A technique for "regular expression" style searching in polyphonic music

Matthew J. Dovey
Visiting Research Fellow
Dept. of Computer Science
Kings College, London
+44 1865 278272
matthew.dovey@las.ox.ac.uk

ABSTRACT

This paper discussed some of the ongoing investigative work on integrating these two systems conducted as part of the NSF/JISC funded OMRAS (Online Music Retrieval and Searching) project into polyphonic searching of music. It describes a simple and efficient "piano-roll" based algorithm for locating a polyphonic query within a large polyphonic text. It then describes ways in which this algorithm can be modified without affecting the performance to allow more freedom in the how a match is made, allowing queries which involve something akin to polyphonic regular expressions to be located in the text.

1. INTRODUCTION

The OMRAS (Online Music Retrieval And Searching) project is a three year collaborative project between Kings College London and the Center for Intelligent Information Retrieval, University of Massachusetts. Its primary aim is to look at various issues surrounding content based searching of polyphonic music; current research in content based music searching has primarily concentrated on monophonic music, that is to say music consisting of only a single melodic line and ignoring the complexities find in a more complex music texture for example as found in an say an orchestral symphony.

Different computer representations of music roughly fall into two basic categories: those representing the audio of the music such as a typical wave or aiff file (or more topically MP3), and those representing the symbolic structure of the music as indicated in a typically written musical score. The audio file formats typically represent an actual performance of a piece, whilst the symbolic formats represent the composer's instructions and guidelines to the performer. In practice, as described in Byrd and Crawford (2001) [1], these are two extremes of a spectrum with various formats falling in between which contain elements of both such as MPEG7 and MIDI. MIDI, for example, was originally designed for representing music performances but is closer to a symbolic notation than an audio representation (and is used to indicate instructions for a performance rather than record a performance). (Byrd and Crawford actually talk of three distinct types – the two referred to above plus MIDI representing the middle ground)

One aspect of OMRAS is to look at conversion of formats moving along this spectrum. Moving from symbolic notations to audio is fairly straightforward but this is to be expected since the role of the symbolic notation is to describe how to enact a performance of the music. Going from the audio to a symbolic description of how to perform that audio is far more difficult. A good comparison would be between a performance of a play, and the script and stage-directions for performing that play.

A second aspect on OMRAS is to consider the searching of music by content. As indicated above we are concentrating on polyphonic music since this is currently neglected in the existing work. One of the groups in OMRAS has been looking at audio work, but this paper concentrates on searching symbolic formats which represent Common Music Notation (CMN) of the Western tradition of music. Most work on music searching has concentrated on searching within monophonic, single voice music, often applying techniques derived from the text-retrieval world (e.g. Downie (1999) [5]). There are been some approaches at polyphonic searching (e.g. Uitdenbogerd and Zobel (1998) [9], Holub, Iliopoulos, Melichar and Mochard (1999) [6] and Lemström and Perttu (2000) [7]). This paper will outline some of the algorithms we have developed for this purpose, which we believe are more versatile for regular expression style searching than previous work in this area.

A key mission statement of the joint JISC/NSF International Digital Library Initiative, which is funding the OMRAS work is that the projects should make existing digital collections more accessible. We feel that the work of OMRAS makes digital collections of music more accessible by providing content based searching possible, in addition to standard metadata searched such as by composer or title. OMRAS is collaborating with the JISC funded JAFER project at Oxford University to provide integration with existing music library catalogues[1].

2. THE OMRAS TEST FRAMEWORK

Within the OMRAS project we have written a test framework using the Java programming language for testing the performance of various search algorithms and techniques. The framework is command line driven (and not really designed for novice users). From the command line we can load a music file into the system), can load in different algorithms and can load in different user interface components for displaying and editing queries and results. The framework allows us to experiment with a number of different algorithms and a number of different representations of music. A screenshot of the command line framework in use is given in figure 1.

