SEMEX – An Efficient Music Retrieval Prototype

Kjell Lemström, Sami Perttu
Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
FINLAND
{klemstro,perttu}@cs.helsinki.fi

ABSTRACT

We present an efficient prototype for music information retrieval. The prototype uses bit-parallel algorithms for locating transposition invariant matches of monophonic query melodies within monophonic or polyphonic music stored in a database. When dealing with monophonic music, we employ a fast approximate bit-parallel algorithm with special edit distance metrics. The fast scanning phase is succeeded by verification where a separate metrics is used for ranking matches. We also offer the possibility to search for exact occurrences of a "distributed'' melody within polyphonic databases via a bit-parallel filtering technique. In our experiments with a database of 2 million musical elements (notes in a monophonic and chords in a polyphonic database) the responses were obtained clearly within one second in both the cases. Furthermore, our prototype is capable of using various interval classes in matching, producing more approximation when it is needed.

1. INTRODUCTION

The origins of music information retrieval (MIR) systems are in manual collections of *incipits*, short melodic fragments obtained from the beginning of pieces of music. The collections were manually compiled by researchers or librarians, and usually covered one narrow field of music. More recently, computerized content-based retrieval systems have begun to appear. The systems interpret melodies as symbol sequences that can be matched against query patterns using standard methods from general string matching (Crochemore and Rytter, 1994; Gusfield, 1997).

In this paper , we introduce our MIR prototype called SEMEX (Search Engine for Melodic Excerpts), which is an efficient implementation of various ideas that we have introduced in the literature (for a summary see (Lemström, 2000)). The cores of the matching algorithms in SEMEX are based on so-called *bit-parallelism.* We claim that the length of an "ideal'' MIR query pattern is shorter than the usual length of the computer word (which is usually 32 or 64 bits). In such case, the bit-parallel cores of our algorithms run in linear time with respect to the size of the database. Such fast implementations are of crucial importance since music databases can be very large. Not only is the extensive use of bit-parallelism something new for a MIR system, but SEMEX also differs from previous systems in other respects, discussed next. In the Appendix we give a comparison table for some current MIR systems.

Some MIR systems accept digital input which is subsequently converted into an internal representation for matching (Ghias et al., 1995; Bainbridge et al., 1999; Pollastri, 1999; Rolland et al., 1999). An apparent problem here is that the digital input is likely to be somewhat distorted. For instance, if a query is given by humming, one cannot expect the intervals to be exactly correct. To overcome this problem, the music is often represented by *contour*, giving only the direction of the intervals (up, down, or repeat). This representation, however, requires long query patterns in order to be discriminative enough (Downie, 1999). In SEMEX, we can achieve better discrimination while still maintaining tolerance to errors by using the *QPI classification* of Lemström and Laine (1998).

In some of the MIR systems durational information can be used in queries, but the present representation of choice is a symbolic sequence of pitches. There are several reasons for this: Firstly, in Western music pitch information is more discriminative, since 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 (Cronin, 1998)). Secondly, 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. Furthermore, although by using only the pitch information loses some characteristics of the music, it certainly facilitates the modeling of polyphonic music[2].

We say that a pitch sequence is an *absolute representation* of music. When dealing with queries on pitches, *transposition invariance* is an important property to be considered. An algorithm taking into account transposition invariance usually

[2] That is, music where more than one note happens at once.

ignores the *musical key* of the pitch sequence by using an *interval representation* that considers the semitonic pitch distances between consecutive elements of the absolute representation. The interval representation, however, involves some known problems (Cambouropoulos et al., 1999). Therefore, in SEMEX, we apply Lemström and Ukkonen's (2000) transposition invariant distance metrics that calculate the intervals "on-the-fly'' from absolute representation and thus avoid some of the shortcomings.

A property of real music databases often completely ignored in MIR systems is that the databases are likely to contain polyphony. Although a few algorithms for locating occurrences of query patterns in polyphonic databases have been presented recently in the literature (Uitdenbogerd and Zobel, 1998; Dovey, 1999; Holub et al., 1999; Lemström and Tarhio, 2000), to our knowledge, besides SEMEX, only Uitdenbogerd and Zobel's system deals with this problem. However, the approaches of these two systems are very different. Uitdenbogerd and Zobel reduce the polyphonic surface to a monophonic melody passage by a melody extraction algorithm, while in SEMEX it is possible to locate occurrences that are distributed among several voices by using the *MonoPoly algorithm* of Lemström and Tarhio (2000).

In SEMEX, all the properties mentioned above have been implemented by using an algorithmic technique, called bit-parallelism, that uses word-level bit-vector operations of the hardware.

The rest of this paper is organized as follows. In the following Section we give some background for this study. First we briefly review the related work and then we define our MIR problem. Having defined our general framework based on string matching and edit distance in Section 2, in Section 3 we briefly describe our ideas and show how they can be applied using bit-parallelism, and thus, included to our prototype. Before concluding the paper, Section 4 introduces our SEMEX prototype.

2. BACKGROUND

One of the first measures for comparing two musical sequences was presented by Mongeau and Sankoff (1990). Later, in his PhD thesis, Hawley (1993) briefly discussed about the idea of making content-based queries over a MIDI database. Since those times, interest in techniques for matching and retrieving melodies has increased greatly, which can be seen in the increasing volume of publications concerning the topic. We refer the reader to (Crawford et al., 1998) and (Lemström, 2000) for a summary of the current state-of-the-art.

In the literature, one can find not only several different algorithms that have been used in MIR, but also whole MIR systems using different approaches (see e.g. (Ghias et al., 1995; Bainbridge et al., 1999; Tseng,, 1999; Uitdenbogerd and Zobel, 1999)). In these systems, the matching is usually based either on dynamic programming or on *the q-gramming technique*. When the source is very large, dynamic programming becomes slow due to the $O(nm)$ time complexity where n and m denote the length of the database and the query pattern, respectively. In q-gramming the source is observed as blocks of symbols that are of length q each. The blocks can be stored in an indexing structure, allowing fast retrieval. However, if the database is large, there might not be enough disc capacity to maintain a huge index structure. Moreover, updating such a structure is always somewhat problematic.

