

PROMS: A Web-based Tool for Searching in Polyphonic Music*

M. Clausen, R. Engelbrecht, D. Meyer, J. Schmitz
Institut für Informatik V, Universität Bonn,
D-53117 Bonn, Germany
{clausen,roland,meyerd,schmitz1}@cs.uni-bonn.de

October 4, 2000

Abstract

PROMS, a computer-music service under development at the University of Bonn, Germany, aims at designing and implementing PROCedures for Music Search. This paper discusses an efficient algorithm for searching a music pattern, e.g., a melody or a sequence of chords, in a large database of polyphonic music, given in a score-like format. Our algorithm is a variant of the now classic inverted file index approach for text retrieval. A prototype is implemented and its performance investigated. Finally, we briefly indicate how to extend this technique to a fuzzy search, to a search with up to k mismatches, and to transposition-invariant search. For demonstration, the PROMS system may be tested under the URL <http://verdi.cs.uni-bonn.de/proms>.

Key words: Digital Music Library, Music Retrieval, Fuzzy Search, Inverted File Index

1 Introduction

One major task of a digital music library (DML) is to provide techniques to locate a queried musical pattern in all pieces of music in the database containing that pattern. For a survey of several computational tasks related to this kind of data retrieval we refer to Crawford et al. [3]. Existing DMLs like MELDEX [1], Themefinder [4], and the Sonoda-Muraoka-System [7] work with melody databases relying on score-like information. Retrieval and matching are performed in a fault-tolerant way by string-based methods which mainly take into account pitch information. Generally, rhythm plays only a subordinate role. The music dictionary of Barlow and Morgenstern [2] shows that music retrieval based on pitch information only leads to results with typically too many false matches. (An example of such absurd matches is given in Selfridge-Field [6], p. 27.) We are convinced that *both* pitch and rhythm are crucial for recognizing

*This work was supported in part by Deutsche Forschungsgemeinschaft grants CL 64/3-1 and CL 64/3-4.

melodies. In the more general context of polyphonic music, one is even forced to consider pitch and rhythm information.

PROMS, a web-based computer-music service under development at the University of Bonn, Germany, is part of the MiDiLiB project [5]. The aim of PROMS is to design and to implement PROcedures for Music Search. Our discussion will take place in a rather general setting: we assume that our database contains several kinds of music such as polyphonic and homophonic music as well as melodies. We also use score-like information. A query to the database is a fragment of a piece of music. This could be a melody or a certain figure of an accompaniment. The task is now to locate efficiently all occurrences of this fragment in all pieces of music in the database. For recognizing fragments within a piece of music we state that pitch and onset time of each note suffice to describe the *musical substance* (but of course not the *musical character*) of that piece. For the moment, we will ignore note duration.

In the sequel we represent a piece of music as a set of notes where each note is described as a pair $[t, p]$ with onset time t and pitch p . Since meter is a crucial aspect of western music, we also take into account meter information. Then a piece of music P in the database matches a given query, if a time-shifted version of the query is a subset¹ of P . But even this simple representation of a piece of music leaves us with the problem of time quantization.

Arbitrary time resolution makes retrieval very difficult, because onset times can be extremely irregular. However, a fixed time-grid eases the implementation considerably. Thus onset times should be quantized to a preselected time-grid, e.g., a sixteenth-note resolution. Since both query and pieces in the database are quantized in the same way, no match of the query will be missed. On the other hand, different note combinations might have the same quantized form. Furthermore, a quantized melody might lead to chords. All this is illustrated in Fig. 1.



Figure 1: Side-effects of quantization using sixteenth quantization.

There is no need to restrict oneself to a time-grid with equidistant grid positions. One can use time-grids which divides a measure irregularly, for example, a time-grid containing grid points for both sixteenths and eighth-triplets. For the sake of simplicity, we will consider only equidistant grids in the sequel.

A pitch can be easily described by an integer. Since our pieces originate from MIDI input, we use the MIDI numbers 0..127 for pitch representation, too.

We explicitly point out that our approach also includes the retrieval in melody databases. The applied method permits novel query and retrieval strategies. On the one hand, the user is allowed to query a non-contiguous melodic fragment leaving out too complex or less familiar passages. This has no impact on the efficiency. On the other hand, the user has the possibility to specify

¹For a review of some mathematical terminology we refer to the end of this introduction.