The framework has been designed to take advantage of Java's object-oriented nature: all the components such as file format modules, user interface widgets and search algorithms are

[1] http://www.lib.ox.ac.uk/jafer and http://www.jafer.org

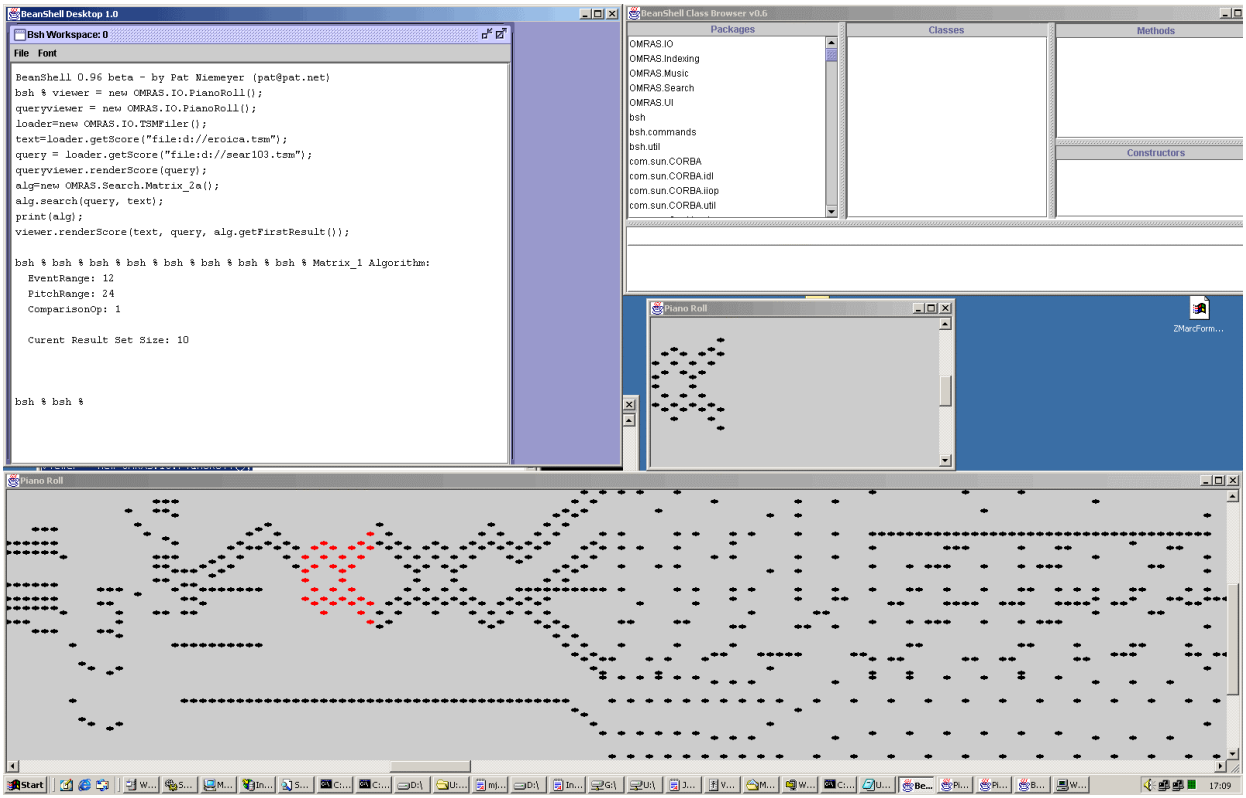


Figure 1

implemented as independent java objects, so that the entire framework is modular. These java objects can be manipulated in the framework using a scripting interface for java such as BeanShell[2], but selected components can be used in other software. In particular, some engineering undergraduates have been working with OMRAS to build a GUI interface to the framework in addition to the current command line interface. Collaborating with the JISC funded JAFER Project at Oxford University[3] we have reused the components to build a prototype demonstrating how our algorithms for content based searching can be integrated into existing bibliographic oriented music library catalogue systems (Dovey, M (2001) [4]).

At the moment we only have modules for handling a small number of file formats (including MIDI) but are working on others most notably GUIDO and Kern[4]. The user interface components are based on piano roll type displays, but we are working on incorporating better CMN software for displaying scores into the framework in the near future. We also have objects for rendering scores into audio, using the Java Media Framework.

[2] http://www.beanshell.org

[3] http://www.jafer.org and http://www.lib.ox.ac.uk/jafer

[4] A good reference of various music file formats can be found in Selfridge-Field (1997) [8].

