Department of Computer Science
Series of Publications A
Report A-2000-04

String Matching Techniques for Music Retrieval

Kjell Lemström

University of Helsinki
Finland

Department of Computer Science
Series of Publications A
Report A-2000-04

String Matching Techniques for Music Retrieval

Kjell Lemström

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XIV, University Main Building, on November 24th, 2000, at 12 o'clock noon.

University of Helsinki
Finland

Contact information

Postal address:

Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 44441

ISSN 1238-8645
ISBN 951-45-9573-4
Computing Reviews (1998) Classification: H.3.3, F.2.2, J.5
Helsinki 2000
Helsinki University Press

String Matching Techniques for Music Retrieval

Kjell Lemström

Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
Kjell.Lemstrom@cs.Helsinki.Fi
http://www.cs.Helsinki.Fi/Kjell.Lemstrom/

PhD Thesis, Series of Publications A, Report A-2000-04
Helsinki, November 2000, 56 + 56 pages
ISSN 1238-8645, ISBN 951-45-9573-4

Abstract

In content-based music information retrieval (MIR), the primary task is to find exact or approximate occurrences of a monophonic musical query pattern within a music database, which may contain either monophonic or polyphonic documents. In tonal music, the occurrences that are searched for may be presented in any musical key, making transposition invariance a useful property for an MIR algorithm. Moreover, as the term 'content-based' indicates, a query should be comprised solely of musical information. It cannot contain, e.g., annotations, such as keywords or lyrics of a musical document.

In this thesis, various aspects of music comparison and retrieval, including both theoretical and practical, will be studied. The general pattern matching scheme based on approximate string matching framework will be applied, and estimates on the theoretical complexity of such pattern matching will be given. Moreover, the aforementioned framework is to be tuned so that it would be possible to retrieve symbolically encoded musical documents efficiently. This modified framework is plausible as regards to findings in musicology. It enables the consideration of context in music and gives novel transposition invariant distance measures, avoiding some shortcomings of the straightforward applications of conventional string matching measures.

A database containing only monophonic music represents a special case, for which the applied techniques can be implemented more efficiently. Nevertheless, in this thesis, both the cases of monophonic and polyphonic music databases will be considered and solutions applying bit-parallelism, a very efficient way to implement dynamic programming, will be presented. In particular, the thesis introduces our novel, very efficient algorithm for polyphonic databases.

Further, the thesis gives two novel music alphabet reduction schemes. They both have strong musical basis, and they are used as essential components in our software system prototype called SEMEX. Finally, the thesis introduces SEMEX and describes how the described techniques and observations are implemented.

Computing Reviews (1998) Categories and Subject Descriptors:

H.3.3	Information Storage and Retrieval: Information Search and Retrieval
F.2.2	Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems — pattern matching
J.5	Computer Applications: Arts and Humanities — music

General Terms:
Algorithms, Distance Measures, Musical Similarity

Additional Key Words and Phrases:
Music Information Retrieval, String Algorithms, Online Pattern Matching

Acknowledgements

First of all, I would like to thank my advisor, Professor Esko Ukkonen, for his support and enthusiasm in my work. Especially, I feel gratitude for his patience: Earlier, I changed the topic of my master's thesis during that project. Now I did it again during my PhD project.

I have been privileged to work and write papers about music retrieval with skilled people, such as Prof. Esko Ukkonen, Prof. Jorma Tarhio, Dr. Lauri Hella, Dr. Pauli Laine, Dr. Ilya Shmulevich, Dr. Olli Yli-Harja, MSc Atso Haapaniemi, and MSc Sami Perttu. I am looking forward to further collaboration!

Dr. Jukka Teuhola and Dr. Pierre-Yves Rolland are greatly acknowledged for their insightful comments that have improved the summary of the thesis. I am also thankful for the comments and suggestions provided by Dr. Geraint Wiggins and MA David Meredith. Moreover, I would also like to thank MA Marina Kurtén for helping me, often with a very short notice, with the language.

For the financial support I would like to thank the Helsinki Graduate School in Computer Science and Engineering (HeCSE), the Finnish Cultural Foundation, the Academy of Finland, and Nokia. The Department of Computer Science, headed by Professors Martti Tienari, Esko Ukkonen and Timo Alanko, has provided excellent working environment and conditions, for which I am very thankful.

I am also indebted of the great official and unofficial support that I have got during these years from many friends and colleagues in our department; Oskari Heinonen, Jani Jaakkola, Mika Klemettinen, Jaakko Kurhila, Matti Luukkainen, Juha Sievänen, and many more.

Above all, I would like to thank my wife Vera and our daughters Anna and Eeva, my parents with their families, my brothers and sisters, and parents-in-law, for their supportive existence.

Permissions of reprint have been granted by the associations in whose proceedings the original papers of this thesis have been published. The permissions are acknowledged.

London, October, 2000

Contents

List of Original Publications

The thesis is based on the following original articles, which are referred to in the text by their Roman numerals.

I Kjell Lemström and Pauli Laine. Musical information retrieval using musical parameters. In *Proceedings of the 1998 International Computer Music Conference (ICMC'98)*, pages 341–348, University of Michigan, Ann Arbor, October 1–6, 1998.

II Kjell Lemström and Jorma Tarhio. Searching monophonic patterns within polyphonic sources. In *Content-Based Multimedia Information Access Conference Proceedings (RIAO'2000)*, pages 1261–1279 (vol 2), Collège de France, Paris, April 12–14, 2000.

III Kjell Lemström and Esko Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *Proceedings of the AISB'2000 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science*, pages 53–60, University of Birmingham, April 17–18, 2000.

IV Kjell Lemström and Lauri Hella. Approximate pattern matching is expressible in transitive closure logic. In *Proceedings of 15th annual IEEE Symposium on Logic in Computer Science (LICS'2000)*, pages 157–167, University of California, Santa Barbara, June 26–28, 2000.

V Kjell Lemström and Sami Perttu. SEMEX - an efficient music retrieval prototype. *First International Symposium on Music Information Retrieval (ISMIR'2000)*, University of Massachusetts, Plymouth, October 23–25, 2000. (to appear)

Chapter 1

Introduction

Combinatorial pattern matching methods for strings of symbols are used as essential components in a large variety of applications. Multimedia information retrieval is one of the main application areas. Until recently, however, practically all the research effort in this area has been directed at applications considering image and text collections, even though audio data should be equally important constituents of the multimedia domain. In the last few years activity in the domain of music retrieval has notably increased, which can be seen both in the increasing number of implementations (e.g., [25, 36, 46, 52]) and scientific publications (see e.g., [28]). Furthermore, the increasing activity will continue since music information retrieval (MIR, for short) issues are related to the forthcoming MPEG-7 standard[1], formally named "Multimedia Content Description Interface".

The possibility for automated access to different kinds of music collections should not only interest librarians who have been gathering thematic collections of musical incipits by hand, but also musicologists involved in the study of musical structure, and computer scientists trying to find efficient solutions for accessing musical databases and developing theories behind these solutions. Actually, it is easy to imagine fascinating possibilities for future MIR-related applications: by humming a short excerpt of a melody into a microphone, a CD player can be requested to play a particular piece of music or MPEG files can be downloaded from the Internet. Moreover, one may be able to order a song to be played by a jukebox by humming an excerpt of some melody to a personal mobile phone while sitting in a pub. Obviously, in a similar manner one could also download and change the ringing tone of a mobile phone. MIR techniques may even be used for solving judicial plagiarism cases [16].

[1] http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm.

There are two starting points for the development of MIR systems. The top-down development starts from music analytic ideas in the first place, and finally tries to find the pattern matching techniques that can be used to implement the decided specifications. In the bottom-up approach, the pattern matching techniques are modified to meet MIR requirements. The former is often used by musicologists, the latter by computer scientists.

The current state-of-the-art of MIR is not adequate for satisfactory implementations of the future applications described above. Should musical documents be retrieved in a musically meaningful way from a real, huge music database (e.g., from the Internet), the technique to be applied has to fulfill various, even somewhat contradictory, requirements. At least, the technique has to be very fast without requiring an excessive amount of memory, and it has to be able to take into account the most important music-specific features such as transposition invariance, polyphony, and context sensitivity. Though some of these requirements are satisfied by the current techniques, there is no technique or software system fulfilling them all. One reason for the lack of such techniques is that top-down designers often do not consider fast performance important enough and, on the other hand, bottom-up designers often overestimate the value of speed at the cost of music-specific properties that remain unimplemented in their solutions.

The goal of this thesis was to take further steps towards future MIR applications by providing useful tools and techniques. More precise goals were to consider and provide: digital representations for music; distances between such representations; a framework formalizing the distances; matching (retrieving) algorithms for MIR using the framework; bit-parallel techniques for the algorithms; and an MIR prototype implementing these all. With a deliberate collaboration between computer scientists, musicologists, and mathematicians, our system development approach was somewhere in between the two mentioned extremes.

The thesis comprises five papers and this summary, which is organized as follows. This chapter will be continued with an introduction to some musical terminology. The chapter also considers an MIR model by Salosaari and Järvelin [47], presents two symbolic encodings for music, gives a brief survey of related work, and states explicitly the contributions of this work. Chapter 2 introduces the underlying general string matching framework in detail. It also shows how the framework is applied to general pattern matching, and gives estimations for the theoretical complexity of such pattern matching.

Chapter 3 describes how the framework is modified to meet the requirements in music comparison and retrieval. It shows how the context of music can be taken into account, and introduces various distance measures and music alphabet reduction schemes. Furthermore, it specifies properly the MIR problem before describing solutions for the cases where the music databases to be searched are monophonic and polyphonic. Chapter 4 gives an overview to our SEMEX prototype. It also illustrates two implementations of our novel techniques, to the case of monophonic databases. Moreover, the chapter describes in detail a new efficient filtering algorithm for finding monophonic query patterns within polyphonic sources. Finally, Chapter 5 concludes the summary and gives some possible directions for future work.

1.1 Music Terminology

This section explains briefly various music-related terms that are used henceforth. For a more comprehensive exposition of music theory, psychoacoustics and psychomusicology the reader is referred to [42, p. 1–76].

Let us start by explaining the difference between the terms *tone*, *note* and *pitch*, which sometimes cause confusion. When a musical instrument is played, it evokes a *tone sensation* in a listener. Tone sensation is comprised of attributes *salience*, *pitch*, *timbre*, *onset time*, and *duration*, of which here the main interest will be in pitch; it is the attribute which may be used to compare two tones in terms of which of them is lower or higher. In contrast, tone refers to a periodic acoustical disturbance that can evoke pitch. The written instruction to play the tone is called a note.

The instruction to play a whole musical work can be given by a sequence of notes, which may be completely serial or may contain simultaneous pitches. In *monophonic music*, there is only one note at a time that is to be played (or sung), while in the case of *polyphony* there are several notes appearing simultaneously. In polyphonic music, however, there is usually a predefined *voicing* that implies that there are monophonic *musical lines* corresponding to particular instruments. In this summary, the terms *melody* and *query pattern* represent particular appearances of such musical lines (or excerpts of monophonic music).

The pitch distance between two tones (notes) is called an *interval*. The *contour* of the music can be represented by the three possible directions of the intervals, i.e., upwards, downwards, and a repeat. In western music, the smallest interval is called a *semitone*. An interval of 12 semitones, called an *octave*, is a very special one among all the intervals. It is rather interesting that not only (almost all)[2] humans, but also, e.g., rats perceive pitches to

[2]There are societies who do not prefer octaves [23].

have 'the same quality', if they are separated by an octave [5]. Secondly, in western music any two notes separated by an octave always share the same note names. Moreover, when considering the frequencies of such two tones, the frequency of the lower tone is always doubled by the upper tone. Therefore, combinations of them are *consonant*, which is not true for any other interval [42]. These facts suggest that usually it would be enough to assume *octave equivalence*[3], and map all the notes onto a range of one octave and consider only such reduced music scale.

The term *mode* refers to a standard pattern of 7 intervals within an octave (e.g., major or harmonic minor). A certain mode together with particular pitches establish *(musical) key* (*tonal context, tonal framework*). If a piece of music is largely comprised of notes from a specific key it is called *diatonic*. The scale of 12 semitones (within an octave) is called a *chromatic scale*.

If a melody is *transposed* from one key to another, it is perceived as similar, but depending on the direction of the transposition, lower or higher than the origin. The concept of *transposition invariance* ignores the key and does not make any distinction between a melody and its transposition.

1.2 Music Information Retrieval

The goal of this thesis is MIR via a musically feasible application of the general string matching techniques. The algorithms to be applied aim at locating the occurrences of a short monophonic query pattern by matching it against musical documents of a large music database, and retrieving such documents from the database according to the result of the matching.

There is only a quite small number of pitches (and durations) that are traditionally used in western music. Therefore monophonic music can be seen as a discrete linear string over a finite alphabet. Thus, a rough strategy for MIR would be to use the (approximate) pattern matching algorithms originally designed for keyword matching in texts, with some symbolic representation for the music. Though this is actually the strategy that is to be used, in its simplest form it does not suffice because it ignores many important musical properties.

In [47] Salosaari and Järvelin presented a model for an MIR system and listed its requirements. The list, which well reflects also our aims, is as follows:

[3]In music theory the precise definition of octave equivalence says that notes separated by octave(s) are assumed to have same *harmonic function*.

1. An MIR system must capture representative, usable and memorable features of music, and must allow content-based queries;
2. the system must be based on (easily available) digital representation;
3. it must be simple to use, and it must hide the 'richness' of music scores and interpretation of them;
4. it has to support approximate matching and retrieval;
5. retrieved documents should be ranked in an order of decreasing similarity.

Feature Extraction. The first step towards any kind of content-based retrieval is feature extraction. When dealing with music, the most prominent features to be used in a content-based query are pitch and duration information. Both of these are representative, usable and memorable. However, if only one of them is to be taken into account, it is claimed here that it should be the pitch information; in Western music it is more discriminative, because there are more natural possibilities for a sequence of n pitches than for a sequence of n durations (in US federal courts, even the copyright of music lies in melody [16]). Using only the pitch information loses some characteristics of the music, but it facilitates the modeling of polyphonic music. Moreover, when a piece of music has been adapted to another western music genre, often only its rhythmic pattern has been changed. Therefore, when retrieving musical documents based only on the pitch information, some kind of style independence can be achieved. This summary will deal with matching based only on pitch information.

