A SIMPLE-CYCLES WEIGHTED KERNEL BASED ON HARMONY STRUCTURE FOR SIMILARITY RETRIEVAL

Silvia García-Díez and Marco Saerens
Université catholique de Louvain
{silvia.garciadiez,marco.saerens}@ucLouvain.be

Mathieu Senelle and François Fouss
Université catholique de Louvain – Site de Mons
{mathieu.senelle,francois.fouss}@fucam.ac.be

ABSTRACT

This paper introduces a novel methodology for music similarity retrieval based on chord progressions. From each chord progression, a directed labeled graph containing the interval transitions is extracted. This graph will be used as input for a graph comparison method based on simple cycles – cycles where the only repeated nodes are the first and the last one. In music, simple cycles represent the repetitive sub-structures of, e.g., modern pop/rock music. By means of a kernel function [10] whose feature space is spanned by these simple cycles, we obtain a kernel matrix (similarity matrix) which can then be used in music similarity retrieval tasks. The resulting algorithm has a time complexity of $O(n+m(c+1))$, where n is the number of vertices, m is the number of edges, and c is the number of simple cycles. The performance of our method is tested on both an idiom retrieval task, and a cover song retrieval task. Empirical results show the improved accuracy of our method in comparison with other string-matching, and graph-comparison methods used as baseline.

1. INTRODUCTION

Since the beginning of the 15th century, motivic elements have made part of Western music, becoming common practice during the 18th century. We can find numerous examples of this phenomenon nowadays in modern pop/rock music which contain repetitive sub-structures, e.g., the chorus, verse, etc. According to [5], such repetitive structures, or *motifs*, act as *cues* in music perception. "A *cue* is a very restricted entity ... often shorter than the group itself, but always embodying striking attributes". This notion of cue, would let a listener encode information in a more efficient way, allowing longer structures to be memorized by means of smaller, more salient, features.

Although motifs can be found in a song's harmony or melody, in this paper we will focus on harmonic motifs for three reasons: (i) many songs share a part of their harmonic structure, as the number of chord progressions that are popular in a musical style (*idioms*) remain limited, while the melodic structure can vary greatly from one song to another; (ii) studies in experimental psychology have shown the essential role of harmony in musical sequence perception [6]; (iii) although the amount of chord progression data is increasing thanks to chord estimation algorithms (see e.g. [13]) and user-generated data (which is readily available from the web), few efforts have been put on harmony-based similarity measures.

On the other hand, human listeners, due to their musical background, are more susceptible to like songs with a familiar harmonic structure, but yet different enough from the songs they already know [14] [1] . We believe, thus, that comparing songs thanks to their harmonic motifs would yield in a similarity measure that takes into account its repetitive harmonic sub-structures.

One efficient way for motif extraction is the use of graphs. Motif extraction on graphs has attracted a lot of attention in the past years, e.g. in community detection [1], or in graph comparison [10]. A *motif* is formally defined in [1] as a connected undirected sub-graph (or weakly connected directed sub-graph) which appears frequently in a graph showing some kind of structure. Examples of motifs are cliques, paths, cycles, or sub-trees. The method presented in this paper relies on the concept of cycle as a motif for similarity detection between graphs (isomorphism). By transforming the chord sequences into graphs, and comparing their simple cycles, we obtain a similarity measure based on the musical motifs of a song (see Section 4.1 for a more precise description). The contributions of our work are as follows:

1. It is based on the **repetitive harmonic features** of songs (which can be easily extracted from web resources, as done in the present work).

2. The similarity measure deals with **large structural**

[1] This is explained by [12] as 'the compromise between the repetition and the surprise" in the expectation of a human listener.

changes in chord progression (e.g., addition of repetitions, bridge, etc.).

3. The similarity matrix can be extracted by means of **kernel functions**.

4. The similarity is **transposition invariant** (the intervals between chords are used, instead of the chords themselves).

5. We provide a simple, general, **methodology for computing similarities** from chord progressions (from the text mining step to acquire the data, to the automatic classification step with an SVM).

