MUSICAL INFORMATION RETRIEVAL USING MUSICAL PARAMETERS

Kjell Lemström *
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
klemstro@cs.helsinki.fi

Pauli Laine
Department of Musicology
P.O. Box 35
FIN-00014 University of Helsinki, Finland
Pauli.Laine@helsinki.fi

Abstract. The application domain for automatical retrieval of melodic excerpts in musical collections is wide; e.g. it would facilitate the work of music researcher trying to find specific features in music. In this paper we consider several parts of the retrieving process. We present our representation for musical data. This *inner representation* is converted and established from MIDI-files. For the matching we use a particular encoding (*two dimensional relative code*), which is formed out of the inner representation. This encoding can be interpreted differently depending on the way the key is given. Furthermore, in the matching phase we use an efficient indexing structure, well-known in string pattern matching, called suffix-trie.

1 Introduction

In the earlier researches concerning musical data representation, researchers seemed to be rather sensible to the delicate details of different styles of music. One example of such a meticulous approach is Leo Plenckers encoding system for Spanish medieval songs [Ple82, p. 59].

Our aim is to develop an effective musical information retrieval system still maintaining the music theoretical and musicological adequacy. We claim that it is important to have an encoding system, which can be expanded to include different style-oriented details. This would both facilitate the tune retrieving in cases where this stylistic information can be used as discriminator, and improve the musicological research possibilities.

The symbolic or numeric representation of musical signal in computer is not standardized; the generally used computer music representation is MIDI. Unfortunately, it has certain well-known drawbacks: In a conversion process many important musical properties are not included. Enharmonic notes, bar lines, dynamic markings and almost all information on musical structure are lost when transcribed to MIDI. To facilitate the analyzing and retrieving processes we have developed an inner representation, which includes several important parameters for each note.

Recently, there have been some proposals in the literature how to implement such a musical information retrieval system. Ghias et. al. [GLCS95] used "melodic contours" to allow a certain kind of errors in the intervals. The melodic contour means that intervals are classified into three distinct equivalence classes: intervals downwards, upwards and prime. We claim that if this alphabet of only three characters is used, every musical sense may be lost. However, some kind of reduction is needed when the key contains lots of "fuzzy" intervals, as it does often in hummed or sung keys. In this kind of situation we reduce the alphabet: we classify the intervals into seven partially overlapping classes: *small*, *medium* and *large*, up- or downwards, and *prime*. By small intervals we mean scale intervals from one semitone to three semitones, by medium tritonic intervals from third to seventh and by large intervals octavic intervals beginning from sixth. This classification of intervals allows small errors in the actual size of the interval because characteristics of intervals are preserved. We call these classified intervals *quantized partially overlapping intervals*, or in short *qpi*. These interval categories resemble those presented by Narmour [Nar90]. In Narmours theory small, "chordal" and large intervals had different implication-realization functionalities. Here the interval categories are used to include small errors and minor transformations of the melodies. By using a similar method we can classify the durations of notes by comparing them into the duration of very preceding note.

To represent musical durations we have adopted a system which tries to facilitate both relatively accurate timing and symbolic manipulation of durations. Timed events can be quantized and mapped to symbols. A classification of the note durations into classes *longer* and *shorter* than previous is in many cases sufficient, in some cases even the only sensible classification. We have also considered the problem that the notes are hardly ever played in exact legato. The problem is that the former note might end perceptibly before the next note begins, although there is not any intended pause. Usually, musical notes are played with varied non-legato. Thus, we chose to use the *inter onset interval*, or *IOI*, in our internal representation. IOI is the time spent between adjacent note-event starting points.

* The author is supported by the Helsinki Graduate School of Computer Science and Engineering, and the Academy of Finland (grant 8745).

We use three categories for the key: *exact*, *semiexact* and *noisy key*. The proper category is chosen according to the way the key is given. An exact key is given by draft notation; we can assume that intervals and note durations are almost free of mistakes. Semiexact key can be given for example by a keyboard. Now we can expect that the intervals are nearly as they are wanted to be, but in durations many errors occur. Noisy key is given e.g. by humming. In this case, the errors are expected to occur both in durations and in intervals. Depending on the key category different alphabets over durations and intervals are used.