Digital Representation. Let us deal with a special case of digital representation, that is, digital symbolic representation, such as MIDI [37] (Section 1.3.1). Symbolic representation can be also non-digital, such as traditional written notation using notes and staves (see Fig. 1.1). Actually, an ideal case for an MIR application would be a direct conversion from notated music to MIDI[4], since this way any normalization, such as quantization or time shifting, is not needed. At this stage of the project, we are not able to do such a conversion. Fortunately, the availability of MIDI files via the Internet is good[5]. There are not, however, any public test databases (or corpora) in general use. Such general corpora are widely used, for instance, among researchers studying matching techniques for texts and biosequences.

[4]This would be possible via Optical Music Recognition (OMR).
[5]Uploading these files, one should be very careful with the possible copyrights.

Mozart: piano sonata no. 6 (theme)

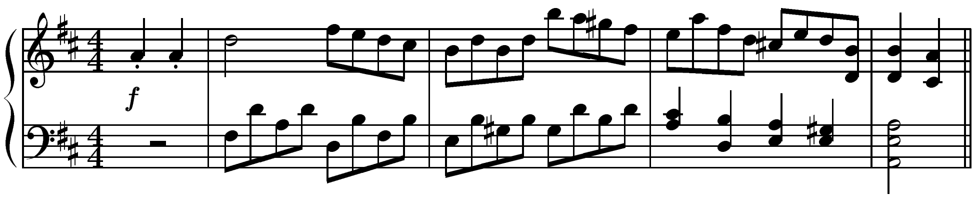


Figure 1.1: *Example of traditional western music notation.*

Usability. The third item in the list above is more demanding than it seems to be; it is not easy to design a simple user interface if the system becomes complex. Naturally, the appearance of the interface depends on whether the system is based on an existing database management system or it is self-implemented starting from the beginning. In any case, the system should offer different ways to give the query pattern, different tailored metrics, as well as different ways to interpret the meaning of approximation for different situations, etc.

Allowing Approximation. There are three ways to allow approximation when dealing with general string matching. The first is to use exact strings with approximate matching, the second uses 'generalized' strings with exact matching, and the third uses generalized strings with approximate matching techniques. Let us illustrate this by considering two strings PAKISTAN and VALENTIN. The strings can be presented by using an extra symbol "-" so that the matching symbols are aligned. This is shown in:

```
P-AK----ISTAN
-VA-LENTI---N.
```

Figure 1.2: *Example of a trace.*

Now this alignment, called a *trace*, suggests that by deleting P, K, S, T, and A from the first, and by inserting V, L, E, N, and T the latter string can be obtained (or vice versa). Since 10 operations is needed (equaling the number of "-"'s in the alignment), the trace suggests that the *dissimilarity* between the two strings would be 10, when allowing *insertion* and *deletion* editing operations[6]. It may be advantageous to generalize the strings, if the symbols in the strings are subject to some kind of systematic noise, e.g., the vowels and consonants are frequently replaced by other vowels and

[6]In this case, allowing also a *replacement* operation, the dissimilarity would be less.

consonants, respectively. In such a case, the vowels may be represented as "y"'s, and all the consonants as "x"'s. Now the two strings considered are both represented by the same generalized sequence: $xyxyxxyx$. Therefore, an exact matching technique on such generalized strings, considers them to be similar.

For us, this particular way to allow approximation is useful; it is a natural assumption that the query pattern will be given by humming or by playing an instrument. Such query patterns will probably contain faults typical of the way they are given. It is necessary to allow the user to make typical (minor) errors without penalizing them too much.

Ranking Documents. The last item of Salosaari and Järvelin's list is considered here as a triviality; since our framework gives a dissimilarity value between strings, it is capable of ranking the matched instances in any distance order.

Interactivity and Flexibility. Missing from their list, *interactivity* (the 'sufficient' efficiency) and *flexibility* of the matching method are here considered as very important MIR properties. Our choice to meet the interactivity is to use bit-parallelism, which makes use of the hardware word-level parallelism of bit-vector operations. According to our experiments, bit-parallel algorithms clearly exceed a scanning speed of 10 MB a second on Pentium III computers. Thus, assuming that the information of one note can be stored in one computer word and that the average length of a musical piece in the database is 100 notes[7], in theory such a database of more than 100, 000 musical documents can be scanned within one second. Not only the speed of this approach, but also its flexibility make it very useful.

1.3 Music Encodings

There are several symbolic representations available to represent music digitally, such as *DARMS*, *MIDI*, *SmOKe*, and *CHARM* [55]. The different encodings reflect the viewpoint of the developers and the selective understanding of the music. In principle, for our purposes a sufficient representation of the music would be encoding music as sequences of triples of attributes. Each triple $(a, b, c)_i$ corresponds to a note i, with the *onset time* (a), *offset time* (b), and the *pitch value* (c) of the note i. Naturally, in a polyphonic case it is possible that there are two such triples sharing any

[7] This is the case often with simple monophonic documents containing, e.g., folk music. Naturally there are musical documents containing significantly more notes than that.

two of the three possible attributes. However such a representation omits many kinds of information useful for some application; e.g., the voicing (or orchestration), the expression markings, and device control information.

Our SEMEX system supports two formats for music, i.e., MIDI [37] and our own symbolic representation IRP [I]. MIDI is probably the best known digital symbolic music representation, and it describes music accurately enough for most applications. As compared to MIDI, IRP totally ignores the control information included in MIDI but has a more accurate music description capacity for MIR applications. These two formats will now be overviewed briefly.

1.3.1 MIDI

MIDI (Musical Instrument Digital Interface) was originally developed for data communication between musical devices, such as synthesizers, in the early 1980's. MIDI operates at 31.25 kBaud using an asynchronous serial data word comprising of 1 *start bit*, 8 *data bits*, and 1 *stop bit*. The main concept of MIDI is a *message*, which is composed of a *status word* followed by *data words*. There are two types of messages, *channel* and *system messages*. For the thesis, all the relevant messages are channel messages, since both the *note on* and *note off messages* are channel messages. These messages are comprised of: a four bit *channel number*, which associates the message with one of the 16 MIDI channels; a seven bit *key number*, which represents the pitch (key number 60 equals the 'middle `c`'); and a seven bit *velocity number*. The duration of a note can be obtained by observing corresponding note on - note off pairs.

Later, a need for playing synthesizers automatically by devices called sequencers arose, which was the reason for designing a binary file format for storing symbolic data. For each message, the *Standard MIDI File (SMF)* format stores the MIDI messages together with a *time stamp*, i.e., the exact time of the message. The format allows also saving some extra information, such as tempo, names of tracks, time and key signatures. SMF format saves data in *chunks*, i.e., groups of bytes preceded by an ID and size. Due to the generic and flexible nature of the format, any device or algorithm can read or write an SMF file without losing the important data. Moreover, they can store their own data without confusing any other device or algorithm.

1.3.2 Inner Representation

For MIR purposes we developed our own representation, called *Inner RePresentation* (or *IRP*, for short) [I]. IRP files are plain ASCII text files so they can be edited using a standard text editor. An IRP file starts with an optional part storing information on the musical document. That part is followed by 17 tracks corresponding to the MIDI channels; one for all the textual information (the lyrics) appearing within the music, and 16 for the instruments. These latter tracks contain sequences of *events* comprising various musical features, some of which are directly available in MIDI and some of which will require some clever analyses of music.

1.4 Related Work

As early as 1967, Lincoln [35] presented some general requirements for using computers in music indexing. The ideas were at a very general level, such as: transcription by hand had to be avoided; and there had to be an effective input language for the music and economic means for printing the music. However, the first attempts to automate music pattern extraction[8] and music retrieval, where the musical sequences were allowed to contain various transformations and distortions, were first made in the early 1980's [18, 49].

In 1990, Mongeau and Sankoff [38] presented the first practical automated framework for comparing musical sequences. They adapted the edit distance measure by extending the usual set of editing operations containing insertion, deletion and replacement, to contain two novel operations, i.e., *fragmentation* and *consolidation*, as well (see Section 3.3). Their framework, however, suffers from the need to tune a large number of parameters, an operation that has to be done for each pattern-source pair at hand. A few years later, in 1993, Hawley mentioned in his PhD thesis [27] the possibility of making content-based queries over a MIDI database. Since then, interest in techniques for matching and retrieving melodies has greatly increased, which can be seen in the increasing volume of publications concerning the topic. The reader is referred to [28] for a summary of the current state-of-the-art.

The following gives an overview of the literature. The related studies have been classified in three categories: algorithms for monophonic databases; algorithms for polyphonic databases; and MIR systems. Appendix A gives a summary of the methods cited in this section with some extra information (such as the time complexities). The appendix of [V] summarizes current MIR systems.

[8] The term *pattern extraction* (pattern induction, pattern discovery) refers to techniques enabling the extraction of repeating (significant) patterns from strings.

1.4.1 Monophonic Databases

In [25], Ghias et al. presented the first experiments on query-by-humming. In order to allow error tolerance they adapted an idea already used in thematic catalogues: the chromatic alphabet was reduced so that only the contour of the music was considered. They experimented with a small database, containing 183 monophonic melodies. They found out that the number of musical documents, containing a match with a query pattern of 10–12 intervals, was around 10% of the total number of documents in the music database.

Since that study, different approaches for the problem have been suggested. Some of them have been based on music theoretic properties, but the actual implementation may have had little in common with the music theory itself, although the methods themselves may have been effective and efficient. The danger in overloading some well-known terms is that it may lead to peculiar impressions of music theory. For instance, Chou et. al. [10] used a kind of diatonic (not *functional*) chordal reduction of musical lines. Chen and Chen [9] retrieved music by using a rhythmic alphabet bearing a somewhat slight resemblance to general music theory.

In [13, 48], Shmulevich et al. presented a measure for musical sequences, that evaluates a distance combining both rhythm and interval information. Moreover, the interval difference part itself is a combination of *objective* and *perceptual errors*. The objective part is the difference between the two corresponding intervals, while the perceptual part incorporates musical context to the measure. In this case the considered musical context is the musical key, which is determined in a preprocessing phase by running a key-finding algorithm (which is refined in [57]).

1.4.2 Polyphonic Databases

When dealing with databases containing real music, the methods have to be able to deal with polyphony. It is still possible to try to cope with the methods above, but it would be too slow to generate all the possible musical lines out of a polyphonic sequence and match the query pattern against them all.

To find a heuristic capturing the (monophonic) musical line that best represents polyphonic music, Uitdenbogerd and Zobel [51] used a music psychological approach. They played first the polyphonic music itself, and then some musical lines obtained by different heuristics. The human listeners were asked to judge which of them performed best. They found the heuristic always choosing the highest note of a chord outperforming the others. We claim, however, that in many cases the highest voice does not even contain the melody (which a user most probably wants to use in a query). Furthermore, a musical line that the user is trying to find may be distributed across several distinct voices (consider, e.g., Elgar's *Cockaigne*). Thus, the capability to cope with all possible combinations of the polyphonic source, would be useful.

The first such algorithm was sketched by Crawford et al. [14]. The algorithm, which is adaptable to the case of transposition invariance, is based on the pattern recognition automaton by Aho and Corasick [1].

Dovey [19] presented a straightforward algorithm using a representation which considers only the note on messages of MIDI streams. The algorithm allows parameterized spacing (in time) between the consecutive elements within the musical sequences (see Fig. 1.3). It tries to find an occurrence of the given query pattern recursively in every possible location, with every allowed spacing, in the database. The algorithm is not transposition invariant, but allows the query pattern to be polyphonic, as well.

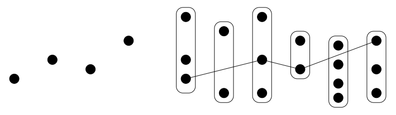


Figure 1.3: *An illustration of the effect of Dovey's spacing parameter t. An occurrence of the pattern, given to the left, has been found with the setting $t = 1$. Thus, is it allowable to skip at most one chord between two consecutive local matches.*

Recently, Holub et al. [29] presented an algorithm that uses the bit-parallel technique. They started by using the well-known shift-or algorithm [3] to find *multi-patterns* (several query patterns combined in a single query) within monophonic sources, and arrived at an algorithm capable of finding multi-patterns within polyphonic sources. However, they did not consider transposition invariance.

1.4.3 Other MIR Systems

MELDEX [36] is the New Zealand Digital Library's Web-based[9] melody retrieval system. Query-by-humming in the form of a URL given by the user is transformed into an internal representation. Melody retrieval is based on pitch and/or absolute duration. Pitch is matched either by chromatic intervals or by contour. A dynamic programming algorithm augmented

[9]http://www.nzdl.org/musiclib.

with consolidation and fragmentation operations [38], or a faster algorithm of rudimentary application of Wu and Manber's bit-parallel algorithm [56] is used for matching[10]. Recently, optical music recognition facility and text-based queries have been added to the system [4].

Themefinder [31], another Web-based[11] melodic search tool, is a closer successor to the thematic catalogue than most other systems: its database consists of hand-picked monophonic incipits from classical and folksong repertoires. Queries are based on pitch information and can be given in many forms, e.g., as intervals, scale degrees, pitch classes or contour. In addition, the search may be constrained by the name of the composer and a time signature. Only exact matching is available.

Pollastri's prototype [43] provides a pitch tracking module and several different representations both for pitch and rhythm information (including absolute pitches and chromatic intervals). The intervals are quantized to the range of -15 – 15, other values are assigned to an "undefined symbol" which has been included in their alphabet. The matching is based on Chang and Lampe's algorithm [8], which is found to be faster than Ukkonen's cutoff algorithm [53], in their experiences.

Uitdenbogerd and Zobel [52] performed MIR in a database of monophonic melodies extracted from polyphonic MIDI files (as mentioned above). They considered also different reductions of the melodies, such as contour. For matching, they conducted a comparison between four different similarity measures; they found the edit distance to perform best.

Melodiscov [46] supports various ways to represent pitch and durational information, and also query-by-humming, that they call "WYHIWYG" (What You Hum Is What You Get). Currently, retrieving is based on conventional edit-distance-based string matching, but they are embedding the FlExPat algorithm [44] in their system.

1.5 Contributions

The main new contributions of this thesis can be summarized as follows.

- A symbolic representation for music. Though there are various symbolic representations for music (see e.g., [55]), we introduce yet another. Our IRP representation [I] comprises exactly those attributes that we considered to have promising potential for current and future MIR applications. Since IRP representation is implemented as a text file, it is easy to edit it using standard text editors.