6. We exploit a **novel source of user-generated data** that is readily available on the Internet (in form of guitar chord progressions).

7. Empirical tests show that **music similarity retrieval** can be performed solely on the basis of chords.

We will briefly review the related work about chord sequence similarity in Section 2. Section 3 introduces the cyclic pattern kernels, on which our method is based. The details of our algorithm can be found in Section 4, as well as the graph extraction technique. Empirical testing is presented on two music retrieval tasks in Section 5, and eventually, Section 6 presents our conclusions.

2. RELATED WORK

Harmonic similarity has recently attracted the attention of the MIR (Music Information Retrieval) community thanks to the improvement in chord estimation techniques [13], as well as the increase of the available data. One of the advantages of harmonic similarity is its ability to infer similar songs whose melodies differ. In this context, [4] proposes an approach based on the Tonal Pitch Space (TPS) which compares the change of chordal distance to the tonic over time. This local distance is then used to build a step function that computes the global distance between two chord progressions by minimizing the non-overlapping area of the two step functions. However, this method requires information about the key of the piece and does not support structural changes (e.g., introduction of repetitions).

We can also find techniques based on approximate string matching, such as the one proposed by [9]. This technique extracts the most similar regions of the two chord sequences, and computes a distance based on the number of simple operations (insertion, deletion, substitution) that are needed to transform the first region into the second. This algorithm has complexity $O(nm)$ where n and m are the length of the sequences to compare, and edition costs must be provided.

Generative models are the third type of harmony similarity techniques. Such models assume that harmony variations occur according to an underlying model. The authors of [15] propose to model chord transitions of a song by means of a nth-order Markovian model, which serves as basis for a Kullback-Leibler scoring function. A generative model based on linguistics has also been applied in [3]. This harmony similarity method is based on the assumption of a generative grammar of tonal harmony. By applying a weighted version of this grammar, a unique parse tree representing the chord sequence is obtained for each song – note that context free grammars produce multiple ambiguous parse trees, thus a weighting of the rules is needed to choose among all possibilities. In order to measure the similarity of a pair of parse trees, the largest embeddable tree is extracted. However, its time complexity is $O(min(n,m)nm)$ and this technique may reject a sequence which is considered as ungrammatical.

3. CYCLIC PATTERNS KERNEL

Cyclic patterns represent harmonic motifs in chord progressions. In order to extract these motifs for music similarity, we will rely upon the cyclic pattern kernels from [10]. In this section we will present the key concepts of this technique which computes a kernel based on the set of cyclic and tree patterns of a graph.

3.1 Graphs and cycles

Let us first give some definitions concerning graphs and cycles. Let $G = (V, E, label)$ be a directed graph defined as a finite set of vertices V, edges $E \subseteq [V]^2$, and their labels. The cardinalities of V and E are n and m, respectively. We define a *simple cycle* on G as a sequence

$$C = \{v_0, v_1\}, \{v_1, v_2\}, ..., \{v_{k-1}, v_k\} \quad (1)$$

where $v_0 = v_k$ and all others $v_i \neq v_j$ for every i, j $(1 \leq i \leq j \leq k)$. Although a cycle may have several permutations, only one of them (and the same in all cases) will be kept for our purposes. We can now define the set $\mathcal{S}(G)$ as the set of simple cycles of G, the set of unrestricted cyclic patterns as $\mathcal{C}(G)$, and its relation:

$$\mathcal{S}(G) \subset \mathcal{C}(G) \quad (2)$$

Similarly, we can define the set of *tree patterns*, $\mathcal{T}(G)$, as:

$$\mathcal{T}(G) = \{T \text{ is a connected component of } \mathcal{B}(G)\} \quad (3)$$

where $\mathcal{B}$ is the set of bridges of G (see [10] for more details).

