Efficient Repeating Segments Discovery in Music using Adaptive Motif Generation Algorithm

Lei Wang*, Eng Siong Chng* and Haizhou Li*†
* School of Computer Engineering, Nanyang Technological University, Singapore 639798
E-mail: {wang0161, aseschng}@ntu.edu.sg Tel: +65-6790 4964
† Institute for Infocomm Research, 1 Fusionopolis Way, Singapore 138632
E-mail: hli@i2r.a-star.edu.sg Tel: +65-6408 2773

***Abstract*—This paper introduces an efficient unsupervised algorithm to discover motifs in multivariate data sequence. Specifically, we apply our proposed work to detect repeating segments on music feature vectors. The proposed algorithm, namely Adaptive Motif Generation, scans the music features online to construct a list of repeating candidate segments in linear time. The candidate list is then used to populate a sparse self-similarity matrix for further processing to generate the final selections. The experimental results showed that the proposed approach was able to obtain similar average F1 score compared to the traditional self-similarity approach with significant reduction in computational cost and memory usage.**

I. Introduction

Automatic motif discovery is an important research focus in many diverse research areas. Examples include biological sequence analysis [1], [2], video structure analysis [3], sub-words/words discovery in speech [4], [5], repeating segments discovery in music [6], [7], [8], [9], [10], [11], [12], and the analysis of many other time series data [13], [14], [15], [16], [17].

One popular application of motif discovery is to detect repeating segments in music and it has received much attention during the past decades. The repeating segments in music are defined as segments that have similar melody. To automatically detect music repeating segments, many unsupervised techniques were proposed. For example, a break-down RP-Tree [18] and a recursively scanning method [19] were examined to discover patterns in MIDI sequence; In addition, self-similarity matrix [6], [7], [8], [9], [10], [11], beats detection [20] and dynamic time warping [21] have been explored to detect repetitions in polyphonic music.

To analyze music data, one effective method is to construct a self-similarity matrix [6] followed by searching for the distinguishable stripes along the diagonals [7], [8], [9], [10], [11]. A brief description of a self-similarity matrix can be found in the next session. The use of self-similarity matrix is popular as it allows robust techniques to be applied to detect repeating sequences. Specifically, algorithms to 'search' for the repetition are performed on the 2-D similarity matrix and hence can exploit advanced image processing techniques to recover noisy repetition sequences across neighboring regions. The use of traditional similarity matrix however requires $O(N^2)$ computational resources where N is the number of vectors in the sequence to be analyzed. Hence, the use of self-similarity matrix is limited to short sequences due to the high computational requirement. The analysis on longer music pieces remains a difficult problem.

In reality, the occurrence of true repeating patterns for an actual music piece in the self-similarity matrix is $\ll 1\%$. This motivates us to examine an approach that can efficiently and robustly generate the self-similarity matrix without the $O(N^2)$ computational cost. The efficiency factors we are concerned with are the computational complexity and memory requirements of the method. We propose the Adaptive Motif Generation (AMG) algorithm to efficiently construct a sparse similarity matrix using suffix tree construction algorithm. Our proposed method modifies the symbolic suffix tree algorithm to accept vectors as inputs so that global quantization of the input vectors to symbols is avoided. In our tree construction process, the insertion of a new vector not only compares its similarity to each node's template vector, we also measure the current sequence's similarity to each node's sequences. In addition, each node maintains an adaptive threshold to control the insertion process. With these criteria, our proposed method is more robust than global quantization. Repeating sub-sequences can then be extracted from the constructed tree to populate a sparse self-similarity matrix. As the number of repeating segments found in the tree structure is very small, the self-similarity matrix constructed is very sparse. Our experimental results showed that the occupancy of the similarity matrix is approximately 1%.

Using a corpus of 30 songs, we compare the recall and precision of repetitions obtained using the AMG versus a full self-similarity matrix. Our results showed that the use of AMG achieved similar performance with significantly reduced computation performance in terms of memory and computation.

The remainder of the paper is organized as follows: Section II briefly reviews related techniques for motif discovery in multivariate sequence, Section III proposes our approach to construct the sparse self-similarity matrix using AMG, Section IV introduces the refinement framework to post-process the candidate repeating segments, Section V reports the experimental results and finally, we conclude in Section VI.

II. Related Work

This section briefly discusses two categories of related techniques to discover motifs from multivariate sequence:

symbolization and direct search techniques.

A. Symbolization Techniques

This section describes the symbolization techniques for motif discovery. The common strategy of these techniques is to first convert the raw multivariate sequence into symbols or signatures and then applying symbolic mining techniques such as suffix tree, R*-tree [22], random projection [23], etc to discover motifs. Example of classical symbolization techniques include partitioning clustering [24], [25], [26] such as K-means and hierarchical clustering algorithms such as Clustering Feature-tree [27].

A closely related technique to symbolization is to create indices from the raw data [28], [29], [30] - Such approach was explored for fast similarity search in univariate time series database by extracting and matching signatures. For example, Agrawal *et al.* [28] transformed time series into frequency domain and retained the first few Fourier coefficients to index the whole sequence using R*-tree [22]. In an extension of [28], [29] employed a sliding window over time series and extracted the signatures so that sub-sequences can be matched using efficient indexing methods. Dimensionality reduction techniques such as Singular Value Decomposition [31] and Discrete Wavelet Transform [32] were also explored to extract signatures from time series for indexing.

