SEMEX – An Efficient Music Retrieval Prototype *

Kjell Lemström and Sami Perttu
Department of Computer Science
P.O. Box 26 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 within one second in both cases. Furthermore, our prototype is capable of using various interval classes in matching, producing more approximation when it is needed.
Key words: music retrieval, bit parallelism, polyphonic databases

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 MIR systems have begun to appear. The systems interpret melodies as strings that can be matched against query patterns using standard methods from general string matching.

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 earlier (for a summary of these ideas, 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 (i.e. 32 bits). Thus, by using the bit-parallel techniques presented later, we can expect that the cores of the 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

* A work partly supported by Nokia Research Center.

also differs from previous systems in other respects, which are discussed below (see Appendix for a comparison table of current MIR systems).

Some MIR systems, such as MELDEX (Bainbridge, Nevill-Manning, Witten, Smith & McNab 1999), Melodiscov (Rolland, Raskinis & Ganascia 1999), and the prototypes of Ghias, Logan, Chamberlin & Smith (1995) and Pollastri (1999) accept digital input which is subsequently converted into an internal representation for matching. 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 reach good *discriminatory power* (Downie 1999). In SEMEX, we can achieve better discrimination while still maintaining tolerance to errors by using the *QPI classification* introduced by Lemström & Laine (1998).

The representation of choice for melodies in current MIR systems is a symbolic string of pitches. Durational information can be used in some systems but the emphasis is usually put on pitch information. When dealing with queries on pitches, *transposition invariance* is a very important property that should be considered. An algorithm taking into account transposition invariance usually ignores the *musical key* of the pitch sequence by using an *interval representation* that considers only the pitch distances in semitones between consecutive elements within the musical sequences. However, as noted by Cambouropoulos, Crawford & Iliopoulos (1999), some problems are involved with that encoding as well. Lemström & Ukkonen (2000) presented how transposition invariance can be achieved without using interval representation, and thus, some of those shortcomings are avoidable. We apply their transposition invariant distance metrics in SEMEX.

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 literature (see e.g., Uitdenbogerd & Zobel 1998, Dovey 1999, Holub, Iliopoulos, Melichar & Mouchard 1999, Lemström & Tarhio 2000), to our knowledge, besides SEMEX, only the system by Uitdenbogerd & Zobel (1999) considers this problem. However, the approaches of these two systems are different. Uitdenbogerd and Zobel reduce the polyphonic surface to a monophonic melody 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 by Lemström & Tarhio (2000).

2 Problem Setting

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, three values of r are of particular interests; $r = 2$ (representing musical contour); $r = 10$ (representing the QPI classification); and $r = 127$ (as in MIDI (MID 1996)).

A musical *source* $S = S_1 S_2 \cdots S_n$ 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 alphabet Σ.

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

s, is represented by a similar structure to that of the pattern representation.

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

We define that there is an exact occurrence of p at position j, if $p_i \in S_{j+i-1}$ holds for $i = 1, \ldots, m$, and there is a *transposition invariant occurrence* of p at position j, if there is an integer d such that each $(p_i + d) \in S_{j+i-1}$. Currently, we do not allow approximate queries to a polyphonic source.

When the source s is monophonic, exact and transposition invariant occurrences of p within s mean that $p_i = s_{j+i-1}$ and $(p_i + d) = s_{j+i-1}$ for each i, respectively. 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). The conventional editing operations are:

- replacement: the aligned symbols p_i and s_{j+i-1} are distinct,
- insertion: the symbol s_j is missing in p,
- deletion: the symbol p_i is missing in s.

3 Approximate Pattern Matching

Approximate string pattern matching is based on the concept of *edit distance* (Crochemore & Rytter 1994, Gusfield 1997). The edit distance $D(A, B)$ between strings $A = a_1, \ldots, a_m$ and $B = b_1, \ldots, b_n$, $A, B \in \Sigma^*$ (Σ^* denotes the set of all sequences over Σ), is the minimum number of editing operations (given above) required to transform string A into string B. The special case in which deletions and insertions are not allowed is called the *Hamming distance*.