3.2 Kernel methods

The method presented in this paper belongs to the family of kernel methods [7, 17], a well-founded technique in statistical learning which comprises three steps:

1. A *mapping ϕ of the data* from the input space, x, (directed labeled graphs $\mathcal{G}$, in our case) into some meaningful, application-dependent, feature space, $\mathcal{F}$, (simple cycles):

$$\phi : x \rightarrow \phi(x) \in \mathcal{F} \quad (4)$$

2. An *inner product* defined in the feature space, $\phi(x)$, in order to obtain the kernel matrix (a positive definite matrix of similarities):

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \quad (5)$$

3. A *learning algorithm* for discovering patterns in that space (in our case, an RBF SVM for the automatic classification based on the similarity matrix).

One interesting property of kernel functions is that, although the feature space may have infinite dimension (the number of possible cycles, in our case), it is often possible to compute them in polynomial time. The obtained kernel matrix can then be used as a similarity matrix for music retrieval tasks.

3.3 Cyclic patterns kernel function

A cyclic patterns kernel function is proposed by [10], which takes two graphs as input, extracts their cyclic $\mathcal{C}(G)$ and tree patterns $\mathcal{T}(G)$ and uses them to build a mapping $\Phi_{\text{CP}}(G)$ into the feature space:

$$\Phi_{\text{CP}}(G) = \mathcal{C}(G) \cup \mathcal{T}(G) \quad (6)$$

The *cyclic pattern kernel* is defined as the set of all simple cycles and tree patterns that appear in both graphs:

$$k_{\text{CP}}(G_i, G_j) = |\mathcal{C}(G_i) \cap \mathcal{C}(G_j)| + |\mathcal{T}(G_i) \cap \mathcal{T}(G_j)| \quad (7)$$

However, the problem of computing cyclic pattern kernels is NP-hard. For overcoming this issue, the authors in [10], restrict the set of cyclic patterns to $\mathcal{S}(G)$, so that only *simple cycles* are computed (those cycles whose only repeated nodes are the first and the last one). The advantage of simple cycles is that they can be computed in polynomial time. The authors use the algorithm from [16], which extracts the simple cycles of a graph by means of a depth-first search in time $O(n+m(c+1))$, where n is the number of vertices, m is the number of edges, and c is the number of simple cycles. It is important, thus, that there exists a bound (*well-behaved* data) on the number of simple cycles for the sake of efficiency of the algorithm. As empirically shown in Section 5 (see Figure 2), this is the case for our chord data.

4. PROPOSED SIMPLE-CYCLE WEIGHTED KERNEL

In this section we present the proposed kernel, which is a variant of the cyclic pattern kernels [10] introduced in the previous section. We propose to focus our kernel only on simple cycles which will represent the repetitive harmonic sub-structure of a song. In order to favor longer simple cycles, a weighted (normalized) version of the kernel will be computed.

4.1 Graph extraction

Chord sequences represent the harmonic progression of a song which may modulate over time, i.e., its key changes through time. This is an important issue for the detection of harmonic similarities, as the transposed chords may not coincide. In order to make our method transposition-invariant we will thus convert the chord sequence into interval sequences, from which input graphs will be extracted. As only structure matters for us, and not the "musical distance" between a pair of chords (in semitones), a label λ_i will be assigned to each chord transition with the same "musical distance" (key invariant) [2] [3]. For example, the transition $C \rightarrow D\#m$ will share the same label as $F \rightarrow G\#m$ and its enharmonic $C \rightarrow E\flat m$, i.e.,

$$(C, D\#m) = (C, E\flat m) = (F, G\#m) = \lambda_k \quad (8)$$

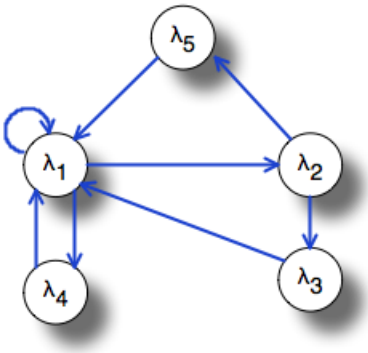
Chords	C,G,Am,F,C,G,F,C,G,Am,C,G,F,C,G,Am,...
Labels	$(C,G) = (F,C) = \lambda_1$, $(G, Am) = \lambda_2$ $(Am, F) = \lambda_3$, $(G, F) = \lambda_4$, $(Am, C) = \lambda_5$
Intervals	$\lambda_1, \lambda_2, \lambda_3, \lambda_1, \lambda_1, \lambda_4, \lambda_1, \lambda_2, \lambda_5, \lambda_1, \lambda_4, \lambda_1, ...$
Graph	

Table 1. Transformation of an extract of the chord progression of "Let it be" from The Beatles into an interval graph.

