MODELING MUSICAL SOUNDS WITH AN INTERPOLATING STATE MODEL

Anssi Klapuri, Tuomas Virtanen, and Marko Helén

Institute of Signal Processing, Tampere University of Technology
Korkeakoulunkatu 1, 33720 Tampere, Finland
phone: +358 3 3115 2124, fax: +358 3 3115 4954, email: {klap,tuomasv,heln} @cs.tut.fi
web: www.cs.tut.fi/∼klap/

ABSTRACT

A computationally efficient algorithm is proposed for modeling and coding the time-varying spectra of musical sounds. The aim is to encode individual data sets and not the statistical properties of the sounds. A given sequence of acoustic feature vectors is modeled by finding such a set of "states" (anchor points in the feature space) that the input data can be efficiently represented by interpolating between them. The achieved modeling accuracy for a database of musical sounds was approximately two times better than that of a conventional "vector quantization" model where the input data was k-means clustered and the input data vectors were then replaced by their corresponding cluster centroids. The computational complexity of the proposed algorithm as a function of the input sequence length T is $O(\mathrm{T}\log\mathrm{T})$.

1. INTRODUCTION

This paper proposes an algorithm for modeling and coding the time-varying spectra of musical sounds. In particular, we focus on modeling individual data sets, not the statistical properties of the sounds. This has applications in structured audio coding (object-based music coding) and in sound source recognition.

Many musical sounds are very poorly modeled using a conventional state model where each state generates its characteristic spectral energy distribution and time-varying spectra are modeled by switching between the states. On the contrary, most musical sounds can be represented very efficiently by *interpolating* between spectra that are taken from appropriate temporal positions. Several examples of this can be found in sound synthesis.

In modeling speech signals, the limitations of e.g. hidden Markov models (HMM) are well known. Improvements towards addressing the context-dependent nature of observations include e.g. triphone models and segment models [1], [2], [3]. The spectral energy distribution of musical sounds is generally more slowly-varying than that of speech sounds and the limitations of the conventional state models are even earlier encountered. As an example, consider the power envelope of a piano sound at three subbands as illustrated in the left panel of Fig. 1. The input data (solid line) consists of three-dimensional feature vectors that encode the rough spectral energy distribution as a function of time. The conventional "vector quantization" approach where the input data is replaced by a sequence of quantized vectors (i.e., discrete "states", see the dashed line) would require a very large number of states to represent the data accurately. A drawback of the segment models [1], in turn, is that they are better suited for modeling the statistical properties of (speech) signals and increase the model complexity, i.e., the amount of stored parameter data. Triphone models are not always well matched to music signals.

The right panel of Fig. 1 illustrates the proposed interpolating state model. The amount of stored model data is the same as in the left panel (four different model vectors, i.e., state vectors), yet the model achieves much better fit to the data. The occurrence times of the four model vectors are indicated as vertical lines, with the number of the occurred state at the x-axis, and the original data is then represented by interpolating between each two consecutive state vectors. The data-driven interpolation model proposed by Sun in [4] for the statistical modeling of speech signals is closely related to the present model but the author did not propose an algorithm for estimating the state vectors (anchor points) of the model.

In this paper, a computationally efficient algorithm is proposed for estimating the parameters of the interpolating state model which will be more exactly defined in Sec. 2. Whereas globally optimal parameters for this model are computationally intractable, the method finds a suboptimal solution. Extensive simulation experiments were carried out which show that the proposed model approximately halves the modeling error of a conventional "vector quantization" approach where the input data is k-means clustered and the input data vectors are replaced by the corresponding cluster centroids. The time-complexity of the proposed method is proportional to $O(\mathrm{T}\log(\mathrm{T})\mathrm{K}^2\mathrm{D})$, where T is the length of the input data sequence, K is the desired number of states between which the input data is interpolated, and D is the dimensionality of the data.

2. MODEL FORMULATION

The input data to be modeled is a sequence of feature vectors $\mathbf{x}(i)$ which have been extracted in successive time frames $i = 0,\ldots,\mathrm{T}-1$ of an acoustic signal. The dimensionality of the feature vectors is denoted by D and a matrix $X = [\mathbf{x}(0),\ldots,\mathbf{x}(\mathrm{T}-1)]^{\mathrm{T}}$ of size $(\mathrm{T}\times\mathrm{D})$ is used to denote the complete data set to be modeled.

