Inferring Score Level Musical Information From Low-Level Musical Data

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades Dr. rer. nat.

vorgelegt von Dipl.-Informatiker Jürgen Kilian
geboren in Bensheim

Tag der Einreichung: 9. August 2004
Tag der Disputation: 8. Oktober 2004

Referenten:
Prof. Dr. Hermann K.-G. Walter, Darmstadt
Prof. Dr. Holger H. Hoos, Vancouver, Kanada

Darmstädter Dissertationen D17

Abstract

The task of inferring score level musical information from low-level musical data can be accomplished by human listeners – depending on their training – almost intuitively, but an algorithmic model for computer aided transcription is very hard to achieve. In between there exist a large number of approaches addressing different issues related to the process of musical transcription. Especially for the two core issues in the context of transcription, *i.e.,* tempo detection and quantisation, still no general, adequate, and easy-to-use approach seems to be available. Many of the approaches described in the literature have been implemented only in a prototypical way or are restricted to certain styles of music or input data.

This thesis gives an introduction to the general issue of computer aided transcription and describes related approaches known from literature as well as new approaches developed in the context of this thesis. It also describes the implementation and evaluation of these new models by the implementation of a complete system for inferring score level information from low-level symbolic musical data as an usable transcription tool. The described system consists of several modules each addressing specific issues in the context of musical transcription. For each module the thesis includes a discussion of related work known from literature as well as a description of the specific implemented approaches. For the two main tasks during transcription – tempo detection and quantisation – new pattern-based approaches have been developed and implemented. Beside these main issues the thesis addresses also approaches for voice separation, segmentation and structural analysis, inferring of key- and time signature, pitch spelling, and the detection of musical ornaments. Also approaches for inferring other secondary score elements such as slurs, staccato, and intensity marking are discussed.

A general intention behind the here developed approaches is adequacy in the sense that somehow simple input data should be processed automatically but for more complex input data the system might ask the user for additional information. Where other approaches try to infer always *the* correct transcription, the here described system was built under the assumption, that a correct, fully automatic algorithmical transcription is not possible for all cases of input data. Therefore the here described system analyses the input and output data for certain features and might ask the user for additional information or it might create warnings if potential errors are detected.

Because the processing of audio files for the detection of the start- and end-positions of notes, and their pitch and intensity information is a complex, challenging task on its own, the here described system uses low-level symbolic data – consisting of explicit note objects with pitch, intensity and absolute timing information – as input. Different from most other approaches in this area the system uses the Guido Music Notation format as file format for the inferred output data (*i.e.,* musical scores). Where other file formats (*e.g.,* MIDI) are not able to represent all types of high-level score information, they cannot be converted into graphical scores (*e.g.,* ASCII note lists), or it becomes a complex task to create them (*e.g.,* proprietary binary formats), Guido Music Notation can be created algorithmically in a straight forward way. It also offers the advantage that it is human readable, that it can be converted into graphical scores by using existing tools, and that it can represent all score level information required for conventional music notation and beyond.

Zusammenfassung

Die hier vorgelegte Arbeit mit dem Titel 'Inferring Score Level Musical Information From Low-Level Musical Data' beschäftigt sich mit verschiedenen Problemstellungen im Bereich der computergestützten Transkription von Musik. Ziel ist es, Modelle zu finden, um musikalische Daten von einer symbolischen, audio-nahen Darstellung in eine Partitur-Darstellung zu konvertieren.

Die Arbeit besteht aus verschiedenen Kapiteln, wobei sich (mit Ausnahme von Prologue und Epilogue) jedes Kapitel mit verschiedenen Aspekten und Teilaufgaben eines Teilgebietes der computergestützten Transkription befasst. Bei der Entwicklung von Modellen und Algorithmen für die verschiedenen Teilbereiche wurde auf drei Aspekte wert gelegt: Adäquatheit, Flexibilität und Interaktivität. Diese sind so zu verstehen, dass ein System automatisch die Komplexität der Eingabe Dateien erkennen soll und *einfache* Daten auch automatisch verarbeiten kann. Im Falle von komplexeren Eingabedaten, sollte ein in diesem Sinne adäquates System aber eventuell jeweils benötigte zusätzliche Informationen beim Benutzer anfragen können (Interaktivität). Ausserdem sollen die hier vorgestellten Modelle nicht auf bestimmte Formen von musikalischen Daten (z. B. nur monophon, bestimmter Stil) begrenzt sein (Flexibilität). Insbesondere Adäquatheit (im beschriebenen Sinn) und Interaktivität wurden in den aus der Literatur zum Thema Transkription bekannten Arbeiten bisher kaum bzw. gar nicht diskutiert.

Prologue

Der Prozess der Komposition, Aufführung, Wahrnehmung und Analyse von Musik kann im weitesten Sinne als eine Art 'Spiel' zwischen Komponist bzw. Musiker und Zuhörer betrachtet werden. Auf der einen Seite versuchen Komponist und Musiker (letzterer vor allem bei improvisierter Musik), ihre 'Ideen' in musikalische Strukturen zu *kodieren*, auf der anderen Seite versuchen die Zuhörer, die Struktur der Musik zu entschlüsseln und die Ideen des Komponisten zu verstehen.
Beide Seiten dieses hypothetischen Spiels bestehen aus einer Hierarchie von verschiedenen Prozessen. Die kompositorische Seite kann hierbei als top-down Ansatz beginnend bei der musikalischer Idee, über Partitur, Aufführung, bis hin zum Audio Signal betrachtet werden. Die Zuhörerseite kann als bottom-up Vorgang verstanden werden: Vom Audio Signal, über die Wahrnehmung von Grundelementen (wie z. B. Tempo und Lautstärke), Transkription einer Partitur, bis hin zum Verständnis der tieferen Struktur und Idee des Komponisten.

Für nahezu alle Teilprozesse, sowohl auf der kompositorischen als auch auf der Zuhörer Seite wurden inzwischen algorithmische Modelle entwickelt. Die vorliegende Arbeit beschäftigt sich mit einem Teilbereich auf der Zuhörerseite, der eine Stufe über den reinen Audio Daten beginnt, und mit der Transkription einer Partitur endet. Die Konvertierung von reinen Audio Signalen in eine einfache symbolische Darstellung, bestehend aus einer Liste von Noten mit notationsbezogener Tonhöhe, Lautstärke, und absolutem Anfangs- und Endzeitpunkt (in Sekunden) wird hier nicht untersucht.

1 Computer Aided Transcription

Der Prozess der Transkription von musikalischen Daten in eine Partiturdarstellung besteht aus verschiedenen Teilprozessen: Pre-Processing, Stimmentrennnung, Erkennung von Ornamenten, Tempoerkennung, Quantisierung, sowie der Erkennung von weiteren sekundären Partitur-Informationen. Ein Kernproblem besteht hierbei darin, die in absoluten Einheiten (Sekunden) gegebenen Zeitinformation der Noten (Anfangs- und Endzeitpunk), in den diskreten Wertebereich der in Partituren darstellbaren Tondauern zu konvertieren. Hierbei ist davon auszugehen, dass die gegebenen Zeitinformation erstens spielbedingte Ungenauigkeiten enthalten und zweitens, bedingt durch verschiedene Arten von Tempoänderungen, gleiche absolute Tondauern auch unterschiedlichen Partiturtondauern entsprechen können. D. h. eine einfache Rundungsoperation ist hier nicht ausreichend.

2 Voice Separation

Polyphone Eingabedaten müssen, um sie in einer Partitur korrekt darstellen zu können, in verschiedene musikalische *Stimmen* verteilt werden und/oder zu Akkorden zusammengefasst werden. Bei der Zuordnung von Noten eines Stückes zu verschiedenen Stimmen, gibt es in der Regel immer mehrere korrekte Lösungen, wobei es von den Wünschen des Benutzers abhängt, welche Lösung bevorzugt werden soll (z. B. viele Stimmen mit wenigen Akkorden oder wenige Stimmen mit vielen Akkorden). In der Regel ist die Anzahl der gleichzeitig klingenden Noten nicht konstant, so dass diese nicht durch ein einfaches Verteilverfahren einer festen Anzahl von Stimmen zugeordnet werden können.

Im Rahmen der Arbeit wurde ein heuristisches Verfahren zur Stimmentrennung entwickelt, welches mithilfe eines lokalen Suchverfahrens und einer Zielfunktion eine entsprechend dieser Funktion optimale Stimmentrennung bestimmt.

3 Similarity And Segmentation

In diesem Kapitel werden Grundlagen und existierende Arbeiten im Bereich Struktur- und Ähnlichkeitsanalyse von Musik beschrieben. Im Rahmen der Arbeit wurde der in der Bioinformatik bekannte und dort als Standard-Werkzeug eingesetzte BLAST Algorithmus für die Verarbeitung von musikalischen Daten adaptiert. Dieser MusicBLAST Algorithmus erlaubt eine effiziente Suche nach ähnlichen oder sich wiederholenden Strukturen in einem einzelnen bzw. zwischen zwei verschiedenen Musikstücken. Das benutzte Ähnlichkeitsmaös kann frei definiert werden. Je nach Einsatzzweck werden hier z. B. eher melodische oder eher rhythmische Gesichtspunkte von Interesse sein.

4 Tempo Detection

Ein Kernproblem für alle Transkriptions Systeme ist Tempoerkennung. Während es selbst für ungeübte Zuhörer relativ leicht ist, dem Tempo eines Stückes zu folgen (z. B. durch Klatschen), ist es nach wie vor schwierig dieses Verhalten algorithmisch zu modellieren. Im Gegensatz zu Ansätzen zur Erkennung des Grundrhythmus eines Stückes, liegt der Schwerpunkt der vorliegenden Arbeit darin, für alle Noten eines Stückes eine korrekte Partitur-Tondauer zu bestimmen. Da es hier von Vorteil ist, die Partitur-Tondauer einer Note in Abhängigkeit vom rhythmischen Kontext zu bestimmen, wurde ein Muster basiertes Modell entwickelt, welches es erlaubt, bevorzugte bzw. häufig auftretende Rhythmusmuster aus einer Musterdatenbank zur Tempoerkennung zu benutzen. Desweiteren wird ein Verfahren zur kontextsensitiven Bestimmung von Partitur-Tondauern vorgestellt, welches die Häufigkeit der Auswahl einer Tondauerklasse in der lokalen Vergangenheit einer Note berücksichtigt. Dieses Verfahren wird für alle Bereiche eines Musikstückes benutzt, für die in einer Datenbank keine entsprechenden Muster gefunden werden können.

Bestimmte Fehler (z. B. falsch erkannte Tondauern, falsche Positionen für Anfangszeitpunkte von Noten), welche unter Umständen während der Tempoerkennung auftreten, können automatisch von einem geeigneten Algorithmus erkannt werden, aber nicht unbedingt deren Ursache. Da im Allgemeinen davon auszugehen ist, dass sich solche Fehler nicht vermeiden lassen, ist das hier beschriebene Verfahren zur Tempoerkennung so ausgelegt, dass in solchen Fällen eine Warnung an den Benutzer ausgegeben wird, und dieser dann die Möglichkeit hat, manuell die Ursache des Fehlers zu korrigieren.

Am Ende dieses Kapitels wird eine Methode beschrieben, um direkt aus den Eingabedaten, mittels statistischer Analyse eine Güte für die Spiel-Genauigkeit der Eingabedaten abzuleiten. Diese Güte wird während der Tempoerkennung, der nachfolgenden Quantisierung, sowie in anderen Modulen genutzt um z. B. die Größe von Suchfenstern bzw. Tondauerrastern adaptiv einzugrenzen. Die Grundannahme hierbei ist, dass es nicht sinnvoll wäre, extrem ungenau gespielten Stücken mit äusserster Genauigkeit zu transkribieren.

5 Quantisation

Bei der Quantisierung von musikalischer Daten geht es darum, ungenaue Partitur-Zeitinformation aus einem stetigen Wertebereich in grafisch darstellbare Zeiten eines quasi diskreten Wertebereichs zu konvertieren. Hierbei ist im allgemeinen Fall davon auszugehen, dass die beobachteten Tondauern auch Ungenauigkeiten enthalten. Ähnlich wie die Termpoerkennung, ist auch dieser Prozess wieder kontextabhängig und kann daher nicht als einfaches *Rundungsverfahren* modelliert werden. Insbesondere ist zu berücksichtigen, dass auch hier – abhängig vom Kontext – beobachtete Tondauern gleicher Länge, durch die Quantisierung in verschiedene Partitur-Tondauern konvertiert werden können. Bei der Quantisierung werden daher auch die beiden für die Tempoerkennung benutzten Verfahren (Analyse der Häufigkeit und Vergleich mit einer Musterdatenbank) eingesetzt.

6 Secondary Score Elements

Neben den musikalischen Basisinformationen (Tonhöhe, Tondauer) gibt es noch weitere Elemente einer Partitur, welche für ihre Les- und Spielbarkeit relevant sind. Hierzu gehören z. B. Taktart, Tonart, korrekte Vorzeichen, Ornamente, oder auch Bindebögen. In diesem Kapitel werden hierzu verschiedene Verfahren aus der Literatur vorgestellt, sowie die hierzu im Rahmen der Arbeit entwickelten und implementierten Ansätze erläutert.

Die Erkennung von musikalischen Ornamenten (z. B. Triller, Vorschläge) kann die Qualität anderer Funktionen wie Tempoerkennung und Quantisierung entscheidend verbessern. Das Filtern von diesen, in der Regeln kurzen, mit expliziten Tondauern nur schwer darstellbaren Noten, entspricht im weitesten Sinne einer Rauschunterdrückung. Während die Ornament-Noten von den eigentlichen Melodie-Noten durch eine statistische Analyse der Tonlängen getrennt werden, wird zur Bestimmung des eigentlichen Ornamettyps ein k-Nearest-Neighbour Verfahren eingesetzt.

Epilogue

Im letzten Kapitel der Arbeit werden noch einmal die Grundprinzipien (Interaktivität, Flexibilität, Adäquatheit) und Besonderheiten (z. B. Musterbasierte Verfahren) der hier entwickelten Ansätze und ihre Implementierung als Kommandozeilen Werkzeug `midi2gmn` bzw. Internet Dienst[1] zusammengefasst. Im Gegensatz zu vielen anderen in der Literatur beschriebenen Ansätzen wurde im Rahmen dieser Arbeit auch ein System implementiert, welches einfach zu benutzen ist und welches Ausgaben auf Partiturebene erzeugt, die mittels entsprechender GUIDO Werkzeuge direkt in Notengrafiken konvertiert werden können. Weiterhin beinhaltet dieses Kapitel einen Ausblick und Vorschläge, welche Aspekte bei der Entwicklung von zukünfigen Transkriptions-Systemen besonders beachtet werden sollten.

[1] Unter `http:\\www.noteserver.org/midi2gmn/midi2gmn.html`.

Contents

Prologue

"Die Musik drückt das aus, was nicht gesagt werden kann und worüber es unmöglich ist, zu schweigen." **Victor Hugo**

When humans listen to music, any kind of reaction to the perceived audio signal – mental or physical, like for example, clapping hands or dancing – might be interpreted as a kind of subconscious musical analysis. Depending on the knowledge, skills, training, and background information of the listener, this analysis might be rather simple (*e.g.,* only categorising the music into good or bad) or more complex, such as the recognition of rhythmic features and relative or absolute pitch classes. In the musical domain these process of converting perceived music from the audio level to the structural and especially score level is called *transcription*.

Listening to music might be viewed as a rather passive task, but it might also be interpreted as the inverse task to composition, musical arranging, or performing. This interpretation can be seen as a generalisation of Goto's interpretation of beat tracking as the inverse of the tasks of performing, sound production, and sound transmission [Got01].

The process of composing music and listening to music can be somehow viewed as a game where the composer tries to encode rhythmic and melodic structure in a way that it becomes challenging – but not impossible - for an human listener to decode this structure. The compositional part of this hypothetical game can be seen as a top-down process starting with the vague notion of a compositional idea (or an invention), continuing with symbolic and structural representations, interpretation by a performer, down to the pure audio signal.
The listener part of this game becomes then a bottom-up process starting with raw audio input, continuing with the perception of timbre and instruments, with perception of pitch and rhythm, followed by the perception of structure on different levels, up to the creation of a score or another high level representation of music, and finally the decoding and understanding of the composer's intention.

> "Listening to a concert, I often find myself unexpectedly in a foreign country, not knowing how I got there: a modulation had occurred which escaped my comprehension. I am sure that this would not have happened to me in former times, when a performer's education did not differ from a composer's." (A. Schönberg as cited in [Tod85])

Supported by the increase of computational power during the last 40 years, a large number of approaches for supporting, simulating, and imitating the composition and listening processes by algorithms have been developed. A part of these approaches have been built with a dedicated focus on exploring the way the human perception works; others focus on using general or specific existing computational models for creating a best possible result.

On both sides of the described musical game and on every level of the top-down and bottom-up processes single tasks or groups of consecutive sub-tasks might be replaced – with more or less success – by algorithmic models. For example, on the compositional part there exist approaches for algorithmic composition on symbolic level, which still require the intention and control of an human composer and also human performers. There also exist approaches which try to replace only the performer by imitating, for example, the expressive playing of highly trained pianists (*e.g.,* [Wid95]). Other approaches try to replace the physical instruments (*e.g.,* piano, violin, saxophone) of the performer by virtual instruments based on sampling technology or physical modelling.
On the listener side of our game there exist, for example, approaches for detecting the pitch classes, start- and end-position of notes in raw audio files, approaches for the detection of the used instrument by an analysis of the audio files, different kinds of approaches for inferring the rhythmical structure of low-level symbolic musical data.

The *data exchange* between the composer and listener side traditionally happens on the audio level, when humans listen to music. But also at the score level, data is exchanged when (experienced) listeners analyse music only using the graphical score. Approaches to optical music recognition (OMR) address both sides of this game and also connect them at the score level. At the composer/performer side OMR is used to convert (existing) graphical scores into symbolic input data for algorithmic performing systems, on the listener side OMR is used to convert graphical scores into symbolic input data for higher level analysis systems.

The scope of this thesis is on the listeners part of this hypothetic game. Its focus starts slightly above the raw audio level, where already pitch information based on semitone steps, the intensity, and absolute timing information (in units of seconds) of the start- and end-positions of notes are available. The goal of our approach is the creation of a maximum amount of score level information from the given input data. Ideally, the result would be exactly the original score that has been used by the performer.
Tasks, such as an harmonical or functional analysis of the given music – not explicitly represented in scores and not necessarily needed for performing – or perceptional aspects of an human listener are not addressed in this thesis. Nevertheless, specific knowledge about the mechanisms how certain musical features are perceived by listeners will be used.
Our approach is implemented as a ready-to-use system, using newly developed algorithms as well as improved or adapted existing algorithms that have been described by other authors.

In general an automatic transcription system should provide the user an unconstrained performance condition ([CDK00]) during the creation of input data. In addition to this performance related goal the here proposed approaches and their implementation have been developed according the following 'rules':

- **Adequacy** – simple input data should require less user input and less specification of parameter settings than more complex input data. The algorithms should recognise automatically, if default settings can be used, if these can inferred from the data itself, or if an ambiguous respectively complex situation requires additional user input. It should be avoided that the user needs to specify not necessarily required parameters in advance.
- **Flexibility** – the algorithms should recognise and adapt automatically to the type of given input data. For example, it should be automatically inferred by an analysis of the input data if this is a mechanical performance or an human played performance.
- **Interactivity** – if not used in batch mode – where no interactivity is desired – the different modules of the system should ask for needed additional user input if required. This might be by simple yes/no questions, select questions, or requests that the user resolve ambiguous situations manually.

Our implementation of the approaches described in the following includes the basic functionality for the interactivity features, but no high-end graphical user interface. The design of a graphical user interface (GUI) is not in the scope of this thesis.

In the following we first give a more detailed introduction to the process of algorithmic musical transcription, then we briefly present some existing systems with a scope similar to our approach. In Section 1.4 we give an overview about our implementation and its different modules. The following chapters then address the specific key issues in the area of computer aided transcription. A conclusion and possible directions for future research can be found in the Epilogue. The Appendix includes detailed information about file formats, command line parameters, specific test files, as well as other background information on specific topics.

Acknowledgements

This work has been written between 1997 and 2004 during my membership in the Salieri Project at the Darmstadt University of Technology. Parts of it have been written on places all around the world like Vancouver, Gstaad, Pont Saint Esprit, Hornby Island, or Gran Canaria. It would have been impossible to write this thesis without the support of many people. First I would like to thank my family, especially Ruth Kilian, my loving and caring wife, and my parents for providing me an environment that made

it possible for me to start this work and finish it successfully. I am greatly indebted to Prof. Holger H. Hoos, whom I would like to thank for all the time he spent on co-supervising this Ph.D. project, the many fruitful discussions, and the joint work on research projects in context of the Salieri Project.

I would like to thank Prof. Hermann K.-G. Walter for his support. Without him, research in the field of computer music would not have started at the computer science department of the Darmstadt University of Technology. I am very grateful that I had the opportunity to work as a member of his "Automata Theory and Formal Languages" Group at TUD; during this time, I enjoyed great freedom in pursuing my research interests and gathered a wealth of invaluable experience in many aspects of academic life.

I also thank the members of the "Automata Theory and Formal Languages" Group, in particular Oliver Glier, Dr. Ulrike Brandt, our Diploma Student Nima Barraci, and Elfriede Steingasser for their support and many interesting discussions. Furthermore, I would like to thank Prof. Keith Hamel for his support and my colleague and friend Dr. Kai Renz for many fruitful discussions during our joint work in context of the Salieri Project. Many thanks also to Matthias Dehmer for the intensive proof reading of the mathematical formulas.

1 Computer Aided Transcription

The domain of computer aided musical transcription includes different areas of challenging issues which address the different levels of the transcription process. At the bottom layer of the complete transcription process the conversion of audio signals – given as raw audio files or continuous audio streams – into a *semi-symbolic* respectively *low-level symbolic* representation needs to be performed.[1] In general here the audio signal is converted into a sequence of events including information about pitch, intensity, and timing of the corresponding notes.
Raphael calls this problem the "signal-to-piano-roll" [Rap01b] conversion. The piano roll notation is a commonly used low-level symbolic format. Here quantised as well as unquantised musical data is organised in a two dimensional diagram, where the abscissa (x-axis) represents the time (in seconds or score time units) and the ordinate (y-axis) the pitch in semitone steps. Different from standard graphical scores where the note durations are indicated by symbols (noteheads with flags or beams) and time positions are specified indirectly by the cumulative sum of note durations, the piano roll representation allows the specification of absolute time positions and arbitrary (unquantised) durations on a continuos time line.

Pitch tracking approaches focus on the conversion of pitch information from frequency units (Hz) into units of semitones.[2] For audio data of a well tuned monophonic instrument this might be a quite simple task, but if we assume that the audio data includes miss-tuned notes, vibrato, bending, and glissando effects it becomes a non-trivial issue to determine the correct original pitch. For polyphonic audio data of a single instrument (*e.g.,* piano) or – in worst case – of a set of different instruments, pitch tracking becomes highly complex and still not completely solved task.
Pitch tracking approaches usually also infer the absolute time positions (in units of seconds) of the start- and endpoints of inferred notes. If we assume that the intensity envelope of a note performed on a non-electronic instrument is usually never a perfect square (with a dedicated start and end position) and series of notes might overlap or might be overlaid by hold notes in other voices, the detection of start- and endpoints of notes also becomes a complex, non-trivial task.
In the literature, there exist various pitch tracking approaches and implementations with a focus on different types of music (*e.g.,* [Tan93], [Rap02], [Mar01], or the Solo Explorer system described in Section 1.3).

Another research area that focusses on processing and analysing audio data, is instrument or timbre recognition. Here the research focusses on algorithms which can infer the used instrument(s) by analysing the audio signal of a performance. Even for untrained human listeners it is quite easy to distinguish at least between instrument families (*e.g.,* horns, strings) when listening to a performance. For trained listeners or musicians it is also rather simple to discriminate even closely related instruments (*e.g.,* tenor saxophone and alto saxophone) just by their sound. In contrast, the algorithmic modelling of these capabilities becomes a rather complex issue. Fujinaga proposed in [FM00] a powerful approach for instrument recognition based on a k-Nearest Neighbour model.
The direct processing of audio files is not in the scope of this thesis, instead we assume that the input data has been already converted to (or recorded at) a piano roll type of semi-symbolic level information, where pitch information is available in semitone steps and the timing information for note on- and offset points.is available in units of seconds.

Instead of extracting symbolic musical data from audio files of performances it also is possible to extract symbolic information directly from graphical scores. As an advantage over inferring low-level symbolic data from audio files here more additional score level information (*e.g.,* key signature, slurring, precise score time information) is explicitly available and needs not to be inferred by additional parts of a system.

[1] The semi-symbolic or low-level symbolic representation is an intermediate level between audio (continuous wave signal) and score level information (notes with discrete pitch and duration information).
[2] Each semitone step is equivalent to a key of a piano keyboard.

There exist commercial systems for this task of optical music recognition (OMR), such as SmartScore or PhotoScore but also very powerful academic systems, such as for example, the Gamera system by Ichiro Fujinaga and Michael Droettbom (see [Fuj96]). Depending on the context, the symbolic representation obtained as output of an OMR system can be used is input for an algorithmic performance system or is input to higher level structural analysis approaches or musical databases containing music in a symbolic representation.

In music, sequences of notes which are played parallel in time are organised as parallel monophonic voices and/or sequences of chords. In graphical scores and also when listening to music, experienced listeners can identify these voices and infer the correct chord groupings. There exist approaches focussing on this type of musical voice separation which try to distribute the notes of polyphonic, symbolic data to a number of logical voices and/or group them into chords. If we assume that the data might contain imprecise timing information and that the number of voices is not constant through the complete performance this voice separation becomes a computationally complex task (see Chapter 2).

Beside creating readable graphical scores a correct voice separation is also important for higher level tasks, such as quantisation or pitch spelling. Because here the local context of a note is evaluated for inferring certain attributes (score duration, pitch spelling) of the note, the quality of the output depends highly on the correctness of this context information. If, for example, a chromatic scale cannot be detected because the corresponding notes have been incorrectly distributed to different voice streams, a pitch spelling module might create an incorrect output. Also in the domain of music information retrieval (MIR) where many approaches are using monophonic data as input, a correct separation into musical voices is required for obtaining correct retrieval results.

Approaches for filtering ornamental notes (*e.g.,* grace notes, trills, glissandi) from the *real* performance notes (*i.e.,* melody notes) also work on the same level of semi-symbolic input data (see Section 6.4). The task of detecting and removing this type of notes from the input data can be viewed as pre-processing for tempo detection, quantisation, and MIR. The removal of these notes can be seen as equivalent to the removal of rhythmical noise from the input data. Higher level approaches become much more robust if this noise can be reduced. A quantisation algorithm, for example, would fail in most cases to find displayable and readable score durations for ornaments, such as grace notes or trills. By trying to represent these rather short notes as explicit notes in a score, the rhythmical complexity of the score would be significantly increased, which is the opposite of the desired output. To avoid this (hard to read) complexity in traditional graphical scores the ornamental notes are indicated by special symbols and not as standard notes.[3]
In addition to just filtering and removing the ornamental notes, transcription systems should also address the issue of identifying the ornament type (*e.g.,* mordent or turn) and indicate these by the correct ornament symbol.

Related to and depending on voice separation are also approaches that search for boundaries of melodic phrases within musical voices and related issues, such as segmentation, similarity analysis, pattern matching, and pattern induction (see Chapter 3). The analysis of the segment structure of a performance is not necessarily required for a general performance-to-score (transcription) approach. But knowledge about the structure (*e.g.,* repetitions) of the piece might improve the quality of its output. If, for example, the overall structure of a piece is known in advance algorithms for key or time signature detection might be applied separately to the individual segments. Related to the analysis of the similarity between performances is also the analysis of similarity of different parts of a single performance (self-similarity analysis). Especially in the growing field of music information retrieval (MIR), approaches for fast methods for approximate similarity analysis are needed.

Tempo detection respectively beat induction is one of major research areas and key issues in the context of computer aided transcription (see Chapter 4). In general, beat induction approaches try to simulate the human listener who tries to clapping hands or tapping a foot synchronous to a certain beat level of an audio signal. In [Rap01b] beat induction is described as: "The main issue here is trying to follow the tempo rather than transcribing the rhythm". As input data for this task audio or low-level symbolic data containing absolute timing information given in seconds is used.

Beat induction approaches need not necessarily infer the beat level of inferred beats which would be indicated by a corresponding score duration. For some applications (*e.g.,* synchronising a light controller

[3]The term *standard notes* should indicate notes which increment the musical time by their duration.

to an audio signal) it is sufficient to know just the position of a beat. If an approach is able to infer the score durations for the estimated beats the tempo profile of the performance can be calculated from the inferred mapping between score and performance timing. These types of tempo detection approaches can be used to convert low-level symbolic data – including absolute timing information in seconds – into low-level symbolic data including timing information in (continuous) score time units. In general the inferred score time positions will, especially for the non beat positions, be still imprecise and not displayable in graphical scores without a quantisation. Section 4.2 includes an overview about existing approaches for tempo detection.

Quantisation of musical time is another key issue and complex task during the process of transcription. Different from tempo detection here the detailed rhythm indicated by the score duration of each single note must be inferred. Where tempo detection converts the time positions from absolute timing (given in seconds) into continuous score time information, the quantisation now converts this information into time positions of a discrete, metrical grid of score time positions.
The readability and usability of a transcribed score depends highly on the quality of the used quantisation and tempo detection approaches. Many approaches for quantisation already exist in the literature. The major developments in this field and a new pattern-based approach are shown in Chapter 5.

In addition to the key issues of tempo detection and quantisation there also exist approaches addressing the inference of additional high level score elements. Related to rhythm transcription – and sometimes used as required input information – is the detection of the time signature of a performance which is described in Section 6.1. For transcribing readable scores also a correct key signature and the correct spelling of ambiguous pitch information is required. For the detection of key signatures there exist several approaches which are discussed in Section 6.2. Related to the detection of the key signature is the analysis of the harmonical structure of a piece. There exist several systems and approaches addressing this problem (see, for example, [Tem01, TS, Win68]). Because an harmonical analysis is not necessarily needed (and usually not explicitly indicated) in graphical scores and also requires different concepts than the here primarily described rhythmical approaches, this issue is not in the scope of this thesis.

For each of the issues and research areas shown above there exist several approaches by different authors. Beside their actual scopes they can be categorised as approaches that address just single issue (*e.g.,* quantisation, pitch tracking) and rather complete transcription systems. They can also be categorised whether they just try to simulate the quality of manual transcription by an human listener or if they address the understanding and simulation of perception processes of human listeners.

1.1 The Educated Listener

When humans listen to music they intuitively and unconsciously analyse the perceived data. In many articles and works in the context of psychology and computer science research, algorithmic approaches and models for describing and understanding these mechanisms have already been addressed (*e.g.,* [SS68, Bre90, LHL84, Par94b, LJ99, PK90]), but the underlying issues are still not completely understood. Without focussing on the underlying mechanisms we can make the following *simple* statements about educated listeners who perceive and analyse music:

- The human listener will use rhythm, as well as implicit melodic and harmonic information for the rhythmical analysis.
  In a rhythm perception experiment performed in the context of this thesis, participants were asked to transcribe two rhythmical identical but melodical different sequences of notes (see Section A.14, Task 1 and 3). The evaluation showed that most of the participants used different rhythm transcription for the two rhythmically identical sequences. In another test we replaced the pitch information of a piece (Chopin *Op. 6, Mazurka Nr. 1*) with by pitch information but kept the original expressive timing of the performance. Where it was even for non-musicians possible to follow the rhythm and beat of the original version, it became nearly impossible (even for musicians) to follow the rhythm and beat of the performance with the randomised pitch information. Another evidence for a strong relation between the perception of melody and rhythm is stated by Deutsch (as cited in [LK94]):

> "In one study, memory for pitch sequences was found to be dependent on a perceived temporal frame. Pitch structures that coincided with temporal structures enhanced recall, while pitch structures that conflicted with temporal structures negatively affected recall." [Deu80]

- The analysis depends on the rhythmical context. The rhythm and note durations of one (musical) voice might be interpreted different depending on the rhythm and melody of other voices played in parallel.
  With Task 2 and 4 of our rhythm perception experiment (see Section A.14) we could show that, for example, a sequence of notes with an equal duration at a certain tempo becomes perceived as a series of eighth notes if they are aligned with an equidistant sequence of longer notes with duration ratio of 2:1. If instead these notes have been aligned with other notes in a ratio of 3:1, most participants preferred a transcription using eighth triplets instead.

- The listener will prefer a simple solution (transcription) over a complex solution. For example: a sequence of equidistant notes will be perceived more likely as a sequence of eights or quarter notes instead as a sequence of triplets, as long as no other indication is given by the rhythmic or melodic context.
  An human transcriber will also prefer simple representations for other score level features, such as the time and key signature. A well known (already learned) solution will be perceived as 'simpler' than a new, unknown solution never seen or used before.

- The human listener tries to match the heard music with previously heard (learned) patterns or motives. It is commonly known that listeners are able to recognise learned melodies even if they are presented in a somehow distorted way (*e.g.,* including incorrectly pitched notes, changes in rhythm). The evaluation of Task 5 of our rhythm perception experiment (see Section A.14) shows some evidence for this. In one task the participants have been asked to transcribe a perceived rhythm pattern. Except for one participant all the other participants decided to use two commonly known rhythm pattern for their transcription. An exact transcription of the perceived rhythm would have required a much more complex, unusual rhythm pattern. In general we assume that listeners try to categorise the perceived music into learned categories of rhythmic pattern.

- During transcribing the rhythm of a performance, the human listener might lose track (or context) during long notes or complex rhythmical structures. But he will try to synchronise again at the next 'simple' passage. For example, Parncutt [Par94b] and also Roberts [Rob96] mention that if the distances between two events becomes larger than approximately two seconds, it gets very difficult for human listeners to perceive the rhythmic context. We could test this assumption with Task 2 of our perception experiment, where a sequence of rather short notes includes a single long note. The typical transcription preferred by most of the participants was to restart with a new tempo after the long note.
  From own experiences we also know that it becomes difficult to transcribe a perceived structure if the rhythm does not follow the binary and ternary structure commonly used in Western tonal music.

This thesis shows that the rhythmical structure of many pieces and performances can be analysed (inferred) without using an harmonic or melodic analysis. Goto and Muraoka state that even untrained listeners are able to perceive a beat structure and even the occurrence of chord changes which not necessarily requires the skill of naming chord types or pitch classes [GM99]. Because the harmonic content and complexity of compositions has changed during the different periods of Western tonal music – the harmonic content of a Bach chorale is obviously different from pieces of the 19th Century, and there exists contemporary music which cannot even be analysed with the traditional models of harmonic analysis – it is easy to see that the analysis of these musical features would require style dependent algorithmic models for the respective musicological background knowledge. By restricting an approach – particularly for tempo detection and quantisation – to the analysis of rhythmical features the model stays style independent and can be applied to different types of music. Such an approach can also be used to determine the boundary where it becomes impossible to infer the correct rhythm of compositions and performances without using the information of an harmonic and melodic analysis (*e.g.,* for expressive style with high tempo fluctuations, contemporary music with complex structures).

1.2 Representation of Musical Time

One of the core tasks in the context of transcription is the conversion of time information from absolute timing (given in seconds) into metrical score timing. Where a (trained) human listener can intuitively create this mapping, algorithmic models have still difficulties to solve this task.

The timing information of musical data can be represented on different levels indicated by different unit systems for the given timing information. On the score level the *score timing* information of notes is specified as fractions, where 1/1 is equivalent to a whole note. By limiting the denominators of these fractions to a typical set $D = \{2^n \mid n = 0, 1, \ldots, 5\} \cup \{3 \cdot 2^n \mid n = 0, 1, \ldots, 4\}$ scores represent a discrete grid of possible time positions. In very uncommon situations slightly larger denominators might be used (64, 128, 48) and also for arbitrary tuplets multiples of larger prime numbers may occur as denominators. But still the set of time positions and durations used for traditional graphical scores is equivalent to a discrete timing system.

> "*Score time* is defined as the relative timing information derived from durations of notes. In conjunction with a metronome setting, score time can be converted to (nominal) note onset time measured in second. The term *performance time* is used to refer to the concrete, measured, physical timing of note onsets." [Dix01a]

On the audio level the *performance timing* information (for durations and time positions) can be measured in seconds. Instead of indicating the tempo of a song by the relation between score duration and performance duration of a dedicated note (*e.g.,* quarter note = 0.5s) in musical scores the tempo is indicated as the inverse relation that is *beats per minutes* (bpm). If no beat duration is specified, bpm will commonly be interpreted as quarter notes per minutes.

> "*Tempo* refers to the rate at which musical events are played, expressed in score time units, for example, quarter notes per minute. When the metrical level of the beat is known, the tempo can be expressed by the number of beats per time unit [...], or inversely as the *inter-beat interval*, measured in time per beat." [Dix01a]

For a performance with constant tempo s given in beats per minute (bpm), and beat duration b_{dur} the relation between performance timing t_{perf} in units of seconds, and score timing t_{score} given as fraction of a whole note, is then defined as

$$t_{perf} = t_{score} \cdot \frac{60}{s \cdot b_{dur}} \ . \tag{1.1}$$

For example, a quarter note (score duration = $\frac{1}{4}$) played at a tempo $s = 120$bpm has a performance duration of 0.5s, played at 150bpm it has a performance duration of 0.4s.

If the tempo of the performance changes, this is indicated by so-called *tempo changes*. In the following a tempo change s is denoted as a vector $s = (tempo, scoretime)$ where $t_{score}(s)$ is the score time of tempo change s and $bpm(s)$ the tempo indicated by s. All tempo changes of a piece can be denoted as an ordered set with $S = \{s_1, \ldots, s_{|S|}\}$ of tempo changes, with $t_{score}(s_i) < t_{score}(s_{i+1})$ and where $|X|$ denotes the number of elements contained in a set X. Because for every time position of a piece a valid tempo indication must be available, we require $t_{score}(s_1) := 0$. The relation between score time, performance time and tempo can be represented in a graphical way like shown in Figure 1.1. For a given set of tempo changes $S = \{s_1, \ldots, s_{|S|}\}$, the performance time position of an arbitrary score time position t, with $t_{score}(s_i) < t < t_{score}(s_{i+1})$ can be calculated with

$$t_{perf}(t) = t_{perf}(s_i) + (t - t_{score}(s_i)) \cdot \frac{60}{s_i \cdot b_{dur}}, \tag{1.2}$$

and

$$t_{perf}(s_i) = \begin{cases} t_{perf}(s_{i-1}) + t_{perf}(t_{score}(s_i) - t_{score}(s_{i-1})), & \text{if } i > 1, \\ 0, & \text{else.} \end{cases} \tag{1.3}$$

In the context of transcription, only the observed performance timing information is available but the score timing and the tempo changes must be inferred. As stated by Raphael ([Rap01b]) this can be seen

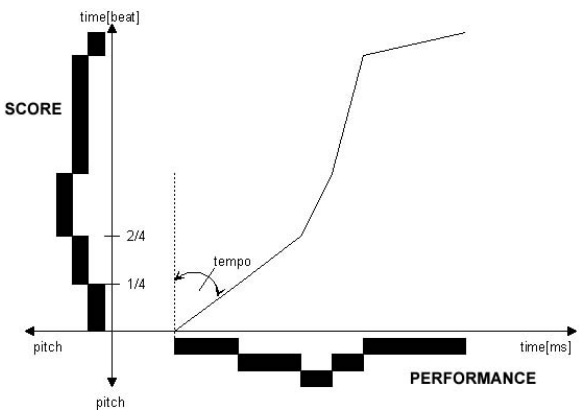


Figure 1.1: Relation between performance timing, score timing and local tempo (as shown in [Kil96]).

as a "chicken and egg" problem.
In general there exist two different classes of performance data: mechanical performances and so-called live performances created by human players and musicians.

> "A *mechanical or metrical* performance is a performance played strictly in score time. That is, all quarter notes have equal duration, all half notes are twice as long as the quarter notes, and so on. An *expressive performance* is any other performance, such as any human performance." [Dix01a]

For the general transcription problem there always exist two trivial solutions

1. Chose an arbitrary, constant tempo s_{const} – with a corresponding beat duration (*e.g.,* $b_{dur} = \frac{1}{4}$) – for the complete piece and calculate the score positions for all onset times and note offset points. When choosing a constant score duration of a crotchet (*i.e.,* a quarter note), a sequence of observed performance time positions $t_1, \ldots, t_n$ will be converted into a sequence of score time positions $t'_1, \ldots, t'_n$, with
$$t'_i = t_i \cdot \frac{s}{60 \cdot 4}. \tag{1.4}$$
For non-mechanical performances with tempo fluctuations this would result in a very complex, nearly unplayable and unreadable score with very uncommon note durations (which in general may not be displayable) as shown in Figure 1.2(a).

2. Chose for every note (and rest) of the score an arbitrary, constant, and equal score duration d_{const} and calculate the resulting tempo for each note as shown in Figure 1.2(b). A sequence of performance time positions $t_1, \ldots, t_n$ (where each t_i correspondents to an onset time of a note m_i, with $t_1 = 0$) will here be converted into a sequence of score time positions $t'_1, \ldots, t'_n$, with $t'_{i+1} - t'_i = d_{const}$ – an arbitrary constant score duration – and a sequence of tempo changes $s_1, \ldots, s_{n-1}$ (in units of quarter notes per minute) where
$$s_i = \frac{d_{const} \cdot 60 \cdot 4}{t_{i+1} - t_i}, \quad \text{for } i = 1, 2, \ldots, n-1. \tag{1.5}$$
The score time positions of the tempo changes $s_1, \ldots, s_{n-1}$ are equivalent to the score time positions $t'_1, \ldots, t'_{n-1}$. Also this strategy will usually result in – for human musicians – unplayable scores. Nevertheless, for scores, such as many of Bach's Inventions, with many equidistant onset times and a constant performance tempo, this naive approach would give good, correct results for large sections of the score, if d_{const} is chosen correctly.

A correct or optimum solution of any tempo detection or beat induction algorithm should be an optimal (balanced) transcription between the two trivial transcriptions shown in Figure 1.2 ([CK03]). It is easy

to see that even for the transcription of mechanical performances, there exists always a set of different but rhythmically correct and readable transcriptions which can be obtained by multiplying the durations of any correct (readable) transcription by a small integer or its inverse. As shown later in Chapter 4 and

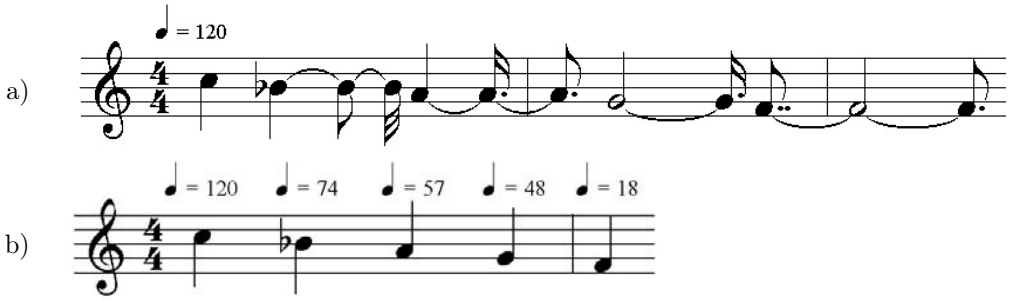


Figure 1.2: Trivial solutions for transcription: a) fixed tempo, b) fixed duration.

Chapter 5 the described transcription process becomes even more complex if we assume that the input files represent human performance data including tempo fluctuations and inaccuracies. In general we must assume that the mapping between performance and score as well as the mapping between score and performance is a relation and not a function. Because of the different interpretation of music by different performers there exist multiple correct but different performance for a single score. There also exist several correct but correct transcriptions for a single performance. The key issue for any transcription system is to infer for a given performance a score of the set of possible scores which is close as possible to the unknown original score.

1.3 Existing Systems

Beside approaches focussing on single issues in the area of computer aided transcription – implemented rather prototypically for usage in research – there exist also some complete transcription systems. In the following we give a brief overview on some of these systems. The underlying models and approaches will then separately be discussed and compared in the specific chapters.

Transcription System by Cambouropoulos

Cambouropoulos presented in 2000 a transcription system [Cam00b] including modules for beat induction, onset and duration quantisation, streaming (voice separation), and pitch spelling. The system uses MIDI files as input but it is unclear what file format is used for the output of the transcribed data. The proposed models for streaming and pitch spelling are discussed in more detail in Chapter 2 and Section 6.3. The system has been prototypically implemented by its author using the Prolog programming language. Unfortunately no running version of this system was available for a detailed evaluation. Also the original data set (MIDI files of 13 live performed Mozart Sonatas) for the described results in [Cam00b] could not be obtained for a detailed comparison between the output of this system and our system because of copyright issues.

The Melisma Analyser

Temperley and Sleator present in [Tem01] and [TS] the Melisma Analyser, a system for computer-based simulation of the human cognition of basic musical structures. The system consists of several separate modules addressing the analysis of the metrical structure, the melodic phrase structure, the contrapuntal structure, pitch spelling, harmonic structure, and key structure. Each module represents an implementation of a rule system for the specific task, where for each module an overall best solution is generated by using a dynamic programming approach.

	onset time	duration	pitch
Note	174	348	60
Note	348	523	62
Note	523	697	64
Note	697	872	65
Note	872	1046	62
Note	1046	1220	64
Note	1220	1395	60
Note	1395	1744	67

Table 1.1: Beginning of Bach *Inventio 1* in ASCII note list format as required as input by the Melisma system.

	time	beat level
Beat	0	3
Beat	70	0
Beat	175	1
Beat	245	0
Beat	350	1
Beat	420	0
Beat	525	2
Beat	595	0
Beat	700	1
Beat	770	0
Beat	875	1
Beat	945	0
Beat	1050	4
Beat	1120	0
Beat	1225	1
Beat	1295	0
Beat	1400	1

Table 1.2: Sample output of Melisma's meter module for the beginning of Bach *Inventio 1*. The centre column represents the time position of the corresponding beat in milliseconds.

