Indexing Music Databases Using Automatic Extraction of Frequent Phrases

Anna Pienimäki
Department of Computer Science, University of Helsinki
PO Box 26, FIN-00014 University of Helsinki, Finland
+358 40 838 9350
Anna.Pienimaki@cs.Helsinki.FI

ABSTRACT

The Music Information Retrieval methods can be classified into online and offline methods. The main drawback in most of the offline algorithms is the space the indexing structure requires. The amount of data stored into the structure can however be reduced by storing only the suitable index terms or phrases instead of the whole contents of the database.

Repetition is agreed to be one of the most important factors of musical meaningfulness. Therefore repetitive musical phrases are suitable for indexing purposes. The extraction of such phrases can be done by applying an existing text mining method to musical data. Because of the differences between text and musical data the application requires some technical modification of the method.

This paper introduces a text mining-based music database indexing method that extracts maximal frequent phrases from musical data and sorts them by their length, frequency and personality. The implementation of the method found three different types of phrases from the test corpus consisting of Irish folk music tunes. The suitable two types of phrases out of three are easily recognized and separated from the set of all phrases to form an index data for the database.

Key words: music retrieval, indexing, text mining.

1. INTRODUCTION

The Music Information Retrieval (MIR) methods have developed rapidly since the introduction of the Query by Humming system [6]. Since then the main interest has moved from the signal processing tasks to representational and algorithmic issues. Also some aspects of musicology and cognitive psychology have been taken into consideration.

The MIR methods can be classified in many ways using different criteria. One of the most fundamental ways to classify MIR methods is to divide them into those that process digitalized audio signals using digital signal processing methods and those that process digital symbolic representations. Because of the complexity of polyphonic signals, most of the MIR systems use symbolic representations of musical data as their inner representations. Such representations usually are processed by using variations of existing string matching algorithms.

There are several different symbolic representations used as inner representations in MIR systems. The first MIR methods used the three letter alphabet describing the melodic contour of the monophonic melody [6]. Even though most of the existing representations describe each note with one or more symbols, there are also some compressed representations in which the object described is a combination of sequential notes [4, 3]. The functionality of most of the methods used depends strongly on the structure of the representation and applying them to process other representations requires modifications to the method. However there are few methods that are also capable to process a set of different representations [9, 10].

Another quite fundamental classification of MIR methods is based on the use of indexing. In the online methods the whole database is processed when executing a query while in the offline methods the query is executed using the index. Even though there are efficient index structures like suffix-tree [12], in many cases the whole index requires too much space. Therefore there is a need for new kinds of methods that also reduce the amount of the data stored into the index structure.

There are certain similarities in the use of text and musical data which allow also applying text mining methods to process musical data. When applying such text processing methods, one has to take care of the technical differences between text and musical data. The main differences include polyphony, multidimensionality of data items, and transposition invariance.

This paper introduces a text mining-based indexing method for symbolic representation of musical data. The method extracts maximal frequent patterns from musical data and sorts them by their length, frequency and personality. The found phrases or some subset of them can be stored into the index structure and used as index terms in the MIR system.

2. INDEXING MUSIC DATABASES

As mentioned above, MIR methods can roughly be categorized into online and offline methods. The offline methods can also be divided into two groups on the basis of the data stored into the index structure. Most of the offline methods use indexing which stores the whole contents of the database into a more efficient structure like a suffix-tree [12] or its variants [4]. The main drawback of this approach is the amount of space such an index structure requires [7]. Therefore the size of the structure is often reduced by using simple alphabets [4, 3].

Another approach to indexing is to use similar structures but store only some parts of the data into it, so the reduction of index size does not limit the alphabet [8]. In this approach the most descriptive musical patterns are extracted from the data and used as index terms. Even though the data might include many long meaningful patterns, the reduction of the amount of data is considerable. If the data includes many meaningful patterns, these may also be sorted by some of their properties. Because the repetition is widely agreed to be one of the most important factors that indicate the musical meaningfulness of the phrase, the extraction of the meaningful phrases is done on the basis of repetition in this work.

The extraction of repetitive patterns is of interest also in the area of text mining. When the analysed data is semi- or unstructured, the important index words or phrases can not be identified using the knowledge of the text structure. Because the most typical symbolic representations of musical data can be seen as unstructured data, the methods developed for the needs of text mining may also be applied to musical contexts.

Ahonen-Myka [1] has introduced a text mining algorithm, which extracts maximal frequent sequences from unstructured text data. Because of the properties of the Ahonen-Myka method, it is suitable for indexing unstructured musical data, as well. However, the different context requires some modifications and added parts to the algorithm.

