Evolutionary Search for Musical Parallelism

Søren Tjagvad Madsen[1,2], Gerhard Widmer[1,2]

[1] Austrian Research Institute for Artificial Intelligence,
Freyung 6/6, A-1010 Vienna, Austria
[2] Department of Computational Perception, University of Linz, Austria
soren@oefai.at, gerhard.widmer@jku.at

Abstract. This paper presents an Evolutionary Algorithm used to search for similarities in a music score represented as a graph. We show how the graph can be searched for similarities of different kinds using interchangeable similarity measures based on *viewpoints*. A segmentation algorithm using the EA for automatically finding structures in a score based on a specific-to-general ordering of the viewpoints is proposed. As an example a fugue by J. S. Bach is analysed, revealing its extensive use of inner resemblance.

1 Musical similarity

Repetition or parallelism is a fundamental feature of western tonal music. Ockelford cites musicologists arguing that music is a self-contained art form [1]. Music can not refer to the phenomenal world as well as other art forms are able to, so the mind's longing for reference can only be satisfied through repetition.

This paper addresses the search for parallelism in a symbolic representation of music. Looking at the music as symbols or events, parallelism can appear as repetition of phrases, but also as systematic changes in the quantitative information of the notes/events (pitch and duration), or through elaborations and simplifications of phrases (inserting/deleting notes). The artistic compositional effects can be varied infinitely.

Our approach is an attempt to see how far one can go by analysing western tonal music based on musical similarities solely calculated from information present in a score. We will demonstrate this with a fugue by J. S. Bach.

West et al. introduce the term *transformation* when a musical object (composed of a number of events/notes) can be inferred from another musical object in a perceptible way, and when it is possible to specify the transforming function [2]. This function determines the type of their relation and describes in what way they can be said to be different inflections of the same musical idea. Our strategy will be to carefully define a set of transformation functions, and then to use them in searches.

What types of transformation functions could we expect? The problem is often divided in two. The easiest task is describing similarities involving a one-to-one correspondence between the notes in the respective phrases – transformations that do not change the number of notes, but only their parameters. We will call these simple transformations. This could for example be the transposition of a phrase. The real challenge is to find phrases that sound similar while

allowing elaboration/simplification (inserting/deleting notes). To detect such a non-simple parallelism there is not a single strategy, since there are numerous ways of embellishing the music and consequently many transformation functions. Although the non-simple transformations are quite important in music, we will in this paper concentrate on the simpler case of defining (and finding) similarities between equal-numbered groups of notes. Simple transformation functions are quite common means of creating parallelisms, and can be thought of as systematic changes of the notes like transpositions (of various types) and inversions. A small set of these transformations are able to account for a great deal of similarities in western tonal music.

Describing musical similarity and proposing similarity measures is a field which has already been given much attention, due to its central position in automatic music analysis (see, e.g., [3–5], to name but a few). Segmentation based on similarities has also been studied by [6]. We have built our similarity measures upon the ideas by Conklin et al. [7, 8]. Their method of computing similarities are defined on sequences of notes solely. We have adopted this sequential approach to the problem, which turns out to work fine for the fugue analysed here.

2 Data acquisition and representation

To be able to analyse music in the best way, we decided to depend mainly on a reasonably detailed music source: the MuseData format (`www.musedata.org`). This source contains information about enharmonic pitch spelling. We can use MIDI as input too, but MIDI lacks the diatonic information (relating every note to a 7 step scale) that is crucial when searching for harmonically related phrases.

Musical events are put into a graph structure – the *music graph*. Each note/rest or 'event' in the score is represented as a vertex containing the pitch (spelling, octave, alterations), duration, start time in the score, key, and time signatures etc. The temporal relations of the notes/rests are present as directed edges of the graph. Vertices representing events following each other in time (a note/rest that starts immediately where another ends) are connected by an edge of type *follow*. Notes with the same start time are connected by *simultaneous* edges. The graph representation does not bias the representation of the music to be mainly 'vertical' (homophonic) or 'horizontal' (polyphonic), but any connected subset of the notes – a subgraph – can be considered as an entity of the music. The features of this graph representation is explained in detail in [9]. For the experiments explained here, *follow* edges only exist between vertices belonging to the same voice in the score. The MuseData representation and scores in general are by nature divided into such voices or parts.