Different from our approach where the focus is mostly on inferring score level information which can be explicitly displayed in musical scores, the Melisma system is build with a focus on musical analysis. Melisma takes so-called note lists as input where the note on-time, off-time (in milliseconds), and pitch (in semitone steps) for each note of the piece is specified in a simple ASCII format (see Table 1.1). The musical information of these note files is similar to the input required for our approach, but Melisma evaluates no intensity information of performed notes. Using a command line tool (`mftext`), MIDI files can be easily converted into this format.
The basic module of the Melisma system is the meter module for inferring the metrical structure of a piece. It works somehow similar to tempo detection and quantisation modules of transcription oriented systems. But, instead of inferring directly the score position and duration of notes, Melisma infers beats and generates a mapping between absolute time positions of these beats and a corresponding metrical level. These metrical level have been introduced in the *Generative Theory of Tonal Music* (GTTM) by Lerdahl and Jackendoff [LJ83]. Assuming that the metrical level has been inferred correctly, it should be a rather simple task to assign specific score durations to each level for converting the metrical structure into score time information. Melisma's melodic phrase structure and harmonic structure module are addressing challenging issues which are beyond the scope of this thesis. The streamer module for creating a contrapuntal analysis will be compared to our approach for voice separation in Chapter 2 and the modules for pitch spelling and key detection will be briefly discussed in Section 6.3 and Section 6.2.

Because the modules of Melisma system generate only an ASCII-list-based output (see, for example, Table 1.2, Table 1.1, and Figure 1.3) which cannot directly be used as input for any standard notation system, the results of its modules cannot be compared visually (by comparing graphical scores) to the score level output of other systems. Therefore, with exception of the key detection module the comparison requires a large amount of manual work.

Cypher

Robert Rowe proposed in [Row93] an interactive music system named Cypher. This system consists of two parts: a listener part and a compositional part. The listener part analyses input MIDI streams with

```
                        C1          C2          C3          C4          C5          C6          C7
Seg    0 (3):           .           .           .           X           .           .           .
Seg    1 (1):           .           .           .           | 2         .           .           .
Seg    2 (1):           .           .           .           |   2       .           .           .
Seg    3 (2):           .           .           .           |    2      .           .           .
Seg    4 (1):           .           .           .           | 2         .           .           .
Seg    5 (1):           .           .           .           |   2       .           .           .
Seg    6 (4):           .           .           .           1   |       .           .           .
Seg    7 (1):           .           .           .           |     2     .           .           .
Seg    8 (1):           .           .           1           .     2     .           .           .
Seg    9 (2):           .           .           . 1         .           2           .           .
Seg   10 (1):           .           .           .   1       .           2           .           .
Seg   11 (1):           .           .           .    1      .          2.           .           .
Seg   12 (0):           .           .           .    1      .        2 .           .           .
Seg   13 (3):           .           .           . 1         .          2.           .           .
Seg   14 (1):           .           .           .   1       .           2           .           .
Seg   15 (1):           .           .           1           .           2           .           .
Seg   16 (2):           .           .           .     1     .           . 2         .           .
Seg   17 (1):           .           .           .     1     .     2     .           .           .
```

Figure 1.3: Sample output of the streamer module of the Melisma system for the beginning of Bach *Inventio I*. Each column represents a certain pitch class, each row represents a time position of the score.

several separate modules (harmonic analysis, key identification, beat tracking, structural analysis); the compositional part includes modules for interactive, algorithmic composition which will not be discussed here. The system is implemented using the MAX programming environment (see [Puc91]),[4] where the separate modules are implemented as so-called agents. Different from other systems mentioned here Cypher is designed to be an interactive, real-time system and does therefore not create any score output. Nevertheless, it includes some models (*e.g.,* key finder, beat tracker) which are of interest in the context of transcription. The details of these models will be discussed later in detail in the corresponding sections (*i.e.,* Section 3.4, Section 6.2.1, Section 4.2.3).

Transcribe

The Transcribe[5] system proposed in 1993 by Pressing and Lawrence uses audio files with (manually) marked start positions of events or MIDI files as input. The main focus of this approach is on rule-based quantisation, but also a simple beat induction algorithm is implemented. After a segmentation of the input data – which is not described in detail by the authors – a grid-template-based quantisation algorithm (see Section 5.1.3 for a detailed description) is applied to the events. The quantised output is then converted back into MIDI files. Transcribe does not infer any other score level information, such as key, meter, or slurring. Originally the system has been implemented for Macintosh computers, but unfortunately no running version is publicly available.[6]

Solo Explorer

A commercial system called Solo Explorer for the transcription of audio files containing monophonic lines (of an arbitrary instrument) into score notation is available at `http://www.recognisoft.com`. In addition to the transcription of pitch and timing information the system includes also a key detection module. The system's website contains some examples (vocal lines) of sample output which promises a decent transcription quality. Our own tests with an audio recording (piano sound) of the upper voice of Mozart *Sonata in D, KV 284*, measures 19–21 showed significantly worse results. The pitch tracker

[4] MAX is a graphic programming language for building (MIDI-based) interactive music systems.

[5] The Transcribe system by Pressing and Lawrence should not be confused with the commercial Transcribe! system available at `http://www.seventhstring.com`

[6] Dr. Jeffrey Pressing died in 2002.

algorithm seems to have problems with short notes and fast passages. In our test a major number of notes have been inferred as pitch bendings instead of separate notes. We assume that for the vocal examples of the system's website this issue does not occur because of the natural limitations of the human voice. It is also not clear which methods have been used for tempo tracking and quantisation. The note durations of the generated output files (MIDI) seem to become quantised to integer multiples of sixteenth notes, but the files do not include a tempo profile.
Because of the worse results for piano music input, the limitation to monophonic pieces, and also the lack of documentation about underlying models and approaches we did not evaluate the output of this system in detail.

1.3.1 Music Notation Software

Beside approaches and systems build for computer aided transcription there also exist software systems for music typesetting which could be used to display the transcriptions as graphical score. In the main focus of these systems are powerful algorithms for generating a correct and readable spacing in graphical musical scores. Many systems also include a graphical user interface for creating and editing notes or other score elements. In addition to a step by step input with computer keyboard or computer mouse most systems also offer functions for importing MIDI files – which need to be quantised – or real-time input with a piano keyboard. Because MIDI files include musical information below score level (*e.g.,* performance data, real-time timing information, ambiguous pitch spelling, see Section 1.4.2) these import functions represent implementations of transcription approaches.

Sibelius

Since 1987 Jonathan and Ben Finn are developing a commercial score notation software named Sibelius.[7] For our tests (see Section 5.4) we used the version Sibelius 2.1.1. Currently Sibelius and the Finale music notation system can be seen as the market leaders for music notation software and currently appear to be the most commonly sold notation software packages. Both are offering powerful features for creating professional scores via a (GUI).

In general, graphical user interfaces for musical typesetting seem to be fast and intuitive. Our own experience has shown that this might be true for editing existing, large scores (*e.g.,* adding slurs, changing the pitch of notes). But for the creation of new scores or simple score examples (*e.g.,* scales) the usage of a text-based format, such as Guido ([HHFK98, HHRK01]), is typically much more adequate. We believe that a 'perfect' system would include the graphical, GUI-based edit operations of the current notation software packages on top of a powerful ASCII-based music representation language. Such an approach could be compared to a what-you-see-is-what-you-get (WYSIWYG) text editor that uses the ASCII-based HTML format as underlying representation which could be changed manually if needed.

Sibelius includes offers only a tempo detection functionality for real-time MIDI input (*i.e.,* via a MIDI keyboard) called Flexitime: the user enters a number of count-in clicks (by pressing a keyboard key) and starts then to play his performance with that tempo; the Flexitime module tries then to follow the beats of the performance which can include tempo fluctuations. For unknown reasons this mechanism is not provided for the input of performed MIDI files, such that tempo detection on MIDI files is not possible. When importing MIDI files into the Sibelius system, a simple – split-point-based – voice separation can be applied (see Section 2.1.1). A fixed split point can be set by the user or inferred automatically by the software. An inferred split point respectively its pitch class will be valid for the complete piece. Overlapping notes that not become split by the split point rule will not be split by any other heuristic. Instead they are drawn into the corresponding staff as chords, where single chord notes are tied with successive or previous notes (see Figure 1.4 for an example). The creation of parallel voices (instead the sequence of chords) would produce a much more readable score containing more logical information about the melody lines (see Figure 1.4(b)).

In the context of the Salieri project at the Darmstadt University of Technology software plug-ins for exporting Guido files from Sibelius and Finale have been developed which can be downloaded at `http://www.salieri.org/GUIDO`.

[7] Copyright by Sibelius Software Limited.

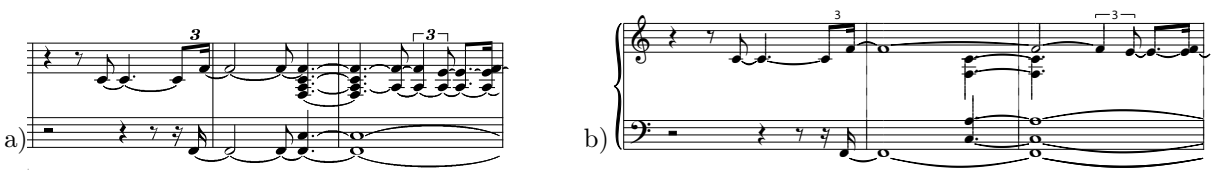


Figure 1.4: Sibelius: merging of overlapping notes into a sequence of tied chords (a) instead of intelligent voice separation (b).

NoteAbility

NoteAbility is a powerful, commercial music notation system developed by Keith Hamel. Currently the full version is available only for the Mac OS-X operating system. Beside the full version there also exists a 'light', freeware version NoteAbility Light available for OS-X and Windows. The source code and functionality of our approach as described in [Kil96] has been integrated into NoteAbilityLight to allow a MIDI import. Different from other commercial notation software NoteAbility provides a native im- and export of files in GUIDO syntax. This allows, for example, the import of the GUIDO output of the implementation of the here proposed system. Currently it is planned to integrate the latest version of our system also into the full version of NoteAbility in the next future. By combining the GUIDO im- and export functionality of NoteAbility with its GUI-based edit functions, the user can edit scores on the graphical level as well as on the ASCII level and can switch between both representation levels when needed.

GUIDO **NoteViewer**

The GUIDO NoteViewer and its online version named GUIDO NoteServer have been developed in the context of the SALIERI project at the Darmstadt University of Technology. Kai Renz developed and implemented the underlying powerful spacing model for musical typesetting as proposed in [Ren02]. The source code of this system is publicly available at the homepgae of the open source GUIDOLib project ([GUI]). The NoteViewer can display files in GUIDO syntax or MIDI files by using the here described midi2gmn implementation as pre-processor. The freeware NoteViewer distribution whcih can be downloaded at the GUIDOLib homepage (see [GUI]) includes the current version of our midi2gmn system as an integrated component which is automatically executed if the user tries to open a MIDI file within the NoteViewer. Different from other notation software packages this system has been developed with a main focus on automatic, high quality layout of musical scores. Therefore it does not provide any GUI-based edit functionality. All score examples (except the Sibelius examples and the scanned image in Section A.11) in this thesis have been created with the current version of the NoteViewer.

1.4 The HEISENBERG Project

Before describing the general outline of our system we need to explain the name of project associated with this thesis. The project was named after the German physicist and Nobel laureate Werner K. Heisenberg because the relation between score timing, performance timing, and local tempo reminded us in the widest sense on Heisenberg's uncertainty principle. Heisenberg showed that in domain of quantum theory it is impossible to measure momentum and position of an electron with total accuracy at the same time. In the context of musical transcription this could be translated into 'it is impossible to measure at the same time the performance time, the score time, and the local tempo with total accuracy'. By estimating (measuring) the performance time and the local tempo the score duration becomes fixed and cannot be observed independently from the first two parameters. It should be noted that we do not intend to compare our work or its significance in any way to the outstanding research of Heisenberg. The name should just express that the process of transcription has to deal also with a large amount of fuzziness and uncertainty.

Our system is divided into separate modules addressing different components and issues of the transcription process. As shown in the system overview in Figure 1.5 the different modules work in a sequential order, where the use of many modules is optional (*e.g.,* the tempo detection can be skipped if the input data already includes a tempo profile). Only the use of the pre-processing, the voice separation, the quantisation, and the actual output module are mandatory to ensure that displayable output scores are created. The sequential order of the single modules have been chosen in a way that modules can benefit

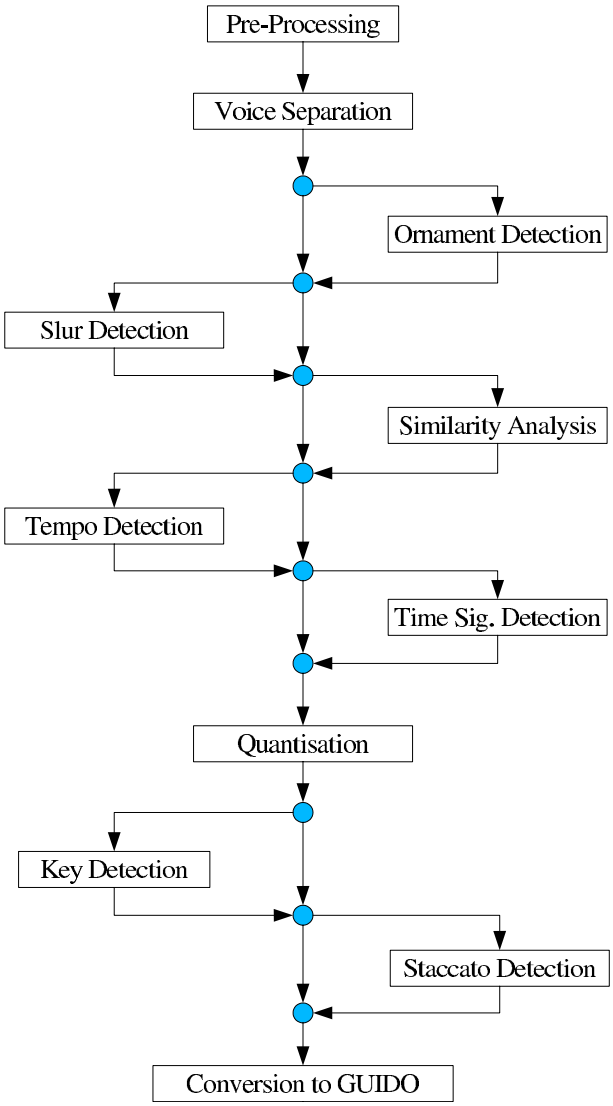


Figure 1.5: Overview about the modules of the `midi2gmn` implementation.

from the results of other modules as much as possible. For example, the ornament detection should precede the tempo detection because the removal of ornamental notes will increase the output quality of the tempo detection (see Chapter 1). Also, the knowledge about the correct time signature will improve the quality of the quantisation because parts of it depends on the time signature. In the following, we briefly describe the modules of our implementation:

- Pre-processing – After converting audio data into semi-symbolic data (including pitch, time, and intensity information) by a separate pitch tracking algorithm, during the pre-processing very close onset times and end points of notes become merged to equal time-positions. The quality and

correctness of this pre-processing step effects several other parts of the approach (*e.g.,* the voice separation depends on correct merged onset times). This module is described in Section 1.4.3.

- Voice Separation – The separation of polyphonic input data into voices representing sequences of notes and chords (see Chapter 2) could actually be performed at any stage of the transcription process – especially before or after the tempo detection. As we will show in the corresponding sections the later modules evaluate the context information of notes and depend therefore on the quality and correctness of this information. Therefore, in our approach the voice separation is performed before tempo detection and not later on the quantised data which would actually decrease the complexity of the voice separation task.

- Ornament Detection – The approaches and algorithms used for inferring the location and different types of musical ornaments are discussed in Section 6.4. By filtering the ornamental notes from the input data before starting the tempo detection the output quality of the tempo detection and also of the quantisation module can be increased.

- Similarity Analysis - As shown in Section 4.3.2 and Section 5.3 some of the here described approaches are based on rhythmical patterns that can be obtained by similarity analysis and segmentation of musical data. In Section 3.4 two new approaches for the detection of significant, repeated segments of a performance and similarity and self-similarity analysis of musical data are described. Our approach to self-similarity analysis can be also used for the similarity analysis between two arbitrary performances (or scores) which is of interest on the context of music information retrieval (MIR).

- Tempo detection – In Section 4.2, we present several existing tempo detection approaches and in Section 4.3 a new approach based on pattern matching and statistical analysis is described. If the input data already includes tempo profile information (*e.g.,* recorded along the sequencers metronome clicks) tempo detection can be skipped.

- Time signature detection – Because the pattern-based part of our quantisation approach is based on the time signature of the performance this module is located between tempo detection and quantisation. If the given input data already includes valid time signature information the execution of this module can be skipped. Possible approaches for inferring a time signature are shown in Section 6.1.

- Quantisation – Our quantisation module is implemented as a context-based, multi-grid quantisation approach in combination with a pattern-based approach. Because the output of our system is intended to be a readable or at least displayable score, the execution of the quantisation module is mandatory. This ensures that the output file contains only rhythmical constructs which can be displayed in regular graphical scores. (For example, a score duration of 191/192 could not be displayed correctly in a graphical score.) Beside our own quantisation approach we discuss in Chapter 5 also the details of several existing approaches.

- Key detection – This module implements a method for the estimation of a key signature based on the analysis of the statistical distribution of the observed intervals. It contains also a description of methods for correct pitch spelling. A detailed description of this module and possible approaches can be found in Section 6.2.

- Articulation related score elements – Our implementation includes two rule-based modules for inferring slurs and staccato information. These modules (see Section 6.6 and Section 6.7) are based on the comparison of the given performance data and the inferred score data and do not require a statistical analysis or specific algorithmical models.

- The final output module converts the internal data structure into files in GUIDO syntax (see Section 1.4.1 for a description). This part of our system is relatively straight-forward, since it does not include any scientific model or approach and will therefore not be discussed in detail in this thesis.

The described modules have been implemented as a command line application called midi2gmn. It is written in ANSI C++ and can therefore be compiled and used on any standard operating system. To avoid any platform or library depencies or any additional overhead we only used the standard C++

class libraries. In addition to the standalone command line implementation we also implemented a web interface where the user can specify the required parameters and can retrieve the corresponding score level representation of his input data in GUIDO syntax.

1.4.1 Representation of Score Level Information

For storing the inferred score level information, a file format is needed that is powerful enough to represent at least all inferred score level information. This requires that this format must be able to represent timing information in metrical score time units, tempo indications, explicit chords (not just overlapping notes), non-ambiguous pitch spelling, explicit ornaments, time and key signatures, as well as slurs, staccato, and similar markings. It is obvious that the commonly known MIDI file format (see Section 1.4.2) cannot be used because of its rather performance oriented representation of musical data. Other possibilities for an output file format would be the native file formats of commercial notation software applications which would cause platform and system depencies for using the output of our system. Beside file formats developed for commercial software, there exists also non-commercial file formats for representing score level information (*e.g.,* DARMS [SF97], MuseData [Hew97], MusixTEX [TME99], or GNU LilyPond [NN04]). A non-commercial, platform independent, and human readable file format which supports all the required features from above is GUIDO Music Notation. Because it also has major advantages (*e.g.,* adequacy, flexibility) over other existing score level file formats [HHRK01] it was chosen as output file format for our approach.

GUIDO[8] has been developed since 1998 by Hoos *et al.* ([HHFK98], [HHRK01]). Files in GUIDO syntax can be rendered into graphical score by using applications, such as the GUIDO NoteServer (web service at `http://www.noteserver.org`), the GUIDO NoteViewer (application for Windows, Mac OS-X, LINUX (prototype) see [Ren02] or [GUI]), or commercial notation software packages, such as NoteAbility or QuickScore. There also exists a variety of command line tools and plug-ins for converting other notation formats (*e.g.,* MuseData, Sibelius, Finale, MusicXML) into GUIDO or GUIDO into other formats (*e.g.,* MIDI, CSound score files).

In GUIDO pitch and duration information of notes is specified by a letter for the pitch class, an optional indication of accidental(s), the octave information (range $(-\infty, +\infty)$) and the duration given as a fraction separated by an asterisk from the octave information. For example, a g-sharp in the first octave with duration 3/16 would be denoted as `g#1*3/16`. A rest can be denoted by an underscore symbol followed by the duration information (also separated by an asterisk). For example, `_*3/4` would be the GUIDO representation for a rest with a duration of three quarter beats.

Additional score level information, such as time signature, key signature, slurring, *e*tc. can be specified by using tags. GUIDO tags are indicated by a backslash symbol followed by the tag's name. Tags might have a list of parameters and also a range of notes to which they should be applied can be specified. For example, in GUIDO syntax `\meter<"7/8">` denotes a $\frac{7}{8}$ time signature. Please see [HHFK98] for a detailed description of the syntax and the file format specification.

1.4.2 Representation of Input Data

For the here described transcription process the input data must be available as symbolic data. Different from a continuous audio stream (wave data) the information for each performance note must be explicitly specified. For each note the following basic information is required:

- pitch – given in semitone steps
- timing – start position (onset time or onset) and end position (offset) in seconds for each note
- intensity – given as a float number in a range $(0, 1]$, where 1 encodes a maximum intensity.

The pitch information given in semitone steps includes enharmonic ambiguities which need to be solved depending on a given or inferred key signature and local context information (see Section 6.3). Tracking

[8] In the following GUIDO and GUIDO Music Notation will be used equivalently.

the correct pitch frequencies and onset time from audio data with respect to articulations, such as pitch bend, grace notes, or glissando effects is a research topic on its own and is beyond the scope of this thesis.

For the current implementation of our approach the Standard MIDI File format (SMF, see [HSF97]) was chosen as standard input format. For analysis and tests files in Basic GUIDO format and a special Low-Level GUIDO format (see Section A.10) can also be used as input.

MIDI File Format

A commonly used and well known data format for exchanging and storing symbolic, musical performance information is the Standard MIDI File format.[9] The MIDI specification was published in 1988.[10] MIDI files are capable of storing time stamped symbolic information – on a performance level – about the intensity of pressed or released keys of a keyboard. The pitch information is encoded as the number of the equivalent piano key (range $= 0, 1, \ldots, 127$; middle c in first octave $= 60$) equivalent to semitone steps. Because the intensity of notes performed on an electronic keyboard or synthesizer is derived from the speed with which the corresponding key has been pressed, in the MIDI specification the term *velocity* is used as synonym for the intensity of notes. In addition to the basic note information MIDI files can also store information about changes of control elements (*e.g.,* sustain pedal, volume pedal, patch number), about meta data (tempo, key, song-, voice-, and instrument-names), and about low-level system data (*e.g.,* patch dump of audio samplers).
The MIDI format is event-based where the time distance between two successive events is specified as a *delta time*. The absolute time position of an event e_i can only be calculated by summing up all delta times of events $e_1, \ldots, e_{i-1}$. The events are organised in tracks (numbered from 0 to $256^2 - 1$) and MIDI channels where track 0 contains global information (*e.g.,* tempo, time signature) all other tracks can contain an arbitrary number of events for the 16 defined MIDI channels (*i.e.,* channel 1 to 16).

Musical notes are stored as a pair of time stamped note-on and note-off events with equal pitch information. No direct information (*e.g.,* pointer) is available for retrieving the corresponding partner for a note-on respectively note-off event. During parsing a MIDI file therefore a list of pending note-on events is needed where a note-on event can be removed from the list at the first time a note-off event with an equal pitch class is parsed. A MIDI file parser should be aware of nested note-on note-off events; it might be possible that two or more note-on events with the same pitch information are parsed *before* a corresponding note-off event occurs in the input data.[11] The ambiguities (the correct original note-on, note-off pairs are unknown) cannot be resolved with total certainty.

Score information, such as slurring, ornaments, dynamic information, or accidentals cannot be stored explicitly in a MIDI file. These information is only implicitly encoded in the note duration respectively in the length of rests or gaps between successive notes.
Because the pitch information of notes is stored as an integer number in the range of 0 to 127, for example, an e-flat and d-sharp in the same octave will have equal pitch numbers. The time and meter signature, as well as a tempo profile can be stored by so-called *meta events* in a special control track of a MIDI file (*i.e.,* track 0).

All timing information (encoded as delta times as shown above) are stored as integer numbers in units of *MIDI ticks* where a global resolution parameter *ppq* (parts (ticks) per quarter note) sets the number of ticks equivalent to a quarter note duration. The tick timing information t_{tick} can be converted into (low-level) score timing information t_{score} using

$$t_{score} = \frac{t_{tick}}{4 \cdot ppq}, \tag{1.6}$$

where t_{score}, t_{tick} and *ppq* must be integer numbers. From the *integer* constraint for the tick timing information follows that for a given resolution *ppq* there exists always some score durations which can be expressed only approximately in MIDI tick timing because of round-off errors. If a MIDI file data has been exported from a notation software or live recorded to a metronome click triggered by the tempo

[9] MIDI = Musical Instrument Digital Interface, see [HSF97] for a detailed description.
[10] The development of the MIDI hardware interface goes back to 1982, it was used the first time in a commercial product in 1986 two years before the official specification was published.
[11] This could be the result of a track merge operation on the MIDI file.

profile of the MIDI file, the timing information for the events is correct or *approximately* correct. Using this type of MIDI file as input data, the tempo profile is already known and does not need to be inferred by an algorithm. If a MIDI file has been live recorded without using the tempo profile, the tick timing information will not reflect the correct score time positions, because the metronome of the recording device was not synchronised with the performance. But all tick timing information t_{tick} can be normalised into absolute performance timing information t_{perf} (in seconds) with a modified version of Equation 1.1:

$$t_{perf} = \frac{t_{tick}}{4 \cdot ppq} \cdot \frac{60}{s} \tag{1.7}$$

Here s denotes the metronome speed of the recording software device during the recording. It is typically given in quarter beats per minute and explicitly stored in the MIDI file. The timing information of a MIDI file therefore satisfies our requirements (see Section 1.4.2). If, as described above, the tempo profile is unknown because of not synchronised metronomes the profile needs to be inferred by a tempo detection algorithm, if it is known the tempo detection might be skipped. In the following we will assume that *live recorded* MIDI file do not contain any tempo profile information which implies that a tempo detection is mandatory.

Using Guido Music Notation as Input Format

In addition to MIDI files, the current implementation of our system also supports Guido files as input. There exist three different types of Guido files which can be evaluated by our implementation:

1. Guido files according the Basic- or Advanced-Guido specification, where the duration of events (notes and rests) is specified as fractions of metrical score time units (as shown in Section 1.4.1). Because Guido supports arbitrary durations (*e.g.,* 17/57) it is possible to represent unquantised data in this format and to use it as input for the quantisation module. The Extended Guido specification also allows the specification of durations in seconds or milliseconds which allows to use Extended Guido as input for the tempo detection module. Using Guido files – containing quantised, score level data – as input is also of interest for processing them with the similarity analysis module of our algorithm.

2. Files in Guido syntax according to the Low-Level Guido specification (see Section A.10). In Basic and Advanced Guido files the musical information is organised in sequences (parallel in time) of non-overlapping notes, chords, and rests (sequential timing). This concept is strong enough to express arbitrary scenarios of overlapping and partly overlapping notes by specifying them in parallel sequences. To also support the one-to-one translation from file formats where notes are organised as sequences of separate note-on and note-off events (*e.g.,* MIDI) the Low-Level Guido format can be used. Here notes are specified with separate \noteOn and \noteOff tags using `empty` events for specifying the time (in milliseconds) between two tags. Please refer to Section A.10 for a detailed description of the Low-Level Guido format.

3. Customised Guido files using a non-standard \note tag. This tag allows to specify the absolute time position, the absolute duration, and the pitch of a note as parameters of a tag which allows the use of one-to-one translations of arbitrary note list formats – used, for example, by the Melisma System (see Section 1.3) – into Guido syntax as input for our system.
   Specifying time positions and durations as parameters of Guido tags is somewhat at odds with the Guido specification where tags are not allowed to have any duration. In a future version of our implementation the preliminary \note tag will be therefore replaced by equivalent mechanisms of the Streamed Guido specification which is currently under development.

By allowing Low-Level Guido as input and output, all modules of our approach can be viewed and also in principle be implemented as Guido-to-Guido transformations. For example, the voice separation can be implemented as a standalone tool, using Low-Level Guido including \noteOn and \noteOff tags or Streamed Guido as input and creating Extended Guido files with durations specified in real-time units as output. These files can be parsed by a standalone implementation of the tempo detection module and converted into Basic Guido files where all note durations are specified in unquantised score time units.

Then a quantisation module can parse these files and create quantised Basic GUIDO files, which then can be rendered to a graphical score by any notation software that supports GUIDO.

1.4.3 Pre-Processing

Before starting the advanced steps of voice separation, tempo detection and quantisation a pre-processing step can and needs to be done. Performance recordings of human players will always include minor inaccuracies in onset time and release positions of notes which result from inaccuracies in playing the music and also from delay and latency of MIDI recording devices. When recording a performance through a MIDI connection, events (*e.g.,* key down or key up) played at equal time must be transmitted as a stream of successive events. Usually the delay is small enough that this type of error can be eliminated completely in the pre-processing step. Figure 1.6 shows an example for the removal of small timing inaccuracies during the pre-processing step. Furthermore, minor differences in the onset times of two

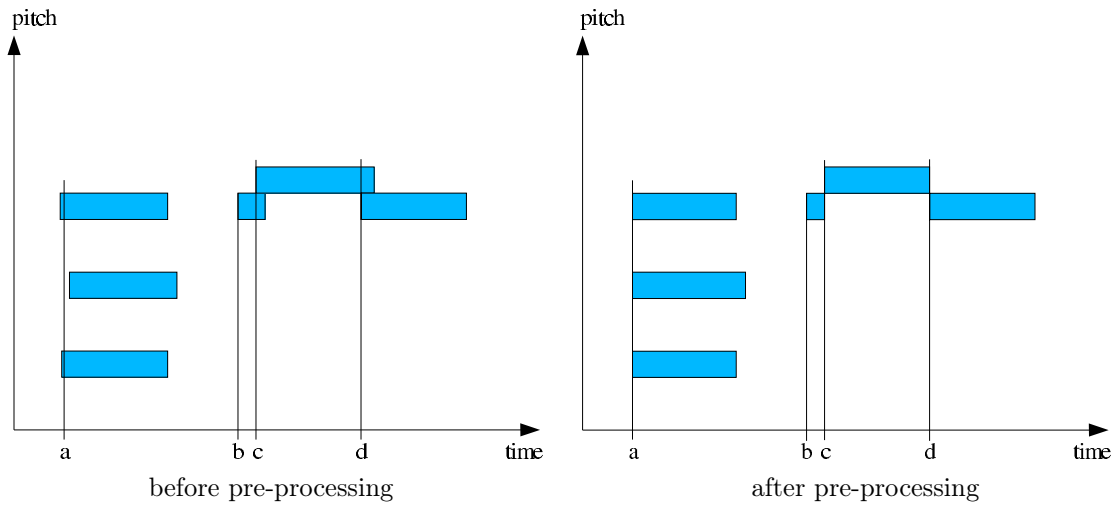


Figure 1.6: Removing overlaps and collecting onset times by pre-processing.

notes that result from imprecise playing can be resolved by replacing both onset times with their average if the durational overlap of the two notes is large compared to the onset time distance and if the onset time distance is very small. It has been shown experimentally that onset times with a distance of less than approximately 50ms are perceived as simultaneous by human listeners (*e.g.,* [Par94b, Rob96]). There seems to be also an upper limit for the categoric perception of the distance between consecutive events. For event distances larger than 2s it become very difficult to perceive a rhythmic context ([Par94b, Rob96]).

Our implementation uses MIDI files as input data; these can be obtained from notation software, by recording an human performance using a MIDI sequencer, from a wave to MIDI converter, or as the result of converting musical data from other formats.

The inaccuracies which should be eliminated during the pre-processing can be separated into three separate types:

1. **Onset times of inexactly played notes** should be merged if they are close enough together, if the amount of moving through the merge operation is small compared to the duration of the played note, and if the size of the durational overlap is nearly the complete duration of at least one note. Because two onset times which are closer together as 50ms will be perceived as simultaneous the threshold for the detection of equal onset times should be set to approximately 50ms. As new onset time position a weighted average of all merged onset times can be calculated. Because longer notes often are played more accurately than shorter notes, the note duration should be used as weight for the average calculation also.

2. **Small overlaps caused by legato playing** should be removed by cutting the duration of a note to the onset time of a following note (not necessarily the direct successor) if its onset time positions is close to the offset of the earlier note, and if the size of the cut duration is small compared to the complete note duration.

3. **Fill small gaps caused by inexact duration playing** should be removed by increasing the duration of a note up to the onset time of a following note if this onset time is the earliest onset time close to the end of the earlier note, if the size of the gap is small compared to the duration of the note, and if no overlap with any other note will be caused by expanding the note. During the development of the proposed algorithm it turned out that it is not necessary to fill these gaps during the pre-processing. Using an intelligent quantisation approach, these small gaps can be removed also during quantisation. Removing them already at the pre-processing level in our experience does not improve the output quality significantly.

Sustain Pedal Duration

sustain pedal duration If an input MIDI file was generated by a piano performance – using a MIDI piano instrument or electronic keyboard – it is possible that the data contains not only explicit note duration information (as indicated by note-on and note-off events) but also implicit duration information given by the recorded sustain pedal events (down, up) of the instrument. As long as the sustain pedal of an acoustic piano is pressed down all key release information will be ignored until the sustain pedal will be released (up) again. A release of the pedal will stop the sound (duration) of all notes with a previously ignored key-up event during the sustain (pedal = down) phase. Keys/notes which were not released during the sustain phase will not be stopped by releasing the sustain pedal, they will sound until the corresponding piano key will be released by the player.

A MIDI file will contain the note-on and note-off information corresponding to the key actions and additionally also MIDI controller information about the press down and release of the sustain pedal (see Figure 1.7). If an algorithm for analysing MIDI data simply ignores the sustain information, notes

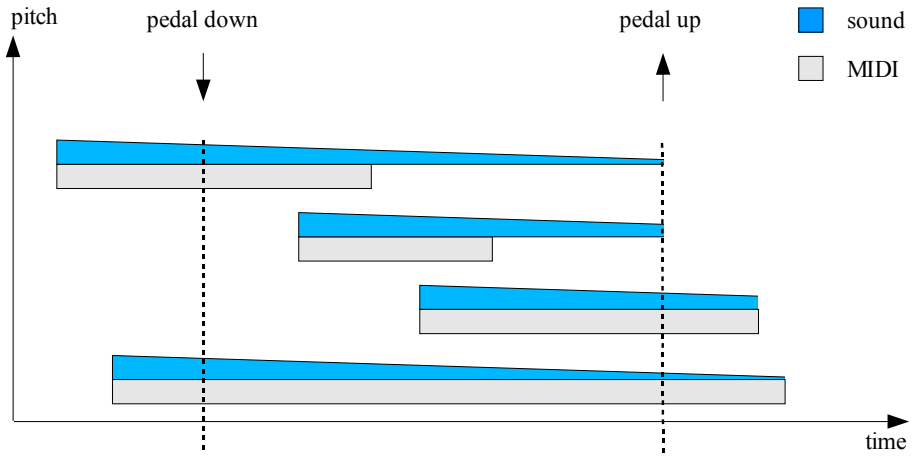


Figure 1.7: Explicit MIDI note-on/note-off duration and sustain pedal duration. The grey rectangles represent MIDI notes and the attached wedges the intensity decay of the corresponding audio sound of an decaying source like a piano string.

in sustained passages may be transcribed with very short (wrong) durations – significantly different from the original sound of the performance. This will lead to an incorrect sound (for playback of the transcription) and can also cause errors in quantisation and even more in voice separation. It seems that no other existing transcription system distinguishes between the key press duration and the sustain pedal

duration of notes. Especially the Melisma system (see Section 1.3) does not evaluate this information for voice separation and beat tracking.

When playing on a real piano or software piano the intensity of a note will decay during a sustain phase and it could be also be 'hidden' by other notes with a close pitch or higher intensity. So the actual duration as perceived by a listener is different (shorter) than the duration given by the sustain events. A naive approach of expanding all note durations to the duration as indicated by the sustain information would result in a large number of overlapping notes which would increase the complexity of the voice separation significantly and also interfere with the operation of the quantisation module. To avoid the large numbers of undesired overlapping notes caused by sustain events – which also might have been played imprecisely – it turned out that an analysis algorithm can cut the duration of sustained notes at the onset time position of the following note in a certain interval range. A first idea would be a very small interval range of two semitone steps up and down. But by analysing existing scores and MIDI examples it turned out that this range can be extended up to an octave without negative side effects. The octave was also chosen because overlapping notes with intervals smaller than an octave usually can be played correctly with one hand and without using the sustain pedal (explicit duration). As shown in Figure 1.8 only the additional sustain duration of a note can be cut automatically by a following note, Explicit durations indicated by note-on and note-off events will never be cut by following notes and can only slightly be modified if they have been detected as short legato overlaps. If an input MIDI files was generated from an audio-to-MIDI device the perceived sustain information will already be converted into explicit note duration including perceived overlappings which should be nearly equivalent to the resulting durations of the sustain cut approach. It is not clear if an audio-to-MIDI converter can successfully infer sustain pedal information.

1.5 A Test Library

For testing, evaluating, and comparing the output of different quantisation and tempo detection approaches – or general transcription systems – of different authors a general test library with a set of MIDI files of different styles and type (*e.g.,* live performed, mechanical performance, multi voice, single voice) would be very useful. Because of the very time consuming manual work for evaluating the correctness of a system's output the test library should include a small core set of pieces addressing different *hard* features for transcription approaches. Some of the in the following proposed pieces are already publicly available as performance files in the context of existing transcription systems (*e.g.,* Melisma), others are available on the world wide web but only as mechanical performances. With our proposal we aim to start a discussion between researchers in the area of computer aided transcription, which will hopefully result in some contributions to a general, publicly available test database of performance files.

In the context of this thesis project we collected a small test library including examples from other works and new recorded files. The test library covers different kinds of styles, and pieces with different complexity and different features (*e.g.,* monophonic, polyphonic, key changes, time signature changes). We also decided that this test library should contain pieces of different time periods of music (*e.g.,* Bach, Mozart, Chopin, Bartók), different genres (*e.g.,* traditional, Jazz, contemporary), pieces for different instruments and different types of ensembles (*e.g.,* piano music, guitar, monophonic solo instruments, quartet).

In the following a description of specific pieces that we used for the evaluation of our system:

- Bach *Inventios*, *Sonatas*, and content of *Well Tempered Clavier* – Because of the homogeneous note durations (in many cases a merge operation of all onset times creates a regular sixteenth note grid) these files should be rather simple for tempo detection approaches. Because of the larger number of ornaments in combination with short melody notes the files are of interest for the evaluation of ornament detection approaches. Especially because the *Well Tempered Clavier* books consists of pieces using all possible key signatures these files are of interest for the evaluation of key detection approaches.
- Bach *Minuet in G* – A performance of the first part of this rather simple piece is part of the distribution of Temperley's Melisma system.

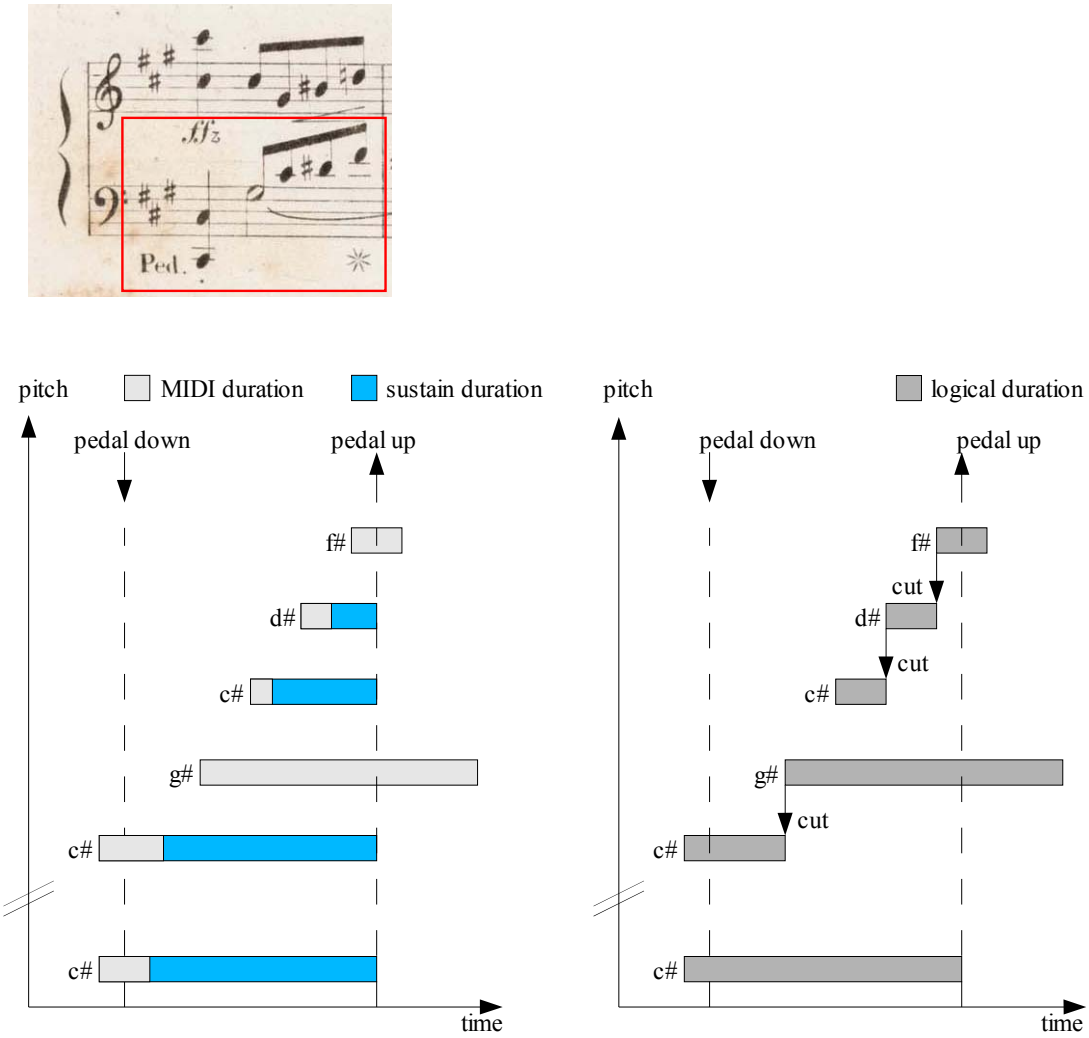


Figure 1.8: Chopin, *Op. 6 Mazurka 1*, measure 19: score (top), performance data for left hand (bottom left), and logical data for left hand (bottom right).

- Beethoven *Sonata Nr. 20 in G, Op. 49, 2* - We used this composition for the evaluation of our tempo detection and quantisation approach because it contains a large number of sudden changes between binary and ternary timing (*i.e.,* quavers and triplet quavers). Several performance files of this piece are part of a collection we could obtain from Haruto Takeda, who used these files for evaluation of the approach proposed in [TNS03].
- Chopin *Mazurkas* – These compositions (we used Op. 6, Nr. 1 and Op. 67, Nr. 2) are hard for tempo detection approaches because they contain complex rhythmical structures and syncopations and are also supposed to be played in an expressive style containing many tempo fluctuations.
- Bartók *Mikrokosmos* – Because of their non-standard rhythmical structure and tonality (we used *Mikrokosmos 101, 133*, and *144*) these files are of interest for the evaluation of key detection, pitch spelling, and also time signature detection approaches.
- Brubeck *Take Five* – Because of its rather uncommon $\frac{5}{4}$ time signature this piece is of interest for evaluating time signature detection approaches. But also its swing style is of interest for the evaluation of quantisation approaches.

- Mozart *Clarinet Quintet K581* – The scores of the different parts include different time and key signatures and are therefore of interest for approaches addressing this features. It can also be assumed that an ensemble performance includes other types of tempo fluctuations than solo performances which might be of interest for evaluating tempo detection approaches. Unfortunately we could not obtain a live performed MIDI files of this pieces for this kind of evaluation.

Clearly, this selection is only a first starting point for a general test library which should be completed by 'worst case' files proposed by other authors. Chew, for example, proposes in [CC03] Beethoven's *Piano Sonata No. 30 in E Major, Op. 109.* as a challenging piece for pitch spelling algorithms.

Beside the decision and agreement on a general test library it must be decided which file format should be used for this library. As shown in the Section 1.4.2, the MIDI format is currently the most common used file format for this type of performance files. Because MIDI does not allow to store arbitrary score level information (*e.g.,* exact pitch spelling, slurs) in an adequate way inside a MIDI file, this seems not be the best file format for a general test library. We therefore propose the use of a file format that allows to store arbitrary performance level information in combination with arbitrary score level information in an adequate and native way. Here the Low-Level GUIDO format (see Section 1.4.2 and Section A.10) seems to be a good candidate. It allows to specify arbitrary note durations (as fractions or in milliseconds), arbitrary intensity information (as floats in the range $(0, 1]$) required for performance data, but also exact pitch spelling and other standard score level information (*e.g.,* slur, staccato) required for the evaluation. It also supports adding any other arbitrary information to notes or series of notes by using special (non-standard) GUIDO tags. This information could be used for partly automatic or fully automatic evaluation of transcription approaches. Using for example, a (non-standard) \scoreInfo tag for each note in a Low-Level GUIDO file (*e.g.,* \scoreInfo<pos=1/4, dur=1/8>(c2*123ms)) the correctness of the output of a tempo detection implementation could be evaluated automatically. For the evaluation of systems that are using only MIDI files or note list ASCII files as input, the Low-Level GUIDO can easily be converted these file types by using already existing GUIDO tools (gmn2midi) or by implementing new tools using the existing GUIDO parser kit and/or the leanGUIDO class library.[12]

[12]The leanGUIDO class library can be obtained at the GUIDO homepage: `http://www.salieri.org/GUIDO`.