The dynamic programming approach, however, can be speeded up by using bit-parallel algorithms. Recently, Myers (1998) presented an elegant bit-parallel dynamic programming algorithm whose scanning capacity with the present computers is clearly over 10 MB/sec). Thus, if the information of one note can be stored in a byte, at least 10^7 notes can be scanned within a second.

2.1 Related Work

Let us briefly describe some of the current MIR systems in a chronological order. Ghias et al.'s (1995) MIR prototype accepted input via microphone and used contour in matching. Moreover, a dynamic programming algorithm was used to locate matches.

Meldex is the New Zealand Digital Library's Web-based[3] melody retrieval system (McNab et al., 1997). Digital wave input 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. Close matches are ranked and displayed. Pitch is matched either by chromatic intervals or by contour. A faster *k-mismatches* algorithm or a slower dynamic programming algorithm augmented with Mongeau and Sankoff's (1990) editing operations for music comparison. Recently, an optical music recognition facility and text-based queries have been added to the system (Bainbridge et al., 1999).

Themefinder[4] (Kornstädt, 1998), another Web-based melody search tool, is a closer successor to the thematic catalogues than 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 meter. Only exact matching is available.

Pollastri's prototype (1999) provides a pitch tracking module and several 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 the alphabet. The matching is based on Chang and Lampe's algorithm (1992), which is found to be faster than Ukkonen's cutoff algorithm (1985a) in their experiences.

Uitdenbogerd and Zobel (1999) performed MIR in a database of monophonic melodies extracted from polyphonic MIDI files. They considered different schemes of melody extraction as well

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

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

as different reductions of the melodic surface, such as contour. For matching, a comparison between four different similarity measures was conducted; edit distance was found to perform best.

Melodiscov (Rolland et al., 1999) 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 Rolland's FlExPat algorithm (1999) in their system.

2.2 Problem Setting

Before defining our problem, let us recall some preliminaries of string combinatorics. A sequence (string) $A=a_1 \ldots a_n$ is a sequence of n (zero or more) symbols over an alphabet Σ. The set of all sequences over Σ. is denoted by Σ^*. The length of the string X is denoted as $|X|$. Let the sequence X be of form $X=\alpha\beta\gamma$, where α, β, γ are in Σ^*. We say that α is a *prefix* of X, and γ a *suffix* of X. Moreover, α, β, and γ are *substrings* of X, while a string X' is a *subsequence* of X if it can be obtained from X by deleting zero or more symbols, i.e., $X'=x_{i_1} x_{i_2} \ldots x_{i_m}$, where $i_1 \ldots i_m$ is an increasing sequence of indices in X.

In our current prototype, we use a simplified representation for music comprising only the pitch levels of the notes. The pitch levels are represented as small integers $0, \ldots , r$ which form our alphabet Σ. Here, four values of r are of particular interests; $r=2$ (representing musical contour); $r=10$ (representing the QPI classification); $r=11$ (representing octave equivalence); and $r=127$ (as in MIDI (MIDI Manufacturers Association, 1996)). [5]

A musical *source* $S=S_1 S_2 \ldots S_n$ is a sequence of sets of integers. Each S_i corresponds to a chord of notes, and is comprised of notes having their onsets simultaneously. Each S_i is therefore formally a subset of the underlying alphabet.

A musical *query pattern* $p=p_1 p_2 \ldots p_m$ is a sequence of integers, more precisely, $p_i \in \Sigma$, for $i=1,\ldots, m$. Obviously, the degenerate case for S (that is, when the source is monophonic), denoted by s, is represented by a similar structure to that of the pattern representation.

The setting for our MIR problem is as follows. Given a long source sequence $S=S_1 S_2 \ldots S_n$ ($s=s_1 s_2 \ldots s_n$) and a relatively short music query pattern $p=p_1 p_2 \ldots p_m$, the task is to find all locations in S (or s) where p occurs as a substring. Here an occurrence might mean an *exact*, *transposed*, or even *approximate occurrence*.

[5] Since the first two alphabets actually comprises symbols associated with intervals, Lemström and Ukkonen's (2000) distance metrics (mentioned above and defined below) are applicable only with the latter two alphabets.

When the source s is monophonic, we define that there is an exact occurrence of p at position j, if $p_i = s_{j+i-1}$ for $i=1,\ldots,m$, and there is a *transposition invariant occurrence* of p at position j, if there is an integer d such that each $(p_i + d) = s_{j+i-1}$. An 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 transposed occurrence is defined accordingly). We present the conventional editing operations in the next section.

In the case of polyphonic sources, exact and transposition invariant occurrences of p within S mean that $p_i \in S_{j+i-1}$ and $(p_i + d) \in S_{j+i-1}$ for each i, respectively. Currently, we do not allow approximate queries to polyphonic sources.

3. THE FRAMEWORK: APPROXIMATE STRING PATTERN MATCHING

3.1 Edit Distance

Approximate string pattern matching is based on the concept of *edit distance*, also known as *Levenshtein distance or evolutionary distance* (Crochemore and Rytter, 1994; Gusfield, 1997). This well-known metric has been used to measure the dissimilarity between two strings. The edit distance $D(A,B)$ between strings $A=a_1 \ldots a_m$ and $B=b_1 \ldots b_n$, where $A, B \in\Sigma^*$, is the minimum cost of a sequence of editing steps required to convert A into B.

In general, the operations for the steps are drawn from a *local transformation set* T, where $T \in \Sigma^* \times \Sigma^*$, is a finite set of ordered pairs of sequences (α,β) in $\Sigma^* \times \Sigma^*$. The pair (α,β) corresponds to a rewriting rule often written as $\alpha \rightarrow \beta$. Moreover, for each rewriting rule a cost function w is associated: $w(\alpha\rightarrow\beta) \geq 0$. The local transformation set T of edit distance $D(A,B)$ contains replacement, insertion, and deletion operations. Let λ denote an empty string. The editing operations are defined as follows (the corresponding rewriting rules are in parentheses):

1. Replacing the symbol a_j by another symbol b gives: $a_1 \ldots a_{j-1}\, b\, a_{j+1} \ldots a_m\ (a \rightarrow b);$

2. Inserting a symbol b at position j gives: $a_1 \ldots a_j\, b\, a_{j+1} \ldots a_m\ (\lambda \rightarrow b);$

3. Deleting the symbol a_j gives: $a_1 \ldots a_{j-1}\, a_{j+1} \ldots a_m\ (a \rightarrow\lambda).$

The conventional edit distance uses unit costs: $w(a\rightarrow b) = w(\lambda\rightarrow b)= w(a\rightarrow\lambda) = 1$ (naturally, $w(a\rightarrow a) =0$). The special case in which deletions and insertions are not allowed is called *Hamming distance*, and denoted $D_H(A,B)$. In this case equal length of the strings is required.