For practical reasons, subgraphs can either be sequential (graphs of vertices connected by edges of type *follow* only) or non-sequential. The similarity measures presented in this paper are defined for sequences of notes and the music analysed is mainly polyphonic, so in this paper we are searching for similar sequential subgraphs. [9] presents a way to extend the similarity measures to the more general case of comparing any (equal sized) subgraphs. We will from now on refer to sequential subgraphs as simply subgraphs.

3 The Search Algorithm

The EA maintains a population of *Similarity Statements*. A similarity statement (SS) is a guess that two subgraphs of the same size (same number of vertices) are similar. The population is initialised with SSs each pointing at two random subgraphs of the 'mother' graph. By doing crossover, mutation, and selection in terms of altering the guesses it is possible to change the SSs to point at increasingly more similar subgraphs, dynamically adjusting their size and position.

A new generation is composed of three parts. A given percentage is chosen through tournament selection (tournament size of 2), another percentage is created by crossover of two selected individuals, and the remaining percentage is created through mutation of the selected or crossbred individuals. Finally, after a new population is created, the mutation rate is also used to determine a number of random mutations that are applied to the new generation. The most fit similarity statement in each generation always survives to the next (elitism). (The numbers used in the presented experiment were: selection: 0.45, crossover: 0.05, and mutation 0.5).

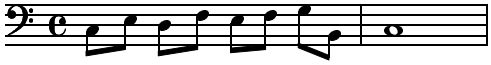
Mutation on a similarity statement with sequential subgraphs s_1 and s_2 can take several forms. The different mutation operations add and remove edges and vertices to/from the subgraphs. It should be noted that subgraphs are implemented as pointers to a subset of the vertices and edges in the 'mother' graph. Mutations on subgraphs do not change the graph structure itself. Common to all operators is that they must preserve the constraints that make s_1 and s_2 sequential (connected and only including edges of type 'follow').

- **Extension** Extend both s_1 and s_2 once, either at the 'beginning' of the subgraph (against the *follow* edge direction) or at the 'end' (along the follow edge direction), chosen at random and independently for each graph. The size of both subgraphs is increased by 1. (Applied with probability 0.3)
- **Shortening** Shorten both s_1 and s_2. As with extension, a subgraph can be shortened at either of the ends chosen at random and independently for each graph. The size is decreased by 1. (Probability 0.3)
- **Slide** Slide both subgraphs once along or against the follow edge direction (s_1 and s_2 do not need to slide the same way). The size of the subgraphs is not altered by a slide. (Probability 0.3)
- **Substitution** Substitute either s_1 or s_2 with a new and randomly generated subgraph of the same size. (Probability 0.1)

The crossover operation takes two SSs and combines them by picking one subgraph from each (chosen at random). The two parent statements most often have different sizes, so the smallest of them is extended at random until the sizes match. However, the chances that this will improve the fitness are low, because often the two subgraphs that are chosen are very different. As a result, the crossover parameter is often set low.

3.1 Evaluation

The fitness function evaluates the degree of similarity of two equal sized subgraphs according to an interchangeable similarity measure. The fitness function has to balance four conflicting goals:



Viewpoint	View
Absolute MIDI Pitch	[48,52,50,53,52,53,55,47,48]
MIDI Pitch Interval	[4,-2,3,-1,1,2,-8,1]
Pitch Class	[0,4,2,5,4,5,7,11,0]
Diatonic Absolute Pitch (A)	[C3,E3,D3,F3,E3,F3,G3,B2,C3]
Diatonic Pitch Interval (T)	[2,-1,2,-1,1,1,-5,1]
Diatonic Interval mod 7 (M)	[2,6,2,6,1,1,2,1]
Diatonic Inversion Interval	[-2,1,-2,1,-1,-1,5,-1]
Absolute Duration	$[\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},\frac{1}{8},1]$

Fig. 1. Example of the viewpoints.

- Optimise musical similarity
- Prefer larger matches to smaller ones
- Prefer phrases that conform to grouping structure rules
- Forbid overlap between subgraphs in any SS

The size of the graphs in a SS contribute in the fitness calculation to make the algorithm explore larger matches even if a perfect one is already found. A decreasing function of the size of the subgraphs is added to the fitness to make the EA prefer longer matches.

An evaluation of both subgraphs' agreement with *grouping structure* (a quick implementation of some simple rules suggested by [10]) is also included in the evaluation. The rules essentially propose a computable way of telling how well the two phrases individually correspond to phrase boundaries in the music. Even though only a few rules are implemented, the effect is noticeable. There is unfortunately no space here to go into details about this.