We propose a method to implement the retrieval. The method is based on a suffix-trie, a well-known data structure for indexing, originally designed to string pattern matching [CR94, Gus97], not yet used in musical information retrieval. The usage of such an indexing structure is strongly suggested by our previous experiments [LHU98]. We modify this structure to fill our requirements. By using a suffix-trie the exact matching can be done very fast and the case of k-mismatches (i. e. at most k characters of the key are changed) still rather effectively. Now the exactness means that the two melodic strings contain the same characters in the same order (it should be pointed out that the characters can be from a reduced alphabet, i.e. the strings can be composed of qpi characters). Using the method presented here, efficient searches can be made in musical databases using fairly natural musical search keys.

The rest of this paper is organized as follows: In the next Section we will review some other works of musical information retrieval. Section 3 gives an overview to our representation of music (the inner representation). In Section 4 we define our two-dimensional relative encoding and in Section 5 we present the different interpretations of the encode. In Section 6 we give a short description of the suffix-trie, and in the next Section we consider how the occurrences of a given sample can be found in both the exact and the inexact case. Eventually, in Section 8 we present some accomplished experiments with our prototype system and we conclude the paper with discussion in Section 9.

2 Related Work

To our knowledge, Lincoln [Lin67] presented first the idea of using computers for musical indexing and retrieval. He presented three criteria which must be met for automatic indexing of musical materials. These where: 1) eliminating the transcription by hand, 2) effective input language for music and 3) an economic means for printing the music. Only the second criterion has turned out to be met with difficulty. Automatic music information retrieval systems have been practically nonavailable, one of the main reasons being the lack of standardized means of music input. However, research has continued in implementing different types of music information retrieval. These researches, as well as ours, are waiting the advent of standardized music input system and systematic collections of encoded musical data.

In his PhD thesis [Haw93], Hawley considered how to extract melodies from complex audio signals. Furthermore, he also shortly described a method for searching exact matches of a key given as sequence of pitch differences. The field of applications for pattern matching in music is rather wide; e.g. in [Bel97] Bell studied how statistical pattern recognition techniques (and signal processing) can be used in a guitar-to-MIDI converter by using only one transducer.

The first real experiments of music retrieval were done by Ghias et. al. [GLCS95]. As already mentioned, they used a reduced alphabet of intervals; upwards, downwards and the same as previous (the alphabet contained the letters U, D and S, respectively). They claimed that using such an alphabet and a k-mismatches algorithm by Baeza-Yates and Perleberg [BYP92] they could discriminate 90 % of the 183 melody database by using a sample of 10–12 intervals.

One rather innovative approach was presented by Chou, Chen and Liu [CCL96]. They used chordal reductions based on melodic line to represent the musical data. Their definition of the chords was rather diatonic than functional. Although the chordal reduction can effectively be used for retrieval, it also can lead to peculiar impressions of music theory, because the actual harmonization can considerably differ from the “retrieval harmonization”. Furthermore, they assumed that the bar-line information of the melodies is available. We claim, that this can be assumed of the database melodies, which can easily be analyzed. However, the analyzing of an arbitrary key, in the sense of putting the bar lines into correct positions, may be too difficult. If the key is mis-analyzed, their algorithm is probable to fail.

Another research was done by Chen and Chen [CC98]. They used rhythmic information for retrieving pieces of music. Though their approach is efficient in computational terms, it have certain features which may need further musical consideration. Firstly, the “rhythms” seems to be totally quantized and notated. In this way all the nuances of rhythmic interpretation is lost. Also the rhythm-concerned vocabulary defined by them bear rather thin resemblance with general music theory.

A different approach was presented by Foote in [Foo97]. He gave a method for retrieving both audio and music. The retrieving of the music was based on a genre classification of the music in the database. A template was produced for each genre that was considered (e.g. jazz, pop, rock, etc.). For each piece of music in the database the similarities to the templates was measured. The final classification of the piece of music was based on the distribution of the similarities.