The researchers [33] however claimed that the data clusters and indices extracted by the above approaches are essentially random. They instead propose the SAX [13] to transform the univariate time series data into low-complexity symbolic representation so that symbolic sequence search techniques such as Approximation Distance Map [34] and random projection [23] can be applied to find the motifs. The advantage of SAX is its robustness to measurement noise, and hence it had been used in different areas [35]. To apply SAX for multivariate time series, the multivariate data is first projected into one dimension using principle component analysis (PCA) and then SAX is applied. Motifs are then found using minimum description length [17]. As this approach only uses the first component of PCA, it is limited to analyze data that can be correctly represented by its first principal component [14].

B. Direct Search Techniques

This section describes the direct search techniques that bypass the symbolization process by searching directly on the multivariate time series data.

One example of such techniques is the self-similarity matrix [7], [6], [8], [9] method used to visualize and detect repeating segments in music. The music piece is first divided into short overlapping frames to generate feature vectors, and the pairwise similarity measure among these vectors are evaluated to construct a self-similarity matrix. For example, Foote [6] and Lu *et al.* [9] constructed an $N \times N$ vector-to-vector similarity matrix to visualize the similarities among the N features of a music segment as shown in Figure 1. The repeating musical patterns are then found using image processing techniques by extracting the stripes in the similarity matrix. Unfortunately, the self-similarity approach requires a complexity of $O(N^2)$ for memory and computational requirement. This high computational requirement limits the application of self-similarity approach to short sequences.

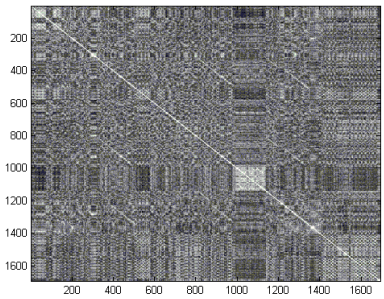

Fig. 1. A self-similarity matrix generated for the pop song "When You Say Nothing at All" by Ronan Keating using the temporal chroma features (in Section V).

In another approach, Oates [36] proposed PERUSE algorithm to detect frequently occurring patterns from a set of multivariate sensory data series. It adopts a sliding window to step over the entire data. Dynamic programming is used to identify the best occurrence of the candidate motifs. PERUSE allows the repeating patterns found to be of variable lengths but it assumes the motifs occur densely. The implementation for the dynamic programming process is exhaustive and requires high computation.

To circumvent the exhaustive search in [36], modeling techniques such as HMM are explored to detect motifs from general multivariate time series. Minnen *et al.* [4], [14] selected the candidate motifs by estimating density mode for each subsequence along with its k-nearest neighbors, and HMM was built for each candidate to fit the time series while the likelihood scores are used to rank the candidates. Another extension of HMM is the hierarchical HMM [3] used to discover recurrent patterns in video archives by modeling newly occurred events.

The above mentioned techniques [3], [4], [14], [36] can effectively detect motifs from multivariate data but they require high computational cost. This motivates us to explore efficient solutions for motif discovery in the next sections.

III. Sparse Self-similarity Matrix Construction

This section describes our proposed algorithm to efficiently construct the sparse self-similarity matrix.

The main idea of our approach is to construct a data representation structure that is similar to a suffix tree to locate the repeating subsequences. In a nutshell, we extend Ukkonen's online suffix tree construction algorithm [37] to analyze the input vector sequence without the symbolization process.

A suffix tree is a data structure that can store all the suffixes of a symbolic sequence [37], [38] into a tree structure. By examining the nodes of a suffix tree, all the repeating

substrings within the sequence can be obtained. Hence, the suffix tree can be exploited to generate candidate motifs directly.

The Ukkonen's online suffix tree construction technique is popular as the algorithm can build the suffix tree at linear computational cost. The algorithm scans the input symbolic sequence sequentially and inserts the new input symbol to a set of active suffix nodes. The active nodes are the nodes of the tree that had matched the current input symbol and is updated for each new input symbol. This implies that the active node's branch had matched the existing input subsequence. However, Ukkonen's method can only build suffix trees for symbolic sequences.

Our approach exploits the suffix tree structure [38] to detect repeating subsequences by constructing a list of truncated suffixes with equal length to locate the repeating subsequences of a feature vector sequence. To achieve an efficient construction process, we organize these suffixes using a tree structure as shown in Figure 2, i.e., only the root node can have multiple children.

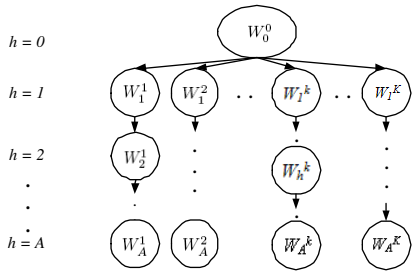

Fig. 2. The resulting candidate motif representation with Adaptive Motif Generation algorithm.

The Ukkonen's algorithm cannot be directly applied to scan sequences of vectors. This section discusses our novel extension to [37] to accept vectors as input. Specifically, we modified the insertion criteria to i) compare similarity measures between vectors with an adaptive threshold for robust insertion, ii) to consider similarity across current segment as opposed to only considering current input vector as part of the insertion criteria, iii) we restricted the tree's internal node to have only one child for practical realization. We name this new online tree construction algorithm as the Adaptive Motif Generation (AMG) in which "motif" means the repeating segments (or subsequences) of the music piece in this paper. The new algorithm retains the efficiency of [37] as it also constructs the tree structure in linear time.

A. Adaptive Motif Generation

Figure 2 shows the overall structure of a tree constructed by the proposed AMG algorithm. Each circle in the tree represents a node W_h^k where h denotes the depth and k the branch number of the tree. Let K denotes the total number of branches. To manage the tree's growth, the maximum length of each branch will be set to A where $A << N$. The value of A is database dependent, and should be large enough to capture segments of actual motifs. We will discuss the selection of A in the next session.

