Department of Computer Science
Series of Publications A
Report A-2015-5

Algorithms for melody search and transcription

Antti Laaksonen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium CK112, Exactum, Gustaf Hällströmin katu 2b, on November 20th, 2015, at twelve o'clock noon.

University of Helsinki
Finland

Supervisors
Esko Ukkonen, University of Helsinki, Finland
Kjell Lemström, University of Helsinki, Finland

Pre-examiners
Erkki Mäkinen, University of Tampere, Finland
David Meredith, Aalborg University, Denmark

Opponent
Pekka Kilpeläinen, University of Eastern Finland, Finland

Custos
Esko Ukkonen, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911, telefax: +358 9 876 4314

ISSN 1238-8645
ISBN 978-951-51-1701-4 (paperback)
ISBN 978-951-51-1702-1 (PDF)
Computing Reviews (1998) Classification: F.2.2, H.3.3, H.5.5, I.2.8
Helsinki 2015
Unigrafia

Algorithms for melody search and transcription

Antti Laaksonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
ahslaaks@cs.helsinki.fi
http://cs.helsinki.fi/u/ahslaaks/

PhD Thesis, Series of Publications A, Report A-2015-5
Helsinki, November 2015, 36+54 pages
ISSN 1238-8645
ISBN 978-951-51-1701-4 (paperback)
ISBN 978-951-51-1702-1 (PDF)

Abstract

This thesis studies two problems in music information retrieval: search for a given melody in an audio database, and automatic melody transcription. In both of the problems, the representation of the melody is symbolic, i.e., the melody consists of onset times and pitches of musical notes.

In the first part of the thesis we present new algorithms for symbolic melody search. First, we present algorithms that work with a matrix representation of the audio data, that corresponds to the discrete Fourier transform. We formulate the melody search problem as a generalization of the classical maximum subarray problem. After this, we discuss algorithms that operate on a geometric representation of the audio data. In this case, the Fourier transform is converted into a set of points in the two-dimensional plane.

The main contributions of the first part of the thesis lie in algorithm design. We present new efficient algorithms, most of which are based on dynamic programming optimization, i.e., calculating dynamic programming values more efficiently using appropriate data structures and algorithm design techniques. Finally, we experiment with the algorithms using real-world audio databases and melody queries, which shows that the algorithms can be successfully used in practice. Compared to previous melody search systems, the novelty in our approach is that the search can be performed directly in the Fourier transform of the audio data.

The second part of the thesis focuses on automatic melody transcription. As this problem is very difficult in its pure form, we ask whether using certain additional information would facilitate the transcription. We present two melody transcription systems that extract the main melodic line from an audio signal using additional information.

The first transcription system utilizes as additional information an initial transcription created by the human user of the system. It turns out that users without a musical background are able to provide the system with useful information about the melody, so that the transcription quality increases considerably. The second system takes a chord transcription as additional information, and produces a melody transcription that matches both the audio signal and the harmony given in the chord transcription. Our system is a proof of concept that the connection between melody and harmony can be used in automatic melody transcription.

Computing Reviews (1998) categories and subject descriptors:
F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems
H.3.3 [Information Storage and Retrieval] Information Search and Retrieval
H.5.5 [Information Interfaces and Presentation] Sound and Music Computing
I.2.8 [Artificial Intelligence] Problem Solving, Control Methods, and Search

General terms:
algorithms, pattern matching, information retrieval

Additional keywords and phrases:
dynamic programming optimization, maximum subarray problem, point set pattern matching, music cognition, melody and harmony

Acknowledgements

I am grateful to my supervisors Esko Ukkonen and Kjell Lemström for their guidance and support during my doctoral studies. I have learned a lot about scientific research from them.

The detailed feedback from my pre-examiners Erkki Mäkinen and David Meredith helped me to finalize the thesis.

The Department of Computer Science at the University of Helsinki has been a good place to study and work. I have received funding from the Helsinki Doctoral Programme in Computer Science and the Finnish Centre of Excellence for Algorithmic Data Analysis Research.

I have enjoyed being part of the algorithm community of the University of Helsinki. This has allowed me to discuss algorithms with a large number of people from secondary school students to professors.

This thesis is dedicated to my parents who led me to the worlds of music and programming.

Helsinki, November 2015
Antti Laaksonen

Contents

Original papers

The thesis consists of a summarizing overview and the following five publications, referred to as Paper I–V. These publications are reproduced at the end of the thesis.

I Antti Laaksonen: Finding maximum-density chains of segments with an application in music information retrieval. Submitted.

II Antti Laaksonen: Efficient and simple algorithms for time-scaled and time-warped music search. In *10th International Symposium on Computer Music Multidisciplinary Research (CMMR 2013)*, pages 621–630.

III Antti Laaksonen and Kjell Lemström: On finding symbolic themes directly from audio using dynamic programming. In *14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, pages 47–52.

IV Antti Laaksonen: Semi-automatic melody extraction using note onset time and pitch information from users. In *Sound and Music Computing Conference 2013 (SMC 2013)*, pages 689–694.

V Antti Laaksonen: Automatic melody transcription based on chord transcription. In *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, pages 119–124.

Chapter 1

Introduction

In this chapter, we state the problems that are studied in the thesis, and discuss related topics in the literature. After this, we describe the roles of the papers and specify the contributions of the author of the thesis. Finally, we outline the structure of the overview part of the thesis.

1.1 Research problems

We study two music information retrieval problems: (1) symbolic melody search in an audio database, and (2) automatic melody transcription. The problems are interesting both as theoretical challenges and as building blocks for real-world applications.

In both of the problems there are two components: an audio track and a symbolic representation of a melody. The audio track consists of digital samples of an audio signal, such as a track extracted from a CD. The symbolic melody consists of musical notes with onset times and pitches.

In the symbolic melody search problem, a collection of audio tracks is given, together with a melody query. The task is to find the audio tracks that contain the melody. Melody search algorithms can be used in music search engines that allow users to hum or whistle a melody, for example, and search for songs that contain the melody.

Automatic melody transcription is a natural subproblem in automatic music transcription. Given an audio track, the task is to extract the main melodic line and represent it using a symbolic notation. While melody transcription is an easy task for experienced music listeners, it has proven to be a difficult problem for computers.

Interestingly, the two problems are connected with each other: solving one of them would also solve the other. First, with an automatic melody

transcription algorithm, an audio database could be converted into a symbolic database of melodies. After this, the melody search problem would be easy to solve using standard pattern matching algorithms.

Second, a melody search algorithm could be transformed into a brute force melody transcription algorithm. Assume that we could check if any given melody appears in the audio. Thus, we could go through all possible melodies of certain length, check whether they appear in the audio, and create the final transcription by combining the appearing melodies.

Both signal processing techniques and symbolic algorithms are needed in the above problems. We focus on developing symbolic algorithms and use existing signal processing methods.

1.2 Related work

In this section we provide background for the topics of the thesis. Both symbolic melody search and automatic melody transcription are actively studied problems in the literature.

1.2.1 Symbolic melody matching

Symbolic melody matching is a pattern matching problem where patterns are symbolic representations of melodies.

A popular approach for melody matching is to represent melodies as strings so that each character in the string corresponds to one musical note [18, 36]. Usually, only the pitches of the notes are considered and the onset times are ignored. The benefit of the string representation is that standard string algorithms, such as approximate string matching with dynamic programming [38], can be used.

Another way to represent a melody is to describe the melody notes as events on a timeline [61]. In this representation, both the pitches and the onset times are considered. Pattern matching is performed using a technique called dynamic time warping [6]. This method also uses dynamic programming and resembles approximate string matching.

Finally, symbolic melody matching can be seen as a geometric problem [34, 56]. The idea is to represent each melody note as a point in the two-dimensional plane so that x coordinates denote onset times and y coordinates denote pitches. Using this representation, melody matching becomes a point set pattern matching problem [45].

1.2.2 Music search engines

Music search engines allow users to query information in a database that contains music. There are two common types of music search engine: query-by-humming systems, and audio fingerprint systems.

In a query-by-humming system [18], the user can hum a melody as a query, and the system searches for songs that contain the melody. Most query-by-humming systems [11,27] use a symbolic database and convert the query melody into a symbolic form. After this, symbolic melody matching algorithms can be used for searching for the melody.