3. BASE ALGORITHM FOR SEARCHING POLYPHONIC MUSIC

3.1 "Piano Roll" Model of CMN

Common Music Notation (CMN) is a very complex pictorial language for describing music with a number of cues as to how it should be performed. Its pictorial nature proved very difficult for in the history of the printing press; in many cases the most efficient means to produce a printed score was to create an engraving rather than attempt to produce a generic type-set for music. It is not surprising, therefore, that CMN produces complex problems for computer based representations. We are working with a very simple model for representing music, but one, which can provide a skeleton for re-introducing some of the complexities of CMN.

The representational model of music we are currently using can be compared to that of a piano roll, where punched holes indicate whether a note should be playing. The horizontal position in the roll indicates the time of the note, whilst the vertical position indicates the pitch of the note. In our base model we only concern ourselves with the beginnings of notes (in MIDI terms with the Note On events). We consider a musical "event" to be a list of notes which begin to play at the same time (in some ways this is similar to a chord but is more generalized). Whilst not an ideal terminology, we will use the term "event" in this manner, for the purposes of this paper. We only introduce a new musical event where needed, i.e. because a new note has begun to play. In this model we can regard a piece of music to be a sequence of musical events. For example the extract in Figure 2,

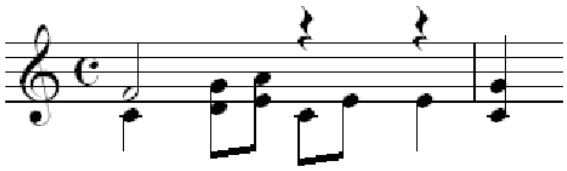


Figure 2

could be represented as the following list of musical events:

1. F, C
2. G, D
3. A, E
4. C
5. E
6. E
7. G, C

This leads an obvious representation as a matrix of zeros and ones indicating whether a note begins to play at that time and pitch and this is used as an effective way to display queries and results. For example the more complex extract in Figure 3

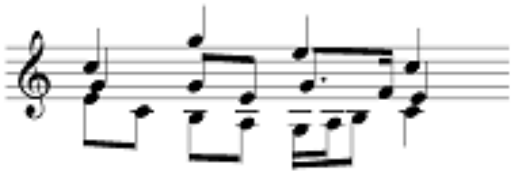


Figure 3

can be represented in this piano roll format as in Figure 4

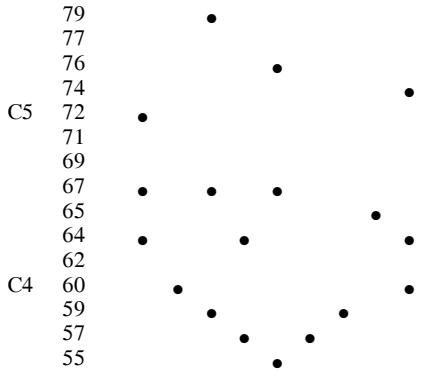


Figure 4

In the OMRAS framework, we represent as an XML structure such as:

```
<score>
  <event>
    <note pitch="72"/>
    <note pitch="67"/>
    <note pitch="64"/>
  </event>
    <note pitch="60"/>
  <event>
  </event>
```

etc…

By representing this structure as an XML Schema document we can generate the Java object model for this format from the XML Schema description. Representing CMN in this manner allows us to add additional music information. For example onset time (in milliseconds as for MIDI, or metrical information such as bar number of beat within the bar) can be added as additional attributes for the event element; duration of notes, voicing, instrumentation etc. can be added as additional attributes to the note element.

3.2 Searching in a "piano roll" model

In the model, described above the typical search problem can be expressed as follows: given a query as a sequence of musical events and a text to perform the query on as a sequence of musical events, we are trying to find a sequence of musical events in the text such that each musical event in the query is a subset of the musical event in the text. This again is best illustrated by an example. Consider the musical extract in figure 5.

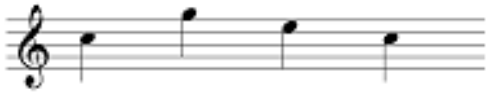


Figure 5

As a piano roll, this would be represented as in Figure 6.