3. APPLYING TEXT MINING METHODS TO MUSICAL DATA

The idea of applying the text mining method to musical contexts is based on some similarities between musical and text data. Musical data is comparable to text for many reasons in spite of the differences between music and language as phenomena. First, both musical and text data are considered to be hierarchical – the sentences or terms and musical phrases consist of smaller units like letters, words or notes. Second, in both musical and text data, the order of units is of importance. Third, it is possible that there are some additional units between the units of the meaningful phrase. In music such additional units are ornaments for example.

The ways music and language are used also have some similarities. Repetition is of high importance in conversation but also in written text. The important ideas, opinions or facts are stressed by repeating them with some variation. Even though in music there are no such direct opinions of facts, repetition still has quite an important meaning – repeating patterns are used as themes or hooks. Themes and hooks usually occur also with or without variation and are easily remembered [2].

In addition to repetition the variation and contrasts are also used when drawing one's attention to important ideas. Burns [2] stresses that repetition actually is quite insignificant without its counterpart variation. The more contrasting the variation is, the more likely it is to be noticed. However, the pure variation without any repetition is correspondingly meaningless.

When applying a text mining method to musical data, one has to take care of the above-mentioned technical differences between text and musical data. Two different approaches have been introduced to the processing of the polyphonic data. The simpler way is to process each voice separately [11]. This approach, however, does not find patterns, which occur between different voices. Another approach is to give a total ordering on all the notes [9]. In this approach a suitable gap sould be allowed between the note items when matching the pattern.

The multidimensional data can be represented using sets or vectors [9] that describe data items. In pattern matching the matched items are whole sets or vectors instead of individual letters or numbers. Even though some dimensions in sets or vectors might represent time events, their effect on pattern matching can easily be eliminated calculating the distinction between two vectors or sets instead of using absolute vectors or sets.

Using such calculated relative sets or vectors also solves the problem of transposition invariance – the similar themes starting from different pitches should be recognized to be the same. Even though the absolute pitch values of notes are not the same, the relation between pitches stays similar. Because the relative vectors represent these relations, themes are recognized to be transposes of each other.

4. EXTRACTING PHRASES FROM MUSICAL DATA

The indexing method described in this paper manages both homophonic and polyphonic data. In the inner representation the properties of notes are represented as vector dimensions. Therefore the algorithm is capable of handling many kinds of numerical symbolic representations of music.

The basic idea of the algorithm is to extract maximal frequent sequences from the given musical piece and sort them by their properties. The sequences, which are not subsequences of already found maximal sequences, are expanded to be maximal. A maximal frequent sequence is then a sequence, which appears at least σ times in a piece and which is not contained in another maximal frequent sequence.

The sequences are repeatedly constructed from shorter sequences until all the sequences of the same length are either subsequences of some maximal frequent sequence or not frequent. There may be at most α data items between items of sequence, enabling the treatment of the polyphonic data. Both σ (frequency) and α (gap) values are defined by the user.

The extraction algorithm is based on the Ahonen-Myka [1] algorithm with some extra phases. The technical requirements of the musical data have also been taken care of. The algorithm consists of three parts: the technical preprocessing phase, the discovery phase, and the sorting phase.

The algorithm takes symbolic representations of musical data, such as MIDI, as its input and uses the above-mentioned relative vectors as its inner representation. The properties of the data items that are used as vector dimensions are defined by the user.

4.1 Definitions

DEFINITION 1. *K-dimensional relative vector v' is the distinction $v_2 - v_1$ between two k-dimensional vectors v_1 and v_2.*

DEFINITION 2. *The relative data set D' is an ordered set of relative vectors v'.*

DEFINITION 3. *Phrase p is an ordered set of relative vectors v', having a maximal gap of α between every sequential vector v'.*

DEFINITION 4. *Phrase q is a subphrase of phrase p if each relative vector v' in q appear in the same order also in p.*

DEFINITION 5. *Phrase p is frequent in the relative data set D' if p appears at least in σ locations in D', σ being the given frequency threshold.*

DEFINITION 6. *Phrase p is a maximal frequent phrase in D', if there does not exist any phrase q which is also frequent in D' and p is a subphrase of q.*

4.2 Conversion Phase

The conversion phase includes conversion from MIDI data to inner representation which presents every note as a vector. In this phase the needed relative vectors within the limit of the gap value are also calculated and stored into the table structure. After the construction of the table, the unfrequent relative vectors are removed because they can not form any frequent phrases either. The table of frequent relative vectors is then used in several situations in the main phase.