A standard way to locate approximate occurrences of p in s, is to search for subsequences p' of s, such that $D(p, p') \leq k$ (k is a threshold value used to control the accuracy of the pattern matching). This can be done efficiently by using *dynamic programming*. Such a procedure tabulates all distances $d_{ij} = D(p_1 \cdots p_i, s_{j'} \cdots s_j)$ (where $j' \leq j$) between the *prefix* of p and any *factor* (substring) of s (Sellers 1980, Ukkonen 1985*a*). The pattern matching problems associated with edit and Hamming distances are called k *differences* and k *mismatches*, respectively.

Bit-Parallelism. The development of dynamic programming methods computing the edit distance has lead to an algorithm family applying a fast bit-parallel approach. When replacing a conventional implementation where each variable resides in its own memory location or register with a corresponding implementation where the variables can be manipulated in parallel with bit operations, a significant speed-up can be achieved. One of the first algorithms of the family was presented by Baeza-Yates & Gonnet (1992). Their shift-or algorithm was originally designed for the exact matching. Denoting by W the number of bits in a computer word used, their algorithm runs in time $O(\lceil \frac{m}{W} \rceil n)$, i.e., in linear time if the pattern is short enough as compared to the length of the (used) computer word (if $m \leq W$). The MonoPoly algorithm, which we will use with polyphonic databases, is based on the shift-or algorithm.

Recently Myers (1998) presented a fast bit-parallel implementation for the k differences problem. It also runs in time $O(\lceil \frac{m}{W} \rceil n)$. The performance of the algorithm rests on a certain property of the dynamic programming table; the difference between adjacent vertical cells is always -1, 0, or 1 (Ukkonen 1985*b*). In Myers' algorithm, the current column in the matrix is represented by two bit mask variables P and M that store positive and negative changes (deltas) in the matrix values, respectively. We apply Myers' algorithm when dealing with monophonic databases.

4 Semex Prototype

4.1 Supported Formats

SEMEX supports three file formats: the Standard MIDI File (SMF) format (MID 1996), which is a common interchange format for symbolically encoded music; the IRP (Inner RePresentation) format (Lemström & 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, though there is more information available in the formats, we are currently considering only the pitch levels of notes.

4.2 System Architecture

The components of the SEMEX prototype system are shown in Fig. 1. The User Interface component makes use of the other components in order to present the user with a single interface. A separate Pitch Estimator component (Tolonen & 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.

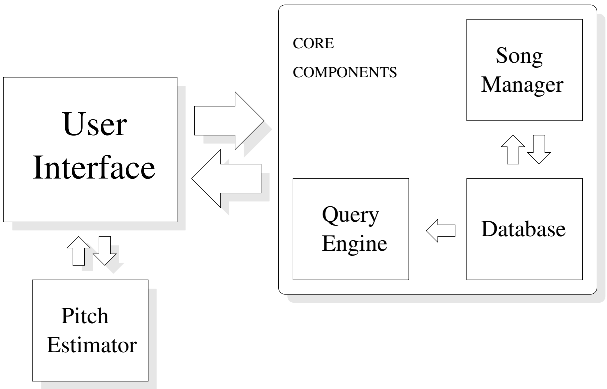

Figure 1: *Components of the SEMEX Prototype.*

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 use any interval reduction scheme and can be applied on any type of source data. The latter property is due to the algorithms being implemented as template functions.

4.3 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 Fig. 2. 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 2: *A sample invocation of the SEMEX prototype.*

4.4 Retrieving from Monophonic Sources

In the special case when the database is monophonic, we apply Myers’ algorithm (Myers 1998). Bit-parallel string matching algorithms have proven themselves to be very flexible; it is rather straightforward to apply the transposition invariant distance measures by Lemström & Ukkonen (2000) and different classifications of pitches (such as contour and QPI classifications (Lemström & Laine 1998)), while still preserving the linear time complexity.