In AMG representation (Figure 2), each branch represents a candidate motif, and the tree structure can be interpreted as a list of K candidate motifs with length equals to A. The details of the implementation is as follows:

To capture the repeating segment information, each node W_h^k contains four types of information: the node's template vector $\mathbf{v}_h^k$, the time stamp f_h^k where $\mathbf{v}_h^k$ first occurred, a similarity threshold value α_h^k, and a list Ψ_h^k that contains the time stamps of all input vectors that have been inserted into the node.

The AMG algorithm is summarized in Table I. The following symbols are used: W_0^0 is the root node, $\mathbf{u}_i$ is the input vector at frame index i, N is the number of features in the sequence, γ_1 is a user defined threshold value, $set1$ is the set of active nodes and $set2$ is the set of modified nodes during each iteration of the algorithm. The node pool $set1$ is the set of candidate nodes that may accept $\mathbf{u}_i$ as its child during each iteration of the construction algorithm. The set of candidate nodes are generated by $f_{root}(W_0^0, \mathbf{u}_i)$ and $f_{branch}(W_h^k, \mathbf{u}_i)$. These two functions evaluate if the input vector $\mathbf{u}_i$ needs to be inserted into the tree at the root node and non-root node respectively. The insertion process to the tree is similar to [37]. The following paragraphs present the details of $f_{root}(W_0^0, \mathbf{u}_i)$ and $f_{branch}(W_h^k, \mathbf{u}_i)$:

1) Insertion at root node: $f_{root}(W_0^0, \mathbf{u}_i)$: The function $f_{root}(W_0^0, \mathbf{u}_i)$ evaluates the input vector $\mathbf{u}_i$ at the root node to determine if

- **H1**: $\mathbf{u}_i$ can be used to create a new branch in the tree,
- **H2**: $\mathbf{u}_i$ can be inserted into existing branch(es).

The function examines both hypothesis (**H1**, **H2**) and returns the set of nodes that satisfies either hypothesis (either newly created nodes or modified nodes).

H1: This condition evaluates if a new candidate motif branch will be created at depth 1. A new branch will be created if no existing branches' template vector is similar to the input vector $\mathbf{u}_i$. In other words, to create a new branch in the tree,

TABLE I
THE AMG ALGORITHM USED TO CREATE CANDIDATE MOTIFS.

Input: vector sequences $\mathbf{u}_i, i = 1, 2, \ldots, N$
Initialize: create first branch's node W_1^1 for input $\mathbf{u}_1$
1) $K = 1, i = 1, h = 1$
2) Create W_h^K with $\mathbf{v}_h^K \leftarrow \mathbf{u}_i$, $f_h^K \leftarrow i$, $\Psi_h^K \leftarrow \{i\}$, $\alpha_1^k = \gamma_1$.
3) $set1 \leftarrow \{W_h^K\}$ // Active branches
4) $set2 \leftarrow \{\}$
For $i = 2 : N$
  {modified nodes} $= f_{root}(W_0^0, \mathbf{u}_i)$
  $set2 =$ {modified nodes}
  For each node $W_h^k \in set1$
  - {modified nodes} $= f_{branch}(W_h^k, \mathbf{u}_i)$
  - $set2 \leftarrow set2 \cup$ {modified nodes}

  End
  $set1 \leftarrow set2$
  $set2 \leftarrow \{\}$
End

the criterion is that the input vector $\mathbf{u}_i$'s similarity to all current depth 1 nodes W_1^k's template vector $\mathbf{v}_1^k$ is less than a user defined threshold γ_2. If this condition is satisfied, a new node will be created and this node is returned by the function.

In our work, the Pearson correlation coefficient [39] is used to measure the similarity between two vectors. Specifically, the Pearson correlation between two vectors $\mathbf{u}$ and $\mathbf{v}$ is defined as

$$\rho(\mathbf{u},\mathbf{v}) = \frac{\sum_{r=1}^{d}(\mathbf{u}_r-\overline{u})(\mathbf{v}_r-\overline{v})}{\sqrt{\sum_{r=1}^{d}(\mathbf{u}_r-\overline{u})^2\sum_{r=1}^{d}(\mathbf{v}_r-\overline{v})^2}} \quad (1)$$

where u_r, v_r are the r^{th} element of $\mathbf{u}$ and $\mathbf{v}$ respectively, d is the dimensionality of the vectors, and $\overline{u}$, $\overline{v}$ are the mean of vector $\mathbf{u}$ and $\mathbf{v}$.

H2: To verify if the current input vector $\mathbf{u}_i$ can be inserted into the k^{th} branch of the tree, we evaluate β_1^k the similarity between $\mathbf{u}_i$ to the k^{th} branch first template vector,

$$\beta_1^k = \rho(\mathbf{v}_1^k, \mathbf{u}_i) \quad (2)$$

If β_1^k is greater than the node's threshold α_1^k, the corresponding node's Ψ_1^k is updated to include the time stamp i. The set of modified nodes will be returned by the function and those nodes will become the "active" nodes that must be verified for the next input vector.

2) Insertion at branch node: $f_{branch}(W_h^k, \mathbf{u}_i)$: Before we discuss the details of $f_{branch}(W_h^k, \mathbf{u}_i)$, we first discuss the nodes belonging to $set1$ that is used to evaluate $f_{branch}(W_h^k, \mathbf{u}_i)$. The nodes W_h^k in $set1$ are the "active" nodes, i.e., the sequence of template vectors at these active nodes' branch from depth of $1 \ldots h$ has matched the recent input vector sequence $\mathbf{u}_{i-h} \ldots \mathbf{u}_{i-1}$. For the current input vector $\mathbf{u}_i$, the function $f_{branch}(W_h^k, \mathbf{u}_i)$ verifies if this new input vector should be appended to the active branches.