The basic idea of our model is to find a limited number of "state vectors" (anchor points) in the feature space so that the original data can be efficiently approximated by interpolating between these. The proposed representation can be most conveniently described as a state model. There are $\mathrm{K} \ll \mathrm{T}$ states, each of which corresponds to a distinct D-dimensional state vector $\mathbf{s}(k)$. For convenience, the state vectors are collected into a matrix $S = [\mathbf{s}(0),\ldots,\mathbf{s}(\mathrm{K}-1)]^{\mathrm{T}}$.

Transitions from a state to another do not occur instantly, but during a certain period of time. The output of the model is generated at the transitions *between* the states, as a linear interpolation of the state vectors at the two ends of the transition. Figure 2 illustrate this situation. During the transition, the model moves with a constant speed towards the next state and, when that is reached, the model immediately starts moving towards another state. The occurrence times of the states in their pure form are called *nodes* where, mo-

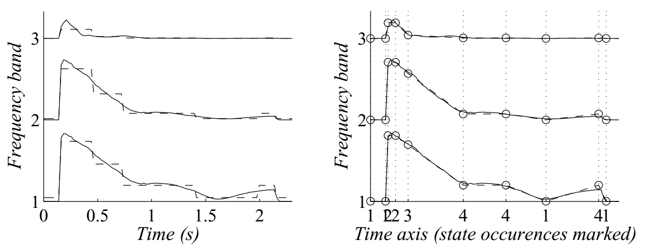

Figure 1: Left: Power envelope of a piano sound at three subbands (solid line) modeled with k-means clustering (dashed line). Right: The same envelope modeled with an interpolating state model (line connecting the circles). The vertical lines indicate state occurences.

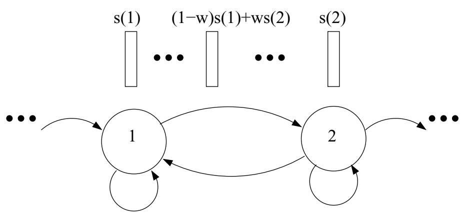

Figure 2: Data generation in an interpolating state model.

mentarily, the output corresponds exactly to one state vector. The durations of the transitions may be arbitrary.

Note that transitions between all state pairs have to be allowed. For example, the sound of a transverse flute was found to loop around a same state sequence during the sustained vibrato portion of the sound. Allowing the transitions back to the previous states greatly reduces the number of states needed for modeling.

The nodes $n = 0, \dots, \mathrm{N}-1$ are characterized by a time stamp $t(n) \in [0, \mathrm{T}-1]$ and the number of the state that occurs at the node, $q(n) \in [0, \mathrm{K}-1]$. The nodes are always kept in ascending temporal order, i.e., $t(n) < t(n+1)$, and the number of nodes N satisfies $\mathrm{K} \le \mathrm{N} \le \mathrm{T}$. One node is always positioned in the beginning and in the end of the data sequence, i.e., $t(0) = 0$ and $t(\mathrm{N}-1) = \mathrm{T}-1$. For convenience, we define $\mathbf{t} = [t(0), \dots, t(N-1)]$ and $\mathbf{q} = [q(0), \dots, q(N-1)$.

The model is completely specified by the three data structures $\mathbf{t}$, $\mathbf{q}$, and S. The output of the model at time i between two nodes, $t(n) \le i \le t(n+1)$ is a linear interpolation of the state vectors that occur at the two nodes:

$$\hat{\mathbf{x}}(i) = (1 - w_n(i))\mathbf{s}(q(n)) + w_n(i)\mathbf{s}(q(n+1)), \qquad (1)$$

where

$$w_n(i) = \frac{i - t(n)}{t(n+1) - t(n)}. \qquad (2)$$

It should be noted that the model is completely deterministic, contrary to e.g. hidden Markov models or stochastic segment models [1]. No probability distributions are applied.