[10]They prefer the less efficient dynamic programming.

[11]http://www.themefinder.org/.

- A musical alphabet reduction scheme to enable error tolerance in query-by-humming. The *QPI (Quantized Partially overlapping Intervals) classification* [I] is a good compromise between the previously used two extremes, i.e., original intervals (no error tolerance, good discrimination ability) and music contour (good tolerance, poor discrimination) often used in MIR systems allowing query-by-humming (see e.g., [25, 36]). Moreover, our classification is based on findings in music cognition [40].

- Algorithms for retrieving monophonic query patterns in polyphonic music databases. With a few exceptions [14, 19], previous MIR algorithms have run only with monophonic databases. Though such databases may be carefully established so that they contain the melodic lines of the corresponding polyphonic music documents (a heuristic was proposed in [51]), it is useful to enable retrieving based on other, even distributed, music lines. In [II], we introduce algorithms for finding distributed occurrences of monophonic query patterns in polyphonic databases. These filtering algorithms are based on bit-parallelism and interval representation of the music. Independently of us, Holub et al. [29] presented similar ideas. Starting from a straightforward adaptation of shift-or algorithm [2] their main result was an algorithm capable of using multiple monophonic patterns within a single query (allowing proximity but not transposition invariance), while our main result is a transposition invariant algorithm for exact matching in static databases. Let us denote the length of the database (in chords) by n, and the maximum number of elements in a chord by q. By using a bit-parallel preprocessing phase taking $O(nq)$ time, the marking phase of the filtering algorithm is capable of finding candidate occurrences in time $O(n)$, assuming that the length of the query pattern is smaller than the size of the computer word. Then, the proper occurrences are found among the candidates by using a somewhat slower checking algorithm.

- A precise and general formalization using edit distance framework to enable consideration of context in the strings [III]. Albeit the context in music has been used in music comparison and retrieval (see e.g., [13, 45]), a detailed description of the adaptation of the string matching machinery has not been presented before. Using this framework, we generate novel transposition invariant MIR distance functions. By applying these functions on pitch sequences one is able to avoid the known drawbacks [7] of the frequently used straightforward application of conventional edit distance on interval sequences. Furthermore, we study the relations in applying edit distance measures on pitch and on interval sequences.

- Representing approximate pattern matching in strings by using transitive closure logics. Unlike the programming languages which are equivalent in their expressive power [26], separate mathematic logics are not. Therefore, finding the simplest logics to express pattern matching is interesting in itself. Moreover, to obtain expressibility in transitive closure logics, as we have done in [IV], may be particularly useful, because the relation algebra **A** by Date and Darwen [17], and the SQL:1999 standard [22] both comprise an explicit transitive closure operator. Therefore, the formalism can be used in developing a query language for a database management system[12]. This approach may be useful with very large music databases, where *indexing approaches* require excessive amount of space and *online approaches* become too slow to use in practice

  Further, the expressibility of approximate pattern matching in transitive closure logics leads to the possibility of assessing the complexity of general pattern matching. In [IV], we show that k mismatches (and k differences when k is fixed) is LOGSPACE complete when the model is equipped with a successor relation. In another setting, k mismatches becomes expressible already in the complexity class ThC^0.

- A comprehensive application of bit-parallelism in MIR. Chapter 4 and [V] describe implementations of our novel ideas, such as QPI classification, new distance metrics given by our framework, and algorithms for polyphonic databases, by using the flexible bit-parallelism approach. The implementations preserve the original time complexity of the approach. Before us (and Holub et al. [29]) the approach had nearly passed unnoticed among MIR community. Only McNab et al. [36] had proposed a rudimentary application of Wu and Manber's algorithm [56].

- A prototype of MIR software system. The SEMEX system, that implements most of the ideas presented in the thesis, is described in [V]. By applying the fast bit-parallel algorithms for retrieving, it typically achieves a retrieving speed of nearly 10 million notes a second, on current computers.

[12]Nykänen has used such an approach by using modal logic for querying string databases [41].

Chapter 2

Approximate String Matching

The music retrieval task is to find good enough occurrences of a given query pattern in a music database. To be able to search such occurrences, first a distance measure has to be defined. Then the measure is to be used in a matching process together with a proximity threshold value. The distance measure frequently used in approximate string matching is so-called edit distance. It gives the minimal number of *local transformations* (or *editing operations*) needed to transform one string (or sequence)[1] into another. Algorithms evaluating edit distances are mainly based on the algorithmic method called *dynamic programming*.

This chapter introduces a framework for measuring distances between sequences using edit distance, and shows how the framework is applied for pattern matching in sequences. The chapter also presents a particular, efficient way to implement dynamic programming, i.e., to use bit-parallelism. Furthermore, the chapter introduces the concept of filtering, and gives theoretical results on the complexity of the dynamic-programming-based pattern matching. The introduced framework is subsequently refined in Chapter 3 in order to meet MIR requirements.

Before going into those details, let us present some notation to be used henceforth in the summary.

Notation. Let Σ be a finite set of symbols, called an *alphabet*. Then any $A = (a_1, a_2, \ldots, a_m)$ where each a_i is a symbol in Σ, is a *sequence* over Σ. Usually we write $A = a_1 \cdots a_m$. The *length* of A is $|A| = m$. The sequence of length 0 is called the *empty sequence* and denoted λ. The set of sequences of length i over Σ is denoted by Σ^i, and the set of all sequences over Σ by

[1]Henceforth, we prefer to use the term sequence due to the ambiguity of the term string in the context of music.

Σ^*. If a sequence A is of form $A = \beta\alpha\gamma$, where $\alpha, \beta, \gamma \in \Sigma^*$, we say that α is a *factor* (substring) of A. Furthermore, β is called a *prefix* of A, and γ a *suffix* of A. A sequence A' is a *subsequence* of A if it can be obtained from A by deleting zero or more symbols, i.e., $A' = a_{i_1} a_{i_2} \cdots a_{i_m}$, where $i_1 \ldots i_m$ is an increasing sequence of indices in A.

2.1 Edit Distance

A conventional way to measure the approximate similarity between two sequences $A = a_1 \cdots a_m$ and $B = b_1 \cdots b_n$, is to calculate local transformations. Usually the considered *local transformations* are the following:

- replacement: $a_i \rightarrow b_j$;
- insertion: $\lambda \rightarrow b_j$; and
- deletion: $a_i \rightarrow \lambda$.

The differences appearing in the considered two sequences can be viewed differently, e.g., one replacement can be viewed as one insertion and one deletion. Therefore, it is natural to observe the *minimum number* of such differences. The *edit distance* between sequences A and B, denoted $D(A, B)$, is the minimum number of local transformations required to transform A into B. Actually, $D(A, B)$ is a *metric*, i.e.,

i) $D(A, B) \geq 0$ ($D(A, B) = 0$ iff $A = B$);

ii) $D(A, B) = D(B, A)$; and

iii) $D(A, C) \leq D(A, B) + D(B, C)$.

2.1.1 The General Scheme

The underlying general framework of the edit distance and its many variations is as follows; c.f. [53].

To define a distance between sequences in Σ^*, one should first fix the set of local transformations $T \subseteq \Sigma^* \times \Sigma^*$ and a non-negative valued *cost function* w that gives for each transformation t in T a cost $w(t)$. Each t in T is a pair of sequences $t = (\alpha, \beta)$. Observing such a t as a rewriting rule, suggests another notation for t, $\alpha \rightarrow \beta$ (α is replaced by β within a sequence containing α), which we will use below. For convenience, if $\alpha \rightarrow \beta \notin T$, then $w(\alpha \rightarrow \beta) = \infty$.

The definition of the distance is based on the concept of *trace* (or alignment), which gives a correspondence between two sequences[2]. Formally, a trace between two sequences A and B in Σ^*, is formed by splitting A and B into equally many factors:

$$\tau = (\alpha_1, \alpha_2, \ldots, \alpha_p; \beta_1, \beta_2, \ldots, \beta_p),$$

where $A = \alpha_1 \alpha_2 \cdots \alpha_p$, and $B = \beta_1 \beta_2 \cdots \beta_p$, and each α_i, β_i (but not both) may be an empty sequence over Σ. Thus, sequence B can be obtained from A by local transformation steps $\alpha_1 \to \beta_1$, $\alpha_2 \to \beta_2, \ldots, \alpha_p \to \beta_p$.

The cost of the trace τ is $w(\tau) = w(\alpha_1 \to \beta_1) + \cdots + w(\alpha_p \to \beta_p)$. As stated before, the distance between A and B, denoted $D_{T,w}(A, B)$, is defined as the minimum cost over all possible traces.

Functions D_L and D_H. As an example of a distance that can be given by the scheme presented above, consider the familiar *unit-cost edit distance* (*Levenshtein distance*), where the local transformations are of the forms $a \to b$ (replacement), $a \to \lambda$ (deletion), and $\lambda \to a$ (insertion), where $a, b \in \Sigma$. The costs are given as $w(a \to a) = 0$ for all a, $w(a \to b) = 1$ for all $a \neq b$, and $w(a \to \lambda) = w(\lambda \to a) = 1$ for all a. We denote this edit distance function as D_L (in the original articles, we have also denoted this distance as $D(A, B)$).

The *Hamming distance*, D_H, is obtained exactly as D_L but the only allowed local transformations are of form $a \to b$ where a and b are any members of Σ, with cost $w(a \to a) = 0$ and $w(a \to b) = 1$, for $a \neq b$.

Evaluation of $D_{T,w}(A, B)$, $D_L(A, B)$, and $D_H(A, B)$. Distances $D_{T,w}(A, B)$ can be evaluated by using dynamic programming. Such a procedure tabulates the distances between the prefixes of A and B: each cell d_{ij} of the distance table (d_{ij}) stores the distance between $a_1 \cdots a_i$ and $b_1 \cdots b_j$ ($0 \leq i \leq m$, $0 \leq j \leq n$). Then, (d_{ij}) can be evaluated by proceeding row-by-row or column-by-column and using the recurrence:

$$\begin{cases} d_{00} = 0 \\ d_{ij} = \min\{d_{r-1,s-1} + w(a_r \cdots a_i \to b_s \cdots b_j) \mid \\ \qquad (a_r \cdots a_i \to b_s \cdots b_j) \in T\}. \end{cases} \tag{2.1}$$

Finally, d_{mn} equals the distance $D_{T,w}(A, B)$.

The evaluation of $D_{T,w}(A, B)$ in this way may in the worst case take time $O(m^2 n^2)$. However, in practice, T is often very sparse which can

[2]Example (1.2) represents a trace between two sequences.

be utilized to speed-up the dynamic programming. For example, the edit distance $D_L(A,B)$ can be evaluated in time $O(mn)$ from the recurrence

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min \begin{cases} d_{i-1,j}+1 \\ d_{i,j-1}+1 \\ d_{i-1,j-1} + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1). \end{cases} \end{aligned}$$

When evaluating $D_H(A,B)$, the latter equation above is replaced by

$$d_{ij} = d_{i-1,j-1} + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1).$$

The distance table (d_{ij}) can be viewed as a weighted graph. The cells (i.e., d_{ij}'s) form the nodes of the graph, while any rewriting rule $a_r \cdots a_i \to b_s \cdots b_j \in T$ forms a weighted arc from node $d_{r-1,s-1}$ to node d_{ij}. The arc has weight $w(a_r \cdots a_i \to b_s \cdots b_j)$, similarly the path from d_{00} to d_{ij} has the total weight equaling d_{ij}. The path from d_{00} to d_{mn} having the smallest total weight, is called a *minimizing path*. We illustrate distance evaluation and minimizing path of $D_L(aaabbb, bbaaab)$, in Fig. 2.1 a). As can be seen in the figure, it is possible that a minimizing path is not unique.

2.1.2 The Approximate Pattern Matching Problem

Let us consider two particular sequences, $P = p_1 \cdots p_m$ and $S = s_1 \cdots s_n$, called the *pattern* and the *text* (or *source*), respectively. Typically the pattern is very short as compared to the length of text, i.e., $m \ll n$. Let k be a threshold value, $k \geq 0$, that will be used to control the accuracy of the pattern matching. Then the problem of approximate pattern matching is to locate such factors P' within S, that are k *similar* to P, i.e., $D_{T,w}(P,P') \leq k$. The pattern matching problems associated with D_L and D_H are called k *differences* and k *mismatches*, respectively.

The evaluation of approximate pattern matching is analogous to that of approximate string matching, with few exceptions. The first row of the table (d_{ij}) is initialized with zero values: $d_{0j} = 0$, for $0 \leq j \leq n$. Moreover, the approximate occurrences are obtained by observing the lowermost row of the table. The value of cell d_{mt} in the bottom row represents the minimum number of allowed local transformations used to transform P to any $s_r \cdots s_t$, for $1 \leq r \leq t+1$. Let $s_{r'} \cdots s_t$ be the factor corresponding to the value of the cell d_{mt}. If $d_{mt} \leq k$ a match starting at r' is found, and the minimizing path traverses from $d_{0,r'-1}$ to d_{mt} (see Fig. 2.1 b) for an illustration of 0 differences, i.e. the exact case).

In [53], Ukkonen stated the following useful property of (d_{ij}) table.

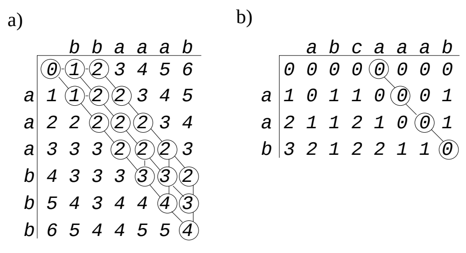


Figure 2.1: *Distance evaluation and minimizing path. The encircled cells belong to minimizing paths, and the weights of the arcs are either 0 or 1. Figure a) implies that $D_L(aaabbb, bbaaab) = 4$, b) that $P = aab$ matches exactly $s_5s_6s_7$.*

Theorem 2.1 A table (d_{ij}) applying D_H or D_L distance has the property that the difference between adjacent cells (in vertical, horizontal or diagonal directions) are -1, 0, or 1. Furthermore, the value along each diagonal does not decrease.