McNab et. al. [MSBW97] have constructed a working retrieval system, called MELDEX, with a database of 9 400 melodies. For the matching they use *dynamic programming* or its variant by Wu and Manber [WM92] [1].

In [SJ98] Salosaari and Järvelin considered a model, called MUSIR, for musical information retrieval. They proposed another technique known in string pattern matching, i.e. *q-gram* technique. They also considered the main properties of a retrieval model and posed a list of requirements that is as follows: 1) it must capture representative, usable and memorable features of music and it must allow queries that are music, not just about music, 2) it must be simple enough for retrieval of rich music documents and it must hide the richness of scores and interpretation, 3) it must be based on an easily available digital representation of music, 4) it must support inexact retrieval, and 5) it must rank the matched documents according to decreasing similarity.

In this paper and in our prototype system we respond to these requirements.

3 Inner Representation

We have developed our own representation for the music to enable pre-processing of the musical information. By using our *inner representation* (or *irp*) we could speed up the retrieving processes. We store several components both for duration and for pitch information. We have also several other components that we expect to be valuable in the continuation of our research project.

The irp-file starts with an information on the piece of music it represents. The file continues with 17 *tracks*, which correspond to the MIDI tracks. The first track (track 0) of the irp-file is reserved for all the texts or lyrics of the piece of music. The remaining 16 tracks are for the instruments and they are formed by *events*. An event is a 12-tuple composed of the following information:

$$\langle strt, dur, symb, IOI, ptch, int, dyn, acc, deg, typ, txt_{strt}, txt_{end}\rangle.$$

The meanings of the previous abbreviations are as follows:

$strt$	the starting time of the note	dur	the actual duration of the note
$symb$	the symbolic representation of the duration	IOI	inter onset interval of the note
$ptch$	the pitch of the note	int	the interval from the previous note
dyn	velocity of the note (as in MIDI)	acc	is the note on accented position
deg	the root of the chord the note is in	typ	the type of the chord the note is in
txt_{strt}	the corresponding starting index of the text	txt_{end}	the corresponding ending index of the text.

As can be noticed, our inner representation is defined thereby, that for each note of each track a part of the lyrics can be assigned. Thus, the track 0 contains mass of text, about which portions can be referred by the events. We did not include (yet) neither the qpi characters nor the longer–shorter duration information. At least for the time being, they are interpreted at the run-time. Later on, we are going to do some experiments on how much the process can be speeded up if they too are included. It will be a trade-off between the performance and the cost of memory.

4 Two-Dimensional Relative Encoding

The *two-dimensional relative code* (*tdr-code*), is a sequence S consisting of *elements* s_i

$$S = s_1, s_2, \ldots, s_n,$$

where each element $s_i(\, i = \{1, \ldots, n\})$ consists of a pair of components: $\langle x_i, y_i\rangle$. The pair $\langle x_i, y_i\rangle$ corresponds to the ith note of the song that is to be coded. The first component x_i, $x_i \in \{*, \mathcal{Z}\}$, is the difference between the two notes corresponding to s_{i-1} and s_i, respectively, using a chromatic scale. We call a sequence

$$X_j = x_j, \ldots, x_n, j = \{1, \ldots, n\}$$

an *interval sequence*. The first element x_j of an interval sequence X_j is always assigned with a "don't care" value '$*$'. It indicates that, by definition, the first element of a sequence cannot have an interval, because there are no elements

[1] A small demonstration of the MELDEX system can be found at: http://repulse.iss.nus.sg:8080/cgi-bin/audio-cbr

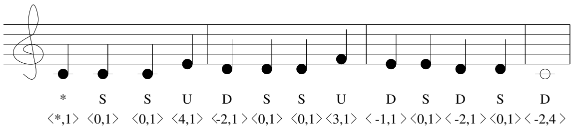

Figure 1: *The conventional notation and corresponding sequences of [GLCS95] and our tdr-sequence.*

before s_1. If the interval goes downwards (the tune corresponding to the successing note is lower than the tune of the previous one), x_i is negatively valued. By definition, any suffix X_{j+k} of an interval sequence X_j is also an interval sequence (if $0 \leq k \leq n - j$).