No overlap between two subgraphs in a similarity statement is permitted. We want phrases to be non-overlapping so we later can substitute the subgraphs in an unambiguous way. Overlap result in a bad evaluation.

We have used the notion of *viewpoints* presented in [7] when calculating the musical similarities between subgraphs. A viewpoint is a function taken on a sequence of musical objects, yielding a *view* of the sequence. The sequence is in our case a sequential subgraph, and the view is a list of values. Figure 1 shows a note sequence and examples of viewpoints and corresponding views. As shown, values from both the pitch and time domains can be included (other information available could also be used). Viewpoints can either be absolute – using values from each note – or relative – calculated from the relation between notes. The list is by no means complete.

To compare if two subgraphs of equal length are equal under a given viewpoint, we compare their views. This is simply done by counting the number of pairwise disagreeing values in the view vectors. When this *view difference* evaluates to zero, the sequences are equal under this view.

A motif and its exact repetition have the same view under the absolute pitch viewpoint A (the identity transformation, 'Absolute') and the pitch interval viewpoint T ('Transposition'). A motif and its transposed version will however have different views under A, but equal under T. A is said to be more specific than T. In the segmentation of the fugue presented in Sec. 5, we chose to use three viewpoints from the pitch domain – the ones denoted as A, T and M in Fig. 1. They can be ordered by pitch in this way:

$$DiatonicAbsPitch(A) \leq DiatonicInt(T) \leq DiatonicIntMod7(M) \quad (1)$$

If two patterns are found similar (regarding pitch) under a viewpoint, they will also be equal under a less specific viewpoint.

The fitness function combines evaluations of the above mentioned parameters and selected viewpoints into a single value. The global optimal solution is thus based on a mixture of melodic similarities and the size and boundaries of the subgraphs. In the measures used for the fugue segmentation we have adjusted the weighting of these parameters in such a way that pitch is the dominating factor.

4 The segmentation algorithm

We now perform multiple runs of the EA to discover and categorise different kinds of similarities inside a piece. Segmenting a performance consists in iteratively using the EA to find similar passages in the piece. When an EA terminates, the most fit SS is evaluated against a threshold, determining if the fitness is 'good enough' for the music to be claimed similar.

The EA is also used to search for more occurrences of a pattern – the occurrence search. It works by keeping one subgraph fixed in each SS (the original pattern) while doing crossover and mutations on the other, never changing the size of the graphs. The multiple EA searches are done with different similarity measures in a special order, controlled by the segmentation algorithm. The goal of the segmentation algorithm is to find patterns and derivations and categorise motifs with as specific a measure as possible.

When the segmentation algorithm finds a subgraph sufficiently similar to another, it removes it from the original graph and replaces it with a *compound vertex* (CV) now representing the pattern. This CV is labelled with the similarity measure it was evaluated with (its relation), as well as an identifier representing its 'type'. The edges to and from the subgraph that was substituted now connect the CV. A subsequent EA run on the updated graph now needs to take account of CVs in the graph and compare these on equal terms with other vertices. CVs can then be included in other subgraphs making further nesting possible.

A CV can only be similar to another CV if it has the same type (representing the same motif), and CVs should only be included when they were not found found with a more general measure than the current. This is necessary to preserve the relation of the measures that the CVs were labelled with. Otherwise it would, e.g., be possible to allow transposed phrases (labelled T) to be part

of a compound labelled with an absolute measure A, which is wrong since the phrases are not copies of each other.

```
SimilaritySegmenter(array of similarityMeasures, MusicGraph) {
  i=-1
  for(SimMeasure sm_i = 1..k){
      //search for new repeated patterns
      while(it is possible to find similar patterns with sm_i){
          let s_1 and s_2 be the most similar subgraphs of a EA run with sm_i
          if( SimMeasure(s_1,s_2) is within the similarity threshold){
              i++
              substitute s1 and s2 in the graph with CVs of type i and measure sm_i
              for(SimMeasure sm_j = i..k){
                  while(there are more occurrences of one of s_1 and s_2 to find with sm_j){
                      let s_n be the graph found in an Occurrence search for s_1 with sm_j
                      if( SimMeasure(s_1,s_n) within threshold)
                          substitute s_i in the graph with a CV of type i and measure sm_j
                      let s_n be the graph found in an Occurrence search for s_2 with sm_j
                      if( SimMeasure(s_2,s_n) within threshold)
                          substitute s_i in the graph with a CV of type i and measure sm_j
                  }
              }//advance to next SimMeasure in the occurrence search
          }//no more patterns to be found with this SimMeasure
      }//advance to next SimMeasure
} }
```