The function $f_{branch}(W_h^k, \mathbf{u}_i)$ evaluates $\mathbf{u}_i$ for non-root node W_h^k, i.e. $h > 0$ to determine if

- **H3**: $\mathbf{u}_i$ can be used to create a new child node W_{h+1}^k,
- **H4**: $\mathbf{u}_i$ can be inserted into the existing child node W_{h+1}^k.

The function will examine either hypothesis (**H3** or **H4**) and return only one or no modified node.

H3: If W_h^k has no child node and $h < A$, a new child node W_{h+1}^k will be created and returned by the function. By growing the branch with a new child, we can view the operation as extending the current candidate motif by 1 vector.

H4: If W_h^k has a child node, we will verify if the current input vector $\mathbf{u}_i$ can be inserted into the existing k^{th} branch of the tree. Different to **H2**'s Eq. 2 which evaluates only the similarity between the current input vector to one node's template vector, we extend Eq. 2 to measure the similarity between the current input vector sequence to $W_{1\ldots(h+1)}^k$ motif. Specifically, we evaluate the segment similarity by

$$\beta_{h+1}^k = \frac{\sum_{r=0}^{min(h,\lambda-1)}\rho(\mathbf{v}_{h+1-r}^k, \mathbf{u}_{i-r})}{min(h,\lambda-1)} \quad (3)$$

where λ is the user defined segment length.

To verify if $\mathbf{u}_i$ can be inserted into the k^{th} branch of the tree, β_{h+1}^k is compared to threshold α_{h+1}^k. If β_{h+1}^k is greater than α_{h+1}^k, the input vector time stamp information i will be included by Ψ_{h+1}^k and the function $f_{branch}(W_h^k, \mathbf{u}_i)$ returns the node W_{h+1}^k, otherwise null.

In our system, each α_{h+1}^k is adaptive so that the sequence being analyzed can influence the threshold value. This makes the insertion process more robust as opposed to having a single fixed threshold. The α_h^k is updated by

$$\alpha_{h+1}^k = \alpha_h^k - (\beta_h^k - \beta_{h-1}^k), \quad (4)$$

where $h \geq 1$, and $\beta_0^k = \beta_1^k$. The adaptive threshold α_{h+1}^k has the following behaviors: α_{h+1}^k decreases when the past segment similarity β_h^k is better than its predecessor, and α_{h+1}^k increases when β_h^k is poorer than its predecessor. The adaptive threshold is modified thus so that sequences with intermittent poor segment similarity will be retained if past segment similarity has been good.

B. The AMG Parameter Settings

The motif discovery process of the AMG algorithm is controlled by the following four parameters, namely, A, λ, γ_1 and γ_2. The purposes of these parameters are discussed below:

1) The parameter A specifies the maximum depth of the tree and its value should be set to a length that can capture a meaningful portion of the motif. The value of A is however not critical as the refinement step of our proposed system has the ability to merge neighboring motifs.
2) The parameter λ specifies the length of the segment to evaluate the similarity measure as specified in Eq. 3. As in the selection of parameter A, the value of λ should be chosen long enough to capture a motifs sequence. A larger value of λ will smooth the segment similarity measure more significantly.
3) The threshold γ_1 is used to initialize α_1^k. As α_1^k is the first node's threshold at the k^{th} candidate motif, its setting is important since it determines if each new input vector will be inserted into the branch. An unreasonably high setting will stop a true repeating sequence from being inserted, while an unreasonably low setting will generate too many false alarms. Hence, the value of γ_1 is critical to the success of AMG. In our experiments, we select a small development corpus from the same test domain. We then evaluate all pairwise similarity between the vectors and select the top 10^{th} percentile pairwise similarity value as γ_1.
4) The threshold γ_2 is used in the evaluation of hypothesis **H1**. The value determines whether a new branch should be created for the input vector. The suitable value for γ_2 is $\gamma_1 < \gamma_2 \leq 1$. A higher γ_2 value will allow more branches to be created. In our experimental work, γ_2 is set to the top 5^{th} percentile pairwise similarity value of the development set (as discussed in Step (3)).

C. Discussion: AMG vs. Ukkonen's Suffix Tree Construction

The insertion criteria of Ukkonen's algorithm for symbolic suffix tree are based on exact match evaluation which compares the input symbol to the node's template. In contrast, the insertion criteria for AMG are not as straight forward since the input are vectors - this implies that the similarity comparison must use threshold for the decision process. However, a simple pairwise similarity evaluation of current input vector to the node's template vector cannot reflect the global correlation of two sequences and hence is not robust. To overcome this shortcoming, the AMG employs a segment-to-segment similarity measure with an adaptive threshold to evaluate if the current vector should be inserted into the tree. Hence, the insertion process can tolerate intermittent poor pairwise similarities measured with candidate motifs. Finally, the AMG is also different to the general suffix tree construction as we restrict each non-root node to have only one child. This restricts the tree's growth and allows for a practical realization. Our experimental results show that this realization did not cause serious degradation in repeating segments detection.

IV. Candidate Repeating Segments Refinement

In Figure 2, each branch of the AMG generated structure represents a candidate motif. These candidate motifs must be further processed to extract true repeating patterns because: (i) the candidate motifs will often contain redundant patterns such as trivial matches; (ii) the candidate motifs can be merged to form longer patterns, (iii) the AMG algorithm may miss detecting true motifs, and (iv) we need to align the boundaries of the found motifs.