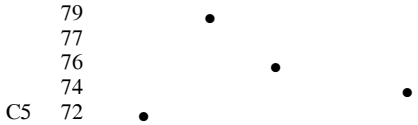
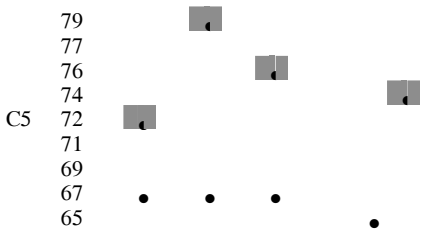


Figure 6

A potential match is illustrated in Figure 7. As can be seen, there is some allowance in that intervening musical events are allowed between matched events in the text. The freedom of this can be limited to avoid too many spurious matches.



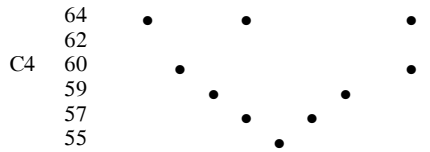


Figure 7

In musical terms, it is also necessary to allow transpositions of pitch. Figure 8 gives a second match at a different transposition.

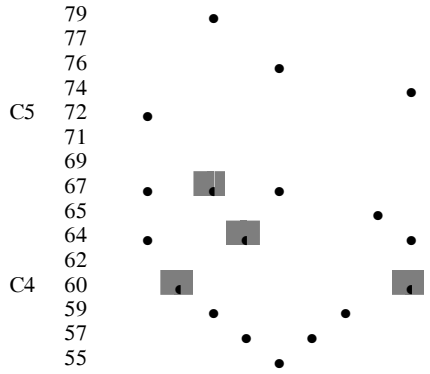


Figure 8

There are other degrees of freedom that need to be allowed in searching, in order to accommodate inaccurate recall of the user, and also inaccuracies in the data, for example if the musical information has been created by automatic transcription of either audio data or via optical recognition of the printed score. A fuller description of these areas of fuzziness is given in Crawford and Dovey (1999) [2].

3.3 A "piano roll" based algorithm

In Dovey (1999) [3], we presented a matrix-based algorithm for searching the music using the piano roll abstract model described above. That algorithm has underpinned much of our subsequent work, however there were some serious performance issues with the algorithm described there in pathological cases. The algorithm described here is a more refined version of that algorithm which performs in effectively linear time for a given text.

Let the text be represented by the sequence $<T_m>$ and the query by the sequence $<S_n>$ and consider the case when we can allow k intervening musical events in the text between the matches for consecutive events in the query.

In essence, we are performing a pass over the text for each musical event in the query. In the first case we are looking for the matches for the first event in the query to occur in the text. For all subsequent passes we are looking for an occurrence of the n^{th} term of the query occurring within k musical events of an occurrence of the $n\text{-}1^{th}$ term found in the previous pass.

Mathematically speaking, we construct the matrix M, where

M_{ij} = k+1 iff $S_j \subseteq T_i$ and ($M_{i-1\,j-1} \neq 0$ or $i = 0$)

$M_{i\,j-1}$-1 iff (not $S_j \subseteq T_i$) and ($M_{i\,j-1} \neq 0$)

0 otherwise

A result can be found by traversing the last row of the matrix constructed for the value k+1, this indicates by it horizontal position the match in the text for the last event in the query, reading to the left from this position in the row above for the value k gives the match for the penultimate event in the query and so on. We then repeat the process for each pitch transposition of the query sequence (i.e. where the value pitch of note in the sequence $<S_n>$ is transposed up or down by the same amount). We clearly need not consider any transposition such that the transposed sequence of $<S_n>$ and the sequence $<T_m>$ have not pitches in common.

In the worse case this algorithm can never take more that m x n steps to locate each pitch transposition, and the number of transpositions will never exceed the range of pitches which occur in $<T_m>$.

The following worked example would make this clearer. Let use consider the text in figure 4. Then our text is

T_0 = 72, 76, 64

T_1 = 60

T_2 = 79, 67, 59

etc.

Let us consider a simple query

S_0 = 67, 59

S_1 = 67, 55

S_2 = 59

For this case we allow one intervening gap, i.e. k=1.