2 Voice Separation

Voice separation, along with tempo detection and quantisation, is one of the core problems of computer aided transcription of music. An adequate separation of notes into different voices is crucial for obtaining readable and usable scores from performances of polyphonic music recorded on keyboard (or other polyphonic) instruments; for improving quantisation and key detection results within a transcription system (see Chapter 5, Section 6.2); and in the context of music retrieval systems that primarily support monophonic queries. In this chapter we propose a voice separation algorithm based on a stochastic local search method.[1]

Different from many previous approaches, our algorithm allows and is able to detect chords in the individual voices; its behaviour is controlled by a small number of intuitive and musically motivated parameters; and it is fast enough to allow interactive optimisation of the result by adjusting the parameters in real-time. We demonstrate that compared to existing approaches, our new algorithm generates better solutions for a number of typical voice separation problems. We also show how by changing its parameters it is possible to create score output suitable for different needs (*e.g.,* piano-style *vs* orchestral scores).

In the remainder of this chapter we first give an overview about existing approaches for voice separation and describe then in Section 2.2 the details of our heuristic approach.

2.1 Existing Approaches

Beside approaches which address the issue of auditory scene analysis in a general way by focussing on psychological and physical aspects of human perception (*e.g.,* [Bre90, MD97]) there exist various approaches for finding good voice separations of a given input piece, which have been proposed in the literature and/or are used in practice. Most commercial sequencer software products implement the extremely simple split point separation technique, while more complex approaches appear to be only implemented in academic systems, such as Temperley and Sleator's Melisma Music Analyzer [TS] (see also Section 1.3).

2.1.1 Split Point Separation

One of the simplest methods for voice separation is to split the range of all possible pitches into a number of disjoint intervals and to assign notes to voices depending on which pitch range they fall into. For two voices, this is achieved by fixing one pitch as a split point and by assigning all notes with equal or higher pitch to the first voice, while all notes with a lower pitch are assigned to the second voice. In general, for separating a piece into k voices, $k-1$ split points are used to define the k respective pitch ranges. This approach is easy to implement and is used in most commercial sequencer software packages; however, it works correctly only for pieces in there are no overlaps in the ranges of different voices and will produce errors if this condition is not satisfied (see Figure 2.1(a)).

2.1.2 Rule-Based Approaches

Because composers/arrangers are typically using voice-leading rules when composing or arranging musical pieces, it is possible to use the same rules for the inverse task of separating notes into different musical

[1] The main part of this chapter is based on [KH02].

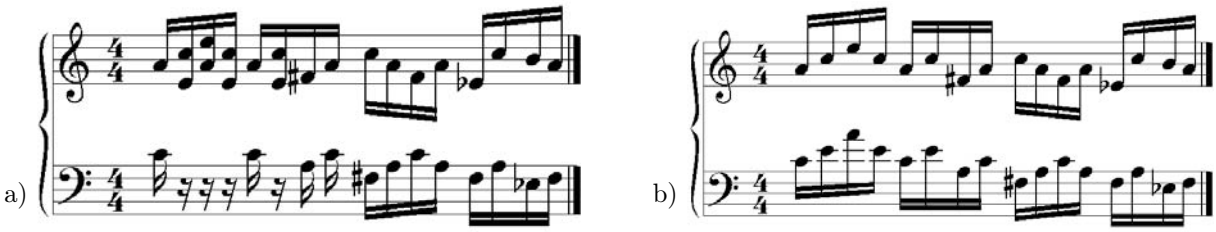


Figure 2.1: Voice separation for a piano style piece: a) with an approach using a fixed split point and b) with our new approach.

voices. At first glance this rules seem just give respect to the limitations of specific instruments or the human voice (*e.g.,* range), but studies of human perception of music also show that if melodies are following such rules they become easier the perceive by human listeners [Bre90]. Examples of voice leading rules include the following:

(1) prefer small intervals between succeessive notes

(2) keep range (ambitus) of a voice small

(3) use a small number of voices

(4) avoid crossings of voices

(5) avoid large gaps within a voice

For an elaborated, scientific description of voice leading rules please refer to [Hur01]. Because there exist many such rules, including 'weak' rules which depend on the musical context, this approach typically results in incorrect voice separations for pieces with more than two voices or with a changing number of voices. An implementation based mainly on the 'nearest path rule' (1) is described by Cambouropoulos in [Cam00b]. The nearest path rule implementation in Fermata [Kil96] showed that the results are mostly correct if the input data can be cleanly separated into a fixed number of continuous voices for the complete piece. If, however, the number of active voices changes during a piece or the pitch ranges of two voices show major overlaps, this approach often produces erroneous separations. Improved performance can be achieved by segmenting the given piece into fragments during each of which the number of active voices remains constant, but unfortunately, finding such segmentations can be difficult.

As the number of rules increases, finding optimal voice separations with respect to a given rule system becomes a non-trivial task. Temperley and Sleator solved this problem by using a dynamic programming (DP) approach [Tem01, TS]. Their Melisma Music Analyzer includes a contrapuntal analysis module (named s*treamer*) which is based on such a preference rule system. Different from our approach, the focus of Temperley and Sleator is more on the correct musicological analysis than on creating reasonable and flexible score-notation. One major difference from our approach is the fact that their system does not detect chords. For the analysis here the piece is divided into so-called 'columns' (similar to the slices used in our approach) based on the metrical structure of the piece. For each column then all possible voice separations are constructed. A globally optimised set of selections (one separation per column) is then retrieved by using DP. The use of an optimisation approach based on DP results in hard constraints for the type of cost function c, which calculates the costs for a transition between two columns s_i and s_{i+1}. For calculating $c(s_i, s_{i+1})$ only the local context of these two states can be evaluated. If $c(s_i, s_{i+1})$ is based on general information about the best path from s_1 to s_{i-1}, it cannot longer be guaranteed that the DP retrieves a global best solution.
Using DP for a column-based voice separation also requires the calculation of the complete set of possible solutions (a dedicated voice for each note in s_i) for each column s_i, which requires that the maximum number of possible voices is limited. Because the rules used within the Melisma Analyser strictly forbid the crossing of voices and also do not allow chords, there the number of possible distributions for a single column with n notes is in $O(v_{\max} \cdot n)$, where $v_{\max}$ denotes the maximum number of possible voices .

By allowing chords and especially inexactly played chords the generic construction of all possible voice separations for a column (or slice) turns into a non-trivial and time consuming task. With n notes and a small number of only two allowed voices ($v_{\max} = 2$) the number of possible chords is already in $O(2^n)$. For a larger number of voices it grows exponentially in the number of notes of the column and the number of allowed voices.

2.2 A Parametrised Heuristic Approach

The main focus of our voice separation approach is on the creation of voice separations for various needs, particularly as arising in score generation and notation tasks as well as in the context of music information retrieval. Hence, the goal is not to find 'the correct' voice separation, which could hardly be defined without making restrictive assumptions about musical style and would be difficult to capture accurately even in the presence of such assumptions (see Section 1.2).

> "However, in nineteenth and twentieth century piano music, the voicing is often ambiguous and not explicitly represented in the score." [MLW03]

Rather, we pursue the more pragmatic goal of creating an adequate algorithm that is capable of finding a range of voice separations that can be seen as reasonable solutions in the context of different types of score notation (*e.g.,* only monophonic voices, only one voice including chords, multiple voices and chords; see Figure 2.2). The underlying idea is to allow a user by controlling a small number of intuitive parameters of the algorithm in real-time to interactively find a voice separation for a given piece that suits her specific but not necessarily explicitly known needs, and requires only minimal manual modifications in order to obtain a satisfying result. Our approach splits a complete piece into small slices of overlapping notes. These slices are processed iteratively, assigning each note of a slice to a voice. During this process, chords can be created by grouping multiple notes from a slice to a chord and assigning them to the same voice. A randomised local search algorithm is used for finding assignments that minimise a parametric cost function which is used to assess the quality of partial voice separations. This cost function includes components that reflect the relationship between the notes within a slice (possible chord groupings) as well as between a slice and the partial voice assignment of previously processed slices (voice leading).

The task of assigning the notes of a slice to a number of existing voices becomes challenging if the number of notes differs from the number of voices available at that point. If there are more notes within the slice than voices, new voices need to be created, or chords have to be introduced in one or several voices. Our approach works with a maximum number of possible voices that can either be specified by the user or derived from the maximum number of notes overlapping at any point in time. (It may be noted that we pre-process the input data prior to performing voice separation in order to deal with certain types of inaccuracies and noise; this is be described in more detail in Section 1.4.3.) However, voices are only used when required and – depending on the parameter setting for the cost function – the result of the voice separation process will not always utilise the maximum number of voices.

An empirical evaluation of our algorithm shows that by using different (user accessible) settings for the parameters of the cost functions, a range of reasonable voice separations can be obtained for a given input. In many cases that are known to be problematic for other voice separation methods, such as static split-point separation or the approach described by Temperley [Tem01, TS], our algorithm achieves good results. In other cases, including randomly generated music, pieces with a high degree of pitch overlaps between voices, or pieces without any voice structure, our method encounters similar problems as other approaches.

2.2.1 Definitions

In the following we assume that the piece of music for which voice separation is to be performed is given in the form of a list of notes sorted by the time positions of their respective onset times. The i-th note m_i in this list is represented by a feature vector

$$m_i = (a, d, p), \tag{2.1}$$

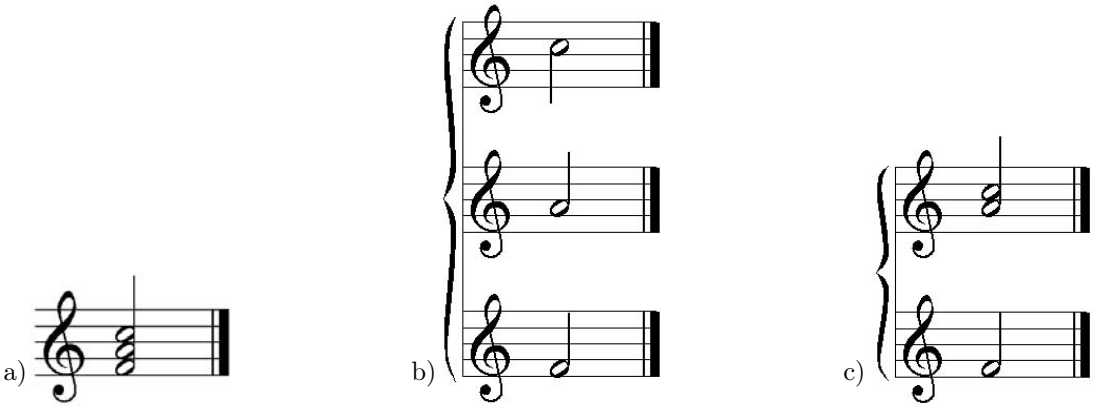


Figure 2.2: Different separations of three notes with equal onset times and equal durations: a) single chord, b) three voices, c) two voices with chord.

where a, d, and p denote onset time, duration, and pitch of note m_i. Depending on the type of input data the timing information can be equivalent to the performed time positions or to score time positions. In the following we assume that all score time positions can be converted into equivalent (absolute) mechanical, performance time positions (see Equation 1.1). We also use $\mathit{onset}(m_i)$, $\mathit{duration}(m_i)$, and $\mathit{pitch}(m_i)$ to refer to a, d, and p for a given note m_i. Likewise, we associate two integer values v and c with each note m_i to represent the voice and chord m_i is currently associated with; we also refer to these values as $\mathit{voice}(m_i)$ and $\mathit{chord}(m_i)$, respectively. Furthermore, we define $\mathit{offset}(m_i) = \mathit{onset}(m_i) + \mathit{duration}(m_i)$, which effectively indicates the endpoint of note m_i. Next, we define some relations between notes that are needed in the following:

$$m_i \leq_V m_j \quad :\Longleftrightarrow \quad \mathit{onset}(m_i) \leq \mathit{onset}(m_j) \tag{2.2}$$

$$m_i =_V m_j \quad :\Longleftrightarrow \quad \mathit{onset}(m_i) = \mathit{onset}(m_j) \tag{2.3}$$

$$m_i >_V m_j \quad :\Longleftrightarrow \quad \mathit{onset}(m_i) > \mathit{onset}(m_j) \tag{2.4}$$

It should be noted that '$=_V$', '$>_V$' and '$\leq_V$' evaluate only the temporal order of the notes, the pitch information is ignored. Furthermore, the function $\mathit{overlap}(m_i, m_j)$ indicates whether notes m_i and m_j overlap in time:

$$\mathit{overlap}(m_i, m_j) \text{ is } \mathit{true} \Longleftrightarrow \\ \big(m_i \leq_V m_j \text{ and } \mathit{offset}(m_i) > \mathit{onset}(m_j)\big) \text{ or } \big(m_i >_V m_j \text{ and } \mathit{offset}(m_i) < \mathit{onset}(m_j)\big) \tag{2.5}$$

Using these definitions, the input of our algorithm can be formally written as $M = (m_1, \ldots, m_{|M|})$ such that for all $i \in \{1, 2, \ldots, |M| - 1\} : m_i \leq_V m_{i+1}$, *i.e.,* as a list of notes sorted in ascending order of their respective onset times.

In the output of our voice separation algorithm, we allow two or more notes with the same onset time only to be assigned to the same voice if they form a chord:

$$\forall\ i \neq j : m_i =_V m_j \quad \Longrightarrow \quad \mathit{voice}(m_i) \neq \mathit{voice}(m_j) \vee \mathit{chord}(m_i) = \mathit{chord}(m_j) \tag{2.6}$$

Because all notes that are grouped to a single chord should belong to the same voice we require:

$$\forall\ i \neq j : \mathit{chord}(m_i) = \mathit{chord}(m_j) \Longrightarrow \mathit{voice}(m_i) = \mathit{voice}(m_j) \tag{2.7}$$

For quantised input data, we restrict chords to consist only of notes with equal onset times. In the case of unquantised input data, onset times may be imprecise (*chord spread* see [DH91]) and hence we have to

allow combining overlapping notes with different onset times into the same chord, but only overlapping notes are allowed to become grouped to a chord:

$$overlap(m_i, m_j) = false \implies chord(m_i) \neq chord(m_j) \tag{2.8}$$
$$chord(m_i) = chord(m_j) \implies overlap(m_i, m_j) = true \tag{2.9}$$

Dixon proposes in [Dix01a] a fixed threshold of a maximum distance of 70ms between successive onset times of chord notes. Because there can always exist chords where the onset times have a distance greater than 70ms (*e.g.,* arpeggio) and we want to avoid fixed thresholds in general, we require here only the overlapping constraint for potential chord notes in first step. By using the proposed optimisation approach then automatically those chord groupings with closer onset times and similar durations will be preferred over those with large differences in the onset times position and duration.

In our implementation, we recognise and eliminate small overlaps and other inaccuracies during a pre-processing phase (see Section 1.4.3 for more details). As a result of these constraints, each voice generated by our algorithm is a sequence of non-overlapping notes and chords.

Before assigning the notes of the input piece M into voices, we will partition M into slices y_i of consecutive overlapping notes $(m_k, \ldots, m_{k+p})$ such that there is an overlap between any pair of notes within each slice and that between any two consecutive slices, y_i and y_{i+1} there are at least two notes that do not overlap. This implies that all notes from y_i except for the one with the smallest offset time may overlap with notes in y_{i+1} (see Figure 2.3 for an example).

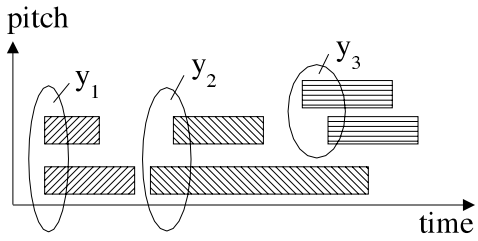


Figure 2.3: Example for partitioning a simple piece into slices.

Mathematically, the partitioning of M into slices can be modelled as follows. First, we define the set B of all indices of notes in M that become the first notes of the slices $y_1, \ldots, y_{|B|}$:

$$\begin{aligned} B := \{\ b_1, \ldots, b_{|B|} \mid \forall\ i \in \{1, \ldots, |B|\} : b_i \in \mathbb{N} \wedge b_1 = 1 \wedge b_{|B|} \leq |M| \wedge \\ \big(\forall\ q, r \in \{b_i, \ldots, b_{i+1} - 1\} : overlap(m_q, m_r) = true\big) \wedge \\ \big(\nexists\ s \geq b_{i+1} : \forall\ p \in \{b_i, \ldots, b_{i+1} - 1\} : overlap(m_p, m_s) = true\big)\ \} \end{aligned} \tag{2.10}$$

Based on B, we can now define the set of all slices y_i:

$$Y := \{\ y_1 \ldots y_{|B|} \mid y_i = (m_{b_i}, \ldots, m_{b_{i+1}-1})\ \} \tag{2.11}$$

This induces the following partitioning of M into slices y_i:

$$M = (\underbrace{m_1, \ldots, m_{b_2-1}}_{y_1}, \underbrace{m_{b_2}, \ldots, m_{b_3-1}}_{y_2}, \ldots, \underbrace{m_{b_{|B|}}, \ldots, m_{|M|}}_{y_{|B|}})$$

We denote a voice separation for a slice $y_i = (m_{b_i}, \ldots, m_{b_{i+1}-1})$ by $S(y_i) = (s_{i,1}, \ldots, s_{i,z})$ where $z = |y_i|$ and $s_{i,j} = (voice(m_{b_i+j-1}),\ chord(m_{b_i+j-1}))$ represents the voice and chord that the j-th note of slice y_i is assigned to under separation $S(y_i)$. As an abbreviation, we use $S_i := S(y_i)$. Furthermore, we use S_i^* to denote the set of all possible voice separations $S(y_i)$ for slice y_i. Any voice separation $\mathcal{S}$ for the complete input piece corresponds to the combinations of voice separations for all slices y_i, *i.e.,* $\mathcal{S} = (S_1, \ldots, S_{|B|})$, and the set of all possible voice separations of M is denoted $\mathcal{S}^*$.

The number of different voice separations for a single slice y_i. *i.e.,* the size of the set S_i^*, depends on the number of notes in y_i, z and on the maximum number of voices in the desired output of our separation algorithm, $nVoices$. More precisely, in the worst case, in which any subset of notes in y_i can be combined into a chord, there are at least $nVoices^z$ possible voice separations of slice y_i. Hence, in the worst case, the number of possible voice separations of a given input piece M is exponential in the number of notes in M.

This suggests that in order to find voice separations that are optimal with respect to a given set of criteria (which will be more precisely defined in the next section), the naive method of enumerating all possible separations and selecting the best of these can be prohibitively expensive, especially when the goal is to allow the user to interactively tune the parameters of the voice separation process in order to obtain a desired output. Consequently, our voice separation algorithm uses a substantially more efficient, heuristically guided process for iteratively constructing voice separations using a stochastic local search procedure for optimising the separation of individual slices.

2.2.2 The Voice Separation Algorithm

The main idea underlying our algorithm is to construct a voice separation for the given input piece M from locally optimised separations for the slices of M. This local optimisation is based on a parametric cost function C that assesses the quality of the voice separation of a given slice, S_i, given the separations of all previous slices, *i.e.,* $(S_1, \ldots, S_{i-1})$. The definition of this cost function will be given below.

Given an input piece M and a maximal number of voices $nVoices$, our voice separation algorithm works as follows: After segmenting M into slices $y_1, \ldots, y_{|B|}$ (as described above), a cost-optimised voice separation for the first slice, y_1, is computed (see Figure 2.4). Then, this voice separation is iteratively extended by cost-optimised separations for $y_2, \ldots, y_{|B|}$, resulting in a complete voice separation for M. However, particularly in the case of unquantised input data, this voice separation might contain chords with notes that slightly differ in their onset times or durations as well as overlapping notes within the same voice. Therefore, every time after the voice separation is extended by a slice separation, these situations are resolved by adjusting the durations or onset times. This ensures that in the final result there are no overlaps between notes within any of the voices and all notes within any chord have the same onset time and duration.

```
procedure voiceSeparation(M, k)
    input:
        sorted list of notes M
        maximal number of voices k
    output:
        voice separation S
    segment M into slices y_1, ..., y_|B|
    S := ()
    for i := 1 to |B|
        S_i := separateSlice(y_i, S)
        S := S + S_i
        eliminate overlaps within voices of S
        and regularise chords where required
    end
end
```

Figure 2.4: Outline of our voice separation algorithm; for unquantised input data, this procedure is called after removing inaccuracies and noise in the input piece M.

In the following, we describe the cost function and the cost-optimising voice separation of slices in more detail.

2.2.3 The Cost Function

The cost function C used for assessing and optimising the quality of a voice separation S_i of a slice y_i, given separations $S_1, \ldots, S_{i-1}$ for all previous slices, is a weighted sum of terms that penalise individual,

undesirable features:

$$C(S_i, \mathcal{S}) \quad = \quad p_{pitch} C_{pitch}(S_i, \mathcal{S}) + p_{gap} C_{gap}(S_i, \mathcal{S}) + p_{chord} C_{chord}(S_i) + p_{ovl} C_{ovl}(S_i, \mathcal{S}) \qquad (2.12)$$

Here, $\mathcal{S}$ denotes the partial voice separation $(S_1, \ldots, S_{i-1})$ (see Figure 2.4). Intuitively, C_{pitch} and C_{gap} penalise large pitch intervals and gaps (rests) between successive notes in a voice, respectively; C_{chord} penalises chords with a large pitch interval between the highest and the lowest note, as well as irregular chords containing notes with different onset times or durations; and C_{ovl} penalises overlaps between successive notes in the same voice. By adjusting the weights of these terms, different trade-offs between these features can be achieved, leading to qualitatively different voice separations (chordal, single voices, etc.; see Figure 2.2 and Figure 2.5). In the following, we describe in detail how the four penalty terms are calculated.

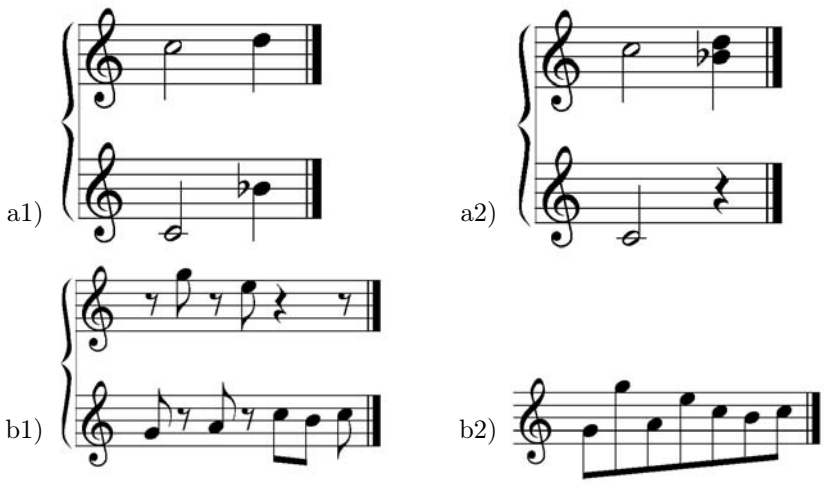


Figure 2.5: Different separations of non-overlapping notes: a1) $p_{chord} \gg p_{pitch}$, a2) $p_{pitch} \gg p_{chord}$, b1) $p_{pitch} \gg p_{gap}$, b2) $p_{gap} \gg p_{pitch}$; Example (b) is also discussed in [Gje94], p. 350.

Pitch Distance Penalty C_{pitch}

The segregation of multiple melodic lines by human listeners depends very strongly on the frequency distribution and separation of the melodies [Bre90]. Consequently, it makes sense to use similar features in the context of automatic voice separation. The pitch distance penalty increases with the interval size between two successive notes in a voice. For the first note of a voice, a fixed penalty is imposed for starting a new voice. In some cases (melodies including short sequences of large intervals), a 'lookback mechanism' can be advantageous by which the pitch at the end of an existing voice is calculated from the average pitch of the last n notes in the respective voice. This mechanism behaves somewhat similar to the approach of Gjerdingen [Gje94] in which the motion-tracking system moves with some delay from a current pitch to the pitch of an incoming note. Only if the incoming note is long enough, the motion tracker reaches the exact pitch level of that note and stays there.

If a note m_j is assigned to a chord c, we define the pitch distance (interval size) between note m_j and pitch p as the pitch distance to the note of the chord c which is closest in pitch to p_l. Therefore we define a function $cPitch(m_j, p)$ which returns $pitch(m_j)$ if m_j is a non-chord note and otherwise returns the pitch of that chord note of c, which is closest to p. If no pitch lookback is used, the pitch distance between p and an existing voice v – when comparing to pitch p – is defined as

$$cPitch(v, p) = cPitch(\arg\max_{m \in v}\{onset(m)\}, p), \qquad (2.13)$$

i.e., the pitch distance between p and the latest note in voice v.[2] If pitch lookback is used, the average pitch at the current end of voice v is calculated as shown in Figure 2.6; $prev(m_j, p_l)$ denotes the note directly preceding note m_j in the same voice as m_j and not assigned to the same chord as m_j. If the directly preceding note is assigned to a chord, the chord note which is closest to pitch p_l returned instead *i.e.,* $prev(m_j, p_l) := m_k$ where k is the largest value such that $m_k < m_j$ with $voice(m_k) = voice(m_j)$ and $chord(m_k) \neq chord(m_j)$. The weighting of 0.8 for the current pitch and 0.2 for the previous pitch that is used in this calculation was found empirically to give good results for the tested input data. This relation can be adjusted be using the `SPLITVOICEDECAY` setting of `midi2gmn` (see Section A.3). .

function cPitch(v, l, p)
  input:
    voice index v, lookback size l
    pitch p of note j
  output:
    average pitch p' of voice v
    for comparison to p
  $prevNote := prev(lOnset(v), p)$
  $p' := cPitch(lOnset(v), p)$
  $i := 1$
  while $i \leq l$
    $p' := 0.8 \cdot p + 0.2 \cdot cPitch(prevNote, p)$
    $prevNote := prev(prevNote)$
    $i := i + 1$
  end
  return(p')
end

Figure 2.6: Pitch calculation for voice v with pitch lookback $l > 0$, $lOnset(v)$ denotes the latest note currently attached to voice v.

The pitch distance penalty C_{pitch} for a single voice can then be calculated as shown in Figure 2.7. Different from the implementation showed in [KH02] where the a linear interval penalty function was used ($pDist$ = interval/128), the current version now uses the non-linear Gaussian window function.[3] By adjusting its σ parameter to a value of approximately twelve semitone steps the penalty for unusual, large intervals greater than an octave will be significantly higher than for small intervals. The optimisation algorithm will then prefer a set of medium size intervals in all voices of a slice against a solution with very small intervals in some voices and very large intervals in one or several other voices. Based on the C_{pitch} values for each voice v, the overall pitch distance penalty for a complete slice separation S_i can be calculated as shown in Figure 2.8.

Gap Distance Penalty C_{gap}

Studies of human perception of music showed that melodies with few and short rests are perceived more easily as a coherent melodic line by an human listener than melodies with many long rests [Bre90]. The structure of many melodic lines in Western music is consistent with this observation. Therefore, we impose a gap penalty if adding a note from the current slice to a voice introduces a rest; furthermore, the penalty increases with the duration of the rest. If the added note m is the first of the respective voice, the time difference between time position zero, (*i.e.,* the onset time of the first note in M), and

[2] $m \in v$ should denote the set of notes with $voice(m) = v$.
[3] The details of the Gaussian window function are described in Section A.9.

```
function C_pitch(S_i, S, v)
    input:
        slice separation S_i
        separation S for previous slices
        voice index v
    output:
        pitch distance pvD
    prevNote := prev(v, pitch(m_j))
    pvD := 0;
    for all m_j ∈ S_i with voice(m_j) = v do
        iv := cPitch(prevNote, pitch(m_j)) − pitch(m_j))
        pDist := W_Gauss(iv, σ)
        pvD := pvD + (1 − pvD) · pDist
        if chord(prevNote) ≠ chord(m_j) then
            prevNote := m_j
        end
    end
    return(pvD)
end
```

Figure 2.7: Calculation of C_{pitch} for a single voice v in S_i

```
function C_pitch(S_i, S)
    input:
        slice separation S_i
        voice separation S for previous slices
    output:
        pitch distance penalty pD
    pD := 0;
    for all v used in S_i do
        pD := pD + (1 − pD) · C_pitch(S_i, S, v)
    end
    return(pd)
end
```

Figure 2.8: Calculation of pitch distance penalty C_{pitch} for slice separation S_i given separation S for previous slices.

the onset time of m is penalised. Because all notes in a slice y are overlapping each other, gaps between notes within y cannot occur.

In the original version of the here described voice separation approach ([KH02]) the gap distance penalty C_{gap} for a single note m and a voice v was defined as the length of the gap introduced in voice v by adding m divided by the maximal gap length introduced by adding m to any voice in S.
Further development and improvements of the implementation showed, that this calculation of the gap distance penalty is not robust against cases where there are small gaps between a note m and all currently existing voices. For the voice with a resulting maximal gap to m the penalty would be the maximum penalty of 1, independent from the actual size of the gap. Even if this voice would have only a small penalty for the other voice separation features, it would have very few chances to be selected as the best

solution for m.
In the current implementation of `midi2gmn` now the length of gaps between two notes m_j and m_i measured in milliseconds is evaluated for the gap distance penalty and normalised to the interval $[0, 1)$ by using a Gaussian window function:

$$c_{gap}(m_i, m_j) := 1 - W_{Gauss}(gap(m_i, m_j), \sigma), \tag{2.14}$$

where $gap(m_i, m_j)$ calculates the distance in seconds between the endpoint of note m_i and the onset time of note m_j. If $m_i \geq m_j$ or $overlap(m_i, m_j) = true$ then $gap(m_i, m_j)$ returns 0. In the current implementation σ is set to 1.9s. Based on this measure, the overall gap distance penalty for S_i and S can be calculated as shown in Figure 2.9, where the note-voice penalty function $C_{gap}(m, v)$ is defined as

$$C_{gap}(m, v) := c_{gap}(m, \arg\max_{m \in v}\{offset(m)\}) \tag{2.15}$$

function $C_{gap}(S_i, S)$
  input:
    slice separation S_i
    voice separation S for previous slices
  output:
    gap distance penalty gD
  $gD := 0$
  for all voices v **used in** S_i
    $m :=$ earliest note in S_i with $voice(m) = v$
    $gD := gD + (1 - gD) \cdot C_{gap}(m, v)$
  end
  return(gD)
end

Figure 2.9: Calculation of gap distance penalty C_{gap} for slice separation S_i given separation S for previous slices. See Equation 2.15 for the definition of the note-voice gap penalty function $C_{gap}(m, v)$.

Chord Distance Penalty C_{chord}

Both, the limitations of human physiology in playing chords with very large ranges, (*e.g.,* pitch differences between the highest and lowest note), as well as compositional practice suggest that when combining notes into chords, chords with small ranges should be preferred over chords with large ranges. Furthermore, in most cases, we would expect all notes belonging to the same chord to have identical or very similar onset times and durations (note that we allow the grouping of notes with different onset times into the same chord only for unquantised input data) and also usually – if played by a single, polyphonic instrument – similar intensities. Hence, we use a chord distance penalty that increases with the range of a chord, with the differences in durations and intensity[4] of its notes, and with the distance between the respective onset times (in the case of unquantised data).

Based on these considerations, the following penalty terms are used as components of the overall chord distance penalty for a given slice separation S_i: The range penalty *pRange* for a chord c is defined as $pRange(c) = 1 - W_{k\text{-}Gauss}(range(c), 12, 2)$,[5] where $range(c)$ is the pitch difference between the lowest and the highest note in c (measured in semitones). Because of the saturation characteristic of the k-Gaussian function all chords with a range above one octave will get over proportional high penalties.

The penalty for differences in the onset time position and in the durations of the notes of a chord c can be calculated by evaluating the earliest onset time the latest onset time, the earliest endpoint, and the

[4]The implementation described in [KH02] did not evaluate intensity features.
[5]Please see Section A.9 for description of the higher order Gaussian window function $W_{k\text{-}Gauss}$.

latest endpoint of all notes of c:

$$d_{\min}(c) = \min_{m \in c}\{offset(m)\} - \max_{m' \in c}\{onset(m')\} \tag{2.16}$$

$$d_{\max}(c) = \max_{m \in c}\{offset(m)\} - \min_{m' \in c}\{onset(m')\} \tag{2.17}$$

$$r_d(c) = d_{\min}(c)/d_{\max}(c) \tag{2.18}$$

Because only groups of overlapping notes can become a chord $d_{\min}(c) > 0$ and $d_{\max}(c) \geq d_{\min}(c)$ will always be true. For a mechanical chord $d_{\min}(c) = d_{\max}(c)$ will be true. The duration penalty for a chord can now be defined as

$$pDuration(c) := 1 - W_{Gauss}(\log r_d(c), \sigma). \tag{2.19}$$

In the current implementation σ is set to 2.0.

Finally, with $i_{\min}, i_{\max}$ are the minimum and maximum intensity of all notes of chord c, the intensity penalty for a chord c can be calculated as

$$pIntens(c) := 1 - W_{Gauss}(i_{\min}, i_{\max}, \sigma). \tag{2.20}$$

We assume that the intensity values are given as floats in range (0:1], where 1 indicates a maximum intensity (see also Section 6.5). If σ is set to a value > 1 (the maximum possible distance between $i_{\min}$ and $i_{\max}$, the range of *pIntens* can be limited and so its influence to the overall chord distance penalty decreased.

Based on these three penalty terms for individual chords, the overall chord distance penalty for a complete slice separation S_i is calculated as shown in Figure 2.10. This particular way of combining the penalty terms is chosen to ensure that if one of the terms is large, the overall chord penalty is large as well; note that by using a (weighted) arithmetic average of the three penalty terms, this property cannot be guaranteed (see also Figure A.16).

```
function C_chord(S_i)
    input:
        slice separation S_i
    output:
        chord distance penalty cD
    cD := 0;
    for all chords c in S_i do
        p := pRange(c)
        p := p + (1 − p) · pDuration(c)
        p := p + (1 − p) · pIntens(c)
        cD := cD + (1 − cD) · p
    end
    return(cD)
end
```

Figure 2.10: Calculation of chord distance penalty C_{chord} for slice separation S_i.

It should be noted that the detection of chords during the voice separation here focus only on the grouping of notes to chords. By using approaches for tonal analysis as described in [Row93] [SMJ89] also their type (*e.g.,* dominant seventh chord in G) or function could be inferred, but tonal and harmonical analysis is not in the scope of this thesis.

Overlap Distance Penalty C_{ovl}

Although notes within a voice generally should not overlap, depending on the instrument and style of music, there are cases in which the notes of a single melodic line are played with substantial overlaps that cannot be removed reliably by pre-processing (see Section 1.4.3), as illustrated in the fingered pedal

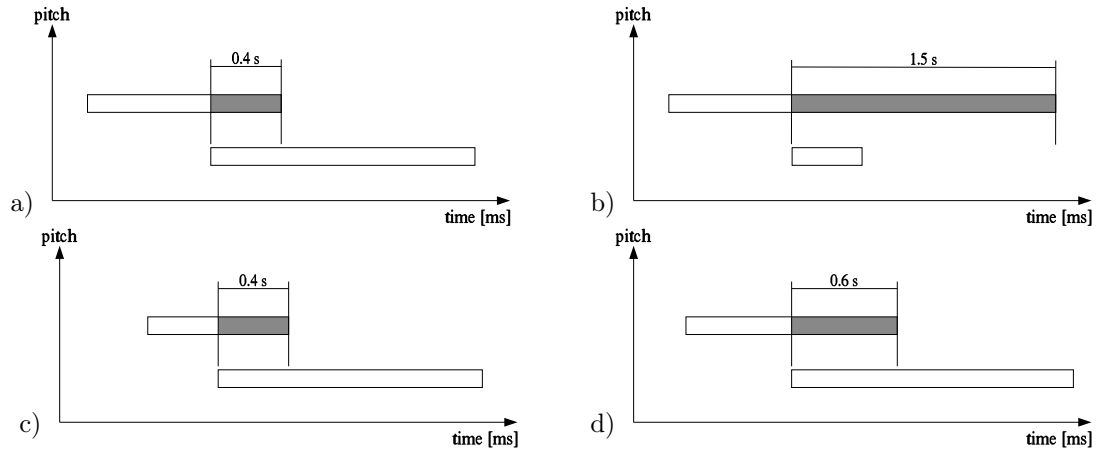


Figure 2.11: Overlap scenarios. Order of desired penalty: b) > d) > a) and b) > c) > a)

example shown in Figure 2.12. Therefore, we allow overlapping notes to be assigned to the same voice without combining them into a chord, but impose a penalty that increases with the amount of overlap. (Note that such overlaps are ultimately eliminated in our algorithm by shortening the duration of the earlier note.)

For two overlapping notes m_i, m_j the size of the overlap (in seconds) is given by:

$$d_{ovl}(m_i, m_j) := \max\{onset(m_i), onset(m_j)\} - \min\{offset(m_i), offset(m_j)\} \tag{2.21}$$

The overlap penalty p_{ovl} for a pair of notes m_i, m_j with $m_i < m_j$ and $voice(m_i) = voice(m_j)$ consists of two components: a penalty p_o for the absolute duration of the overlap and a penalty p_d for the absolute remaining duration of the earlier (cut) note:

$$p_o(m_i, m_j) = 1 - W_{Gauss}(d_{ovl}(m_i, m_j), \sigma_o) \tag{2.22}$$

$$p_d(m_i, m_j) = W_{Gauss}(dur(m_i) - d_{ovl}(m_i, m_j), \sigma_d) \tag{2.23}$$

The final overlap penalty for two notes is then calculated by a multiplication of these two values:

$$C_{ovl}(m_i, m_j) := \begin{cases} p_o(m_i, m_j) \cdot p_d(m_i, m_j), & \text{if } m_i <_V m_j \wedge overlap(m_i, m_j) = true \\ 0, & \text{otherwise} \end{cases} \tag{2.24}$$

Thus, the penalty for an overlap is low if the duration of the overlapping region is short or if the remaining duration of the cut note (m_i) is long. If a significant part of the duration of a note must be cut because of an overlap the penalty will be high. Also short notes overlapped by a successive note will get a high overlap penalty.
Cutting notes by overlapping notes is a typical issue for piano – or percussive string instruments in general. Because here the intensity of long notes show a natural decay the behaviour of p_d can also be justified by perceptional reasons. If the intensity has decayed to some degree and new notes are played the decaying note is not longer perceived as a sounding note. Nevertheless, the opened string on the piano will change the overall timbre or sound impression.

In earlier versions of the algorithm ([KH02]) the overlap penalty had only depend on the ratio between overlap and note duration, but not on the actual length (in units of seconds) of the overlap. As shown in Figure 2.11 there exist situations where the intended behaviour of the overlap penalty function requires the evaluation of the absolute overlap length. Otherwise example (b) and (c) would get an equal penalty. Using only the absolute overlap duration is also not possible because then Figure 2.11 (a) and (c) would get equal penalties.

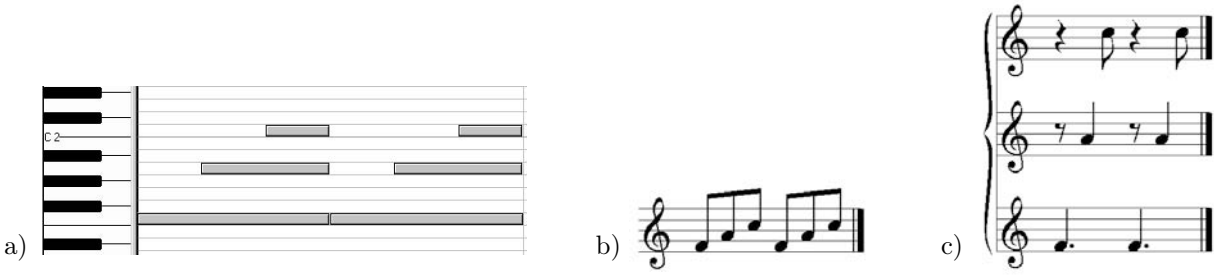


Figure 2.12: Different separations of three overlapping notes. a) input data b) removed overlaps and single voice grouping, c) split into three voices

The overlap distance penalty for a single voice v used in a slice separation S_i and the overall overlap distance for a slice separation S_i are then calculated as shown in Figure 2.13 and Figure 2.14.

function $C_{ovl}(S_i, \mathcal{S}, v)$
  input:
    slice separation S_i
    voice separation $\mathcal{S}$ for previous slices
    voice-ID v
  output:
    overlap distance penalty ovD
  $prevNote := lOn(v)$
  $ovD := 0$
  for all m_j in S_i with $voice(m_j) = v$ **do**
    $oDist := C_{ovl}(prevNote, m_j)$
    $ovD := ovD + (1 - ovD) \cdot oDist$
    if $chord(m_j) \neq chord(prevNote)$ **then**
      $prevNote := m_j$
    end
  end
  return(ovD)
end

Figure 2.13: Calculation of overlap distance penalty for single voice.

2.2.4 Cost-Optimised Slice Separation

Based on the cost function C as defined in Equation 2.12 and given a separation $\mathcal{S}$ of slices $y_1, \ldots, y_{i-1}$, we use a stochastic local search approach for finding a cost-optimised voice separation S_i for slice y_i: Starting with an initial separation $S_i := S_i^0$, a series of randomised iterative improvement steps is performed during each of which one note is reassigned to a different voice. Whenever such step results in an assignment with lower cost than the best assignment seen so far, this assignment and its cost are memorised. This search process is terminated when no such improvement has been achieved for a fixed number θ of steps. In the current implementation, we use $\theta = 3 \cdot |y_i| \cdot nVoices$. Figure 2.15 gives a pseudo-code specification of this randomised iterative improvement procedure.

An initial separation S_i^0 for the given slice S_i is obtained by assigning all notes of y_i to the first voice. During this process, notes with equal onset times are combined into chords. Another natural choice for the initial separation is to distribute all notes in y_i into voices such that overlaps and chords are avoided

```
function C_ovl(S_i, 𝒮)
    input:
        slice separation S_i
        voice separation 𝒮 for previous slices
    output:
        overlap distance penalty ovD
    ovD := 0
    for all v used in S_i
        oDist := C_ovl(S_i, 𝒮, v)
        ovD := ovD + (1 − ovD) · oDist
    end
    return(ovD)
end
```

Figure 2.14: Calculation of overlap distance penalty C_{ovl} for slice separation S_i given separation S for previous slices.

as far as possible. Empirical tests (not reported here) suggested that the former initialisation method leads to better results than the latter approach.

Subsequently, in each local search step we move from the current separation of y_i to a neighbouring separation; two separations S_i and S_i' are neighbours if and only if S_i and S_i' are both valid separations of y_i that differ in the voice and/or chord assignment of exactly one note in y_i. A separation is valid if and only if any notes with identical onset times that are assigned to the same voice are also combined into a chord.

The selection of the actual search step to be performed is based on a randomised greedy choice: with a certain probability (in our implementation, we used a value 0.8 for which we obtained good empirical results),[6] the neighbouring separation with minimal cost is selected, otherwise, a neighbour of the current assignment is selected uniformly at random. The randomisation prevents the search process from getting stuck in local minima of the cost function C.

While there is no theoretical guarantee that this randomised iterative improvement algorithm will find the globally optimal separation for the given slice, it finds optimal or close-to-optimal separations very efficiently in practice. Similar stochastic local search strategies have been very successfully applied to many prominent combinatorial problems (see [Hoo99]).

2.2.5 Implementation

The for the here described voice separation approach input data can be quantised or unquantised low-level musical data. Unquantised data (where the tempo might also be unknown) requires pre-processing to remove inaccuracies that could have detrimental effects on the performance of the voice separation algorithm. As show in Section 1.4.3 very close onset times can be merged to an equal, average onset time before starting the voice separation. Notes with a longer duration or greater intensity can have a 'higher influence' on the calculation of the resulting average onset time. This onset time correction forces our voice separation algorithm to combine the respective notes into chords if they get assigned to the same voice and hence facilitates the recognition of chords.

The four penalty parameters (*i.e.,* $p_{pitch}, p_{gap}, p_{chord}, p_{ovl}$, see Equation 2.12) and the pitch lookback parameter can be defined by the user in an initialisation file (see Section A.3). Parameters for which no value is specified are set to default values predefined in our implementation. The maximum number of voices, *nVoices*, to be used in the final separation of the given piece also can be specified in the initialisation file. If this parameter is not specified by the user, the maximum number of voices is set to the maximum number of overlapping notes at any time position of the input piece.

[6] In the current implementation this probability can be adjusted using the `RWALKTRESH` setting, see Section A.3.

```
function separateSlice(y_i, S)
    input:
        slice y_i
        voice separation S for previous slices
    output:
        optimised selection S_i^opt
    obtain S_i by setting all notes of y_i to voice 0
    and combining all notes with equal onset times
    into chords
    S_i^opt := S_i
    noImpr := 0
    while noImpr < |y_i| · nVoices · 3
        with probability 0.8 do
            S_i := neighbour S'_i of S_i with
                   minimal cost C(S'_i, S)
        otherwise
            S_i := randomly selected neighbour of S_i
        end
        if C(S_i, S) < C(S_i^opt, S) then
            S_i^opt := S_i
            noImpr := 0
        else
            noImpr := noImpr + 1
        end
    end
    return(S_i^opt)
end
```

Figure 2.15: Randomised iterative improvement algorithm for finding a cost-optimised separation for a single slice y_i.

filename	p/q	notes	slices	voices	vlimit	errors
Bach *Minuet in G*	p	104	78	2/2	-	0
Beethoven *Sonata Nr. 20*	p	1630	1109	2	2	2
Chopin *Op. 67, Mazurka 2*	p	164	77	2/5	-	2
Chopin *Op. 6, Mazurka 1*	p	890	550	2	2	12
Bartók *Mikrokosmos 144*	q	717	152	2	2	6
Bartók *Mikrokosmos 133*	q	449	232	2	2	0(3)
Bartók *Mikrokosmos 101*	q	214	135	2	-	0

p/q	performance or pre-quantised data
notes	total number of single notes
slices	number of slices during voice separation
voices	number of used/possible voices
vlimit	possible number of voices ($nVoices$) set by user - = no user limitation
errors	number of errors

Table 2.1: Overview about results of voice separation. The number of errors indicates the number of incorrect separated voice slices.

2.3 Results

We tested our approach on different types of music: most of the inventions by J. S. Bach, some chorals by the same composer, a waltz by F. Chopin, parts of B. Bartók's *Mikrokosmos*, and selected live performed MIDI files (see Table 2.1). For input files containing data already separated into single voice and stored as separate MIDI tracks we merged all tracks into a single track before the evaluation.