By sequentially reading the obtained interval sequence $x = \{\lambda_1, \lambda_2, ..., \lambda_1, ..., \lambda_l\}$, we will extract a directed graph G (see Table 1) where each node represents a chord transition or interval ($n = |\{\lambda_i\}|$), and each interval transition is represented by an edge ($m = |\{\lambda_i \rightarrow \lambda_j\}|$).

[2] For the sake of consistency we have not made the distinction between ascending or descending intervals.

[3] Please note that the chord type (minor, major, diminished, etc.) is already incorporated in the graph representation through the λ values, e.g., $(Cdim, Am) = (Edim, C\#m) = \lambda_k$.

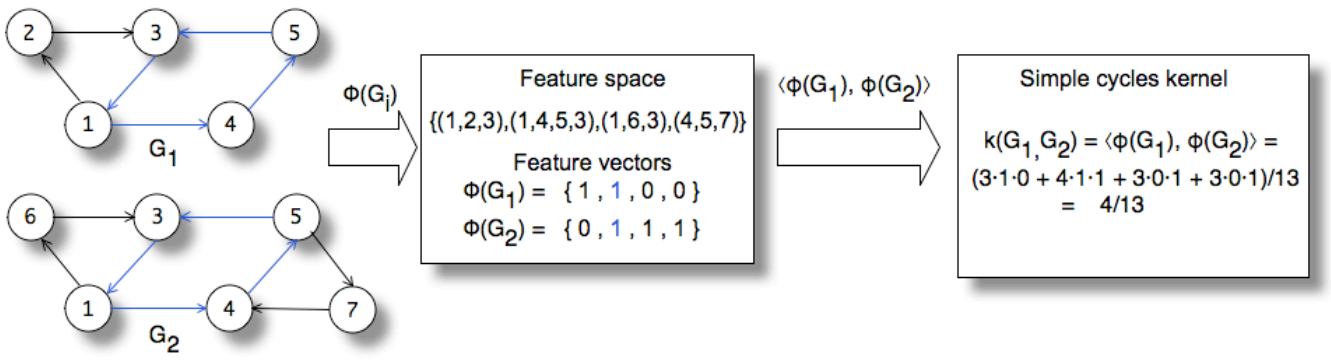


Figure 1. Computation of the simple-cycles weighted kernel on two initial graphs, G_1 and G_2.

4.2 Kernel function

Based on the algorithm from [10], we build a kernel which takes any two interval graphs from the input space, extracts their simple cycles to build a feature space, and computes a similarity as the weighted inner product in the feature space. In our case, the mapping function Φ is defined as a mapping to the set of all possible simple cycles of the graph

$$\boxed{G \rightarrow \Phi_{\text{SC}}(G) = \mathcal{S}(G)} \tag{9}$$

which represent the repetitive sub-structures of an interval graph. For a particular graph G_j, its feature vector has entries $[\phi(G_j)]_i$ which are equal to 1 if the simple cycle with index i (denoted as cycle i in the sequel) is present in the graph and 0 otherwise. We then compute the kernel function as the weighted inner product between the feature vectors (simple cycles vector)

$$\boxed{k(x,y) = \langle \phi(x), \phi(y) \rangle_{\tilde{W}} = \phi(x)^T D \phi(y)} \tag{10}$$

where D is the normalized diagonal weight matrix

$$[D]_{ii} = d_{ii} = \frac{w_i}{\sum_{j \in \mathcal{S}(G_k) \cup \mathcal{S}(G_l)} w_j} \tag{11}$$

and w_i is the length of the i-th cycle. The motivation for this weighting is to favor longer cycles, so that two graphs sharing a long cycle are considered as more similar as two graphs sharing one short cycle. Furthermore, the kernel weights are normalized by dividing them by their sum. The complete procedure is described in Algorithm 1 and an example on how to compute the weighted kernel is given in Figure 1.