“fuzzy notes” that reflect a certain amount of fault-tolerance with respect to pitch and time. In addition, given a query Q and a positive integer k , the PROMS system supports an efficient computation of all occurrences of Q with at most k mismatches.

Section 2 gives both a rigorous specification of musical pattern matching and a theoretical solution of the problem. Section 3 shows how to apply an inverted file index structure to obtain a fast exact musical pattern matching. Section 4 describes a prototypical implementation and its performance. In Section 5 we indicate fault-tolerant music retrieval. Section 6 discusses transposition-invariant search via Chinese Remaindering.

According to the interdisciplinary character of our topic we conclude this introduction by recalling some standard mathematical notation and terminology. A set A is said to be a *subset* of a set B , written $A \subseteq B$, if every element a of A is also an element of B , for short, $a \in A$ implies $a \in B$. The *intersection* of two sets A and B , written $A \cap B$, is a set consisting of all elements that belong to both A and B . $\bigcap_{i=1}^n A_i$ is the shorthand of the intersection $A_1 \cap \dots \cap A_n$. $A \cup B$, the *union* of A and B , consists of all elements x that belong to A , or B , or to both of them. The *direct product* of A and B , written $A \times B$, is the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$. In the sequel, we will encounter two different kinds of direct products. In order to avoid any confusion, we will use two different kinds of brackets. Thus, a note with onset time t and pitch p will be denoted by $[t, p]$, whereas (i, M) is an index entry pointing to the M th measure of piece i .

\mathbb{N}_0 denotes the set $\{0, 1, 2, 3, \dots\}$ of all non-negative integers and \mathbb{N} is the set $\{1, 2, 3, 4, \dots\}$ of all positive integers. The set of all integers from $a \in \mathbb{N}_0$ through $b \in \mathbb{N}_0$ is denoted by $[a : b]$; thus $[3 : 6] = \{3, 4, 5, 6\}$. The integer division of two positive integers a and b leads to a quotient $q = \lfloor a/b \rfloor$ and a remainder $r = a \bmod b$ satisfying $a = q \cdot b + r$ and $r \in [0 : b - 1]$. For example, the integer division of 39 by 16 gives $q = 2$ and $r = 7$, i.e., $39 = 2 \cdot 16 + 7$.

2 Musical Pattern Matching

Our database consists of N pieces of music P_1, \dots, P_N . We assume that the time signatures of all pieces are known. For simplicity we only discuss the case that all pieces have the same time signature b/u (beats per unit of measure). Furthermore, a time resolution $r \in \mathbb{N}$ is given. All pieces are quantized with respect to this resolution, e.g., $r = 16$ implies a sixteenth-note resolution. (In all examples that follow, we take $r = 16$.) It must hold that r/u is an integer. Usual selections are $r = 2u$ or $r = 4u$. The value $\ell := br/u$ is the number of time-grid positions within a measure. In the example shown in Fig. 2, the above parameters are given by $b = 3$, $u = 4$, and with $r = 16$ we get $\ell = 12$.

Each piece P_i is a finite subset of $\mathbb{N}_0 \times [0 : 127]$ where $[t, p] \in P_i$ means that at time t a note of MIDI pitch p occurs in piece P_i . Such a pair $[t, p]$ will be called an *absolute note*. The ℓ -relative version of $[t, p]$ is defined to be the pair $[t \bmod \ell, p]$, which will also be called an $(\ell$ -)relative note. Thus, the relative note $[t \bmod \ell, p]$ describes the absolute note $[t, p]$ viewed from measure $\lfloor t/\ell \rfloor$, reflecting in particular the metrical position within that measure. For example, if a waltz starts with an upbeat of a quarter note then, with $r = 16$, the onset time t of this note equals $t = 8$. With these conventions, the fragment of Beethoven’s



Figure 2: From Beethoven's Hammerklaviersonate, Scherzo

Scherzo of the Hammerklaviersonate, shown in Fig. 2, is represented by the following data:

$$P := \{[8, 74], [11, 77], [11, 69], [12, 74], [12, 72], [16, 74], [16, 65], [20, 70], [23, 74], [23, 66], [24, 74], [24, 69], [28, 70], [28, 62]\}.$$