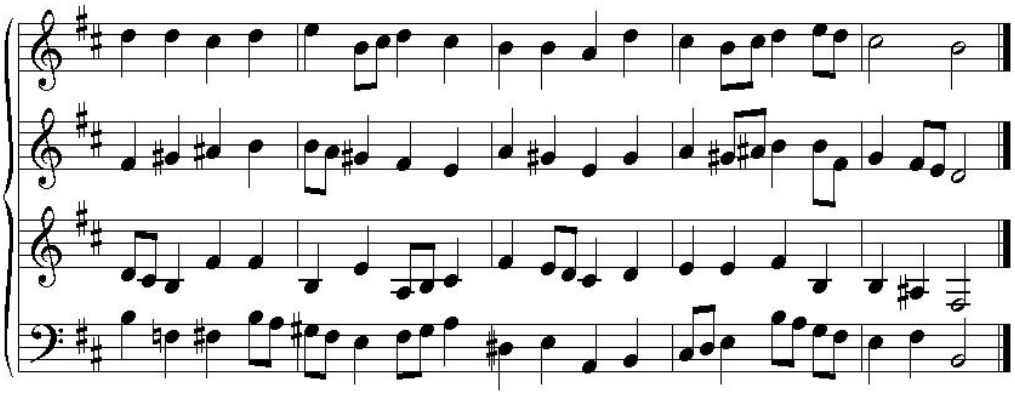


Figure 2.16: Ending of the choral *Mitten wir im Leben sind* by J. S. Bach, BWV 383, separated as a four voice score.

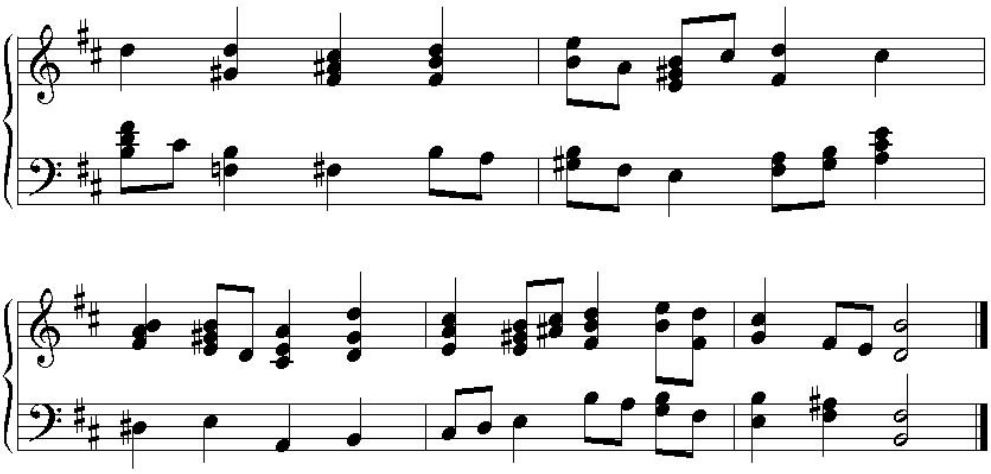


Figure 2.17: Ending of the choral *Mitten wir im Leben sind* by J. S. Bach, BWV 383, separated as a piano-style score.

With the correct parameter settings, the tested Bach chorals and inventions were separated almost entirely correctly (see Figure 2.16 and Figure 2.17). Only in a few cases where the voices nearly meet at the same pitch, localised errors sometimes occurred. With different parameter settings, it was possible to separate the chorals into single voices, into a two staff piano score, or to collect all overlapping notes as chords in a single voice. Figure 2.18 shows a correct separation of a part of a waltz by Chopin obtained from our voice separation algorithm. When using higher chord penalties and lower overlap penalties, however, the same input data is (incorrectly) separated as shown in Figure 2.19.

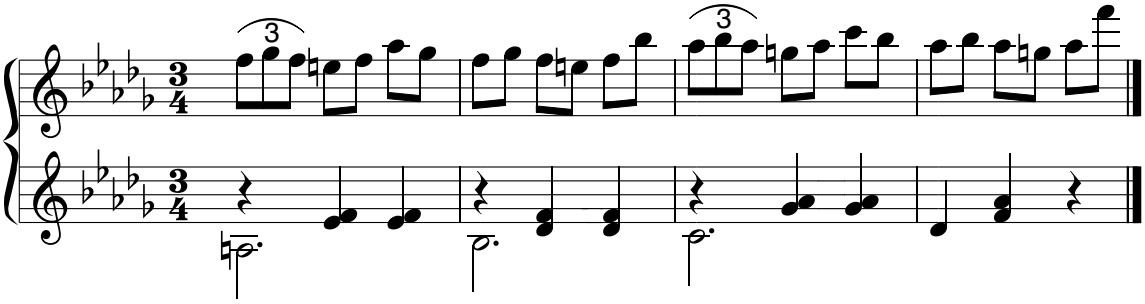


Figure 2.18: Chopin, *Valse*, Opus 64 Nr. 1, measures 101–104.

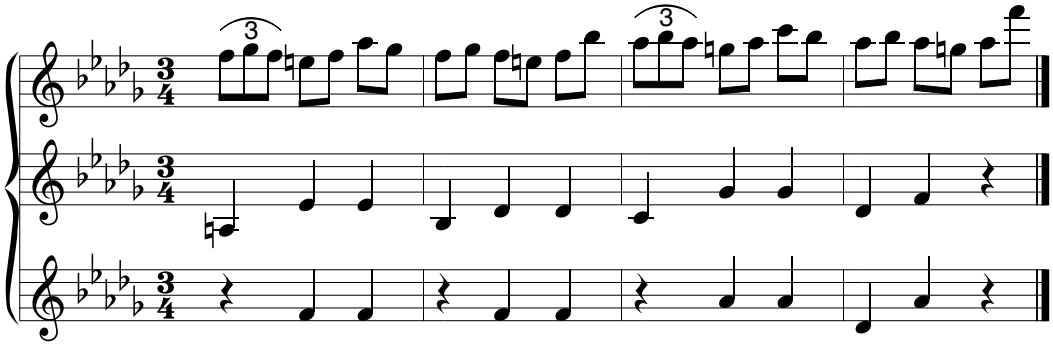


Figure 2.19: Chopin, *Valse*, Opus 64 Nr. 1, measures 101–104, incorrect separation.

Situations in which a voice continues with a large interval step after a rest can lead to incorrect voice separations. In the example of Figure 2.20, the alto and tenor voices pause in the middle of measure 10. In the original score, the alto voice continues at the end of measure 10 and the tenor voice has still tacet. Because the first note of the continuing motive (d') has a smaller pitch distance and a smaller gap distance to the tenor voice (g') than to the alto voice (b') and because there are only three notes to separate into four voices, the algorithm assigns this fragment incorrectly to the tenor voice.

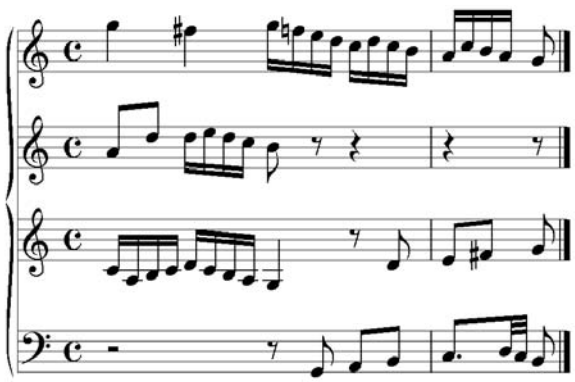


Figure 2.20: J. S. Bach, Well-Tempered Clavier, Book I, *Fugue No. 1 in C Major*, mm. 10–11.

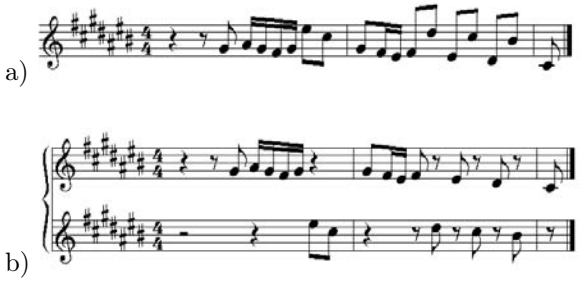


Figure 2.21: J. S. Bach, Well-Tempered Clavier, Book I, *Fugue No. 3 in C♯ Major*, mm. 1–3. a) Correct separation with $pGap > pPitch$. b) Incorrect separation with $pPitch > pGap$.

The same example is also discussed by Temperley [Tem01], whose approach encountered the same problems. We could not improve this result by changing parameter settings. It seems that when only considering the notes, without any knowledge about the composer and style of the piece, there is no reason why another separation should be preferred. This case shows that there exist scores or compositions where at some points the intention of the composer differs from the results obtained by applying standard voice-leading rules or cost functions. In more complex pieces (*e.g., Mikrokosmos 153* by Bartók), the same effect occurs at some positions. In another example (Figure 2.21), we could avoid the problems which Temperley discusses in [Tem01] by using our algorithm with appropriately chosen parameter settings.

3 Similarity And Segmentation

Pattern matching and similarity analysis are two closely related research areas in the domain of musical analysis. In general, the search for approximate instantiations of a given search pattern requires the same techniques than determining the similarity between two arbitrary musical sequences. The segmentation of a musical piece based on structural features and pattern induction is also closely related to the area of similarity analysis. By analysing the similarity between different regions of a performance, multiple repetitions and occurrences of patterns can be detected which then can be used to derive the overall structure of the pieces or induce typical rhythm patterns as well as segment boundaries. Also closely related to this area is score following, where an observed performance should be matched (typically in real-time) to a given score. Here it is assumed that the performance includes inaccuracies (*e.g.,* missing, additional, and incorrect notes; skipped or repeated passages).
In the past years, approaches for similarity analysis also have become of large interest in the context of music information retrieval (MIR). In this domain the general issue of searching for approximate instantiations of a given performance or melodic fragment in large musical databases is determined. Here the content of a database as well as the queries might be given as audio files or as files containing information on a symbolic level.[1]

Especially for symbolic input data – where music can be represented as lists (sequences or strings) of notes – adapted string matching algorithms in combination with dynamic programming have been frequently used in the literature.[2] We agree with Meredith *et al.* ([MLW03]) that almost all string representation of music fall in one of two categories: *event strings* – each symbol represents a musical event (note) and *interval strings* – each symbol represents the interval (or relation) between a feature or a feature vector of two musical events. By using an interval representation the similarity measure between two sequences can be modelled invariant to different tempi or key transpositions.

In [CIR98] Crawford *et al.* give an overview on known string matching related issues and techniques in the context of musical similarity analysis and melodic recognition. They distinguish between two different general categories for pattern matching algorithms and several subcategories:

- Exact-Match Algorithms - these can be applied to input data for which voicing leading information is available. They can be divided into categories of exact matching, matching with deletions, repetition identification, overlapping repetition identification, transformed matching (allowing inversion, retrograde, a combination of both), distributed matching (pattern distributed over several voices), chord recognition, approximate matching, and evolution detection approaches.

- Inexact-Match Algorithms - working on unstructured polyphonic data. These approaches can be divided into methods for unstructured exact matching, unstructured repetition identification, and unstructured approximate matching.

Another, more recent discussion of different string matching techniques for MIR can be found in [Lem00]. Lemström shows that still a major number of issues related to this area is solved by string matching algorithms based on dynamic programming (DP) for retrieving an optimal alignment.

In the remainder of this chapter we first give an overview about existing approaches addressing different issues in the context of similarity analysis, pattern induction, and segmentation; then we propose a new adaptation of the BLAST algorithm to musical data in Section 3.3; in Section 3.4 we then discuss the

[1] Section 3.3 included in this chapter is based on [KH04].

[2] Some basic information about string matching by using dynamic programming can be found in Section A.1. More detailed information on this can be found, for example, in [Gus97].

issue of segmentation in music and present an approach for the segmentation of sequences based on a floating average model (Section 3.4.1).

3.1 Score Following, Pattern Matching, MIR

The general issue of analysing the similarity between two performances or a performance and a score or searching for an optimum alignment between such two musical objects can be divided into at least three categories with different complexity:

- Searching for a single optimum, complete alignment between two musical objects S and T, where both might be sequences of score or performance data. In this case S and T usually have a similar length but one or both include additional, missing, or incorrect notes. A typical example for this category is *score following*. Beside analysis, score following systems are typically used as real-time systems for a computer generated live accompaniment of a soloist. These real-time score following systems create an approximate match between an automatically (in real-time) transcribed score and the corresponding original score. The general task of score following could be describes as "Finding the best alignment between two sequences of symbols from the same alphabet" ([PB02]).

- Searching for approximate occurrences of a given melodic or rhythmic fragment X inside a piece or a larger database S. Different from the search for a complete alignment here only an optimum alignment between the given fragment X and an arbitrary subset of S should be retrieved. Because the number of subsets and their start positions in S are unknown in advance, this task typically requires a higher time complexity then the search for a single, complete alignment between musical objects S and T.

- The required time complexity increase again, if we search for all possible, good approximate alignments between any subset of two musical objects S and T. In terms of string matching these alignments are the common substrings of two musical strings. For the processing of performance data and as well for some types of score data (*e.g.,* variation of a theme) we must allow a certain amount of inaccuracy (*i.e.,* approximate matching of single symbols and insertion of gaps) between the substrings which again increases the complexity of the search. If we use here $T := S$ the self-similarity of a single score or performance S can be analysed and repeated sections and occurrences of significant patterns can be detected. This issue known as *pattern induction* is described in Section 3.2.

Dedicated score following systems have been proposed by many authors in the past twenty years, for example, by Dannenberg [Dan84], Dannenberg and Mont-Reynaud [DMR87], or Pardo and Birmingham [PB01]. According to [Dan01] and [SP93], Dannenberg's approach of a real-time score follower ([Dan84]) supposed to be the first use of dynamic programming (DP) for melodic similarity respectively score following. The score follower module is designed as part of a real-time accompaniment system where it should track the current score position of a live performance. It evaluates only the transposition invariant absolute pitch information of the given score and the observed performance. By assuming that the performer skips only single notes or smalls groups of notes, and also never jumps back in time, the required DP table needs only to be filled and evaluated inside a small window around the position of the current match. The target function (indicating the matching quality between score and performance) which gets maximised by DP, simply counts the number of exact matches (notes with correct pitch).

In [MS90] Mongeau and Sankoff propose an approach for comparing musical sequences based on DP that allows fragmentation (splitting of notes) and consolidation (merging of notes) during the retrieval of a cost optimal alignment. Here the costs for inserting n single gaps step by step become different than the costs for inserting a complete group of n gaps at once. The costs for aligning (equivalent to replacing) notes a_i to note b_j depend on the duration of these notes and on the pitch interval between a_i and b_j. Different from other authors Mongeau and Sankoff propose to use the level of dissonance between the notes and not only the interval size for calculating these costs. The level of dissonance depends on the relation between the observed pitch and a given key signature or scale. For example, the tritone of a scale has a significantly large level of dissonance than the root of a scale. For modelling an edit distance for the

fragmentation and consolidation operations (similar to dynamic time warping techniques) the standard DP recurrence equation (see Equation A.1) here becomes modified to

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + w(a_i, \emptyset), & \text{deletion,} \\ d_{i,j-1} + w(\emptyset, b_j), & \text{insertion,} \\ \{d_{i-1,j-k} + w(a_i, b_{j-k+1}, \ldots, b_j),\ 2 \leq k \leq \min\{j, F\}\}, & \text{fragmentation,} \\ \{d_{i-k,j-1} + w(a_{i-k+1}, \ldots, a_i, b_j),\ 2 \leq k \leq \min\{i, C\}\}, & \text{consolidation,} \\ d_{i-1,j-1} + w(a_i, b_i), & \text{replacement,} \end{cases} \tag{3.1}$$

where $w(a_i, b_{j-k+1}, \ldots, b_j)$ and $w(a_{i-k+1}, \ldots, a_i, b_j)$ are the predefined costs for fragmentation respectively consolidation, and F, C two constants which can be chosen in advance depending on the length of a respectively b. This modification can be done as shown above because Mongeau and Sankoff assume that the input data is given as quantised score data and all durations can be expressed as integer multiples of a small duration (*e.g.,* sixteenth). The consolidation and fragmentation model can be viewed as a special type of deletion and insertion. Instead of inserting gaps respectively skipping notes multiple notes in one sequence become aligned to a single note in the other sequence. Similar to string matching by standard DP (using Equation A.1) their algorithm is considered still to be in $O(nm)$, where n and m denote the lengths of the two sequences a and b.
Where the first part of [MS90] deals with the search for a single optimal alignment between two sequences, in the second part also the search for local similarities is addressed. As already used in the context of molecular biology (see Section 3.3.1) the authors introduce a cost function which calculates positive and negative values for the similarity between two symbols combined with expressing the DP recurrence (Equation A.1) as a maximisation task (*i.e.,* maximise similarity instead of minimising distance or costs). The best local alignment can be found be a trace back starting at the maximising $d_{i,j}$. The second best local alignment can be retrieved by starting a trace back at the maximising $d_{k,l}$ that is not included in the trace back path of $d_{i,j}$. By repeating this steps all good local alignments can be retrieved.

Based on the approach of Mongeau and Sankoff, Smith *et al.* propose in [SMW98] an approach for retrieving melodic similar themes from quantised input data. The used cost function depends on the duration of notes which is given as multiples of, for example, sixteenth notes. This approach retrieves only a single cost optimised alignment between the search pattern and the performance data. To allow the retrieval of matches embedded within a song (pattern length $\ll$ performance length) the leading gaps during DP get a zero penalty ($d_{0,j} := 0, \forall\ j \geq 1$), but in every case the complete DP table must be filled and evaluated.
The authors present a large scale evaluation of their approach on a database consisting of two large folk song corpora (Digital Tradition database and Essen database). The experiment simulates users which known the begin of a melody and ask the system if this melody can be found in the database. Their results show that DP for searching of embedded patterns in large databases requires some optimisation (*e.g.,* the authors propose the solutions based on `agrep`, which is fast but not flexible as DP, because of predetermined number of insertions, deletions, replacements) to become fast enough for usage on really large databases.

The GUIDO/MIR approach proposed in [HRG01] by Hoos *et al.* is an example for a model for MIR that is not based on string matching. Here, an index structure based on Hidden Markov Model transition matrices is generated (off-line) for the database and used for approximate retrieval. Depending on the estimated query type, the index structure might contain, for example, pitch transition matrices, or duration transition matrices. The index structure allows a fast search and fast comparison between the melodies in the database and the query. Because the used matrices and feature vectors contain ambiguous and reduced information of the original sequences, there will false positive matches occur. The model focusses only on retrieving melodies from the database that are similar to the query with a high chance. The search for any type of alignment between the query and the retrieved melody is not in scope of this approach.

In [Cop03] Cope proposes an approach for retrieval of pieces from a database which are similar to a given target work. The here used similarity measure evaluates the rhythmic similarity (the durations are normalised to proportions of the shortest duration in the piece) and the melodic similarity (pitch intervals). The model is based on an approximate, incremental pattern matching technique; complete pieces are stored as a collection of all possible sub-patterns in the database, which allows fast searching

but requires an huge amount of disk space.

In [Dov99] and [Dov01] Dovey proposes an approach for locating polyphonic queries within a polyphonic database. The search space is limited here by allowing in the basic version only the insertion of a maximum of k gaps between any two events of the query. In a more elaborate version the user can specify different limits for the number of allowed gaps between any two successive notes of the query. According to [MWL01] the time complexity of Dovey's approach (as proposed in [Dov99]) is in $O(n'm(t+1)^{(m-1)})$, where m is the length of the query, n' the length of the data set (given as sequence of chords), and t the allowed spacing (insertions, deletions). Dovey points out that the refined version proposed in [Dov99] performs in linear time because of further optimisations.

An approach for melody matching directly on audio data is proposed in [MD01], which uses dynamic time warping (DTW) as extended version of DP.[3] In [HDL02] Hu *et al.* propose an approach for string-matching-based similarity analysis that can be applied to non-symbolic audio data. Instead of using symbolic notes as characters to be aligned, the monophonic audio file is split into time slices of a fixed length. These time slices are used as the 'characters' to be aligned. Because the model uses monophonic audio files it is possible to calculate the pitch frequency for each time slice. The similarity between two slices is the derived from the resulting pitch distance.

In [PB01] Pardo and Birmingham propose an approach focussing on automated accompaniment of an improvising musician. Different from typical score following approaches which try to match an observed performance along a fully specified score (explicit notes), this model addresses the issue of score following using a lead sheet type (partly specified melody, chord name, and overall structure) scores. In a first step the performance is segmented into a sequence of chords, *i.e.,* each segment represents a single chord. Then the name[4] of the best matching chord is attached to each segment, where the model can distinguish six basic chord types: major, minor, dominant seventh, diminished, half-diminished, and fully diminished for each of the 12 possible keys. The sequence of chords given by the lead sheet score is then matched against this inferred chord sequence by using standard DP algorithms. For retrieving an optimum alignment the model uses a chord to chord cost function during the creation of the DP table.
A related approach for melody matching is proposed in [PB02]. Because this model focusses on matching audio performances to a given score it gives respect to possible errors (*e.g.,* wrong pitch, additional or missing notes) caused by the pitch tracker. If the type of the recorded instrument is known a probability distribution for transcription errors can be derived. The probability that a certain pitch occurs in a score depends, for example, on the type of instrument (*e.g.,* a high pitch note has a low probability for bass singer or a bass trombone). For the DP cost function the authors propose to use a skip penalty depending on the duration of the skipped note:

$$skipPenalty(a_i) = k\frac{duration(a_i)}{\sum_{j=1}^{|A|} duration(a_j)} \tag{3.2}$$

Here the skip penalty depends on the length of the note in relation to the length of complete performance.

In [LL98] Lemström and Laine describe an approach for MIR based on a suffix tree representation of the melody database. This representation allows a fast the retrieval of queries in linear time, but it is restricted to the search for ungapped k-approximate alignments. This restrict retrieved results in a way that only a maximum of k symbols of the (musical) query string can be different between query and source, all other symbols must be perfect matches. Another disadvantage here is the large amount of required memory space for the suffix trie data structure: suffix tries require enormous amount of main memory. Therefore indexing should be limited to significant (repeated) parts of performance ([MLW03]).

In [LP00] Lemström and Perttu present the SEMEX (Search Engine for Melodic Excerpts) system for locating transposition invariant matches of monophonic query melodies within monophonic or polyphonic music stored in a database. The used similarity measure evaluates the interval structure, respectively melodic contour. For reducing the time complexity of their algorithm the authors propose an optimised DP approach based on bit parallelism.

For monophonic data and under some restrictions:

[3] See also Section 3.2 for some more details on DTW.
[4] The name of a chord determines the root note and the chord type.

1. the length of the query pattern is short enough that in can be represented by a single machine word with length W,
2. the cost difference between two adjacent vertical cells of the DP table (the column height represents the query length) is always $-1, 0$, or 1,
3. the model retrieves only k-approximate matches, where k specifies a fixed, maximum number of edit operations (insertion, deletion, replacement) for an optimum alignment between two musical strings,

their model is able to run with time complexity $O(n)$, where n is the length of the used database. If the first constraint cannot be satisfied the system runs only in $O(\lceil \frac{m}{W} \rceil n)$, where m denotes the length of the query pattern. Thus, for larger patterns in this case the bit parallelism would result only in a constant speed-up factor, and then therefore runs with time complexity $O(nm)$. Constraint 2 restricts the type of cost function for inserting gaps in the search query for an optimum alignment. It is not possible to use arbitrary gap cost functions (*e.g.,* $c(\text{gap}) \propto$ note duration, or the cost function proposed in Equation 3.2) in this model.

3.2 Pattern Induction and Self-Similarity

Where the previously described approaches search for single or multiple approximate occurrences of a given pattern inside a performance or database, pattern induction (or pattern discovery) approaches are searching for occurrences of (longest) common sub-sequences of two performances. If equal performance or score data is used as source and query then (approximately) repeated regions and typical patterns of a composition can be detected by this *self-similarity analysis.* Similar to the issues discussed in the previous section here also gapped and approximate alignments might be retrieved, but in difference here also arbitrary best matching subsets of the query pattern should be detected.
After retrieving all (longest) common subsets, pattern induction also requires the calculation of a significance for each detected pattern. This can be done, for example, by analysing the rhythmic complexity of the pattern itself, but typically also a pair-wise comparison of the set of retrieved pattern is performed. Pattern induction approaches can be used for deriving the overall structure (musical form) of a composition. The automatic detection of significant patterns is of interest for other pattern-based approaches for musical purpose (*e.g.,* tempo detection or quantisation as described in Section 4.3.2 and Section 5.3). There also exist pattern-based composition systems (*e.g.,* EMI by Cope [Cop96]) which make use of automatic pattern extracting from existing compositions (see [RG02]). (A general discussion of pattern induction and related issues can also be found in [RG02].)

> "...have stressed that the identification of perceptually significant instances of repetition is an essential step in the process by which an expert listener achieves a rich interpretation of a musical work." [MLW03]

The required algorithmic complexity of pattern induction (repetition detection) approaches depends highly on the type of input data and if the similarity analysis allows gapped alignments. For a string with length n the number of possible substrings is in $O(n^2)$ and therefore the number of possible (ungapped) pairs of substrings is in $O(n^4)$.

> "...given a finite set W of words, find a pattern, p that is the longest substructure (*i.e.,* factor or sub-sequence) of every word in W. [...] that if $|W|$ is not constant and the substructures to be considered are sub-sequences, the problem [pattern induction] becomes $\mathcal{NP}$-complete. However, if factors instead of sub-sequences are considered, the problem is solvable in polynomial time. This illustrates that finding repetitions with 'gaps' (*i.e.,* repeated sub-sequences) is much more complex than finding repetitions without gaps (*i.e.,* repeated factors)." [MLW03]

Similar to the pattern similarity and alignment approaches discussed in the previous section, it depends on the actual focus of the implementation what type of similarity measure should be used (*e.g.,* rhythmic,

melodic) and if they should be invariant against tempo changes or transposition. In the following we describe some existing pattern induction approaches from literature.

An early work about representing the structure of performances as sequences of patterns and the relation to human perception of music has been proposed in [SS68] and cited by many authors (*e.g.,* [DH02]). In [SP93] Stammen and Pennycook present an approach for matching melodic fragments against a collection of fragments in real-time based on a dynamic time warping (DTW) model. As pre-processing their model segments a melodic phrase into short patterns of 4 to 16 notes. To allow tempo and transposition invariant matches the patterns are represented as pitch interval and duration ratio series. The DTW calculates a distances measure between all observed patterns and each pattern of the fragment database. This distance measure represents the edit distance for the optimal (gapped) alignment between two pattern. DTW represents a special type of DP. At the initial step a similarity matrix is calculated where each entry $v(i,j)$ represent the costs for aligning symbol a_i of the sequence A against symbol b_j of sequence B:

$$v(i,j) = w(a_i, b_j),\ 1 \leq i \leq |A|, 1 \leq j \leq |B| \tag{3.3}$$

Stammen and Pennycook here only evaluate the pitch distance respectively the melodic contour of the fragments. The second phase of DTW now searches directly for an cost optimal path from $v(1,1)$ to $v(|A|,|B|)$ in the similarity matrix, where the size and direction of *jumps* is limited by constraints. The costs for the optimal path can be calculated by a recurrence function c:

$$c(i,j) = v(i,j) + \min\{c(i-1,j), c(i-1,j-1), c(i-1,j-2)\} \tag{3.4}$$

This function is the DTW version of the recurrence equation used for DP (see Equation A.1). The complexity of the DTW algorithm is in $O(nm)$, where n and m are the lengths of the two sequences to be compared. The complexity for comparing a pattern with length m against a database containing k patterns (fragments) of a maximum length n is then $O(kmn)$.

In [PK02] Paulus presents an approach for measuring the similarity of patterns given as audio files that is also based on DTW. Similar to [SP93] his approach requires a segmentation of the input data into patterns. Here the audio files become pre-processed with a multifrequency filter model which is used to detect periodic signals.

In [RL94] Rowe describes the outline of an algorithm for pattern induction which is based on [SP93]. As pre-processing step here the piece gets first segmented into phrases. A first implementation has used the rules proposed in [SP93], but for future versions Rowe proposes the use of the rules of his Cypher system, which try to detect discontinuities in the features of a sequence. For allowing transposition invariant matches, melodic patterns are represented by their intervallic contour instead of absolute pitch information. Then the similarity between stored pattern and the elements of the set of phrases is calculated. If multiple occurrences of an unknown pattern are detected it will be added to the pattern database.

An approach for discovering patterns using a neural network approach instead of string matching can be found in [Rob96]. Roberts describes several implementations (generations) of his approach. The event driven SONNET1 was built for detecting temporal patterns in a real-time stream of note onsets. "The network's task is to learn and classify common patterns of inter-onset intervals". Roberts also describes a time driven version of SONNET1 which allows to handle expressive timing.

The SIA and SIATEC approaches proposed in [MWL01] and [MLW03] by Meredith *et al.* search for maximal repeated patterns (SIA) and translated equivalence classes (SIATEC) inside musical scores. A third approach (a generalisation of SIA) proposed in [MWL01] named SIA(M)ESE can be used to search for all occurrences of a query pattern inside a data set. The notes of a performance here are represented by a five dimensional vector: m = (onset time, chromatic pitch, morphetic pitch, duration, voice). Because the onset time and duration are specified as integer multiples of sixteenth notes (semi-quavers), all elements of the vector become integer numbers and the vectors denote therefore points in a discrete five-dimensional space. On orthogonal projections of the data set into for example, a two dimensional space geometric analysis algorithms are applied to discover the repeated patterns in time $O(kn^2 \log_2 n)$ for SIA, $O(kn^3)$ for SIATEC, and $O(kmn \log_2 n)$ for SIA(M)ESE, for a k-dimensional data set. The constant k depends on the size of the feature vector for a note that is needed for the intended similarity function (*e.g.,* rhythm only, pitch and rhythm). It can be assumed that if the timing information cannot be expressed in discrete units (integer multiples of sixteenth notes) the time complexity of the algorithm

will significantly increase and/or the used geometric approaches will fail, because the data points will not longer be a on discrete grid in a k-dimensional space.

The problem of calculating the significance of an inferred pattern is addressed by Cambouropoulos in [Cam00a]. Here an *a posteriori* pattern significance f is calculated as

$$f(L, F, DOL) = F^a \cdot L^b / 10^{c \cdot DOL}, \tag{3.5}$$

with L = pattern length, F = frequency of occurrence, DOL = degree of overlapping, a, b, c = constants.

Based on [CCI+99] Iliopoulos *et al.* propose in [ILMP00] an approach for retrieving all δ-approximate repetitions, the longest δ-approximate repetition, and (δ, γ)-approximate repeats inside a performance. Here δ limits the maximum *distance* between pairs of aligned symbols and γ the overall distance between the two repeated sub-sequences.[5] Their approach is focussed on computing direct repetitions (*e.g.,* AAA not AxxxAxxxA) and does not allow the insertion of gaps in the alignment. The model uses a similarity matrix where the similarity between each possible pair of events (notes) of the performance is calculated. The (δ, γ)-approximate repetitions can then be calculated by a successive evaluation of the matrix in $O(n^2)$.

In Rolland's approach proposed in [Rol99], the discovered (ungapped) repeated patterns are restricted by a minimum and maximum length constraint which allows an overall time complexity of $O(n^2)$. If patterns of arbitrary length should be discovered the time complexity rises to (at least) $O(n^4)$ ([MLW03]).

Meek describes in [MB01] an approach for the detection of the main/major themes (*i.e.,* musical keywords) of a performance. The proposed model calculates all sub-sequences of length $2, 3, \ldots, n$, then – after sorting – sub-pattern are extracted again.

In [DH02] Dannenberg and Hu present and compare approaches for pattern discovery in monophonic and polyphonic performances. The focus of the discussed approaches is on inferring the overall structure of a performance (*e.g.,* AABA) by discovering the repeated passages. The discussed algorithms should create a matrix M where each entry $M(i, j)$ "gives the length of a segment starting at note i and matching a segment at note j."
The first described algorithm – for monophonic, symbolic input data – creates this matrix by evaluating only perfect matches (exact equal pitch) between note i and j by allowing also some ways of skipping notes (similar to standard DP) in one or both segments. The authors also discuss pattern discovering algorithms for monophonic audio files input. As pre-processing the files must be divided into small frames and for each frame a so-called *chroma* must be calculated. This chroma is a twelve element vector where each element indicates the energy of a corresponding pitch class (octave invariant). The authors state that a DP approach and also related models from DNA research (*e.g.,* FASTA and BLAST) cannot be used here, because these algorithms would return only a single best matching pattern at the main diagonal of a similarity matrix. But they assume that "It seems likely that better [than their heuristic approach] and faster music similarity algorithms could be derived from these and other biological sequence matching algorithms."
To avoid the complexity of $O(n^4)$ for calculating all possible alignments Dannenberg and Hu propose an approach based on heuristic search. Their algorithm tries to identify the beginning of alignment paths which then are evaluated along the diagonal of the similarity matrix until the path rating falls below a threshold. The path rating depends on the similarity of the single matched symbols but gets normalised by the division of the current length. For polyphonic data the authors propose to separate the input data into chord sequences. If it is possible to calculate for each chord a set of contained pitch classes, the similarity σ between two chords A, B can be calculated by:

$$\sigma(A, B) = |A \cap B| - |A \cup B - A \cap B|, \tag{3.6}$$

where $|X|$ denotes the number of elements (pitch classes) of the set (chord) X. It is easy to see that $\sigma(A, B)$ will return positive values for large, perfect matching chords and negative values for non-matching chords. Similar to DP the matrix M is calculated by

$$M(i, j) = \max\{M(i, j-1) - p, M(i-1, j) - p, M(i-1, j-1)\} + \sigma(i, j) - c, \tag{3.7}$$

[5] See also [CCI+99] for a detailed description of (δ, γ)-approximate string matching algorithms.

where p represents the costs for inserting gaps and c a global constant. Alignment paths then start at position i, j if a $M(i, j)$ becomes positive and ends when the local costs have returned to zero. Because only non-overlapping pattern should be discovered, each cell is evaluated only once and the required runtime is in $O(n^2)$.

After retrieving a set of repeated patterns with one of the described algorithms the set of pattern must be clustered and analysed, for each cluster a prototypical pattern should be selected. Then it should be possible to describe the complete performance by a sequence of these prototype patterns and variations, respectively.
Especially the proposed approach for polyphonic input is very similar to the – in the domain of bioinformatics well known – gapped BLAST algorithm which will be presented in the next section. It remains unclear why the authors cited the original "ungapped" version of BLAST ([AGM+90]) but did not mention its improved, gapped version as proposed in [AMS+97].

3.3 MusicBLAST

The automatic induction of repeated, significant patterns of a performance is from interest in the context of algorithmic composition, musical analysis, or MIR and there exist various approaches addressing the different issues of pattern induction and pattern similarity. Also in the context of pattern-based approaches for computer aided transcription these models are of interest. Beside automatic extraction of significant patterns also the song structure, inferred by analysing the detected repeated segments, is of interest. The knowledge about the song structure can be used to apply, for example, approaches for inferring the key signature or the time signature, separately to different segments. This would allow the indirect inferring of key or time signature changes.

> "Music is composed, to an important degree, of patterns that are repeated and transformed. Patterns occur in all of music's constituent elements, including melody, rhythm, harmony, and texture." ([Row01] p. 168 as cited in [PK02])

As shown in the previous sections of this chapter there exist already many approaches which address this optimisation problem. In the following we propose an algorithm, MusicBLAST, for approximate pattern search/matching on symbolic (including discrete duration information) and semi-symbolic (including continuous duration information) musical data.[6] MusicBLAST is based on the BLAST algorithm, one of the most commonly used algorithms for similarity search on biological sequence data [AGM+90], [AMS+97]. MusicBLAST can be used in combination with an arbitrary musical similarity measure (*e.g.,* melodic, rhythmic) and retrieves multiple occurrences of a given search pattern and its variations. Different from many other pattern matching techniques, it can find incomplete and imperfect occurrences of a given pattern and produces a significance measure for the accuracy and quality of its results. Like BLAST – and also different from other musical pattern matching approaches – MusicBLAST retrieves heuristically optimised bi-directional alignments searching in forward and backward direction by starting at a dedicated *seed note* position of a performance.

The use of the original, ungapped BLAST algorithm [AGM+90] for musical similarity analysis has already been suggested in [CIR98], also other approaches in the previous sections (*e.g.,* [DH02, PB02]) briefly discuss the use of gene matching algorithms in the musical domain. [PB02]: "Dynamic-programming-based implementations that search for a good alignment of two strings have been used for over 30 years to align gene sequences based on a common ancestor (Needleman and Wunsch 1970),"
Where [AGM+90] has been cited in some of these articles we could found no citation of the more flexible gapped version of BLAST as proposed in [AMS+97]. To our knowledge the gapped version of BLAST has not been implemented and evaluated on musical data before.

In the following we first give a short outline about the original BLAST algorithm and then explain its adaptation to retrieval on symbolic musical data. This is followed by a summary of preliminary results on the performance of the new MusicBLAST algorithm, and an outlook to future work.

[6] Parts of this section are based on [KH04].

3.3.1 BLAST

The gapped BLAST algorithm (Basic Local Search Alignment Tool) [AMS+97], based on an earlier *ungapped* version [AGM+90], is a commonly known and widely used search tool in biological sequence analysis.[7] Applied to DNA or amino acid sequences it works as follows:

First a *window-based similarity matrix* V of size $(m \times n)$ between two arbitrary DNA or amino acid strings is created. Each entry $v(i, j)$ of the matrix represents the similarity between sub-sequences of the two strings (starting at position i in string one and position j in string two), with a certain, fixed length w = window size. The similarity between two *single characters* of the sequences is calculated by using a *scoring matrix* which specifies a similarity measure between pairs of DNA symbols, for example, amino acids. For the alignment of amino acid sequences, for example, the PAM matrix is used (see Section A.15).
Next, a *limited number of high-scoring hits* (*i.e.,* best matching windows) within the window-based similarity matrix are selected as start positions for possible alignments. A high-scoring entry (high-scoring window) will only be used as such, if in the same diagonal within a certain distance a second single high-score entry occurs. In the case that there exist multiple high-scoring hits within a small distance of the same diagonal, only one of them will be used for the next step.
Starting from the centre (or any dedicated best position) of each high-scoring window – the *seed* position – a *bi-directional gapped alignment* is then retrieved using a performance optimised version of string matching by dynamic programming (DP), which produces a cost optimised local alignment.

As shown in Figure 3.3.1(a) the two DP tables for the right and left directed alignments are filled alternatingly via the inverse diagonals, starting at the seed position. The entry for each cell $c(i, j)$ is calculated according the standard DP recurrence equation (see Equation A.1), using, for example, the PAM scoring matrix as distance measure between two entries. During the iterative filling of the two DP

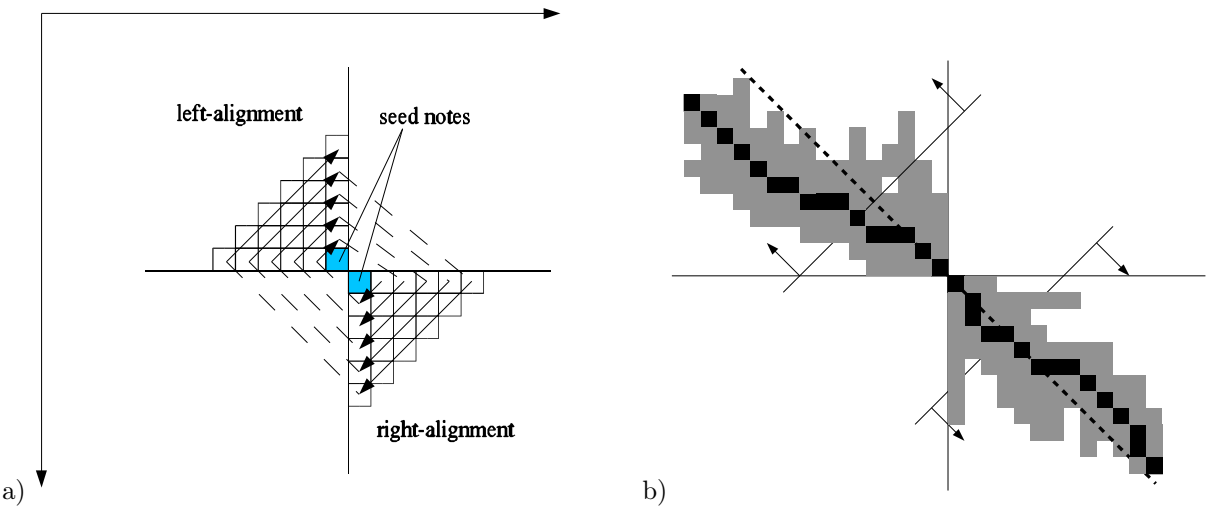


Figure 3.1: BLAST: a) filling of the DP table along the inverse diagonals of the right and left alignment. b) example of resulting optimised number of calculated and evaluated cells in DP table. The black trace marks the optimum path.

tables (one for the left and one for the right directed alignment), a cell $c(i, j)$ is marked as invalid if the distance between its value (costs of a potential alignment path crossing this cell) and the currently best value (of both tables) is higher than a certain threshold. Because the used scoring matrix (*e.g.,* PAM) returns positive values for good matches and negative values for the alignment of non-similar symbols or

[7] If not explicitly specified we will use BLAST as synonym for the gapped version of BLAST as proposed in [AMS+97].

gaps, the alignment of symbols with a high similarity can compensate the insertion of gaps. If a cell $c(i.j)$ in the current border column or border row is marked as invalid, the remaining cells of this column or row can be omitted in the following iterations of the DP table filling, because any alignment path using one of this cells would need to cross the invalid cell $c(i, j)$. The filling of the left or right DP table is stopped, of cell $c(1, 1)$ or $c(m, n)$ is reached, or if no more cell can be calculated because for a current inverse diagonal the corresponding cells have been all marked as invalid (see Figure 3.3.1). The filling process stops if the filling of both tables has ended.

Similar to the (δ, γ)-approximate matching strategy as described in [CCI+99], this abort heuristic decreases the time complexity by reducing the number of cells which need to be evaluated in the DP matrix. Different from the (δ, μ)-approximate strategy it does not limit the total number of insertions (gaps). It also does not restrict the length of the alignments. The retrieved alignments can include an arbitrary number of gaps if the gapped regions are intersected by regions with high similarity. The heuristic reduces the number of cells that need to be evaluated to a band around the optimal path, without restricting the way of the path itself.

Already in the model proposed in [Dan84] (see Section 3.1) the number of cells which should be calculated for filling the DP table had been restricted to a small window around the current cell to be evaluated. But different from the adaptive strategy of BLAST here a more restrictive hard limit of a fixed number of cells has been used.

Although in principle, the DP procedure guarantees an optimal local alignment because the start positions for alignment are selected heuristically and the search space around the best entries is limited, BLAST may not always find an optimal alignment; however, in practice, it has been found to give an excellent combination of accuracy and efficiency. The most time consuming task within BLAST is the generation of gapped alignments. By using the threshold optimisations, the complexity for computing a single alignment can be reduced far below $O(nm)$. Furthermore, the window-based similarity matrix is used to limit the number of calls to the DP procedure to promising start positions.

3.3.2 BLAST on Musical Data

By using the outline of the original gapped BLAST approach and adapting the similarity measures to musical data it should be possible to exploit the performance advantages of the BLAST algorithm for pattern retrieval and similarity analysis in the musical domain. For creating the similarity matrix on musical data (a series of notes and chords), the basic scoring matrix for amino acids (see Section A.15) needs to be replaced by a similarity function that assigns a score to any combination of two notes or chords. Depending on the precise application context, this similarity function can be based on any feature of a single note or chord, in particular, pitch, pitch ratio (interval), duration, IOI, IOI ratio, or intensity. In general it should be possible that any similarity measure (*e.g.,* skip penalty function in [PB02], see Equation 3.2) that has been successfully used within DP-based string matching approaches applied to musical data (see Section 3.1 and Section 3.2) can also be used for MusicBLAST.

By evaluating the IOI ratio and/or the duration ratio instead of absolute IOI and duration, it is possible to allow tempo invariant rhythmical pattern matching between any combination of unquantised performance data and quantised score data. By extending the window similarity function with a similarity measure for a pair of notes and a *gap* value it can also be used for calculating the similarity during the gapped alignment step using DP. It is also possible to use any arbitrary similarity function that satisfies the general requirements of a DP cost function.
The abort criteria of the gapped BLAST algorithm requires, that the used DP cost function returns positive and negative values for the similarity between two entries. It should return positive values (which improve the alignment quality) for high similar notes and negative values for the insertion of gaps and non similar notes. The total alignment costs for a random sequence of notes must be negative, therefore positive cost function values should be used only for high similar notes.

3.3.3 The MusicBLAST Algorithm

Following the outline of the original gapped BLAST algorithm our approach uses a sorted list M of notes as input and works in different stages (see also Figure 3.2):

Building a window-based similarity matrix – The window size w needs to be adjusted according the used similarity/penalty function (see Figure 3.3). In the current implementation we tested a melodic similarity measure (absolute pitch) and also a tempo invariant similarity measure based on IOI ratio. If the window size is too low, the number of wrong high-scoring hits will increase and if it is too high the number of missed high-score entries will decrease. Depending on the data and application a stepsize s might be adjusted between 1 and the window size w, where the window start position in M becomes increased by s when switching from $v(i,j)$ to $v(i+1,j)$. The similarity of a single window $v(i,j)$ of the similarity matrix can be calculated as the mean of all pairs of note similarities or by the multiplication of the associated similarity values. The multiplicative similarity measure

$$v(i,j) = \prod_{k=0}^{w-1} sim(m_{i+k}, p_{j+k}),\ 1 \leq i,j \leq |M| - w + 1, 1 \leq j \leq |P| - w + 1 \tag{3.8}$$

results in a low window similarity if any pair m_i, p_j in the window have a low similarity. The additive window similarity has a lower discrimination rate but might be more adequate for larger search windows:

$$v'(i,j) = \frac{1}{w} \sum_{k=0}^{w-1} sim(m_{i+k}, p_{j+k}),\ 1 \leq i \leq |M| - w + 1, 1 \leq j \leq |P| - w + 1 \tag{3.9}$$

Searching for the high-scoring windows – The threshold for detecting a high-score entry $v(i,j)$ might be set in advance or derived by a statistical analysis (mean, variance) of the similarity matrix entries. In our prototypical implementation a fixed threshold is used.