ALGORITHM 1. *Conversion phase*

```
Input: MIDI file
Output: RVectors
   for each note in MIDI file
       convert note to a vector v
       Vectors := Vectors U v
   sort Vectors
   for each v_i in Vectors
```

```
        for each v_j in Vectors, v_i < v_j ≤ gap + 1
            RVectors[v_i][v_j] = v_j − v_i
            Loc[v_j − v_i] := Loc[v_j − v_i] U i,j
    for each v' in Loc
        if |v'| < frequency
            remove instances of v' from RVectors
    return RVectors
```

4.3 Discovery Phase

The discovery phase is very similar to the Ahonen-Myka algorithm [1]. The first preprocessing phase uses the relative vector table to form frequent pairs within the limit of the gap value. The found pairs and their locations are stored into the 2-Grams set.

ALGORITHM 2. *Preprocessing*

```
Input: RVectors
Output: 2-Grams
    for i := 0 to |RVectors| − 2
        for j := (i + 1) to (i + gap + 1), j
          ≤ |RVectors| − 1
            if head := RVectors[i][j] exists
                for k := (j + 1) to (j + gap + 1), k
                  ≤ |RVectors|
                    if tail := RVectors[j][k] exists
                        pair := (head,tail)
                        2-Grams[pair] := i,j,k
    for each pair in 2-Grams
    if |pair| < frequency
        remove pair from 2-Grams
    return 2-Grams
```

The main phase takes the 2-Grams set as its input. For each round of the main loop each gram in the k-Grams set is first compared to the phrases in the maximal frequent phrase set Max. If the gram is not a subphrase of any of the found maximal frequent phrases it is then expanded to be maximal. The found maximal phrase and all of its locations are stored into the set Max.

If the gram was already maximal before the expansion phase it is removed from the k-Grams set. The remaining grams are joined to each other to form $k + 1$ sized grams which are then stored into the new k-Grams set, the value of k being now increased. When storing the grams into the set, the unfrequent new grams are being removed. The algorithm stops when all of the grams are removed from the k-Grams set because of their unfrequency or maximality. The k-Grams set can also be pruned before the joining phase.

ALGORITHM 3. *Main phase*

```
Input: 2-Grams
Output: Max
    k := 2
    Max := ()
    while k-Grams is not empty
        for each k-gram g
        if g is not a subphrase of any m in Max
            max := Expand(g)
            Max := Max U max
            if max = g
                remove g from k-Grams
        (k-Grams := Prune(k-Grams))
        k-Grams := Join(k-Grams)
        k := k + 1
    return Max
```

The expansion phase takes a gram p as its input and repeatedly searches for the new frequent gram q, $|q|$ being $|p| + 1$. The gram p can be expanded by adding one vector at the beginning, end or in the middle of the gram. The possible expansion vectors are found from the frequent relative vector table.

ALGORITHM 4. *Expansion phase*

```
Input: p
Output: p'
    repeat
        let l be |p|
        find phrase p', |p'| = l + 1, p is a subphrase
          of p'
        if p' is frequent
            p := p'
    until frequent p' can not be found
    return p
```

The joining phase joins k sized grams to form $k + 1$ sized grams. Per each two grams $p = a_1, ..., a_k$ and $q = b_1, ..., b_k$ the algorithm examines if the subgrams $p' = a_2, ..., a_k$ and $q' = b_1, ..., b_{k-1}$ are similar and then joins them to form a new gram $pq = a_1, ..., a_k, b_k$. The frequency of the new grams is examined at the end of the phase.

ALGORITHM 5. *Join*

```
Input: k-Grams
Output: (k + 1)-Grams
    for each gram g_i = a_1, ..., a_k
        for each gram g_j = b_1, ..., b_k
            if a_2, ..., a_k = b_1, ..., b_{k-1}
                gram := a_1, ..., a_k, b_k
            (k + 1)-Grams[gram] := location(gram)
    for each gram in (k + 1)-Grams
    if |gram| < frequency
        remove gram from (k + 1)-Grams
    return (k + 1)-Grams
```

Because some of the grams may expand to many maximal frequent phrases, all of the phrases may not be found during the first round of the main loop. Therefore only the unfrequent and already maximal grams can be removed from the set of grams in every round. Without the pruning phase, the algorithm takes $l-1$ rounds, l being the length of the longest found maximal phrase. When using the pruning phase, the amount of the needed rounds is much smaller.