A difficulty in constructing a query-by-humming system is how to create the symbolic database, because a large amount of music is available only in audio form and not in symbolic form. One possibility is to extract features, such as approximate melody pitches, automatically from the audio data [13, 46, 54]. However, automatic conversion of audio data into a symbolic representation is a difficult and unsolved problem.

Another type of music search engine is an audio fingerprint system [7,22]. In such a system, the user can submit an audio query that is searched for in an audio database. For example, the user can record music from the radio and submit the recording to the system to find out the name of the song. To make the search efficient, the database contains audio fingerprints that represent features of the songs in a compact form.

Both query-by-humming and audio fingerprints have been used in commercial music search engines. At the moment, popular commercial systems include SoundHound[1] and Shazam[2].

1.2.3 Automatic melody transcription

Automatic melody transcription is the problem of extracting the main melodic line from an audio recording [19]. It is one of the subproblems in automatic music transcription [2].

Various systems for automatic melody transcription have been developed during the last decade [42, 50]. The transcription usually begins with the discrete Fourier transform or a similar technique for calculating the strengths of frequencies within audio frames. After this, the problem is to determine which frequencies correspond to musical tones.

Most automatic melody transcription systems are based on calculating a salience function [39,49,50]. The salience function estimates the frequencies of musical tones in the audio signal using knowledge of the typical structure

[1] `http://www.soundhound.com/`
[2] `http://www.shazam.com/`

of musical tones. An alternative approach for salience methods is to use signal separation techniques [14].

The final step in automatic melody transcription is to construct the melody. A popular method for this is to use heuristic rules that describe features in typical melodies [19, 39, 50]. Some systems also use hidden Markov models with the Viterbi algorithm [14, 49].

While there have been many approaches for automatic melody transcription, no currently available algorithm reliably produces good melody transcriptions [2]. A central problem in current automatic melody transcribers is that the quality of the transcription varies a great deal. Transcriptions may be excellent for some inputs but poor for other inputs, and it is also difficult to estimate the quality of the transcription.

Being a difficult problem, automatic melody transcription can be facilitated by providing additional musical information for the transcription system. Typically, the information is created by the user of the system, which results in a semi-automatic transcription system [15, 25, 26].

1.3 Original papers

The thesis consists of five papers. Papers I, II and III present algorithms for symbolic melody search, and Papers IV and V discuss automatic melody transcription. In this section we briefly summarize the contents and the contributions of the papers.

Paper I. The paper introduces a new algorithmic problem that is a generalization of the classical maximum subarray problem. This problem corresponds to symbolic melody search in a matrix that is generated by the discrete Fourier transform. The main contributions of the paper are efficient algorithms for solving the problem.

Paper II. The paper discusses geometric algorithms for melody search. An $O(n^2m)$ time algorithm for time-scaled search and an $O(n(m + \log n))$ time algorithm for time-warped search are presented, where n and m denote the number of notes in the database and in the pattern, respectively. The proposed algorithms are more efficient than the previously published algorithms for the tasks, both in theory and practice.

Paper III. The paper presents an $O(nm \log n)$ time algorithm for time-warped melody search. The paper also defines a new search problem, approximately time-scaled search, and shows how the new algorithm can be extended to that problem. In addition, the paper contains experiments where symbolic melodies are searched for in an audio database.

Paper IV. The paper is based on a user experiment where participants listened to excerpts of audio tracks and marked down approximate onset times and pitches of melody notes. The paper studies what kind of information can be obtained from human listeners, and how the information can be used in semi-automatic melody transcription.

Paper V. The paper presents a melody transcription algorithm that uses a chord transcription as a starting point. The algorithm is based on the fact that the melody and the chords are connected with each other in music. The motivation for the algorithm is that automatic chord transcription seems to be easier than automatic melody transcription.

The author of the thesis has designed and implemented the new algorithms in the papers and conducted the experiments in the papers. He has also written all content in the papers, except that Paper II was written together with Kjell Lemström. The supervisors have given feedback for paper drafts and discussed the topics with the author.

1.4 Outline

The structure of the rest of the overview part is as follows:

Chapter 2 introduces topics and techniques that are used in the papers of the thesis. First, we discuss musical terminology and methods for processing and representing audio data. After this, we present algorithm design techniques that are used in our algorithms.

Chapter 3 discusses symbolic melody search, and summarizes the contents of Papers I, II and III. We define the algorithmic problems and present the main theoretical results and the ideas behind them. Finally, we compare the algorithms using real-world data.

Chapter 4 deals with automatic melody transcription, and describes the contents of Papers IV and V. We discuss the current difficulties in automatic music transcription, and present our transcription systems.

Chapter 2

Preliminaries

In this chapter, we provide background material for the topics of the thesis. We discuss the elements of music that we focus on, and show how an audio signal can be converted into a symbolic representation. Finally, we review algorithm design techniques that will be used later in the thesis.

2.1 Elements of music

Elements of traditional Western music include melody, harmony, timbre, and dynamics [59]. In this thesis, we mainly focus on melody, but also address the interplay between melody and harmony.

Melody and harmony are important elements of music because they form the basis of musical themes. As an example, consider the excerpt from Rachmaninov's Second Piano Concerto [43] in Figure 2.1. The melody and the harmony of the excerpt are shown in Figure 2.2, representing the musical content of the excerpt in a compact form.

Next we discuss the representation of melody and harmony in our algorithms. Using a standard convention, we model both the melody and the harmony as a sequence of events on a timeline. The melody consists of note events, while the harmony consists of chord changes.

2.1.1 Melody

We model the melody as a sequence of note events. Each note event (t, p) consists of two parts: the onset time t and the pitch value p.

The onset time is the beginning time of the note, measured in seconds. In a monophonic melody, all the onset times are distinct, whereas in a polyphonic melody, multiple notes may have the same onset time.

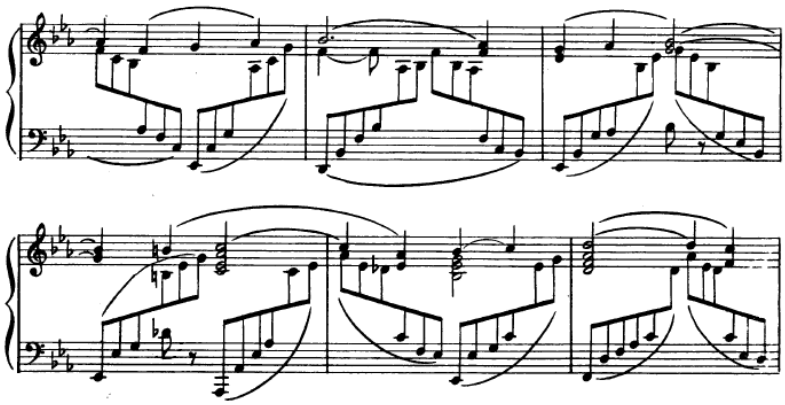

Figure 2.1: Complete score.

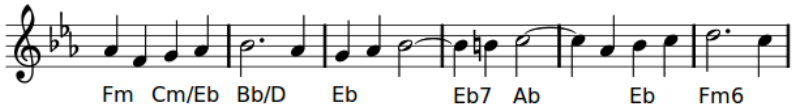

Figure 2.2: Melody and harmony.

The pitch denotes how high or low the note is located on the musical scale. We represent pitches using integer numbers so that the interval between pitches a and b is $|a - b|$ semitones.

Often, we use MIDI note numbers [21] for referring to the pitches. A MIDI note number is an integer in the range $[0, 127]$, and can be calculated from pitch f Hz by the formula $69 + \lfloor 12 \log_2(f/440) + 0.5 \rfloor$. The MIDI note number of middle C (261.6 Hz) is 60.

For example, the first four notes in the melody of Figure 2.2 can be represented as $[(0, 68), (0.5, 65), (1, 67), (1.5, 68)]$, assuming that the onset time difference between each consecutive note pair is 0.5 seconds.

Note that, unlike traditional musical notation, we do not specify the durations of the notes in the melody representation. The reason for this is that the pitches and the onset times of the notes describe the melody precisely enough for our purposes.