Selection of the start position (seed) in the high-scoring window – For the selection are different approaches possible: *e.g.,* the centre of the search window, the best matching note, or the longest note of the window. In the context of rhythmic similarity the longest note of the search window should represent an agogic accent and therefore a good anchor point for an alignment. In the current implementation the pair $c(i,j), c(i+1,j+1)$ of locally best matching notes is used as start position for the bi-directional alignment. The left alignment starts with $c(i,j)$ and the right alignment with $c(i+1,j+1)$.

Retrieving a bi-directional gapped alignment for each high-scoring window – The bi-directional alignment becomes retrieved with the same heuristic, iterative dynamic programming approach as used by the original gapped BLAST algorithm described in the previous section. The only difference here is that the cost matrix must be replaced by a cost function that evaluates musical features. This cost function might be any arbitrary musical cost function used by a musical string matching approach such as described at the beginning of this chapter.

3.3.4 Time Complexity

As shown in the previous sections standard DP methods for string matching have already been successfully applied to MIR, score following, and pattern induction approaches; compared to the models for optimising the DP as proposed in these approaches, the here proposed MusicBLAST approach offers more flexibility regarding the cost function and less restrictions to the retrieved alignment (*e.g.,* way of path, number of inserted gaps). For example, a bi-directional heuristic alignment (starting from the seed note in forward and backward direction) promises advantages for all scenarios where two patterns have a high (ungapped) similarity on a small range of notes only. By starting the alignment search in both directions from that high similarity region, the number of calculations during the dynamic programming can be decreased significantly. In these situations standard approaches, such as [SMW98] would require the calculation of all $n \cdot m$ positions of the DP table. The required number of operations for BLAST respectively MusicBLAST consists of two components: operations for creating the similarity matrix and the selection of the high-scoring entries, and k times the number of operations for retrieving a bi-directional alignment starting at an high-scoring position of the DP matrix. Here k should be the number of detected high-scoring entries in the matrix.

For filling the similarity matrix for a window size w each cell requires $O(w)$ operations, independent from the length of the given data sets. Given two data sets with length m and n the size of the similarity matrix is $(m \times n)/s$, where s is the stepsize. If $s < w$ the windows of the similarity matrix will overlap.

```
procedure MusicBLAST(M, P, s, w, l_high)
  input:
    sorted list of notes M
    search pattern as sorted list of notes P
    stepsize s
    window size w
    limit for high-score entries l_high
  output:
    list of rated alignments S
  lenM := |M| − w
  lenP := |P| − w
  i := 1
  while (i · s) − 1 + w − 1 ≤ lenM
    j := 1
    while (j · s) − 1 + w − 1 ≤ lenP
      v(i, j) := windowSim(M, P, w, (i · s) − 1, (j · s) − 1)
      j := j + 1
    end
    i := i + 1
  end
  S := {}
  forAll v(i, j) > l_high
    d := index of seed note in M((i · s) − 1), ..., M((i · s) − 1 + w − 1)
    S′.al := lrAlignment(M, P, d, d − i + j)
    S′.alRate := significance(S′.al)
    S := S ∪ S′
  end
  return(S)
end
```

Figure 3.2: Outline of the MusicBLAST algorithm.

```
procedure windowSim(M, P, w, i, j)
  input:
    sorted list of notes M
    search pattern as sorted list of notes P
    window size w
    start position in M i
    start position in P j
  output:
    one-to-one similarity of notes in window sim_w
  sim_w := 1
  for d := 0 to w − 1
    sim_w := sim_w·noteSim(M(i + d), P(j + d))
  end
  return(sim_w)
end
```

Figure 3.3: Similarity calculation in search window.

The general time complexity for filling the similarity matrix is therefore in $O(nm)$. For a self-similarity analysis of a single performance with length n the complexity is in $O(n^2)$. The required time for retrieving a single alignment between two strings with length n, m with non-optimised DP is in $O(nm)$. Assuming that the number of cells calculated during filling of the DP table is restricted to a band around the best alignment, then for the bi-directional approach the number of evaluated cells would be in $O(m')$, where m' represents the total length of the gapped alignment. Assuming that the length of the retrieved alignment is linear in the query length m, the gapped alignment can be retrieved in $O(m)$.

3.3.5 Significance of Retrieved Alignments

As shown, for example, in [DH02] in the case of self-similarity analysis the set of inferred patterns should be clustered and a prototype for each cluster must be selected. Therefore we must calculate the similarity between each pair of retrieved segments respectively patterns.
Like the similarity measure during the pattern discovery, also the similarity function used for clustering the set of patterns depends on the focus of the actual implementation. Beside the selection of the features used for similarity calculation between two symbols (notes), it must be decided how the overall similarity should be calculated and how the pattern length or its (rhythmic or melodic) complexity should influence the similarity measure.
Several measures for the significance of patterns have been proposed in existing works: Cambouropoulos proposes in [Cam00a] to calculate the significance measure of a pattern by its rhythmic complexity (see Equation 3.5 or [SP00]); Dannenberg and Hu describe in [DH02] a similarity measure depending on the range of overlap in the similarity matrix of the source performance or by calculating the edit distance between pairs of pattern by using an arbitrary type if gapped string matching algorithm; for equal length, ungapped patterns the cosine distance of a feature vector might be used ([PE03]) or the Hamming distance ([KHI+01]); depending on the desired usage of the pattern also a significance measure based on the pattern length in relation to the number of insertions might be calculated. After clustering the set of pattern (or even without clustering) the set of cluster prototypes (or the complete set of pattern) might be sorted or ranked according their significance. In [MLW03] Meredith *et al.* point out several issues in the context of detecting the perceptual significance of repeated patterns and their classification.

For approaches where the patterns should be extracted for creating a pattern database (*e.g.,* for tempo detection or quantisation), we would propose that pattern prototypes of clusters with a low variance can be added automatically to the database. If the clusters show a higher variance between the included pattern these would indicated, that it is ambiguous which of these pattern should become the *correct* prototype for the cluster. Here an interactive implementation where the user should manually select from the top ranked patterns might be the most adequate solution.

3.3.6 Results

The here proposed MusicBLAST algorithm has been prototypically implemented within our `midi2gmn` system. The MusicBLAST module can be used in two ways: for retrieving approximate (complete or partial) occurrences of a search pattern given as a single voice GUIDO file, or for analysing the overall structure of an arbitrary input file by performing a self-similarity analysis.

Figure 3.4 shows the typical difference between the similarity matrices of performed and quantised input data, using an IOI-ratio-based similarity measure. Where the similarity matrix for the quantised file (a) shows large regions with high similarity (bright square regions) the matrix for the performance data (b) shows less similarity. Bright entries along the diagonal indicate repeated passages.

The similarity analysis between the original score and performance of *Alouette* as also discussed in [PB02] (see Figure 3.5) resulted in the similarity matrix as shown in Figure 3.6(a) and (b). For this short file we used for calculating the matrices a stepsize of 1, a window size of 4, and an absolute pitch similarity measure. Our MusicBLAST implementation selected six high-scoring entries of the similarity matrix where each triggered a bi-directional DP for retrieving an optimum alignment starting at the best match of the high-scoring window. As shown in Table 3.1 the use of the BLAST optimisation decreased the number of cells that needed to be evaluated for the retrieval of all six alignments from 12054 down to 2143 (17.8%). As shown in the trace of the alignment paths in Figure 3.6 c), the start positions of the

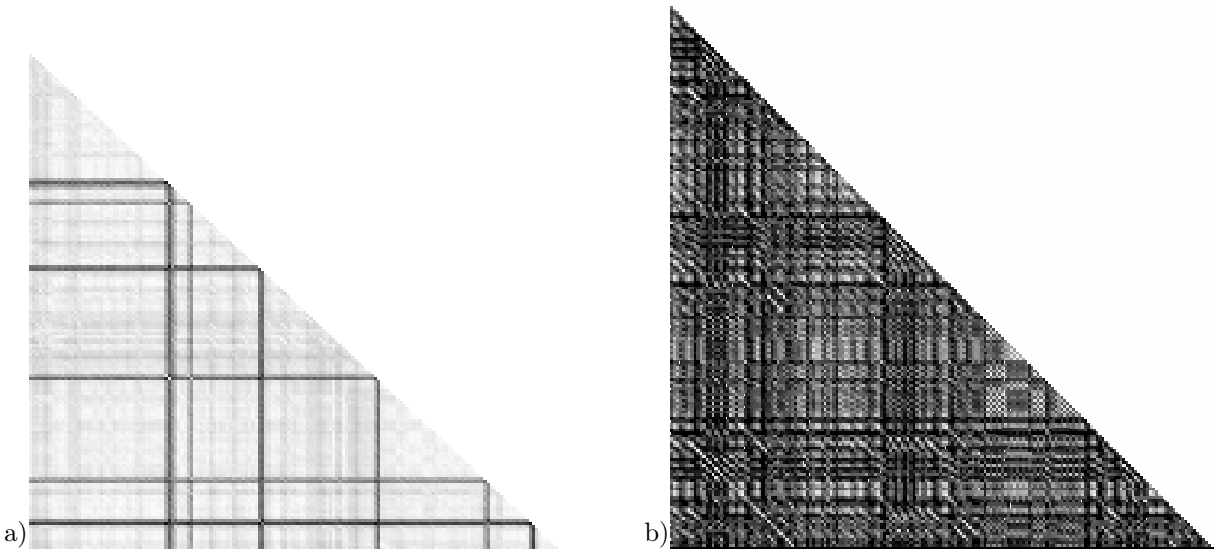


Figure 3.4: Similarity matrices (window size $w = 1$, stepsize $s = 1$) for IOI ratio in a merged, single-voice track of a) Bach, *Inventio 2* and b) Chopin, *Op. 6, Mazurka 1* expressive performance. (similarity = $[0, 1]$ is represented as colours in the range [black, white])

alignments 3 to 5 are subsets of the path of alignment 2. Assuming that it is possible (without increasing the overall time complexity) to mark those start positions invalid that are part of an already retrieved alignment, then the number of evaluated cells would have been decreased to 11.8% of the number of cells required for a non-optimised DP implementation. The very constant relation between the number of evaluated cells and the length of the retrieved pattern gives an indication that the assumption that with the optimised DP table filling of BLAST (mark high penalty cells as invalid) in average case a single bi-directional alignment can be retrieved in $O(m)$, where m is the length of the query.

The evaluation of a performance of the first part of A. Dvořak's, *Humoresque no. 7*, Op. 101 (upper voice including chords, structure = $AABAC$, where the substructure of A is A_1A_1') is shown in Figure 3.7. The MIDI file was created with Maroldt's SONIC system for audio-to-piano-roll (MIDI) transcriptions of piano music [Mar01, Mar04].[8] For our evaluation we used an *ad hoc* implementation of a polyphonic pitch similarity measure evaluating the smallest interval between the chord root any note of a successive chord. A similarity matrix (224×224, upper triangle matrix) with window size 1 and step size 1 resulted in 241 high-scoring positions.
The evaluation with window size 5 and step size 1 resulted in a more adequate number of only 13 high-scoring positions. The corresponding alignments could be retrieved within 0.02s on an Apple iBook. Figure 3.7 shows the similarity matrices for this example and the traces of the retrieved high similar regions. The retrieved alignments correspond to all the occurrences of the main theme of this piece. The gaps in the trace of the first repetition, are caused by different voice separations for the repetition and also by some errors (missed notes) during the audio to MIDI transcription by the SONIC system.

Compared to the standard string matching techniques on musical data the MusicBLAST shows three significant features:

- the number of starts of the time consuming DP is reduced by the similarity matrix pre-processing step and the heuristic for the selection of high-scoring entries,
- retrieving bi-directional alignments starting at high-scoring entries of the similarity matrix,
- optimised, iterative filling of the DP table without limiting the direction of the optimum alignment path.

[8] This and other files can be downloaded at `http://lgm.fri.uni-lj.si/~matic/SONIC.html`.

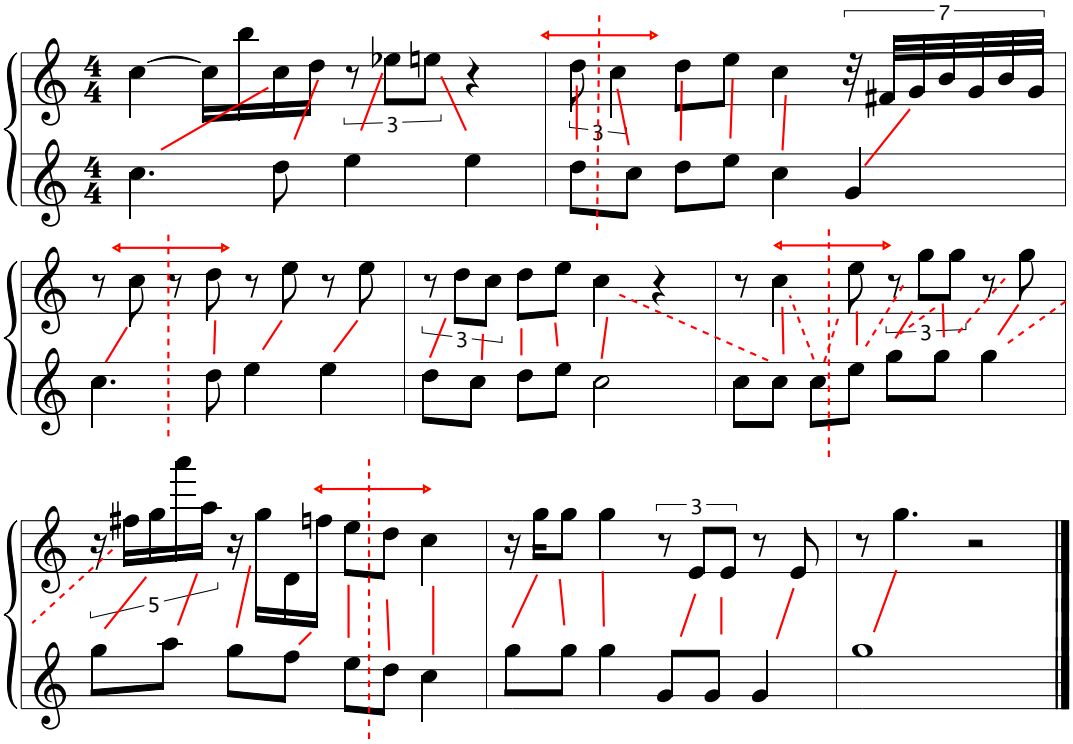


Figure 3.5: Example for original score and performance for *Alouette*. The example has also been discussed in [PB02]. The solid, red lines connecting two notes indicate that they have been aligned by our implementation of the MuscBLAST algorithm. Dashed vertical lines (with arrows on top) indicate the start positions of the alignments included in the path of overlapping alignments around the main diagonal of the similarity matrix. Dashed lines connecting notes indicate that at this positions different alignment paths included different alignments for these notes.

One of the algorithms for pattern matching proposed in [DH02] (named Algorithm 3) seems to be very similar to MusicBLAST respectively gapped BAST but there exist some differences in important details: The in [DH02] proposed algorithm searches in two directions for start and end points of a discovered pattern, but not in the iterative bi-directional way used within the BLAST approach. In our model the abort criteria depends on the distance between local alignment costs and the best seen local alignment costs so far, this is less restrictive than the global threshold proposed by Dannenberg and Hu. The dynamic abort criteria of BLAST avoids the insertions of gaps (which must be trimmed) at the end of the alignment. The window-based MusicBLAST selection strategy for start positions of alignments seems to be more selective (higher discrimination rate) than the single note similarity based strategy of Algorithm 3. As shown in Figure 3.7 and Figure 3.6 the window based similarity matrix includes a significantly higher discrimination rate than the one-to-one similarity measures. With the MusicBLAST approach it is possible – if desired – to retrieve also overlapping pattern, which seems not to be possible with the approach described in [DH02]. Different from Dannenberg's and Hu's implementation where a complete second matrix is used for marking invalid cells, MusicBLAST needs only two arrays for storing the index of the last evaluated cell (first invalid cell, respectively) in each row and each column.

There exist faster algorithms for pattern matching than MusicBLAST (such as suffix-tree-based methods with complexity of $O(m)$ [LHU98]), but it needs to be evaluated how and if the underlying indexing techniques can be applied to approximate, gapped matching approaches for handling queries with missing or additional notes and arbitrary similarity measures. The MusicBLAST approach is robust against these errors, can be adapted to different similarity measures, and can be used for quantised and live performed input data.

nr	dir	startpos	length	# cells evaluated/total	cells evaluated%	#cells/length
	left	15:7	7	62/128	48.4%	8.8
1	right	16:8	12	95/1092	8.7%	7.9
	complete		19	157/2009	7.8%	8.3
	left	5:7	4	36/48	75.0%	7.5
2	right	6:8	42	427/1512	28.2%	10.17
	complete		47	463/2009	23.0%	9.85
	right	22:27	25	233/418	55.7%	9.32
3	left	23:28	21	258/644	40.0%	12.23
	complete		46	491/2009	24.4%	10.67
	left	11:18	13	95/228	41.7%	7.31
4	right	12:19	31	368/930	39.6%	11.9
	complete		44	463/2009	23.0%	10.52
	left	31:39	36	391/1280	30.5%	10.86
5	right	32:40	9	88/100	88.0%	9.78
	complete		45	479/2009	23.8%	10.64
	left	2:19	2	9/60	15.0%	4.5
6	right	3:20	9	81/1170	6.9%	9.0
	complete		11	90/2009	4.5%	8.18
		average	1-6:	2143/12054	17.8%	11.08
		average	1,2,6:	710/6027	11.8%	8.24

Table 3.1: Evaluation of the retrieved alignments for the *Alouette* example as shown in Figure 3.5 and Figure 3.6.

Because of the voice separation (stream segregation) functionality available in `midi2gmn` (see Chapter 2), MusicBLAST can be applied to non-separated polyphonic data as well as to single voices (containing notes and chords) after voice separation has been performed. Even in systems where the data cannot be separated into voices it is possible to use MusicBLAST in combination with a polyphonic similarity measure (*e.g.,* as proposed in [DH02]).

After retrieving a set of cost optimised gapped alignments – built by the concatenation of pairs of left- and right-alignments – these can be ranked by their significance or matching quality. This significance can be determined as the total cost already calculated by the DP procedure or calculated by applying a general cost function, which need not satisfy the constraints for DP. By allowing gaps in the alignment of a search pattern and the performance data, the query will be more robust against extraneous or missing notes in the pattern and/or the performance data. Using MusicBLAST, it is also possible to retrieve substring alignments and multiple occurrences of a pattern in a performance.

With the current direction in developing data formats for representing hybrid combinations of audio and symbolic information (*e.g.,* MPEG7) the number of databases with symbolic musical data and the need for performance optimised retrieval algorithms should increase even more in future. Because DP string matching techniques have already been successfully applied to audio data (*e.g.,* [MD01, HDL02]) it should be possible to use the described approach also for this non-symbolic input data. Given the huge popularity and success of BLAST in biological sequence analysis and retrieval, we believe that MusicBLAST has substantial potential for MIR research and applications.

3.4 Segmentation

For the segmentation of a score or performance into smaller segments exist different general strategies: segmentation by structural analysis, by musicological analysis, or by significant changes in features in the sequence of notes. Where the first type of segmentation can be achieved by the pattern induction and similarity analysis approaches shown in the previous sections of this chapter, the two other types of segmentation require different algorithmic models.

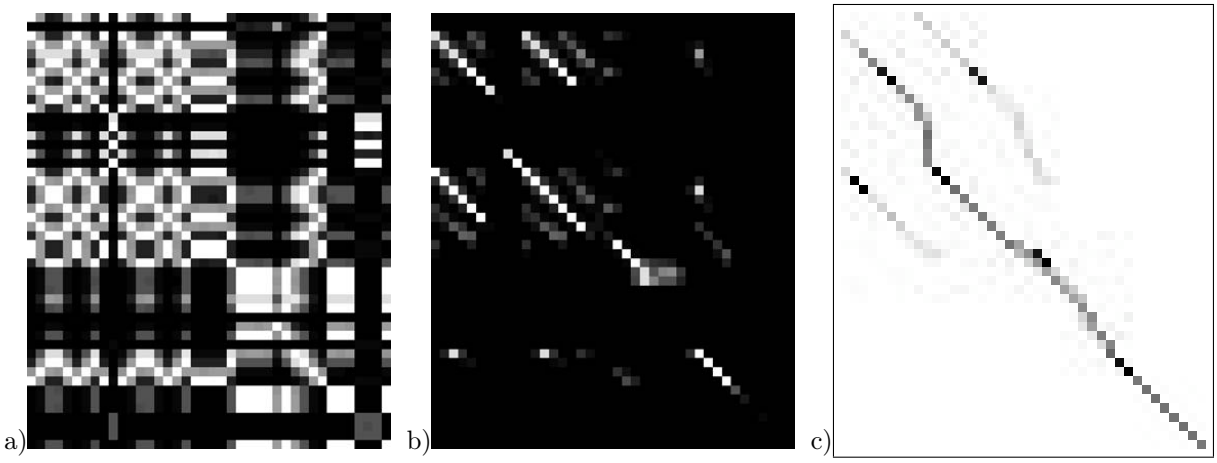


Figure 3.6: *Alouette* example, score *vs* performance, absolute pitch similarity measure: a) similarity matrix (window size = 1, step size = 1); b) similarity matrix (window size = 4, step size = 1) the window similarity has been calculated by multiplication (see Equation 3.8); c) trace of retrieved alignments. In a) and b) the bright regions have a high similarity. In c) the start positions of the traces are indicated by a pair of black squares (on for the left and on for the right direction). Positions included only in a single alignment are coloured in light grey. A dark grey coloured position indicates that it is included in more than one alignment. The slight deviations between the four overlaid traces around the main diagonal are caused by ambiguities of the cost function (*e.g.,* for a series of three perfect matches and a single gap, the cost function is independent of the gap position) and the different directions (left/right) in which they have been passed by the alignment.

The issue of segmentation into musicological meaningful phrases has been addressed by several authors. In [TSH02] Thom *et al.* show that the segmentation of a performance into such phrases is ambiguous. Even human experts typically create different segmentations for a given score. Therefore it becomes somehow difficult to decide if the output of an algorithmic segmentation algorithm is correct or not. In [Har03] Harford shows a segmentation approach based on a self organising neural network model (SONNET). A SONNET-based approach is also proposed by Roberts and Greenhough in [RG94]. An approach based on a neural network technique can be found in [LPP95]. Here Large *et al.* propose a neural net model for creating pattern-based, reduced representation of a given input. The basic assumption here is that human listeners also use this type of reduced representation for remembering perceived melodies. The proposed network tries to decompose an observed performance into fragments of the set of trained melody patterns.

If it is possible to describe a performance as a list or sequence of symbols and calculate for each symbol a specific feature value (*e.g.,* intensity, local tempo) the performance respectively the sequence can also be segmented at those points where the local average of these feature values changes significantly. Such approaches for segmentation based on discontinuities in the features of note sequences have been proposed and implemented, for example, by Rowe in the context of his Cypher system ([Row93]). In [Cam01a] Cambouropoulos presents the *local boundary detection model* (LBDM) for segmenting a performance by analysing its melodic surface and rhythmical structure. The detection of boundaries is based on two rules:

> "Change Rule (CR): Boundary strengths proportional to the degree of change between two consecutive intervals are introduced on either of the two intervals (if both intervals are identical no boundary is suggested).
> Proximity Rule (PR): If two consecutive intervals are different, the boundary introduced on the larger interval is proportionally stronger."

In [MO03] Melucci and Orio compare the output of the LBDM model to manual segmentations.

In the following we present an approach which can detect *breaks* in sequences of float or integer numbers.

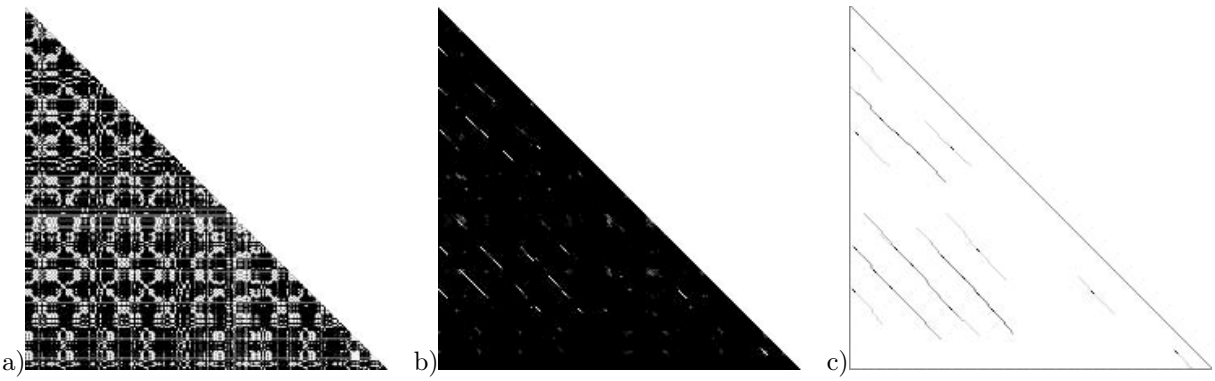


Figure 3.7: *Humoresque* self-similarity analysis. a) similarity matrix (window size $w = 1$, step size $s = 1$); b) similarity matrix (window size $w = 5$, step size $s = 1$) the window similarity has been calculated by multiplication (see Equation 3.8) ; c) trace of retrieved alignments. In a) and b) the bright regions have a high similarity. In c) the start positions of the traces are indicated by a pair of black squares (one for the left and one for the right direction). Positions included only in a single alignment are coloured in light grey. A dark grey coloured position indicates that it is included in more than one alignment.

Similar to LBDM a *break* within a sequence should be detected at positions of significant chances of the local average value. Different from LBDM our approach focus not on the detection of single changes (glitches) but on changes of the overall level.

3.4.1 Floating Average Model

On several sub-problems (*e.g.,* tempo or intensity indications) in this work a sequence of values needs to be divided into different segments (clusters), where a segment should start at positions – in the following also called breakpoints – where the local average of the sequence values changes significantly. For example, a new intensity indication (*e.g., p, mf, ff*) should only be indicated in a score if the average intensity level changes significantly and for a certain number of successive notes. If the intensity would be indicated for each single note the resulting score would become very hard to read and hard to play by human musicians. To solve this segmentation problem a so-called *floating average* algorithm has been developed and implemented in the context of this thesis.

For a sequence $X = (x_1, \ldots, x_n)$, where each x_i denotes an arbitrary value (*e.g.,* the performed intensity of a note), and a range r, and a decay rate $0 \leq d < 1$ we define two average measures for position s of the sequence X:

$$av_{right}(X, s, r, d) = \frac{1}{r'} \sum_{i=0}^{r-1} x_{s+i} \cdot \tilde{d}^i, \; 0 \leq s \leq n - r - 1 \tag{3.10}$$

$$av_{left}(X, s, r, d) = \frac{1}{r'} \sum_{i=0}^{r-1} x_{s-i-1} \cdot \tilde{d}^i, \; r \leq s \leq n, \tag{3.11}$$

with

$$\begin{aligned} r' &= \sum_{i=0}^{r-1} \tilde{d}^i \\ &= \frac{\tilde{d}^r - \tilde{d}^0}{d - 1} \\ &= \frac{\tilde{d}^r - 1}{d - 1} \end{aligned}$$

Thus, r' represents a geometric series. It is easy to see, that for the special case with $d = 0$, av_{right} and av_{right} return the mean of the sub-sequence $x_s, \ldots, x_{s+r-1}$ respectively $x_{s-1}, \ldots, x_{s-r}$. With a decay rate $d > 0$ the influence of elements x_i decreases exponential with the distance between i and s (see Figure 3.8).

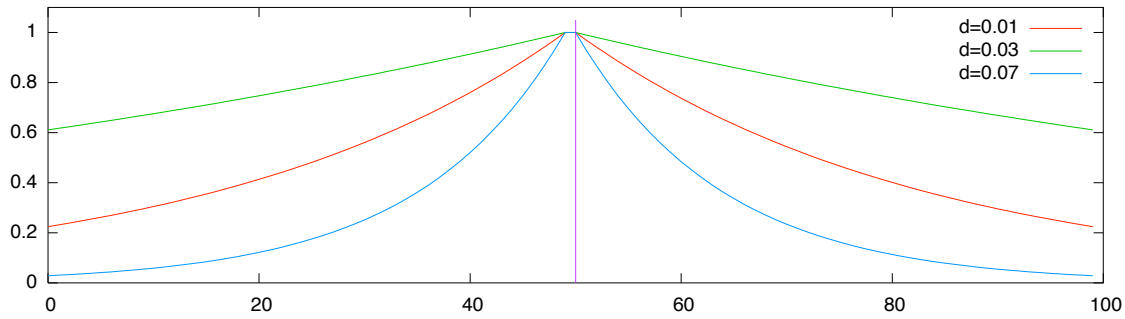


Figure 3.8: Decay of influence for floating average with $s = 50$.

To allow also the calculation of av_{right} for $n - r < s \leq n$ and av_{left} for $0 < s \leq r - 1$ the range of i can be limited to $i = 0, \ldots, \min\{n, s + r - 1\}$ respectively $i = \max\{1, s - r - 1\}, \ldots, n$ where the selected range then also needs to be used for the calculation of the current r'.

Instead the exponential decay caused by the multiplication with d^j actually any other window function could be used (*e.g.,* a Gaussian window function) within the two average measures. In this thesis the exponential decay was chosen because it allows to model also the mean average (with $d = 0$) and the iterative calculation of average values. It is possible to define av_{right} and av_{left} in a recursively way which is of advantage when stepping through the data iteratively:

$$av_{right}(X, s, r, d) = \begin{cases} \frac{1}{r'} \sum_{i=0}^{r-1} x_{s+i} \cdot \tilde{d}^i, & \text{if } s = 0 \\ \left(av_{right}(X, s-1, r, d) - \frac{1}{r'} x_{s-1} \right) \cdot \tilde{d}^{-1} + \frac{1}{r'} x_{s+r-1} \cdot \tilde{d}^{r-1}, & \text{otherwise} \end{cases} \tag{3.12}$$

$$av_{left}(X, s, r, d) = \begin{cases} \frac{1}{r'} \sum_{i=0}^{r-1} x_{s-i-1} \cdot \tilde{d}^i, & \text{if } s = 1 \\ \left(av_{left}(X, s-1, r, d) - \frac{1}{r'} x_{s-r} \cdot \tilde{d}^{r-1} \right) \cdot \tilde{d} + \frac{1}{r'} x_{s-1}, & \text{otherwise} \end{cases} \tag{3.13}$$

It should be noted that for large r and/or large d the iterative calculation of $av_{right}(X, s, r, d) = f(av_{right}(X, s-1, r, d))$ is sensitive to floating point errors. An element x_i is added as $x_i \cdot \tilde{d}^{r-1}$ to the average sum, then $r - 1$ times multiplied by $\tilde{d}^{-1}$, and then subtracted as x_i from the sum.
With a non-exact floating point unit this will result in:

$$\frac{x_s}{\tilde{d}^{r-1}} \cdot \tilde{d}^{r-1} = x_s + \epsilon$$

The errors for all individual x_i will sum up and increase the total error, when iterating several times through av_{right}! It is also easy to see, that for large r a rather small value for d should be used, so that the last elements in range x_{s+r-1} and x_{s-r} still have a significant weight $\tilde{d}^{r-1}$. Therefore, combinations of r, d with $\tilde{d}^{r-1} \gg 0$ ($\Rightarrow d \ll 1$) should be preferred. Otherwise the average function is very insensitive against single exception values in X if $(1 - d)^{r-1}$ is too high and very sensitive if $(1 - d)^{r-1}$ is too low.

By comparing right and left average and/or their ratio for equal or different range parameters r at a position s potential *breaks* in the series of elements of vector X can be detected. Figure 3.9 and Figure 3.10 show two examples for two sequences X_1, X_2 of integers values . X_1 includes clusters of alternating data values and X_2 clusters of ascending values (values marked by dots in figures). The

other parameters of other models have been set to $r = 19, d = 0.1$ and $r = 7, d = 0.2$. The two plots at the bottom of the figures show the value for a comparison between left and right average measures: $\delta^2 = (av_{left} - av_{right})^2$. As shown in the δ^2 diagrams (Figure 3.9 bottom and Figure 3.10 bottom), the square of the distance between left and right average for both combinations shows characteristic maximum peaks at the breakpoints of X_1 and X_2. The detailed rules for the detection of breakpoints may depend on the expected data, but a general rule can be expressed as

$$(av_{left}(X, s, r, d) - av_{right}(X, s, r, d))^2 = \text{local maximum} \implies \text{high chance that } s \text{ is a breakpoint position.} \quad (3.14)$$

In an actual implementation several floating averages with different range r should be calculated simultaneously where then this criteria can be confirmed for the different floating average values for s. For the data shown in Figure 3.9 a non-weighted average calculation (=mean) with a range $r \geq\approx 18$ would be nearly constant for all positions s which would indicate no break in the series of data values. For reasons of visualisation the plots that actually are defined only at positions $x_1, \ldots, x_n$ have been linear interpolated at all intervals (x_i, x_{i+1}).

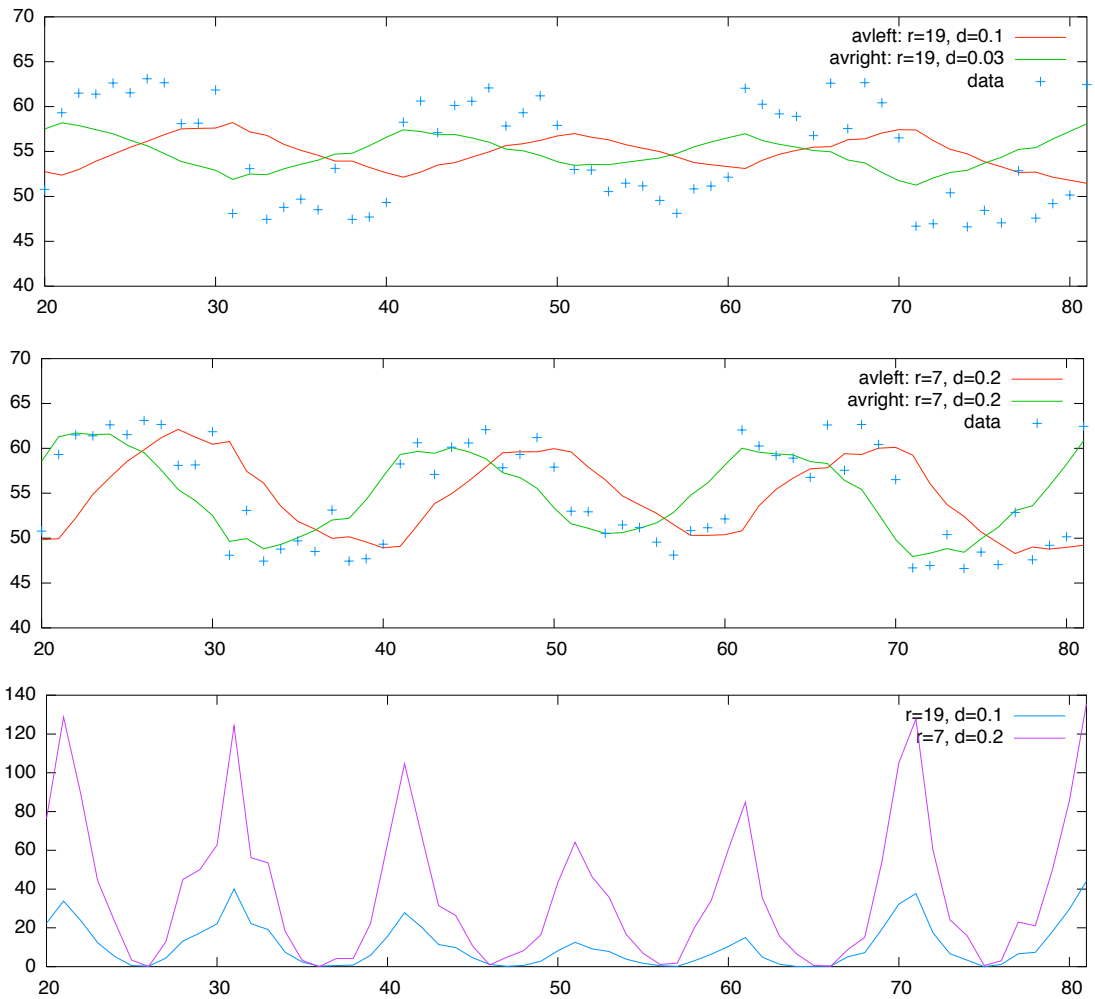


Figure 3.9: Example 1 for floating average calculation with $19 \leq s \leq 81$, and two combinations of $r = 19, d = 0.1$ (top) and $r = 7, d = 0.2$ (centre) and the resulting δ^2 (bottom).

Originally, the floating average concept was developed in the context of our voice separation model. For each onset time of the performance data the currently used number of voices was calculated and this

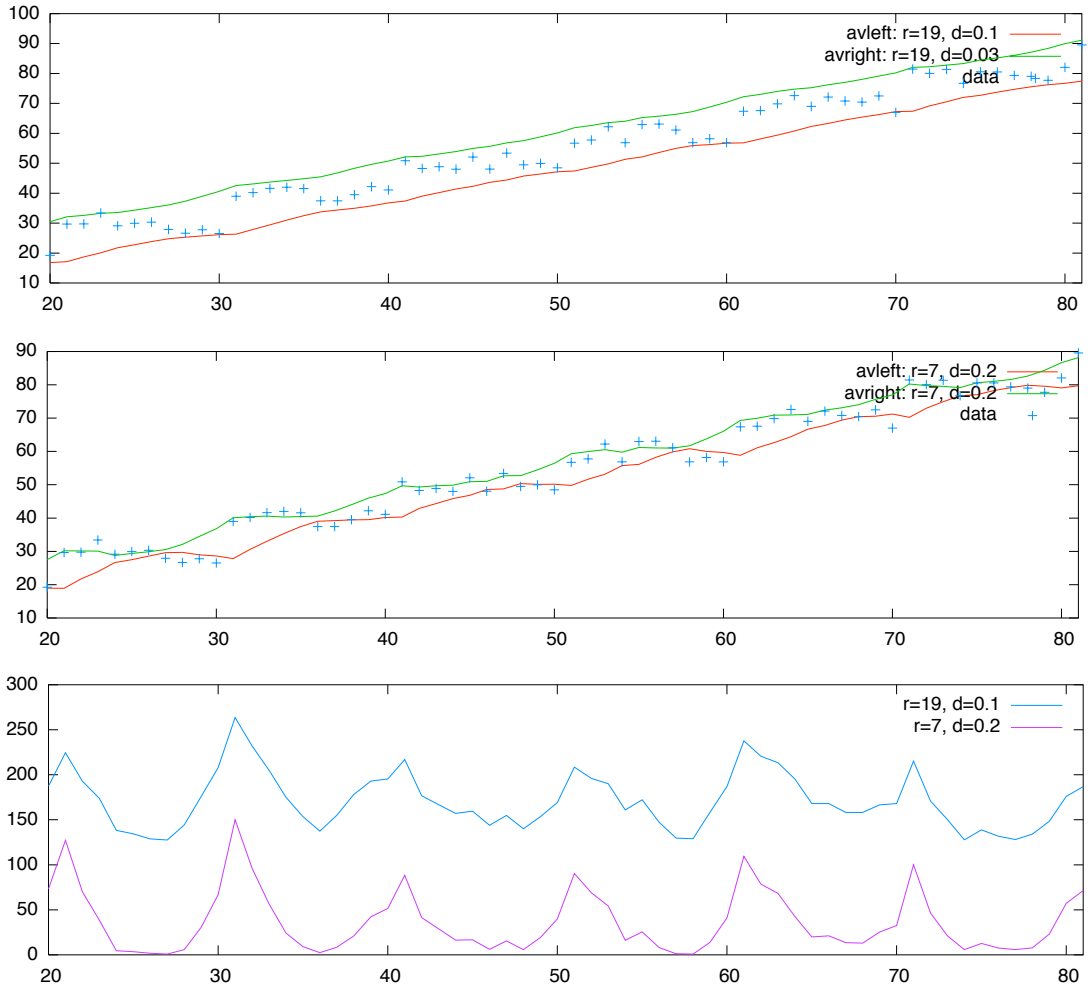


Figure 3.10: Example 2 for floating average calculation with $19 \leq s \leq 81$, and two combinations of $r = 19, d = 0.1$ (top) and $r = 7, d = 0.2$ (centre) and the resulting δ^2 (bottom).

voice profile was then used as input sequence X for the described floating average analysis. With this analysis breakpoints should be detected at which the number of sounding voices significantly changes. For each segment – with then a nearly uniform number of voices – the voice separation should be performed separately. The research showed that connecting the n voices of one segment with m voices of the successive segment correctly can get very complex with a high chance of errors. To overcome these issues the voice separation approach using local search has been developed as shown in Chapter 2. This approach does not need any pre-segmentation to obtain good results.

In the current version of our system the floating average approach is used for inferring the intensity profile of the performance data, where only significant changes in intensity should be denoted in the score (see Section 6.5). The model also can be used for any arbitrary segmentation task, where it is possible to encode a data vector X, such as the detection of key or meter changes which are discussed in Section 6.2 and Section 6.1, or the *smoothing* of the tempo profile inferred by tempo detection approaches as described in the next chapter.

4 Tempo Detection

Following the beat of a performance – by tapping the foot or clapping hands – is a task that even untrained human listeners are able to do.[1] This task of beat tracking respectively tempo detection by inferring the beat of a performance is one of the essential tasks in the area of computer aided transcription. Tempo detection can be interpreted as the worst case of score following: the performance must be matched against an unknown score.

> "*Beat*, as a phenomenon, refers to the perceived pulses which are approximately equally spaced and define the rate at which the notes in a piece of music are played. For a specific performance, the beat is defined by the occurrence times of these pulses (*beat times*), which are measured relative to the beginning of the performance." [Dix01a]

The so-called beats of a performance are equivalent to *tactus strokes* which typically coincide with the strokes give by a musical conductor. If their corresponding score duration is known we can calculate a local tempo for a pair of successive strokes by measuring their absolute distance (in seconds) and comparing it to their distance in the score time domain (see Equation 1.1). As shown, for example, in [DG02] and [Par94b] there is some evidence that the perceived time positions of beats might slightly differ from the observed onset time positions. The perceived tempo profile usually includes less hard jumps between different tempi than then calculated profile. Nevertheless, by allowing the inferred onset times to be slightly shifted (quantised) after the beat induction process we assume that a tempo detection approach can evaluate only the observed (measured) onset time positions.
The process of inferring the corresponding score durations (beat level) of the beats (tactus strokes) is often called tempo induction or tempo detection and requires different approaches than pure beat tracking.

The task of beat tracking can be processed on the audio level where the beat positions can be inferred by evaluating the signal strength and frequency spectrum of audio wave signals (see [GM94, RGM94]) or it can be processed on low-level symbolic data where timing information is given in absolute time units (units of seconds). Some (pure) beat tracking approaches (mostly using audio as input) (*e.g.,* [GM94, Got01, Dix01c]) focus primarily on inferring the time positions of beats of a performance. The output of these approaches is sufficient for triggering other real-time components (*e.g.,* light controllers, interactive performance systems) or it can be used as input to higher level transcription approaches focussing more on transcription than on just beat tracking. In addition to the detection of the beats itself here also a score duration (or inter-onset interval) for the absolute distance between two successive beats needs to be inferred.

> "The principal reason that beat tracking is intrinsically difficult is that it is the problem of inferring an original beat structure that is not expressed explicitly. The degree of beat-tracking difficulty is therefore not determined simply by the number of musical instruments performing a musical piece; it depends on how explicitly the beat structure is expressed in the piece. For example, it is very easy to track beats in a piece that has only a regular pulse sequence with a constant interval." [Got01]

Because of the high chance of ambiguities the estimation of score durations for given performance durations becomes a complex task. For example, if a single note was played with a performance duration of 500ms it could be transcribed as a crotchet, played at a tempo of 120bpm, or as a quaver played at 60bpm, or any other score duration played with a corresponding tempo (see Equation 1.1). In general there exist two different types of approaches: some approaches try to infer the positions of beats assuming

[1] In [DPB00] Drake *et al.* compared the foot tapping of non-musicians against the tapping of musicians.

a constant score duration for all beats (*e.g.,* quarter notes) others try to infer directly the score duration for arbitrary distances between the onset times of significant performance notes. Here the score duration of beats might not be constant. In both cases the estimation of score positions for beats (and the local tempi) of a performance is not necessarily equivalent to directly create a completely quantised output. Depending on the beat level, there will be more or less imprecisely played notes between two inferred beat positions which then still need to be quantised (see Chapter 5 for details on quantisation).

The focus – regarding beat tracking and tempo detection – of this thesis is on tempo detection using low-level symbolic performance data as input. It can be assumed that the given performance data includes notes that coincide with the position of beats, that it includes notes that do not coincide with beat positions, and that it also does include beat positions at which no note onset time exists. For inferring the tempo and score duration information from low-level musical data several existing constraints and typical features of performances can be used: known physical and practical limitations for the overall performing tempo of pieces and score durations of notes, heuristics about existing pieces, known patterns for sequences of note durations. The tempo detection task becomes even more difficult, if it is assumed that the pieces (input data) includes written tempo changes (subito or as ritardando, accelerando) and also inexactly played durations (respectively inter-onset intervals) which are equivalent to small, local tempo changes. In particular it is not trivial to decide if two successive notes with different performance durations should have the same score duration and a tempo change should be added to the score or if the tempo should stay constant and the notes should also have different score durations. As stated by Desain and Honing the tempo detection process might be expressed as an optimisation task: "correct tempo is the one that results in a simpler quantisation" ([CK03]).

For several reasons all beat tracking approaches evaluate the onset time of notes and omit the analysis of their offset points:

- Experiments showed that the onset time of notes usually are played more precise than their offset respectively duration.
- As stated in [Rob96] psychologists commonly agree that the inter-onset intervals (IOI, distance between successive onset times) create the rhythm of a musical sequence. The actual durations of notes have only a minor influence to the rhythmic perception. It is easy to see that the onset times of notes typically create the rhythm and that their durations are responsible for creating a certain feeling, motion or style (*e.g.,* staccato, legato).