The second component y_i of a tdr-code sequence is the duration of the element s_i, where $y_i \in \{\mathcal{Q} \setminus 0\}$. The scaling of the components $y_i, i = \{2, \ldots, n\}$ is determined by the first component y_1. The values of the remaining components are settled according to the value of the y_1. The ratio y_i/y_1 gives the duration of the element s_i (w. r. t. the duration of the element s_1). The sequence

$$Y_j = y_j, \ldots, y_n, j = \{1, \ldots, n\}$$

is called a *duration sequence*. The negative values can be used with pauses. Again, it should be pointed out that by definition every suffix Y_{j+k} of a duration sequence Y_j is a duration sequence.

An example of this encoding can be found in Fig. 1, where the notes and the corresponding sequence S are given.

5 Interpreting the tdr-Code

To have a status of a widely used musical information retrieval system, the system has to give the user the opportunity to give the search key in natural fashion. For example, if only a (maybe drafted) musical notation is accepted, it is difficult to express uncertainty (the user may not know, how exactly the musical line is going). It is also possible that the user does not know how to notate the line at all. Therefore, we consider the possibility of giving the search key by singing, humming, whistling, playing a MIDI instrument or musical notation.

We claim that this multitude of possibilities of giving the key has to be taken into account, because the typical errors in the key depend obviously on the way the key is given. When the key is transcribed to internal representation it may contain errors both in timing and in intervals. For the present, we do not have any pitch-to-MIDI converter, thus we suppose that the key is presented with MIDI format or our inner representation format.

We use three categories for the key: *exact*, *semiexact* and *noisy key*. The proper category is chosen according to the way the key is given. An exact key is given by draft notation; we can assume that both the intervals and the note durations are rather exact. Semiexact key can be given e.g. by a keyboard. Now we can expect that the intervals are nearly as is wanted but in durations many errors may occur. Noisy key is given e.g. by humming. Now many errors are expected to occur as well in durations as in intervals. Furthermore, if the user is not very certain of the exactness of his key, he can lower the level of the default key category. Depending on the key category different alphabets over durations and intervals can be used.

We have three different types of intervals, i.e. exact, our qpi and up-down-same (*uds* in short) category of Ghias et. al. [GLCS95]. This uds category can be used in a case where the key is assumed to have several, rather big interval errors. Naturally, the smaller the alphabet the longer the key which is needed to discriminate unrelevant melodies from the result of the query. In the qpi category intervals are classified as follows: 0 semitone is *prime*, semitones $1, 2, 3$ are interpreted as *small*, semitones $3, 4, 5, 6, 7$ as *middle size* and semitones $6, 7, 8, \ldots$ as *large* intervals. Moreover, small, medium, and large intervals can be either up- or downwards giving us total of seven partially overlapping classes.

If the qpi characters are used in the query the interval sequences are interpreted at the run-time as given in Tab. 1.

Thus, an alphabet of eleven characters is used. The qpi classes overlap because we believe it makes the retrieval more error tolerant. The characters $\overline{ab}$, ab, $\overline{bc}$ and bc of this alphabet are used for the overlapping parts of the classes.

tdr-code interval	$\ldots, -8$	$-7, -6$	$-5, -4$	-3	$-2, -1$	0	$1, 2$	3	$4, 5$	$6, 7$	$8, \ldots$
qpi character	$\overline{c}$	$\overline{bc}$	$\overline{b}$	$\overline{ab}$	$\overline{a}$	o	a	ab	b	bc	c

Table 1: *Interval-sequence characters of tdr-code and the corresponding qpi characters.*

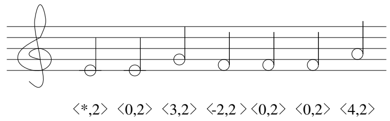

Figure 2: *A search key given by draft notation. Since, in this case qpi characters are used an exact matching of the key can be found starting at position 2 of the melody presented in Fig. 1. Now, the corresponding qpi character sequence of the key is* $o, o, ab, \overline{a}, o, o, b$ *and the sequence of the melody starting at position 2 is* $o, o, b, \overline{a}, o, o, ab$. *Thus, a match is found.*

These characters match the characters in both corresponding classes (e.g character ab matches the characters a and b). In Fig. 2 an example where this alphabet is used for matching, is given.