2.1.2 Harmony

We model the harmony as a sequence of chord changes. Each chord change (t, s) consists of two parts: the onset time t and the chord symbol s.

The onset times are measured in seconds similar to note events. The

chord symbol can contain the following parts:

Root note: The most important note in the chord, encoded using note names (C, C#, D, etc.).

Quality: Major (default), minor (letter "m"), augmented ("aug"), or diminished ("dim").

Interval: An extra note in the chord, represented by a diatonic interval above the root.

Bass note: If the bass note is different from the chord note, it is marked after the character '/'.

For example, "F", "Fm", "Fm7", and "Fm7/C" are valid chord symbols. Symbol "F" denotes an F major triad, symbol "Fm" denotes an F minor triad, symbol "Fm7" denotes an F minor triad with added seventh, and symbol "Fm7/C" contains C as the bass note.

Traditionally, chord symbols are mostly used in popular music, while Roman numeral analysis is preferred in classical music [5]. However, chord symbols can also be used in classical music, and this is the standard representation in automatic chord transcription [23].

For example, the first four chords in Figure 2.2 can be represented as [(0, "Fm"), (1, "Cm/Eb"), (2, "Bb/D"), (4, "Eb")].

2.1.3 Ambiguity

Melody and harmony are widely used concepts in the theory of music, but it is difficult to precisely define how they appear in real-world music. There can be several interpretations, and it is not possible to state that only one of them would be correct. For example, sometimes it is not clear whether a note is part of the melody or part of the accompaniment.

The problems studied in this thesis are inherently ambiguous, and one consequence of this is that evaluating the algorithms is difficult [28, 52]. Still, experienced music listeners "know" what melody and harmony are and can mark them down, even if they may disagree on some details. Databases with hand-made annotations are used for evaluating the algorithms [35].

2.2 From audio to symbols

The algorithms studied in the thesis are symbolic, i.e., they work with symbolic representations of melody and harmony. However, we use the algorithms for music audio processing.

To convert the audio data into a symbolic representation, we use the discrete Fourier transform that is a standard technique in music audio pro-

cessing. The Fourier transform reveals which frequencies are present in the audio signal, and the transform can be calculated efficiently.

After performing the Fourier transform, a symbolic representation of the potential musical notes in the audio signal can be constructed. We use two symbolic representations in our algorithms: matrix representation and geometric representation.

2.2.1 Discrete Fourier transform

A standard technique in music audio processing is the discrete Fourier transform (DFT). Given a list of digital samples of an audio signal, the DFT constructs a set of sinusoids whose sum corresponds to the samples. Each sinusoid has a fixed frequency, and its amplitude denotes the strength of the frequency in the audio signal.

Usually, the audio data is divided into small audio frames, each of which consists of consecutive samples within a time interval. Typically, the duration of a frame is between 0.01 and 0.1 seconds. Each frame is processed separately using the DFT during the conversion.

The result of the process is a spectrogram of the audio signal. The spectrogram is a matrix where each column corresponds to one audio frame, and the elements in the columns denote the strengths of different frequencies used in the analysis. The spectrogram shows which frequencies are present at each frame.

The DFT can be calculated in $O(n \log n)$ time using the FFT algorithm [10, Chapter 30], where n is the number of audio samples.

2.2.2 Musical tones

A musical tone consists of a set of frequencies that are approximate integer multiples of the fundamental frequency of the tone. The fundamental frequency determines the pitch that the listener hears, and the other frequencies contribute to the timbre of the tone. Musical instruments sound different because they have different timbres.

The challenge in music audio processing is that the audio signal is a complex combination of frequencies of different musical tones. In addition, some frequencies can be non-musical noise that should be ignored. For this reason, it is difficult to, for example, identify which musical tones are present in the signal or follow a melody played by a specific instrument.

As an example, Figure 2.3 shows the spectrogram of the excerpt from Rachmaninov's Second Piano Concerto. The horizontal axis denotes the

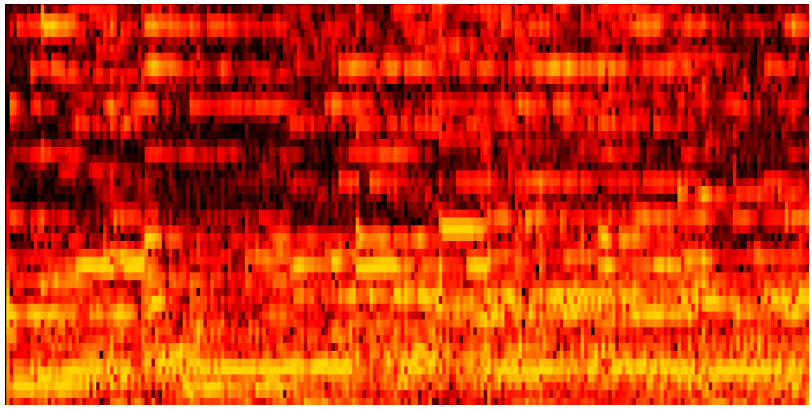

Figure 2.3: Spectrogram of an audio signal.

time, and the vertical axis denotes the frequencies. The lighter the color in the spectrogram, the stronger the frequency.

2.2.3 Matrix representation

The first symbolic representation that we use for the audio data in the thesis is the matrix representation. The matrix representation is a straightforward representation that resembles the spectrogram. The representation consists of n audio frames, each having m pitch elements.

The matrix representation is a matrix M of m rows and n columns. We use the notation $M[i, j]$ to access the matrix elements where $1 \leq i \leq m$ and $1 \leq j \leq n$. Each matrix element is a real number that denotes the strength of pitch i within audio frame j. The pitches are integer numbers and are measured in semitones.

Each pitch in the matrix representation is associated with a set of consecutive frequencies in the spectrogram. For example, the middle C (261.6 Hz) could be associated with the frequency range $[255, 265]$. After this, the strength for the pitch can be calculated as a sum of the corresponding elements in the spectrogram.

The matrix representation is used in Papers I, IV and V.

2.2.4 Geometric representation

The second symbolic representation that we use for the audio data is the geometric representation [34]. In this representation, each musical note is a point in the two-dimensional plane.

The geometric representation consists of a set S of n points. Each point $p \in S$ is a pair of real numbers, and we use notations $p.x$ and $p.y$ to refer to the x and y coordinates, respectively. The x coordinate corresponds to the onset time, and the y coordinate corresponds to the pitch. In addition, the point can be assigned a value that denotes the strength of the note.

When a and b are two points, $a < b$ exactly when either $a.x < b.x$ or $a.x = b.x$ and $a.y < b.y$. Often, we assume that the set S is sorted and can be accessed using the notation $S[1], S[2], \ldots, S[n]$.

The matrix representation can be converted into the geometric representation by representing each matrix element as a point. The conversion can be seen as a primitive automatic music transcription. To improve the quality of the representation, points with zero or near-zero strengths can be omitted because they are unlikely to represent real musical notes.

The geometric representation is used in Papers II and III.

2.3 Algorithm design

In this section we review algorithm design techniques that are used as building blocks for the algorithms in the thesis. First we present a data structure for maintaining the minimum value in a queue. After this, we discuss dynamic programming optimization and the increasing pointer method.

2.3.1 Queue minimum structure

In several of our algorithms, we use a data structure that maintains a queue of numbers and provides the following operations:

- *Insertion:* Insert number x at the end of the queue.
- *Removal:* Remove the first number from the queue.
- *Minimum:* Return the minimum number in the queue.

All the above operations can be implemented in amortized $O(1)$ time [55] using a data structure that can be seen as a simplification of a general Cartesian tree [17, 58]. This data structure is often used for solving the sliding window minimum problem.

The idea is to use two queues: queue Q contains the actual numbers, and queue A is an auxiliary structure that contains pointers to queue Q. The first pointer in A points to the minimum number in Q, and each following pointer points to a larger number in Q than the previous pointer.

Insertion: Let x be the new number that is inserted into the structure. As long as the last pointer in A points to a number that is equal to or greater than x, the pointer is removed from A. After this, x is inserted into Q, and a pointer to x is inserted into A.