Nevertheless, we assume that the duration and intensity of a note contributes to its salience (or weight) which might be of interest for beat tracking and tempo detection.

In the remainder of this chapter we first give an introduction and a formal definition on the conversion between absolute timing and metrical score-timing, using a sequence of beats given as a so-called clicktrack. Then we give an overview about different types of existing approaches for tempo detection. In Section 4.3 we introduce a newly developed interactive tempo detection approach based on pattern matching and statistical analyses. Finally we show results obtained by using this hybrid approach.

4.1 The Clicktrack Model

The mapping between performance time information and score time information can be modelled by a so-called *clicktrack* consisting of a sequence of *clicknotes* which should represent a general form of *beats*. Different from a sequence of beats where all beats usually have an equal score duration, each clicknotes of a single clicktrack can have a specific score durations. A sequence of clicknotes with equal score durations is equivalent to the strokes of a hardware metronome where the distance between two strokes (clicks) is set to a specific absolute time interval (performance timing) and interpreted as a certain score duration. For example, 60 M.M.[2] denotes 60 quarter beat strokes per minute resulting in an absolute duration of one second for a single quarter beat. With each stroke (click) the score time will be increased by the beat duration, if not specified in detail a quarter note is used as the default beat duration. The unit M.M. for

[2] M.M. denotes Mälzels Metronome

indicating the tempo of a performance is equivalent to the also commonly used units of *beats per minute* (bpm).

In the following we consider that a performed note m can be described by a vector

$$m = (onset_{perf}, duration_{perf}, pitch, intens, voice),^{3} \tag{4.1}$$

where

$onset_{perf}$	=	time position of performed onset time (*i.e.,* the onset time) of note m given in ms
$duration_{perf}$	=	performance duration of note m given in ms
$pitch$	=	pitch of note m given in semitone steps
$intens$	=	intensity of note m given as a float in range $(0, 1]$
$voice$	=	index of the voice that note m has been attached by the voice separation module.

In the following the specific data properties for a note m will be retrieved by using an equivalent function for each feature (*e.g.,* $pitch(m)$).[4]

A complete piece can be described as an ordered list M of notes:

$$M = \{\ m_1, \ldots, m_{|M|} \mid onset_{perf}(m_i) \leq onset_{perf}(m_{i+1})\ \} \tag{4.2}$$

We assume that notes with equal onset times have been split into different voices or merged into chords by the voice separation module. A single voice $V_s \subseteq M$ with voice index s can then be described as an ordered set of notes:

$$V_s = \{\ m' \mid m' \in M \wedge voice(m') = s\ \}, \tag{4.3}$$

with

$$\begin{aligned} &\forall m_i \in V_s : onset_{perf}(m_i) \leq onset_{perf}(m_{i+1}) \\ &\qquad \wedge \forall\ m_i \in V_s : onset_{perf}(m_i) = onset_{perf}(m_{i+1}) \Longrightarrow chord(m_i, m_{i+1}) = true \end{aligned} \tag{4.4}$$

Where $chord(m_i, m_j) = true$ indicates that m_i and m_j have been merged to a chord by the voice separation module (see Chapter 2). The voice separation ensures that if $chord(m_i, m_j) = true$ then $voice(m_i) = voice(m_j)$ therefore $\forall s, t : s \neq t \Rightarrow V_s \cap V_t = \emptyset$. It is also trivial to see that $M = \bigcup_s V_s$ (please see Chapter 2 for details about voices and chords). If only time positions and durations of notes are of interest (no pitch information), in the following a note m_i might also denote a chord which actually consists of a group of notes with equal onset times and durations.

From the performance timing information for each pair of successive notes m_i, m_{i+1} of a voice V_s two characteristic values, the inter-onset interval (IOI) and the inter-onset interval ratio (IOI ratio) can be calculated. These measures are defined as

$$IOI_{perf}(m_i) = onset_{perf}(m_{i+1}) - onset_{perf}(m_i) \tag{4.5}$$

$$IOIratio_{perf}(m_i) = \frac{IOI_{perf}(m_i)}{IOI_{perf}(m_{i-1})}. \tag{4.6}$$

The details for the IOI and the IOI ratio and their representation are discussed in Section A.2.

A basic clicknote c can be described as a vector $c = (onset_{perf}, duration_{score})$. Which represents a mapping between a score duration, $duration_{score}$, starting at a performance time position, $onset_{score}$. For retrieving the properties of a clicknote c we define two functions:

$$onset_{perf}(c) = onset_{perf} \text{ of } c \text{ in ms} \tag{4.7}$$

$$duration_{score}(c) = duration_{score} \text{ of } c \tag{4.8}$$

[3] Different from the definition in Equation 2.1 we must here distinguish between score and performance time information.

[4] In the following we will us the MIDI pitch system where 60 indicates the pitch of the c in the first octave. As long as only relative pitch information (intervals) are evaluated also an arbitrary reference point could be used.

We define a clicktrack C as an ordered set of clicknotes:

$$C = \{\ c_1, \ldots, c_{|C|} \mid onset_{perf}(c_i) < onset_{perf}(c_{i+1})\ \} \tag{4.9}$$

For a clicknote $c_i \in C$ now the onset time $onset_{score}(c_i)$ in score time can be calculated by

$$onset_{score}(c_i) = \begin{cases} onset_{score\,def}, & \text{if } i = 1, \\ onset_{score}(c_{i-1}) + duration_{score}(c_{i-1}), & \text{otherwise.} \end{cases} \tag{4.10}$$

Usually the onset time (attackpoint), $onset_{score\,def}$, of the first clicknote will be set to zero. In special cases (upbeat situation or incomplete clicktrack indicated by notes m_i with $onset_{perf}(m_i) < onset_{perf}(c_1)$), the score position of the very first clicknote might be set to a position greater than zero.

The clicknotes define a mapping between score timing and performance timing only for their corresponding time positions. Because we assumed that there might be more performance notes than beats respectively clicknotes also a mapping between arbitrary score and performance time positions must be defined. For a given clicktrack $C = \{c_1, \ldots, c_{|C|}\}$ and an arbitrary performance time position t_{perf}, a normalisation function $norm_{score}$ returning an equivalent score time position for t_{perf} can be defined as

$$norm_{score}(t_{perf}, C) = onset_{score}(c_i) + duration_{score}(c_i) \cdot \frac{t_{perf} - onset_{perf}(c_i)}{onset_{perf}(c_{i+1}) - onset_{perf}(c_i)}, \tag{4.11}$$

where $c_i \in C$ and $onset_{perf}(c_i) \leq t_{perf} \leq onset_{perf}(c_{i+1})$.
For $t_{perf} < onset_{perf}(c_1)$ or $t_{perf} > onset_{perf}(c_{|C|})$ additional clicknotes can be inferred by extrapolating the distances between c_1 and c_2 or $c_{|C|-1}$ and $c_{|C|}$ (see also [Kil96]).

For a note $m \in M$ and a given clicktrack C now the onset time $onset_{score}$ (in score time) can be calculated by

$$onset_{score}(m) = norm_{score}(onset_{perf}(m), C) \tag{4.12}$$

and the score duration of m by

$$duration_{score}(m) = norm_{score}(onset_{perf}(m) + duration_{perf}(m), C) - onset_{score}(m). \tag{4.13}$$

It should be noted that the resulting score timing information here still not necessarily fits into the discrete timing grid of displayable scores, if the performance data does not represent a mechanical (total accurate) performance the score timing information must still be quantised (see Chapter 5).

It is easy to see that for a clicknote or performed note $\tilde{m}$ with $IOI_{perf}(\tilde{m})$ and $duration_{score}(\tilde{m})$ now two parameters of Equation 1.1 are known and a local tempo for $\tilde{m}$ can be calculated. With a given clicktrack C and usage of the defined functions the normalisation of performance timing information into score timing information becomes a straightforward task.

4.1.1 Tempo Detection With Manual Recorded Clicktrack

For this approach[5] the input data consists of two parts: the musical data and the metronome data. The metronome data consists of events (*e.g.,* note onsets, pedal hits) on every beat of the piece and a score duration for this events and represents a clicktrack as defined in the previous section. By comparing the recorded (absolute) performance time position and the score time position of each click-event the performance/score relation – resulting in a local tempo – for each clicknote can be calculated. The set of all local tempo values represent the overall tempo profile of the performance. The recording of the clicktrack can be done parallel to the recording of the musical data (*e.g.,* by triggering a pedal or a particular key) or in a second step during a playback of the previously recorded musical data. In both cases the clicktrack itself might again include inaccuracies which need to be filtered during the tempo-detection. Tempo detection using a manual recorded clicktrack is a rather straightforward calculation without the need of using advanced statistical models. A disadvantage lies in the fact that it is very inadequate for the user to create the clicktrack manually during or after creating the musical recording.

[5] The conversion of performance timing information into discrete score timing using a manual recorded clicktrack is used in commercial sequencers (*e.g.,* Cakewalk) and has also been described and implemented in the context of [Kil96].

More advanced approaches for tempo detection are using different kinds of statistical analysis models for creating a clicktrack automatically from the performance data. These (implicit or explicit) generated clicktrack can then be used as input to the approach described above. A selection of some popular approaches will be described in the next section. An hybrid approach based on pattern processing and statistical analysis will be described in detail in Section 4.3.

4.2 Existing Approaches for Tempo Detection

The issue of beat tracking , tempo induction, and tempo detection has been described by a large number of authors using different models. In general we can distinguish between approaches which try to model, describe, and understand the rhythm perception process of the human brain and approaches that might use knowledge about mental processes but with a primarily focus on the creation of robust models for tempo detection. The existing approaches for tempo detection also can be categorised by the assumed pre-conditions. Some approaches rely on the assumption that the basic (or initial) tempo of a performance is known or constant (*e.g.,* [DH91, Cam00b, CDK00]) while others can be applied to arbitrary expressive performance data. Similar to Seppänen [Sep01] we decided to categorise the tempo detection by the type of algorithmic model that is used (*e.g.,* rule-based, probabilistic). In the remainder of this section we show some typical examples for existing models.

4.2.1 Rule-Based Approaches

One of the first approaches for tempo detection used a set of (musical) rules for tempo detection and has been proposed by Longuet-Higgins and Lee in [LHL82]. Their model is based on the ratio of successive inter-onset intervals (IOI ratios).[6] using the basic assumption that a score can be expressed as an hierarchical tree equivalent to the syntax tree of a Chomsky grammar. The resulting hierarchical structure is similar to the hierarchical model for music as proposed in [LJ83] by Lerdahl and Jackendoff. Longuet-Higgins and Lee propose the use of a Grammar G which depends on the selected time signature of the score. It is assumed that an human listener intuitively builds similar trees when listening to music. The grammar G for the example shown in Figure 4.1 consists of

- a set of non-terminals $V = \{$W, H, Q, E, $\tilde{\text{E}}$, S$\}$,
- a set of terminals $T = \{$w, w', h, h', q, q', e, e', ẽ, ẽ', s, s'$\}$ where w, h, e, ẽ, s denote notes with duration whole, half, quarter, eighth, eighth triplet and sixteenth and w', h', q', e', ẽ', s' the equivalent rests,
- as start symbol the non-terminal W is used,
- and production rules, which for time signature $\frac{2}{4}$ or $\frac{4}{4}$ are shown in Figure 4.1 (left).

The grammar-based approach works for *simple* scores but for more complex scores including syncopations, short notes, or complex tuplets the complexity of the needed grammar would increase significantly. In this case also the task of inferring a syntax tree to a given, fuzzy word of terminal symbols would become a non-trivial task with many ambiguous solutions. Syncopated structures, such as `[ c*1/8 g*1/2 c*1/8 e*1/4 ]` cannot be expressed with this type of grammar without allowing also musically very uncommon or impossible structures (*e.g.,* `[ c*1/8 c*1/12 g*1/2 c*1/8 e*1/4 ]`).

> "...there seems to be no obvious way of constraining the grammatical rules so that they fail to generate such [musically impossible] structures." [LHL82]

For the tempo detection task a small set of five rules: Initialize, Stretch, Update, Conflate, and Confirm is used to extract a beat grid from a list of note onset time positions. The beat duration is initialised with the distance of the first incoming onset times. Parsing now the list of note onsets this beat estimation is changed or confirmed by comparing the measured IOI to the currently estimated beat duration

[6] The authors used the term *relative durations* as description for IOI ratios.

Production rules for grammar G:

$$
\begin{array}{lcl}
W & \rightarrow & HH \mid w \mid w' \\
H & \rightarrow & QQ \mid h \mid h' \\
Q & \rightarrow & EE \mid \tilde{E}\tilde{E}\tilde{E}\tilde{E} \mid q \mid q' \\
E & \rightarrow & SS \mid e \mid e' \\
\tilde{E} & \rightarrow & \tilde{e} \mid \tilde{e}' \\
S & \rightarrow & s \mid s'
\end{array}
$$

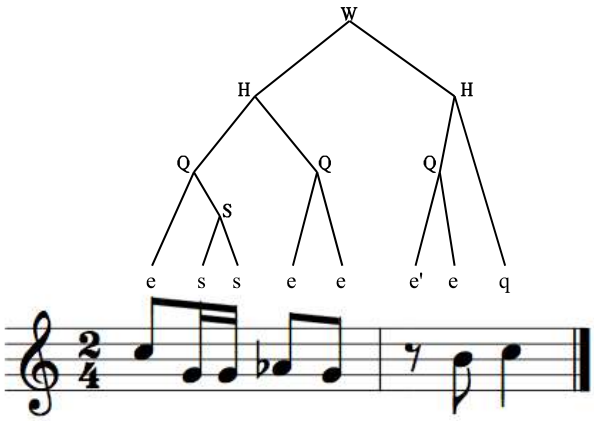


Figure 4.1: Example for a Chomsky grammar for musical rhythm (left) and tree structure for musical rhythm (right) (copied from [LHL82]).

and applying the best matching rule. A significance function (*e.g.,* evaluating the intensity or absolute duration of a note) for the onset time time position is not used.
A detailed description and discussion of this approach (and several improvements by Lee) can be found in [LK94] and [Rip03].

Another rule-based approach for tempo tracking also can be found in [Dri91] (discussed by [Rob96] p.205) Driesse proposes an approach based on a selection of five rules of the grouping structure and metrical structure rules of the Generative Theory of Tonal Music (GTTM) by Lerdahl and Jackendoff ([LJ83]):

1. a relatively long note likely begins on a beat
2. a relatively loud note likely begins on a beat
3. a bass note more likely begins on a beat than a higher note
4. several nearly simultaneous notes likely indicate the beat
5. a change in the harmonic relationship between currently sounding notes likely indicates a beat

Unfortunately Driesse gives no evaluation of his approach and it remains unclear if it has ever been successfully implemented.

4.2.2 Probabilistic Approaches

Beside the fact that nearly all tempo detection approaches need to use at least some statistical methods for estimating any kind of optimum solution, there exist some approaches which are based on statistical models (*e.g.,* Kalman filtering, Sequential Monte Carlo sampling) in a much stronger sense. In the following we show some examples for a way to express the tempo detection issue using such statistical models.

Sequential Monte Carlo Sampling

In [CK02] and [CK03] Cemgil and Kappen propose a model for tempo detection and quantisation based on Sequential Mote Carlo sampling (*i.e.,* particle filtering). Their approach tries to "balance score complexity versus smoothness in tempo deviation" (similar to [Rap01b]) which is realised by avoiding *hard jumps* of local tempo values in the inferred score level data. They describe a model for formulating both, tempo detection and quantisation, as filtering and maximum a posteriori (MAP) state estimation tasks which

are solved by using sequential Monte Carlo integration techniques. Their model uses an ordered set of observed time positions $y_{0:K} = y_0 \dots y_K$ (corresponding to onset times), a set of corresponding score time positions $c_{0:K} = c_0 \dots c_K$, the set of resulting score durations $\gamma_{1:K} = \gamma_1 \dots \gamma_K$ with $\gamma_i = c_i - c_{i-1}$, and a set of corresponding tempo indications $z_{0:K} = z_0 \dots z_K$. The relation between corresponding elements of these three sets is modelled using Bayes Theorem:

$$p(\gamma_{1:K}, z_{0:K}|y_{0:K}) = \frac{1}{p(y_{0:K})} p(y_{0:K}|\gamma_{1:K}, z_{0:K}) p(\gamma_{1:K}, z_{0:K}) \tag{4.14}$$

Their algorithm tries to distribute the performance onset times to positions of a discrete grid, where a generic prior probability for each possible grid position c_k is calculated by $p(c_k) \propto \exp -\lambda d(c_k)$. Here $d(c_k)$ denotes the number of significant digits in the binary expansion of the fraction $c_k \mod 1$ (see [CDK00]) which represents a complexity measure of the grid position c_k. There exists a set of states for $c_k \mod 1$, for example $c_k \mod 1 = \{0, 1/4, 1/2, 3/4\} \cup \{0.1/3, 2/3\}$. It should be noted that the expression $c_k \mod 1$ requires that the length of a measure (bar length) is equal to 1 which represents a whole note. Therefore the time signature (which defines the bar length) must be known in advance.
The grid follows a predefined or generic hierarchical structure of subdivisions which must be defined in advance and depend on performance style (*e.g.,* classical: binary, jazz: ternary) and the time signature. The probability of a complete sequence of quantised time positions $c_{0:K}$ can then be defined as

$$p(c_{0:K}) \propto \exp\left(-\lambda \sum_{k=0}^{K} p(c_k)\right). \tag{4.15}$$

The relation between two successive local tempo values, z_i, z_{i+1}, is modelled as $\Delta_i = \Delta_{i-1} + \zeta_{\Delta_i}$, where Δ_i denotes the inverse of the tempo value z_i and the change of tempo ζ_{Δ_k} follows a distribution $\mathcal{N}(0, Q_\Delta)$ (where Q denotes a covariance matrix). The authors distinguish between tempo fluctuations and notes which have been played 'only' ahead or back in time and therefore do not change the tempo immediately. This model is based on the assumption that the (local) tempo measured directly between successive onsets of a score/performance might be different from the tempo perceived by human listeners.

Cemgil *et al.* model the observed performance time position y_k for onset k as the *perfect*, intended performance position τ_k with an additional noise component ϵ_k: $y_k = \tau_k + \epsilon_k$. Given a sequence of tempo changes (*i.e.,* tempo profile) the perfect intended position τ_k can be calculated by $\tau_k = \tau_{k-1} + \gamma_k \Delta_{k-1} + \zeta_{\tau_k}$, where $\zeta_{\tau_k} \sim \mathcal{N}(0, Q_\tau)$. This equation is equivalent to the standard relation between score.time, performance time, and tempo (Equation 1.1) with an additional noise component ζ_{τ_k}.
For the a priori of the initial tempo, p_{Δ_0}, a Gaussian distribution with a broad variance is used. Using the described definitions the overall relation between tempo, time positions can compactly defined as

$$z_k = \begin{pmatrix} 1 & \gamma_k \\ 0 & 1 \end{pmatrix} z_{k-1} + \zeta_k, \tag{4.16}$$

where $\zeta_k = (\zeta_{\Delta_k}, \zeta_{\tau_k})$.

As shown by the authors ([CK03], Equation 9 & 10) it is possible to define the quantisation problem as a MAP state estimation problem using all the definitions and probabilistic assumptions from above

$$\gamma^*_{1:K} = \arg\max_{\gamma_{1:K}} p(\gamma_{1:K}|y_{0:K}) \tag{4.17}$$

$$p(\gamma_{1:K}|y_{0:K}) = \int dz_{0:K}\ p(\gamma_{1:K}, z_{0:K}|y_{0:K}) \tag{4.18}$$

and the tempo detection as a filtering problem

$$z^*_k = \arg\max_{z_k} \sum_{\gamma_{1:k}} p(\gamma_{1:k}, z_k|y_{0:k}) \tag{4.19}$$

Cemgil *et al.* describe how these problems can be solved by using the Monte Carlo Sampling technique. The offsets of notes are evaluated in a similar way than the note onsets but only for quantisation not during tempo detection.

Unfortunately the authors describe (in both articles [CK02, CK03]) only the evaluation of three examples of a son-clave pattern (see Figure 4.2) including artificial tempo changes and several live performances of the Beatles songs *Yesterday* ([CK02]) and *Michelle* ([CK03]).
Their approach is able to track the (because highly syncopated) hard son-clave example (see Figure 4.2) and also extracts the desired smooth tempo profile of the *Yesterday* performance data files. The authors omitted to publish more detailed information on the number and type of errors in the resulting scores of the output of their system.

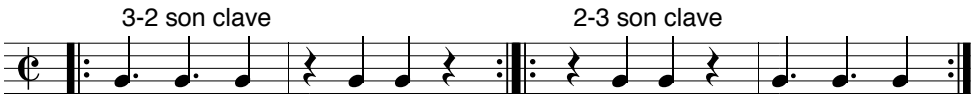


Figure 4.2: The two bar son-clave pattern. In Latin style music, clave patterns usually become continuously repeated during the complete performance by a dedicated percussion instrument (*e.g.,* claves, cow bell). Depending on the rhythm of the melody the pattern starts with the three note bar or two note bar and is then called a 3-2 clave respectively a 2-3 clave.

Kalman filtering

Cemgil, Kappen, Desain and Honing proposed in [CKDH01] an approach for tempo detection based on Kalman filtering and the tempogram representation. The core of this model is also based on the Bayes Theorem (see Equation 4.14). This approach and the results is also briefly discussed in [Dix01b]. Their approach tries to infer the beat positions (in absolute performance time) of tactus strokes.
The used similarity measure evaluates the distance between the beats of a target track (*i.e.,* a sequence of onset times) and the beats of the estimated track. The position of beats of the target track are explicitly given by the input data if they are equal to a note's onset time, but beats between a pair of successive performance notes are also inferred. The beat positions of the estimated track are completely inferred by their algorithm. Similar to the Monte Carlo Sampling (see above) approach Cemgil *et al.* model the tempo-performance-score relation as a dynamic system including state variables and state transitions, including a *noise* component which represents the deviations between performance and mechanical score.

The introduced two dimensional tempogram representation for $x(t)$ is defined as

$$Tg_x(\tau, \omega) = \int dt x(t)\ \psi(t; \tau, \omega) \tag{4.20}$$

where t is an onset list $(t_0, \ldots, t_I)$, τ the local beat, and ω the local log-period. The tuple (τ, ω) represents a possible beat interpretation for t. The function $x(t)$ denotes a continuos signal derived from the onset list t by $x(t) = \sum_{i=1}^{I} G(t - t_i)$, with $G(t) = \exp(-t^2/2\sigma_x^2)$, a Gaussian window function. The function $\Psi(t; \tau, \omega)$ denotes a pulse train defined as $\sum_{m=-\infty}^{\infty} \alpha_m \delta(t - \tau - 2^\omega \cdot m)$, where $\delta(t - t_0)$ denotes a Dirac delta function representing an impulse at time position t_0.

Different from most other systems this approach was tested with 216 performances by 12 different players of the Beatles songs *Michelle* and *Yesterday.* The authors used the performance data set of *Michelle* for training their system and estimating the needed parameters and subsequently evaluated the results for applying the trained system to the *Yesterday* data set. For the evaluation they used an error metric defined as

$$\rho(\psi, t) = \frac{\sum_i \max_j W(\psi_i - t_j)}{(I + J)/2} \times 100. \tag{4.21}$$

where $[\psi_i]\, i = 1, 2, \ldots, I$ are the positions of beats of the target track given in absolute performance time and $[t_j]\, j = 1, 2, \ldots, J$ are the positions of beats of the estimated track. The distance, d, between two beats ψ_i and t_j is weighted by using a Gaussian window function $W(d) = \exp(-d^2/2\sigma_e^2)$. For the width of the window they propose to choose $\sigma_e = 0.04s$. Here false positive inferred beats give only an indirect penalty by a large value for J in the denominator. According to Cemgil *et al.* the model can be adapted to different behaviours by changing its parameters.

Optimal Rhythmic Parse

Raphael proposes in [Rap01b] a probabilistic model for tempo detection combined with a quantisation-like rhythm transcription, based on similar model assumptions than the model of Cemgil *et al.* shown above. Similar to most tempo detection approaches his model evaluates only the onset times of performance notes. He introduces "Rhythmic Parsing" as description for converting a sequence of time positions into corresponding score times. He describes the relation between tempo, score time and performance time positions as a "chicken and egg" problem. The proposed model consists of three separate processes:

1. A *rhythm process* which tries to put the observed performance time positions into a finite set S of allowed score time (measure) positions – similar to the multi-grid approach (see Section 5.2) the time positions in the set S need not necessarily be equally spaced, but they need to be specified in advance. Different from most other models Raphael's model includes transition probabilities for quantised score time positions of successive notes. For example, it is very likely that a note starting at the last sixteenth in a $\frac{4}{4}$ measure will be followed by a note at the first beat of the next measure. To model these transition probabilities a Markov chain approach consisting of an initial distribution $p(s_0)$ and a transition probability matrix $R(s_{n-1}, s_n) = p(s_n|s_{n-1})$ is used. For polyphonic instruments also self-transitions for notes with equal onset times become introduced.

2. A *tempo process* – A local tempo is given by the relation between score IOI, $l(S_n, S_{n+1})$, and observed IOI, $o_{n+1} - o_n$.[7] The initial tempo T_1 (given in seconds per measure) is modelled by a normal distribution ($N(\mu, \sigma^2)$). Successive tempo changes are modelled as a kind of random walk model by $T_n = T_{n-1} + \delta_n$, where $\delta_n \propto N(0, \tau^2(S_{n-1}, S_n))$. The parameter τ^2 takes relatively small values and S_i denotes a position on the discrete grid of score times.

3. An *observable process* – Here it is assumed that the observed note lengths $y_n = o_n - o_{n-1}$ are approximated by the product of the score length of the note and local tempo T_n:

$$Y_n = l(S_{n-1}, S_n) \cdot T_n + \epsilon_n, \tag{4.22}$$

   where $\epsilon_n \propto N(0, \rho^2(S_{n-1}, Sn))$. Similar to the approach of Cemgil the assumptions are now expressed as an optimisation problem and solved using a *maximum a posteriori* (MAP) estimate with dynamic programming.

Raphael trains his model with original performance data. Also the set of possible measure score time positions needs to be calculated in advance and given to the algorithm. Instead of filtering ornamental notes (*e.g.,* grace notes) Raphael adds equivalent score positions to the set of possible grid positions. In his evaluation of Chopin *Mazurka, Op. 6, No. 3* this resulted in the following set of possible grid positions:

$$S = \left\{ \frac{0}{1}, \frac{1}{3}, \frac{2}{3}, \frac{1}{6}, \frac{11}{12}, \frac{23}{24}, \frac{1}{4}, \frac{1}{9}, \frac{2}{9}, \frac{1}{2}, \frac{5}{6}, \frac{1}{12}, \frac{13}{24}, \frac{7}{12}, \frac{1}{24} \right\}$$

We assume that it gets very complex to define such a set in advance without a given score. We also assume that by using a general, regular grid with a high resolution, the number of errors will increase significantly. The output quality will definitely benefit from the Markov chain model, but for an high resolution grid errors can be expected. We assume that a model using explicit patterns (see Section 4.3.2 and Section 5.3) which allow a finer and more intuitive definition of transition probabilities should be preferred.

An Expectancy-Based Approach

In [Des92] an approach for beat induction based on an expectancy function has been proposed by Desain. Similar to [LHL82] the first incoming pair of onset times (the first IOI) is used to create a beat hypothesis which is then updated by later observed time positions. Different from [LHL82], here an expectancy function for the beat positions is created by mathematical methods instead of using an explicit, finite set of rules.

[7] Because Raphael evaluates only onset times he needs not to distinguish between duration and IOI.

For a pair of successive onset time t_i, t_{i+1} and the corresponding $IOI_i = t_{i+1} - t_i$ the next beat is expected with a low probability at $t_{i+1} + \frac{IOI_i}{2}$, with highest probability at $t_{i+1} + IOI_i$, and with decreasing probability at positions $t_{i+1} + n \cdot IOI_i, n > 2$. For a pair of time intervals A, B a basic expectancy function, E_b, is defined as

$$E_b(A, B) = \sum_{R \in \{\frac{1}{m}, \ldots, \frac{1}{2}, 1, 2, \ldots n\}} \text{GAUSS}\left(\frac{A}{B} - R, R, \frac{A+B}{T_{pref}}\right), \tag{4.23}$$

with

$$\text{GAUSS}(x, R, S) = C(R, S) \cdot e^{-D(R,S)x^2}, \tag{4.24}$$

where T_{pref} is set to 600ms (equivalent to a tempo of 100bpm) as the time with highest perceptional sensitivity. A complex expectancy function can be formulated by summing up all expectancy functions of observed inter-onset intervals. For a vector X of basic time intervals with $X = (X_1, X_2, \ldots, X_N)$ Desain defines the time period $S(X, p, q)$ spanned by the possible combination of time intervals p through q as

$$S(X, p, q) = \sum_{i=p}^{q} X_i \quad \text{with } 1 \leq p \leq q \leq N. \tag{4.25}$$

For a given sequence, X, an interval expectancy E_i for a specific time position T can be defined as

$$E_i(X, p, q, T) = E_b(S(X, p, q), T - S(X, 1, q)), \quad \text{with } T \geq S(X, 1, q). \tag{4.26}$$

Desain then defines the complex expectancy $E(X, T)$ of an event at time T as

$$E(X, T) = \sum_{p=1}^{N} \sum_{q=1}^{N} E_i(X, p, q, T), \quad \text{with } T \geq S(X, 1, N). \tag{4.27}$$

The output value of this function can be interpreted as an *expectancy* measure that time position T coincides with the time position of a beat. The position of beats can therefore be derived from the positions of characteristic peaks of function $E(X, T)$. Unfortunately Desain and Honing propose a detailed evaluation of their interesting model only as future work.

In [TNS03] Takeda *et al.* propose a model for tempo detection based on a Hidden Markov Model (HMM) which also can be interpreted as an expectancy-based model. Their model uses trained, statistical knowledge about transitions of score IOI ratios. One basic assumption here is that the IOI ratio for note x_i depends on the (already inferred) IOI ratios of the previous notes $x_1, \ldots, x_{i-1}$. The process of inferring score durations for the performance notes of the input data is modelled as a maximum a priori problem which is solved by the Viterbi algorithm.
Because this approach is very similar to our pattern-based approach we will discuss and compare its details in Section 4.3.2.

4.2.3 Multiple Agent Systems

As shown in the introduction of this chapter there exists always a large high degree of ambiguity for the score duration of beats and the resulting tempo during the tempo detection process. Instead selecting respectively evaluating only a single possibility at each stage of this process, there exist several approaches which try to evaluate several possibilities in parallel and propagate the final selection to a later stage of the tempo detection process. Different systems have been described in the literature that are using a set of *agents* – each one corresponding with a possible beat period or tempo – for beat tracking. Prominent examples for this multiple agent approaches are the beat tracking component of the Cypher system by Rowe [Row93], the *Real-time Beat Tracking System for Musical Acoustic Signals (BTS)* described by Rosenthal in [RGM94], or a more recent system described by Simon Dixon in [Dix01a].
Here Dixon describes an approach for beat induction based on multiple (tempo) agents. The proposed system should be style independent with a main focus on pure beat induction – estimating the tempo and the positions of beats – rather than quantisation or score transcription. The system does not infer the metrical level (the score duration) of the beats directly. It works on MIDI and audio data and is designed

for processing music with a continuous beat. Input data played with very expressive timing will result in higher error rates. Different from many other approaches Dixon evaluates the duration and intensity of notes in addition to the inter-onset interval, but he does not evaluate the inter-onset ratios of successive events. The author distinguishes between tempo induction – inferring a local tempo – and beat tracking – inferring the phase of the beat. As a pre-processing step the approximate simultaneously played onset times will be merged to a *rhythmic event* which is equivalent to a clicknote as described in Section 4.3.1. Dixon uses a threshold for merging several notes to a rhythmic event of 70ms. Similar to the calculation of weights for clicknotes (see Section 4.3.1), Dixon calculates a *salience* value for each rhythmic event depending on the intensity, duration, pitch and number of the merged notes. The rhythmic events then are used for tempo induction and beat tracking.

For the tempo induction part, his system then clusters all direct inter-onset intervals $t_{i+1} - t_i$ and also the compound inter-onset intervals, where a compound inter-onset interval is defined as $t_{i+a} - t_i$, for $a > 1$.[8] Each cluster should consist of intervals with a similar length. Dixon uses a tolerance of 25ms between the observed interval and the current mean of a cluster. It is allowed that the mean of a cluster may shift during the analysis. If no cluster exists for an observed interval a new one will be created. After the evaluation of the inter-onset intervals clusters might by merged if their mean values are very close, then they become ranked by the number of attached elements. If the mean value of a cluster, c_i, is an integer multiple d of a cluster c_j, c_i and c_j will be viewed as related. For the strength of this relation Dixon defines a function

$$f(d) = \begin{cases} 6 - d, & 1 \leq d \leq 4 \\ 1, & 5 \leq d \leq 4 \\ 0, & \text{otherwise.} \end{cases} \tag{4.28}$$

We would prefer a more continuous definition for $f(d)$ (*e.g.*, $e^{-\alpha d}$) which might be easier to control. For the ranking of clusters then the number of elements of related classes, weighted by $f(d)$ is added to their own weight.

This part of Dixon's approach is very similar to the tempo detection algorithm we proposed in [Kil96]. Here also classes of notes with similar durations are build and can be merged if their average values had become very close. Different from Dixon's approach, for each of these classes a correspondent score duration is directly inferred without other higher level steps. By using the clustering just as a kind of pre-processing for more advanced steps, Dixon obtains better results than the approach described in [Kil96].

After the clustering then from the top ranked clusters a set of hypothesis for the basic tempo of the piece can be inferred. (With the constraint that the tempo does not change very much during the piece!). For each hypothesis an *agent* is instantiated. By using the hypothesis the beat rate can be inferred. The beat phase – the actual position of beats – is still unknown. The instantiated set of beat agents now are used for tracking the beat positions starting at the beginning of the piece.

Each agent can be characterised by his state – his current hypothesis about the beat rate, beat phase and its history – and the set of events which have been accepted by the agent. Because it is not known if the first event is on a beat or not, for each hypothesis not only a single but a set of agents will be instantiated, where the agents of a set will start at different events at the beginning of the piece. From a more musical view this can be justified by the fact that at least one of the event at the beginning of piece must be on the beat. Dixon creates agents for all events in the first five seconds of the input data. If the introduction of a piece does not have any real tempo (rubato intro) all agents will start with a wrong tempo and a wrong phase. Dixon states "...an agent with the approximately correct tempo will be able to adjust its tempo and phase in order to synchronise with the beat."
Depending on his current state each agent can make predictions of expected beat positions by adding integer multiples if its current beat duration to the last beat position. If an incoming event is inside a so-called inner window (size = 40ms) of a predicted beat position the event is accepted as a beat, the agent's tempo is updated, and a fraction s' of the event's salience s is added to the agent's score. The fraction depends on the relative error between predicted and observed beat time and the width of the window. It is normalised to $0.5 \cdot s \leq s' \leq s$. If an incoming event falls into the so-called outer window (size = 20% of beat duration before and 40% of beat duration after the predicted beat) of a predicted

[8] The total number of possible inter-onset intervals (including the compound intervals) for a set of n rhythmical events is in $O(n^2)$

beat position, the agent accept the beat as a new beat and updates its tempo, phase and score. Because the chance for an error is very high, a copy of the agent with its state before accepting the event will be created. If an incoming event is outside the inner and outer window it will be ignored.

Instead of the proposed hard limited window size (inner and outer) a smooth distance or probability function (*e.g.,* a Gaussian window function as used in other approaches) might be a more appropriate solution.

If a situation arises where the beat duration of two agents are closer than 20ms and the agents are in phase (accepting both the current event) the agent with the lower score will be deleted. After sending all rhythmical events to the agents, the correct beat positions and the tempo can be inferred from the agent with the total highest score. It is easy to see that (each) agent needs to remember the time positions of predicted and observed beats.

Different from most other authors Dixon also discusses the meaning and calculation of musical salience depending on duration, pitch, and intensity of notes. In a more recent work [DGW02] Dixon *et al.* implemented this approach combined with a very powerful graphical output system for tempo and loudness variation visualisation. A sample output from this system is shown in Figure 4.3. The front end and user

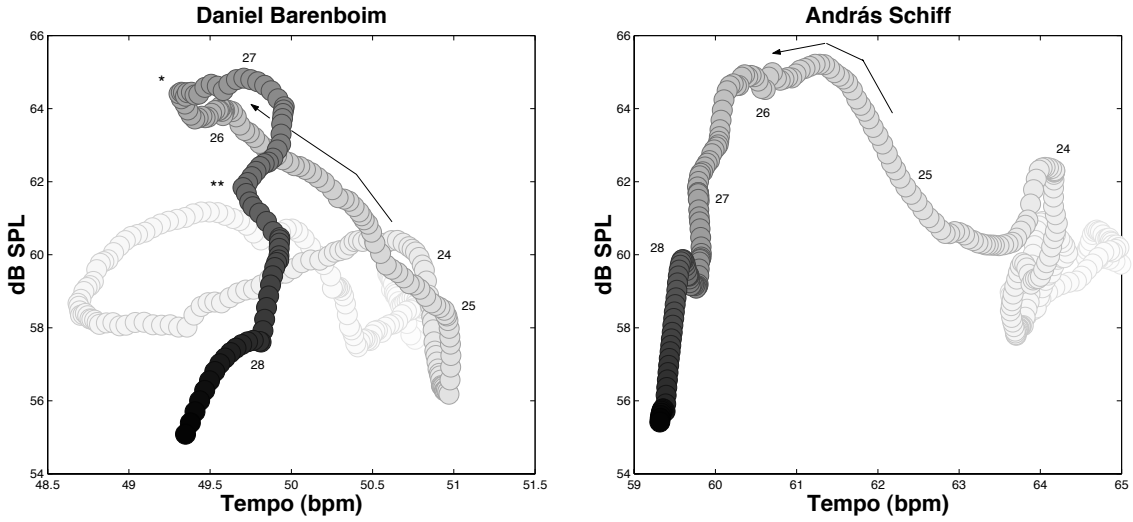


Figure 4.3: Sample output of two different performances of the last bars (measures 24–28) Mozart's *Piano Sonata K. 279*, second movement, first section. x axis: tempo in beats per minute; y axis: dynamics ('loudness') in decibel. The darkest point represents the current instant (third beat of measure 28), while instants further in the past appear fainter. (copied from [DGW02])

interface of this system seems to be very powerful and in terms of beat tracking and dynamic visualisation the obtained results also seem to be very good. Different from our thesis the main focus of this system is on the analysis, understanding, and visualisation of expressive timing and its relation to expressive dynamic (intensity) of music and not on the estimation of performance-to-score mapping in general.

4.2.4 Multiple Oscillator Models

Large and Kolen introduced in [LK94] and [Lar94] an approach for beat tracking based on an oscillator model. This models is somehow similar to the multi agent systems described in Section 4.2.3. But different from these models the *integer multiples* of beat intervals are described as the frequency (or rate) of an complex oscillator. Instead a set of agents here a set of coupled oscillators is used, where the different oscillators represent the different metrical levels of a score. Usually the sets of oscillators are much smaller than the set of agents of a multiple agent approach. Both approaches target on finding

the frequency and phase of the beats at the tactus level and also on higher metrical levels. But they do not try to infer the complete metrical structure (quantisation). In a recent approach by Large and Palmer ([LP02]) a dedicated complex oscillator is used for each metrical level. Each oscillator should be synchronised in phase and rate by incoming events, where the strength of adjustment depends on the distance between the observed time position and the predicted time position estimated from current rate and phase of the oscillator. "...the oscillator attempts to synchronize to events that occur near 'the beat' ...while ignoring events that occur away from the beat." ([LP02]). The strength of influence is calculated by a Gaussian window function defined in a circle ("von Mises distribution"). The oscillators of different levels are also coupled with each other. This simulates rate and phase adaptation of higher level oscillators through incoming events at lower levels.
Different from the multi agent approach in [Dix01a], here no clustering of inter-onset intervals is done in advance instead the initial rate/period is set manually to the first observed inter-onset interval and its score representation.
Even if the oscillator model of Large was designed for beat tracking only it should be possible to use it also for quantisation/transcription tasks by attaching a score duration to each oscillator (representing a metrical level).

4.2.5 Connectionist Approaches

As discussed by Large in [LK94] the standard models of recurrent networks have difficulties in handling temporal structures: "A network trained to recognize a melody played at 80 beats per minute, for example, may not recognize the same melody played at 90 beats per minute". Therefore special adaptations of artificial neural nets need to be developed for the area of tempo detection or beat tracking. We assume that this might be the reason that native neural network approaches for tempo detection are kind of uncommon.

Roberts proposes in [Rob96] an approach for a neural foot-tapper based on a self-organising neural network (SONNET). Because the main focus of this work is on pattern recognition by neural net approaches his foot-tapper module represents only a kind of pre-processing within a larger system. The output of the net is directly send to the pattern recognition part of his system (see also Section 3.4 for a brief description). We assume that this might be the reason why he does not give a detailed evaluation of the foot-tapper (beat induction) implementation that could be compared to other approaches.

Ohya shows in [Ohy94] a possibility to model beat/rhythm perception by a pair of neurones that represent a neural rhythm generator. Similar to Robert's approach also here it is not clear how and if this approach can be applied to score related implementations.

4.3 Hybrid Tempo Detection

Similar to the previously discussed existing approaches for tempo detection the output (result) of the in the following proposed newly developed hybrid approach will be an algorithmically generated (not manually played) clicktrack.[9] This clicktrack gives then the required mapping between performance time information and score time information. Different from a manual created clicktrack (see Section 4.1.1) and different from approaches which try to estimate a regular pulse or beat, here the clicknotes of the generated clicktrack will have different (but discrete) score durations. Instead of inferring the actual beat of a performance our approach tries to estimate directly score durations (inducing score onset times) for the inter-onset intervals of the clicknotes, that are created from the onset times of observed performance notes.
Different from the described approaches in the previous section – where always a single model in each approach has been used – the here proposed approach uses a combination of pattern matching (structure oriented) for inferring groupings of notes and statistical analysis for inferring score information for single notes. The approach works in two phases: first a pattern matching between patterns of a database – containing rhythmic patterns – is performed and then for all regions where no best matching pattern can be found a statistical tempo detection, evaluating only single notes, is performed.

In the following first in Section 4.3.1 the creation of a clicktrack – containing only performance information at this initial step – is described. Then we described in Section 4.3.2 the pattern matching and in Section 4.3.3 the statistical tempo detection part of our approach. In Section 4.3.4 and Section 4.3.5 we describe a method for error detection and for estimating additional general accuracy information from the input data.

4.3.1 Merging Performance Data Into a Clicktrack

As initial step here all onset time positions of all notes of all voices become merged into a single clicktrack by creating clicknotes at their observed performance onset time positions. The score time information (*i.e.*, $onset_{score}, duration_{score}$) of these clicknotes is unknown at this initial step. The onset times of notes of different voices with very close onset times become merged to a single clicknote c, assuming that the intention was to play them at an equal time position.
Let $M = \{m_1, \ldots, m_{|M|} \mid onset_{perf}(m_i) \leq onset_{perf}(m_{i+1})\}$ be the already known complete set of observed notes (see Section 2.2.1), then M can be partitioned in $p \leq |M|$ disjoint subsets, M_q, of notes with equal or very close onset times:

$$M = M_1 + \ldots + M_p, \text{ with } \forall\, i \neq j : M_i \cap M_j = \emptyset \tag{4.29}$$

with

$$\begin{aligned} M_q = \{\ m_i, \ldots, m_{i+e} \mid l = 0, 1, \ldots, e : m_{i+l} \in M \wedge onset_{perf}(m_{i+e}) - onset_{perf}(m_i) < \epsilon \\ \wedge\, onset_{perf}(m_{i+e+1}) - onset_{perf}(m_i) \geq \epsilon \\ \wedge \forall\, j \geq i : m_j \notin M_{q-1} \wedge \forall\, j \leq i+e : m_j \notin M_{q+1}\ \} \end{aligned} \tag{4.30}$$

In general $M_q \subset M$ is true, only if a piece would contain only a single note, or only notes played at the same performance time and split into separate voices, or a single chord, $M_q = M$ can be true.
The size of ϵ can be derived from some known perceptual thresholds: two notes with onset times closer than approximately 40-50ms are perceived with equal onset times [Dix01a]; for more than two notes the threshold is approximately 70ms [Dix01a]. If ϵ is chosen small enough then situations where for all notes m_i with $a \leq i < b : onset_{perf}(m_{i+1}) - onset_{perf}(m_i) < \epsilon$ and at the same time $onset_{perf}(m_b) - onset_{perf}(m_a) \geq \epsilon$ can only be caused by series of ornamental notes, such as trills or glissandi. In the following it is assumed that these ornaments have been detected and filtered by the ornament detection module (see Section 6.4) before starting the tempo detection. For each $M_i \subseteq M$ now an *extended* clicknote, c_i, will be created (see Figure 4.4). This should be a basic clicknote as defined in Section 4.1 extended with some additional information about the notes in the set M_i. It follows that our clicktrack

[9] The clicktrack concept is described in detail in Section 4.1.

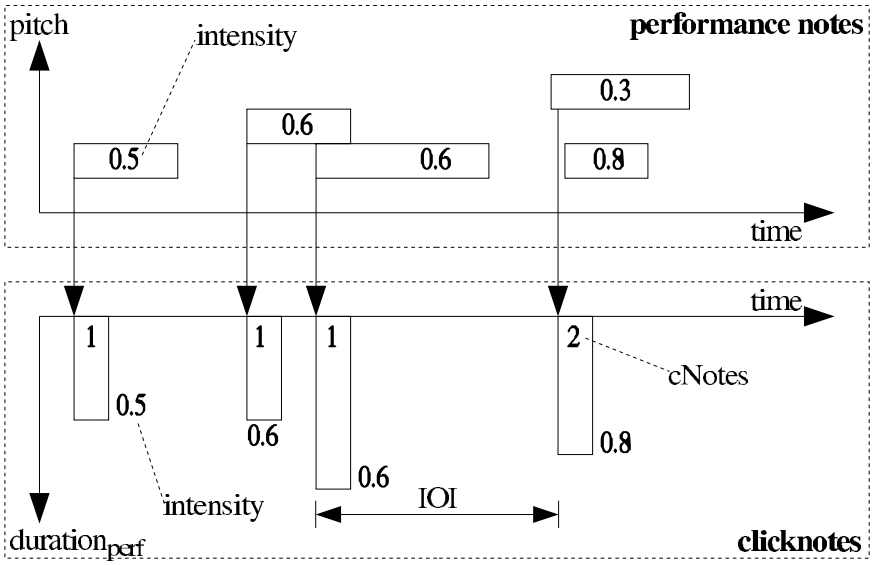


Figure 4.4: Merging of performance notes to clicknotes.