By using those three different categories the user can adjust the desired accuracy of the search and the amount of returned answers. In future, it may be desirable to represent the intervals in more perceptually relevant fashion, like representing the inverted intervals with same character, or taking into account the musical scale the current melodic line is supposed to be in, etc. However, these can be fairly complicated.

The interval representation system can be combined with durational alphabet where (quantized) note durations are represented as three character alphabet: *longer*, *shorter* and *the same as previous*. This timing representation is rather tolerant to tempo changes.

6 Suffix-Trie

In [LHU98] we have done some experiments on the retrieval performances. We have compared an on-line technique, namely the Boyer-Moore algorithm [BM77] to an indexing technique. As an indexing structure we have used suffix-trie. According to our experiments the usage of such an indexing structure is desirable.

Formally, a trie is a rooted tree with two main properties, i.e. 1) each node, except the root, is associated with a symbol of an alphabet, 2) any two descendants of the same node cannot be associated with the same symbol.

In Fig. 3 a) a suffix-trie is presented. That trie corresponds to the interval sequences of the song presented in Fig. 1. It consists of each suffix X_j ($j \in \{1, 2, \dots, n\}$) of the interval sequence X_1. The leaves (and possibly some internal nodes) of the trie are buckets that contain indices to the corresponding positions in the irp file. If the retrieving is principally based on durations, the trie should be constructed for the duration sequences instead of the interval sequences.

A bucket that has an offspring is an *internal bucket*. It has a particular status because it corresponds to at least two different sequences; If the matching ends at an internal bucket all the descendant buckets contain also indices to occurrences of the key.

By using a suffix-trie the occurrences of a given pattern can be found in the structure, and therefore in the sequence, very fast (in fact, in time $O(m)$ where m is the length of the sample). However, the trie structure is not optimal w.r.t. the space requirement; it grows quadratically in the length of the database. Therefore we use an abridged version that

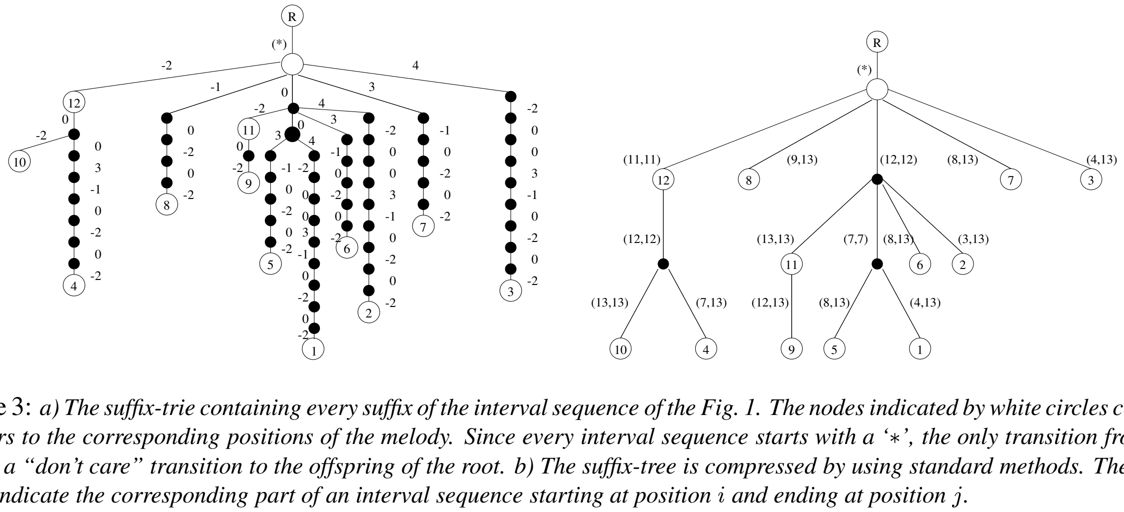

Figure 3: *a) The suffix-trie containing every suffix of the interval sequence of the Fig. 1. The nodes indicated by white circles contain pointers to the corresponding positions of the melody. Since every interval sequence starts with a '$*$', the only transition from the root is a "don't care" transition to the offspring of the root. b) The suffix-tree is compressed by using standard methods. The pairs* (i, j) *indicate the corresponding part of an interval sequence starting at position* i *and ending at position* j.