In Fig. 3, we illustrate the performance of the adapted Myers’ algorithm (including the pruning but excluding the verification phase of the following paragraphs) by comparing it to the standard dynamic programming algorithm and Ukkonen’s cutoff algorithm (Ukkonen 1985*b*). 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.

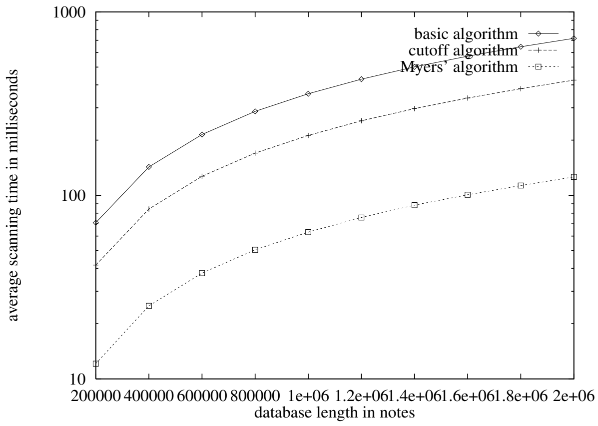

Figure 3: *A comparison of the running times of the basic dynamic programming algorithm, Ukkonen's cutoff algorithm, and Myers' bit-parallel algorithm.*

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 the dynamic programming matrix 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:

$$D^v_{i,j} = \min \begin{cases} D^v_{i-1,j-1} + (\text{if}\, p_i = s_j \,\text{then}\, 0 \,\text{else}\, 0.992 + 0.001), \\ D^v_{i-1,j} + 1, \\ D^v_{i,j-1} + 1. \end{cases} \quad (1)$$

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.

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.

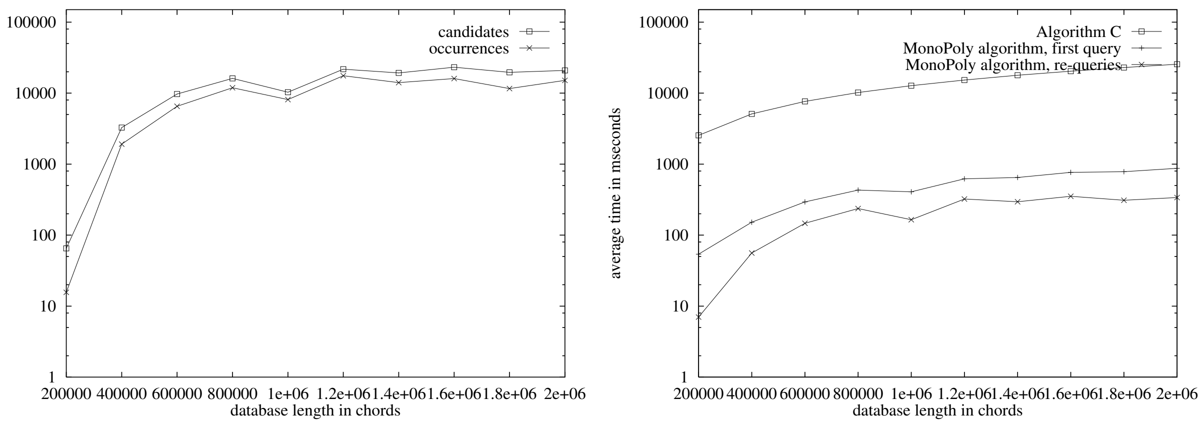

Figure 4: *Running the MonoPoly algorithm on the test database. On the left, the numbers of candidates and proper occurrences (note the logarithmic scale). On the right, the times for a first query (preprocessing+marking+checking) and re-queries (marking+checking) of MonoPoly, as compared to a 'stand-alone' version of the checking phase (referred to as Algorithm C).*

4.5 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 (as suggested by Uitdenbogerd & Zobel (1999)), 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.