Removal: If the first pointer in A points to the first number in Q, the pointer is removed from A. After this, the first number is removed from Q.

Minimum: The first pointer in A points to the minimum number in Q.

The amortized time complexity of each operation is $O(1)$ because each number and pointer is inserted and removed only once, and only the first and last elements of the queues are inspected.

The queue minimum structure is used in Papers I and III.

2.3.2 Dynamic programming optimization

Dynamic programming [10, Chapter 15] is a general algorithm design technique that we use in several of our algorithms. Our contributions in the algorithms lie in dynamic programming optimization: how to implement the dynamic programming computation as efficiently as possible.

As an example, consider the following problem: Given an array X of n numbers $X[1], X[2], \ldots, X[n]$, what is the maximum sum of numbers in a subarray whose length is between a and b?

Let $s(k)$ denote the sum of subarray $X[1], X[2], \ldots, X[k]$:

$$s(k) = \begin{cases} 0 & k \leq 0 \\ s(k-1) + X[k] & \text{otherwise.} \end{cases}$$

Let $f(k)$ denote the maximum sum of a subarray whose length is between a and b and whose last element is located at index k. Thus, the answer to the problem is $\max_{i=1}^{n} f(i)$. The value $f(k)$ can be calculated using the formula $f(k) = s(k) - \min_{i=a}^{b} s(k-i)$.

A direct computation of the $f(k)$ values would take $O(n^2)$ time because for each k, there are $b - a + 1 = O(n)$ possible choices for the length of the subarray. However, the computation can be implemented in $O(n)$ time using the queue minimum structure.

The idea is to calculate the value $f(k)$ by maintaining a queue that contains the elements $s(k-a), s(k-a-1), \ldots, s(k-b)$. Thus, the minimum value of $s(k-i)$ can be found in amortized $O(1)$ time from the queue. In addition, after each calculation, one element is added to the queue and one element is removed from the queue.

Most algorithms in Papers I, II and III are based on dynamic programming optimization in one way or another.

2.3.3 Increasing pointer method

Another technique worth mentioning is the increasing pointer method. In this technique, we maintain a pointer that has n possible values. At each step in the algorithm, the pointer value can be increased arbitrarily, but the pointer value is never decreased. Hence, the total time needed for updating the pointer is $O(n)$.

As an example of the technique, consider the following point set pattern matching problem:

Given two point sets S and P in the two-dimensional plane, the problem is to find all translations t such that $P + t \subset S$. Let n be the number of points in S, and let m be the number of points in P. The problem can be solved in $O(nm)$ time assuming that the point sets are sorted [45, 56].

The idea is to maintain values $X[1], X[2], \ldots, X[m]$ that refer to points in S. During the search, these values are used for matching the points of P with the points of S. Initially, $X[k] = 0$, for $k = 1, 2, \ldots, m$.

The search consists of n phases. At the beginning of each phase, the value $X[1]$ is increased. After this, for each $k = 2, 3, \ldots, m$, the value $X[k]$ is increased as long as $S[X[k]] - S[X[1]] < P[k] - P[1]$. Finally, if $S[X[k]] - S[X[1]] = P[k] - P[1]$, for all $k = 2, 3, \ldots, m$, one translation has been found and $t = S[X[1]] - P[1]$.

The time complexity of the algorithm is $O(nm)$ because the total increment for each pointer is $O(n)$ during the algorithm.

The increasing pointer method is used in Papers I and II.

Chapter 3

Melody search from audio

The first part of the thesis (Papers I, II and III) focuses on symbolic pattern matching algorithms that can be used in the following real-world application: given a collection of audio tracks and a symbolic melody query, find the track that contains the melody. The speciality in our work is that we use symbolic algorithms for searching for melody occurrences in audio data.

We approach the problem from both a theoretical and a practical viewpoint. The efficiency of the algorithms is important because audio files contain a lot of data, and the queries should be as fast as possible. The problems are also interesting as independent algorithm design problems, without connecting them to musical applications.

We discuss two types of algorithms for the melody search problem: matrix algorithms, that are based on the matrix representation of the audio data, and geometric algorithms, that are based on the geometric representation. We also conduct experiments where melodies are searched in an audio database using the algorithms.

3.1 Matrix algorithms

In this section we discuss algorithms that search for symbolic melody occurrences in the matrix representation of an audio track. We model the melody as a sequence of horizontal segments in the matrix, so that each segment corresponds to one note in the melody.

3.1.1 Problem statement

The input consists of an $m \times n$ matrix M of real numbers (audio data), and a set of k segments (melody pattern), indexed $1, 2, \ldots, k$. Matrix M is as defined in Section 2.2.3. Each segment in the pattern corresponds to one

3	1	7	5	2	9	6	4	1	8	5	3	9	7
5	2	9	6	4	1	8	5	3	9	7	4	2	8
6	4	1	8	5	3	9	7	4	2	8	6	3	1
8	5	3	9	7	4	2	8	6	3	1	7	5	2
9	7	4	2	8	6	3	1	7	5	2	9	6	4
2	8	6	3	1	7	5	2	9	6	4	1	8	5

Figure 3.1: A melody pattern occurrence in a matrix

note in the melody. The segment i is assigned range $[a_i, b_i]$, which denotes the allowed duration of the note, and if $i < k$, integer d_i, which denotes the pitch difference between notes i and $i + 1$.

The task is to find an occurrence of the melody in the matrix. In an occurrence, each segment is assigned row r_i ($1 \leq r_i \leq m$) and columns $[s_i, e_i]$ ($1 \leq s_i \leq e_i \leq n$). There are three requirements:

Note durations: The duration of each note in the occurrence must be within the given bounds, i.e., $a_i \leq e_i - s_i + 1 \leq b_i$, for $i = 1, 2, \ldots, k$.

Intervals: The intervals (pitch differences) between consecutive notes in the pattern and in the occurrence must be the same, i.e., $d_i = r_{i+1} - r_i$, for $i = 1, 2, \ldots, k - 1$.

Continuity: The next note begins immediately after the previous note ends, i.e., $e_i + 1 = s_{i+1}$, for $i = 1, 2, \ldots, k - 1$. Thus, we focus on "legato" melodies, that are common in real-world music.

In addition, an evaluation function is given, and the task is to find an occurrence that maximizes the value of the evaluation function. We study the following evaluation functions:

(E_1) *Sum:* $\sum_{i=1}^{k} \sum_{j=s_i}^{e_i} M[r_i, j]$.

(E_2) *Minimum of sums:* $\min_{i=1}^{k} \sum_{j=s_i}^{e_i} M[r_i, j]$.

(E_3) *Average:* $(\sum_{i=1}^{k} \sum_{j=s_i}^{e_i} M[r_i, j])/(e_k - s_1 + 1)$.

(E_4) *Minimum of averages:* $\min_{i=1}^{k} (\sum_{j=s_i}^{e_i} M[r_i, j])/(e_i - s_i + 1)$.

Figure 3.1 shows an example of a pattern occurrence. Assuming that the evaluation function E_1 is used, the evaluation value of this occurrence is $21 + 26 + 17 = 64$.

In Paper I, we present efficient algorithms for finding the best pattern occurrences in terms of the above functions. Each function poses a separate algorithm design problem: how to find a pattern occurrence that maximizes the value of the evaluation function as fast as possible.

3.1.2 Background

To our knowledge, the present problem has not been discussed before in the literature. However, the special case where $k = 1$, i.e., there is only one segment, is a well-known problem in computer science.

Sum: Finding the subarray with the maximum sum is a classical problem, and it can be solved in $O(n)$ time [4]. The extended version of the problem, where a length range for the subarray is given, can also be solved in $O(n)$ time [33] (Section 2.3.2).

Average: When working with averages, the maximum subarray problem is nontrivial only if a length range for the subarray is given. If there is no length range, an optimal solution is always a single element in the array. This problem is more difficult than the maximum sum problem, but an $O(n)$ time solution has been discovered [9].

3.1.3 Sum

Function E_1 calculates the sum of all matrix elements within the segments in the occurrence. This is the most straightforward evaluation function, and the techniques for maximizing the value of this function can also be extended to other evaluation functions.