The pruning phase generates per each gram $g = a_1, ..., a_k$ two sets of phrases, LMax and RMax. The set LMax consists of the maximal phrases that have gram $g' = a_1, ..., a_{k-1}$ as its subphrase. The set RMax is generated similarly now using the subgram $g'' = a_2, ..., a_k$. For each phrase in LMax the possible suffixes for the gram g' are extracted and stored into set LStr. The set RStr is generated similarly by extracting prefixes for the gram g''. Each combination of suffixes, gram and prefixes are tested against the found maximal phrases. If a new frequent maximal phrase is found, all of its subphrases are examined and the k-grams of those that are not subphrases of any maximal phrase are marked. When all the grams of the set k-Grams are processed, grams that are not marked are removed from the set.

ALGORITHM 6. *Prune*

```
Input: k-Grams
Output: k-Grams (pruned)
    for each g = a_1, ..., a_k in k-Grams
```

```
    LMax = {p|p in Max and a_1, ..., a_{k-1} is a
        subphrase of p}
    RMax = {p|p in Max and a_2, ..., a_k is a
        subphrase of p}
    for each p in LMax
        LStr[g] = {b_1, ..., b_i|a_1, ..., a_{k-1} exists
            in p starting from location b_{i+1}}
    for each p in RMax
        RStr[g] = {b_{m+1}, ..., b_n|a_2, ..., a_k exists
            in p ending at location b_m}
    for each s1 in LStr
        for each s2 in RStr
            s-new = s1 . g . s2
                if s-new is not a subphrase of
                    any m in Max
                    for each frequent subphrase s
                        of s-new
                        if s is not a subphrase of
                            any m in Max
                            mark each k-gram in s
for each g
    if g is not marked
        remove g from k-Grams
return k-Grams
```

4.4 Sorting Phrases

When all of the maximal frequent phrases are found, the phrases are sorted by their length, frequency and personality. The personality property measures the average divergence between the sequential notes. The idea of personality is based on contrasts – the more contrasting the sequential notes on average are the more personal the phrase is considered to be.

The personality value for each phrase is calculated using the scaled values of each vector dimension. The personality of one phrase is the average of the personality of its vectors: $\frac{|a_{11}|+...+|a_{1k}|+...+|a_{nk}|}{n}$ where n is the length of the phrase, k the cardinality of the vector and a_{ij} is the scaled value of the jth dimension of the ith vector. The total rate for each phrase is calculated using the formula: $x * length + y * frequency + z * personality$, x, y, and z being the user defined parameters for sorting.

ALGORITHM 7. *Sort*

```
Input: Max
Output: Maxsorted
    for each phrase p in Max
        length := |p|
        frequency := |locations|
        for each vector v = a_1, ..., a_n in p
         personality := personality + |a_1|+...+|a_n|
            personality := personality / length
            rate := (x*length + y*frequency +
              z*personality)
            Maxsorted[rate] := Maxsorted[rate] U p
    return Maxsorted
```

Because the number of found phrases can be quite high, depending on the analysed data, it may be useful to choose only the best phrases from the set of all found phrases. The chosen phrases can then be stored into the index structure.

4.5 Requirements for the Search Algorithms

The found phrases or some subset of them can be stored into a suffix-tree structure [12] for example. A query can be executed using the existing string matching algorithms. The form of the found phrases, however, sets some requirements for the search algorithms.

As mentioned above, the polyphonic structure is managed by allowing gaps between the notes of the phrase. Therefore some of the found phrases may also include chords in addition to melodic structure. Because the phrases are stored in their entirety the search algorithm should be able to process such partly polyphonic structures, as well.

4.6 Open Problems

One of the main principles of the algorithm is to support different numerical symbolic representations of musical data. In this approach the resulting phrases and also the managing of some polyphonical situations are highly dependent on both the given representation of the data and the parameter values. The order of the found phrases depends on the user-given parameters, as well. The main drawback in this approach is that it does not use any knowledge of the order or type of the given vector dimensions and that it is highly user dependent.

Even though the personality property adds some semantics into the sorting phase, the idea of contrast or originality has to be considered further and some new properties added into the sorting formula. Some other sorting formulas could be added to the algorithm, as well.