5. EMPIRICAL TESTING

To evaluate empirically the retrieval performance of our kernel, two different tasks will be evaluated: (i) a cover song retrieval task, and (ii) an idiom retrieval task. We will first present the data used in the experiments, as well as the chosen lexicon. Our simple-cycles weighted kernel method is compared to several measures from string matching, as well as graph comparison techniques.

Algorithm 1 Simple-cycles Weighted Kernel: computation of the kernel matrix.

Input:

- $maxL > 0$: maximum length of extracted simple cycles.
- $s_1, ..., s_r$: list of chord sequences to be compared.

Output:

- K: the Simple-cycles Weighted Kernel matrix.

1. **for** k,l = 1 to r **do**
2. Transform chord sequences s_k and s_l into directed labeled graphs G_k and G_l following the procedure from Table 1.
3. Extract all simple cycles of length $< maxL$, $\mathcal{S}(G_k)$ and $\mathcal{S}(G_l)$, from G_k and G_l, with the algorithm described in [16].
4. Create the feature vectors, $\phi(G_k)$ and $\phi(G_l)$, of length $|\mathcal{S}(G_k) \cup \mathcal{S}(G_l)|$, whose entry $[\phi(G_k)]_i = 1$ if the i-th cycle is in $\mathcal{S}(G_k)$ and 0 otherwise.
5. For all the cycles of $\mathcal{S}(G_k) \cup \mathcal{S}(G_l)$, compute the corresponding elements i of the diagonal matrix D from Equation (11).
6. Compute $[K]_{kl} = \phi(G_k)^T D \phi(G_l)$.
7. **end for**

5.1 The chord data sets

The *cover song data set* has been extracted from two different sources: the Beatles chord annotations from the Isophonics [4] data base (Queen Mary, University of London), and the user-generated chord files from the Ultimate-Guitar [5] data base. Although our first source of chord progressions has already been used in MIR, we are the first to use, to the best of our knowledge, a popular Internet guitar's chord

[4] isophonics.net
[5] www.ultimate-guitar.com

data base for similarity retrieval. The Ultimate-Guitar data base contains more than 250,000 user-generated sequences of guitar's chords of popular pop/rock music. Although several versions are available for each of the Beatles' songs, only well-ranked songs have been extracted (5 star rated songs with at least 5 votes), making a total of 71 songs. These same songs have been extracted afterwards from the Isophonics data base, forming 71 classes of two songs each (142 songs in total), where the songs from the Isophonics data base are used as query over the remaining 71 songs from the Ultimate-Guitar data base (one relevant song per query). Although there exists a well-known MIREX audio cover song task, this evaluation task takes audio signals as input while our work is centered on chords, so that it cannot be applied here.

The *idiom data set* has been fully extracted from the Ultimate-Guitar data base and contains 296 songs partitioned in two classes (101 songs for the first class, sharing a common 4-chords idiom [6], and 195 songs for the second class). Both data sets are available from `www.isys.ucl.ac.be/staff/silvia/research.htm`.

In both cases, a modest lexicon containing all major and minor root chords (flat and sharp) has been used. We believe that this choice is representative enough for our purpose, while avoiding bad transcription issues from users in the Ultimate-Guitar data base, e.g., the chord $C5$ appears instead of C.

5.2 Cover song retrieval task

Cover song retrieval (see for instance [2]) is a popular task in MIR which aims at identifying the versions of a given song. For this purpose, the cover song data set described above has been used. We query the Ultimate-Guitar database with each song from the Isophonics chord annotation (the "query song"), providing a ranking of the Ultimate-Guitar songs in decreasing order of similarity with the Isophonics query song (please see Section 5.1 for more details). The *average ranking* position of all retrieved songs, as well as two recall measures describing the accuracy of our method have been reported in Table 2: the *average first tier* (the number of correctly retrieved songs among the best $(n_c - 1)$ matches divided by $(n_c - 1)$ with n_c the class size, i.e., in our case $n_c = 2$), and the *average second tier* (number of correctly retrieved songs among the best $(2n_c - 1)$ matches divided by $(n_c - 1)$).

