20 (1987), p. 53–65

[37] Rowland, D.: *dRowAudio - A JUCE module for high level audio application development.* `http://drowaudio.co.uk/docs/`. – Accessed: 2015-08-31

[38] Schmidt-Jones, C.: *Time Signature. OpenStax CNX.* `http://cnx.org/contents/68100121-0efa-4bd8-a0ca-3336e8d01a10@16`. – Accessed: 2015-08-31

[39] Sengpiel, E.: *The human perception of loudness.* `http://www.sengpielaudio.com/calculator-loudness.htm`. – Accessed: 2015-08-31

[40] Shi, J.; Malik, J.: Normalized cuts and image segmentation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), No. 8, p. 888–905

[41] Siegel, S.; Castellan, N.J.: *Nonparametric statistics for the behavioral sciences.* Second edition. McGraw–Hill, Inc., 1988

[42] Tauscher, J.; Wenger, S.; Magnor, M.: Audio Resynthesis on the Dancefloor: A Music Structural Approach. In: *Proceedings of Vision, Modeling and Visualization (VMV)*, 2013, p. 8

[43] Verhelst, W.; Roelands, M.: An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* Vol. 2, 1993, p. 554–557

[44] Webster, P.; Jiříček, O.: A Brief Comparison of Loudness Evaluation Methods. In: *Acoustic Sheets, Department of Physics, Czech Technical University in Prague* 20 (2014), No. 2

[45] Weisstein, E. W.: *Cross-Correlation.* `http://mathworld.wolfram.com/Cross-Correlation.html`. – Accessed: 2015-08-31

[46] Wenger, S.; Magnor, M.: Constrained Example-Based Audio Synthesis. In: *Proceedings of the International Conference on Multimedia and Expo (ICME)*, 2011, p. 6

[47] Wenger, S.; Magnor, M.: A Genetic Algorithm for Audio Retargeting. In: *Proceedings of ACM Multimedia (ACMMM)*, 2012, p. 705–708

[48] Wenner, S.: *Music Retargeting and Synthesis*, Swiss Federal Institute of Technology Zurich, Diploma thesis, 2012

[49] WFMU: *Free Music Archive.* `http://freemusicarchive.org/`. – Accessed: 2015-08-31

[50] Zapata, J.; Davies, M. E. P.; Gómez, E.: Multi-feature Beat Tracking. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* 22 (2014), No. 4, p. 816–825

[51] Zar, J. H.: Significance testing of the Spearman rank correlation coefficient. In: *Journal of the American Statistical Association* 67 (1972), No. 339, p. 578–580

[52] Zheng, F.; Zhang, Gu.; Song, Z.: Comparison of different implementations of MFCC. In: *Journal of Computer Science and Technology* 16 (2001), No. 6, p. 582–589

[53] Zwicker, E.; Fastl, H.; Widmann, U.; Kurakata, K.; Kuwano, S.; Namba, S.: Program for calculating loudness according to DIN 45631 (ISO 532B). In: *Journal of the Acoustical Society of Japan (E)* 12 (1991), No. 1, p. 39–42

Index