Following from that fact, Ukkonen proposed a *cutoff algorithm*, which processes such table (d_{ij}) column-by-column. At each column j, the algorithm records the last row i, such that $d_{ij} \leq k$. It follows from Theorem 2.1, that it is enough to evaluate the values of cells $d_{0,j+1}, \ldots, d_{i+1,j+1}$ in the next column, since only they can be assigned with a value of at most k. In literature, the cutoff algorithm is frequently used as a comparison algorithm, as also we do in [V].

2.2 Bit-Parallelism

The table (d_{ij}) can be evaluated more efficiently by using bit-parallelism, by putting certain restrictions on the set T of local transformations and on the cost function w. For instance, D_L and D_H distances (including the exact matching as a special case) are implementable by using bit-parallelism[3]. To represent the (d_{ij}) table the conventional implementation of dynamic programming uses one memory location for each cell, while a bit-parallel implementation uses a few, even one, computer word for a whole column of (d_{ij}). Then a column-by-column evaluation is simulated, and the cells in the current column are manipulated in parallel by using word-level bit-vector operations of the hardware.

[3]It is not necessary, however, to use such simple forms of T and w. For instance, the local transformations can be applied to generalized character classes, and the weights can be non-uniformly assigned to small integer values [2, 56].

One of the first algorithms of the bit-parallel family, the *shift-or algorithm*, was presented for exact matching by Baeza-Yates and Gonnet [2]. In presenting their algorithm, let us assume that the size of the pattern, denoted m, is no more than the size of the used computer word, denoted W, i.e., $m \leq W$. The algorithm uses an external bit-array $\mathtt{T}$ [4] of size $n \times m$. For each character c_l appearing in the pattern, there is a column $\mathtt{T}[l]$ (of m bits) such that $\mathtt{T}[l].r = 0$ if $p_r = c_l$, otherwise $\mathtt{T}[l].r = 1$. Thus, $\mathtt{T}[l]$ gives the indices of the pattern that contains c_l.

A bit-vector $\mathtt{E}$ of size m represents the current column of (d_{ij}) table. Let us denote by $\mathtt{E}_j$ the value of vector $\mathtt{E}$ when j symbols of S has been processed. $\mathtt{E}_j.i$ is set to 0 iff $p_1 \cdots p_i = s_{j-i+1} \cdots s_j$. Thus, $\mathtt{E}_j$ gives all matches of the prefixes of P ending at j in S. The transition from $\mathtt{E}_j$ to $\mathtt{E}_{j+1}$ is performed by using the recurrence:

$$\mathtt{E}_{j+1}.i = \begin{cases} 0, & \text{if } \mathtt{E}_j.(i-1) = 0 \text{ and } p_i = s_{j+1}, \\ 1, & \text{otherwise.} \end{cases}$$

In effect, this extends any partial match $p_1 \cdots p_{i-1}$ ending at j provided that the corresponding symbols p_i and s_{j+1} match. In such a case a 0-diagonal is continued one step further. Whenever $\mathtt{E}_{j+1}.m = 0$, a match starting at $j - m + 2$ is found.

The transition given by the recurrence above can be implemented very efficiently by using one `shiftleft`[5] applied on bitvector `E`, and one bitwise `or` operation between `E` and a column of `T` corresponding to the symbol of S at hand. Thus, the shift-or algorithm becomes as follows.

Algorithm 2.2 (shift-or)

```
begin
1. for each $a \in \Sigma$ do $\mathtt{T}[a] := 2^m - 1$;
2. for $i := 1$ to $m$ do $\mathtt{T}[p_i] := \mathtt{T}[p_i] - 2^{i-1}$;
3. $\mathtt{E} := 2^m - 1$;
4. for $j := 1$ to $n$ do
5.    $\mathtt{E} :=$ shiftleft(E)|(T[$s_j$]); /* | denotes bitwise or-operation */
6.    if $\mathtt{E}.m = 0$ then write($j$); /* the end position of the occurrence */
end
```

Denoting by W the size of the computer word, the algorithm 2.2 runs in time $O(n\lceil \frac{m}{W} \rceil)$[6] (more precisely, in time $O(n\lceil \frac{m}{W} \rceil + m + |\Sigma|)$, i.e., time that

[4]Please note the difference between $\mathtt{T}$ and T. The typewriter style of the former will be used with bit-arrays, the latter denotes the set of local transformations.

[5]Here we assume 0-filled shifts as with C language.

[6]$\lceil\rceil$ is the *ceiling function*.

T:

	a	b	c
a	0	1	1
a	0	1	1
b	1	0	1

	E	shiftleft(E)	T[a]	E	shiftleft(E)	T[b]	E	shiftleft(E)	T[c]	E	shiftleft(E)	T[a]	E	shiftleft(E)	T[a]	E	shiftleft(E)	T[a]	E	shiftleft(E)	T[b]	E
a	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
a	1	1	0	1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1
b	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	⓪

Figure 2.2: *The shift-or algorithm. Note that the shifting is done downwards in the figure, and that one occurrence is found (encircled).*

is linear in n with the presumption of $m \leq W$. In Fig. 2.2, we illustrate its computation, when $S = abcaaab$ and $P = aab$ (cf. Fig. 2.1 b).

Baeza-Yates and Gonnet showed also how to perform k mismatches queries by using $\log_2 k$ bits for every d_{ij}. This idea was subsequently refined to the k differences problem by Wu and Manber [56]. By maintaining k copies of the bit vector `E` their algorithm runs in time $O(kn\lceil \frac{m}{W} \rceil)$.

Recently Myers [39] presented a fast bit-parallel implementation of the dynamic programming algorithm for the k differences problem. It runs in time $O(\lceil \frac{m}{W} \rceil n)$. The performance of the algorithm rests on Theorem 2.1: since the difference between adjacent vertical cells is always -1, 0, or 1, it is possible to represent the current column by using only two bit masks. One stores positive and the other negative changes along the adjacent cells.

2.3 Filtering Techniques

Filtering techniques work in two phases. Usually the first phase, called *marking*, is very fast. It aims at finding starting positions of potential occurrences, called *candidates*. The output of the first phase is then subsequently given as an input to the second phase, called *checking*. In this phase, each candidate is checked whether it is a proper occurrence.

In the thesis, we will only consider filters having the property that every proper occurrence is also a candidate, in other words, the filter will not miss any proper occurrence. Typically, a filtering method becomes efficient when the marking phase is fast, and the number of found candidates is not too large as compared to the number of proper occurrences. The checking phase might be somewhat slower.

For instance, it is possible to calculate the distribution of the symbols appearing in the source in advance. When a pattern has been given, it can also be analyzed and the symbol in the pattern having the lowest probability can be picked up. Now, the occurrences of that symbol in the source can easily be found in linear time (by using linear search). The indices of those occurrences are stored as candidate locations, and the area around the candidate locations are then subsequently checked, e.g., by using the dynamic programming algorithm.

2.4 On the Complexity of Pattern Matching in Strings

In [IV], we present logical formalisms for expressing k mismatches and k differences pattern matching. In the formalism, the minimizing paths are expressed by using a formula in an extension of first-order logic [21]. When k is given as a part of the input, the problems are expressible by using *deterministic transitive closure logic* and *transitive closure logic* [21, 30], respectively. It is likely, that in the general case k differences is not expressible in deterministic transitive closure logic[7]. However, if k is fixed, we are able to express k differences in deterministic transitive closure logic, as well.

The formalism can be utilized in two ways. First, expressing the problems in a logic is a step towards constructing a string database query language (see e.g., [41]). Such query languages are usually based on first-order logic. Recently Date and Darwen [17] proposed a new relation algebra, called **A**, which departs from the Codd's original relational algebra [11, 12] in a few respects. The most interesting, from our point of view, is that **A**, as well as SQL:1999 [22], includes an explicit transitive closure operator. Thus, by using them, our formalism can straightforwardly be put into use.

Second, the formalism can be used to estimate the complexity of the pattern matching problems. Our most interesting results for the complexity matter arose with a model theoretic setting of the problem, where the instances are models encoding the two sequences P and S. In this case, the natural order of the sequences is not directly available in a Turing machine computation, but it has to be computed from the input model. It turns out that if the model is equipped with a successor relation, k mismatches (and k differences when k is fixed) is LOGSPACE complete. However, the result seems to be sensitive to the way it is formulated; if the successor relation is replaced by linear order, k mismatches becomes expressible in first-order logic with counting quantifiers, and hence, the problem is in the complexity class ThC^0 [8]. This apparent contradiction in the complexity is avoided if LOGSPACE collapses to ThC^0, which is an open question.

[7]Solving this question in the affirmative is at least as hard as separating two complexity classes LOGSPACE and NLOGSPACE, which is an open problem in complexity theory.

[8]The class of problems computable by uniform sequences of polynomial size constant depth circuits with threshold gates.

Chapter 3

Tuning the Framework for Music Retrieval

In this chapter, we present how we have tuned the general framework described in Chapter 2, to be used in music comparison and retrieval. The first sections are valid for both of the tasks, but starting at the Section 3.6, the rest of the chapter is devoted to music retrieval only.

3.1 Encoding the Music

In our simplified representation for monophonic music, the pitch levels (or intervals) are represented as small integers $0, \ldots, r$. These integers form our alphabet $\Sigma = \{0, 1, \ldots, r\}$. In the thesis, four values for r are of particular interest; $r = 2$ (representing the contour), $r = 10$ (reduces notes into the QPI classification [I]), $r = 11$ (reduces notes into an octave), and $r = 127$ (as in MIDI).

We represent polyphonic music by sequences comprising sets of integers. Each set contains pitch information of notes having their onset simultaneously. This representation will be described more precisely in Section 3.6.

For monophonic music, we have also considered representations containing temporal information. In [I], we represent music by using *two-dimensional relative encoding*. Letting S be a musical sequence, $S = s_1 \cdots s_n$, now each s_i consists of a pair of components. The first component gives the interval between two successive notes, while the second gives the relative duration of the latter note. In [34], we experimented with a representation which combines pitch and duration information in a single value, and which achieves invariances both under transposition and under different tempi. These representations, however, are not considered any further in this summary.

3.1.1 Transposition Invariant Encoding

Two equally long sequences $A = a_1 \cdots a_m$ and $A' = a'_1 \cdots a'_m$ in Σ are *transpositions* of each other if there is an integer c such that $a'_1 = a_1 + c, a'_2 = a_2 + c, \ldots$, and $a'_m = a_m + c$[1]. Then we write $A' = A + c$. The *interval representation* of a sequence $A = a_1 \cdots a_m$ in Σ^* is a sequence:

$$\overline{A} = (a_2 - a_1, a_3 - a_2, \ldots, a_m - a_{m-1}) = \overline{a}_1 \cdots \overline{a}_{m-1}[2].$$

Each symbol $\overline{a}_i$ in $\overline{A}$ is a difference between two integers in Σ. Hence $\overline{A}$ is a sequence in $\overline{\Sigma}^*$ where $\overline{\Sigma} = \{a - b \,|\, a, b \in \Sigma\}$ is the *interval alphabet*.

The crucial property of the interval representation is that it is *transposition invariant*. This means, that if A and A' are transpositions of each other, then, obviously, they have the same interval representation, i.e.,

$$\overline{A} = \overline{A'}.$$

In music comparison and retrieval, it is often natural to use transposition invariant measures for the distance between sequences. Formally, a sequence distance function D is transposition invariant if

$$D(A, B) = D(A + a, B + b)$$

for any sequences A, B in Σ^*, and any possible a, b in $\overline{\Sigma}$ such that the symbols of $A + a$ and $B + b$ are in Σ.

Absolute vs. Transposition Invariant Distances. The sequence distance functions of the considered framework can be applied as such for comparing sequences both in Σ^* ('absolute' sequences) and in $\overline{\Sigma}^*$ ('relative' sequences). The functions induce two distances in this way. We say that $D_L(A, B)$ is the *absolute* edit distance and $D_L(\overline{A}, \overline{B})$ is the *transposition invariant* edit distance between A and B.

Let $A = a_1 \cdots a_m$. The strength of the interval encoding is that if B is a version of A, otherwise similar but containing one *modulation*, i.e., $B = (a_1, \ldots, a_{l-1}, a_l + c, \ldots, a_m + c)$ (for $1 \le l \le m$, and where c is an integer) then $D_L(\overline{A}, \overline{B}) = D_H(\overline{A}, \overline{B}) \in \{0, 1\}$[3]. Though one single modulation in the original sequences can affect any number of elements, the distance of the interval encoded versions stays small.

In [III], we show that the relations between the corresponding absolute and transposition invariant distances are rather loose.

[1]Note that as our symbols in Σ are integers, we can also use their normal arithmetic operations, such as addition and subtraction.

[2]Sometimes the front of the interval sequence is padded with the first element of the original sequence A to achieve a *lossless* representation.

[3]The value 0 represents the special case of modulation, i.e., transposition ($l = 1$).

3.2 Allowing Context Sensitivity

In the distance scheme described in Chapter 2 the cost $w(\alpha \to \beta)$ of each local transformation was independent of the context in which α and β occur in A and B; transformation $\alpha \to \beta$ had always the same cost. When comparing musical sequences, the use of context-sensitive cost functions may sometimes be useful. The cost of $\alpha \to \beta$ should depend on the context around α and β.

In [III], we formalize this by including the context in the definition of the function w. There are several possibilities for limiting the size of the context; we suggest a parameterized limitation. Assume that A can be written as $A = \mu\varphi\alpha\varphi'\mu'$ where $|\varphi| = u$ and $|\varphi'| = v$. Then α has (u, v) *context* (φ, φ'). It is said that w is a *cost function with* (u, v) *context* if w is an integer-valued function defined on $T \times \Sigma^u \times \Sigma^v \times \Sigma^u \times \Sigma^v$. Now the cost function w takes into account the (u, v) context of each local rule when evaluating the cost of a trace; if $\alpha \to \beta$ occurs in a trace such that the (u, v) context of α and β is (φ, φ') and (ϱ, ϱ'), respectively, then $w(\alpha \to \beta, \varphi, \varphi', \varrho, \varrho')$ is the cost associated with this particular occurrence of $\alpha \to \beta$.

The (u, v) context of each candidate transformation $\alpha \to \beta$ can be obtained during dynamic programming when evaluating each d_{ij}. The running time might get slower but the overall architecture of the evaluation process remains the same.

3.3 Editing Operations for Music Comparison and Retrieval

It would be impossible to define a universal distance function for music comparison and retrieval due to the diverse musical cultures, styles, etc. Therefore, in the literature various editing operations have been suggested for music comparison and retrieval. Actually, it would be a good idea to provide a set of possible operations of which a suitable subset for the problem at hand is selected.