C consists of p clicknotes: $C = \{c_1, \ldots, c_p\}$. The data vector for an extended clicknote c_i is given by

$$c_i = (onset_{perf}, duration_{score}, cNotes, intens, duration_{perf}), \tag{4.31}$$

with

$$\begin{aligned} cNotes &= |M_i| \\ intens &= \max_{m \in M_i} \{intens(m)\} \\ duration_{perf} &= \max_{m \in M_i} \{duration_{perf}(m)\}. \end{aligned}$$

For the special case of a chord note $m' \in M_i$, each note of the chord will be evaluated as separate element of M_i. The weight function for clicknotes also gives respect to Large who states: "Phenomenal accent can be conferred upon an event by the manipulation of many possible physical variables, including duration, pitch, and intensity." ([LK94]).

Different from the basic clicknote as defined above, with directly measured performance time information now the performance time for an extended clicknote c_i (denoted as $onset_{perf}(c_i)$) is the average of all performance onset time position of the notes $m \in M_i$. Assuming that longer notes are more prominent and therefore played with a higher accuracy then short notes, they might have a higher influence to the average onset time than short notes. But because of the rather small size of ϵ – the time interval for the notes in M_i – this weighting cannot change the onset time position of c_i significantly. All other functions defined for the basic clicknotes in Section 4.1 can be applied to the extended clicknotes without any changes. Analogous to the basic clicknote definition we assume that there also exists a set of functions $cNotes(c_i)$, $intens(c_i)$, and $duration_{perf}(c_i)$ for retrieving the corresponding entries of the clicknote data vector.

After performing the merge operation, in best case the clicknotes of an optimal clicktrack represent a regular, equal spaced, complete grid of onset times. The distance between two successive clicknotes – given by their IOI – would then be equal to a tactus level beat. In average case the clicktrack will include clicknotes with different score durations and IOIs different from the tactus level. A large distance between clicknotes result from a passage where all voices include only long or even none performance notes (before the merge operation). Short distanced clicknotes with a small IOI are resulting from short, insignificant

notes in at least one of the merged voice or notes with slightly different onset times distributed to different voices.

For each clicknote, c, a weight function $w(c)$ indicating the salience or significance of a clicknote c can be defined. This weight function will be evaluated during the statistical analysis. This weight of a clicknote, c, depends on

- $cNotes(c)$ – the significance of a score positions t is high if many notes start at t.
- $intens(c)$ – notes with an high intensity are perceived as accented ([LHL82]) and should have more influence to the rhythm perception than soft played notes.
- $duration_{perf}(c)$ – long notes have more influence to rhythm perception than short ones.
- $IOIratio_{perf}(c)$ – so-called *durational* or *agogic* accents are notes with a longer duration than their neighbour notes. They can be perceived as anchor points (see [MRRC82]) for rhythm perception. Notes, longer than their neighbour notes imply an accent ([LHL82]). A durational accent is also a special kind of *phenomenal* accent introduced in [LJ83]. The IOI ratio can be used as measure for the strength of durational accents The characteristics of these accents will be discussed in detail in the following paragraph.

These weight measures are similar to the three accent type categories described in [LJ83] (see also [Rob96]): phenomenal, structural, and metrical accents. They are also similar to the salience of notes as calculated in [Dix01a].

Durational Accents. Notes with a local maximum IOI are so-called *durational accents*. Because they are longer than the notes in their neighbourhood they will be perceived as accented, even if their intensity is equal or lower than the neighbours intensity ([LHL82]). The phenomenon of accented notes is addressed in a similar way by other authors: Lerdahl and Jackendoff define in [LJ83] so-called *phenomenal accents* (see [PK90]) also Dannenberg's model, for example, uses *accented* notes ([DMR87]). .

> "If the durations in a note pair are significantly distinguishable as short and long values in succession, then the latter one is marked as *agogic accent*." [MRRC82]

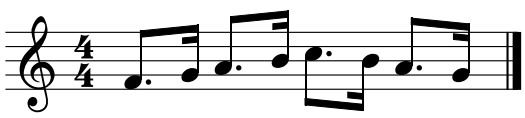


Figure 4.5: Example for notated durational accents and inverse durational accents.

Notes which are very short compared to their neighbours will in the following be called *inverse durational accents*. Experiments showed that the evaluation of these inverse durational accents during tempo detection increases the error rate; during the pattern matching step they reduce the number of correct pattern matches. During single note processing their correct score duration is hard to detect, because they are often the result of incorrect playing and show very unusual time position, which also has influence to the duration of the previous or following note.

Typical examples for inverse durational accents – beside inexact playing – are groups of dotted eighths followed by sixteenth notes as shown in Figure 4.5. Evaluation of performance data showed – depending on player and style – the onset time position of, for example, the sixteenth notes might be very inexact; sometimes too far away from the correct position that even the pattern matching fails. It is easy to see, that the removal of these sixteenth notes, or in general the removal of inverse durational accents, increases the regularity of the clicknote grid. This can be seen equivalent to removing a noise component from the clicktrack (beat) data which will improve the output quality of the tempo detection module. The score time information for the removed (filtered) clicknotes can be obtained later by the quantisation module and need not necessarily be inferred by the tempo detection module. The here defined inverse durational accents are similar to *weak* notes as defined in [DMR87]. For identifying them we define a

function $duracc(c_j)$ which gives a measure for the durational accentuation of a clicknote c_j as

$$duracc(c_j) = \begin{cases} W\left(\dfrac{IOI_j}{\max\{IOI_{j-1}, IOI_{j+1}\}}\right), & \text{if } IOI_j > \max\{IOI_{j-1}, IOI_{j+1}\}, \\ -W\left(\dfrac{\min\{IOI_{j-1}, IOI_{j+1}\}}{IOI_j}\right), & \text{if } IOI_j < \min\{IOI_{j-1}, IOI_{j+1}\}, \\ 0, & \text{otherwise,} \end{cases} \tag{4.32}$$

where IOI_j denotes $IOI_{perf}(c_j)$ and $W(x) = 1 - W_{Gauss}(x-1, \sigma)$. Applying the Gaussian window function W to the fraction of IOIs is similar to the calculation of the strength of phenomenal accents as proposed in [Par94b](p. 431).

For calculating the total weight $w(c)$ of a clicknote c we define several weight functions for the significance of specific features of a clicknote c:

$$\begin{aligned} w(duration_{perf}(c)) &= 1 - W_{Gauss}(duration_{perf}(c, \sigma_1)) \\ w(intens(c)) &= intens(c) \\ w(IOIratio(c)) &= duracc(c) \\ w(cNotes(c)) &= 1 - W_{Gauss}(cNotes(c, \sigma_2)) \end{aligned}$$

Using this feature weights the total weight for a clicknote c can be defined as a linear combination of these functions:

$$w(c) = \alpha \cdot w(duration_{perf}(c)) + \beta \cdot w(intens(c)) + \gamma \cdot w(cNotes(c)) + \delta \cdot w(IOIr(c)), \tag{4.33}$$

with $\alpha + \beta + \gamma + \delta = 1$.

For the filtering operation we now search for all clicknotes $c_i \in C$, with

$$\begin{aligned} duracc(c_i) < \tau \wedge w(c_i) < w(c_{i+1}) \\ \text{or } duracc(c_{i-1}) < \tau \wedge w(c_{i-1}) > w(c_i) \end{aligned} \tag{4.34}$$

Because the inverse durational accent might be the result of a clicktrack merge operation on polyphonic data, we remove from the clicktrack C that clicknote of the pair c_i, c_{i+1} with the lower weight $w(c)$. In first tests we estimated the threshold τ dynamically by evaluating the mean μ and the variance σ of the weights of all clicknotes in the unfiltered clicktrack. Tests showed that a fixed threshold (in the current implementation τ is set to -0.8) is also sufficient.

	complete clicktrack	filtered clicktrack
# c-notes	196	141
mean	117.12	115.35
stddev	27,41	24,50
min	46.08	46,08
max	200	168.42

Table 4.1: Chopin, *Op. 6, Mazurka 1*, measures 1–36: statistical values for performed tempo in bpm.

If too less insignificant clicks are filtered from the clicktrack, the resulting clicktrack, $C' \subseteq C$, will contain many incorrect played notes resulting in hight or low peeks or glitches in the tempo profile. If too many significant clicknotes are filtered (removed) from the clicktrack the profile will be smoother but the distance between the clicknotes will be increased what could lead to more errors during the later performed quantisation.

A perfect filter algorithm would filter all notes except the tactus level beats of the performance. If the filtered clicktrack includes still enough clicknotes to infer a score time for the major beat positions then all other onset times can be approximately inferred by a linear normalisation between successive beats. The approximate onset time positions can then be finalised (corrected) later by the quantisation module which will be called after the tempo detection. In best case we obtain now a complete, regular, equidistant grid of onset times given by $onset_{perf}(c), c \in C'$, where all clicknotes $c \in C'$ should have a similar weight $w(c)$.

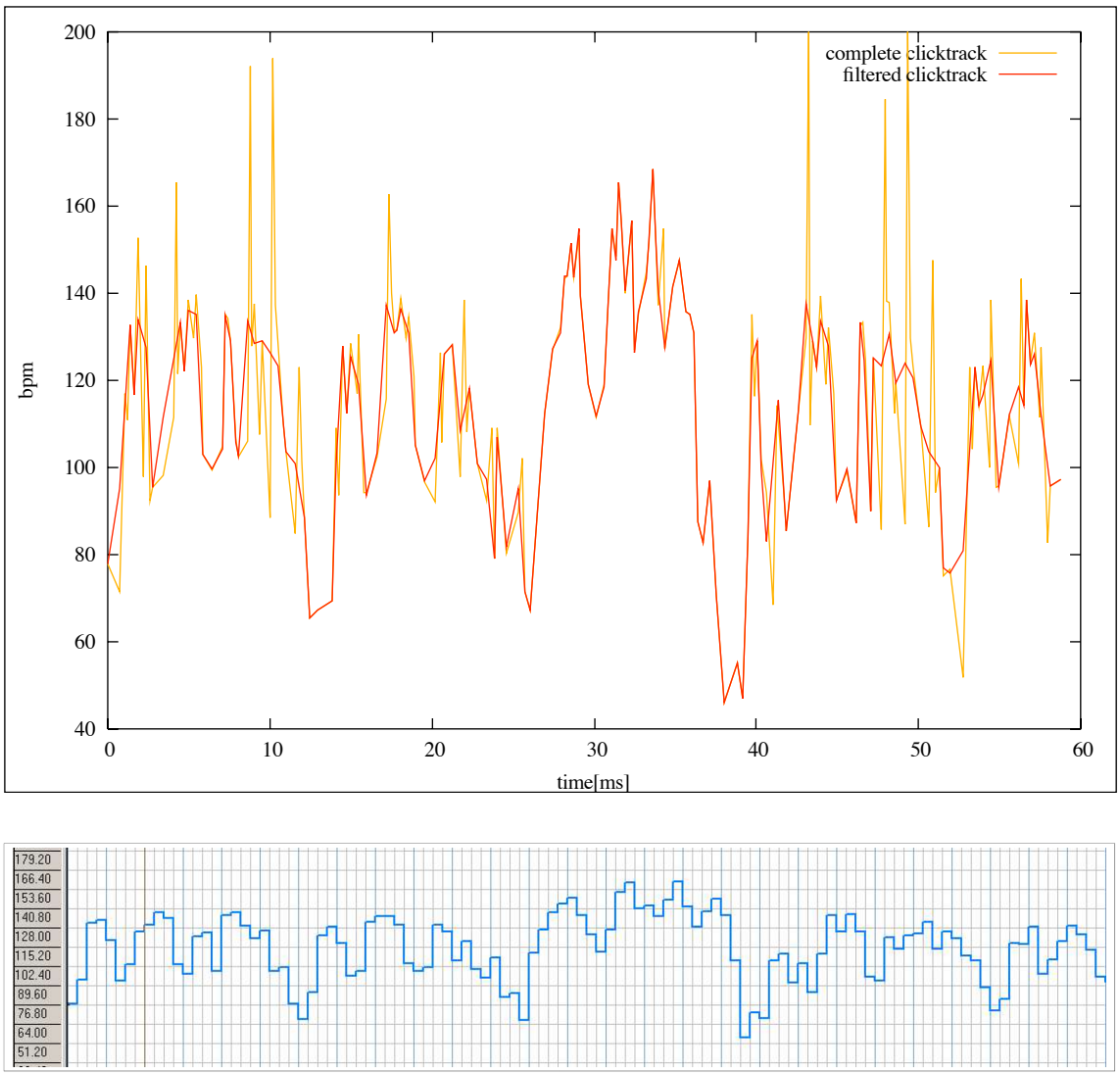


Figure 4.6: Chopin, *Op. 6, Mazurka 1*, measures 1–36: performed tempo calculated at all 196 onset time positions of a merged clicktrack (top, bright curve); performed tempo calculated at 143 filtered significant clicknotes (top, dark curve), and performed tempo calculated for each quarter note beat positions of a performance (bottom).

The analysis of the tempo fluctuations of an expressive performance of Chopin's *Op. 6, Mazurka 1* showed that the fluctuations are spread over a wide range (see Figure 4.6(top) and Table 4.1), where a statistical analysis also showed that they follow a normal distribution (Figure 4.7). The large up and down peeks in Figure 4.6(top) result from small errors in absolute timing with a high influence on the relative timing given by the IOI ratio. If, for example, a semi-quaver melody note, followed by a crotchet at a desired tempo of 100bpm is played with an IOI of 120ms instead of (the mechanical) 150ms the absolute error (shift) is only -30ms but the IOI ratio changes from 4 to 5 and the local tempo increases to 125bpm. If we assume that the reduction of the first note's duration is compensated by a prolongation of the second note, the IOI ratio changes actually to 5.25. As shown in Figure 4.6(bottom) the undesired peaks of the local tempo plot can be reduced by using the filtered clicktrack. A calculationon on the quarter beat level (manually marked) shows that on this metrical level the local tempo curve is again smoother. Figure 4.6 shows an example of tempo deviations for an expressive piano performance (Chopin, Op.6 *Mazurka I*)

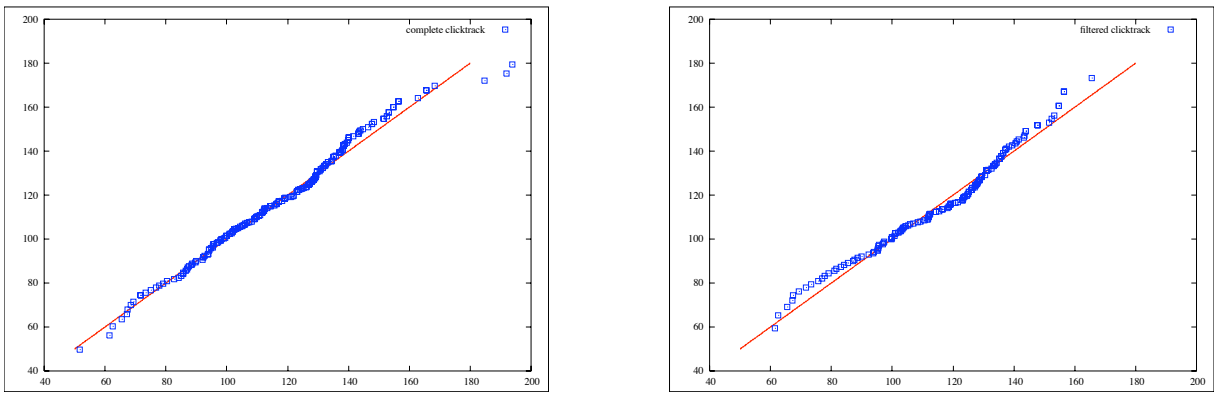


Figure 4.7: Chopin, *Op. 6, Mazurka 1*: qq-plot of performed tempo calculated from all onset time positions of a merged clicktrack of an expressive performance (left); qq-plot of performed tempo calculated from significant clicknotes after filtering the clicktrack (see also Table 4.1.)

evaluated for all events of a merged, unfiltered clicktrack (top, bright graph), all events of after filtering (top, dark graph) and evaluated only for events at quarter beat positions of (bottom) (see Section A.12 and Section A.13 for details about the data).

After the clicktrack creation (including only performance time information) and the filtering process, now the – still unknown – score time information (onset times and durations) need to be inferred for the filtered set of clicknotes C'. As shown in Section 4.2 there exist various approaches for this task. In the following a new hybrid approach is proposed which uses a combination of pattern-based tempo detection and a statistical model for single note processing. In a first step the data of C' is matched against (rhythmic) patterns of a pattern database and for all regions where no matching pattern can be found then a single note tempo detection strategy is applied.

4.3.2 Pattern for Tempo Detection

The development of our module for statistical analysis tempo detection (see Section 4.3.3) showed that there exist situations where the correct score duration of a note might not be inferred correctly without knowledge about typical, standard rhythmic patterns or groupings. For example, if the duration of a clicknote was recognised to be approximately the duration of an triplet quaver (1/12) it should only become a triplet, if the note itself and its neighbour notes build a complete triplet group ($3n \cdot 1/12$).

> "Simon and Sumner [SS68] propose that listening to music could similarly be modelled as a process of pattern induction and sequence extrapolation, using alphabets and rule-based transformations, such as same (repeat) and next (next element in the alphabet)." [LPP95]

> "Listening to a piece of music is similar in at least one important way. For even moderately complex pieces, most listeners do not literally remember every detail; instead, they understand a complex piece by a process of abstraction and organization, remembering its musical 'gist'." [LPP95]

The assumption that tempo detection can benefit from the use of pattern is also supported by psychological studies:

> "Other studies have demonstrated similar memory constraints, by showing that the reproducibility of rhythms is affected by the patterns of phenomenal accentuation in the to-be-reproduced rhythm. The evidence suggests that sequences of events that imply a metrical organization are easier to memorize and reproduce than sequences lacking such organization (Essens & Povel, 1985; Povel & Essens, 1985)." [LK94]

There exist different approaches how the knowledge about patterns could be modelled:

- Rule-based – a large, complex set of rules would be required; it would be inconvenient (or impossible) for users to extend the rule system with new patterns.
- Neural networks – the algorithm would require to be trained with a large number of performance data sets and corresponding (correct) score data sets. This would require a high effort in creating training data; the score-performance matching during the training would require a large amount of manual work.
- Hidden Markov Model (HMM) – In [TNS03] Takeda *et al.* propose an approach for tempo detection based on HMM. Their basic assumption is that the score IOI ratio of a clicknote, c_i, statistically depends on the score IOI ratios of the previous clicknotes $c_1, \ldots, c_{i-1}$ and on the obvious fact that the IOI ratio is robust against small tempo changes. Similar to neural networks in general, Hidden Markov Models require training and show therefore the same disadvantages as neural nets.
- Explicit score pattern database – The rhythmic patterns are stored explicitly in a pattern database; a distance function gives the distance between performed data and score data (*i.e.,* a pattern); the database can easily be extended by the user or automatically by the system.

The approach of using a pattern database with explicit pattern defined in score notation has the advantage that at the initial step (before any usage or training) only the score data of the pattern must be given. Using the score-performance distance function the system needs not to be trained. If required the approach can be extended with machine learning aspects where (automatically) learned information about typical inaccuracies between performance and score data can be stored in the pattern database and also evaluated by the distance function. Therefore this strategy was chosen for the here described pattern-based tempo detection approach.
For a given set of rhythmic patterns $\mathcal{P} = P_1, \ldots, P_{|\mathcal{P}|}$ (*i.e.,* the pattern database) and a (filtered) click-track, C, our approach can be described with the following outline:

1. Select for clicknote $c_i \in C$ a set $\mathcal{P}_{best} \subseteq \mathcal{P}$ of the n best matching patterns ($n = |\mathcal{P}_{best}|$) by using a distance function $d(c_i, P)$ for $P \in \mathcal{P}$ which evaluates the structural information (IOI ratio) of the pattern P and the context information (local tempo) given by a previously matched pattern.
2. If possible, select a pattern $P_a \in \mathcal{P}_{best}$ as new match starting at c_i according the following constraints:
   - The pattern P_a must have a small distance to the performed data, below a threshold, τ: $d(c_i, P_a) < \tau$.
   - If the current clicknote c_i is in the range of a previously matched pattern P_m that starts at position c_j, then P_a starting at c_i must exceed the range of P_m and the postfix of P_m and the prefix of P_a must be equal for c_i and successive notes. This case is indicated by $eFit(P_m, P_a, i - j) = true$ (see Figure 4.8, Equation 4.35 for the definition of $eFit$).
     If the distance between c_i and P_a is much smaller then the distance for P_m and c_j ($d(c_j, P_m) \ll d(c_i, P_a)$), P_a can be accepted as a match starting at c_i, where in this case the previous match c_j, P_m might be completely discarded.

   If a pattern $P_a \in \mathcal{P}_{best}$ satisfies these constraints, set $P_m := P_a$.
3. Proceed with clicknote c_{i+1}.

Our tests showed that with a size of $n = 3$ sufficient results can be obtained (see Section 4.4). At first glance the calculation of a distance between all $|C|$ clicknotes and all $|\mathcal{P}|$ patterns of the database seems to be very exhaustive (in worst case $O(|C| \cdot |\mathcal{P}|)$), but the calculation of a single distance measure is very fast and the calculation of all distance measures can be optimised by using abort criteria for the calculation of the distance function between a pattern and a sequence of clicknotes.

Initially the pattern-based tempo detection should be implemented as a two phase model: first select n best matching pattern for every clicknote $c_1, \ldots, c_{|C|}$ using the distance function d and then in step

Given two pattern $P, P' \in \mathcal{P}$, $P = (p_1, p_2, \ldots, p_{|P|})$ and $P = (p'_1, p'_2, \ldots, p'_{|P'|})$, and an index distance δ:

$$eFit(P, P', \delta) := \begin{cases} true, & \text{if } \forall\ i = 1, 2, \ldots, (|P| - \delta) :\ p_{\delta+i} = p'_i \wedge |P'| > (|P| - \delta) \\ false, & \text{else} \end{cases} \tag{4.35}$$

If the last j clicknotes (postfix) of a pattern P (with length n) have the same duration than the first j clicknotes (prefix) of a pattern P' (with a length $> j$) then the function $eFit(P, P', \delta)$ returns *true* for $\delta = n - j$.
If the complete pattern P' is an infix or postfix of pattern P then *eFit* will return *false*.

Figure 4.8: Definition of extension fit function $eFit(P, P', \delta)$.

two select a set of overlapping pattern for the complete performance using all n best pattern of each clicknote as input of an optimisation algorithm. The optimisation approach ensures a minimal deviation between the score created by the complete set of pattern and the performance data. If using this strategy, there exist no information about a current tempo during the first phase (selection of n best pattern for each clicknote) *i.e.,* when processing clicknote c_i it is unknown which patterns have been selected for the previous notes. In this case the distance function d could evaluate only the structural information of the IOI ratios. Tests showed that a distance measure based only on structural information leads to a high rate of false positive pattern matches, because of ambiguities in the IOI ratios of different pattern. Therefore a single pass architecture combining the calculation of n best patterns and a final selection of one of these best patterns for each single note was chosen.

In the remainder of this subsection the details of the distance function d and the estimation of the matching threshold τ will be described.

The Pattern Distance Function

As shown in Figure 4.9 the significant IOI ratios of a rhythmic pattern of the original score (bottom) are still existent in the correspondent expressive played performance data (top). So it should be possible to use the observed IOI ratio information of the performance and compare it to the IOI ratio information of a pattern for calculating a distance measure. The distance $d(c_j, P)$ between a pattern $P = (p_1, \ldots, p_{|P|})$ and a sequence $\tilde{C} = (c_j, \ldots, c_{j+|P|-1}) \subseteq C$ of successive clicknotes (sorted ascending by their onset times) should consist of two separate distance measures:

1. The *structural distance* d_{IOIr} depending on the IOI ratios of pattern notes and performance notes.

2. The *context distance* d_{norm} depending on the local relation between score and performance timing.

For defining the context distance we first define a normalisation distance measures f_{norm}, which depends on the local relation between score and performance time.
Because a pattern, P, is given with score time information and the clicknotes of C in performance timing (*e.g.,* milliseconds or MIDI ticks), we define the normalisation factor, f_{norm}, for applying a pattern, P, to a sequence of clicknotes starting at c_i (pattern match) as

$$f_{norm}(c_i, P) := \frac{c_i}{p_1}, \quad \text{where } p_1 \text{ represents the first note of pattern } P. \tag{4.36}$$

Assuming that for all clicknotes $c_1, \ldots, c_{i-1}$ the pattern matching has already been performed (using the pattern database $\mathcal{P}$), we can define a function F returning a normalisation factor for each clicknote c_j

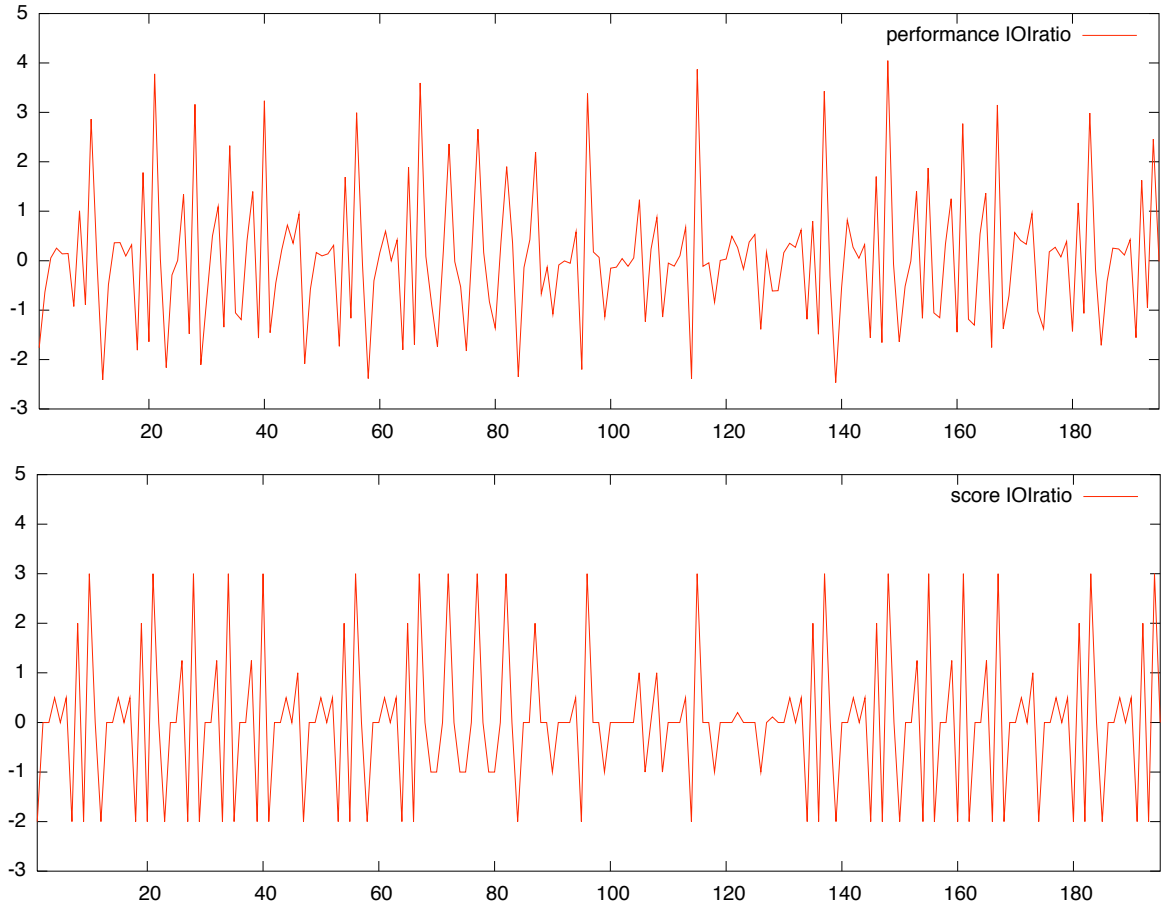


Figure 4.9: Chopin, Op. 6 *Mazurka no. 1*, measures 1–36: performed IOI ratio (top) and score IOI ratio (bottom) in unfiltered clicktrack (see also Section A.12 and Section A.13).

with $j < i$ as

$$F(c_j) := \begin{cases} F_{def}, & \text{if } j = 0, \\ F_{norm}(c_{j-1}), & \text{if no pattern match for } c_j, \\ \alpha \cdot F_{norm}(c_{j-1}) + (1-\alpha) \cdot f_{norm}(c_j, P_k), & \text{if match of } P_k \in \mathcal{P} \text{ at } c_j, \end{cases} \tag{4.37}$$

where $0 \leq \alpha \leq 1$ is a parameter that determines the influence of previously calculated values to $F(c_j$. The default constant F_{def} – used if there is no pattern match for the very first clicknote – might be estimated from the input data itself or set to a default value. In our implementation we chose a default duration that results in a quarter note duration of approximately 600ms equivalent to a tempo of 100bpm. A quarter beat duration of 600ms represents the most sensitive beat duration for human perception ([Par94a, Fra82]). It is also close to the centre (log scaled) of the typical used range for performances of approximately 40bpm to 250bpm.

Now we can define the context distance d_{norm} for a pattern P starting at a clicknote $c_i \in C$ as

$$d_{norm}(c_i, P) := W_{Gauss}(F_{norm}(c_{i-1}) - F_{norm}(c_i, P), \sigma). \tag{4.38}$$

It is easy to see that the context distance is high if the relation between local score and performance timing changes. This relation is equivalent to a local tempo change.
When processing the clicknotes in ascending order, the variance σ of the Gaussian window function should

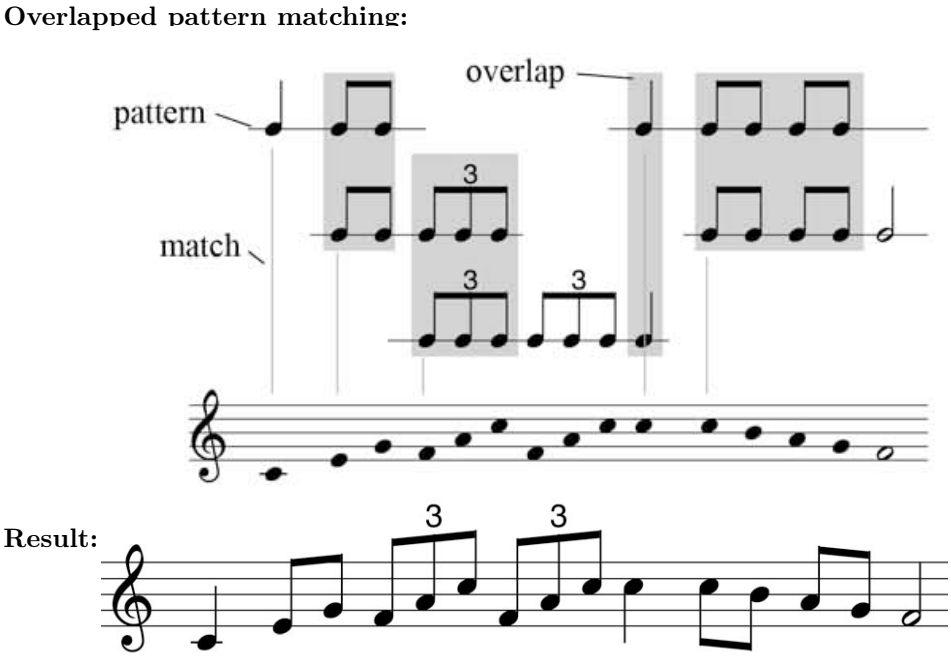


Figure 4.10: Example for overlapping alignment of patterns to performance notes.

be increased as long as no pattern has matched for any clicknote. During this state no local tempo has been confirmed respectively estimated by a pattern match and the significance of the context distance should therefore be decreased by increasing σ.

For the calculation of the structural distance d_{IOIr} between a pattern P with length $|P|$ and a set of clicknotes $\tilde{C} = \{c_j, \ldots, c_{j+|P|-1}\} \subseteq C$ there exist at least three different possibilities:

1. Calculate the relation between performance time and score time for a dedicated clicknote (*e.g.,* the first or the longest) and the corresponding pattern note, calculate the resulting score times (durations, onset times) for all clicknotes in $\tilde{C}$ and compare these information to the score timing information of the pattern notes. If assuming that there are no tempo fluctuations during the pattern this approach could work very well.

2. Direct comparison of the IOI ratios of pattern notes to the IOI ratios of the clicknotes.

3. Calculate for each clicknote c_{i+j} (with $j > 1$ a score IOI depending on the relation between performance time and score time of the previous notes p_{i+j-1}, c_{i+j-2}. The overall distance can then be calculated by comparing the resulting score durations.

The last item (3) works similar to item 2, but here the comparison of score durations (or IOIs) can be done with higher precision than comparing IOI ratios (item 1.). For a clicktrack, C, and a pattern, P, as defined above, this n-time normalised approach can be described in mathematical terms as

$$R := \left(r_1, \ldots, r_{|P|-1} \mid r_i = \frac{IOI(p_i)}{IOI(c_{j+i-1})}\right) \tag{4.39}$$

$$P' := \big(IOI(p_2), \ldots, IOI(p_{|P|})\big) \tag{4.40}$$

$$C^j := \big(r_1 \cdot IOI(c_{j+1}), \ldots, r_{|P|-1} \cdot IOI(c_{j+|P|-1})\big). \tag{4.41}$$

Similar to [ZP03] we use the angle φ between the vectors C' and P' as distance measure between these vectors:

$$\begin{aligned} \cos(\varphi) &= \frac{< C^j, P' >}{\|C^j\| \cdot \|P'\|} \\ d_{IOI}(C^j, P') &:= \arccos\left(\frac{< C^j, P' >}{\|C^j\| \cdot \|P'\|}\right). \end{aligned} \tag{4.42}$$

The arccos is used because for small angles $\arccos(\cos\varphi)$ shows a higher discrimination rate than $\cos\varphi$; the range of $d_{IOI}(C', P')$ is therefore the interval $[0, \pi]$. The first pattern note p_1 is not compared directly but has indirectly high influence to the pair p_2, c_{j+1}. Tempo fluctuations and inexact played notes have only minor influence to the distance $d_{IOI}(C', P')$. The behaviour of this distance could be described in words by: "what would be the IOI for clicknote c_{j+i} if the IOI for clicknote c_{j+i-1} would have been equal to pattern note p_{i-1}?". This means that for calculating the distance between c_{j+i} and p_{j+i}, the score duration of c_{j+i-1} is assumed to be the score duration of p_{i-1}.

The structural distance $d_{struct}(c_j, P)$ between a pattern P and a sequence of successive clicknotes starting at clicknote c_j can now be defined as

$$d_{struct}(c_j, P) := d_{IOI}(C^j, P'), \tag{4.43}$$

where C^j and P' are calculated as defined in Equation 4.41 and Equation 4.40.
Because the dimension of the P' and C^j is rather small for short pattern the selective behaviour (discrimination) of the structural distance is lower for short than for longer pattern. To overcome this issue the dimension can be increased by comparing also the IOI ratios of compound notes. In this case the sum of the duration of two successive pattern and clicknotes will be used as additional entries to C^j and P'. Compound notes created from more than two notes should be avoided, otherwise possible tempo fluctuations can have undesired negative influence to the IOI ratios. It should be noted that if a pattern P contains rests they will not be represented as explicit pattern notes $p_j \in P$ for the tempo detection. But they will be evaluated for calculating the correct IOI between two clicknotes c_j, c_{j+1}.

The total pattern matching distance between a clicknote $c \in C$ and a pattern P can now be defined as

$$d(c, P) = \alpha \cdot d_{norm}(c, P) + (1 - \alpha) \cdot d_{struct}(c, P), \tag{4.44}$$

where $0 \leq \alpha \leq 1$ is a parameter that determines the relation between structural distance and context distance.

Because during the selection of a best matching pattern for a clicknote c_j depends on the selections made for clicknotes $c_1, \ldots, c_{j-1}$ a dynamic programming (DP) approach (see Section A.1) would not be adequate for finding a global optimal solution. Because the distance function $d(c_j, P)$ also evaluates context information (induced tempo) resulting from accepted matches for earlier clicknotes c_{i-e}, it seems to be very hard to model d as a context independent cost function as required for DP.
Instead a powerful pattern classification mechanism, such as the *support vector machine (SVM)* approach ([Vap95, CST00]) seems to be more adequate. But because in our model during the tempo detection the patterns are compared to subsections of the clicktrack with an a priori unknown length, major adaptations of the SVM model would be required for using it here. It is also non-trivial to add new patterns (classes) to a SVM-based approach without a minimum number of existing instantiations (performances) for required training.

The HMM model proposed by Takeda *et al.* ([TNS03]) can be seen as related to our model. In both approaches score durations for a set of clicknotes are inferred by evaluating the already inferred score durations of earlier notes. Different from Takeda *et al.* who evaluate only the IOI ratio, our pattern distance function evaluates IOI ratio and score duration (resulting in a local tempo) simultaneously. Because of the ambiguity of the IOI ratio pattern Takeda's model requires an additional post-processing step where sudden tempo changes to integer multiples or integer fractions of the local tempo need to be resolved. Because our distance model evaluates the IOI ratio and the resulting score duration of notes in the pattern distance function the chance of an undesired switch in timing is reduced.
Takeda's evaluation for several performances of three different songs shows for the rhythmically rather simple Bach Fuga a result of $\approx 94.2\%$ correct estimated notes. For the more complex (because of tempo

fluctuations and a larger set of occurring score durations) performances his results are between $60\% - 78\%$ of correct notes.
Compared to the HMM approach of Takeda *et al.* our model of using explicit patterns and evaluating IOI ratio and duration might be more robust against unusual rhythms and easier to maintain by the user. We assume that without a pre-processing (filtering) of a merged clicktrack – which is not part of Takeda's model – the number of possible transitions becomes significantly large and will include a high number of ambiguities. The model of Takeda *et al.* includes a pre-processing for merging closely played onset times (*e.g.,* grace note followed by a non-grace note) to single time position but it does not filter any other (for tempo detection) insignificant notes before starting the actual tempo detection.

Matching Threshold

By rejecting patterns with a distance $d(P, c) \geq \tau$ (see outline at p. 86) false positive pattern matches should be avoided. If the threshold τ is chosen too high, too many correct pattern will be rejected. One strategy for avoiding this could be preferring patterns which have already accepted in the past and give a penalty to pattern not used in the local past. During the development of the current implementation it turned out that this concept is not correct. Preferring already used patterns against unused patterns caused a high number of false positive matches. Much better results could be obtained by keeping track of a *typical, average matching distance.*

It can be assumed that a performer does not change the accuracy of his playing drastically between close or successive notes c_i and c_{i+1}. If the performer has already played a sequence of notes that match to a pattern P with a distance d, than the next occurrence of this pattern in the performance should have a matching distance d' close to d. Using the previous assumption it can be expected that the matching quality for close clicknotes stays in a rather small range. Thus, a pattern should be rejected if its matching distance $d(P, c)$ is much higher than the typical, average matching distance d_{av} until this point. If a new pattern has been accepted as a match, the typical, average matching distance d_{av} should be updated with $d_{av} := (1 - \alpha)d_{av} + \alpha d(P_i, c_j)$ with $0 < \alpha < 1$. The parameter α controls the influence of a new matching distance to the already established average distance.

In addition to a global (for all pattern) average matching distance a typical matching distance for each pattern can be calculated. Depending on the complexity of a pattern or the skills of the player the playing accuracy might be different for different patterns. By using the global average matching distance d_{av} and the typical pattern matching distance d_{typ} the fixed matching threshold τ can be replaced by an adaptive threshold depending on the relation between $d(P, c)$ and d_{av}, d_{typ}. As result now the matching threshold in very precise performances (or even regions), with only small tempo fluctuations will be automatically lowered which avoids false positive matches. For performances with high tempo fluctuations it will automatically be raised to allow pattern matches with a higher distance to the performance.

After performing the pattern matching now in best case every clicknote $c \in C$ is in range of a matching pattern and the score duration information can be copied from the pattern data. In average case there will still exist sequence of clicknotes outside of any matched pattern range. In the following subsection it is described how score time information can be obtained for these clicknotes.

4.3.3 The Binclass Approach

For all regions of a performance/clicktrack where no matching pattern could be found by the previously described approach, a tempo detection (inferring score time information) using standard – non-pattern-based – techniques needs no be performed. For the current implementation an approach based on statistical analysis of the local past of a clicknote c has been developed. Different from the pattern-based approach described in the previous section, it cannot evaluate structural information (*e.g.,* after two eighth triplet durations must follow a third one) but it adapts automatically to history information, such as "a duration d should be preferred for clicknote c_i, because it was used very often for previous clicknotes $c_{i-k}, \ldots, c_{i-1}$".
The basic assumption behind this behaviour is that an human listener might prefer to use previously already used note durations for transcribing an ambiguous note duration. Hints to that behaviour can be found in a recent (unpublished) study by Zanette [Zan04] which shows that during listening to a

performances permanently expectations about the expected typical note durations are created. If these expectations are not fulfilled (at least to some level) by the composer (or performer/improviser), an human listener will have difficulties in understanding the composition or following the performance.

The general outline of the history-based approach can be described as:

1. Infer a reasonable start duration for the first clicknote. The resulting local tempo should be similar to the last known tempo estimated by the pattern processing.

2. Find for each following clicknote a score duration, where simple durations should be preferred over complex durations (*e.g.,* prefer 3/16 over 10/48) and the resulting tempo should be close to the known local tempo. Try to use score durations or score IOI ratios that are common and have been used in the local past of the current clicknote.

As shown in [TC02] there exists styles of music (*e.g.,* rock, pop) where the beat histogram of a performance shows significant peaks (see Figure 4.11), which could be used to estimate directly the beat/tactus level and the corresponding tempo by clustering the observed beat durations. In the general case and especially for classical music or jazz – which typically include tempo variations or non-integer IOI ratios – we must assume that it is not possible to extract the performed tempo directly by clustering the observed inter-onset intervals. Table 4.2 and Table 4.3 show the observed distribution of duration classes, *i.e.,* IOI classes, and IOI ratio classes for the exemplary performance of Chopin, Op. 6, *Mazurka 1* as described in the previous section.

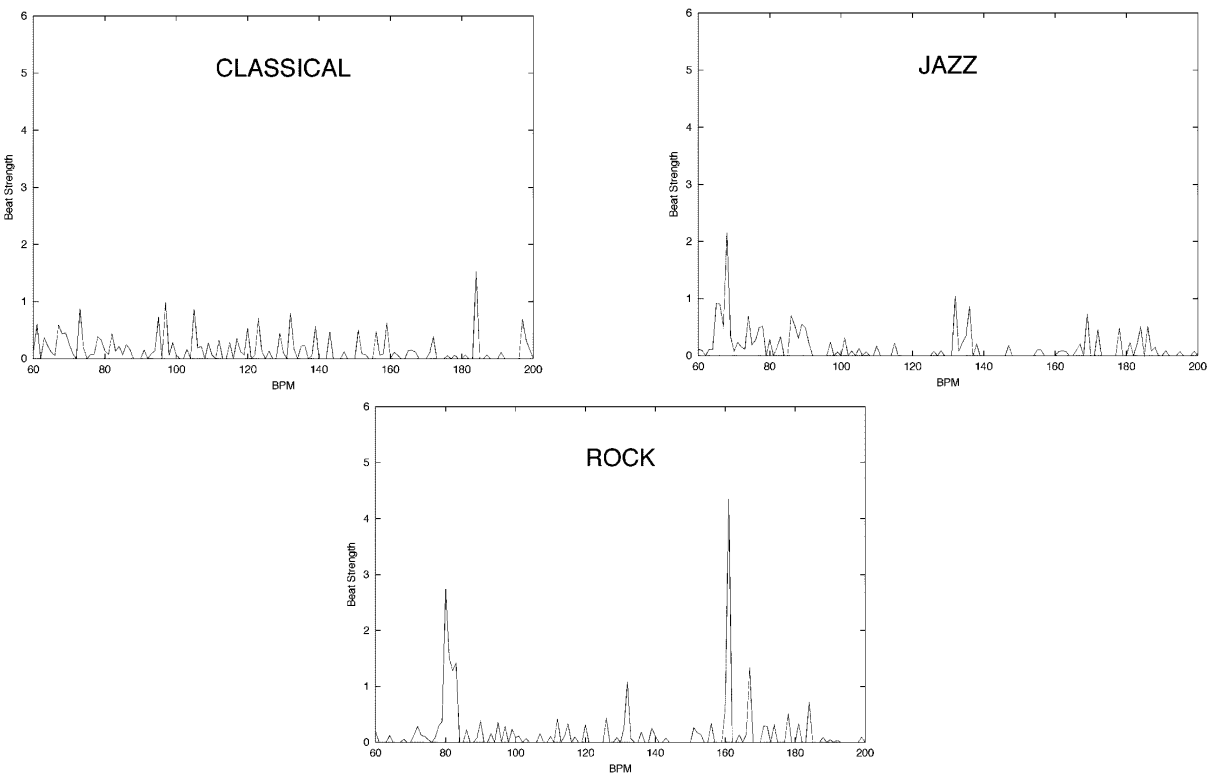


Figure 4.11: Beat histogram examples for different styles of music. (Copied from [TC02])

In the following first the mathematical model for the history-based selection strategy will be explained in detail and then the use of this model for the actual tempo detection will be presented. As already introduced we intend to obtain for a given score duration d a measure which reflects how often d was *used* (*i.e.,* selected as a score duration for a clicknote) in the local past/history of the current tempo detection

state. If, for example, a sequence consisting of $n + m$ notes would consist of n eighth notes followed by m sixteenth notes, then at position $n + m + 1$ the preference measure (or weight) for the eighth note class should be different from the measure calculated for a sequence consisting of of m sixteenth notes, followed by n eighth notes. If we would just count the number of usage for each duration class, we would

score	complete		filtered	
IOI	#	%	#	%
1/16	27	13.85	4	2.84
3/40	2	1.03	2	1.42
1/12	51	26.15	6	4.26
1/8	49	25.13	40	28.37
3/20	4	2.05	4	2.84
1/6	-	-	6	4.26
3/16	21	10.77	4	2.84
1/4	42	21.45	74	52.48
1/2	-	-	1	0.71
	196 notes		141 notes	
	7 classes		9 classes	

Table 4.2: Chopin, *Op. 6, Mazurka 1*, measures 1–36: distribution of score IOIs in complete and filtered clicktrack.

		complete		filtered	
IOIratio	ratio	#	%	#	%
-4	(1:4)	-	-	1	0.71
-3	(1:3)	37	18.97	7	5.00
-2	(1:2)	12	6.15	14	10.00
-1.5	(2:3)	-	-	2	1.43
$-1.\overline{33}$	(3:4)	-	-	3	2.14
1		88	45.13	84	60.00
$1.\overline{11}$	(10:9)	1	0.51	1	0.71
1.125	(9:8)	-	-	2	1.43
1.2	(6:5)	1	0.51	1	0.71
1.5	(3:2)	19	9.74	2	1.43
2		7	3.59	18	12.86
2.25	(9:4)	6	3.08	-	-
3		7	3.59	1	0.71
4		-	-	4	2.86
		195 values		140 values	
		9 classes		13 classes	

Table 4.3: Chopin, *Op. 6, Mazurka 1*, measures 1–36: distribution of score IOI ratios in complete and filtered clicktrack.

obtain a distribution about the used durations but this simple measure would not reflect the local past component. For the example above it would be in both cases only depend on the size of n and m but not on the order of occurrence.