A query Q is also a finite subset of $\mathbb{N}_0 \times [0 : 127]$ with the same semantics as above. Again, for simplicity, we assume that Q has the same meter as the pieces of the database. The task is now to list all occurrences of Q in the various pieces P_i . Note that Q may occur several times in a fixed P_i . Hence we have to specify the piece as well as the exact location where Q occurs. By definition, Q occurs in P_i at measure M if and only if

$$[M \cdot \ell, 0] + Q := \{[M \cdot \ell, 0] + q \mid q \in Q\} \subseteq P_i,$$

where the addition of notes is componentwise: $[t, p] + [t', p'] := [t + t', p + p']$. In other words, each occurrence of Q in the database is described by a pair (i, M) where M refers to the M th measure of piece P_i . Note that we work only with those time-shifts of Q that respect the metrical positions of the notes. Fig. 3 illustrates two different queries $Q_1 = \{[0, 74], [4, 70]\}$ and $Q_2 = \{[4, 74], [8, 70]\}$. As our system disregards note durations, Q_1 as well as Q_2 occur in P defined as above. However, as we only allow time shifts by multiples of ℓ (corresponding to whole measures) each of Q_1 and Q_2 occurs exactly once in P . Supposing that P is the piece P_1 of our database, the occurrence of Q_1 is described by $(1, 2)$ whereas the one of Q_2 corresponds to $(1, 1)$.

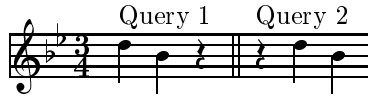


Figure 3: Query examples

In general, given a query Q , our task is to compute the set

$$\mathcal{M}(Q) := \{(i, M) \mid [M \cdot \ell, 0] + Q \subseteq P_i\}$$

of all query matches. In order to construct this list of query matches, we first collect all occurrences of notes with pitch $p \in [0 : 127]$ and metrical position $m \in [0 : \ell - 1]$ in a list $L(m, p)$, more precisely:

$$L(m, p) := \{(i, M) \in [1 : N] \times \mathbb{N}_0 \mid [M\ell + m, p] \in P_i\}. \quad (1)$$

Now let $Q := \{[t_1, p_1], \dots, [t_n, p_n]\}$ be a query with $t_k = \ell \cdot M_k + m_k$, $M_k = \lfloor t_k / \ell \rfloor$, and $m_k \in [0 : \ell - 1]$. Then the list of matches may be obtained by the following intersection:

$$\mathcal{M}(Q) = \bigcap_{k=1}^n (L(m_k, p_k) - (0, M_k)), \quad (2)$$

where $(L(m_k, p_k) - (0, M_k)) := \{(i, M - M_k) \mid (i, M) \in L(m_k, p_k)\}$. We prove this claim through the following chain of equivalences:

$$\begin{aligned} (i, M) \in \mathcal{M}(Q) &\iff [M \cdot \ell, 0] + Q \subseteq P_i \\ &\iff \{[(M + M_k) \cdot \ell + m_k, p_k] \mid k \in [1 : n]\} \subseteq P_i \\ &\iff (i, M + M_k) \in L(m_k, p_k), \text{ for all } k \in [1 : n] \\ &\iff (i, M) \in L(m_k, p_k) - (0, M_k), \text{ for all } k \in [1 : n] \\ &\iff (i, M) \in \bigcap_{k=1}^n (L(m_k, p_k) - (0, M_k)). \end{aligned}$$

This result constitutes the theoretical basis of an efficient solution of the music pattern matching problem, discussed in the sequel.

3 Inverted File Index

A standard index technique for a large text database is an inverted file index. An inverted file contains, for each word w appearing in the database, a list $L(w)$ of references to the documents containing that word [9]. A typical query like "word₁ AND ... AND word_n" is processed similar to equation (2) by computing the intersection of inverted lists for each query-term (word):

$$\mathcal{M}(Q) := \bigcap_{k=1}^n L(\text{word}_k). \quad (3)$$

For large document collections the inverted lists are stored on disk. A table, called vocabulary, stores for each word the disk-location of its inverted list $L(\text{word}_k)$. To process a query, each query-word is looked up in the vocabulary; the inverted list is read from the disk and the lists are merged, taking the intersection of the sets of referenced documents. In most cases, inverted lists are very large and have to be processed successively, keeping only the result of the last intersection in main memory.

Since in the worst case an inverted file index can be as large as the whole database, it is usually compressed to save disk storage. To this end, references are not stored absolutely as a document-address, but relatively to the last reference. For example, the inverted list $L(\text{example}) = \langle 5, 11, 13 \rangle$ is stored in relative form as $L'(\text{example}) = \langle 5, 6, 2 \rangle$. The two forms are equivalent, but in practice the numbers in the relative form are much smaller than in the absolute form. Additionally, these numbers can be *Golomb-coded* in order to achieve a better compression. For more details we refer to the next section. A compressed inverted file usually requires approximately 30% disk space of the whole database.