To address the above four issues, a direct approach is to capture the candidate motifs' occurrences in a sparse $N \times N$ matrix where N is the number of vectors in the input sequence, and then refine the obtained patterns to generate the final selection. We call this matrix the candidate motif matrix, and the elements of this matrix whose values are '1' indicate the occurrence of a repeating segment. The interpretation of this matrix is similar to the binarized self-similarity matrix used for motif discovery [9]. Hence, existing techniques such as [8], [9] to refine the similarity matrix and pattern merging can be applied. In addition, missing motifs can also be immediately identified from this matrix (see Section IV-C).

Different from [8], [9], our proposed candidate motif matrix generated by the AMG is immediately a sparse $N \times N$ matrix as opposed to the conventional method to generate the full $N \times N$ self-similarity matrix. Our experimental results (Section V) show that the occupancy of the candidate motif matrix is only about 1% of the full matrix and the memory usage to construct the AMG structure is $O(AK)$.

Figure 3 illustrates our proposed refinement framework. The candidate motifs generated by AMG is first used to populate the sparse candidate matrix. This matrix is subsequently converted to a time-lag matrix similar to [8], [9] for the refinement process such as: pattern merging, pattern duplication and boundary refinement. The details of the procedure are described in the following sub-sections.

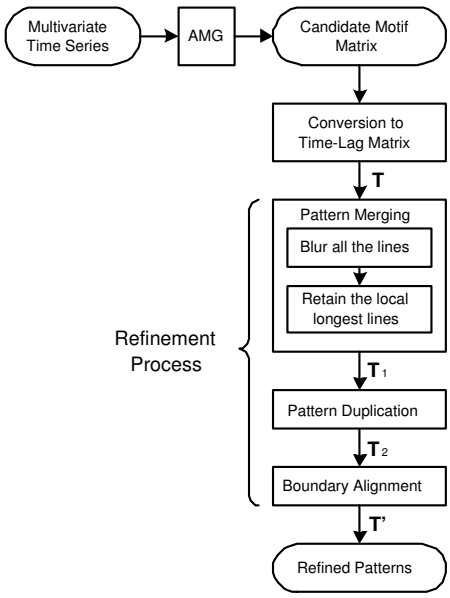

Fig. 3. The proposed process to refine the candidate motifs obtained by AMG algorithm.

A. Candidate Motif Matrix Generation

The candidate motifs can be easily retrieved from the structure (Figure 2) constructed by AMG by detecting Ω, the set of nodes W_h^k which satisfy the following conditions:

$$h > \lambda, \tag{5}$$

$$|\Psi_h^k| > 1, \tag{6}$$

$$|\Psi_h^k| \neq |\Psi_{h+1}^k|, \tag{7}$$

where "| |" indicates the number of elements in the set. Eq. 5 restricts the length of candidate motifs to be at least λ, Eq. 6 guarantees that the captured candidate motifs have occurred at least twice, Eq. 7 is used to choose the set of detected patterns with the longest length.

These detected patterns are used to populate the candidate motif matrix by assigning '1' to the positions where the patterns occur, specifically, the repeating patterns are at coordinates $(\Psi_m^k(j), f_m^k)$ of $W_{m=1..h}^k$ where $W_h^k \in \Omega$, and $\Psi_m^k(j)$ are the elements of Ψ_m^k

For the convenience of processing, the candidate motif matrix is then converted to a time-lag matrix $\mathbf{T}$ [9]. By this conversion, the repeating patterns will be reflected as parallel vertical lines in $\mathbf{T}$ and redundant neighboring patterns can be easily detected and removed.

B. Pattern Merging

During the construction of the AMG structure, the algorithm will record every possible candidate motif. Hence, many neighboring candidates will be generated. These neighboring motifs should be merged to find longer repeating patterns. To merge these neighboring motifs, each vertical line in the time lag matrix is first dilated and then eroded to form a new sparse matrix $\mathbf{T_1}$. The objective is to extract the longest vertical line for each neighborhood.

C. Pattern Duplication

If the input sequence is noisy, the AMG algorithm may not be able to capture every repeating pattern of a candidate motif. However, the repetition occurrence in the candidate motif matrix can be further analyzed to sought out missing repeating pairs. To describe what we mean, we describe the following scenario: if segments Q_1, Q_2 and Q_3 are sequences of the same motif, but due to noise, the sequences $\{Q_1, Q_2\}$ were found under one candidate motif selection, while the sequences $\{Q_2, Q_3\}$ were found in a separate motif selection. Let δ_{jk} denote the frame index differences between pattern $\{Q_j, Q_k\}$. Due to the characteristics of the time-lag matrix, the pattern Q_2 will be marked in two locations, one is a vertical line that is parallel to Q_1 with a distance of δ_{12} along the x-axis, and the other vertical line is along the y-axis with a distance of also δ_{12} to Q_1(see Figure 4).

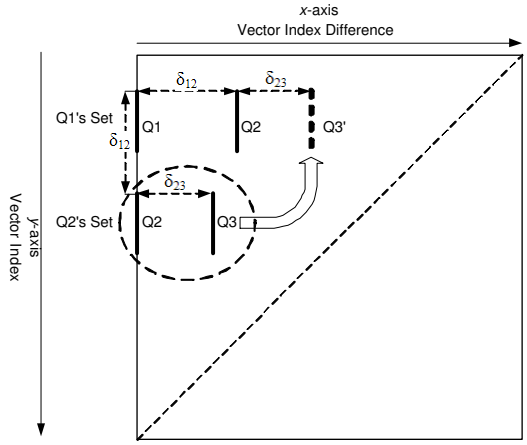

Fig. 4. The illustration of the repetition occurrence and pattern duplication process in the time-lag matrix.