Fig. 2. The segmentation algorithm

The segmentation algorithm resembles the one proposed by [11] in the way all derivations of a recurring pattern are found before a new pattern is addressed. The idea is to iteratively find new patterns with as general a measure as possible (in successively weaker order) and next to find all derivations of it in successively weaker order. A new pattern is not considered before all derivations are believed to be found (when the search for more occurrences fails). The algorithm (shown in Fig. 2) takes as input the graph, and a list of (ordered) similarity measures. The algorithm iterates through a double for-loop. Both of the loops start EA searches with the similarity measures in the order given. The outer loop searches for new patterns and the inner searches for occurrences of these. The similarity measure sm_i advances one step (becomes more general) every time the while-condition is not fulfilled: when it is not possible to find a pattern with sm_i. In the inner loop sm_j advances when it is not possible to find more occurrences with sm_j. The segmentation terminates when the EA fails to find a 'good' repeated pattern with the most general similarity measure.

An alternative to the strict segmentation order would be to calculate all pitch related viewpoints in every evaluation and grade the fitness according to similarity in highest pitch specificity. That evaluation approach resembles the one presented in [12]. A more drastically but less controllable change would be to always evaluate all implemented viewpoints and see which one that 'clicks' and based on this information calculate a rating or a description of the similarities of the subgraphs (e.g. "s_1 is a rhythmically augmented diatonic inversion of s_2"). We plan to do experiments along these lines.

5 Experiments: Segmenting a fugue

We will present a single segmentation by the algorithm: an analysis of J.S. Bach's Fugue in C minor, BWV 847 from WTK Book 1. This will illustrate how well the search algorithm does the job of selecting phrases and finding derivations.

The threshold was set so that only patterns having identical views regarding pitch (under the given viewpoint) were considered similar. Duration and grouping were given less importance. The EA was generously given 800 generations for finding new patterns and 500 generations for finding occurrences. The population size was 120 similarity statements. The segmentation was set to terminate when the best pattern found was of size 2 or it's fitness above the threshold. The segmentation algorithm found 19 patterns (some of them extending others) of sizes 3-26 notes/rests and made a total of 72 substitutions.

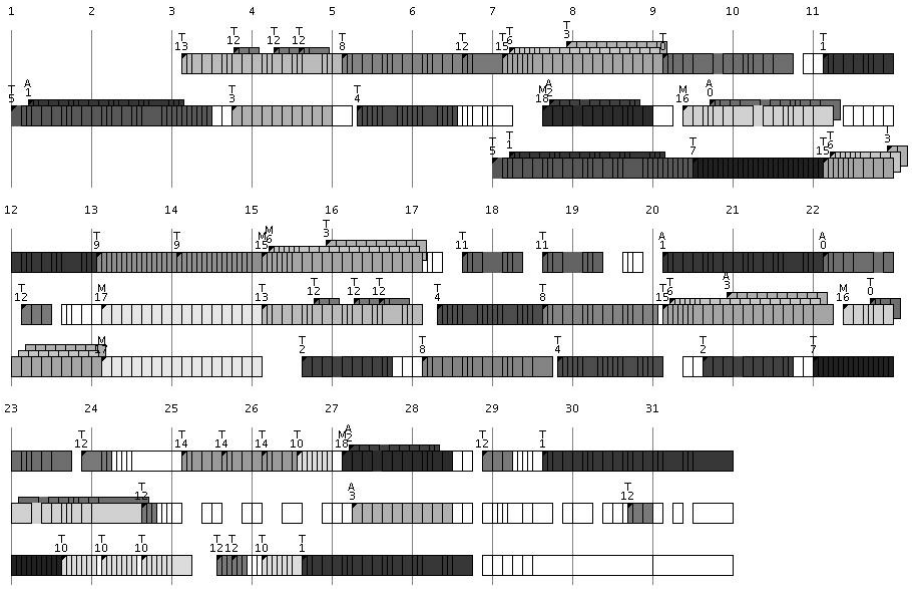


Fig. 3. A segmentation of J. S. Bach, Fugue in C minor

A graphical representation of the segmentation is shown in Fig. 3, giving an overview of the musical material found in the piece. The figure shows the entire three part fugue. Each part is represented as a row of boxes. Every note in the score is represented by a box. The width of the box shows the relative duration of the note. Rests are not printed but present as 'missing boxes'. Every compound found in the segmentation is labelled with a number and a letter indicating the measure with which it was found similar to another pattern. The beginning of a compound is also indicated by a black triangle. Every compound type is given a unique shade of grey (unfortunately not too clear in the figure). White boxes

are thus notes which have not been found belonging to any pattern. Subpatterns in a nested compound are shown in the background. Bar numbers are printed above the vertical bar lines.