Compared to our music database, a word in a text corresponds to a relative note. That means, each absolute note $[t, p]$ corresponds to the “word” $[t \bmod \ell, p]$. With this notion, we can implement our music retrieval system based on an inverted file index.

There are two differences between text and music retrieval. Firstly, the vocabulary for text databases can grow with new documents. The size of the music-vocabulary, however, does not grow with new documents: there are $128 \cdot \ell$ inverted lists. Secondly, for text retrieval, the relative position of the queried words in the retrieved text document is not important as indicated by Eq. (3). For music retrieval, the situation is quite different. If a query Q involves more than one measure, the lists in question have to be preprocessed according to Eq. (2) before intersecting the modified lists. This is a variant of the normal query processing of inverted files.

4 Implementation

In this section we describe some details of our music retrieval system and report on running times and memory requirements.

Our music database consists of over 12,000 classical pieces of music in the MIDI file format. Most of these pieces are interpretations played by various artists, made available via internet. In general, the MIDI files are not monophonic and not explicitly structured into musical components like melodies, themes, chords, etc. The time signature, which is essential for our approach, is known for each MIDI file.

We apply inverted files to index our music database. As mentioned in Section 3, an inverted list exists for each combination of pitch and metrical position. Thus, we have $128 \cdot \ell$ lists $L(m, p)$, see Eq. (1). In practice, however, a substantial portion consists of empty lists.

Recall that our inverted lists consist of pairs (i, M) referring to measure M in piece P_i . In order to apply the relative form for storing the inverted files, we have to code each pair into one unique number. This can be accomplished as follows. Let m_i denote the total number of measures in P_i , for all $1 \leq i \leq N$. Now assume all pieces P_1, \dots, P_N of our database are concatenated to one large data stream P , and assume all measures of P and the original MIDI files are subsequently numbered, starting with 0. Then the m_1 measures of P_1 correspond to measure $0, \dots, m_1 - 1$ of P , and the m_2 measures of P_2 correspond to measure $m_1, \dots, m_1 + m_2 - 1$ of P . Generally speaking, P_i corresponds to the part of P which starts with measure $m_1 + \dots + m_{i-1}$, and ends with measure $m_1 + \dots + m_i - 1$. Now given a position table which stores for each MIDI file i its starting position s_i in P , it is a rather easy task to determine the original pair (i, M) from its corresponding measure M' in P . Namely, we can use a *binary search* to find the index i of the piece P_i such that $s_i \leq M' < s_{i+1}$ holds. Then, in a second step, M can be determined from the fact that $M = M' - s_i$. Using this strategy, we are left with a homogeneous list of integers which is easy to handle and yields good compression results.

Our database of MIDI files requires 327 MB of disk space. This amount includes besides notes also tempo, meter, key-signature, text, and other data. These pieces of music consist of more than 33 million notes. Our uncompressed inverted file index requires 110 MB space. Compression of the inverted list with

a	4	6	8	10	12	14	16	18	20	30	50	100
b	51	70	86	87	92	93	97	96	100	107	125	159
c	1	3	5	6	7	8	10	11	12	19	31	64

Table 1: The average total system response time (row b) in milliseconds depending on the number of notes in the query (row a). Row c describes the time required for disk access to fetch inverted lists. Most of the remaining time (total response time - disk access time) is consumed by decompressing inverted files (Pentium II, 333 MHz, 256 MB RAM, Windows NT 4.0).

Golomb coding [9] shrinks the space demand to 22 MB.

The time to process a query depends mainly on the query length and on the code chosen for compression. Our experimental results are similar to those in [9]. To speed up the decompression process we use a special Golomb coding vector which only yields a suboptimal data compression, but the decoding routine is substantially simplified (using bit-shifting techniques) while at the same time the compression ratio is only slightly reduced.

Typical response times for queries of various length are summarized in Table 1. The response times for each query length were averaged over 100 randomly generated queries.

As the table demonstrates, our query processing is very fast. In addition, the index requires only a small amount of disk space. Hence, our approach seems to be applicable to much larger music databases.

5 Fault-tolerant Music Search

In the previous sections we dealt with exact query retrieval. Now we describe first steps towards a fault-tolerant search in music databases which also is based on inverted files.