First, we present a simple dynamic programming algorithm that works in $O(nmkw)$ time where $w = \max_{i=1}^{k} b_i - a_i$. After this, we show how the time complexity of the algorithm can be improved by calculating the dynamic programming values more efficiently.

The efficient dynamic programming calculation is based on the queue minimum structure (Section 2.3.1). The running time of the final algorithm is $O(nmk)$, so it is independent of the segment lengths.

3.1.4 Minimum of sums

Function E_2 calculates the minimum segment sum among all segments in the occurrence. The motivation is that the function attempts to ensure that all the notes in the melody occurrence are strong.

We present several algorithms for maximizing the value of this function. First, the $O(nmkw)$ time dynamic programming approach for E_1 can also be applied to E_2. However, the dynamic programming calculation is more difficult to speed up for this function.

For the general case, we present an $O(nmk \log w)$ time algorithm. The algorithm uses a balanced binary search tree to calculate the dynamic programming values more efficiently.

For the special case where the matrix is nonnegative (for example, a spectrogram), i.e., $M[i,j] \geq 0$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, we present another algorithm whose running time is only $O(nmk)$. In this case, the binary search tree can be avoided, which yields both a more efficient and simpler algorithm. This algorithm also uses the queue minimum structure.

3.1.5 Average

Function E_3 calculates the average of all matrix values within the occurrence. The benefit in calculating averages instead of sums is that long durations of notes do not increase the evaluation value.

Interestingly, calculating the maximum average can be reduced to calculating the maximum sum. The idea is to convert the problem "what is the maximum average?" into "is the maximum average at least x?".

Suppose that we have a sequence $a_1, a_2, \ldots, a_n$ of real numbers. The crucial observation is that the average of the sequence is at least x exactly when the sum of the sequence $a_1 - x, a_2 - x, \ldots, a_n - x$ is at least 0. Thus, we can binary search for the maximum average using an algorithm that calculates the maximum sum.

The number of steps in the binary search depends on the required precision of the result. Assuming that each matrix element can be represented using a constant number of bits, the number of steps is $O(\log n)$, and the time complexity becomes $O(nmk \log n)$.

3.1.6 Minimum of averages

Finally, function E_4 calculates the minimum average among all segments in the occurrence. The function is a combination of E_2 and E_3, and the maximum value for the function can be found in $O(nmk \log n)$ time by using binary search in a similar way.

3.1.7 Future work

We believe that $O(nmk)$ is the best possible time complexity for the problem because the matrix contains nm elements and the pattern contains k segments that may all have different lengths. For this reason, it does not seem possible to combine subproblems of different segments.

Using the algorithms in Paper I, the maximum value for function E_1 and nonnegative E_2 can be found in $O(nmk)$ time. It is an interesting open question whether the maximum value for general E_2 and functions E_3 and E_4 could also be calculated in $O(nmk)$ time.

Another interesting question is whether there is a simpler $O(nmk \log w)$ time algorithm for function E_2, for example, based on simple sorting. The current algorithm uses a complex balanced binary tree which results in large constant factors and a difficult implementation.

3.2 Geometric algorithms

In this section we approach the melody search problem from another viewpoint using the geometric representation of music. Using this representation, the melody search problem can be seen as a special case of two-dimensional point set pattern matching.

3.2.1 Problem statement

The input consists of two point sets in the two-dimensional plane: set S contains n points (musical piece), and set P contains m points (melody pattern). This corresponds to the definition in Section 2.2.4. We assume that the points in the sets are sorted and can be indexed like array elements.

The problem is to find functions $f\colon P \to S$ that correspond to occurrences of pattern P in point set S. The intervals between consecutive notes must be the same in the pattern and in the occurrence, i.e., $p_1.y - p_2.y = f(p_1).y - f(p_2).y$, for each $p_1, p_2 \in P$.

Depending on the type of search, we introduce additional constraints for x coordinates in an occurrence:

Exact search: The tempo of the melody is not changed, i.e., $p_1.x - p_2.x = f(p_1).x - f(p_2).x$ for each $p_1, p_2 \in P$.

Time-scaled search: The tempo of the melody is scaled by a constant scaling factor α, i.e., $\alpha(p_1.x - p_2.x) = f(p_1).x - f(p_2).x$ for each $p_1, p_2 \in P$. Each occurrence may have a distinct scaling factor.

Time-warped search: The tempo of the melody can be altered without restrictions, but the order of the notes cannot be changed, i.e., $p_1.x < p_2.x$ always when $f(p_1).x < f(p_2).x$.

Figure 3.2 shows examples of exact, time-scaled and time-warped melody occurrences in a point set.

In Papers II and III, we present new algorithms for time-scaled and time-warped melody search. The algorithms are both more efficient and conceptually easier than the earlier algorithms for the tasks.

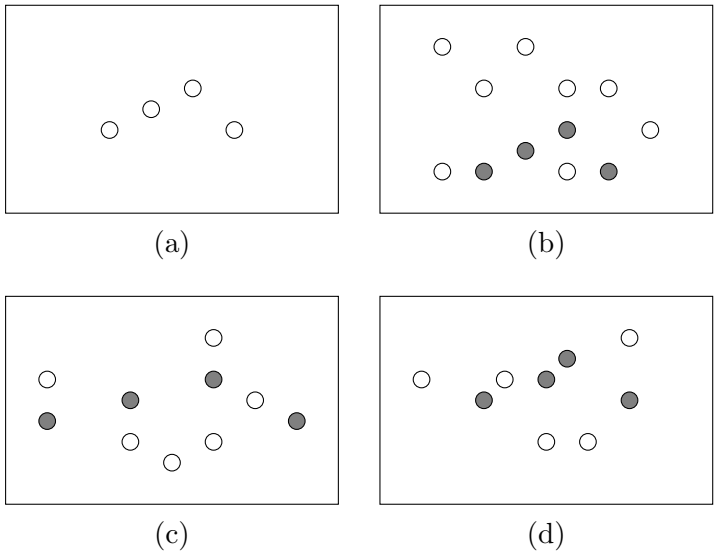

Figure 3.2: Geometric melody search: (a) melody pattern, (b) exact occurrence, (c) time-scaled occurrence, (d) time-warped occurrence

3.2.2 Background

Compared to traditional two-dimensional point set pattern matching [45], the speciality in the present problem is that scaling is allowed only horizontally but not vertically. For this reason, the problem is different from standard point set pattern matching.

All the above problems have been studied before. The exact search problem can be solved in $O(nm)$ time [45, 56]. The best previous algorithm for time-scaled search works in $O(n^2 m \log n)$ time [32], and the best previous algorithm for time-warped search works in $O(n^2 m)$ time [30].

3.2.3 Time-scaled search

In time-scaled search, the pattern can be scaled horizontally using constant factor α. Different pattern occurrences can have different scaling factors. The special case $\alpha = 1$ is similar to the exact search problem.

Time-scaled search is a difficult problem, because partial occurrences that have different scaling factors cannot be combined. For this reason, it seems difficult to use any dynamic programming techniques. It can be proved that the problem is 3SUM complete [29], so it is not probable that it could be solved considerably faster than in $O(n^2)$ time.

In Paper II, we present an $O(n^2m)$ time algorithm for the time-scaled search problem. The new algorithm is an extension to the previous $O(nm)$ time algorithm for exact search [56]. Like the previous algorithm, it uses a set of pointers to track pattern note positions.

3.2.4 Time-warped search

Time-warped search is the most flexible search type because any tempo changes are allowed as long as the notes appear in the correct order in the occurrence. Although the problem resembles time-scaled search, it is a very different problem from the viewpoint of algorithm design. Now dynamic programming can be used, because there are no fixed scaling factors.

In Paper II, we present an $O(n(m + \log n))$ time algorithm for the time-warped search problem. The algorithm uses a merge-like technique to maintain partial pattern occurrences. If the set of pitches is constant, i.e., the pitches are integers in the range $[0, c]$ where c is a constant, the time complexity of the algorithm is only $O(nm)$.

In Paper III, we present another algorithm for time-warped search. The algorithm is based on dynamic programming and the queue minimum data structure, and its time complexity is $O(nm \log n)$. Again, if the set of the pitches is constant, the time complexity is only $O(nm)$.