consists of the top part of the trie up to certain depth. Even this can be too large. Then the *suffix–tree*, discovered by Weiner [Wei73], should be used. It is of size $O(n)$ and can be constructed on-line [Ukk93]. In the future we are going to use this more economic structure. In Fig. 3 b) the corresponding suffix-tree is illustrated.

7 Finding Occurrences of the Key

If only the exact matches of a given key are considered the algorithm is straightforward: Traverse in the trie a path that corresponds to the key. If a corresponding sequence does not exist in the trie, the key does not occur in the database. If a corresponding sequence exists in the database, the occurrences of the key are found in the buckets that are descendants of the node corresponding to the sequence of the key. The solution is somewhat more difficult if inexactness is allowed. If the height of the trie is bounded to a value that is less than the length of key, e.g. a q-gram method could be used at the cost of performance (See e.g. [SJ98] how to apply q-grams for the musical information retrieval). Let us shortly consider different cases of applying a k-mismatches query when the length of the key is less than the height of the trie.

Mismatches in the interval sequences. To preserve the linear time query only two kinds of edit operations are allowed in the interval sequences, i.e. modulation error and local error (=wrong tune). In principle, the insertion and deletion operations are possible (at the cost of introducing dynamic programming) but they are not considered in this paper.

Because relative encoding is used, the modulation operation makes only one error to the sequence. The distance D_{int} between two interval sequences $X_1 = (1, 4, -3, 2)$ and $X_2 = (1, 2, -3, 2)$ is $D_{int}(X_1, X_2) = 1$, because only one interval is changed. In other words, there is a modulation taking place in the second note.

The second allowed operation is the local error in which only one note is mispresented. In the relative encoding this kind of error can be observed if there are two successive "modulations" resulting to the original mode. For example, the distance D_{int} between two interval sequences $X_1 = (1, 4, -3, 2)$ and $X_2 = (1, 2, -1, 2)$ is $D_{int}(X_1, X_2) = 1$, because there are two successive modulation errors at the locations 2 and 3, and $X_1[2] + X_1[3] = X_2[2] + X_2[3]$. In contrary, sequences $X_1 = (1, 4, -3, 2)$ and $X_2 = (1, 2, 0, 2)$, have a distance $D_{int}(X_1, X_2) = 2$, since though there are two successive modulation errors, the mode is changed twice ($X_1[2] + X_1[3] \neq X_2[2] + X_2[3]$).

Mismatches in the Duration Sequences. The solution is rather straightforward if the mismatches are allowed only in the durations. Now we have to observe the ratios of values in the aligned strings (See Tab. 2).

Now if we calculate the number of all different ratio values and choose the largest of these values (denoted by l), we define the distance $D_{dur}$to be $D_{dur} = m - l$ (m is the length of the key). Thus, in the example given in Tab. 2 $l = 3$ because ratio 2 is the most frequent with 3 occurrences and hence the distance is $D_{dur} = 5 - 3 = 2$.

P	=	8	4	2	2	8
Q	=	4	1	1	1	2
ratio	=	2	4	2	2	4

Table 2: *The duration sequences of the key (P) and the database string (Q) in an alignment.*

The k-mismatches query. If only modulation and local error operations are allowed, a query of the form given earlier, can be applied rather straightforward by using *depth first* and *branch and bound* heuristics: the branches of the three are examined as far as the condition $D_{int} \leq k$ holds. If a bucket node is reached in the examined branch and $D_{int} = l$ a duration sequence that has less than $k - l$ mismatches is searched for from the bucket.

