EFFICIENT SPARSE SELF-SIMILARITY MATRIX CONSTRUCTION
FOR REPEATING SEQUENCE DETECTION

Lei Wang\textsuperscript{1}, Eng Siong Chng\textsuperscript{1} and Haizhou Li\textsuperscript{1,2}

\textsuperscript{1}School of Computer Engineering, Nanyang Technological University, Singapore 639798
\{wang0161, aseschng\}@ntu.edu.sg
\textsuperscript{2}Institute for Infocomm Research, 1 Fusionopolis Way, Singapore 138632
hli@i2r.a-star.edu.sg

ABSTRACT

This paper presents an efficient way to construct the self-similarity matrix, a popular approach, to detect repeating segments in music. Our proposed method extends the sparse suffix tree construction algorithm to accept vectors as input to construct an initial selection of repeating sequences to generate a sparse self-similarity matrix. Our proposed insertion criterion does not only rely on vector-to-vector similarity but also measures the similarity between two subsequences in its insertion criteria. As such, our method is more robust as compared to approaches that simply quantize the input vectors into symbols for suffix tree construction. In addition, the proposed method is efficient in both computation and memory storage. Our experimental results showed that the proposed approach obtains similar average F1 score as compared to the traditional self-similarity approach with much less computational cost and memory usage.

Index Terms— Music, Acoustic signal analysis, Multidimensional sequences, Information retrieval, Pattern recognition

1. INTRODUCTION

Unsupervised repeating pattern detection remains a challenging problem and has attracted a lot of interest. Research works such as the detection of DNA repeating sequences [1], repeating segments in music [2, 3, 4, 5, 6], repeating video subsequence [7] are all applications of this field.

In this paper, we examine the problem to automatically detect repeating segments in music. We first discuss two types of approaches that have been proposed. The first type converts the continuous values into discrete symbols and apply text processing techniques such as suffix tree, RP-tree [8], R-tree [9] and incremental search [10] to detect repeating segments. The other approach is to directly analyze the continuous audio input by comparing the distances among its input vectors using self-similarity matrix [11, 2, 3, 4, 5, 6].

In this work, we focus on the self-similarity approach as it is a popular and robust technique to analyze repeating sequences. This approach is robust as algorithms to ‘search’ for the repetition are performed on the 2-D similarity matrix and hence can exploit advance image processing techniques to recover noisy repetition sequences across neighboring vectors. The use of traditional self-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 a 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 factor we are concerned with is the computational complexity and memory requirements of the method. We propose the AMG (Adaptive Motif Generation) algorithm to efficiently construct a sparse similarity matrix using a structure which is similar to suffix tree. Our proposed method modifies the symbolic suffix tree algorithm to accept vectors as input 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 subsequences can then be extracted from the constructed tree to populate a sparse self-similarity matrix. As the number of repeating segments obtained in the suffix tree step 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 computation and memory.

The remainder of the paper is organized as follows: Section 2 introduces our approach to construct the sparse self-similarity matrix using AMG, and the extraction of repeating patterns from the self-similarity matrix. Section 3 reports the experimental results and finally, we conclude in Section 4.

2. SPARSE SELF-SIMILARITY MATRIX CONSTRUCTION

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

Our approach exploits the suffix tree [12] structure to detect repeating subsequence. The suffix tree can be used as an efficient data representation to capture repeating subsequences of a symbolic sequence. The main idea of our approach is to construct 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 truncated suffixes using a tree structure as shown in Figure 1. We modify the Ukkonen’s online suffix tree construction algorithm [13] to create the sparse suffix tree in a linear
time fashion.

![Fig. 1. The resulting candidate motif representation with Adaptive Motif Generation algorithm.](image)

The original Ukkonen’s algorithm can only examine symbolic sequences, it cannot analyze sequences of vectors. This section introduces our novel extension of the suffix tree construction algorithm to accept vectors as input. Specifically, we modified the insertion criteria to compare similarity measures among vectors with an adaptive threshold for robust insertion. In addition, we restrict the tree’s internal node to have only one child, this is to limit the tree’s growth for practical realization. This new online tree construction algorithm is named Adaptive Motif Generation (AMG) in which “motif” means the repeating segments of the music piece.

2.1. Adaptive Motif Generation

Figure 1 shows the overall structure constructed by the AMG algorithm. Each circle in the tree represents a node $W_h$ where $h$ denotes the depth and $k$ the branch number of the tree. Let $K$ denote 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.

In AMG representation, each internal node is restricted to having only one child. Hence, each branch represents a candidate motif, and the tree structure can be interpreted as a list of $K$ candidate motifs. To characterize the motifs, each node $W_h^k$ contains four types of information: the node’s template vector $v_h^k$, the time stamp $f_h^k$, where $v_h^k$ first occurred, a similarity threshold $\alpha_h^k$, and a list $W_h^k$ that contains the time stamps of all input vectors that have been inserted into the node.