The pattern found in the second iteration (iteration 1) is exactly the main theme, traditionally called *dux*. It has six occurrences in the fugue, and all were found (see Fig. 4 – the numbers and letters indexing the patterns refer to voice (1, 2 or 3) and similarity measure (A, T or M)). The only two occurrences that are exactly the same in pitch (A) were found first (beginning in bars 1 and 20) – the rest were found as transpositions (T). The ending dux (bar 29) differs only on the ending note, which has been transposed half a tone. The occurrences starting in bar 7 and 26 are octave transposition of the theme. The dux in bar 11 is a 'major' version of the theme, starting on $e\flat$ in stead of c.

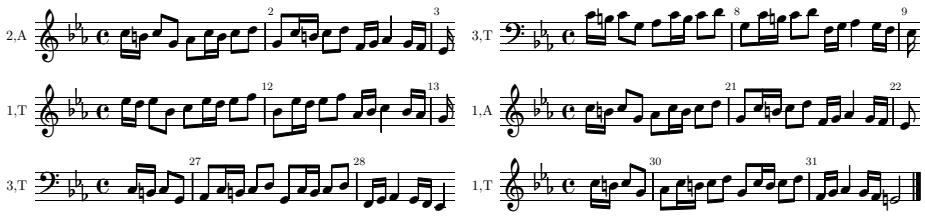


Fig. 4. Dux, iteration 1

A harmonic variant of the theme was found in iteration 13. It differs in one note (or two jumps) from the dux, and is traditionally called *comes*. It was found as an original pattern, since we did not allow a single note difference in the search. It has two occurrences which were correctly found as transpositions of each other – they occur in different octaves (Fig. 5).

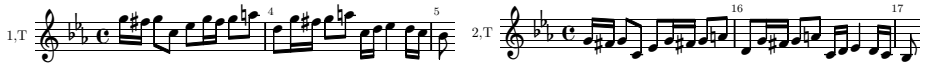


Fig. 5. Comes, iteration 13

One more important pattern in a fugue is the *counterpoint*. It is an accompanying figure to both dux and comes. It has six occurrences, but not all were found in their entirety. The pattern found in iteration 3 is a subpart of this figure and was found in all 6 places. In iteration 15, four counterpoints were found – extending the 3 pattern (Fig. 6). It could be argued that the note preceding this pattern musically belongs to the counterpoint figure. All occurrences are transpositions of each other, starting on $e\flat$, g, b and $e\flat$ again. One was found with the measure M. This is because of the difference in jump between the 7th and the 8th note. In three of the cases this is an octave and a third, but in bar 15 it is only a third.

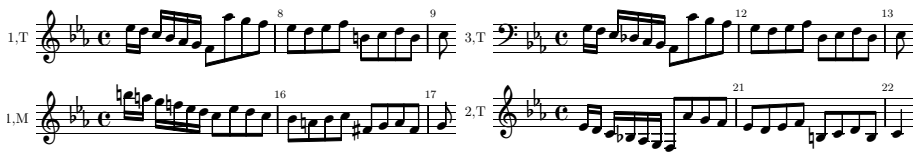


Fig. 6. Counterpoint, iteration 15

The simultaneous occurrences of dux/comes and counterpoint can be seen from Fig. 3. Also worth noticing is bars 17-19 where the patterns 8 and 4 occur simultaneously twice in succession in the two lowest voices – switching place. Furthermore the pattern labelled '0' occurs four times – each time accompanying itself. This happens in bars 9-11 and 22-24 (voices 1 and 2).

The presented segmentation is quite representative for the behavior of the algorithm. Running 10 segmentations with the given parameters, 7 of them found all occurrences of dux in the same iteration. Every time the comes was found. The counterpoint was in each segmentation recognised in 3-6 occurrences of varying extension.