An alternative, more convenient, form for the query could be: *give me all the instances of P, where k mismatches for the interval sequence and k' for the duration sequence, are allowed.* However, the solution for this query is trivial.

8 Experiments

We wanted to study what the difference between the required lengths of the keys is when using the alphabet of [GLCS95], our qpi characters or the exact ones, i.o.w. how long keys are needed to discriminate sufficiently melodies from the set of query results. The exact key represents the optimal case of discrimination. The required length of the key is an important factor, because it is difficult to give a rather long key correctly. In this experiment we had a database

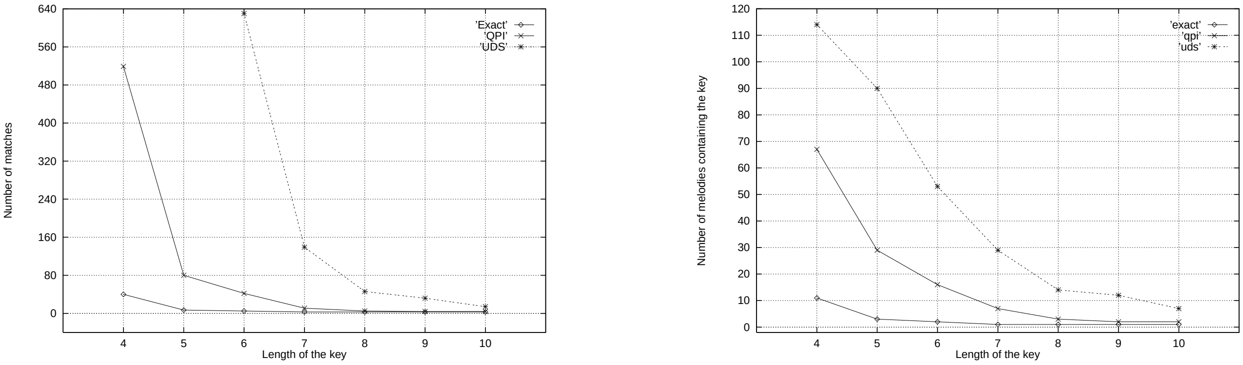

Figure 4: *Occurrences of a search key in the database. The three curves in these figures correspond to uds, our qpi and exact intervals given in top-down order. To the left the total numbers of occurrences of the key are given, to the right the number of melodies containing the key.*

of 154 pieces of music containing automatic compositions of Pauli Laine, folk-songs, etc. With no loss of generality, only the relative intervals were considered. In Fig. 5 a key, used in the experiments, is given. In Fig. 6 two instances of the key is notated. The key of desired length is always cut of from the beginning of the key given in the figure.

Fig. 4 gives the retrieval results in our experiment as a function of the length of the key. The number of occurrences of the key in the database are to the left, the number of melodies containing the given key to the right. To our experience the behaviour presented here is very typical. A pattern tends to occur many times in a melody if it occurs at least once. This phenomena can also be clearly seen by comparing the figures under consideration.

As it was assumed, the discrimination of unrelevant feedback can be done by using rather short exact key. Our qpi method lies always somewhere between the performances of the exact intervals and the uds intervals, but converges close to the optimal noticeably faster than the uds method. E.g. if a resulting set that contained less than 10 occurrences of the key was wanted in the experiment presented in Fig. 4, the required length of the key for exact, qpi, and uds methods was 5, 8 and 11, respectively. When comparing the performances between exact intervals and qpi the important fact to be remembered is that qpi allows the user to make errors when giving the key, which is not the case with the exact key. Thus, qpi takes effect to be a rather nice compromise between an efficient query alphabet and a more practical way of giving the key.

9 Discussion

One purely practical problem arises when considering a search e.g. in Internet. Even if an engine finds a set of candidates for the key, there may still be more problems. The haphazard nature of encoding the MIDI-files without proper naming, composer data etc. leaves us with a set of strangely named (usually 8.mid characters) files each having different tempo, arrangement and maybe even instrument descriptions. It could be a help if the url was returned as the result instead of the MIDI-file, but the feedback would still be inadequate.