The AMG algorithm is summarized in Table 1. The following symbols are used: $W_{h}^k$ is the root node, $u_i$ is the input vector at frame index $i$, $N$ is the number of features in the sequence, $\gamma_i$ is a user defined threshold value, set1 and set2 are the sets of modified nodes during each iteration of the algorithm. The node pool set1 is the set of candidate nodes that may accept $u_i$ as its children during each iteration of the construction algorithm. The set of candidate nodes are generated by $f_{root}(W_0^0, u_i)$ and $f_{branch}(W_h^k, u_i)$. These two functions evaluate if the input vector $u_i$ needs to be inserted into the tree at the root node or non-root node respectively. The insertion process is similar to [13] and has linear time complexity.

### Table 1. AMG algorithm.

<table>
<thead>
<tr>
<th>Input: vector sequences $u_i$, $i = 1, 2, \ldots, N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize: create first branch’s node $W_1^1$ for input $u_1$</td>
</tr>
<tr>
<td>1. $K = 1, i = 1, h = 1$</td>
</tr>
<tr>
<td>2. Create $W_h^K$ with $v_h^K \rightarrow u_i$, $f_h^K \rightarrow i$, $\Psi_h^K \rightarrow {i}$, $\alpha_h^K = \gamma_i$.</td>
</tr>
<tr>
<td>3. set1 $\leftarrow {W_h^K}$</td>
</tr>
<tr>
<td>4. set2 $\leftarrow {}$</td>
</tr>
<tr>
<td>For $i = 2 : N$</td>
</tr>
<tr>
<td>{modified nodes} $= f_{root}(W_0^0, u_i)$</td>
</tr>
<tr>
<td>set2 $\leftarrow {modified nodes}$</td>
</tr>
<tr>
<td>For each node $W_h^k \in set1$</td>
</tr>
<tr>
<td>$\bullet$ {modified nodes} $= f_{branch}(W_h^k, u_i)$</td>
</tr>
<tr>
<td>$set2 \leftarrow set2 \cup {modified nodes}$</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>set1 $\leftarrow set2$</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

**H1:** To create a new branch in the tree, i.e. a new node at depth 1, the criterion is that the input vector $u_i$’s similarity to all current depth 1 nodes $W_1^k$’s template vector $v_1^k$ are less than a user defined threshold $\gamma_i$. Hence, **H1** is used to create a new candidate motif branch if no existing branches of the tree have the similar starting template vector. In this paper, the similarity metric used to measure the similarity between two vectors $v$ and $u$ is the Pearson correlation coefficient $\rho(v, u)$ which is defined as

$$\rho(v, u) = \frac{\sum_{r=1}^{d}(v_r - \bar{v})(u_r - \bar{u})}{\sqrt{\sum_{r=1}^{d}(v_r - \bar{v})^2 \sum_{r=1}^{d}(u_r - \bar{u})^2}}$$

(1)

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

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

$$\beta_i^k = \rho(v_1^k, u_i)$$

(2)

If $\beta_i^k$ is greater than the node’s threshold $\alpha_i^k$, the corresponding node’s $W_1^k$ 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.

2.1.2. Insertion at branch node: $f_{branch}(W_h^k, u_i)$

The nodes $W_h^k$ in set1 can be considered as "active" nodes, i.e., the sequence of template vectors at these active branches from depth of $1 \ldots h$ has matched the recent input vector sequence $u_{i-h} \ldots u_{i-1}$. We now need to examine if the current input vector $u_i$ can be appended to these active branches.

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

- **H3:** $u_i$ can be used to create a child node $W_h^{k+1}$.
- **H4:** $u_i$ can be inserted into the existing child node $W_h^{k+1}$.

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 $u_i$ can be inserted into the existing $k^{th}$ branch of the tree. Different to **H2**’s Eq. 2 which evaluate only the similarity between the
allow more branches to be created. In our work, the γ value determines whether a new branch should be created for the input vector being inserted, while an unreasonably low setting will generate too many false alarms. Hence, the value of γ is needed because: (i) the candidate repeating segments may contain redundant patterns, (ii) the candidate repeating segments can be merged to form longer patterns, (iii) the AMG algorithm may miss detecting true repeating segments, and (iv) we need to align the boundaries of the found segments.

In this paper, the recall is defined as \[\frac{|G\cap T|}{|G|}\] and the precision is defined as \[\frac{|G\cap T|}{|T|}\], where \(|\cdot|\) denotes number of non-zero elements in the matrix.

3.1. Evaluation Criteria

To evaluate the performances, the repeating segments in \(T'\) generated from Section 2.3 are compared with the ground truth patterns. Let \(G\) denote the ground truth patterns in a sparse binarized time-lag matrix for each song. To achieve a robust and stable evaluation, a ±1sec offset error is allowed. This is achieved by blurring all the lines in \(G\) to form a new matrix \(G'\). Therefore, the correctly detected repeating segments can be found in matrix \(\hat{T}\) by carrying out element-wise logical “AND” operation on \(T'\) and \(G'\).