Paper III also introduces two extensions to the problem, which are useful in practice. First, the notes are assigned strengths, and an occurrence with maximum total strength has to be found. Second, each consecutive note pair in the pattern is assigned a range of allowed scaling factors. The proposed algorithm supports both of the extensions.

3.2.5 Future work

An interesting special case for exact and time-scaled search is the one-dimensional problem where each point has the same y coordinate, i.e., each note has the same pitch. It seems that the one-dimensional problem captures the difficulty of the general problem [29], so concentrating on the one-dimensional problem first could be a good approach.

The current best algorithms for exact and time-scaled search, with running times $O(nm)$ and $O(n^2m)$, respectively, both use the idea of maintaining a set of pointers that increase during the search. To get rid of the m factor, this technique should be replaced with something else. Possibly, ideas from efficient string matching algorithms could be used [57].

Another interesting question is whether the time-warped search problem could be solved in $O(nm)$ time without assuming that the set of the pitches

is constant. The algorithm in Paper II is already close to this because it only needs one $O(n \log n)$ time sorting as preprocessing, and the rest of the algorithm works in $O(nm)$ time.

3.3 Experiments

We conducted experiments where we searched for symbolic melodies in a real-world audio database using the algorithms presented in this chapter. We searched for themes in a collection of Tchaikovsky's six symphonies. Papers I and III describe the experiments in detail.

3.3.1 Algorithms

We compared four matrix algorithms and two geometric algorithms in the experiments. The matrix algorithms use evaluation functions E_1, E_2, E_3 and E_4, and the geometric algorithms are based on time-warped search, using (1) unlimited and (2) limited scaling range.

In practice, there are two main differences in matrix and geometric algorithms. First, in matrix algorithms all audio frames within the occurrence contribute to the strength of the melody, while in geometric algorithms each note event corresponds to a single audio frame. Second, in geometric algorithms note events with near-zero strengths can be omitted.

3.3.2 Material

Our audio database consisted of Tchaikovsky's six symphonies [12]. We converted the material from CD tracks into mono WAV files using a sample rate of 44,100 Hz. There were 25 tracks in the database, and the total duration of the tracks was 4 hours and 22 minutes.

The symbolic melody queries were taken from Barlow and Morgenstern's *A Dictionary of Musical Themes* [1]. The melodies used in the experiments are available online as MIDI files [37]. There were a total of 75 melodies, with durations between 5 seconds and 25 seconds.

3.3.3 Evaluation

For each melody query and algorithm, we calculated the rank of the correct audio track in the sorted list of tracks. The sorting criterion was the maximum occurrence value of the melody in the track. For example, rank 1 means that the correct track was the first track in the list. The rank was always between 1 (best) and 25 (worst).

Algorithm	Rank 1	Rank 1–3
Expected random	3	9
Matrix E_1	6	14
Matrix E_2	7	14
Matrix E_4	14	18
Geometric 1	18	26
Geometric 2	22	31
Matrix E_3	26	32

Table 3.1: Results of the experiments

As a baseline in the evaluation we used an algorithm that returns random occurrence values. The probability that this algorithm would yield rank 1 for a melody query is $1/25$, so when searching for 75 melodies, the expected number of rank 1 results is 3.

3.3.4 Results

Table 3.1 shows the results of the experiments. For each algorithm, the number of queries with rank 1 and with rank 1–3 are shown.

It turned out that there are large differences between the evaluation functions for matrix algorithms. The most suitable evaluation function for this material was E_3 that calculates the average of matrix elements within the occurrence. Using function E_3, 26 out of 75 queries had rank 1, the best result in the experiments.

On the other hand, the results of other matrix algorithms were considerably weaker. The evaluation functions E_1 and E_2 that calculate sums do not seem to be good choices because their results were barely better than the results of the random algorithm.

The results of the geometric algorithms were close to each other. This is not surprising because both the algorithms were based on time-warped search, and the only difference was that algorithm 2 restricted the range of allowed scaling factors between melody notes. The geometric algorithms outperformed all matrix algorithms except E_3.

3.4 Discussion

The main contributions of this chapter are in algorithm design. The discussed problems are interesting as independent problems, because the matrix problems extend the well-known maximum subarray problem [4], and

the geometric problems are variations of point set pattern matching [45].

The experiments show that the algorithms can be successfully used in practice for searching for symbolic melodies in audio material. The Tchaikovsky dataset used in the experiments is difficult because many melody occurrences in the symphonies are subtle. Thus, it is challenging evaluation material for the algorithms.

It is evident that the choice of evaluation function in matrix algorithms is important, so it might be worthwhile to study more evaluation functions. However, from the algorithm design viewpoint, each evaluation function is a different problem and it may not be possible to design an efficient algorithm for a complex evaluation function.

Chapter 4

Automatic melody transcription

The second part of the thesis (Papers IV and V) discusses automatic melody transcription, i.e., the extraction of the most important melody from the audio signal. Automatic melody transcription is a difficult problem, and despite many proposed approaches and methods, to date, no automatic system is capable of reliably producing good-quality melody transcriptions.

While melody transcription is difficult for computers, most human listeners can recognize melodies and provide information about the notes in the melody, even if they cannot produce a complete melody transcription. In our first study, we use the information produced by human listeners in a melody transcription system.

Our second study focuses on the connection between melody and harmony in music. Automatic chord transcription seems to be easier than automatic melody transcription, and there are already systems that produce good chord transcriptions. For this reason, we use chord information to improve the quality of the melody transcription.

4.1 User-aided transcription

In general, music listeners have an understanding of what a melody is and can recognize melodies in music, even if they do not have the skills to produce a melody transcription [24]. It turns out that listeners can also help the computer to create a melody transcription.

In Paper IV, we present a system that creates a melody transcription from an audio signal together with a user. First, the user gives information about approximate note onset times and pitches. After this, the system creates the melody transcription using both the information given by the user and the information in the audio data.

Of course, before creating such a system, it is important to know what kind of information about the melody users are able to produce. For this reason, we conducted an experiment with users to find out what characteristics in the melody they can recognize and write down.

4.1.1 Background

Creating the transcription with the help of a user is called semi-automatic transcription. For example, users can provide information about instruments [25], identify some notes in the melody [26], or select the audio source that corresponds to the melody [15].

A system somewhat similar to ours is Songle [20]. This system creates a preliminary transcription automatically, and after this the users can work on the transcription collaboratively.

The ability to recognize musical pitches has been studied in psychology [41]. An interesting phenomenon, that we also noticed in our experiment, is that participants without a musical background could only compare pitches accurately when they were played with the same instrument.

4.1.2 Listening experiment

In the experiment, the participants were given two melody excerpts with an accompaniment extracted from real-world recordings. The first excerpt was taken from the Star Wars theme by John Williams, and the second excerpt was taken from the opera, *Parsifal*, by Richard Wagner.

For the experiment, we created a user interface where it was possible to listen to the original version of the excerpt, mark down an initial melody transcription and listen to the synthesized transcription. Figure 4.1 shows the layout of the user interface.

The participants were asked to perform two tasks. First, they had to mark down the locations where a note in the melody begins. After this, the participants were shown correct locations where notes begin, and they were asked to determine the pitch for each note. To do this, the interface had commands to make the pitch lower and higher and play the original sound and the synthesized pitch repeatedly.

We had a total of 30 participants in the experiment. Group A consisted of 15 participants without a musical background, and Group B consisted of another 15 participants with a musical background. None of the participants were experienced music transcribers.

It turned out that the first task in the experiment was easy for most of the participants. Both listeners without and with a musical background

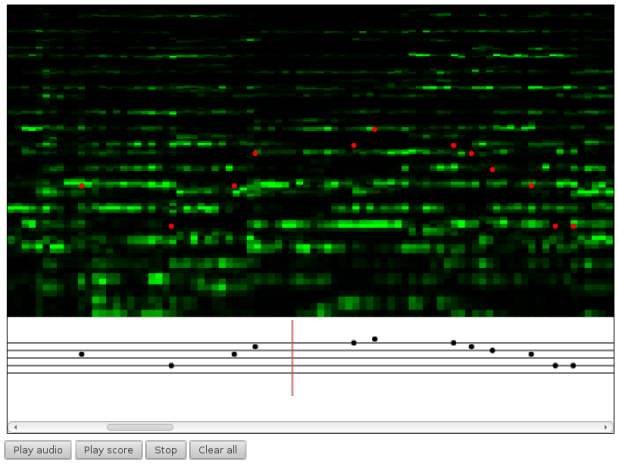

Figure 4.1: The user interface used in the experiment.

were able to accurately mark down the places where melody notes begin. Surprisingly, the best performers in the task were participants without a musical background but who were active computer gamers.