We build the first row of the matrix by writing k+1 (i.e. 2) where S_0 contains all the pitches in T_i, otherwise if the value of the preceding square is non-zero we write that value less one, otherwise zero. So the first line becomes

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_0	0	0	2	1	0	0	0	0	0

For the second line we perform a similar operation for S_1, however we only write k+1 if all the pitches of S_1 are in T_i and the value of the preceding square in the row above is non-zero. So we now have

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_0	0	0	2	1	0	0	0	0	0
S_1	0	0	0	0	2	1	0	0	0

We then repeat for S_2 giving

	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_0	0	0	2	1	0	0	0	0	0
S_1	0	0	0	0	2	1	0	0	0
S_2	0	0	0	0	0	0	2	1	0

Note that although S_2 occurs within T_2 the preceding square above in the second row is zero, so we do not write k+1 (i.e. 2) here. The results can now be found in linear time by reading the matrix backwards looking for the occurrence of the value 2 i.e.

here $T_2\ T_4\ T_6$ is our match. We can then repeat for the other 15 possible pitch transpositions of the query.

There is an optimization to avoid unnecessary comparisons of musical events. For the row i of the matrix there exist a maximal s such that $M_{i\,j} = 0$ for all $j < s$. Similarly there exists a minimal t such that $M_{i\,j} = 0$ for all $j > t$. From this we can deduce that $M_{i+1\,j} = 0$ for all $j < s$ and $j > t+k$, i.e. we can limit our attention when constructing the next row of the matrix to the subsequence T_s, T_{s+1} … T_{t+k} of the text. Of course when $t < s$ there can be no matches for the query in the text. This optimization can cut processing time dramatically.

The above method forms the basis for more complex searching techniques where we have a query which in many ways resembles a regular expression for a musical phrase.

4. EXTENSIONS OF BASE ALGORITHM TO HANDLE REGULAR EXPRESSION TYPE QUERIES

4.1 Comparisons of Music Events

In the algorithm described in section 3, we considered only one way in which a musical event in the query can match a musical event in the text, namely that of inclusion. i.e. we say that a musical event S_j matches T_i if all the notes in S_j are in T_i (i.e $S_j \subseteq T_i$). Dovey (1999) [3] considers four such comparison operators:

- $S_j \subseteq T_i$ – as described here.
- $S_j = T_i$ – for exact matching
- $S_j \cap T_i \neq \emptyset$ – S_j here represents a disjunctive lists of notes we wish to match
- $T_i \subseteq S_j$ – for symmetry of the operators

This is generalized in Crawford and Dovey (1999) [2], so that these become just four of the most typical comparison operators in a lattice of all possible comparison operators with equals being the Top of this lattice and always matches the Bottom. In general we have some relationship **R** which defines whether two events "match" or not. Other typical relationships may include:

- $\mathbf{R_h}$: S_j and T_i are harmonically similar.
- $\mathbf{R_r}$: For each note in S_j there is a note in T_i whose pitch is within a given range of the pitch of the note in S_j
- $\mathbf{R_{r*}}$: For all but x notes in S_j there is a note in T_i whose pitch is within a given range of the pitch of the note in S_j

Our base algorithm described in section 3.3 can easily be extended to accommodate any such relationship **R** without any lose of performance. In this case given our relationship **R** we construct the matrix M where

$M_{i\,j}$ = k+1 iff S_j **R** T_i and ($M_{i-1\,j-1} \neq 0$ or $i = 0$)

$M_{i\,j-1}$-1 iff (not S_j **R** T_i) and ($M_{i\,j-1} \neq 0$)

0 otherwise

We can further generalize this. Given that we perform a pass over $<T_m>$ for each S_j in $<S_n>$, we can use a different relationship for each pass again without any loss in performance. i.e. we now have a sequence $<\mathbf{R_n}>$ where $\mathbf{R_j}$ describes the comparison operation for locating matches of S_j in $<T_m>$. In this case we construct the matrix M where

$M_{i\,j}$ = k+1 iff S_j $\mathbf{R_j}$ T_i and ($M_{i-1\,j-1} \neq 0$ or $i = 0$)

$M_{i\,j-1}$-1 iff (not S_j $\mathbf{R_j}$ T_i) and ($M_{i\,j-1} \neq 0$)

0 otherwise

This allows use for each event in the query to specify how the match will be found in the text using a polyphonic comparison range from exact match to more fuzzy matching comparisons.