In order to compare our method to other base line methods, the same methodology [7] has been applied to three string matching techniques – the edit distance and longest common subsequence widely used in sequence matching (see, e.g., [8]) and the all-subsequences kernel [17] which is an efficient method that compares all sub-sequences of two strings –, and a graph comparison kernel – the fast sub-tree kernel, a similarity measure between graphs that is fast to compute and that outperforms other graph kernels [18]. For methods needing a parameter, the fast sub-tree kernel and the simple-cycles kernel, we have chosen a maximum cycle length (tree depth) of 7 – longer cycles or deeper trees become too song-specific, and are not of interest for us. Although chosen base line methods may appear simplistic, our aim is to compare our algorithm with a variety of methods under the same conditions. Purpose-built methods using different chord representations, or needing parameter tuning are not compared in the present article for obvious reasons of adaptation, leaving this task for further work.

Although results show no improvement for the first tier, and just a slight improvement of the second tier (see average first and second tier in Table 2) from the base line methods, there is a clear improvement in the average general ranking of retrieved songs. These results are encouraging for using the Ultimate-Guitar data base as a future source for chord progression data.

5.3 Idiom retrieval task

Idioms have recently attracted the attention of MIR as a new object of musicological interest. An *idiom* is defined in [11] as a "prominent chord sequence in a particular style, genre or historical period". Users have also discovered this notion of idiom as shown in a youtube video [8], where a sequence of 4 chords is used to assemble the melody of several pop/rock songs. Interestingly, people who liked a few of these songs tended to also appreciate the others.

We have tried to recover the songs containing the idiom "C,G,Am,F" (or "I-V-VI-IV") by applying a 10-fold double cross validation with an RBF SVM on the idiom data set from the Ultimate-guitar web site. Classification rates with a 95% confidence interval are reported in Table 3. These results show an increase of performance of our method of 7% from the closest base-line method.

Similarity	First tier average	Second tier average	Average ranking
Edit distance	78.87% ± 6.76	87.32% ± 5.51	4.169 ± 1.64
Longest common subs.	60.56% ± 8.10	69.01% ± 7.66	8.662 ± 2.59
All-subsequence kernel	28.17% ± 7.45	43.66% ± 8.22	15.929 ± 3.32
Fast sub-tree kernel	52.11% ± 8.27	61.97% ± 8.04	11.943 ± 2.78
Simple-cycles kernel	78.87% ± 6.76	88.73% ± 5.24	2.915 ± 1.09

Table 2. Average first tier, second tier, and average ranking for the cover retrieval task with 95% confidence intervals.

[6] The sequence "C,G,Am,F" is considered as an idiom in modern pop/rock composition. It appears in songs such as *Let it be* (The Beatles), and *With or without you* (U2).

[7] Interval sequences have been provided as input for each baseline method, so that all compared methods are transposition invariant and evaluated under similar conditions.

[8] `http://www.youtube.com/watch?v=qHBVnMf2t7w`

Similarity	Classification rate and confidence interval
Edit distance	68.56% ± 1.53
Longest common subsequence	69.91% ± 2.41
All-subsequence kernel	68.56% ± 1.53
Fast sub-tree kernel	81.06% ± 4.22
Simple-cycles kernel	88.50% ± 2.02

Table 3. Classification rates with a 95% confidence interval for the idiom retrieval task.

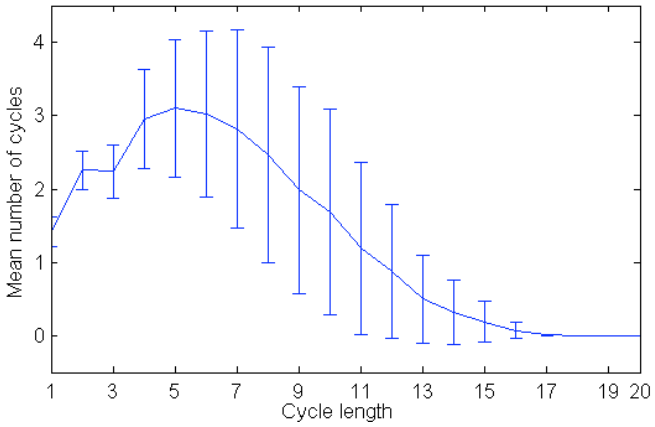


Figure 2. Error bar showing the average number of simple cycles per song and per cycle length of our chord progressions data. 95% confidence intervals are also shown.