The MonoPoly algorithm runs in two phases, out of which the latter comprises two subphases. In the first phase, the algorithm preprocesses the source in time $O(nq)$ (q denotes the maximum size of the chords). The second phase applies a *filtering* technique. First a *marking phase* applying bit parallelism searches possible occurrences (called *candidates*) of the given query pattern. If $m \leq W$, this is done in linear time. Then a *checking phase* scans through the candidates to find the *proper occurrences* among them. For this phase, we use a somewhat slower algorithm with a time complexity $O(nq(q + m))$ (Lemström & Tarhio 2000). Therefore, the speed of the MonoPoly algorithm depends rather heavily on the filtration ability of the marking phase.

In Figs. 4 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.

5 Conclusion

We have presented a prototype of an efficient music information retrieval system. It is the first of its kind to make extensive use of fast bit-parallel algorithms in matching. These algorithms have other advantages besides speed over the traditional algorithms: 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. Our prototype is also the first MIR software system to provide for matching monophonic patterns against sources containing true polyphony.

Acknowledgments

The authors are grateful to Professor Esko Ukkonen for his support on this project.

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* 'Combinatorial Pattern Matching, Third Annual Symposium', 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', Berkeley, CA, pp. 161–169.

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', Edinburgh, pp. 42–47.

Chang, W. & Lampe, J. (1992), Theoretical and empirical comparisons of approximate string matching algorithms, *in* 'Combinatorial Pattern Matching, Third Annual Symposium', pp. 1–14.

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

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', Edinburgh, pp. 48–53.

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', San Francisco, CA, pp. 231–236.

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

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', Perth, pp. 114–128.

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. (to appear).

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

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

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', Birmingham, pp. 53–60.

MID (1996), *The Complete Detailed MIDI 1.0 Specification*.

Myers, G. (1998), A fast bit-vector algorithm for approximate string matching based on dynamic programming, *in* 'Proceedings of Combinatorial Pattern Matching', Piscataway, N.J., pp. 1–13.

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., 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.

Tolonen, T. & Karjalainen, M. (2000), 'A computationally efficient multi-pitch analysis model', *IEEE Transactions on Speech and Audio Processing*. (in press).

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

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

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

Ukkonen, E. (1985*b*), '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

	Pitch representation	Rhythm representation	AP	TI	PT	OR	Other factors	Matching algorithm	Type of matching	Time complexity
Ghias et al.(95)	contour	-	-	+	+	-		Baeza-Yates & Perleberg(92)	k mism	$O(mn)$
MELDEX Bainbridge et al.(99)	intervals & contour	durations	-	+	+	+		dp†	augmented k diff	$O(mn)$
THEMEFINDER Kornstädt(98)	several	-	-	+	-	-		?	exact	?
Uitdenbogerd & Zobel(99)	intervals	-	+	+	-	-		dp	k diff	$O(mn)$
Pollastri(99)	abs. pitches & intervals	durations & duration ratios	-	+	+	-		Chang&Lampe(92)	k diff	$O(kn)$
MELODISCOV (+FlExPat)⋆ Rolland et al.(99)	several (incl. intervals & contour)	durations & duration ratios	-	+	+	-	takes into account metric positions	FlExPat, Rolland(99)	augmented k diff	$O(mn)$
SEMEX	abs. pitches, contour & QPI	-	+	+	+	-	various distances	bp, Myers(98)	exact & augmented k diff	$O(n)^{\ddagger}$

AP Allows polyphony
TI Transposition invariant
PT Pitch tracking
OR Optical music recognition
dp Dynamic programming (conventional)
bp Bit parallelism
† A straightforward implementation of Wu & Manber's (92) $O(kn)$ bit parallel algorithm is available; dp is preferred
⋆ Although Melodiscov is not currently based on the FlExPat algorithm, it is claimed to be shortly.
‡ Cores of the algorithms (assuming that $m \leq W$)