In this situation, we can infer from the time-lag matrix that Q_3 should also be included in Q_1's set [9]. To exploit this relationship, our refinement process will duplicate all vertical lines in $\mathbf{T_1}$ to all possible candidate positions belonging to the same family hence to form a new matrix $\mathbf{T_2}$. The process is illustrated in Figure 4.

D. Boundary Refinement

Noise may also cause mis-alignments in the boundaries of the obtained candidate motifs. The boundaries can be aligned by either extending or shrinking the detected patterns. The details of the boundary alignment process can be found in [9]. Therefore, a new matrix $\mathbf{T}'$ can be obtained by aligning the boundaries in $\mathbf{T_2}$. After the refinement process, the final motifs can be easily retrieved from matrix $\mathbf{T}'$.

V. Experiments and Discussion

Using a corpus of 30 songs [1], we compare the recall and precision of repetitions obtained using the AMG constructed sparse similarity matrix versus a full similarity matrix. Our test corpus consists of 30 pieces of English pop songs performed by both male and female singers. From the test corpus, 63 repeating segments with an average duration 25.6sec are identified. There are a total of 198 instants for these repeating segments. These instants are used as our ground truth patterns.

To carry out the experiment, the test corpus is first down-sampled to 11KHz, mono and 16 bits per sample for feature extraction. The feature used is derived from the 12 dimensional chroma vector [7] which is calculated from every 250msec music segment with 50msec hop size. The spectrum information in seven musical octaves that span from 32.5Hz to 4KHz is used. Each extracted chroma vector is normalized by dividing each element by the vector's maximum value.

As the chroma feature captures information in a 250msec window, slight offset of the analysis window may affect the feature values. As such, researchers [9] have proposed to generate features from a much longer analysis window. Following their strategy, our used feature is the mean and variance of 16 consecutive chroma features with a hop size of 3 frames. This translates to an analysis window of 1 second and a shift of 150msec to generate a feature vector of 24 dimensions.

This experiment compares the performance of our proposed AMG to the traditional self-similarity approach [9] to search for repeating segments. Both approaches will generate the time-lag matrix using the same features and are post-processed by the same refinement framework as described in Section IV.

The parameter settings used for the AMG approach are: maximum depth of the tree $A = 100$ (this setting allows us to capture 16sec music segment), the segment length $\lambda = 61$ (approximates 10sec duration). The thresholds γ_1 and γ_2 are selected using an independent development corpus of 20 English songs, and they were set as 0.367 and 0.684 respectively.

A. Evaluation Criteria

To evaluate the performance of the two approaches, repeating segments in $\mathbf{T}'$ generated from Section IV are compared with the ground truth patterns. Let $\mathbf{G}$ denote the ground truth patterns in a sparse binarised time-lag matrix for each song. The vertical lines in matrix $\mathbf{G}$ are then duplicated as described by Section IV. To achieve a robust and stable evaluation, a $\pm$1sec window mismatch among repeating segments is allowed. This is achieved by blurring all the lines in $\mathbf{G}$ to form a new matrix $\mathbf{G}'$ for the $\pm$1sec offset error. Therefore, the correctly detected repeating segments can be found in matrix $\hat{\mathbf{T}}$ by

$$\hat{\mathbf{T}} = \mathbf{T}' \wedge \mathbf{G}', \tag{8}$$

where "$\wedge$" denotes the element-wise logical "AND" operator.

In this paper, the recall is defined as $\frac{|\hat{\mathbf{T}}|}{|\mathbf{G}|}$ and the precision is defined as $\frac{|\hat{\mathbf{T}}|}{|\mathbf{T}'|}$, where "$|\ |$" denotes number of non-zero elements in the matrix.

[1] The 30 songs' feature database as well as their repeating segments location information are released at http://www3.ntu.edu.sg/home/aseschng/SpeechTechWeb/projects/projects.htm.

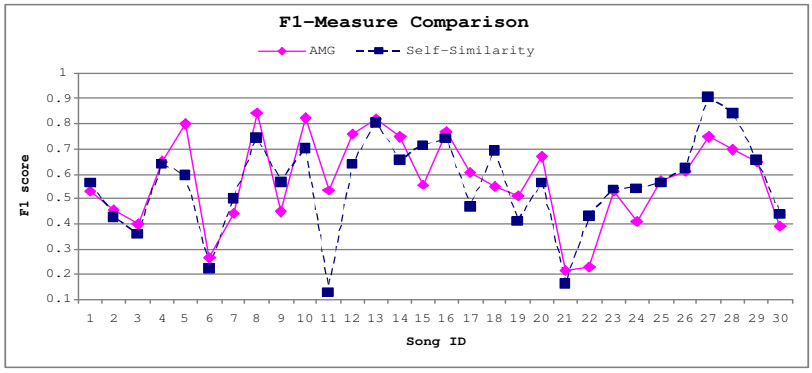

Fig. 5. The F1-measure for each individual song using the self-similarity matrix approach and our proposed AMG approach.

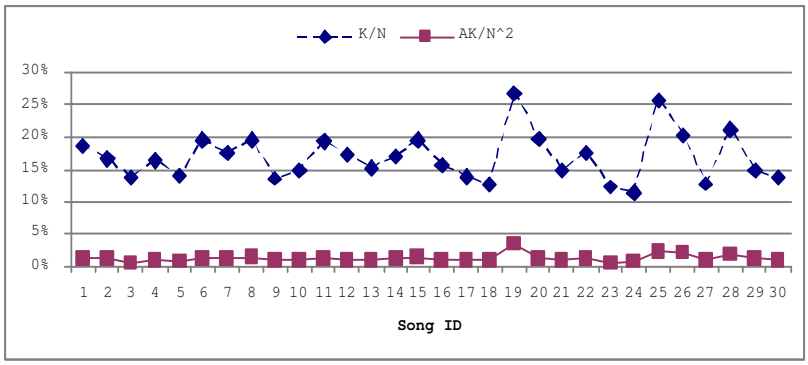

Fig. 6. The above figure compares the memory usage requirement between the AMG and self-similarity approach for each individual song. The average memory requirement of AMG is approximately 2% of that required by the self-similarity approach. The ratio K/N is also plotted to illustrate the number of branches created for each song.