3.2 Pattern Matching

Denoting by p, s, and k the pattern, the text (source), and the approximation threshold, respectively, in approximate pattern matching one task is to locate substrings p' of s, such that $D(p,p') \leq k$. The calculation $D(p,p')$ can be evaluated efficiently using dynamic programming, that is, tabulating all distances $d_{ij}=(p_1 \ldots p_i, s_{j'} \ldots s_j)$ (where $j' \leq j$) between the prefix of p and any substring of s (Sellers, 1980; Ukkonen, 1985a). Given the initializations $d_{0j}=0$ $(0 \leq j \leq n)$; $d_{i0}=i$ $(0 \leq i \leq m)$ of a distance table (d_{ij}), it can be evaluated in time $O(mn)$ from the well-known recurrence

$$d_{ij} = \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + (if\ p_i = s_j\ then\ 0\ else\ 1) \end{cases}$$

The retrieval results can be read from the last row d_{mj}, $0 \leq j \leq n$, of the table (d_{ij}). Each value d_{mj} is associated with the smallest possible value of $D(p,p')$ where p' ends at location j. A well-known and useful property of the table (d_{ij}) is given by the following theorem (Ukkonen, 1985b).

Theorem 1

The table (d_{ij}) has the property that the difference between adjacent elements (in vertical, horizontal, and diagonal directions) are -1, 0, or 1. Moreover, the value along each diagonal does not decrease.

As a consequence of Theorem 1, string pattern matching can be implemented in $O(kn)$ time (Ukkonen, 1985b). However, by making good use of the fact that each bit in a computer word can be used to represent, e.g., a Boolean variable, one can obtain even better performance.

Bit-Parallelism

The development of dynamic programming methods computing the edit distance has lead to an algorithm family applying a fast bit-parallel approach. In such case the conventional implementation where each variable resides in its own memory location or register is replaced by an implementation where the variables can be manipulated in parallel with bit operations, achieving a considerable speed-up. One of the first algorithms of the family was Baeza-Yates and Gonnet's (1992) shift-or algorithm designed for exact matching. Let us take a closer look at their algorithm, for in the sequel we will demonstrate our ideas by modifying this algorithm.

```
Algorithm 1 (original shift-or)
begin
1. for each a ∈ Σ do T[a] := 2^m - 1 ;
2. for j := 1 to m do T[p[j]] := T[p[j]] - 2^(j-1) ;
3. E := 2^m - 1 ;
4. for i := 1 to n do
5.    E := shiftleft(E) | (T[s[i]]);
6.    if E.m = 0 then write(i);
end
```

First, in the preprocessing phase (lines 1–2), an auxiliary table T is created. In T one bit-mask column is allocated for each symbol appearing in the pattern. For a start, these bit-masks are covered with ones. Then for each bit-mask corresponding to a symbol of the alphabet, zeros are assigned to the positions where that particular symbol occurs within the pattern. In the scanning phase, a zero is released to level 1 at each stage (by the *shiftleft* operation). The released zeros either die or survive to the next level, depending on the bit-mask used by the *or-operation* (in line 5). If a zero reaches the level m, an occurrence is found. We illustrate in Figure 1 the computation of the shift-or algorithm.

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 1. An example of the shift-or algorithm for *s=abcaaab* and *p=aab*. Note that the shifting is done downwards in the figure. In the example, one occurrence is found (encircled).

Denoting by W the number of bits in a computer word used, the shift-or algorithm runs in time $O(\lceil \frac{m}{W} \rceil n)$, i.e., in linear time if $m \leq W$.

Recently, Myers (1998) introduced a fast bit-parallel implementation for Levenshtein distance pattern matching. It also runs in time $O(\lceil \frac{m}{W} \rceil n)$. In Myers' algorithm, whose performance rests on Theorem 1, the column to be computed in the (d_{ij}) table is represented by two bit mask variables which store positive and negative changes (deltas) in the table values, respectively. There are other alternatives, such as the BNDM algorithm of Navarro and Raffinot (1998) but the independence of k in Myers' algorithm is attractive in that it offers robust performance.

Even though it is necessary to put some restrictions on the set of local transformations and on the cost function w when dealing with bit-parallel algorithms, the obtained fast performance is of crucial importance in MIR applications working on large databases. The other obtained advantage is flexibility; all our ideas, that are presented next, are implementable without extra overhead.

4. MODIFYING THE EDIT DISTANCE FRAMEWORK TO MEET MIR REQUIREMENTS

4.1 Context-Sensitive Cost Functions

When using the edit distance framework on interval representation, transposition invariance is obtained by definition. However, such application is somewhat questionable, as editing one note has an effect on two intervals, which makes the standard operations on notes more costly than it would be in applying the framework on absolute representation. Furthermore, each editing operation involves modulation, that is, a change in the musical key, which can result in perceptually large changes in a melody (Cambouropoulos et al., 1999).

It is not necessary, however, to make an explicit conversion from absolute representation to interval representation in order to obtain transposition invariance. Lemström and Ukkonen (2000) presented a modified framework for sequence comparison and approximate pattern matching. They augmented the edit distance concept to allow *context-sensitive cost functions*. The sensitiveness for the context is a property used in practice in MIR applications due to many different contexts (e.g. tonal, harmonic, and rhythmic) that are present in music (Rolland and Ganascia, 1996; Shmulevich and Povel, 1998; Shmulevich et al., 2001). In Lemström and Ukkonen's framework the cost function for local transformations is defined in such a way that the resulting distance becomes transposition invariant.