5. IMPLEMENTATION AND RESULTS

The two versions, with and without pruning, of the algorithm were implemented in Perl and experiments were carried out with the MIDI database consisting of 130 polyphonic Irish folk music pieces. The pieces consisted of 300–3500 notes and they were divided into 11 size groups (Table 1). Two of the groups ($600 \leq x < 800$ and $800 \leq x < 1000$) were experimented systematically, the others were used as comparison material. The inner presentation was formed using the pitch and starting-time values. The hardware environment was AMD Athlon 1.33GHz with 512MB of main memory.

Table 1: The size groups.

Notes	Pieces
$x < 600$	9
$600 \leq x < 800$	8
$800 \leq x < 1000$	19
$1000 \leq x < 1200$	19
$1200 \leq x < 1400$	22
$1400 \leq x < 1600$	11
$1600 \leq x < 1800$	16
$1800 \leq x < 2000$	8
$2000 \leq x < 2200$	7
$2200 \leq x < 2400$	4
$x \geq 2400$	7

The experiments were also divided into three sets. The first set of experiments measured the easily recognizable features like the direct effect of amount of starting pairs and parameters (Figures 1 and 2). The second set measured the effects of both parameters and the data contents and the third set the effect of pruning on the efficiency of the algorithm. The resulting phrases sorted by their rate and the time analysis data were collected into a result file.

The first set of experiments was conducted using the version of the algorithm without pruning phase and five different sets of parameter values. The experiments proved that the parameters had a direct effect on the amount of starting pairs (Figure 2) but the exact amount of notes did not. However, the average execution time rose when executing pieces of the next size group (Figure 3). The most interesting observation, however, was that the algorithm had two different behavior models. In case of some pieces and parameter values, especially when the amount of starting pairs was quite small,

for example less than 200, the amount of the grams in the k-Grams set started to decrease from the first round. In other cases, the amount of grams first increased and then decreased.

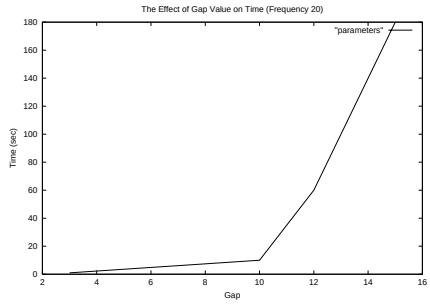

Figure 1: The effect of gap value on time in the second size group

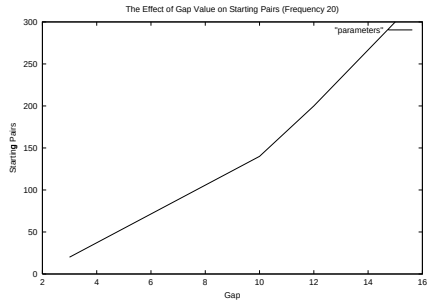

Figure 2: The effect of gap value on starting pairs in the second size group

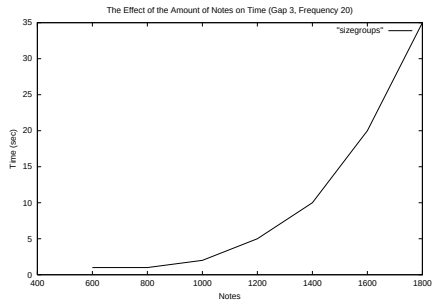

Figure 3: The effect of amount of notes on time

The second set of experiments was also conducted using the simpler version of the algorithm. In this set, both decreasingly and increasingly behaving pieces of second and third size groups were processed using the same parameter values and the resulting phrases were analysed. The results proved that the decreasing behavior ended up with a large set of small phrases consisting of only a couple of notes. When the behavior was increasing the set of phrases included noticeably longer phrases with additional smaller phrases as well (Table 2).

Table 2: The length of the phrases in the second and third size groups with parameter values (10, 20).

Phrases	Increasing(pieces)	Decreasing(pieces)
Ave. Length	8 (11)	4 (16)
Longest	20	7

The third set of experiments measured the efficiency of the two versions of the algorithm. On the average the algorithm with pruning phase was twice as efficient as the one without pruning. In case of the decreasing behavior, however, the version without pruning was considerably more efficient (Table 3).

Table 3: The average efficiency of the algorithm in the second size group with and without pruning phase with parameter values (3, 10).

	Increasing	Decreasing
Without Pruning	1.5 min	3 sec
With Pruning	0.5min	6 sec

Even though the pruning phase was relatively slow when the algorithm behaved decreasingly and also in situations, when there were many maximal phrases with common subphrases, there are still reasons for using the phase. On the average the pruning phase speeded up the algorithm considerably. Also the phrases produced by the algorithm when behaving decreasingly were very short and therefore useless.