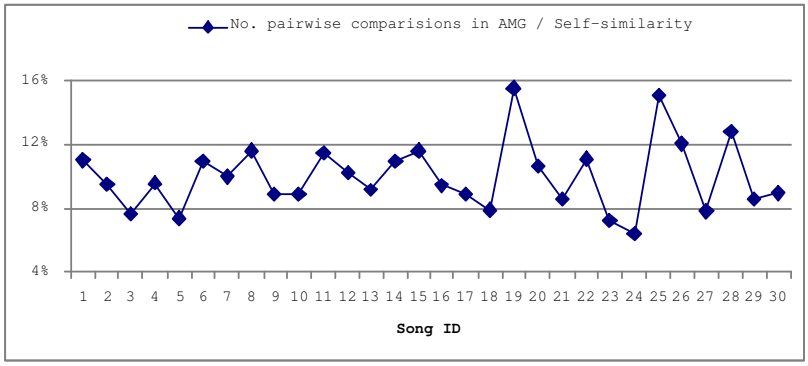

Fig. 7. The above figure compares the computational requirements between the AMG and self-similarity approach for each individual song.

B. Evaluation Results

The F1 measure for each individual song is shown in Figure 5. The average recall, precision and F1-measure of all the testing songs are shown in Table II.

As the self-similarity approach calculates the similarity measure between all pairwise vectors, the matrix will contain all similarity information in the sequence for motif detection. Therefore, the ability to extract and find correct motifs relies on the pattern refinement process. Since the refinement process and features used are the same for the two approaches, this experiment measures how well the AMG approach can capture the "correct" candidate motifs. Our results in Figure 5 and Table II show that the two approaches have almost the same

TABLE II
AVERAGE PERFORMANCE COMPARISON BETWEEN THE AMG AND SELF-SIMILARITY APPROACH FOR 30 SONGS.

	Recall	Precision	F1
AMG	0.590	0.560	0.575
Self-Similarity	0.569	0.558	0.564

performance - This indicates that the proposed AMG approach is able to generate correct candidate motifs.

In addition, we compared the memory usage between the AMG and self-similarity approach. The AMG structure (Figure 2) stores all the similarity information in each node and hence requires $O(AK)$ memory space. However, the self-similarity approach requires $O(N^2)$ memory space to store all pairwise similarity measurements. To illustrate the difference in memory usages between the two approaches, we plot the ratios AK/N^2 and K/N in Figure 6 for each song's experiment.

Furthermore, we evaluated the element occupancy of the candidate motif matrix and found that for all 30 songs data, the candidate matrix is 99% sparse.

To compare the computation complexity requirement between the similarity approach and the AMG approach, we measure the computational requirements needed to create the sparse time-lag matrix by these two approaches (Figure 3). For the self-similarity approach, it requires $N(N-1)/2$ number of pairwise similarity measures (Eq. 1). For the AMG approach, we recorded the total number of vector-to-vector similarity comparisons for the 30 songs (Figure 7). The results show that the AMG approach requires an average of 10% computation cost as compared to the self-similarity approach. Furthermore, the self-similarity approach requires additional processing such as low-pass filtering [8] and other image processing techniques [9] to generate the sparse time-lag matrix. The AMG approach is again computationally more efficient as the sparse time-lag matrix can be directly generated from the candidate motif matrix [9].

VI. Conclusions

We have proposed a novel approach, AMG, to detect the repeating patterns in a sequence of vectors. The online linear-time suffix tree construction algorithm is extended to accept vectors as input so that the exact matching criteria are avoided. To make the algorithm more robust, segment-to-segment similarity is used to smooth the outliers in the sequence. The candidate repeating segments obtained by the AMG method are then refined using a sparse similarity matrix to find the final patterns.

The experimental results show that AMG is significantly more efficient in terms of computation and memory requirement as compared to self-similarity approach and yet achieved similar performance to discover repeating segments in a 30 songs' corpus.

References

[1] A. Mohapatra, P. M. Mishra, and S. Padhy, "Motif search in DNA sequences using generalized suffix tree," in *Proc. ICIT '07*, Dec. 2007.

[2] M. F. Sagot, "Spelling approximate repeated or common motifs using a suffix tree," in *Latin'98: Theoretical informatics*, pp. 374–390. 1998.

[3] L. Xie, *Unsupervised pattern discovery for multimedia sequences*, Ph.D. thesis, Columbia University, 2005.

[4] D. Minnen, C. L. Isbell, I. Essa, and T. Starner, "Discovering multivariate motifs using subsequence density estimation and greedy mixture learning," in *Twenty-Second Conf. on Artificial Intelligence (AAAI-07)*, Vancouver, B.C., July 2007.

[5] A. S. Park and J. R. Glass, "Unsupervised pattern discovery in speech," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 16, no. 1, pp. 186–197, Jan. 2008.

[6] J. Foote, "Visualizing music and audio using self-similarity," in *Proc. ACM MM '99*, 1999, pp. 77–80.

[7] M. A. Bartsch and G. H. Wakefield, "To catch a chorus: Using chroma-based representations for audio thumbnailing," in *Proc. Workshop on Applications of Signal Processing to Audio and Acoustics '01*, Oct. 2001, pp. 15–18.