Let us now briefly present their idea. Let $A=\mu\varphi a\varphi'\mu'$ where $|\varphi|=u$ and $|\varphi'|=v$. Then it is said that (φ,φ'), is the (u,v) context of a. Moreover, w is a *cost function with (u,v) context* if w is an integer valued function defined on $T \times \Sigma^u \times \Sigma^v \times \Sigma^u \times \Sigma^v$, where Σ^x denotes the set of sequences of length x over Σ. Now, if the rewriting $\alpha \rightarrow \beta$ has been used and the (u,v) contexts of α and β are (φ,φ') and (ρ,ρ') respectively, then $w(\alpha \rightarrow \beta,\varphi,\varphi',\rho,\rho')$ is the cost associated with this particular occurrence of $\alpha \rightarrow \beta$.

Now, let $\overline{A}$ and $\overline{B}$ denote the interval encoded versions of the absolute encoded strings A and B, respectively, and let $\overline{D}(A,B)$ be a distance similar to $D(A,B)$ but using a context sensitive weight for the rule $\alpha \rightarrow \beta$. In this case, we consider the $(1,0)$ context of a and b and denote these contexts as (a',λ) and (b',λ), respectively. The particular weight is then defined as follows:

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

The obtained advantage is that there is no need anymore to transform absolutely encoded sequences into interval representation since $\overline{D}(A,B)=D(\overline{A},\overline{B})$ for every A and B. Moreover, because the intervals are now calculated *on-the-fly*, a deletion or insertion does not transpose the rest of the sequence as in the case of working with interval encoded sequences.

4.2 D_N Distance

Lemström and Ukkonen presented also another novel measure based on absolute representation. The D_N -measure combines advantages of both the interval and absolute representations into a single measure; it always uses the approach which locally results in a better match. In this measure the weight given in the previous equation is replaced by

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

It is guaranteed that $D_N(A,B) \leq D(A,B)$ and $D_N(A,B) \leq \overline{D}(A,B)$, 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.

Applying D_N to Pattern Matching

Let us now sketch how D_N measure is implemented using bit-parallelism. The illustration here adapts the shift-or algorithm because of its intuitive-enough nature. Adapting the more complex Myers' algorithm is analogous but much less intuitive.

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:

$$d_{ij} = d_{i-1,j-1} + \begin{pmatrix} if\ (p_i - p_{i-1} = s_j - s_{j-1})\ or\ p_i = s_j \\ then\ 0\ else\ 1 \end{pmatrix} \quad (1)$$

In this case two auxiliary bit-arrays are needed, including T (recall the shift-or algorithm) and an extra bit-array Δ of size $|\Sigma| \times m$. For each interval l between successive notes 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 zero-cost transposition, the algorithm uses a bit-vector J having a zero only as its least significant bit.

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

`Algorithm 2 (shift-or applying` $D_{N'}$`)`

begin

1. **for each** $a \in \Sigma$ **do** $T[a] := 2^m - 1$;
2. **for each** $d \in \Sigma$ **do** $\Delta[d] := 2^m - 1$;
3. J:= $2^m - 1$; $J := J-1;$

4. **for** $i := 1$ **to** $m\text{-}1$ **do** $T[p[j]] := T[p[j]] - 2^{j-1}$;
5. $\quad T[p_i] := T[p_i] - 2^{i-1}$;
6. $\quad \Delta[\,\overline{p}_i\,] := \Delta[\,\overline{p}_i\,] - 2^{i}$;
7. $T[p_m] := T[p_m] - 2^{m-1}$;
8. $E := 2^m - 1$; $E := shiftleft(E) \mid (T[s_1]$ & $J)$;
9. **for** $j := 2$ **to** n **do**
10. $\quad$ E := shiftleft(E) | (T[s[j]] & $\Delta[s_j - s_{j-1}]$ & J);
11. $\quad$ **if** $E.m = 0$ **then** *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 (1) achieved by taking a bitwise *and* of the bit-masks $T[s_1]$ and $\Delta[s_j - s_{j-1}]$ (line 10), because the bitwise *and* preserves the zeros. 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 complexity $O(\lceil \frac{m}{W} \rceil n)$ of shift-or.

4.3 QPI

A natural way to allow error tolerance in a query-by-humming setting is to reduce the size of the underlying alphabet. In this way, while the error tolerance is increased, the discrimination ability is at the same time reduced. For instance, the contour alphabet comprising only 3 symbols implies poor performance for MIR queries (Downie, 1999).

Lemström and Laine (1998) presented a concept of quantized partially overlapping intervals, or *QPI*, for short. The idea is adapted from the implication-realization model of Narmour (1990), 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 QPI 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. Note that QPI can also represent the musical contour.

Table 1. QPI classification.

Semitone	QPI symbol
≤ - 8	-γ
-7 – -6	-B
-5 – -4	-β
-3	-A
-2 – -1	-α
0	o
1 – 2	α
3	A
4 – 5	β
6 – 7	B
≥8	γ

For illustration puposes, we map the 11 symbols of the alphabet $\Sigma_{QPI} = \{0, 1, ..., 10\}$ to *symbols* $-\gamma$, $-B$, $-\beta$, $-A$, $-\alpha$, o, α, A, β, B, and γ, respectively (see Table 1). Note that the symbols $-B$, $-A$, A and B represent the ambiguous intervals -7, -6; -3; 3; and 6,7 (the overlapping parts of the interval classes). The symbols of the alphabet Σ_{QPI} match themselves and the 2 adjacent symbols (excluding symbol o). To be more precise, let us denote by a and b two symbols in the interval alphabet, and by Q the function quantizing a symbol a into a symbol $Q(a)$ in the reduced alphabet Σ_{QPI} according to Table 1. If two symbols $Q(a)$ and $Q(b)$ match we write $Q(a) \cong Q(b)$. The relation $\cong$ on alphabet Σ_{QPI} is defined by Table 2.

Table 2. The relation ≅ on QPI symbols. A QPI symbol at row *i* is in ≅ relation with a QPI symbol at column *j* if and only if the cell *i,j* contains a "y".