6 Conclusion

The best we can do so far to evaluate a segmentation is to make the similarities visible/audible and compare to a manual analysis of the piece. The previous section showed that the algorithm did a fairly good job in finding the common motifs and derivations as well as categorising them correctly. It was not perfect, but taking into account that it depends on a nondeterministic algorithm which can been tuned, we can hope that even better results might be possible.

The overall segmentation certainly shows some structural dependencies in the fugue. A mechanical way of discovering when patterns occur simultaneously would be possible when allowing motifs to be non-sequential and thus spanning over notes from more parts. The data structure and search mechanism support this, but we will need some more effective and efficient similarity measures to be able to do this in practice.

The strength of the EA is it's ability to select the similar motifs in the music of any length. The search for occurrences might however have been more efficient with a deterministic algorithm in this simple sequential graph.

A crucial factor in the segmentation process seems to be the bounding of patterns. It is hard to evaluate the effect of the grouping structure rules. We did not give it much importance in this experiment. More focused experiments will be needed to study the relationship between musical grouping structure and meaningful musical patterns.

Also important is the choice of similarity measures. A larger set of transformation functions would be required to analyse other types of music. For example the MIDI pitch views might be more relevant when segmenting non-diatonic music. The rhythmical aspect should be explored. Fortunately it is easy to integrate new similarity measures into the evaluation function.

The idea of using the MuseData scores is to take advantage of the diatonic information. The different nature of the diatonic pitch viewpoints allows for some variation within the same the view. In our segmentation we only allowed phrases that had exactly equal views to be similar. A natural extension would be to allow for example one or more notes to differ while searching for 'new' themes. One could expect motifs to sound similar even when some notes are different (as for example the dux and comes). Allowing this introduces some uncertainties. We cannot be sure that what we find in every case also will be perceived as similar – especially when the note sequences are short. This approach would produce less 'correct' segmentations, but is an unavoidable next step that we have to examine.

Acknowledgements. This research was supported by the Austrian FWF (START Project Y99) and the Viennese Science and Technology Fund (WWTF, project CI010). The Austrian Resarch Institute for Artificial Intelligence acknowledges basic financial support from the Austrian Federal Ministries of Education, Science and Culture and of Transport, Innovation and Technology.

References

1. Ockelford, A.: 4: The Role of Repetitions in Percieved Musical Structures. In: Representing Musical Structure, Eds.: Peter Howell, Robert West and Ian Cross. Academic Press (1991) 129–160
2. West, R., Howell, P., Cross, I.: 1: Musical Structure and Knowledge Representation. In: Representing Musical Structure, Eds.: Peter Howell, Robert West and Ian Cross. Academic Press (1991) 1–30
3. Cambouropoulos, E., Widmer, G.: Automatic motivic analysis via melodic clustering. Journal of New Music Research **29(4)** (2000) 303–317
4. Grachten, M., Arcos, J.L., de Mantaras, R.L.: Melodic similarity: Looking for a good abstraction level. In: Proceedings of 5th International Conference on Music Information Retrieval (ISMIR'04), Barcelona, Spain (2004)
5. Rolland, P.Y.: Discovering patterns in musical sequences. Journal of New Music Research **28(4)** (1999) 334–350
6. Grilo, C., Cardoso, A.: Musical pattern extraction using genetic algorithms. In: Proceedings of the First International Symposium on Music Modelling and Retrieval (CMMR'03). (2003)
7. Conklin, D., Witten, I.: Multiple viewpoint systems for music prediction. Journal of New Music Research **24** (1995) 51–73
8. Conklin, D.: Representation and discovery of vertical patterns in music. In Anagnostopoulou, Ferrand, S., ed.: Proceedings of Second International Conference on Music and Artificial Intelligence (ICMAI'02), Edinburgh, Scotland, Springer (2002)
9. Madsen, S.T., Jørgensen, M.E.: Automatic discovery of parallelism and hierarchy in music. Master's thesis, University of Aarhus, Århus, Denmark (2003)
10. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. MIT Press (1983)
11. Smaill, A., Wiggins, G., Harris, M.: Hierarchical music representation for analysis and composition. Computers and the Humanities **27** (1993) 7–17
12. Lubiw, A., Tanur, L.: Pattern matching in polyphonic music as a weighted geometric translation problem. In: Proceedings of 5th International Conference on Music Information Retrieval (ISMIR'04), Barcelona, Spain (2004)