As already mentioned in the introduction, our inverted file index supports incomplete queries in the sense that missing notes, e.g., in the middle of a melody, are allowed. For example, one could omit the sixteenth-triplet in the melody depicted in Fig. 4. While omitted notes are one type of fault tolerance, another one is the fuzzy specification of notes. In this case the user only knows the rough time or pitch intervals of queried notes. For example, someone who has the first theme of Elgar’s Pomp and Circumstances (Fig. 5) in mind, may be not sure whether the first sixteenth group consists of the notes (b, d, a, b) or $(b, c\#, a, b)$.

More generally, our retrieval system permits the user to specify (instead of a query Q consisting of a sequence (q_1, \dots, q_n) of notes) a *fuzzy query* \mathbf{Q} which consists of a sequence (Q_1, \dots, Q_n) , where each Q_k is a finite, nonempty set of absolute notes. If Q_k contains more than one element, this might reflect the user’s uncertainty. The set of matches corresponding to such a fuzzy query is defined as

$$\mathcal{M}(\mathbf{Q}) := \{(i, M) \mid \text{for suitable } q_k \in Q_k : [M \cdot \ell, 0] + \{q_1, \dots, q_n\} \subseteq P_i\}.$$

In analogy to Eq. (2) we get (with $q =: [t_q, p_q]$ for an arbitrary absolute note q)

$$\mathcal{M}(\mathbf{Q}) = \bigcap_{k=1}^n \left[\bigcup_{q \in Q_k} (L(t_q \bmod \ell, p_q) - (0, \lfloor t_q / \ell \rfloor)) \right]. \quad (4)$$

Compared to an exact query, the running time for a fuzzy query search is increased at most by a factor proportional to $|Q_1| + \dots + |Q_n| - n$, where $|X|$ denotes the number of elements in the set X . If $|Q_k| = 1$ for all k then the fuzzy search reduces to an exact search.



Figure 4: From Beethoven's piano sonata in F minor, op. 2, no. 1.



Figure 5: From Elgar's Pomp and Circumstances.

Every fuzzy query $\mathbf{Q} = (Q_1, \dots, Q_n)$ defines the set

$$\Pi(\mathbf{Q}) = \{\{q_1, \dots, q_n\} \mid q_i \in Q_i, \text{ for all } i \in [1 : n]\}$$

of corresponding exact patterns. Obviously, any probability distribution on $\Pi(\mathbf{Q})$ leads to a ranking of the set $\mathcal{M}(\mathbf{Q})$ of all fuzzy matches.

Another type of fault-tolerance is to allow up to k mismatches, $k \in \mathbb{N}_0$ fixed. Given an exact query Q with n notes, the set

$$\mathcal{M}_k(Q) := \{(i, M, \kappa) \mid 0 \leq \kappa \leq k, ([M \cdot \ell, 0] + Q) \cap P_i \text{ contains } n - \kappa \text{ elements}\}$$

can be computed efficiently. The parameter κ leads in a natural way to a ranking of $\mathcal{M}_k(Q)$.

Finally, one can combine fuzzy queries, probability distributions, and k -mismatches to obtain an even better ranking. The technical details will be reported elsewhere.

6 Transposition Invariant Search

One desirable retrieval option is to find matches which are transpositions of the query. In the basic version of our retrieval system, transposed queries were not

matched. A brute force strategy to retrieve transposed matches is to look up all potential transpositions. As there are 128 different pitches in MIDI, this strategy reduces the time efficiency by a factor of 128 in the worst case. As an alternative, we sketch now a more efficient procedure based on the Chinese Remainder Technique. Using this technique, we get along with only 16 slightly modified queries applied to three smaller indexes.

For reasons that will become clear in a moment, we will use the interval $[0 : 139]$ to describe the 128 different MIDI pitches. In this context, the numbers $128, \dots, 139$ have no interpretation as pitch, but working with this larger interval has some advantages, to be described next.