≅	-γ	-B	-β	-A	-α	o	α	A	β	B	γ
-γ	y	y	n	n	n	n	n	n	n	n	n
-B	y	y	y	n	n	n	n	n	n	n	n
-β	n	y	y	y	n	n	n	n	n	n	n
-A	n	n	y	y	y	n	n	n	n	n	n
-α	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
A	n	n	n	n	n	n	y	y	y	n	n
β	n	n	n	n	n	n	n	y	y	y	n
B	n	n	n	n	n	n	n	n	y	y	y
γ	n	n	n	n	n	n	n	n	n	y	y

The following shift-or version applies QPI intervals.

Algorithm 3 (shift-or applying *QPI*)
begin
1. **for each** $z \in \Sigma_{QPI}$ **do** $T[z] := 2^m - 1$;
2. **for** $i := 1$ **to** $m-1$ **do**
3. **for each** $z \in \Sigma_{QPI}$ such that $z \cong Q(p_{i+1} - p_i)$ **do**
4. $T[z] := T[z] - 2^{i-1}$;
5. $E := 2^m - 1$;
6. **for** $j := 1$ **to** $n-1$ **do**
7. $E := shiftleft(E) \mid T[Q(s_{i+1} - s_i)]$;
8. **if** $E.m = 0$ **then** $write(j)$;

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 interval $p_{i+1} - p_i$. 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.

4.4 Polyphonic Sources and MonoPoly Algorithm

The bit-parallel approach also supports finding the query pattern in polyphonic sources. Holub et al. (1999) presented how the approach can be used in order to find multiple monophonic patterns within a polyphonic source using a single query. Lemström and Tarhio (2000) presented another bit-parallel algorithm to find exact, transposed occurrences of monophonic query patterns within polyphonic sources. The origin of their MonoPoly filtering algorithm lies in the shift-or method. MonoPoly comprises three phases: preprocessing, marking and checking.

The preprocessing phase calculates all the intervals appearing between the consecutive chords S_{i-1} and S_i. The intervals are first quantized by the *modulo* operator to a reduced alphabet, denoted Σ_{12}, of 12 symbols thus resulting in *an octave equivalence*. By using bitvector manipulation they avoid the apparent $O(nq^2)$ time complexity, where *q* denotes the maximum size of the chords in *S*. Instead, their preprocessing takes $O(nq)$ time.

The core of their algorithm is the marking phase which contains two subphases: pattern preprocessing and scanning. The goal of the pattern preprocessing is to build an array *T* of size $2^{12} \times (m-1)$ bits in linear $O(n)$ time. The array *T* has a bit-mask for every possible interval combination. In order to achieve the linear time complexity, a few auxiliary data structures have to be used. An array *I* has an entry for all possible intervals (*12* entries), while an array *L* stores the positions of the intervals in the pattern. The pattern preprocessing phase is followed by a scanning phase almost identical to the original shift-or algorithm.

Algorithm 4 (MonoPoly: marking)
begin
for $k := 1$ **to** 12 **do** $\{ I[k] := 2^{12} - 1;\ I[k] := I[k] - 2^{k-1} \}$;
for $k := 1$ **to** 12 **do** $L[k] := 2^{m-1} - 1$;
for $i := 2$ **to** m **do**
$b := (p_i - p_{i-1})\ mod\ 12$;
$L[b] := L[b] - 2^{i-2}$;
for $l := 1$ **to** $2^{12} - 1$ **do**
$T'[l] := 2^m - 1$;
for $k := 1$ **to** 12 **do**
$ivect := I[k]$;
if $(ivect \mid l) = ivect$ **then** $T'[l] := (T'[l]\ \&\ L[k])$;
$E := 2^{m-1} - 1$;
for $j := 2$ **to** n **do**
$E := shiftleft(E) \mid T'[S[j]]$;
if $E.m = 0$ **then** $AlgorithmC(j)$;

end

The marking algorithm above runs in time $O(n\lceil \frac{m}{W} \rceil + m + d)$ and in space $O(n+c)$, where $d = 12 \times 2^{12}$ and $c = 2^{12} + 24$.

Finally, a checking algorithm based on a naive string matching algorithm (Crochemore and Rytter, 1994) is applied to the found candidates to discard the spurious occurrences. This algorithm, called Algorithm C, runs in time $O(nq(q+m))$.

5. SEMEX PROTOTYPE

5.1 Supported Formats

SEMEX supports three file formats: the Standard MIDI File format (MIDI Manufacturers Association, 1996), which is a common interchange format for symbolically encoded music; the IRP (Inner RePresentation) format of Lemström and Laine (1998), which is a straightforward ASCII file format for encoding symbolic music; and MDB (Melody DataBase), which is a simple flat-file database format used in the prototype. MDB files contain a single string of pitch values encoding all the music in the database and a track list for mapping indices in the pitch string to tracks in source files. The mapping is needed as otherwise the Query Engine component would have no way of telling in which source file a match originated; neither would it be able to discern matches straddling a track boundary.

Recall that, although there is more information available in the formats, we are currently considering only the pitch levels of notes.

5.2 System Architecture

The components of the SEMEX prototype system are shown in Figure 2. 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 of Tolonen and Karjalainen (2000) facilitates conversion of digital audio data 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 MIDI files into a database.

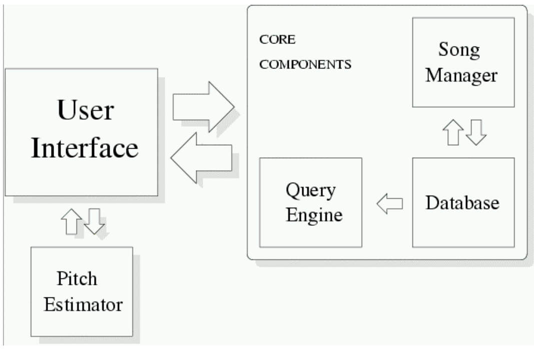

Figure 2. Components of the SEMEX Prototype.

The core of the prototype is written in C++ with a careful object-oriented design, allowing different software components to interact nearly orthogonally. For instance, all matching algorithms can be applied on any type of source data. This property is due to the algorithms being implemented as template functions.