When comparing to the original text mining method the behavior of algorithm was slightly different. The text phrases are typically much shorter than the phrases used in music which was also seen when expanding the phrases. Further, there seemed to be much more repetition in the musical data used as corpus than in the text in general.

The found phrases were of three different types. The phrases of the first type were very short consisting of three to five notes which typically represented chords with one or two extra notes. When searching a melody the information consisting mainly of chords is irrelevant. Therefore these phrases are unsuitable for indexing purposes. Typically this group of phrases was generated when the given gap value was too small.

The phrases of the second group were considerably longer and consisted of both melodic and harmonic parts. The most common form of this kind of a phrase started with a chord followed by one or two sequential melodic notes. A similar structure was repeated several times in the phrase. The third group consisted of purely melodic phrases having approximately five to ten notes.

When examining the phrases of the second and third group closer, it was noticeable that most of the phrases consisted of such repeating structures that could also be heard when listening to the piece. Therefore these phrases also form the set of the phrases to be stored into the indexing structure.

In most cases the phrases of the second and third group had the highest rate values, as well. Still there is a need for finding better sorting formulas for stressing the factors of the rate value.

Because the behavior of the algorithm and therefore also the form of the found phrases depended partly on the parameters given by the user, the suitable parameter values should be determined for each corpus individually. Therefore there is also a need for an algorithm which would analyse the data and decide the best parameter values for each data.

6. CONCLUSIONS

This paper presented a text mining-based indexing algorithm, which extracts repeating patterns from a symbolic representation of polyphonic musical data. The patterns found are first sorted to separate useful and unuseful phrases. The best phrases can then be stored into the index structure.

Although the phenomena of music and language are quite different to each other, especially their use have similar features in many cases. Further, the musical and text data have enough similarities for also applying the text processing algorithms to musical data. Therefore the knowledge gained from the text mining and language technology research can also be useful for further development of MIR methods.

7. ACKNOWLEDGEMENTS

The work reported in this paper has been supported by Finnish Academy grant #68657.

8. REFERENCES

[1] H. Ahonen-Myka. Finding All Frequent Maximal Sequences in Text. In *Proceedings of the 16th International Conference on Machine Learning ICML-99*, Ljubljana, pp. 11–17

[2] Burns, G., A Typology of 'Hooks' in Popular Records. *Popular Music* 6, 1 (1987), pages 1–20.

[3] J. Chen and A. L. Chen. Query by Rhythm: An Approach for Song Retrieval in Music Databases. *IEEE International Workshop on Research Issues in Data Engineering*, pages 139–146, Orlando, Florida, USA, February 23–24, 1998.

[4] T. C. Chou, A. L. Chen, C. C. Liu. Music Databases: Indexing Techniques and Implementation. *IEEE International Workshop on Multi-Media Database Management Systems*, pages 46–53, Blue Mountain Lake, New York, USA, August 14–16, 1996.

[5] J. Hsu, C. Liu, and A. L. Chen. Discovering nontrivial repeating patterns in music data. *IEEE Transactions on Multimedia*, Volume 3, 3 (2001), pages 311–325.

[6] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query By Humming – Musical Infromation Retrieval in an Audio Database. *Proceedings of the ACM Multimedia'95*, 1995, pages 231–236.

[7] K. Lemström. In Search of a Lost Melody. Computer Assisted Music: Identification and Retrieval. *Finnish Music Quarterly*, 3–4 (2000), pages 40–45.

[8] K. Lemström, G.A. Wiggins and D. Meredith. A Three-Layer Approach for Music Retrieval in Large Databases. In *Proceedings of ISMIR 2001 2nd Annual International Symposium on Music Information Retrieval*, pages 13–14, Bloomington, Indiana, 2001.

[9] D. Meredith, G.A. Wiggins and K. Lemström. Pattern induction and matching in polyphonic music and other multidimensional datasets. In *Proceedings of the Fifth World Multiconference on Systemics Cybernetics and Informatics (SCI2001)*, pages 61–66 (vol X), Orlando, FL, 2001.

[10] D. Meredith, K. Lemström, and G.A. Wiggins. SIATEC and SIA: Efficient algorithms for translation-invariant pattern-discovery in multidimensional datasets, prep.

[11] Rolland, P.-Y. Discovering patterns in musical sequences. *Journal of New Music Research*, 28, 4 (1999), pages 334–350.

[12] E. Ukkonen. On-line construction of suffix-trees. *Algorithmica* 14 (1995), pages 249–260.