In general, if $n \geq 2$ is an integer, $\mathbb{Z}/n\mathbb{Z}$ denotes the interval $[0 : n - 1]$ in which addition and multiplication are performed modulo n . Now the classical Chinese Remainder Theorem states that for a sequence n_1, \dots, n_r of pairwise coprime integers there is an isomorphism

$$\Phi: \mathbb{Z}/n\mathbb{Z} \xrightarrow{\sim} \mathbb{Z}/n_1\mathbb{Z} \times \dots \times \mathbb{Z}/n_r\mathbb{Z},$$

where $n = n_1 \cdots n_r$. This isomorphism Φ is given by

$$\Phi(m) := (m \bmod n_1, \dots, m \bmod n_r).$$

Addition and multiplication on the r.h.s. is performed componentwise. Thus, Φ is a bijective mapping, satisfying

$$\Phi(m_1 + m_2) = \Phi(m_1) + \Phi(m_2), \quad \Phi(m_1 \cdot m_2) = \Phi(m_1) \cdot \Phi(m_2),$$

for all $m_1, m_2 \in [0 : n - 1]$. By the way, there are efficient algorithms to evaluate both Φ and its inverse, see, e.g., [8].

In our special situation, we take $n_1 = 4$, $n_2 = 5$, and $n_3 = 7$, to obtain

$$\mathbb{Z}/140\mathbb{Z} \simeq \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/5\mathbb{Z} \times \mathbb{Z}/7\mathbb{Z}.$$

Instead of generating a single index, we create three “modular” indexes: I_4 , I_5 , and I_7 . These indexes are obtained from the original index by replacing each pitch number p by $p \bmod 4$, $p \bmod 5$, and $p \bmod 7$, respectively, followed by eliminating all multiplicities. (Note that different pitches might lead to the same “modular” pitch.) Therefore, each modular index is smaller than the original one. In our case, I_4 , I_5 , and I_7 consume altogether about 37 MB, i.e., the storage requirements increase only by a factor of 1.7.

For a query Q , let $M^t(Q)$ denote the set of all matches of pitch-transposed versions of Q in the database. To compute $M^t(Q)$, we first compute the modular versions Q_4 , Q_5 , and Q_7 of Q by taking the pitches modulo the appropriate modulus. Now observe that in $\mathbb{Z}/n_i\mathbb{Z}$ there are at most n_i different admissible transpositions. Hence, we get along with at most $4 + 5 + 7 = 16$ modular queries that can optionally be computed in parallel. This results in 16 modular sets of matches $M_{p,\pi}(Q_p)$ with $p \in \{3, 5, 7\}$ and $0 \leq \pi < p$. Next, we compute all

$$M_\tau := M_{4,\tau_4}(Q_4) \cap M_{5,\tau_5}(Q_5) \cap M_{7,\tau_7}(Q_7),$$

where $\tau := (\tau_4, \tau_5, \tau_7)$, $0 \leq \tau_p < p$. Note that similar to the brute force method we have to construct 128 M_τ 's, this time, however, the operations involve comparatively small sets. Finally, the set $M^t(Q)$ is just the union of all M_τ .

7 Conclusions

We have shown how to perform exact polyphonic musical pattern matching efficiently by using an inverted file index. Furthermore, we demonstrated how to extend our approach to a fault-tolerant music search, including fuzzy techniques, k -mismatches and transposition-invariant search.

8 Acknowledgements

We would like to thank Heiko Goeman, Frank Kurth, Meinard Müller and Klaus-Dieter Schneider for their suggestions, critique, encouragement and their invaluable assistance.

References

- [1] David Bainbridge. *MELDEX: A Web-based Melodic Index Service*. In: *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 12, pages 223–230. MIT Press, 1998.
- [2] Harold Barlow and Sam Morgenstern. *A Dictionary of Musical Themes*. Crown Publishers, 1948.
- [3] Tim Crawford, Costas S. Iliopoulos, and Rajeev Raman. *String-Matching Techniques for Musical Similarity and Melodic Recognition*. In: *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 3, pages 73–100. MIT Press, 1998.
- [4] Andreas Kornstädt. *Themefinder: A Web-based Melodic Search Tool*. In: *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 13, pages 231–236. MIT Press, 1998.
- [5] MiDiLiB project. Content-based indexing, retrieval, and compression of data in digital music libraries.
<http://leon.cs.uni-bonn.de/forschungsprojekte/midilib/english/>.
- [6] Eleanor Selfridge-Field. *Conceptual and Representational Issues in Melodic Comparison*. In: *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 1, pages 3–64. MIT Press, 1998.
- [7] Tomonari Sonoda and Yoichi Muraoka. A www-based melody-retrieval-system — an indexing method for a large melody database. In *Proc. ICMC 2000, Berlin*, pages 170–173, 2000.
- [8] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
- [9] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Van Nostrand Reinhold, 1994.