In the future we are going to implement a more efficient indexing structure in order to be able to deal with larger databases. We are also considering to make simple analyzing of the music; we are trying to find the accented notes and use this piece of information in the matching. To utilize the durations of the notes some quantization is needed. However, it is rather complicated. The longer-shorter-same durations would be easiest to implement. We have also enabled further analyze and quantization possibilities in our inner representation.

For the present the key has to be given by handwritten irp-code. Thus, a user-friendly interface has to be designed. This may include “piano-roll” notation, simple drafted notation and singing/whistling input. It would also be practical if the musical note information, the textual information (the lyrics) and some stylistic information (waltz, rock etc.) could be combined. That kind of information could be used as an additional discriminator.

Acknowledgements

We would like to thank professor Esko Ukkonen for his guidance and valuable comments during the project. We are grateful also to Atso Haapaniemi for all the implementational issues and his comment on this paper.

Figure 5: *The key used in the experiments. It is copied from a melody in the database.*

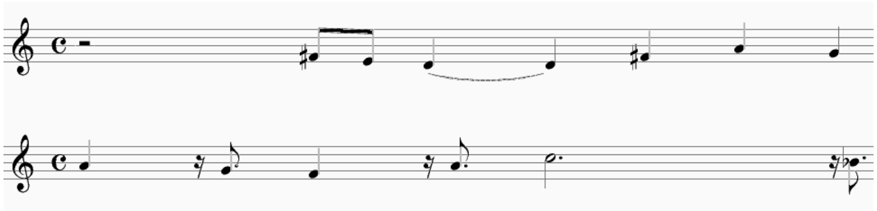

Figure 6: *Two occurrences of the given key in the database. The first (upper) is from the same melody the key is copied (another occurrence). The second is an occurrence in another melody. Note, that we do not consider the pauses for the present.*

References

[Bel97] N. Bell. A guitar to musical instrument digital interface converter using signal processing and pattern recognition techniques. Master's thesis, University of East Anglia, School of Information Systems, 1997.

[BM77] R. Boyer and S. Moore. A fast string searching algorithm. *Communications of the ACM*, (20):762–772, 1977.

[BYP92] R. Baeza-Yates and C. H. Perleberg. Fast and practical approximate string matching. In *Combinatorial Pattern Matching, Third Annual Symposium*, pages 185–192, 1992.

[CC98] 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.

[CCL96] 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.

[CR94] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[Foo97] J.T. Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II, Proceedings of SPIE*, pages 138–147, 1997.

[GLCS95] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming - musical information retrieval in an audio database. In *ACM Multimedia 95*, 1995.

[Gus97] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

[Haw93] M.J. Hawley. *Structure out of Sound*. PhD thesis, MIT, 1993.

[LHU98] K. Lemström, A. Haapaniemi, and E. Ukkonen. Retrieving music — to index or not to index. To appear in: ACM Multimedia 98, 1998.

[Lin67] H. B. Lincoln. Some criteria and techniques for developing computerized thematic indices. In Heckmann, editor, *Electronische Datenverarbeitung in der Musikwissenschaft*. Regensburg, 1967.

[MSBW97] R.J. McNab, L.A. Smith, D. Bainbridge, and I.H. Witten. The New Zealand digital library MELody inDEX. *D-Lib Magazine*, 1997.

[Nar90] E. Narmour. *The Analysis and Cognition of Basic Melodic Structures – The Implication-Realization Model*. The University of Chicago Press, 1990.

[Ple82] L. Plenckers. A pattern recognition system in the study of the cantigas de santa maria. In M. Baroni and L. Callegari, editors, *Musical Grammars and Computer Analysis*. Quaderni Della Rivista Italiana di Musicologia Modena, 1982.

[SJ98] 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. (to appear).

[Ukk93] E. Ukkonen. On-line construction of suffix-trees. Technical Report A-1993-1, University of Helsinki, Department of Computer Science, 1993.

[Wei73] P. Weiner. Linear pattern matching algorithms. In *Proceedings of IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.