In the following we propose a mathematical model based on a set G of classes or bins. Each class $g \in G$ represents a nominal value, v, given in arbitrary units (*e.g.,* units of seconds or score time units), where all values v must be given in the same units for the complete set G. In the context of tempo detection sets of score-durations and IOI ratios will be used. Each class $g \in G$ can be denoted as a vector $g = (v, w)$, where the nominal value, v, should be the duration or the IOI ratio corresponding to the class g, and, w a dynamically changing weight of class g. The vector elements can be retrieved and also set using the utility functions $v(g)$ for the nominal value and $w(g)$ for the weight of g. For our model we require that the sum of all weights of all classed in G is equal to one: $\sum_{i=1}^{|G|} w(g_i) = 1$ with $0 < w(g_i) \leq 1$. Optional a minimal weight $w_{\min}$ can be defined with with $0 < w_{\min} < \frac{1}{|G|}$ and $w_{\min} \leq w(g) \leq 1, \forall\, g \in G$.

We define an increase step-size Δ with $0 < \Delta \ll 1$ specifying the overall weight which should be moved, if a value x (a duration or an IOI ratio) is assigned to a class g. Δ must stay constant during an weight increase cycle, but might be depend on the number of existing classes. The class-weights can be initialised with $w(g_i) := \frac{1}{|G|}$ for $i = 1, 2, \ldots, |G|$ or with any arbitrary distribution that holds

$$\sum_{i=1}^{|G|} w(g_i) = 1 \tag{4.45}$$

resulting in a bias for each class g_i.

Dynamic progression of class-weights

If during the tempo detection process an (unquantised) score duration respective IOI ratio, d, is inferred to be equivalent to the nominal value $v(g)$ of a class g, then d should be *assigned* to the class g. In this case the weights of all classes must be changed (*i.e.,* re-distributed) to reflect, that the latest value have been assigned to class g. During this re-distribution phase the weights of all classes in G become decreased by a certain percentage (depending on Δ) and the weight of the selected class g is increased so that the sum of all weights stays equal to one after this phase.

If g_j was selected as closest or best class for an entry v, the weights of all classes will be recalculated in three steps:

1. Calculate a δ_i for each class $g_i \in G$:

$$\delta_i = \begin{cases} w(g_i) \cdot \Delta, & \text{if } w(g_i) \cdot \Delta \geq w_{\min}, \\ w(g_i) - w_{\min}, & \text{else,} \end{cases} \qquad \text{with } i = 1, 2, \ldots, |G| \tag{4.46}$$

   It is easy too see that if for all $g_i : w(g_i) \cdot \Delta \geq w_{\min}$ then $\sum_{i=1}^{|G|} \delta_i = \Delta$, otherwise $0 \leq \sum_{i=1}^{|G|} \delta_i < \Delta$.

2. Subtract δ_i from each class-weight:

$$w(g_i) := w(g_i) - \delta_i, \quad \text{with } i = 1, 2, \ldots, |G| \tag{4.47}$$

3. Add the sum of all δ_i to the weight of the selected class g_j:

$$w(g_j) := w(g_j) + \sum_{i=1}^{|G|} \delta_i \tag{4.48}$$

The amount of increase of a class-weight should be independent from the total number of classes, which might change during the tempo detection process. Therefore Δ might be adapted if a new class becomes instantiated during tempo detection. Figure 4.12 shows the progression of the weights of three classes during *assigning* 200 values (with $\Delta = 0.01, w_{\min} = 0$). During the assign operation 10 blocks of 20 equal values have been alternately added to class 1 and class 2; to class 3 no value has been assigned. The initial class-weights were unbiased, so each class started with an initial weight of $1/3$. At all positions $x = n \cdot 40$ class 1 and class 2 contain always an equal number of values, but always class 2 has a higher weight at this positions because it has then been more often chosen in the local past than class 1.

For classifying a value x to become assigned to a closest class $g \in G$ a distance measure between x and g is required. A simple distance measure ignoring any class-weight would be the absolute distance $|x - v(g)|$ between x and the nominal value of class g. This measure would also ignore all history information given implicit by the class-weight $w(g)$. To evaluate this information we define the distance $d(x, g)$ between a a value x and class g as

$$d(x, g) = 1 - W_{Gauss}(v(g), x, \sigma(g)), \tag{4.49}$$

with $\sigma(g) = f(w(g), \sigma_G)$, where σ_G should denote a global variance parameter. The class variance $\sigma(g)$ should be proportional to $\sigma_G \cdot w(g) \cdot |G|$, so that the variance $\sigma(g)$ is independent from the number of existing classes $|G|$, because $w(g) \propto 1/|G|$ (as required by Equation 4.45). By the multiplication with $|G|$ it can be ensured that for all $G : w(g_i) = \frac{1}{|G|} : \sigma(g_i) = \text{const}$. The range of $\sigma(g)$ should be limited to avoid very narrow or very wide window functions: $\sigma_{min} \leq \sigma(g) \leq \sigma_{max}$. In the current implementation of `midi2gmn` the calculation of $\sigma(g)$ is implemented as

$$\sigma(g) = \sigma_G \cdot \sqrt{|G| \cdot w(g)}, \quad \text{with } \sigma_G = 0.3,\ \sigma_{\min} = 0.01,\ \sigma_{\max} = 0.7. \tag{4.50}$$

For a given set G and an observed duration (or IOI ratio) x, we define the best fitting or closest class $g_{best}(x, G)$ as

$$g_{best}(x, G) = \arg\min_{g \in G} \{d(x, g)\} \tag{4.51}$$

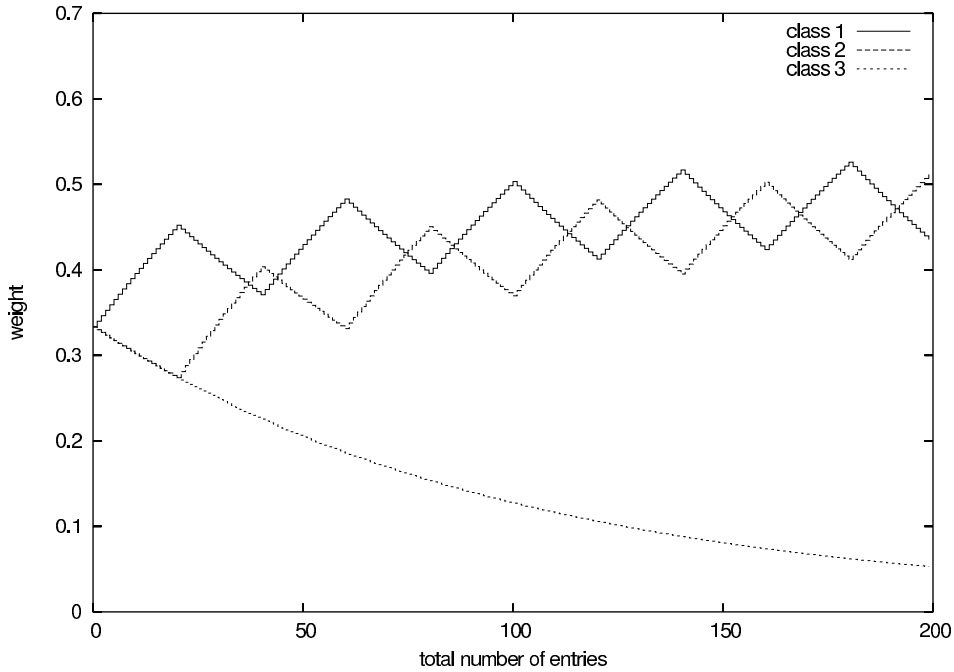


Figure 4.12: Progression of classes weights depending on the number and order of entries that have been assigned to a class.

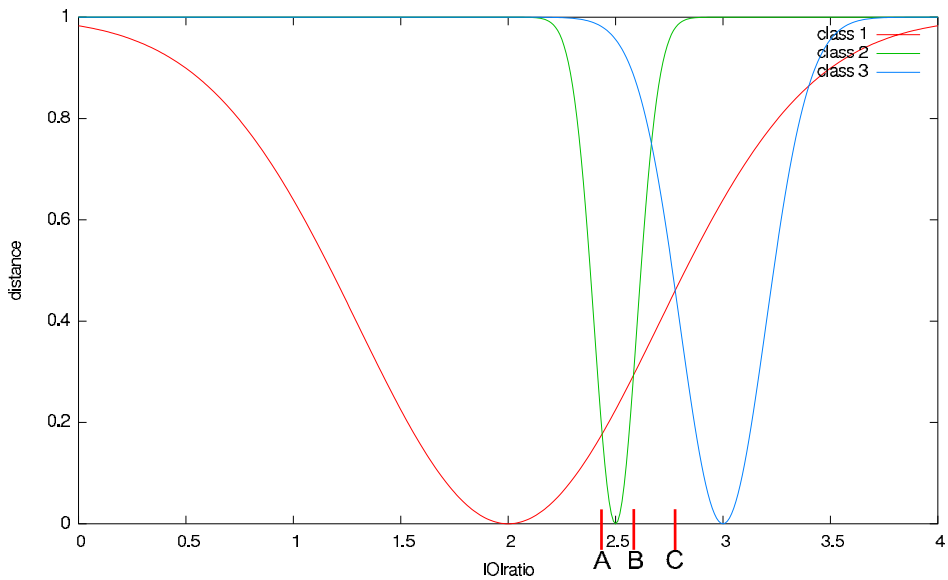


Figure 4.13: Distance between observed IOI x and a set of three IOI-classes G.

As mentioned before the set G can represent a set of duration classes or a set of IOI ratio classes. For the IOI ratio class list x and the nominal values $v(g)$ will be a normalised IOI ratio (see Equation A.18), for the duration class list $\ln(duration)$ will be used as nominal value for a class g. Figure 4.13 shows the distances respectively attractions indicated by $d(x, g_i)$ for three IOI ratio classes ($v_i = 2, 2.5, 3$) with different weights ($w(g_1) > w(g_2) > w(g_3)$). Class 1 (IOIratio = 2) has a high attraction, class 2 (IOIratio = 2.5) a very low attraction and class 3 (IOIratio = 3) a medium attraction. Estimating a minimum confidence distance of 0.8, all values x with $1 < x < A$ (A = first point of intersection between class 1 and class 2) will be assigned to class 1, all values between A and B (second point of intersection between class 1 and class 2) will be assigned to class 2, and all values $> C$ (points of intersection between class 2 and class 3) will be assigned to class 3.
The most interesting case is the range between B and C, the values are closer to the nominal values of class 2 and class 3 than to the nominal value of class 1, but because of the high attraction – which results in a small distance – these values will be assigned to class 1! Values with a distance to all classes higher than a certain confidence level (*e.g.,* 0.8), might be not be assigned to any class and must be treated separately (*e.g.,* skip x or creation of a new class). In the current implementation in this case the user will be asked to enter a correct score duration for the respective duration class.

The tempogram representation proposed by Cemgil *et al.* (see 74) can be seen as somehow related to the here described binclass approach. Both representations give a distance respectively probability measure between an observed duration, IOI, or IOI ratio and a set of score time durations. Different from the tempogram representation the distances given by the binclass histograms change dynamically during processing a performance, depending on the previously made decisions.

Estimating Score Time Information

The previously introduced binclass approach should now be used to infer score time information for observed IOI and IOI ratios of the clicktrack C created from given performance data. In the following we assume that a binclass list G_d for the local history of selected score durations, a binclass list G_{IOIr} for the local history of selected score IOI ratios, and optionally an IOI ratio calculated previously by the pattern-based tempo detection are given, We also assume that for the clicknotes $c_1, \ldots, c_{i-1}$ already score durations have been estimated by pattern matching or by using the binclass approach. Now we can calculate a potential score duration for clicknote c_i in two different ways:

- $d_{score} := v(g_d)$, with $g_d := g_{best}(d', G_d)$ and $d' := dur(duration_{score}(c_{i-1}), IOIratio_{perf}(c_i))$
- $\tilde{d}_{score} := dur(duration_{score}(c_{i-1}), I_s)$, with $I_s := v(g_{IOIr})$ and $g_{IOIr} := g_{best}(IOIratio_{perf}(c_i), G_{IOIr})$

The function $dur(d, r)$ returns a third score duration calculated from a score duration, d, of a predecessor note and an IOI ratio, r, (given according Equation A.16) of the current note:

$$dur(d, r) = \begin{cases} d \cdot r, & \text{if } r \geq 1 \\ -d/r, & \text{if } r < -1 \end{cases} \tag{4.52}$$

If now d is equal to $\tilde{d}$ we set this score duration as $duration_{score}(c_i)$ and proceed with clicknote c_{i+1} until we reach the end of the clicktrack or a clicknote which has been successfully processed by the pattern matching approach. In the case that d is not equal to $\tilde{d}$ we evaluate several measures which indicate which score duration should be preferred:

- the distance measures $d(d', g_d)$ and $d(I_s, g_{IOIr})$, where $d(x, g)$ should be defined according Equation 4.49.
- the weights $w(g_d)$ and $w(g_{IOIr})$

We select that score duration which results in a small distance to the closest element of the corresponding binclass set and where the corresponding closest element has a higher weight.

In an earlier version of our system we tested a model where – similar to the dynamic programming in the context of BLAST (see Section 3.3.1) – the statistical approach started from non-ambiguous clicknotes at arbitrary positions in C and worked then in both directions. As start positions for examples sub sequences of clicknotes with an IOI ratio close to one or close to small integers have been selected. Unfortunately this model showed worse results than the one directional, linear approach above. We could identify two reasons for the worse results:

- By starting at arbitrary positions in the piece no local context is available.
- By processing in the backward direction the benefits of the binclass approach cannot be used.
- Connecting and shifting the processed sub regions results in a high number of errors and additional ambiguities.

4.3.4 Error Detection

By analysing musical scores it can be investigated that there exist constraints for the duration d of a note or rest and its time position in the score. For example, quavers can be placed on every time position $t = \frac{n}{8}$ but it is very unusual that a quaver is placed on a ternary position $t = \frac{2n+1}{12}$. For triplet quavers (duration $= \frac{1}{12}$) all time positions $t = \frac{n}{12}$ are possible onset times, but time positions, such as $t = \frac{2n+1}{8}$ are usually avoided in standard music notation.

For a given duration d placed at time position t, a fitness function $f_{durpos}(d,t)$ should return results as shown Table 4.4. Syncopation scenarios, such as $d = \frac{1}{4}$ and $t = \frac{2n+1}{8}$ should give a result $0 < f_{durpos}(d,t) < 1$. In the following we assume that the duration d and the time position t are given as normalised fractions:

$$d = \frac{c}{a} \text{ and } t = \frac{d}{b}, \text{ with } a = \prod_{i=1}^{s} p_i^{k_i} \text{ and } b = \prod_{i=1}^{s} \hat{p}_i^{\hat{k}_i}, \qquad (4.53)$$

where $p_i, \hat{p}_i$ are prime numbers and $p_i > p_{i+1}$ and $\hat{p}_i > \hat{p}_{i+1}$. Usually the set of prime number factors needed in the context of score notation is very small. It can be assumed that for human playable scores $p_s \leq 13$. Higher level tuplets – which would result in large p_s – would be very hard to play and also very hard to infer correctly by the listeners.

duration	time position	f_{durpos}
$\frac{1}{12}$	$\frac{n}{4}$	1
$\frac{1}{12}$	$\frac{3n+1}{12}$	1
$\frac{1}{12}$	$\frac{2n+1}{8}$	0
$\frac{1}{8}$	$\frac{n}{8}$	1
$\frac{3}{8}$	$\frac{n}{8}$	1
$\frac{3}{8}$	$\frac{3n+1}{12}$	0

Table 4.4: Desired output for function f_{durpos} for exemplary combinations of duration and time position.

Using the described encoding for a, b the desired function f_{durpos} can be defined as:

$$f_{durpos} = \begin{cases} 1, & \text{if } a < b \wedge b \bmod p_1^{k_1} = 0 \\ 1, & \text{if } a \geq b \wedge p_1 = \hat{p}_1 \wedge k_1 \geq \hat{k}_1 \\ k_1/\hat{k}_1, & \text{if } a \geq b \wedge p_1 = \hat{p}_1 \wedge k_1 < \hat{k}_1 \\ 0, & \text{otherwise} \end{cases} \qquad (4.54)$$

The duration-time-position distance function f_{durpos} can be used for error detection during tempo detection, error detection during quantisation, and inferring an upbeat measure, if used as a minimisation function (see Section 6.1.4).

Unfortunately an error detected at time position t by $f_{durpos}(d,t) = 0$ might be caused by a wrong inferred duration d' at any arbitrary time position $t' < t$ with no error indication ($f_{durpos}(d,t') = 1$). It is also not known if the duration d is wrong itself or if the time position t is wrong because of a wrong duration of an earlier note. In Figure 4.14 an error is indicated by $f_{durpos} = 0$ for the first note with pitch d' – denoting a 'd' in the first octave – but for avoiding this error there exist several correct solutions. If the piece was played with tempo fluctuations and expressive timings all correct/possible solutions – there might exist an arbitrary number of possible solutions – will have similar distance values and probabilities during the tempo detection and can therefore not be discriminated. In some cases the

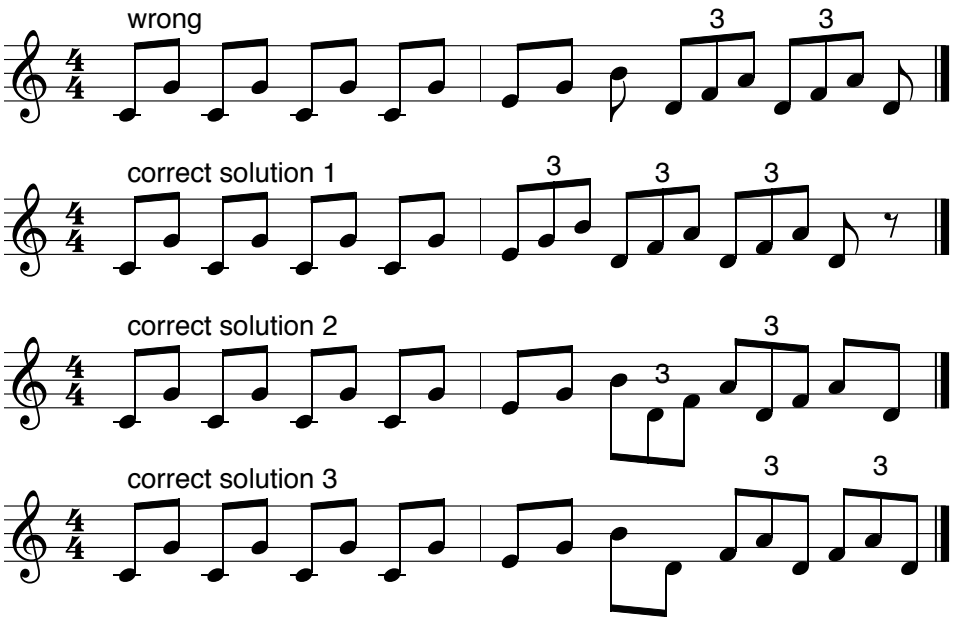


Figure 4.14: Four possible correct solutions (of an arbitrary number of other possible solutions) for a detected error at the first d'.

intended solution might be identified by analysing time positions of other voices playing at the same time but in worst case – and average case – it cannot be assumed that there exists such additional information.

In our current implementation the function f_{durpos} is used for checking the score duration inferred by the tempo detection module. If an error is indicated by the error function and it cannot be solved automatically, the user will be asked to enter a correct score position for the note where the error had occurred or for any earlier note which might have caused the error. The function is also used during the quantisation where in the case of ambiguous situations invalid alternatives can be avoided.

4.3.5 Estimating The Recording Type And Player Level

A statistical analysis of the typical deviations between mechanical timing and performed timing in performance files showed that these deviations follow a normal distribution (see Figure A.6, Figure A.7, and Figure A.8) and that performance files can be categorised by the variance of their deviations. By analysing the performed IOIs and IOI ratios of performance notes, the type of input data (*e.g.,* live recorded, exported from notation software) as well as the skill (level) of the player (*e.g.,* pro, intermediate, beginner) can be estimated.
In general we can assume that every composition contains a number of successive notes with equal, double or half durations. By calculating and analysing the variance of the performed IOI ratios which have been identified to belong to one of these three basic classes the type of input data can be estimated. If the variance is zero – the performed ratios match exactly the integer ratios – we can assume that the file is a mechanical performance; even a professional player would not be able to play with such an exactness. If the variance is very small the file was probably played by an advanced player, if the variance is very high it was played with higher probability by a beginner or it includes a large number of tempo fluctuations. Another indication for mechanical performances (machine played files) is the relation between the performed IOI and the performed duration of each note. If this relation is constant we can also assume that the file is a mechanical performance and not a live recording.

The relations and the variance can be calculated before performing tempo detection and quantisation using the previously described binclass approach. Here we define a set of three IOI ratio binclasses

$G = \{g_e, g_h, g_d\}$, with nominal values $v(g_e) = 0, v(g_h) = -1, v(g_d) = 1$ corresponding to the normlised IOI ratios 1 (equal), -2 (half 1:2), and 2 (double 2:1). All observed IOI ratios of a performance which are very close to one of these classes can then be used for calculating the mean and the variance of the performed IOI ratio for the respective class. From the IOI ratio variance an accuracy measure for onset times and durations can be derived which then can be used for a limitation of search windows for onset time and duration quantisation (see Section 5.2) as well as for estimating some global parameters, such as the σ_G parameter for the binclass approach (see Equation 4.49).

The relation between accuracy and a smallest detectable note duration (delta grid position) can be somehow compared with the Nyquist Theorem for sampled audio signals, which says that for capturing a highest frequency f correctly, the sampling frequency g must be equal or higher than $2f$. For the here described tempo detection and quantisation issue this means that it is not possible to infer very small note durations (or IOIs) correctly, if the performance accuracy is low. Only for a high performance accuracy we can infer very small note durations correctly, otherwise we must assume that small deviations are the result of inexact playing.

4.4 Results, Evaluation

For the evaluation of any tempo detection or beat tracking approach the inferred beat/clicknote time positions must be compared to their positions in the original (correct) score. If for each inferred beat (clicknote) the correct score position is known – this mapping usually requires time consuming manual work – an error metric which gives respect to the deviation between inferred duration and position and the correct score duration and position might be calculated. It depends on the focus of the actual approach how this metric should be calculated. For beat tracking approaches the error measure should increase, if certain time positions have not been recognised as beats. For an approach focussing primarily on transcription, such missing beats might not affect the output quality, as long as the score distance between two inferred beats is correct. For this type of approach the error measure should give only minor weight to missed beats. It is also an open question how a duration error for a single clicknote – which shifts the score onset times of all following clicknotes – should influence the error measure. We propose that an inferred score where all onset times are shifted by a constant offset is much closer to the corresponding correct score than a transcribed score where, for example, 50% of the clicknotes have a correct onset time position but the other 50% have an incorrect onset time.
When evaluating the correctness of transcriptions of a performance in addition to the comparison of the original score and the transcribed score, the transcription should also be compared to the performance itself. If parts of the performance were played in a way, where even an human transcriber would have chosen a transcription different from the original score (*e.g.,* a quaver followed by two semi-quavers instead a group of three triplet quavers) there is no obvious reason why an algorithmic transcription system should infer the original rhythm. Therefore this types of deviations should not be counted as errors during an evaluation.
The following evaluation includes the absolute number of duration errors, the percentage of wrong clicknote durations in relation to the number of clicknotes and detailed descriptions of the position, the reason, and the impact of the errors to other parts of the score. Our approach allows interactions between the system and the user, which can be switched off for batch mode processing. The error rates shown in Table 4.5 and Table 4.6 have been obtained in batch mode. In the following detailed evaluation of test files we will report cases where the system automatically has detected errors which then could have been removed by user interactions. The current implementation includes only a command line interface for user interaction. During tempo detection the systems reports the position (in units of MIDI ticks) of a detected error and asks the user to enter the correct duration (as a score duration) for any arbitrary clicknote in the already processed section of the performance. With a graphical user interface (see Section 6.7) (which is not in scope of this thesis) the correction of these errors could be done in a much more intuitive way.

Another general issue in the context of the evaluation of any type of transcription approach is the limited number of available live performed input files and also the lack of a standard test library. There is a large number of MIDI files available on the world wide web, but only very few are live performed files. Most of these files are mechanical performances and violate the copyright laws. We tried to create some

live performed MIDI input files (required for evaluation of tempo detection) by using audio to MIDI software applications. Even for piano music (Bach Inventions, from compact disks) the available tools created only low quality MIDI files containing many wrong, as well as additional and missing notes. Some existing MIDI file sets (used, for example, in [Cam00b]) underlie copyright restrictions and are therefore not publicly available. Fortunately the distribution of the Melisma Analyser (see Section 1.3) contains some live performed MIDI files and also [TNS03] includes an evaluation of live performed MIDI files. With the friendly help of Haruto Takeda it was possible to get a copy of these files for testing. In [CKDH01] Cemgil *et al.* evaluated 100 different piano performances of the Beatles songs *Michelle* and *Yesterday* which are publicly available (as live recorded MIDI files) at `http://www.nici.kun.nl/mmm/archives/index.php?archive=beatles`. Unfortunately in these files the sustain pedal note durations were converted into explicit note durations representing the (piano) string release phases. As shown in Section 1.4.3 the release time of a piano string is different from the press-down time of the corresponding piano key if the sustain pedal is used. For longer note durations both times are different from the perceived note duration because of the intensity decay of the string. As also shown in Section 1.4.3 the resulting (incorrect) large number of equally sounding notes causes major problems to the voice separation module because notes become merged to chords and actually different onset times will be moved to a single onset time of the chord. These wrong and primarily the missing onset times then decrease the output quality of the tempo detection module.
For an approach which evaluates only the onset time data omitting the previous separation into distinct voices, this files can be used without any drawbacks, To allow an evaluation of these files by our system, we decided to corrected the note durations for one of the *Yesterday* performances manually and additionally recorded an own performances of *Michelle* and *Yesterday*.

Table 4.5 shows the results obtained for processing different performance files of a 'son-clave' pattern (see also Figure 4.2 and Section A.6). In [CK02] this pattern was proposed as hard input data for tempo tracking approaches because of its highly syncopated rhythmical structure. For the evaluation we created a mechanical performance, a live recording of an human player who tried to stay in synch with a metronome, a live recording of a player who tried to play a constant tempo without any additional metronome information, and a live recording of a player who intentionally played significant ritardandi and accelerandi. For the test, the tempo detection pattern database contained eleven patterns (see

file type	# c-notes	# c-note errors	# f-p matches	# p matches
mechanical	106	0	0	20
live to click	55	0	0	8
live, constant tempo	60	0	0	12
live, changing tempo	92	1	0	19

c-notes - number of clicknotes in file
c-note errors - number of clicknotes with wrong score duration
f-p matches - number of false positive pattern matches
p matches - total number of pattern matches

Table 4.5: Evaluation for hybrid tempo detection with different performance types of 'son-clave' as input.

Figure A.3), including seven standard rhythm patterns and four versions of the son-clave pattern. To allow an overlapped matching of the clave pattern it was defined in the 3:2 and the 2:3 version (see Figure 4.2 for a detailed description) each in double time (alla breve) and a $\frac{4}{4}$ version with halved note durations. The set of IOI ratios was defined as shown in Figure A.6.
Using this input data the hybrid tempo detection module showed nearly 100% correct results. Only for the file including the tempo fluctuations an incorrect 1/6 duration has been inferred for the last clicknote. See also Section A.6 for more details on the inferred tempo profiles. It should be noted that without any additional synchronisation information the highly syncopated clave pattern is hard to track even for human listeners. In 'real' compositions this pattern usually occurs only as an additional rhythmical layer over an uniform beat (typically a 16th or eighth beat) created by several types of percussion instruments.

The transcription[10] of Bach *Minuet in G*, mm. 1-16[11] (the performance MIDI file is part of the Melisma

[10] Please see Section 5.4 for the graphical scores of the here discussed songs.
[11] From Notenbüchlein für Anna Magdalena Bach.

Analyser distribution available at `http://www.link.cs.cmu.edu/music-analysis`) contained only two errors: a wrong duration (dotted crotchet instead of a crotchet) for the very first clicknote and a wrong duration (dotted crotched instead of a minim) for the second but last clicknote. The error for the first click note resulted from an initial rubato in the performance. Instead if a mechanical IOI ratio of -2 (2:1) the performance included a ratio of -2.612 which has been estimated to become -3 (3:1) by the tempo detection module. Because of the series of three successive quarter notes starting at $(2n+1)/8$ positions – shifted by the wrong duration of the first note – our system prompted one time for user input. By correcting the duration of the first clicknote the shift errors in the output score could be removed.

The transcribed output of the *Yesterday* (Cemgil *et al.* version) file started with 23 correct inferred clicknotes but then the system switched to a double time version of the rhythm. Rhythmically the double time version (double note durations and doubled tempo) would be also correct. Only if evaluating the chord changes and melody contour with a musicological view the transcription in slow tempo with smaller note durations (*e.g.,* eighth instead quarters) becomes mandatory. Beside this switch of the beat level the output score included six *real* errors, which have been caused by imprecise clicknotes, played very close to allowed IOI ratios and durations. In the interactive mode our system recognised these errors and prompted for manual corrections by the user. By using our own recording of *Yesterday* as input we obtained a completely correct score without giving additional user entries.

Because our voice separation showed major problems with the unusual note durations recordings of *Michelle* used by Cemgil *et al.* ([CKDH01]) we evaluated only the transcription of our own recording. In [CKDH01] these performance files have only been used only for training the model, the there reported evaluation was then done with the *Yesterday* files. For the *Michelle* performance our system inferred the durations of the clicknotes correctly, except few errors in the measures 15–17. Caused by a false positive pattern match the first group of quarter triplets here could not be recognised correctly. This resulted in a switch to double timing and errors for the clicknotes in measure 16. 17. At measure 18 the correct rhythm pattern has been matched and the inferred tempo switched back to the correct tempo. In the interactive mode our system recognised automatically an error (caused by double timing and wrong durations in measure 16) in measure 17 and asked for manual corrections by the user.

The Beethoven *Sonata Nr. 20*, Op. 49, 2, was the largest file we selected for the evaluation. The selected performance is part of the test set used by Takeda *et al.* in [TNS03] for the evaluation of their system. The song includes several changes from binary to ternary rhythm which should be hard to track for tempo detection approaches (see also Section 1.5). The merged clicktrack contained 1096 clicknotes. As expected occurred the major number of the 82 observed errors at positions where the rhythm changed between binary and ternary subdivisions. The pattern processing module recognised these changes, but in some cases only with a delay of a few notes and in one case with a delay of two measures. Some errors were caused by the automatic error correction of our system. At two positions it recognised shifting time position errors (*e.g.,* a crotchet starting at a non-binary time position) and decided that it is possible to correct these errors automatically (instead of asking the user) by ignoring the actual correct clicknote duration, as specified by a previously matched pattern. The very low rate of false positive pattern matches compared to the high number of correct pattern matches (the used pattern database included 21 pattern) gives some indication that the pattern distance function of our approach (see Section 4.3.2) works correctly.
Without the using a pattern database the results for this example and its rhythmical changes were significantly worse (error rate $> 40\%$). After in this case the triplets at the beginning section have been inferred as sixteenth notes, the history-based tempo detection also preferred to use sixteenth notes for other regions of the piece and dropped the class-weight for the eighth triplet class. As result most of the triplets have then been transcribed as sixteenth notes.
In [TNS03] for the evaluation of a set of ten different performances (two recordings of five performers) of this piece an average rate of 60.7% (for the HMM) respectively 78.5% (after post-processing) correctly recognised notes has been reported. Because here the post-processing includes a detection and correction of regions where the system has skipped to integer multiples (or divisors) of the actual tempo we must compare the (uncorrected) error rate of 39.3% to the results obtained with our system (7.5%). Unfortunately the authors did not report the positions and type of the observed errors, so it is not possible to compare them in more detail.

The performance file of Chopin's *Op. 6, Mazurka 1* (measure 1-40) had been played in an expressive style containing many accelerandi and ritardandi. As desired the clicktrack filtering removed most of the

file	notes	c-n	c-note errors		IOI ratio errors	f-p matches		pm
Minuet in G	101	70	2/1	(2.9%/1.4%)	**1(1.4%)**	0		13
Chopin *Op. 67, Mazurka 2*	109	63	6	(7.9%)	**1(1.6%)**	0		23
Chopin *Op. 6, Mazurka 1*	327	179	28	(16.5%)	**10(5.6%)**	1	5.6%	18
Yesterday (Cemgil)	198	138	6+23	(4.3%/16.6%)	**4(2.9%)**	1	3.8%	26
Yesterday	254	142	0		0	0		27
Michelle	207	116	14	(12.1%)	**3(2.6%)**	1	2.5%	40
Beethoven *Sonata*	1499	1096	82	(7.5%)	**18(1.6%)**	10	3.2%	312

notes	number of performance notes in file
c-n	number of clicknotes in file
c-note errors	number of clicknotes with wrong score duration
IOI ratio errors	clicknotes with wrong IOI ratio
f-p matches	number of false positive pattern matches
pm	total number of pattern matches

Table 4.6: Tempo detection errors for performance files.

sixteenth notes and also many of the triplet groups so that the resulting clicktrack contained mostly a regular grid consisting of quarter and eighth notes. The transcribed score contained 28 clicknotes (out of 170) with wrong durations, omitting aa correction through user interaction. These errors were distributed to only six small regions of the score. Except of two errors, caused by a false positive pattern match, all errors occurred at positions where no rhythm pattern could be matched. Because measure 24 (end of B part, 7 errors) contains a strong ritardando in combination with an unusual rhythm of a quarter quintuplet it has been inferred (incorrectly) as a group of quarter and half notes.
In the interactive mode the system recognised three shift errors and asked the user for corrections.

The durations of the (merged) clicknotes in Chopin's *Op. 67 Mazurka 2* mm. 1-16,[12] were inferred correctly until the sixth last clicknote. This note was performed with an IOI ratio of -1,506 ($\approx 2:3$) instead of 1. The pattern similarity analysis actually inferred the correct rhythm pattern as the best matching pattern, but because its similarity to the performed rhythm was significantly higher than the average pattern similarity for the other parts of the performance, it has not been selected as a match. Instead the IOI ratio was inferred (by single processing) to become a ratio of -1.33 resulting in a score duration of a dotted eighth note for the corresponding clicknote. The IOI ratios of the remaining five notes have been inferred correctly but because of the previous duration error the resulting durations are also wrong. Because the error was close to the end of the input file the algorithm did not detect a possible error automatically at suspicious onset time positions of non-existant following notes.

The output of our tempo detection module is always passed to the quantisation module before the conversion into GUIDO file syntax. The quantisation module estimates the score time positions for all notes which have were not included in the merged clicktrack (filtered) and it also infers score positions for the note offset points which are not evaluated during the tempo detection. Small errors of the tempo detection module might also be corrected during quantisation (*e.g.,* small shifts).

Because our system includes interactive features which are not part of the standard approaches known from literature, it is difficult to compare our results directly to the results obtained with other systems. The in our system implemented *guided user interaction* functionality tells the user about positions with potential errors in the transcription. With a graphical user interface (that is not in the scope of this thesis) it would be possible to correct these errors in an intuitive way. Errors where a group of notes has been transcribed with correct IOI ratios but wrong durations because of an incorrect IOI ratio for the first note of the group (*e.g.,* c/8 e/12 g/12 c/12 instead of c/8 e/8 g/8 c/8), could be corrected with only two mouse clicks (select first note and select correct duration). The column 'IOI ratio errors' in Table 4.6 shows the number of incorrect IOI ratios (typically smaller than the number of incorrect durations). The

[12] This performance MIDI file is part of the Melisma distribution which can be downloaded at `http://www.link.cs.cmu.edu/music-analysis`.

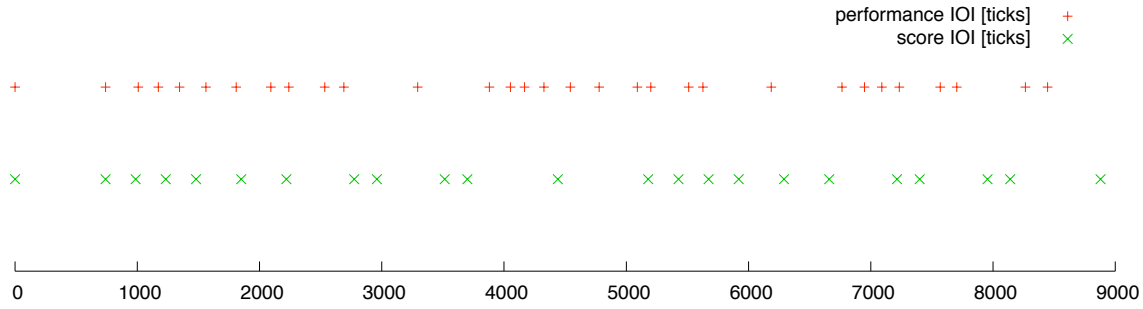


Figure 4.15: Score and performance positions (in MIDI ticks, with 960 ticks/sec) of onset times at the beginning of Chopin, *Op. 6, Mazurka 1.*

current implementation of our system allows to fix these errors on the command line level for reducing the error rates (see detailed description for specific files). To our knowledge there exist no other approach which automatically guides the user directly to an eventual transcription error. This interactivity feature and the way how our system allows the definition and use of rhythmical patterns, omitting any training phase can be seen as two unqiue improvements to the current state in the domain of tempo detection.

Because the different approaches discussed in Section 4.2 are focussing on different issues of tempo detection and beat induction, the authors are using different types of error metrics. As shown in Equation 4.21 Cemgil *et al.* evaluate the in [CKDH01] proposed approach with an error metric that depends on the number of inferred beats and their absolute time position. Different from their algorithm, our approach infers score positions for all notes of a filtered clicktrack and not the absolute time position of beats. The output quality of our approach depends only on the correct positions of the inferred clicknotes, skipped clicknotes (equivalent to missing beats) do not change the quality of our output. Therefore we cannot use an error metric as used by Cemgil *et al.* (see Equation 4.21). For a clicktrack, C, An adaptation of their metric to our system could be based on the difference between correct score IOI ratio, $\overline{IOIratio}$, and inferred score IOI ratio, $IOIratio$, for each clicknote, c:

$$\rho(C) = \frac{\sum_{i=1}^{|C|} W_{Gauss}(IOIratio(c_i) - \overline{IOIratio}(c_i), \sigma)}{|C|} \times 100 \tag{4.55}$$

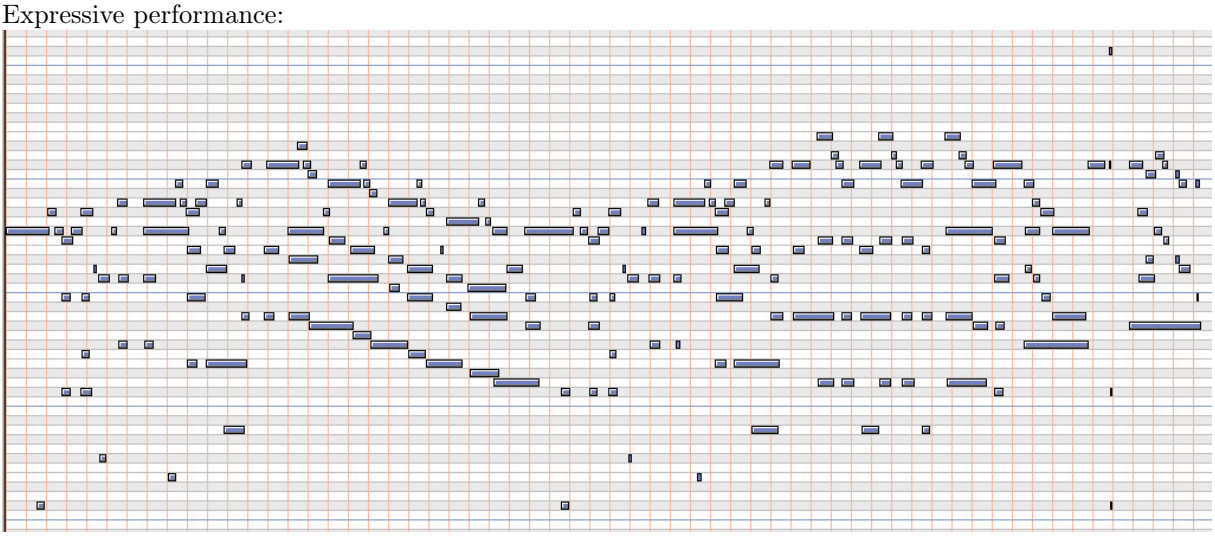
Depending on the value of σ this would result in very optimistic error rates for our system. They would be equal (if $\sigma \rightarrow 0$) or lower than the percental IOI ratio errors given in Table 4.6 (*e.g.,* 2.9% for *Yesterday*). Cemgil *et al.* report in [CKDH01] an average error rate of 8% (calculated with Equation 4.21) for several performances of this piece). We would prefer a comparison of the output quality of different systems based on an evaluation of the number and also on the type of errors that occur when processing files of a standard test library as proposed in Section 1.5. Beside the total number of errors it should be evaluated how many user interactions would be required to remove these errors.

The research during this work showed again that a tempo detection based only on rhythmic features and significance measures of clicknotes has certain limits. As already mentioned in [Kil96] even human, educated listeners are not able to infer a correct score or even a tactus beat from very expressive played performances, if the pitch and harmonical information has been removed or (distorted) from the performance data. Own tests with audio files created from a performance of Chopin's *Op. 6, Mazurka 1* with original rhythm but randomised pitch information showed that in this case it is nearly impossible for musicians or educated listeners to infer correctly the original rhythm and meter. Figure 4.15 shows the performance time positions of the mechanical clicknotes and the positions of the observed clicknotes in an expressive performance of the beginning of Chopin's *Op. 6, Mazurka 1.* It should be easy to see that without knowing in advance that the two data sets represent the same song, on this one dimensional graphical level, without any pitch information, the two data sets can hardly be matched. In a two-dimensional graphical piano roll display (including correct pitch information) it becomes a trivial task to infer the similarity between the mechanical and expressive data (see Figure 4.16).

We assume that there exist at least two categories of compositions: compositions where the beat and the

metrical level of notes is induced by rhythmical features and compositions where the beat information is induced by melody and harmonic (chord) progressions. In compositions, such as for example, *Michelle* or Bach Inventions the significance of the beat positions is induced rather by melodical or harmonical features than by rhythmical features. In these types of compositions the strength (or weight) of clicknotes in a merged clicktrack will therefore be very homogeneous showing no significant peaks which indicate certain strong metrical positions. Other compositions, such as the discussed Beethoven Sonata will have very significant clicknote positions because the number of notes played at equal time changes significantly and even the merged clicktrack contains notes with major differences in their duration.

Expressive performance:



Mechanical performance:

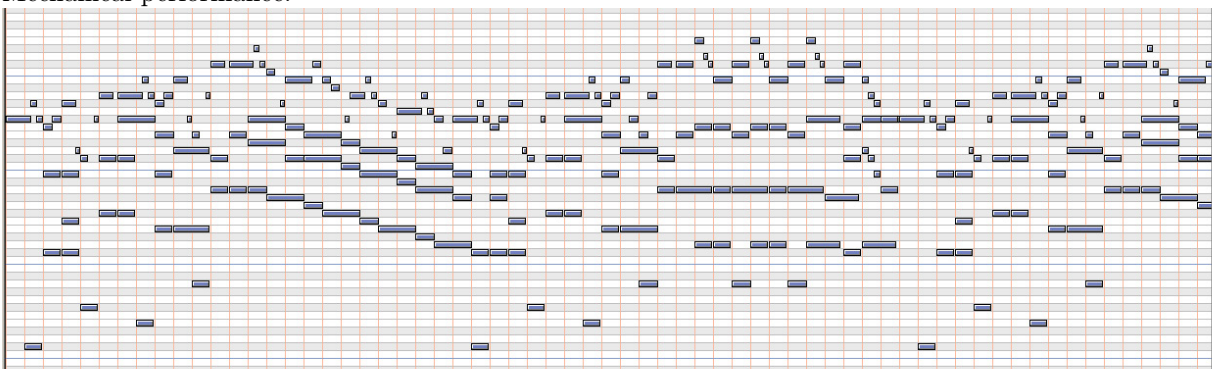


Figure 4.16: Chopin, *Op. 6, Mazurka 1*, first 30 seconds of expressive performance and mechanical performance.