5.3 User Interface

Currently, the prototype has a simple Unix-like user interface: it is invoked from the command line and different options are selected with "switches". A sample invocation of the prototype is shown in Figure 3. In the example, the 10 best matches of a query pattern are shown from "midi.mdb", which is a database file containing monophonic reductions from a hand-picked collection of classical MIDI files. In addition to textual output, the prototype can extract the parts corresponding to matches from the actual MIDI source files referred to in the database and play them back using an external program, with appropriate volume fading.

```
perttu@saapas:~$ Semex -n10 -mq f CDEFGGGGGFEDCCCC 3
midi.mdb
The query pattern is C-5 +2+2+1+2+0+0+0+0-2-1-2-2+0+0+0.
40 matches found.
 # distance   source     sequence
 1 0.997    647-662    A-5 +1+2+2+1+0+0+0+0+0-1-2-2+0+0+0
            Gershwin_Rhapsody_in_Blue.mid track 2
 2 1.990    254-269    C-4 +2+1+2+1-1+0+0+0-2-1-2-1-1+0+0
            Shostakovich_Symphony_No10_1.mid track 5
 3 1.999    587-602    A-4 +1+2+2+13+0+0+0+0+0-1-2-2+0+0+0
            Gershwin_Rhapsody_in_Blue.mid track 6
 4 1.999    534-549    A-4 +1+2+2+13+0+0+0+0+0-1-2-2+0+0+0
            Gershwin_Rhapsody_in_Blue.mid track 8
 5 2.002    1057-1073  C#7 +2+1+2+1+0+0+0+0+2-2-1-2-1+6+0+0
            Beethoven_Symphony_No9_4.mid track 0
 6 2.008    3358-3373  G-6 +2+2+2+1-10+0+0+12-2-1-2-2+0+0+0
            Mozart_Piano_Sonata_No6.mid track 0
 7 2.008    3393-3408  G-6 +2+2+2+1-10+0+0+12-2-1-2-2+0+0+0
            Mozart_Piano_Sonata_No6.mid track 0
 8 2.015    533-548    E-6 +2+2+1+2-19+0+0+12-2-1-2-2+0+0+0
            Mozart_Piano_Sonata_No6.mid track 1
 9 2.015    568-583    E-6 +2+2+1+2-19+0+0+12-2-1-2-2+0+0+0
            Mozart_Piano_Sonata_No6.mid track 1
10 2.016    2179-2194  C-7 +1+2+1-12+0+0+0+0-1-2-1+12+0+0+0
            Beethoven_Piano_Sonata_No14_Moonlight.mid track 0
perttu@saapas:~$
```

Figure 3. A sample invocation of the SEMEX prototype.

5.4 Retrieving from Monophonic Sources

As we mentioned earlier, bit-parallel string matching algorithms have proven themselves to be very flexible; it is rather straightforward to apply the transposition invariant distance measures of Lemström and Ukkonen (2000) and different classifications of pitches, such as contour and QPI classifications of Lemström and Laine (1998), while still preserving the linear time complexity. To make it possible to deal with approximate matching, instead of using shift-or, we apply Myers' (1998) algorithm with analogous modifications to those given in previous Section.

In Figure 4, we illustrate the performance of the adapted Myers' algorithm (including the pruning but excluding the verification phase described in the following paragraphs) by comparing it to the standard dynamic programming algorithm and Ukkonen's (1985b) cutoff algorithm. The comparison was run in a 700 MHz Pentium III with 768 MB of RAM under the Linux operating system. The length of a machine word was 32 bits. The database consisted of 208 classical MIDI files, which were reduced in this comparison to a monophonic form by including only the highest notes from the polyphonic texture. The reported scanning time is the average of the scanning times of 100 patterns of length 12 selected randomly from the database. The approximation parameter k (which mainly affects Ukkonen's cutoff algorithm) was set to 3 during the runs.

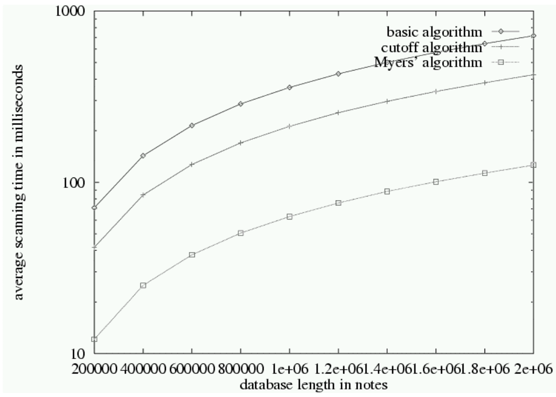

Figure 4. A comparison of the running times of the basic dynamic programming algorithm, Ukkonen's cutoff algorithm, and Myers' bit-parallel algorithm.

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 that, as for columns, the difference in edit distance between two successive entries on a row of a (d_{ij}) table is at most one. This results in an undesirable proliferation of matches on both sides of a center match with an edit distance below k.

In order to combat such "pollution", the matching algorithms are 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 the scanning phase the exact substring of s corresponding to a match is not known - Myers' algorithm computes only edit distances. Therefore, to verify a matching position j, it is fed to the basic dynamic programming algorithm, which resolves the corresponding match. The verification algorithm also computes the final distance $D^v{}_{mj}$ of the match, employing a variant of the edit distance metric:

$$D^v_{ij} = \min \begin{cases} D^v_{i-1,j} + 1 \\ D^v_{i,j-1} + 1 \\ D^v_{i-1,j-1} + \left(\begin{array}{c} if\ p_i = s_j\ then\ 0\ else\ 0.992 + \\ 0.001 \mid p_i - s_j \mid \end{array} \right) \end{cases}$$

The sole purpose of the verification metric is to fine-tune the sorting of matches with equal edit distances. The magical constant 0.992 was arbitrarily selected; its effect is that replacement operations where the pattern and source intervals differ by a pure fifth or less are preferred over insertions and deletions. Nevertheless, the verification metric is easily replaced with another, more suitable one, for the case at hand.

When the exact substring 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.