The success in the second task, however, strongly depended on musical background. Most participants with a musical background were able to determine all pitches correctly, whereas almost all participants without a musical background could not determine exact pitches but only select pitches that were near the correct pitches.

4.1.3 Transcription system

We created a system that takes as input an audio track and an approximate melody transcription created by the user. The transcription consists of a sequence of notes with onset times and pitches. However, the notes in the transcription may not be exactly correct, because the transcription is created by a user without transcription experience.

The system is based on a dynamic programming algorithm, and it creates a melody transcription that is based both on the approximate transcription by the user and the actual audio data. Depending on the configuration, the system may use either only approximate note onset times, or both approximate note onset times and pitches.

The assumption in the design of the system was that the users do not have a musical background. For this reason, the pitches in the approximate transcription are not exact, and the challenge for the algorithm is to find the correct pitches by using the information in the audio data.

4.1.4 Evaluation

We evaluated our system using a collection of orchestral music recordings. For each excerpt, we created a simulated approximate transcription. We estimated the errors in transcriptions by deriving parameters for normal distribution from the data collected in the user experiment.

We compared the results of our system with a state-of-the-art automatic melody transcription system. For each audio file, we calculated the accuracy of the transcription as the ratio of correctly annotated audio frames and all audio frames. The total accuracy was calculated as the average of all transcription accuracies.

The accuracy of the automatic system was 0.33, whereas the accuracy of our system was 0.24 using only note onset time information, and 0.66 using both onset time and pitch information. Thus, the quality of the transcription was much better using our system with full information than using the automatic system.

Of course, the drawback in our system is that it is not an automatic system but it requires that the user helps the computer to create the transcription. However, it is interesting that users without a musical background are able to produce information that clearly improves the quality of the automatic melody transcription.

4.2 Chord-based transcription

For human listeners, melody transcription is an easier task than chord transcription, because it is usually easy to follow the melody, while recognizing the harmony requires more expertise. Surprisingly, for automatic systems, the situation is just the opposite: the current automatic chord transcribers are better than the automatic melody transcribers [35].

In Paper V, we approach the automatic melody transcription task from the chord transcription perspective. If a chord transcription is available, it can be used as a starting point for a melody transcription because there is a strong connection between the melody and the chords. For example, the melody is usually based on the notes of the underlying chord.

Given that chord transcription is easier for computers than melody transcription, we can first create an automatic chord transcription, and then use this transcription to create an automatic melody transcription. Thus, the melody transcription algorithm is given both the audio data and the chord transcription as input.

4.2.1 Background

A chord transcription describes the harmony of the music, and consists of a sequence of chord changes on a timeline. Several systems for automatic chord transcription have been developed [8, 40].

Most automatic chord transcription systems are based on calculating and comparing chromagrams [16]. The quality of the transcription can be improved by smoothing the chord sequence [53]. A popular approach for this is to use a hidden Markov model together with statistical information about probabilities of chord changes [31, 49].

To our knowledge, our system is the first system that creates the melody transcription based on a chord transcription. However, many previous studies have combined key, chord and pitch estimation [3, 44, 47].

4.2.2 Transcription system

Our chord-based melody transcription system is given an audio track and a chord transcription, and it produces a melody transcription. The system divides the transcription into three phases: segmentation, key estimation, and pattern matching.

In the segmentation phase, the audio data is partitioned into segments of approximately equal length using the chord transcription. The idea is to select the segments so that each segment has a stable chord, and the segments can be processed separately later in the algorithm.

In the key estimation phase, the key of the excerpt is estimated using the chord transcription. The key determines which notes are most likely to appear in the melody. For example, if the key is D major, the most typical notes in the melody are the notes that belong to the D major scale, i.e., D, E, F#, G, A, B and C#.

Finally, in the pattern matching phase, the system selects a suitable melody pattern for each segment. A melody pattern is a group of notes that have some onset times and pitches. The system attempts to select a melody pattern that matches the audio data, the key of the excerpt and the chord of the segment.

4.2.3 Evaluation

We evaluated our melody transcription system using a collection of Finnish popular music. The collection consisted of song excerpts in audio form, together with hand-made melody and chord transcriptions. We used the melody transcriptions as the ground truth.

We measured two values to evaluate the melody transcriptions: precision and recall. Precision is the ratio of the number of correctly transcribed notes to the total number of notes in the transcription. Recall is the ratio of the number of correctly transcribed notes to the total number of notes in the ground truth.

We used both automatically created and hand-made chord transcriptions in the evaluation. Using automatic chord transcription, the precision of the melody transcription was between 0.40 and 0.45, and the recall was between 0.60 and 0.65. Using hand-made transcriptions, the precision was between 0.55 and 0.60, and the recall was between 0.50 and 0.55.

The evaluation results show that automatic melody transcription and chord transcription can be successfully combined. As expected, the precision of the melody transcription was better using hand-made chord transcriptions, but there were no large differences between the results using automatic and hand-made chord transcriptions.

4.3 Discussion

Melody transcription is easy for experienced human listeners, because they can hear and follow the melody. Computers, on the other hand, extract from the data something that may or may not be the melody. Sometimes automatic transcriptions are excellent, but usually they contain primitive mistakes that no human listener would ever make.

We feel that adding musical knowledge to the transcription algorithm is an ambivalent technique. Musical knowledge usually improves the transcription result to some extent, but at the same time, it increases the amount of "guessing" in the algorithm, because it typically involves making decisions based on probabilities.

For example, it is true that the melody notes follow the scale of the underlying chord with high probability. However, human transcribers do not trust musical knowledge but their ears. Even if most melody notes are outside the current scale, as it can be in modern music, human transcribers write down the correct melody without hesitation.

Of course, it is not clear that automatic melody transcription should use methods similar to those used by human transcribers, or that guessing should be avoided. In any case, we believe that there should be more collaboration between human transcribers and designers of automatic transcription systems.

References

[1] H. Barlow and S. Morgenstern. *A Dictionary of Musical Themes*, Crown Publishers, 1948.

[2] E. Benetos, S. Dixon, D. Giannouis, H. Kirchhoff and A. Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3), 407–434, 2013.

[3] E. Benetos, A. Jansson and T. Weyde. Improving automatic music transcription through key detection. In *AES 53rd International Conference on Semantic Audio*, 2014.

[4] J. Bentley. *Programming Pearls*, Addison-Wesley, 1986.

[5] B. Benward and M. Saker. *Music in Theory and Practice, Vol. 1*, McGraw-Hill, 2008.

[6] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Workshop on Knowledge Discovery in Databases*, 359–370, 1994.

[7] P. Cano, E. Battle, T. Kalker and J. Haitsma. A review of audio fingerprinting. *Journal of VLSI Signal Processing*, 41(3), 271–284, 2005.

[8] T. Cho, R. Weiss and J. Bello. Exploring common variations in state of the art chord recognition systems. In *Sound and Music Computing Conference*, 1–8, 2010.

[9] K. Chung and H. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM Journal on Computing*, 34(2), 373–387, 2005.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms* (3rd ed.), MIT Press, 2009.

[11] R. B. Dannenberg, W. P. Birmingham, B. Pardo, N. Hu, C. Meek and G. Tzanetakis. A comparative evaluation of search techniques for

query-by-humming using the MUSART testbed. *Journal of the Association for Information Science and Technology*, 58(5), 687–701, 2007.

[12] Deutsche Grammophon 423 504-2. Tchaikovsky Symphonies 1–6, conducted by Herbert von Karajan.

[13] A. Duda, A. Nürnberger and S. Stober. Towards query by singing/humming on audio databases. In *8th International Conference on Music Information Retrieval*, 331–334, 2007.