4.2 Adding additional dimensions to notes

So far we have considered a note only to have a single value indicating its pitch. Clearly the algorithm described in section 3.3 and the enhancements described in section 4.1 would apply to any property. In the case of duration we would clearly be more interested in a comparison operator of the type $\mathbf{R_r}$ or $\mathbf{R_{r*}}$. For instrumentation or voicing we would be more interested in strict equality. If we define each note to be an n-tuple of the properties we are interested in we can perform a match against all these properties by defining a suitable composite comparison operator **R** without affecting the efficiency of our algorithm.

For example, in the case of three properties pitch, duration and instrument then **R** might behave as follows

> S_j **R** T_i if for almost all notes in S_j there is a note in T_j with a similar pitch, a similar duration and in the same voice (with some suitable parameters defining the ranges for similar and almost all).

4.3 Varying gaps between event matches

In section 4.1, we showed that we could have a different comparison operator for each term in the query. However, we still only have a single value of k, determining the allowed "gap" between matched events in the text, for the entire query. This also need not be the case. We can instead consider the sequence $<k_n>$ where k_j indicates the number of allowed gaps that can occur in the text after a match of S_j before a match of S_{j+1} occurs. k_n clearly has no meaning but must be non-zero for the algorithm to work properly. Modifying the generic form of the algorithm from section 3.3 gives

$M_{i\,j}$ = k_j+1 iff $S_j \subseteq T_i$ and ($M_{i-1\,j-1} \neq 0$ or $i = 0$)

$M_{i\,j-1}$-1 iff (not $S_j \subseteq T_i$) and ($M_{i\,j-1} \neq 0$)

0 otherwise

When reading the matrix for results we now look for the value k_j+1 to indicate matches where j is the current row of the matrix.

Modifying the form of the algorithm given at the end of section 4.1 gives

$M_{i\,j}$ = k_j+1 iff S_j $\mathbf{R_j}$ T_i and ($M_{i-1\,j-1} \neq 0$ or $i = 0$)

$M_{i\,j-1}$-1 iff (not S_j $\mathbf{R_j}$ T_i) and ($M_{i\,j-1} \neq 0$)

0 otherwise

So far we have considered the "gap" to be measured in terms of the number of events that can occur. Clearly in a piece of music the time between two consecutive events can vary. We can incorporate this into the algorithm by allowing the "gap" to be specified in terms of the time between matched events rather than the number of intervening events. We define an monotonic increasing function O on $<T_m>$ where

> $O(T_i)$ is the time at which T_i occurs.

The units could be milliseconds as in MIDI or could be based on a musical metric such as number of quarter notes.

In this case we set k to be the maximum allowed duration between matched events in the text. The base algorithm from section 3.3 now becomes

M_{ij} = k iff $S_j \subseteq T_i$ and (($M_{i-1\,j} > 0$ and $M_{i-1\,j} < k$) or $i = 0$))

$M_{i\,j-1}$ - ($O(T_i) - O(T_{i-1})$))

iff (not $S_j \subseteq T_i$) and ($M_{i\,j-1}$ - ($O(T_i) - O(T_{i-1})$)) > 0)

0 otherwise

Reading the results matrix now involves looking for the occurrences of the value of k. The modifications described in sections 4.1 and 4.2 can be applied to this form.

4.4 Omission of events in the query

The final modification to the base algorithm is to allow a match to be found even if some of the events in the query are not present in the matched subsequence of the text. Essentially when parsing the matrix we consider non-zero values in not only the line immediately above but also previous lines. To avoid excessive parsing of the matrix which would degrade the performance of the algorithm we can build a matrix of ordered pairs. For notational purposes, given an ordered pair p, we will denote the first value as p[0] and the second as p[1].

Working with the base algorithm from section 3.3 and allowing a sequences of up to l events from the query to be omitted from the match we build the matrix

M_{ij} = (k+1, l) iff $S_j \subseteq T_i$ and ($M_{i-1\,j-1} \neq (0, 0)$ or $i = 0$)

($M_{i\,j-1}[0]$ - 1, $M_{i\,j-1}[1]$) iff (not $S_j \subseteq T_i$) and ($M_{i\,j-1}[0] \neq 0$)

($M_{i-1\,j}[0]$, $M_{i-1\,j}[1]$ - 1) iff (not $S_j \subseteq T_i$) and ($M_{i-1\,j}[1] \neq 0$)

(0, 0) otherwise

Parsing the matrix for matches is a matter of looking for the value k+1 as the first member of any ordered pairs. Limiting the number of sequences omitted can be performed when parsing the matrix for results. Again the modifications described in sections 4.1, 4.2 and 4.3 can also be applied in conjunction with this modification.