In this paper, the recall is defined as \[\frac{|\hat{T}\cap T'|}{|T'|}\] and the precision is defined as \[\frac{|\hat{T}\cap T'|}{|\hat{T}|}\], where \(|\cdot|\) denotes number of non-zero elements in the matrix.

3.2. Evaluation Results

The F1 measure for each individual song is shown in Figure 2. The average recall, precision and F1-measure of the entire 30 songs are shown in Table 2.

As the self-similarity approach calculates the similarity measure for all the pairwise vectors, the self-similarity matrix contains all similarity information in the sequence for motif detection; Hence 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 will illustrate if

\[\beta_{h+1}^k = \sum_{r=0}^{\min(h,\lambda-1)} \rho(W_{h+1}^k, u_{r-r}) \min(h,\lambda-1) \]

where \(h\) is the user defined segment length.

To verify if \(u_t\) can be inserted into the \(k^{th}\) branch of the tree, \(\beta_{h+1}^k\) is compared to threshold \(\alpha_{h+1}^k\). If \(\beta_{h+1}^k\) is greater than \(\alpha_{h+1}^k\), the input vector time stamp information \(I\) will be included by \(\Psi_{h+1}^k\), and the function \(f_{branch}(W_{h+1}^k, u_t)\) returns the node \(W_{h+1}^k\), otherwise null.

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

\[\alpha_{h+1}^k = \alpha_h^k - (\beta_{h+1}^k - \beta_{h}^k),\]

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

2.2. The AMG Parameter Settings

A set of four parameters control the motif discovery process in AMG algorithm, namely, \(A, \lambda, \gamma_1\) and \(\gamma_2\). 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 repeating segment. The parameter \(\lambda\) specifies the length of the segment to evaluate the similarity measure as specified in Eq. 3. The value of \(\lambda\) should be chosen long enough to capture a candidate repeating sequence. The threshold \(\gamma_1\) is used to initialize \(\alpha_1^k\) and its setting is important since it determines if each new input vector will be inserted into the branch. An unnecessarily high setting will stop a true repeating sequence from being inserted, while an unnecessarily low setting will generate too many false alarms. Hence, the value of \(\gamma_1\) is critical to the success of AMG. In our work, a small development corpus is used to determine \(\gamma_1\) by evaluating all pairwise similarity between the vectors and selecting the top \(10\%\) percentile pairwise similarity value as \(\gamma_1\). The threshold \(\gamma_2\) is used in the evaluation of hypothesis H1. The value determines whether a new branch should be created for the input vector. The range of \(\gamma_2\) is \(\gamma_1 < \gamma_2 \leq 1\). A higher \(\gamma_2\) value will allow more branches to be created. In our work, \(\gamma_2\) is set to the top \(5\%\) percentile pairwise similarity value of the development corpus.

2.3. Candidate Repeating Segments Refinement

The candidate repeating segments found using AMG are used to populate a sparse self-similarity matrix. A refinement process on this sparse matrix to extract the final candidate repeating segments is needed because: (i) the candidate repeating segments may contain redundant patterns, (ii) the candidate repeating segments can be merged to form longer patterns, (iii) the AMG algorithm may miss detecting true repeating segments, and (iv) we need to align the boundaries of the found segments.

To address the above issues, the sparse matrix is converted to a time-lag matrix \(T\) as in [4]. By this conversion, the repeating patterns will be reflected as parallel vertical lines in \(T\) and then we can follow works of [2, 3, 4] to refine all the candidate repeating segments to obtain the final patterns in \(T'\).

3. EXPERIMENTS AND DISCUSSION

Using a database of 30 songs, we compare the recall and precision of repetitions obtained using the AMG constructed sparse similarity matrix versus a full self-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 instances for these repeating segments and they are used as 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 [2] 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 chroma vector is normalized by dividing each element by the vector’s maximum value.

As the similarity measure is approximated by a 250msec window, slight offset of the analysis window may affect the feature values. As such, researchers [4] 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.\(^1\)

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 \(\gamma_1\) and \(\gamma_2\) are selected using an independent development corpus of 20 English songs, and they were set as 0.367 and 0.684 respectively.

\(^1\)The feature database along with the evaluation script are released at www.ntu.edu.sg/home5/wang0161.
the AMG approach can capture the “correct” candidate motifs. Our results in Figure 2 and Table 2 show that the two approaches have almost the same performance. This indicates that the proposed AMG approach is able to generate correct candidate motifs.

In addition, we compared the memory usage between AMG and self-similarity approach. The AMG structure (Figure 1) stores all the similarity information in each node and hence it 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 3.

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 self-similarity approach and the AMG approach, we need to compare the computational requirements needed to create the sparse time-lag matrix by these two approaches as the refinement process. The time-lag matrix by these two approaches as the refinement process and other image processing techniques [4] 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 [4].

4. CONCLUSION

We have proposed a novel approach, AMG, to detect the repeating patterns in a sequence of vectors. The online 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 motifs obtained by the AMG method are used to populate a sparse self-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.

5. REFERENCES