[14] J. L. Durrieu, G. Richard, B. David and C. Févotte. Source/filter model for unsupervised main melody extraction from polyphonic audio signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3), 564–575, 2010.

[15] J. L. Durrieu and J.-P. Thiran. Musical audio source separation based on user-selected F0 track. In *10th International Conference on Latent Variable Analysis and Signal Separation*, 438–445, 2012.

[16] T. Fujishima. Realtime chord recognition of musical sound: a system using Common Lisp music. In *25th International Computer Music Conference*, 464–467, 1999.

[17] H. Gabow, J. Bentley and R. Tarjan. Scaling and related techniques for geometry problems. In *16th Annual ACM Symposium on Theory of Computing*, 135–143, 1984.

[18] A. Ghias, J. Logan, D. Chamberlin and B. Smith. Query by humming. Music information retrieval in an audio database. In *3rd ACM International Conference on Multimedia*, 231–236, 1995.

[19] M. Goto. A real-time music-scene-description system: predominant-F0 estimation for detecting melody and bass lines in real-world audio signals. *Speech Communication*, 43(4), 311–329, 2004.

[20] M. Goto, K. Yoshii, H. Fujihara, M. Mauch and T. Nakano. Songle: a web service for active music listening improved by user contributions. In *12th International Society for Music Information Retrieval Conference*, 311-316, 2011.

[21] R. Guerin. *MIDI Power!: The Comprehensive Guide*, Cengage Learning PTR, 2005.

[22] J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *3rd International Conference on Music Information Retrieval*, 107–115, 2002.

[23] C. Harte, M. Sandler, S. Abdallah and E. Gómez. Symbolic representation of musical chords: a proposed syntax for text annotations. In *6th International Conference on Music Information Retrieval*, 66–71, 2005.

[24] P. Howell, I. Cross and R. West (eds.). *Musical Structure and Cognition*, Academic Press (London), 1986.

[25] H. Kirchhoff, S. Dixon and A. Klapuri. Shift-variant non-negative matrix deconvolution for music transcription. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 125–128, 2012.

[26] H. Kirchhoff, S. Dixon and A. Klapuri. Multitemplate shift-variant non-negative matrix deconvolution for semi-automatic music transcription. In *13th International Society for Music Information Retrieval Conference*, 415–420, 2012.

[27] A. Kotsifakos, P. Papapetrou, J. Hollmén, D. Gunopulos and V. Athitsos. A survey of query-by-humming similarity methods. In *5th International Conference on Pervasive Technologies Related to Assistive Environments*, 5–8, 2012.

[28] A. Laaksonen. Ambiguity in automatic chord transcription: recognizing major and minor chords. In *10th International Workshop on Adaptive Multimedia Retrieval*, 203–213, 2012.

[29] A. Laaksonen. Two-dimensional point set pattern matching with horizontal scaling. In *6th Symposium on Future Directions in Information Access*, 38–40, 2015.

[30] M. Laitinen and K. Lemström. Dynamic programming in transposition and time-warp invariant polyphonic content-based music retrieval. In *12th International Society for Music Information Retrieval Conference*, 369–374, 2011.

[31] K. Lee and M. Slaney. Acoustic chord transcription and key extraction from audio using key-dependent HMMs trained on synthesized audio. *IEEE Transactions on Audio, Speech and Language Processing*, 16(2), 291–301, 2008.

[32] K. Lemström. Towards more robust geometric content-based music retrieval. In *11th International Society for Music Information Retrieval Conference*, 577–582, 2010.

[33] Y. L. Lin, J. Tao and C. Kun-Mao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences* 65(3), 570–586, 2002.

[34] D. Meredith, G. Wiggins and K. Lemström. Pattern induction and matching in polyphonic music and other multidimensional datasets. In *5th World Multiconference on Systemics, Cybernetics and Informatics*, 22–25, 2001.

[35] The Music Information Retrieval Evaluation eXchange (MIREX). `http://www.music-ir.org/mirex/wiki/MIREX_HOME`

[36] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3), 161–175, 1990.

[37] The Multimedia Library's Electronic Dictionary of Musical Themes. `http://www.multimedialibrary.com/barlow/`.

[38] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1), 31–81, 2001.

[39] R. Paiva, T. Mendes and A. Cardoso. Melody detection in polyphonic musical signals: exploiting perceptual rules, note salience, and melodic smoothness. *Computer Music Journal*, 30(4), 80–98, 2006.

[40] H. Papadopoulos and G. Peeters. Large-scale study of chord estimation algorithms based on chroma representations and HMM. In *International Workshop of Content-Based Multimedia Indexing*, 53–60, 2007.

[41] M. Pitt and R. Crowder. The role of the spectral and dynamic cues in imagery for musical timber. *Journal of Experimental Psychology: Human Perception and Performance*, 18(3), 723–738, 1992.

[42] G. E. Poliner, D. P. Ellis, A. F. Ehmann, E. Gómez, S. Streich and B. Ong. Melody transcription from music audio: approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4), 1247–1256, 2007.

[43] Sergei Rachmaninov. Piano Concerto No. 2, Op. 18. Mugziz, Moscow, 1947–1948.

[44] S. Raczyński, E. Vincent, F. Bimbot and S. Sagayama. Multiple pitch transcription using DBN-based musicological models. In *11th International Society for Music Information Retrieval Conference*, 363–368, 2010.

[45] P. de Rezende and D. Lee. Point set pattern matching in d-dimensions. *Algorithmica*, 13(4), 387–404, 1995.

[46] M. Rocamora, P. Cancela and A. Pardo. Query by humming: automatically building the database from music recordings. *Pattern Recognition Letters*, 36, 272–280, 2014.

[47] T. Rocher, M. Robine, P. Hanna, L. Oudre and Y. Grenier. Concurrent estimation of chords and keys from audio. In *11th International Society for Music Information Retrieval Conference*, 141–146, 2010.

[48] C. Romming and E. Selfridge-Field. Algorithms for polyphonic music retrieval: the Hausdorff metric and geometric hashing. In *8th International Conference on Music Information Retrieval*, 457–462, 2007.

[49] M. Ryynänen and A. Klapuri. Automatic transcription of melody, bass line, and chords in polyphonic music. *Computer Music Journal*, 32(3), 72–86, 2008.

[50] J. Salamon and E. Gómez. Melody extraction from polyphonic music signals using pitch contour characteristics. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(6), 1759–1770, 2012.

[51] J. Salamon, J. Serrà and E. Gómez. Tonal representations for music retrieval: from version identification to query-by-humming. *International Journal of Multimedia Information Retrieval*, 2(1), 45–58, 2013.

[52] J. Salamon and J. Urbano. Current challenges in the evaluation of predominant melody extraction algorithms. In *13th International Society for Music Information Retrieval Conference*, 289–294, 2012.

[53] A. Sheh and D. Ellis. Chord segmentation and recognition using EM-trained hidden Markov models. In *4th International Conference on Music Information Retrieval*, 185–191, 2003.

[54] I. Suyoto, A. Uitdenbogerd and F. Scholer. Searching musical audio using symbolic queries. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2), 372–381, 2008.

[55] A. Thruu, J. Radoszewski and E. Panzer. Solution for task "Sound" in *BOI 2007, Tasks and Solutions*. `http://www.boi2007.de/tasks/book.pdf`

[56] E. Ukkonen, K. Lemström and V. Mäkinen. Sweepline the music! In *Computer Science in Perspective*, Lecture Notes in Computer Science 2598, 330–342, Springer, 2003.

[57] E. Ukkonen. Geometric point pattern matching in the Knuth-Morris-Pratt way. *Journal of Universal Computer Science*, 16(14), 1902–1911, 2010.

[58] J. Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4), 229–239, 1980.

[59] J. White. *The Analysis of Music*, Prentice-Hall, 1976.

[60] H.-M. Yu, W.-H. Tsai and H.-M. Wang. A query-by-singing technique for retrieving polyphonic objects of popular music. In *Second Asia Information Retrieval Symposium*, 439–453, 2005.

[61] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *ACM SIGMOD International Conference on Management of Data*, 181–192, 2003.