4.5 An XML query structure

Combined these modifications allow us to search a musical text given a polyphonic query and a number of degrees of freedom. Considering just note pitches and durations, we can use the following XML structures such as the following to write a query allowing some of these degrees of freedom

```
<query omissions="0">
  <event
    following-gap=2
    following-gap.units="events"
    content-omissions.max="0"
    content-omissions.min="0">
     <note
       pitch.min="60"
       pitch.max="65"
       duration.min="1"
       duration.max="1"/>
  </event>
etc...
```

Here the **omissions** attribute of the **query** element tells us that we do not allow any events of the query to be omitted in the match. The **following-gap** attribute of the **event** element tells us the "gap" that can occur after this event in the text before the next event must occur; the **following-gap.units** whether this is measure in events or durations. The **content-omissions.min** and **content-omissions.max** tell us how many notes can be omitted from the match in order for it still to be classified as a match. The **pitch.min**, **pitch.max**, **duration.min** and **duration.max** attributes of the note element define ranges for a note in the text to match.

Whilst the algorithms described here can efficiently cope with this sort of query, there are other queries which can be handled which cannot be articulated in this XML structure.

5. FURTHER WORK

At present the algorithms described here merely locate matches given a number of degrees of freedom. There is no attempt to rank these matches. The calculation of a rank could be made as the matrix is parsed for matches and some pre-parsing could also be performed as the matrix is built. Crawford and Dovey (1999) [2] outline a number of similarity measures which could be used in creating a ranking algorithm such as completeness, compactness, musical salience, harmonic progression, rhythmic similarity, metrical congruity. This ranking process is essential before we can fully evaluate the effectiveness of this type of approach to music information retrieval.

Given the amount of freedom these algorithms allow in the specification of a query there is a need for query languages and GUI query interfaces to allow users to easily express such queries. Some work has already been undertaken in this area. The XML structure above is very much a working structure and not intended for general use.

6. ACKNOWLEDGEMENTS

I would like to thank Tim Crawford of Kings College London and Don Byrd of the University of Massachusetts at Amherst for their assistance in working on this topic and their patience in the amount of time it has taken to put together a description of these algorithms. I would also like to acknowledge the NSF/JISC IDLI and JISC DNER programmes which are funding the OMRAS and JAFER projects upon which this work is based.

7. REFERENCES

[1] Byrd, D. and Crawford, T. (2001) Problems of Music Information Retrieval in the Real World. To appear in Information Processing and Management.

[2] Crawford, T and Dovey, M. "Heuristic Models of Relevance Ranking in Musical Searching", *Proceedings of the Fourth Diderot Mathematical Forum.*, Vienna, 1999.

[3] Dovey, M, 'An algorithm for locating polyphonic phrases within a polyphonic piece', *Proceedings of the AISB'99 Symposium on Musical Creativity*, Edinburgh, April, 1999. Pages 48-53.

[4] Dovey, M, 'Adding content-based searching to a traditional music library catalogue server', *Proceedings of the Joint Conference on Digital Libraries*, Roanoake, VA, 2001.

[5] Downie, J. S., 'Music retrieval as text retrieval: simple yet effective', *Proceedings of the 22nd International Conference on Research and Development of Information Retrieval*, Berkeley, CA, 1999. Pages 297-298.

[6] Holub, J., Iliopoulos, C. S., Melichar, B. and Mouchard, L. 'Distributed String matching using finite automata', *Proceedings of the 10th Australiasian Workshop on Combinatorial Algorithms'*, Perth, 1999. Pages 114-128.

[7] Lemström, K. and Perttu, S., 'SEMEX – an efficient music retrieval protoype', *First International Symposium on Music Information Retrieval*, University of Massachusetts, Plymouth 2000.

[8] Selfridge-Field, E. (editor), *Beyond MIDI*, CCARH 1997. ISBN 0262193949.

[9] Uidebogerg, A. L. and Zobel, J. 'Manipulation of music for melody matching', *ACM Multimedia 98 Proceedings*, Bristol 1998. Pages 235-240.