According to Bainbridge et al. [4] the Levenshtein distance presents a compact set of operations with which two musical sequences can be compared, each of the operations having a plausible musical interpretation. However, applying edit distance to sequences of intervals is not plausible in some respects; deleting or inserting a note in between a sequence would result to a modulation starting from the location where the edit operation took place (see Fig. 3.1, modulating insertion and deletion). Therefore, in the presence of some noise during the storing process the original musical sequence will be lost and it cannot be regained. Furthermore, editing one note has an effect on two intervals[4], which makes the standard operations on notes more costly in the interval representation [7].

In Fig. 3.1 we illustrate the conventional editing operations together with some extra operations suggested for MIR. Mongeau and Sankoff's *consolidation* combines a sequence of notes into one whose duration is the sum of the original durations, while *fragmentation* does the reverse [38].

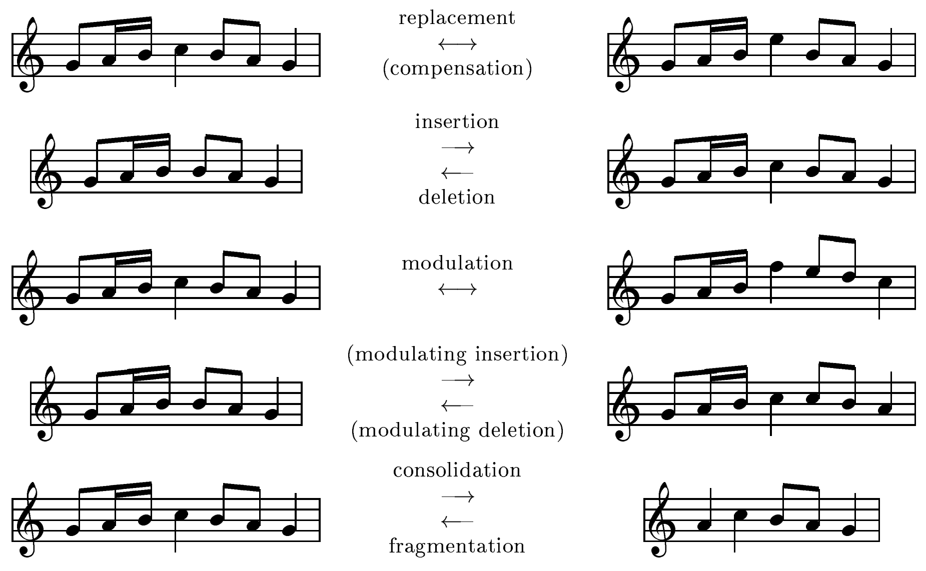


Figure 3.1: *Editing operations applied to MIR. The parenthesized operations correspond to the editing operations used with interval representation.*

3.4 Edit Distances for Music Comparison and Retrieval

We noticed in Section 3.1.1 that each distance function for absolute sequences also gives a transposition invariant distance. This is achieved simply by at first converting the sequences into interval encoding. In the following, we complement this by observing that an explicit conversion is not necessary; we present several distances, introduced in [III], where transpo-

[4]To achieve the 'replacement effect' with interval representation, the *compensation operation* [III] can be used.

sition invariance is obtained by using context-sensitive cost functions. Furthermore, since we do not use interval representation, the aforementioned (Section 3.3) shortcomings can be avoided. Note that in the following cases editing one note will still have an effect on two intervals. Nevertheless, because the intervals are calculated only when accessed, the original information is preserved. Therefore other methods, that take the edition into account only once, can be used, e.g., in a post-processing phase that fine-tunes the actual distances of the found occurrences.

3.4.1 $\overline{D}_H$ and $\overline{D}_L$ Distances

Let us modify the distance D_L (the distance D_H is modified in the same way). The *transposition invariant unit-cost edit distance* $\overline{D}_L(A,B)$ uses similar local transformations as the distance $D_L(A,B)$, and the costs for insertion and deletion are preserved: $\overline{w}(a \to \lambda) = \overline{w}(\lambda \to a) = 1$. However, the cost for $a \to b$ will now be context-sensitive. Let a' be the symbol of A just before a and b' the symbol of B just before b, that is, the $(1,0)$ contexts of a and b are (a', λ) and (b', λ). Then define

$$\overline{w}(a \to b, a', \lambda, b', \lambda) = \begin{cases} 0, \text{if } a - a' = b - b' \text{ or } a' \text{ or } b' \text{ is missing,} \\ 1, \text{if } a - a' \neq b - b'. \end{cases}$$

The recurrence for evaluating the distance table (d_{ij}) for $\overline{D}_L(A,B)$ becomes

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (\text{if } a_i - a_{i-1} = b_j - b_{j-1} \text{then } 0 \text{ else } 1). \end{cases} \end{aligned}$$

Functions $\overline{D}_L$ and $\overline{D}_H$ are transposition invariant distance functions. Furthermore, because the distance tables for them are of the same general form as the table for the standard unit-cost edit distance, the standard tricks to speed up the implementation can be applied.

3.4.2 D_N Distance

The function $D_N(A,B)$ achieves the advantages of both the relative and absolute encodings by always using the approach which locally results in a better match. In this measure the weight $\overline{w}$ is defined as:

$$\begin{aligned} \overline{w}(a \to \lambda) = \overline{w}(\lambda \to a) &= 1 \\ \overline{w}(a \to b, a', \lambda, b', \lambda) &= \begin{cases} 0, \text{if } a - a' = b - b' \text{ or } a = b \text{ or} \\ \quad a' \text{ or } b' \text{ is missing,} \\ 1, \text{otherwise.} \end{cases} \end{aligned}$$

It is guaranteed that $D_N(A,B) \leq D_L(A,B)$ and $D_N(A,B) \leq \overline{D}_L(A,B)$, for all A and B. On the other hand, D_N is not actually a transposition invariant distance function, since it allows the sequence to 'realign' itself to the correct absolute level without a cost.

3.4.3 D_{LCTS} Distance

In [III], we generalize the concept of the *longest common subsequence* (see e.g., [15]) to be used for music comparison and retrieval. The *longest common transposition invariant subsequence* of two sequences A and B in Σ^*, $LCTS(A,B)$, can be seen in musical terms as the longest common melody hidden in both A and B. $LCTS(A,B)$ is obtained by deleting the smallest number of elements from A and B so that the remaining two sequences are transpositions of each other. The deleted elements may be, e.g., decorations in the two variations A and B of the same melody C (see Fig. 3.2).

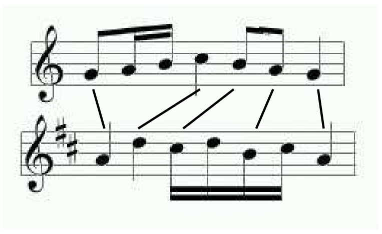


Figure 3.2: *Two musical lines that are basically just transpositions of each other, containing only different musical decorations. The corresponding parts are connected by lines.*

Here we are not interested in $LCTS(A,B)$ itself, but the distance function $D_{LCTS}(A,B)$ that is used to evaluate the length of $LCTS(A,B)$. Also $D_{LCTS}(A,B)$ is given by our framework: Any rule $\alpha \rightarrow \beta$ where α, β are *non-empty* sequences in Σ^*, is a local transformation, and the associated cost function w_{LCTS} has $(1,0)$ context. Let a be the last symbol of α and b the last symbol of β, and let the $(1,0)$ context of α and β be (a',λ) and (b',λ), respectively. Moreover, let $k = |\alpha|$ and $l = |\beta|$, then

$$w_{LCTS}(\alpha \rightarrow \beta, a', \lambda, b', \lambda) = \begin{cases} k-1+l-1, \text{if } a-a' = b-b' \\ k+l, \text{if } a-a' \neq b-b', \end{cases} \tag{3.1}$$

The recurrence for tabulating (d_{ij}) to get $D_{LCTS}(A,B)$ is

$$\begin{aligned} d_{00} &= 0 \\ d_{ij} &= \min_{1 \leq k \leq i; 1 \leq l \leq j} \{d_{i-k,j-l} + w_{LCTS}(\alpha \rightarrow \beta, a_{i-k}, \lambda, b_{j-l}, \lambda)\}, \end{aligned}$$

where $\alpha = a_{i-k+1} \cdots a_i$ and $\beta = b_{j-l+1} \cdots b_j$. Evaluating $D_{LCTS}(A, B)$ can be done in time $O(mn)$, by observing that $LCTS(A, B)$ must be equal to a $LCS(A, B + c)$ for some suitably selected constant c [III].

Consider again the weighting function (3.1). Applying the rule $\alpha \to \beta$, when $a - a' = b - b'$ and $|\alpha| > |\beta|$ hold, can be viewed as a *generalized interval consolidation*, where $|\alpha|$ intervals are coupled into $|\beta|$ intervals (if $|\beta| > |\alpha|$, a *generalized interval fragmentation*). Moreover, using the particular weighting function enables an efficient evaluation of such operations.

3.5 Reducing the Music Alphabet

In an MIR system, it is a natural option that the query pattern can be given by humming. Therefore, a pattern comprised of pitch values will probably contain faults caused by inaccurate humming. Already a minor inaccuracy may lead to a fault if the *pitch estimator*[5] uses fixed tuning, i.e., it does not adapt to the tuning of the query pattern (for further information on such modules see, e.g., [25, 46, 50]). A natural way to allow fault tolerance for such a case is to use exact (or approximate) matching techniques on generalized sequences. Such sequences are obtained by reducing the musical alphabet as already mentioned in Section 1.2.

Next we will present various reduced alphabets that are musically justified. The users may choose one of three reduced alphabet corresponding best to their musical skill to give the query pattern[6]. Moreover, we will use also a particular alphabet reduction method, namely the octave equivalence, to speed-up a matching algorithm (Section 4.3). Having introduced the alphabets, we give an example of a musical excerpt, and its conversions to sequences over these alphabets, in Fig. 3.3.

3.5.1 Music Contour

The music contour alphabet $\Sigma_C = \{0, 1, 2\}$[7] representing intervals downwards, upwards, and a repeat respectively, is widely used in MIR algorithms, especially when input is by humming. The contour alphabet was first used by Ghias et. al. [25]. However, the alphabet has turned out not to be *discriminative* enough for MIR applications with large databases [20, 51],[I].

[5]In fact, these modules do much more than just pitch estimation, but being intuitive the term is preferred here over “automated transcription modules”.

[6]Naturally, these alphabets are applicable also for music comparison.

[7]For illustrating the contour, we will also use symbols d, u, and r (down, up, repeat) obtained via a straightforward mapping $0 \to d, 1 \to u, 2 \to r$.

This is because the contour alphabet is so small that a given contour sequence matches a large fraction of the musical database, unless the contour sequence is very long.

3.5.2 Diatonic Intervals

One abstraction of the exact intervals, based on music theory, classifies the intervals by their *possible diatonic interval sizes*[8]. The correspondence between semitones and the scale steps in the diatonic alphabet Σ_D are given in Table 3.1. Note that the only ambiguous interval is the interval of six semitones (the *tritone*), which corresponds either to three or four scale steps.

semitones	0	$1,2$	$3,4$	5	6	7	$8,9$	$10,11$	12
diatonic scale steps	0	1	2	3	3 or 4	4	5	6	7

Table 3.1: *Diatonic interval classification of one octave. The same classification pattern repeats in every octave.*

3.5.3 Quantized Partially Overlapping Intervals

The idea of *QPI classification* [I], is adapted from Narmour's implication-realization model [40], in which a classification to small, medium size and large intervals is used. Narmour claimed that the size of the interval has an effect on how the melody proceeds. For instance, a small interval increases the continuity while a large one has a supportive effect towards a change.

The intervals in our reduced alphabet are called *Quantized Partially overlapping Intervals (QPI)*. The reduction scheme quantizing chromatic intervals is similar to that of Narmour's, but this time the classes are partially overlapping. There is a class for small (1–3 semitones), medium size (3–7 semitones), and large intervals (at least 6 semitones). Naturally, there are such classes for intervals down and upwards, and an own class for a repeat. Therefore, QPI can also represent the musical contour.

semitones	...,-8	-7,-6	-5,-4	-3	-2,-1	0	1,2	3	4,5	6,7	8,...
QPI symbols	$-\gamma$	$-\mathcal{B}$	$-\beta$	$-\mathcal{A}$	$-\alpha$	o	α	$\mathcal{A}$	β	$\mathcal{B}$	γ

Table 3.2: *QPI classification.*

For illustration purposes, let us denote the symbols of the alphabet $\Sigma_{QPI} = \{0, 1, \ldots, 10\}$ by symbols $-\gamma, -\mathcal{B}, -\beta, -\mathcal{A}, -\alpha, o, \alpha, \mathcal{A}, \beta, \mathcal{B}, \gamma\}$, re-

[8]The classification is done without any knowledge of the actual *tonic*.

spectively (see Table 3.2). Note that the symbols $-\mathcal{B}, -\mathcal{A}, \mathcal{A}$ and $\mathcal{B}$ represent the ambiguous intervals -7, -6; -3; 3; and 6,7 (i.e., the overlapping parts of the interval classes). The symbols of the alphabet Σ_{QPI} (or QPI symbols) do not match only themselves, but also the adjacent symbols (excluding symbol o). To be more precise, let us denote by a and b two symbols in the interval alphabet $\overline{\Sigma}$, and by Q the function quantizing a symbol a into a symbol $Q(a)$ in the reduced alphabet Σ_{QPI} according to Table 3.2. If two symbols $Q(a)$ and $Q(b)$ match we write $Q(a) \simeq Q(b)$. The relation $\simeq$ on alphabet Σ_{QPI} is defined by Table 3.3[9].