[8] M. Goto, "A chorus section detection method for musical audio signals and its application to a music listening station," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1783–1794, Sept. 2006.

[9] L. Lu, M. Wang, and H.-J. Zhang, "Repeating pattern discovery and structure analysis from acoustic music data," in *Proc. ACM MIR '04*, New York, NY, USA, 2004, pp. 275–282.

[10] M. Müller and F. Kurth, "Enhancing similarity matrices for music audio analysis," in *Proc. ICASSP '06*, Toulouse, France, May 2006, pp. V9–V12.

[11] T. Zhang and R. Samadani, "Automatic generation of music thumbnails," in *Proc. ICME '07*, 2007, pp. 228–231.

[12] Meinard Müller, *Information Retrieval for Music and Motion*, Springer Berlin Heidelberg, 2007.

[13] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel, "Finding motifs in time series," in *Proc. the Second Workshop on Temporal Data Mining*, Edmonton, Alberta, Canada, July 2002.

[14] D. Minnen, T. Starner, I. Essa, and C. Isbell, "Discovering characteristic actions from on-body sensor data," in *Proc. the 10th IEEE International Symposium on Wearable Computers*, Oct. 2006, pp. 11–18.

[15] Bill Chiu, Eamonn Keogh, and Stefano Lonardi, "Probabilistic discovery of time series motifs," in *KDD '03: Proc. the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Washington, D.C., Aug. 2003, pp. 493–498.

[16] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, "Detecting time series motifs under uniform scaling," in *KDD '07: Proc. the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 844–853.

[17] Y. Tanaka, K. Iwamoto, and K. Uehara, "Discovery of time-series motif from multi-dimensional data based on MDL principle," *Machine Learning*, vol. 58, no. 2-3, pp. 269–300, Mar. 2005.

[18] J.-L. Hsu, C.-C. Liu, and A. L. P. Chen, "Discovering nontrivial repeating patterns in music data," *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 311–325, Sept. 2001.

[19] I. Karydis, A. Nanopoulos, and Y. Manolopoulos, "Finding maximum-length repeating patterns in music databases," *Multimedia Tools and Applications*, vol. 32, no. 1, pp. 49–71, Jan. 2007.

[20] N. C. Maddage, C. Xu, M. S. Kankanhalli, and X. Shao, "Content-based music structure analysis with applications to music semantics understanding," in *Proc. ACM MM '04*, New York, NY, USA, 2004, pp. 112–119.

[21] W. Chai, "Structural analysis of musical signals via pattern matching," in *Proc. ICASSP '03*, Hong Kong, China, Apr. 2003, vol. 5, pp. 549–552.

[22] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD '90*, 1990, pp. 322–331.

[23] Jeremy Buhler and Martin Tompa, "Finding motifs using random projections," in *RECOMB '01: Proc. the Fifth Annual International Conference on Computational Biology*, Montreal, Quebec, Canada, Apr. 2001, pp. 69–76.

[24] Jiawei Han and Micheline Kamber, *Data mining: concepts and techniques*, Elsevier Inc., 2nd edition, 2006.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, Sept. 1999.

[26] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis, "On clustering validation techniques," *Journal of Intelligent Information Systems*, vol. 17, no. 2-3, pp. 107–145, Dec. 2001.

[27] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, "BIRCH: an efficient data clustering method for very large databases," in *Proc. ACM SIGMOD '96*, Montreal, Quebec, Canada, 1996, pp. 103–114.

[28] Rakesh Agrawal, Christos Faloutsos, and Arun Swami, "Efficient similarity search in sequence databases," in *Proc. the Fourth International Conference on Foundations of Data Organization and Algorithms*, Chicago, Illinois, Oct. 1993, pp. 69–84.

[29] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos, "Fast subsequence matching in time-series databases," in *Proc. ACM SIGMOD '94*, Minneapolis, Minnesota, 1994, pp. 419–429.

[30] H. André-Jnsson and D. Badal, "Using signature files for querying time-series data," in *Proc. Principles of Data Mining and Knowledge Discovery, 1st European Symposium*, Trondheim, Norway, June 1997, pp. 211–220.
[31] K. V. Ravi Kanth, Divyakant Agrawal, and Ambuj Singh, "Dimensionality reduction for similarity searching in dynamic databases," in *Proc. ACM SIGMOD '98*, Seattle, Washington, 1998, pp. 166–176.
[32] T. Kahveci and A. Singh, "Variable length queries for time series data," in *Proc. ICDE '01*, Heidelberg, Germany, 2001, pp. 273–282.
[33] E. Keogh, J. Lin, and W. Truppel, "Clustering of time series subsequences is meaningless: implications for previous and future research," in *Proc. ICDM '03*, Nov. 2003, pp. 115–122.
[34] Dennis Shasha and Tsong-Li Wang, "New techniques for best-match retrieval," *ACM Transactions on Information Systems*, vol. 8, no. 2, pp. 140–158, Apr. 1990.
[35] C.-H. L. Lee, A. Liu, and W.-S. Chen, "Pattern discovery of fuzzy time series for financial prediction," *IEEE Trans. Knowledge and Data Engineering*, vol. 18, no. 5, pp. 613–625, May 2006.
[36] T. Oates, "PERUSE: An unsupervised algorithm for finding recurring patterns in time series," in *Proc. ICDM '02*, Dec. 2002, pp. 330–337.
[37] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, Sept. 1995.
[38] E. M. McCreight, "A space-economical suffix tree construction algorithm," *Journal of the ACM*, vol. 23, no. 2, pp. 262–272, 1976.
[39] J. Pevsner, *Bioinformatics and Functional Genomics*, John Wiley & Sons, Inc., 2003.