5.5 Retrieving from Polyphonic Sources

In SEMEX, we can deal with databases containing polyphony in two ways. First, the database can be reduced to a monophonic form by considering only the highest notes of chords, and then applying the techniques described in the previous subsection (thus, the performance of this approach in SEMEX relies on Myers' algorithm). Second, we can search transposed exact occurrences by using the MonoPoly algorithm.

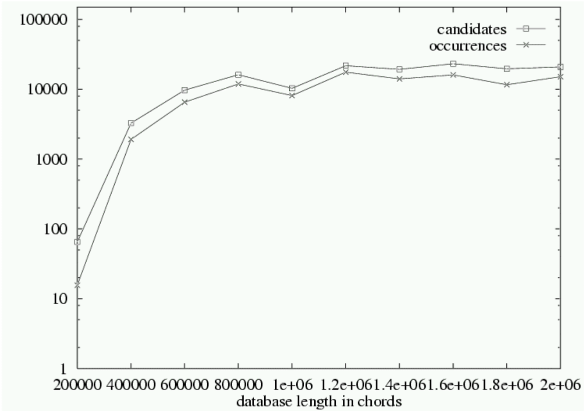

Figure 5. Running the MonoPoly algorithm on test database: number of candidates and proper occurrences.

Recall, the three phases in MonoPoly. The preprocessing phase (with $O(nq)$ time complexity), the marking (if $m \leq W$, this is executed in linear time), and the checking phase (executable in $O(nq(q+m))$ time). Therefore, the total running time depends rather heavily on the filtration ability of the marking phase.

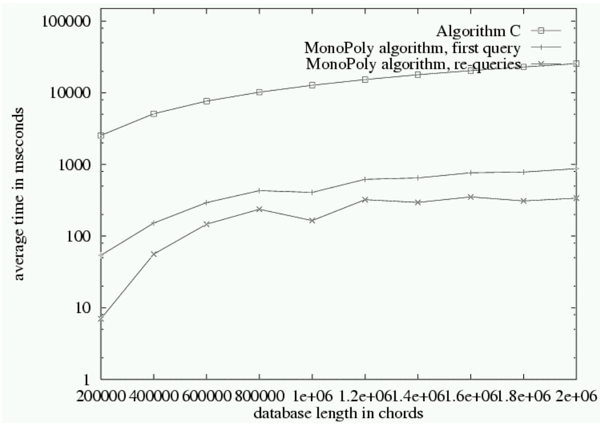

Figure 6. Running the MonoPoly algorithm on test database: times for a first query (preprocessing + marking + checking) and re-queries (marking + checking) of MonoPoly, as compared to a stand alone version of Algorithm C.

In Figures 5 and 6, one can see how the MonoPoly algorithm performs with the test database. We illustrate both the efficiency of the filtration (number of proper occurrences / candidate occurrences), and compare the running times of the MonoPoly algorithm (for a first query to a database, and for "re-queries" to the same database) to the straightforward algorithm which is also used in the checking phase of the MonoPoly algorithm. The settings for this comparison were as in the previous comparison, but in this case the polyphony was preserved. The average number of voices (within MIDI tracks) varied usually between 1 and 3, but the maximal degree was as high as 41. As can be seen in the figures, though the number of candidates grows faster than the number of proper occurrences, the MonoPoly algorithm clearly outperforms the comparison algorithm. When using the

MonoPoly algorithm, even the first queries to the database were completed within one second in all the cases.

6. CONCLUSION

In this paper, we have considered the problem of music information retrieval for which the edit distance framework has constantly been applied to solve the problem. We have reviewed how this framework can be modified to meet better the requirements of MIR, and to implement ideas that we have introduced earlier.

We have shown that those ideas are implementable by using the so-called bit-parallel algorithm technique. Although the illustration has been based on the shift-or algorithm developed for exact matching, in most of the cases they are analogously applicable with approximate bit-parallel matching algorithms, as well.

Bit-parallel algorithms are very fast in practise. Besides the speed, they have also another advantage over the traditional algorithms, a certain kind of flexibility. For instance, any character in the pattern can match an arbitrary set of source characters at no additional cost, making, e.g., different interval reduction schemes easy to implement.

In this paper, our main contribution is the efficient MIR prototype system, called SEMEX, making good use of our previous ideas and the bit-parallel implementations. Actually, SEMEX is the first of its kind to make extensive use of bit-parallelism in matching and retrieval. It is also the first MIR system providing the possibility to retrieve patterns from real polyphonic databases. Whether retrieving from polyphonic databases can be implemented by using the approximate bit-parallel algorithm, is left for the future work.

7. REFERENCES

Baeza-Yates, R., & Gonnet, G. H. (1992). A new approach to text searching. Communications of the ACM, 35 (10), 74-82.

Baeza-Yates, R., & Perleberg, C. H. (1992). Fast and practical approximate string matching. In Proceedings of Combinatorial Pattern Matching, (pp. 185-192).

Bainbridge, D., Nevill-Manning, C., Witten, I., Smith, L., & McNab, R. (1999). Towards a digital library of popular music. In Proceedings of the fourth ACM conference on digital libraries, (pp. 161-169). Berkeley, CA.

Cambouropoulos, E., Crawford, T., & Iliopoulos, C.S. (1999). Pattern processing in melodic sequences: Challenges, caveats & prospects. In Proceedings of the AISB'99 Symposium on Musical Creativity, (pp. 42-47). Edinburgh.

Chang, W., & Lampe, J. (1992). Theoretical and empirical comparisons of approximate string matching algorithms. In Proceedings of Combinatorial Pattern Matching, (pp. 1-14).

Crawford, T., Iliopoulos, C.S., & Raman, R. (1998). String matching techniques for musical similarity and melodic recognition. Computing in Musicology, 11, 71-100.

Crochemore, M., & Rytter, W. (1994). Text Algorithms. Oxford University Press.

Cronin, C. (1998). Concepts of melodic similarity in music-copyright infringement suits. Computing in Musicology, 11, 185-209.

Dovey, M.J. (1999). An algorithm for locating polyphonic phrases within a polyphonic musical piece. In Proceedings of the AISB'99 Symposium on Musical Creativity, (pp. 48-53). Edinburgh.