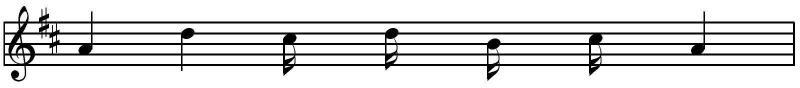
$\simeq$	$-\gamma$	$-\mathcal{B}$	$-\beta$	$-\mathcal{A}$	$-\alpha$	o	α	$\mathcal{A}$	β	$\mathcal{B}$	γ
$-\gamma$	y	y	n	n	n	n	n	n	n	n	n
$-\mathcal{B}$	y	y	y	n	n	n	n	n	n	n	n
$-\beta$	n	y	y	y	n	n	n	n	n	n	n
$-\mathcal{A}$	n	n	y	y	y	n	n	n	n	n	n
$-\alpha$	n	n	n	y	y	n	n	n	n	n	n
o	n	n	n	n	n	y	n	n	n	n	n
α	n	n	n	n	n	n	y	y	n	n	n
$\mathcal{A}$	n	n	n	n	n	n	y	y	y	n	n
β	n	n	n	n	n	n	n	y	y	y	n
$\mathcal{B}$	n	n	n	n	n	n	n	n	y	y	y
γ	n	n	n	n	n	n	n	n	n	y	y

Table 3.3: *The relation $\simeq$ on QPI symbols. QPI symbol at row i of the leftmost column is in $\simeq$ relation with QPI symbol at column j of the topmost row iff the cell i, j contains a "y".*

3.5.4 Octave Equivalence

As discussed in Section 1.1, an interval classification distinguishing pitches only within an octave is very natural in many respects; according to Forte [24], octave equivalence is "one of the most fundamental axioms of tonal music". In octave equivalence reduction, all intervals are reduced into the semitone range $0, 1, \ldots, 11$. Technically this is achieved by using alphabet $\Sigma_{12} = \{0, 1, \ldots, 11\}$ and replacing the actual interval value h (in semitones) by the value $h \bmod 12$. According to our experiments the discrimination ability of this representation is actually very good [II].

[9]Note that $\simeq$ relation is not an equivalence relation, because it is not transitive.



$\Sigma_{MIDI}/\Sigma_{12}$:	69/9	74/2	73/1	74/2	71/11	73/1	69/9
$\overline{\Sigma}/\Sigma_C$:		5/u	-1/d	1/u	-3/d	2/u	-4/d
Σ_D/Σ_{QPI}:		3/β	-1/-α	1/α	-2/-$\mathcal{A}$	1/α	2/-β

Figure 3.3: *The musical line on the top of the figure has been represented by sequences over the alphabets* $\Sigma_{MIDI}, \Sigma_{12}, \overline{\Sigma}, \Sigma_C, \Sigma_D$, *and* Σ_{QPI}. *In the example, the alphabet* Σ_{12} *corresponds to the chromatic scale starting from the middle* `c` *(key number 60 in MIDI).*

3.6 Setting the MIR Problem

The rest of this summary does not consider any more music comparison but only music retrieval. Therefore, let us now set up properly our MIR problem. Henceforth, a musical *source* $S = S_1 S_2 \cdots S_n$ (sometimes also referred to as target) is a sequence of sets of integers. Each S_i models a chord and is formally a subset of the alphabet Σ (so by using capital letters we emphasize the polyphonic nature of the source). We denote by q the size of the largest chord in S, i.e., $q = \max\{|S_i| \mid 1 \leq i \leq n\}$. If $q = 1$, the source is *monophonic*; otherwise it is *polyphonic*.

Each S_i corresponds to a chord of notes, and is comprised of notes having their onsets simultaneously. The pitch components within the chords are ordered from low to high, the lowest component, the *bass*, of a chord S is referred to as the *base* and denoted $\texttt{base}(S)$. To give an example, the sample excerpt given in Fig. 3.4 is encoded by the following sequence: $S = \langle 65, 69, 72\rangle\langle 64, 71\rangle\langle 62, 69\rangle\langle 60, 64, 67\rangle$ (here we have used MIDI values for the pitches, and angle brackets to show the chords).

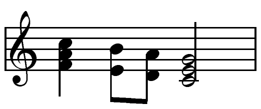


Figure 3.4: *A sample excerpt.*

A musical *query pattern* $p = p_1 p_2 \cdots p_m$ is a sequence of integers, more precisely, $p_i \in \Sigma$, for $i = 1, \ldots, m$. Hence the query pattern is always monophonic.

The content-based music information retrieval problem is as follows. Given a long database $S = S_1 \cdots S_n$ and a relatively short music query pattern $p = p_1 \cdots p_m$, the task is to find all locations in S where p occurs. Here an occurrence might mean an exact, transposed, or approximate occurrence.

We define that there is an *exact occurrence* of p at position j, if $p_i \in S_{j+i-1}$ holds for $i = 1, \ldots, m$. Furthermore, there is a *transposition invariant occurrence* of p at position j, if there is an integer d such that each $(p_i + d) \in S_{j+i-1}$.

When the source s is monophonic, an exact and a transposition invariant occurrence of p within s means that $p_i = s_{j+i-1}$ and $(p_i + d) = s_{j+i-1}$ for each i, respectively. A k approximate occurrence of p is found if there is a subsequence p' of s, such that p' can be obtained from p by using k or fewer editing operations. An *approximate transposition invariant occurrence* is defined accordingly.

3.7 Efficient Retrieval in Monophonic Databases

Let us now consider the possible ways to perform efficient queries of monophonic patterns to monophonic databases. The two alternatives for organizing such queries are the offline and online approaches. Thus, the question becomes whether to use indexing structures or not.

Recall that m and n denote the length of a pattern and a source, respectively. When using suffix structures the exact pattern matching can be performed in $O(m)$ time, but as the approximation is increased the structures become impractical rather soon. The main drawback of the approach, however, is the large amount of space that such a structure needs. In practice, even the most economical implementation of the space-efficient suffix tree needs a space (of main memory) that is 10 times the length of the source [32]. Simpler suffix structure, suffix trie, can be much larger.

Online algorithms can usually be executed with only a minor extra space, but their running times are dependent on the length of the source. As already mentioned, the fastest bit-parallel implementations can achieve linear time requirement $O(n)$ in certain circumstances.

In [33], we experimented on exact matching and conducted a comparison between suffix trie [54] (offline) and Boyer-Moore (BM) algorithm [6] (one of the fastest online approaches). As regarded to the running time, unsurprisingly, the suffix trie clearly outperformed the BM algorithm, and therefore, our first MIR prototype used the offline approach [I].

Let us now re-estimate the practicality of the approaches for music retrieval, by considering their main drawbacks by way of example of two music databases. The first one is a moderate sized database used with MELDEX system [36]. It comprises nearly 10, 000 folksongs from Britain, Ireland, North America, Germany, and China, having an average length of just under 60 notes. Assuming that one note fits one byte, concatenating the whole database in a single sequence would result in a sequence of around 600, 000 notes needing a space of 0.6 MB. In this case, the suffix tree would take at least 6 MB of the main memory. Due to our experiments with a modern computer[10], Myers' algorithm scans through such a database within 0.04 seconds. Therefore, as regards to the considered features, both of the approaches are applicable in this case. In particular, Myers' algorithm obviously meets the interactivity requirement of Section 1.2.

In [4], Bainbridge et al. estimated the number of MIDI files in the Internet by collecting them heuristically. After removing the exact duplicates, they had gathered 99, 000 files, having a total of 528 million notes. A suffix tree for such a database would require approximately 5 GB of the main memory, while Myers' algorithm would scan through it in approximately 40 seconds. Although Myers' algorithm would not meet the interactivity any more, it still could be used. This would not be the case with the excessive amount of space needed by the suffix tree.

Not only the facts above, but also the flexibility of the online approach, make it a sensible choice for us: In Chapter 4, we will illustrate how our novel ideas, such as QPI classification and D_N distance, are adaptable with only minor modifications to bit-parallel algorithms. We will show that bit-parallelism is applicable even for queries to polyphonic databases.

3.8 Online Retrieval in Polyphonic Databases: Algorithm C

The 'naive' string matching algorithm (see e.g., [15, p.34]) can be adapted rather straightforwardly to the problem of finding exact and transposition invariant occurrences of monophonic patterns within polyphonic sources. The algorithm, called here Algorithm C, runs in time $O(nq(q+m))$ [II] (q is the maximum of chord sizes).

The naive string matching algorithm aligns the pattern with each position of the source, step by step. At each alignment the corresponding symbols are compared from left to right, one by one, until a mismatch occurs, or the end of the pattern is reached (in which case an occurrence is reported). Then the pattern is shifted one position to the right.

[10]The experiments were run in a PC with Intel Pentium III of 700 MHz and 768 MB of RAM.

To match musical query patterns in transposition invariant way against polyphonic sources, the intervals between two consecutive chords in S need to be calculated. This is done by a straightforward subroutine `eval_intervals(`$\mathcal{S}$`,S2,S3)`, where `S2` and `S3` are chords in S, and $\mathcal{S}$ is the set of intervals returned by the subroutine. The subroutine uses $O(q^2)$ time for a pair of chords.

At each alignment j, $1 \leq j \leq n - m + 1$, Algorithm C first calls `eval_intervals`. The occurrences of $p_2 - p_1$ in $\mathcal{S}$ are stored in a linked list. If the linked list is not empty, i.e., $p_2 - p_1$ matches some $x_2 - x_1$ (where $x_1 \in S_i$ and $x_2 \in S_{i+1}$), the next step is to check if there is $x_3 \in S_{i+2}$ such that x_3 must be equal to $p_3 - p_2 + x_2$[11]. Intervals $p_j - p_{j-1}$ for $j = 4, \ldots, m$ are treated accordingly. Whenever a matching fails, the next non-checked occurrence of $p_2 - p_1$ in the linked list is tried for the matching.

Though the number of possible intervals between two consecutive chords is q^2, only q of them, at most, are the same. Therefore, if Algorithm C checks all the source positions, its time complexity becomes $O(n(q^2 + qm)) = O(nq(q + m))$ in the worst case, because `eval_intervals` takes $O(q^2)$ and matching the pattern up to q times takes $O(qm)$ for each source position.

[11] The checking requires only a constant time via the use of *bucket sorting*.

Chapter 4

Implementations and the SEMEX Prototype

Having presented the string matching machinery and various ways to tune it for MIR applications in the previous chapters, next our music retrieval prototype SEMEX [V], implementing a sensible subset of those ideas, will be described.

4.1 System Architecture

The components of SEMEX are shown in Fig. 4.1. The user interface component makes use of the other components in order to present the user with a single interface. A separate pitch estimator component, provided by Tolonen and Karjalainen [50], makes the conversion of digital audio into symbolic form. The core of the system, consisting of the components song manager, database and query engine, performs queries and allows the user to compile SMF and IRP files into a database. The databases of SEMEX are in simple flat-file format, which are called *MDB files*.

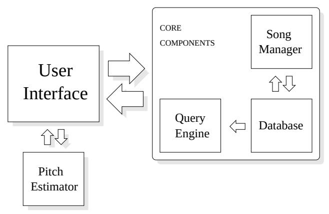


Figure 4.1: *Components of the SEMEX Prototype*

Since SEMEX is controlled via a simple command-line user-interface, all the features that have been implemented in SEMEX are considered as options that are selected by using switches. The options of a current version, most of which are easy to implement, are listed in Appendix B. For instance, by selecting correct switches the query pattern may be given as symbolic strings of notes or intervals, as IRP or SMF file, or even by humming. Moreover, it is possible to play found occurrences together with some context around them.

4.2 Retrieving from Monophonic Sources

In the special case of a monophonic database, SEMEX uses Myers' algorithm [39] as a default (for comparison reasons, the standard dynamic programming and Ukkonen's cutoff algorithm have also been embedded). The distance measure to be applied can be chosen among $\overline{D}_L$ and D_N distances. An optional alphabet reduction scheme (contour, diatonic intervals, or QPI classification) may be chosen, if $\overline{D}_L$ has been selected.

When using Myers' algorithm, SEMEX automatically executes two extra phases that are worth mentioning. In order to report only the locally best matches, some *pruning* is performed. After the matching, a *verification* phase giving the starting positions of the best matches is executed. Let us next briefly introduce these phases.

Pruning. It is useful to eliminate some matches instead of reporting all of them. Specifically, matches which are super- or substrings of a better match should be pruned. One source of such matches is given by Theorem 2.1: the fact that the difference between two adjacent cells on a row of the dynamic programming table is at most one, implies an undesirable proliferation of matches on both sides of a center match with an edit distance below k.

In order to avoid such 'pollution', the scanning algorithm is modified to report only positions where the edit distance does not grow and which are not followed by a better match. Further pruning is done during verification.

Verification. In Myers' algorithm, the exact factor of s corresponding to a match cannot be determined; the algorithm is only capable of computing the distance. Thus, each matching position j is fed to the slower dynamic programming algorithm which resolves the minimizing path. Therefore, it is possible to use another measure to decide the order between occurrences, that were found to be equally dissimilar to the query pattern at the first place. SEMEX uses such a secondary metric preferring a replacement operations where the pattern and source intervals differ by at most 7 semitones (i.e., a *pure fifth*) over insertions and deletions.

When the exact factor of s is known, matches are pruned so that only the best match is retained out of matches beginning from the same position in s. Together with the initial pruning performed by the scanning algorithm, these measures suffice to filter out matches which are super- or substrings of a better match.

4.2.1 Implementational Issues

Let us next sketch bit-parallel implementations for function D_N and QPI classification. These are illustrated here by adapting the shift-or algorithm because of its intuitive-enough nature. Adapting Myers' algorithm is analogous but much less intuitive because of the complexity of the algorithm.

D_N measure. Slight modification to the D_N measure is required in order to be able to apply the shift-or algorithm developed to exact matching. The degenerated form, denoted $D_{N'}$, is as follows:

$$d_{ij} = \{d_{i-1,j-1} + (\text{if } ((a_j - a_{j-1} = b_i - b_{i-1}) \text{ or } (a_j = b_i)) \text{ then } 0 \text{ else } 1\}. \tag{4.1}$$

In this case two auxiliary bit-arrays are needed, including T and an extra bit-array Δ of size $|\overline{\Sigma}| \times (m-1)$. For each interval l appearing in the pattern, a column $\Delta[l]$ is updated by the rule: $\Delta[l].r = 0$ if $\overline{p}_r = l$, otherwise $\Delta[l].r = 1$. Moreover, in order to allow an initial 0-cost transposition, the algorithm uses a bit-vector J having a zero only as its least meaningful bit.