6. CONCLUSION AND FUTURE WORK

In this paper we have introduced a simple-cycle similarity method based on the harmonic progression of a song. We have presented the notions of a theoretically well-founded method, and shown its applicability to our problem. This approach has furthermore been validated on an idiom and a cover song retrieval task. The obtained results suggest the utility of extracting repetitive sub-structures for music similarity purposes by means of a simple-cycles weighted kernel. Further work will try to improve the presented algorithm by performing an approximate cycle matching, and by replacing labels by musical distances between chords.

7. ACKNOWLEDGMENTS

This work is partially supported by the *Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture* (F.R.I.A.) under grant reference F3/5/5-MCF/ROI/BC-21716. Part of this work has also been funded by projects with the "Région Wallonne" and the Belgian "Politique Scientifique Fédérale". We thank these institutions for giving us the opportunity to conduct both fundamental and applied research. We also thank Juan Felipe Avila from the Universidad Nacional de Colombia for his clarifications regarding musical concepts.

8. REFERENCES

[1] A. Arenas, A. Fernandez, S. Fortunato, and S. Gomez. Motif-based communities in complex networks. *Journal of Physics A: Mathematical and Theoretical*, 41:224001, 2008.

[2] J. P. Bello. Audio-based cover song retrieval using approximate chord sequences: Testing shifts, gaps, swaps and beats. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR)*, pages 239–244, 2007.

[3] W. Bas de Haas, M. Rohrmeier, R. C. Veltkamp, and F. Wiering. Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 549–554, 2009.

[4] W. Bas de Haas, R. C. Veltkamp, and F. Wiering. Tonal pitch step distance: a similarity measure for chord progressions. In *Proceedings of the 9th International Society for Music Information Retrieval Conference (ISMIR)*, pages 51–56, 2008.

[5] I. Deliège, M. Mélen, D. Stammers, and I. Cross. Musical schemata in real time listening to a piece of music. *Music Perception*, 14(2):117–160, 1996.

[6] C. Drake. Psychological processes involved in the temporal organization of complex auditory sequences: Universal and acquired processes. *Music Perception*, 16(1):11–26, 1998.

[7] T. Gärtner. *Kernels For Structured Data*. World Scientific Publishing, 2009.

[8] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

[9] Pierre Hanna, Matthias Robine, and Thomas Rocher. An alignment based system for chord sequence retrieval. In *Proceedings of the IEEE/ACM International Joint Conference on Digital Libraries (JCDL)*, pages 101–104, 2009.

[10] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of Knowledge Discovery and Data mining (KDD)*, pages 158–167, 2004.

[11] M. Mauch, S. Dixon, C. Harte, M. Casey, and B. Fields. Discovering chord idioms through beatles and real book songs. In *Proceedings of the 8th International Society for Music Information Retrieval Conference (ISMIR)*, pages 255–258, 2007.

[12] F. Pachet. Surprising harmonies. In D. M. Dubois, editor, *Proceedings of the 2nd International Conference on Computing Anticipatory Systems*, 1998.

[13] H. Papadopoulos and G. Peeters. Joint estimation of chords and downbeats from an audio signal. *IEEE Transactions on Audio, Speech & Language Processing*, 19(1):138–152, 2011.

[14] J. Paulus, M. Müller, and A. Klapuri. Audio-based music structure analysis. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 625–636, 2010.

[15] J. Pickens and T. Crawford. Harmonic models for polyphonic music retrieval. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 430–437, 2002.

[16] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[17] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[18] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1660–1668, 2009.