Downie, J.S. (1999). Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic n-grams as Text. PhD thesis, The University of Western Ontario, Faculty of Information and Media Studies.

Ghias, A., Logan, J., Chamberlin, D., & Smith, B.C. (1995). Query by humming - musical information retrieval in an audio database. In ACM Multimedia 95 Proceedings, (pp. 231-236). San Francisco, CA.

Gusfield, D. (1997). Algorithms on Strings, Trees, and Sequences. Cambridge University Press.

Hawley, M. (1993). Structure out of Sound. PhD thesis, MIT.

Holub, J., Iliopoulos, C.S., Melichar, B., & Mouchard, L. (1999). Distributed string matching using finite automata. In Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms, (pp. 114-128). Perth.

Kornstädt, A. (1998). Themefinder: A web-based melodic search tool. Computing in Musicology, 11, 231-236.

Lemström, K. (2000). String Matching Techniques for Music Retrieval. PhD thesis, University of Helsinki, Department of Computer Science. Report A-2000-4.

Lemström, K., & Laine, P. (1998). Musical information retrieval using musical parameters. In Proceedings of the 1998 International Computer Music Conference, (pp. 341-348). Ann Arbor, MI.

Lemström, K., & Tarhio, J. (2000). Detecting monophonic patterns within polyphonic sources. In Content-Based Multimedia Information Access Conference Proceedings (RIAO'2000), (pp. 1261-1279). Paris.

Lemström, K., & Ukkonen, E. (2000). 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, (pp. 53-60). Birmingham.

McNab, R., Smith, L., Bainbridge, D., & Witten, I. (1997). The New Zealand digital library MELody inDEX. D-Lib Magazine.

MIDI Manufacturers Association. (1996). The Complete Detailed MIDI 1.0 Specification. Los Angeles, California.

Mongeau, M., & Sankoff, D. (1990). Comparison of musical sequences. Computers and the Humanities, 24, 161-175.

Myers, G. (1998). A fast bit-vector algorithm for approximate string matching based on dynamic programming. In Proceedings of Combinatorial Pattern Matching, (pp. 1-13). Piscataway, NJ.

Narmour, E. (1990). The Analysis and Cognition of Basic Melodic Structures – The Implication-Realization Model. The University of Chicago Press.

Navarro, G., & Raffinot, M. (1998). A bit-parallel approach to suffix automata: Fast extended string matching. In Proceedings of Combinatorial Pattern Matching, (pp. 14-33). Piscataway, N.J.

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

Rolland, P.-Y. (1999). Discovering patterns in musical sequences. Journal of New Music Research, 28(4), 334-350.

Rolland, P.-Y., & Ganascia, J.-G. (1996). Automated motive oriented analysis of musical corpuses: a jazz case study. In Proceedings of the 1996 International Computer Music Conference, (pp. 240-243). Hong Kong.

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

Sellers, P.H. (1980). The theory and computation of evolutionary distances: Pattern recognition. Journal of Algorithms, 1(4), 359-373.

Shmulevich, I., & Povel, D. (1998). Rhythm complexity measures for music pattern recognition. In Proceedings of IEEE Workshop on Multimedia Signal Processing, (pp. 7-9). Redondo Beach, CA.

Shmulevich, I., Yli-Harja, O., Coyle, E., Povel, D.-J., & Lemström, K. (2001). Perceptual issues in music pattern recognition - complexity of rhythm and key finding. Computers and the Humanities, 35(1), 23-35.

Tolonen, T., & Karjalainen, M. (2000). A computationally efficient multi-pitch analysis model. IEEE Transactions on Speech and Audio Processing, 8 (6), 708-716.

Tseng, Y.-H. (1999). Content-based retrieval for music collections. In Proceedings of the 22nd International Conference on Research and Development in Information Retrieval, (pp. 176-182). Berkeley, CA.

Uitdenbogerd, A.L., & Zobel, J. (1998). Manipulation of music for melody matching. In ACM Multimedia 98 Proceedings, (pp. 235-240). Bristol.

Uitdenbogerd, A.L., & Zobel, J. (1999). Melodic matching techniques for large music databases. In ACM Multimedia 99 Proceedings. Orlando, FLO.

Ukkonen, E. (1985a). Algorithms for approximate string matching. Information and Control, 64, 100-118.

Ukkonen, E. (1985b). Finding approximate patterns in strings. Journal of Algorithms, 6, 132-137.

Wu, S., & Manber, U. (1992). Fast text searching allowing errors. Communications of the ACM, 35 (10), 83-91.

APPENDIX: MIR SYSTEM COMPARISON

	Ghias et al. (95)	MELDEX Bainbridge et al. (99)	THEMEFINDER Kornstädt (98)	Uitdenbogerd & Zobel (99)	Pollastri (99)	MELODISCOV Rolland et al. (99)	SEMEX
Pitch Representation	contour	interval &contour	several	intervals	abs. pitches & intervals	several (incl. interval and contour)	abs. Pitches, contour & QPI
Rhythm Representation	-	durations	-	-	durations & duration ratios	durations & duration ratios	-
Allows Polyphony	-	-	-	- (+)º	-	-	+
Transposition Invariant	+	+	+	+	+	+	+
Pitch Tracking	+	+	-	-	+	+	+
Optical Music Recognition	-	+	-	-	-	-	-
Other Factors						takes into account metric positions	various distances
Matching Algorithm	Baeza-Yates & Perleberg (92)	dp¹		dp	Chang & Lampe (92)	FlExPat², Rolland (99)	bp, Myers (98)
Type of Matching	k mismatches	augmented k differences	exact	k differences	k differences	augmented k differences	exact & augmented k differences
Time Complexity	O(mn)	O(mn)		O(mn)	O(kn)	O(mn)	O(n)³

Legend:
dp dynamic programming (conventional)
bp bit-parallelism
º Reducing polyphonic surfaces into monophonic
¹ A straightforward implementation of Wu and Manber's (92) O(kn) bit parallel algorithm is available; dp is preferred
² Although Melodiscov is not currently based on the FlExPat algorithm, it is claimed to be shortly
³ Cores of the algorithms (assuming that $m \leq W$)