Denoting the bitwise `or` and `and` operators by | and &, respectively, the shift-or algorithm applying $D_{N'}$ distance becomes as follows:

Algorithm 4.1 (shift-or applying $D_{N'}$)

```
begin
1. for each $a \in \Sigma$ do $\mathtt{T}[a] := 2^m - 1$;
2. for each $d \in \overline{\Sigma}$ do $\Delta[d] := 2^m - 1$;
3. $\mathtt{J} := 2^m - 1; \mathtt{J} := \mathtt{J} - 1$;
4. for $i := 1$ to $m - 1$ do
5.    $\mathtt{T}[p_i] := \mathtt{T}[p_i] - 2^{i-1}$;
6.    $\Delta[\overline{p}_i] := \Delta[\overline{p}_i] - 2^{i-1}$;
7. $\mathtt{T}[p_m] := \mathtt{T}[p_m] - 2^{m-1}$;
8. $\mathtt{E} := 2^m - 1$; $\mathtt{E} := \mathtt{shiftleft(E)|(T}[s_1]\,\&\,\mathtt{J)}$;
9. for $j := 2$ to $n$ do
10.   $\mathtt{E} := \mathtt{shiftleft(E)|(T}[s_j]\,\&\,\Delta[s_j - s_{j-1}]\,\&\,\mathtt{J)}$;
11.   if $\mathtt{E}.m = 0$ then $\mathtt{write}(j)$
end
```

The two auxiliary arrays are established in lines 1–7. After the auxiliary arrays have been computed, the bit-mask corresponding to recurrence (4.1) is achieved by taking a bitwise `and` of the bit-masks $\mathtt{T}[s_j]$ and $\Delta[s_j - s_{j-1}]$ (line 10), because the bitwise `and` preserves the 0-bits. By removing `J` from the end of lines 8 and 10, initial transpositions cannot be used without a cost. Note that this modified algorithm preserves the original time complexity $O(\lceil \frac{m}{W} \rceil n)$.

QPI Classification. Let us denote by Σ_{QPI} the reduced alphabet, by Q the function quantizing a symbol a in $\overline{\Sigma}$ into symbol $Q(a)$ in Σ_{QPI}, and by $\simeq$ the QPI matching relation described in Section 3.5.3. The shift-or algorithm applying the QPI classification becomes as follows.

Algorithm 4.2 (shift-or applying QPI's)

```
begin
1. for each $z \in \Sigma_{QPI}$ do $\mathtt{T}[z] := 2^m - 1$;
2. for $i := 1$ to $m - 1$ do
3.    for each $z \in \Sigma_{QPI}$ such that $z \simeq Q(\overline{p}_i)$ do
4.       $\mathtt{T}[z] := T[z] - 2^{i-1}$;
5. $\mathtt{E} := 2^m - 1$;
6. for $j := 1$ to $n - 1$ do
7.    $\mathtt{E} := \mathtt{shiftleft(E)} \mid \mathtt{T}[Q(\overline{s}_j)]$;
8.    if $\mathtt{E}.m = 0$ then $\mathtt{write}(j)$
end
```

For any QPI symbol there are at most three distinct QPI symbols matching it. Thus, there are at most three entries in `T` that have to be taken into account in lines 3–4 for each $\overline{p_i}$. It is very easy to obtain these entries without needing to check all the possibilities. As can be seen, there is no need to modify the original scanning phase at all.

4.3 Retrieving from Polyphonic Sources: Mono-Poly Algorithm

In SEMEX, we can deal with databases containing polyphony in two ways. The database can be reduced into a monophonic form by considering only the highest notes of chords (as suggested in [51]), and then applying the techniques for monophonic databases. Let us now describe more detailed the second possibility.

The Algorithm C described in the previous chapter is capable of matching a query pattern against any single source position. So, a more efficient solution may be achieved, if it is used as a checking phase of a filtering algorithm. In [II], we present two alternatives for marking phases of such a filter. These alternatives, based on shift-or algorithm, check that the source contains the intervals of the pattern in the same order, but do not 'bind' the corresponding elements of the chords (see Fig. 4.2 for an illustration).

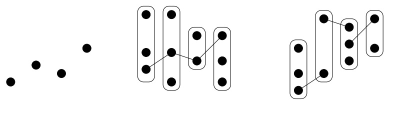


Figure 4.2: *Mono-poly marking. The query pattern, given to the left, has a proper occurrence in the first chord sequence (the corresponding elements are bound), but only a spurious occurrence in the second (the corresponding elements are not bound). Both are marked as candidates.*

The marking phase of mono-poly filter is further divided into *source preprocessing*, *pattern preprocessing*, and *scanning phases*. The source preprocessing is necessary only before the first database query. The two remaining subphases correspond to certain parts of shift-or algorithm (Algorithm 2.2, page 20); pattern preprocessing to lines 1–2, and scanning to lines 3–6.

Source Preprocessing. This subphase forms a sequence $\overline{\mathtt{S}}[1]\cdots\overline{\mathtt{S}}[n-1]$, where each $\overline{\mathtt{S}}[j]$ is a bit-vector of $|\Sigma_{12}|$ bits and storing the intervals between S_j and S_{j+1} reduced within one octave. By using bitwise operations and a fact that only a certain subset of the possible q^2 intervals can appear between two consecutive chords, the subphase avoids the apparent $O(nq^2)$ time requirement. To this end, let us represent chords S_j, for $2 \leq j \leq n$, by bit-vectors $\varphi[j]$ of $|\Sigma_{12}|$ bits:

$$\varphi[j+1].i = \begin{cases} 0, & \text{if there is an element } x \text{ in } S_{j+1}\text{, such that} \\ & \quad (x - \mathtt{base}(S_j) \bmod |\Sigma_{12}|) = i, \\ 1, & \text{otherwise.} \end{cases}$$

Now, a bit-vector $\overline{\mathtt{S}}[j]$ is established as follows:

$$\overline{\mathtt{S}}[j].i = \begin{cases} 0, & \text{if } \varphi[j+1].i = 0, \\ 0, & \text{if } \varphi[j+1].((i-l) \bmod |\Sigma_{12}|) = 0\text{, where } l = y - \mathtt{base}(S_j) \text{ and} \\ & \quad y \text{ is an element in } S_j\text{, such that } y \neq \mathtt{base}(S_j) \\ 1, & \text{otherwise.} \end{cases} \tag{4.2}$$

The intuition above is that $\varphi[j+1]$ gives the intervals between the elements of S_{j+1} and the base of the chord S_j. By shifting these intervals, they are then copied to represent the remaining elements of S_j, as well. Equation (4.2) can be implemented efficiently by using a *right circularshift* bitwise operator, denoted $\texttt{rcs}(a,b)$, which shifts a bit-vector a by b bits to the right in a circular manner. For instance, if $a = 01010$ then $\texttt{rcs}(a,1) = 00101$ and $\texttt{rcs}(a,2) = 10010$. Thus, each copying can be done in constant time. Since at most $(n-1)\times(q-1)$ such copying are needed, $\overline{\texttt{S}}[1]\cdots\overline{\texttt{S}}[n-1]$ can be formed in $O(nq)$ time (assuming that $|\Sigma_{12}| \le W$).

The algorithmic sketch of this subphase is as follows.

Algorithm 4.3 (mono-poly: source preprocessing)

```
begin
1.  for $j := 1$ to $n-1$ do
2.      $\overline{\texttt{S}}[j] := 2^{|\Sigma_{12}|} - 1$;
3.      for each $a \in S_{j+1}$ do
4.          $a := (a - \texttt{base}(S_j)) \bmod |\Sigma_{12}|$;
5.          if $\overline{\texttt{S}}[j].a = 1$ then $\overline{\texttt{S}}[j] := \overline{\texttt{S}}[j] - 2^a$;
6.      $\overline{\texttt{S}}[j] := (\overline{\texttt{S}}[j] \;\&(\bigwedge_{a\in(S_j\setminus\texttt{base}(S_j))} \texttt{rcs}(\overline{\texttt{S}}[j], (a - \texttt{base}(S_j)) \bmod |\Sigma_{12}|)))$;
end
```

Once the source, representing a musical database, has been preprocessed and the sequence $\overline{\texttt{S}}[1]\cdots\overline{\texttt{S}}[n-1]$ is available, an arbitrary number of queries to that sequence can be performed by running the subphases that will be described next.

Pattern Preprocessing and Scanning. The pattern preprocessing subphase constructs a bit-array T' of $2^{|\Sigma_{12}|}\times(m-1)$ bits, in linear time. Array T' is alike the bit-array T of Algorithm 2.2. Instead of having a column for every symbol appearing in the pattern, T' has a column for every possible value of $\overline{\texttt{S}}[j]$. The octave equivalence assumption keeps T' reasonably sized.

To achieve linear time complexity, two extra arrays are used while building T'. A bit-array I of $|\Sigma_{12}|\times|\Sigma_{12}|$ bits has a column for every possible interval in the reduced alphabet, while a bit-array L of $|\Sigma_{12}|\times(m-1)$ bits stores the positions of each interval in the query pattern. Their bits are set as follows (here $1 \le i \le m-1$; $1 \le j,k \le |\Sigma_{12}|$)

$$\texttt{I}[j].k = \begin{cases} 0, & \text{if } j = k, \\ 1, & \text{otherwise,} \end{cases} \qquad \texttt{L}[j].i = \begin{cases} 0, & \text{if } (p_{i+1} - p_i) \bmod |\Sigma_{12}| = j, \\ 1, & \text{otherwise.} \end{cases}$$

Note that every $\overline{\texttt{S}}[j]$ can be viewed as an integer l (and vice versa), whose value is restricted to the range $0 \leq l \leq 2^{|\Sigma_{12}|} - 1$. These values are used as indices to a table T'. Moreover, we use bit-vectors $\texttt{I}[j]$ to locate intervals within $\overline{\texttt{S}}[j]'s$, by 'sliding' them one-by-one over all the values l. This forms the table T':

$$\texttt{T}'[l].i = \begin{cases} 0, \text{ if } \texttt{I}[k].j = 0 \text{ and } l.j = 0 \text{ and } \texttt{L}[k].i = 0, \\ 1, \text{ otherwise,} \end{cases}$$

where $l.j$ denotes the j's bit of l. In this way, building the array T' takes time $O(|\Sigma_{12}| \cdot 2^{|\Sigma_{12}|})$.

The scanning subphase is analogous to that of shift-or algorithm, but in this case the pattern will be matched against the sequence $\overline{\texttt{S}}[1] \cdots \overline{\texttt{S}}[n-1]$ instead of S. The following implements the remaining two subphases.

Algorithm 4.4 (mono-poly: marking)

```
begin
1.  for $k := 1$ to $|\Sigma_{12}|$ do $\{\texttt{I}[k] := 2^{|\Sigma_{12}|} - 1,\ \texttt{I}[k] := \texttt{I}[k] - 2^{k-1}\}$;
2.  for $k := 1$ to $|\Sigma_{12}|$ do $\texttt{L}[k] := 2^{m-1} - 1$;
3.  for $i := 2$ to $m$ do
4.      $b := (p_i - p_{i-1}) \bmod |\Sigma_{12}|$;
5.      $\texttt{L}[b] := \texttt{L}[b] - 2^{i-2}$;
6.  for $l := 1$ to $2^{|\Sigma_{12}|}$ do
7.      $[l] := 2^m - 1$;
8.      for $k := 1$ to $|\Sigma_{12}|$ do
9.          ivect := $\texttt{I}[k]$;
10.         if ivect $|$ $l$ = ivect then $\texttt{T}'[l] := (\texttt{T}'[l]\ \&\ \texttt{L}[k])$;
11. $\texttt{E} := 2^{m-1} - 1$;
12. for $j := 2$ to $n$ do
13.     $\texttt{E}$ := shiftleft(E) $|$ $\texttt{T}'[\overline{\texttt{S}}[j]]$;
14.     if $\texttt{E}.m = 0$ then AlgorithmC($j$);
end
```

The exact time and space complexities of algorithm 4.4 are $O(n+m+d)$ and $O(n+c)$, respectively, where $d = |\Sigma_{12}| \cdot 2^{|\Sigma_{12}|}$ and $c = 2^{|\Sigma_{12}|} + 2|\Sigma_{12}|$.

The Correctness of Mono-Poly Algorithm. In [III], we show that the mono-poly algorithm is correct[1]. Moreover, it is easy to see that the algorithm always terminates; it does not contain infinite loops. Therefore, the algorithm is *totally correct.*

[1]We do not prove the correctness of the checking phase. However, since it is a straightforward modification of the naive string matching algorithm, we can take it for granted.

4.4 About the Performance of SEMEX

SEMEX meets Salosaari and Järvelin's requirements for a MIR system (described in Section 1.2). Let us consider the two important extra requirements, i.e. flexibility and interactivity, mentioned in that section. The bit-parallel approach has proven itself flexible by enabling straightforward and efficient implementations of various ideas presented in this summary. However, as discussed in Section 3.7, the interactivity is not possible to achieve with very large music databases.

To set up a limit for interactivity is a subjective matter. However, running an MIR query within a second, would probably be accepted as interactive performance. Due to our experiments (see [II] and [V]) with recent Pentium III computers, when the database is monophonic, SEMEX is capable of executing MIR queries within a second if the database contains up to approximately 8 million notes. Such an estimation for queries using mono-poly algorithm is much less accurate, because the speed of the algorithm depends on many parameters [II]. Following from our experiments, however, we believe that it is rather typical (when q stays small enough) that the mono-poly algorithm achieves the speed of the augmented Myers' algorithm that SEMEX uses for monophonic databases. This is the case because both the algorithms incorporate a linear-time bit-parallel phase and a slower phase that is executed with a subset of all possible source locations. However, a direct comparison of the algorithms is actually not fair because the augmented Myers' algorithm for the monophonic case allows approximation with transposition invariance, while the mono-poly algorithm allows only transposition invariance without proximity.

Chapter 5

Conclusions

In this thesis, we have studied various aspects of music comparison and retrieval. The first step towards any kind of information retrieval application is to decide which is the feature to be extracted and used in the matching. We have chosen to use pitch information as the feature, to represent the usually very complex musical surface. When making music retrieval queries on pitch information, one should take into account transposition invariance, since any two melodies that are just transpositions of each other (represented by different musical keys) are perceived as the same.

In order to obtain content-based retrieval capability, one must be able to match a query pattern against musical sequences appearing in a music database. The general string matching machinery, as such, is sufficient only for very crude music retrieval. We have tuned the machinery to enable the use of plausible issues based on music theoretic and music psychological findings. We have presented ideas about how to reduce the number of considered pitches (the size of the musical alphabet) sensibly, in order to allow tolerance of systematic errors in the pattern, and to make matching more efficient. We have also presented various editing operations and transposition invariant edit distances that can be used in music retrieval.

Out of the various ideas, we have chosen a subset that is implementable very efficiently by using bit-parallelism, that includes important musical issues, and offers a complete matching scheme for symbolically encoded music. The subset has been implemented and embedded in our music retrieval prototype called SEMEX (Search Engine for Melodic EXcerpts). SEMEX is the first music information retrieval software system applying extensively the fast bit-parallel approach. Moreover, it is the first engine enabling retrieval directly from a polyphonic database.

There is still work to be done. For instance, studies of perception have indicated that the role of rhythm in melody recall and identification can be almost as important as that of pitch. Combining the essential invariances of the two attributes, namely invariance under transposition for pitch and invariance under different tempi for rhythm, in a practical algorithm is a challenging task. In [34], we have already presented an encoding scheme obtaining those invariances. However, it does not yet allow natural error tolerance.

Moreover, in the case of polyphonic sources we have restricted ourselves to exact matching. The bit-parallel technique applied in the marking phase of our algorithm, can be modified to allow approximate pattern matching by using Wu and Manber's technique [56]. However, after such a modification the marking phase would require $O(kn)$ time (where k is the proximity threshold value and n the length of the source). So, we are currently working with other techniques, such as the Myers' algorithm (which we applied to monophonic databases). Should the algorithm be adapted to this case as well, it would allow a marking phase running in a linear time independent of the constant k, and approximate matching also in the case of polyphonic databases.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching. *Communications of the ACM*, 18(6):333–340, 1975.

[2] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[3] R. Baeza-Yates and C. H. Perleberg. Fast and practical approximate string matching. In *Combinatorial Pattern Matching, Third Annual Symposium*, pages 185–192, 1992.

[4] D. Bainbridge, C.G. Nevill-Manning, I.H. Witten, L.A. Smith, and R.J. McNab. Towards a digital library of popular music. In *Proceedings of the fourth ACM conference on digital libraries*, pages 161–169, Berkeley, CA, 1999.

[5] H.R. Blackwell and H. Schlosberg. Octave generalization, pitch discrimination and loudness thresholds in the white rat. *Journal of Experimental Psychology*, 33:407–419, 1943.

[6] R. Boyer and S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[7] E. Cambouropoulos, T. Crawford, and C. S. Iliopoulos. Pattern processing in melodic sequences: Challenges, caveats & prospects. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 42–47, Edinburgh, 1999.

[8] W. Chang and J. Lampe. Theoretical and empirical comparisons of approximate string matching algorithms. In *Combinatorial Pattern Matching, Third Annual Symposium*, pages 1–14, 1992.

[9] J. C. C. Chen and A. L. P. Chen. Query by rhythm - an approach for song retrieval in music databases. In *Proceedings of the Eighth International Workshop on Research Issues in Data Engineering (RIDE'98)*, 1998.

[10] T-C. Chou, A. L. P. Chen, and C-C. Liu. Music databases: Indexing techniques and implementation. In *Proceedings of the IEEE International Workshop on Multimedia Data Base Management Systems*, 1996.

[11] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):247–267, 1970.

[12] E. F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*. Prentice-Hall, 1972.

[13] E. Coyle and I. Shmulevich. A system for machine recognition of music patterns. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3597–3600, Seattle, WA, 1998.

[14] T. Crawford, C. S. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:71–100, 1998.

[15] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[16] C. Cronin. Concepts of melodic similarity in music-copyright infringement suits. *Computing in Musicology*, 11:185–209, 1998.

[17] C.J. Date and H. Darwen. *The Third Manifesto*. Addison-Wesley, 1998.

[18] M. Dillon and M. Hunter. Automated identification of melodic variants in folk music. *Computers and the Humanities*, 16:107–117, 1982.

[19] M. J. Dovey. An algorithm for locating polyphonic phrases within a polyphonic musical piece. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 48–53, Edinburgh, 1999.

[20] J. S. Downie. Music retrieval as text retrieval: simple yet effective. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 297–298, Berkeley, CA, 1999.

[21] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, Berlin Heidelberg, 1997.

[22] A. Eisenberg and J. Melton. SQL:1999, formerly known as SQL3. *SIGMOD Record*, 28(1):131–138, 1999.

[23] C.J. Ellis. Pre-instrumental scales. *Ethnomusicology*, 9:126–144, 1965.

[24] A. Forte, editor. *Tonal Harmony in Concept and Practice.* Holt, Rinehardt and Winston, New York, 1962.

[25] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming - musical information retrieval in an audio database. In *ACM Multimedia 95 Proceedings*, pages 231–236, San Francisco, CA, 1995.

[26] D. Harel. *Algorithmics: The Spirit of Computing.* Addison-Wesley, 1987.

[27] M.J. Hawley. *Structure out of Sound.* PhD thesis, MIT, 1993.

[28] W. B. Hewlett and E. Selfridge-Field, editors. *Melodic Similarity: Concepts, Procedures and Applications.* MIT Press, 1998.

[29] J. Holub, C. S. Iliopoulos, B. Melichar, and L. Mouchard. Distributed string matching using finite automata. In *Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms*, pages 114–128, Perth, 1999.

[30] N. Immerman. *Descriptive Complexity.* Springer-Verlag, New York, 1999.

[31] Andreas Kornstädt. Themefinder: A web-based melodic search tool. *Computing in Musicology*, 11:231–236, 1998.

[32] S. Kurtz. Reducing the space requirement of suffix trees. *Software–Practice and Experience*, 29(13):1149–1171, 1999.

[33] K. Lemström, A. Haapaniemi, and E. Ukkonen. Retrieving music — to index or not to index. In *ACM Multimedia 98: – Art Demos – Technical Demos – Poster Papers –*, Bristol, 1998.

[34] K. Lemström, P. Laine, and S. Perttu. Using relative interval slope in music information retrieval. In *Proceedings of the 1999 International Computer Music Conference*, pages 317–320, Beijing, 1999.

[35] H. B. Lincoln. Some criteria and techniques for developing computerized thematic indices. In Heckmann, editor, *Electronische Datenverarbeitung in der Musikwissenschaft.* Regensburg, 1967.

[36] R.J. McNab, L.A. Smith, D. Bainbridge, and I.H. Witten. The New Zealand digital library MELody inDEX. *D-Lib Magazine*, 1997.

[37] MIDI Manufacturers Association, Los Angeles, California. *The Complete Detailed MIDI 1.0 Specification*, 1996.

[38] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[39] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. In *Proceedings of Combinatorial Pattern Matching*, pages 1–13, Piscataway, N.J., 1998.

[40] E. Narmour. *The Analysis and Cognition of Basic Melodic Structures – The Implication-Realization Model*. The University of Chicago Press, 1990.

[41] M. Nykänen. *Querying String Databases with Modal Logic*. PhD thesis, University of Helsinki, Department of Computer Science, 1997. Report A-1997-3.

[42] R. Parncutt. *Harmony: A Psychoacoustical Approach*. Springer-Verlag, 1989.

[43] E. Pollastri. Melody-retrieval based on pitch-tracking and string-matching methods. In *Proceedings of the XIIth Colloquium on Musical Informatics*, 1999.

[44] P.-Y. Rolland. Discovering patterns in musical sequences. *Journal of New Music Research*, 28(4):334–350, December 1999.

[45] P.-Y. Rolland and J.-G. Ganascia. Automated motive oriented analysis of musical corpuses: a jazz case study. In *Proceedings of the 1996 International Computer Music Conference*, pages 240–243, Hong Kong, 1996.

[46] P.-Y. Rolland, G. Raskinis, and J.-G. Ganascia. Musical content-based retrieval: an overview of the melodiscov approach and system. In *ACM Multimedia 99 Proceedings*, Orlando, FLO, 1999.

[47] P. Salosaari and K. Järvelin. MUSIR – a retrieval model for music. Technical Report RN-1998-1, University of Tampere, Department of Information Studies, July 1998.

[48] I. Shmulevich, O. Yli-Harja, E. Coyle, D.-J. Povel, and K. Lemström. Perceptual issues in music pattern recognition - complexity of rhythm and key finding. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 64–69, Edinburgh, 1999.

[49] D. A. Stech. A computer-assisted approach to micro-analysis of melodic lines. *Computers and the Humanities*, 15:211–221, 1981.

[50] T. Tolonen and M. Karjalainen. A computationally efficient multi-pitch analysis model. *IEEE Transactions on Speech and Audio Processing*, 2000. (in press).

[51] A. L. Uitdenbogerd and J. Zobel. Manipulation of music for melody matching. In *ACM Multimedia 98 Proceedings*, pages 235–240, Bristol, 1998.

[52] A. L. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In *ACM Multimedia 99 Proceedings*, Orlando, FLO, 1999.

[53] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[54] P. Weiner. Linear pattern matching algorithms. In *Proceedings of IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[55] G. Wiggins, E. Miranda, A. Smaill, and M. Harris. A framework for the evaluation of music representation systems. *Computer Music Journal*, 17(3), 1993.

[56] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.

[57] O. Yli-Harja, I. Shmulevich, and K. Lemström. Graph-based smoothing of class data with applications in musical key finding. In *Proceedings of IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, pages 311–315, Antalya, 1999.

Appendix A

Related Work Summary for Chapter 1

The following abbreviations and symbols are used in the table below.

AP Allows polyphony in the database.

DP Dynamic programming.

MC Music comparison.

MR Music retrieval.

OMR Optical music recognition.

PE Pattern extraction (see page 9).

PT Pitch tracking.

TI Transposition invariant.

k The allowed number of editing operations.

m The length of the query pattern.

n The length of the source (database).

s The number of monophonic musical lines in a polyphonic music ($\approx q$).

t The allowed spacing (see Fig. 1.3).

W The length of the computer word (in bits).

* Assuming that $m < W$.

	used for	pitch representation	rhythm representation	AP	TI	other factors	matching algorithm	type of matching	time complexity
Stech, [49]	PE	intervals and contour	durations	-	+			pitch:exact+ *inversion,retrograde* rhythm:exact+retrograde	
Dillon & Hunter, [18]	MR	diatonic pitches, stressed pitches	-	-	+	measure/phrase information		exact (proximity via representations)	
Mongeau & Sankoff, [38]	MC	pitches & diatonic intervals	durations	-	-	weights based on consonance	DP	Levenshtein + consolidation, fragmentation	$O(mn)$
Ghias & al., [25]	MR	contour	-	-	+	PT	Baeza-Yates & Perleberg [3]	k mismatches	$O(mn)$
Chou & al., [10]	MR	pitches in a bar as unity	-	-	+	requires metric position analysis	*B+tree*	exact+*duplication, disorder,elimination*	$O(m)$
Bainbridge & al., [36, 4]	MR	intervals and contour	durations	-	+	OMR	DP	as Mongeau & Sankoff	$O(mn)$
							Wu & Manber[56]	k differences	$O(kn)$
Chen & Chen, [9]	MR	-	note-on times in a bar as unity	-	+	requires metric position analysis	suffix tree	exact	$O(m)$
								+*contraction,extension, shifting,splitting,merging*	
Kornstädt, [31]	MR	several	-	-	+			exact	
Crawford & al., [14]	MR	intervals	-	+	+		Aho-Corasick[1]	exact	$O(mn)$
Shmulevich & al., [13];[48]	MR	intervals	duration ratios	-	+	key finding algorithm		exact +*perceptual errors*	
Uitdenbogerd & Zobel, [52]	MR	intervals	-	+	+	polyphony reduction (=melody extraction)	DP	k differences	$O(mn)$
Dovey, [19]	MR	abs. pitches	-	+	-	polyphonic patterns	straightforward recursion	exact+spacing	$O(mn\,(t+1)^{(m-1)})$
Holub & al, [29]	MR	abs. pitches	-	+	-		shift-or	mono vs. poly (exact)	$O(n)^{\star}$
								poly vs. poly (exact)	$O(ns)^{\star}$
								poly vs. poly (k mismatches)	$O((k+1)ns)^{\star}$
Rolland, [44]	PE	several (incl. intervals & contour)	durations & duration ratios	-	+	takes into account metric positions	FlExPat	augmented k differences	$O(mn)$

For a comparison of MIR systems see the appendix of [V].

Appendix B

SEMEX Options for Chapter 4

```
SEMEX Prototype version 1.11 18-OCT-2000.
Usage: Semex [options] command parameters
  Semex ?                           Display this help.
  Semex [-r] c database files...    Create a database from a set of files.
  Semex [-r] a database files...    Add files to an existing database.
  Semex l database                  List the contents of a database.
  Semex i source [target]           Convert an SMF or MDB file to IRP format.
  Semex s files...                  Display some statistics on the given files.
  Semex [-options...] R m file k n  Make 100 random searches and display the time taken.
  Semex [-a{bcdm}] [-{dfw}] [-m{l[{cdq}]n}] [-n#] [-p[{ac}][#]] [-r] [-s{dfn}] f pattern k files...
                                    Search for a pattern in a set of files. k is an
                                    integer parameter controlling the number of editing steps
                                    permitted (default: 0). Optional arguments are:
                                     -a$  Selects the algorithm $ to be used:
                                           b  Myers' bit-parallel algorithm (the default),
                                           c  Ukkonen's cutoff algorithm,
                                           d  The basic dynamic programming algorithm,
                                           m  mono-poly filter,
                                     -d   Obtains the pattern from a sound device
                                          given in place of the pattern.
                                     -f   The pattern is an IRP, MDB or SMF file
                                          instead of a symbolic string of notes.
                                     -w   The pattern is a wave file instead of a
                                          symbolic string of notes. This option
                                          invokes the pitch estimator.
                                     -m   Sets the metric (ignored if -am is set):
                                           l  Levenshtein (the default),
                                              c  uses music contour,
                                              d  uses diatonic intervals,
                                              q  uses QPI classification,
                                           n  D_N.
                                     -n#  Sets the number of matches to
                                          display (default: all).
                                     -p#  Plays match # (default: best); optionally
                                           a  plays the whole song containing the match,
                                           c  includes some context around the match.
                                     -r   reduction mode:
                                           i  ignores polyphonic tracks,
                                           r  reduces them (the default).
                                     -s$  Sets the sort key $:
                                           d  distance (the default),
                                           f  file, producing a sparse output,
                                           n  track name.
```

ISSN 1238-8645
ISBN 951-45-9573-4
Helsinki 2000
Helsinki University Press