3. ALGORITHM FOR PARAMETER ESTIMATION

A remaining problem is to estimate the model parameters $\mathbf{t}$, $\mathbf{q}$, and S given the input data X and the desired number of states K so as to minimize the sum of squared error between the original data X and the output of the model, $\hat{X}, \hat{X} = [\hat{\mathbf{x}}(0), \dots, \hat{\mathbf{x}}(\mathrm{T}-1)]$:

$$e = \sum_{i=0}^{\mathrm{T}-1} \sum_{d=0}^{\mathrm{D}-1} \left(X_{i,d} - \hat{X}_{i,d}\right)^2. \qquad (3)$$

A basic version of the parameter estimation algorithm is first described in Secs. 3.1–3.3 and then the efficient version in Sec. 3.4.

3.1 Least-square error solution for a *given* state sequence

To begin with, consider the situation where the state sequence (as defined by $\mathbf{t}$ and $\mathbf{q}$) is given in advance. In this case, optimal state vectors which minimize (3) can be solved in a closed form using the linear least squares approach ([5], Chapter 8).

The solution can be formulated in the present context as follows. Let W be a $(\mathrm{T} \times \mathrm{K})$ matrix of weights where each row i, $t(n) \le i \le t(n+1)$, contains only two non-zero values:

$$W_{i,k} = \begin{cases} 1 - w_n(i) & \text{for column } k = q(n) \\ w_n(i) & \text{for column } k = q(n+1) \\ 0 & \text{otherwise} \end{cases}. \qquad (4)$$

Then, the least squares solution for the state vectors is [5]:

$$S = (W^{\mathrm{T}}W)^{-1}W^{\mathrm{T}}X. \qquad (5)$$

Next, consider a situation where only a subset of all the state vectors are being solved and the other states are kept fixed (i.e., are already known). Let us denote by $\mathscr{K}$ the set of states that are being updated. First, all columns k for which $k \notin \mathscr{K}$ are removed from W. Then a $(\mathrm{T} \times \mathrm{D})$ matrix Y is defined where each row i, $t(n) \le i \le t(n+1)$ is defined as

$$\mathbf{y}(i)^{\mathrm{T}} = \mathbf{x}(i)^{\mathrm{T}} - I(n)(1 - w_n(i))\mathbf{s}(q(n))^{\mathrm{T}} - I(n+1)w_n(i)\mathbf{s}(q(n+1))^{\mathrm{T}}, \qquad (6)$$

where the indicator function $I(n)$ is defined counterintuitively as

$$I(n) = \begin{cases} 0 & \text{if } q(n) \in \mathscr{K} \\ 1 & \text{if } q(n) \notin \mathscr{K} \end{cases}. \qquad (7)$$

The matrix Y represents a version of the input data sequence where the effect of the fixed states is compensated for.

Finally, the rows i, $t(n) \le i \le t(n+1)$, for which $q(n) \notin \mathscr{K}$ and $q(n+1) \notin \mathscr{K}$ can be removed both from W and Y to get matrices $\tilde{W}$ and $\tilde{Y}$, respectively. The least-squares solution for the state vectors of the states $k \in \mathscr{K}$ can then be written as

$$\tilde{S} = (\tilde{W}^{\mathrm{T}}\tilde{W})^{-1}\tilde{W}^{\mathrm{T}}\tilde{Y}, \qquad (8)$$

where $\tilde{S}$ contains only the states $k \in \mathscr{K}$. Note that the size of the square matrix $(\tilde{W}^{\mathrm{T}}\tilde{W})$ depends on the number of unknowns (size of the set $\mathscr{K}$) and cannot be singular due way the matrix is composed.

3.2 Iterative algorithm to find all the model parameters

A problem with the above-described least squares solution is that there are approximately K^{T} different configurations for the vectors $\mathbf{t}$ and $\mathbf{q}$, i.e., different state sequences that can occur within the time interval $[0, \mathrm{T}-1]$. As testing all these configurations for error minization is not feasible, an iterative algorithm is proposed which finds a suboptimal solution. The processing steps are the following:

1. The algorithm begins by initializing a distinct state vector to *all* the data points of the original data so that, in the beginning, there are $K = \mathrm{T}$ states and $N = \mathrm{T}$ nodes with the state vectors $\mathbf{s}(k) = \mathbf{x}(k)$ for $k = 0, \dots, \mathrm{K}-1$, and the node sequence $t(n) = n$, $q(n) = n$ for $n = 0, \dots, \mathrm{N}-1$.

2. During the iteration, the number of states K is reduced by performing simple atomic operations one-by-one so that, at each step, the added modeling error ("cost") is minimized. Note that during the runtime of the algorithm, K and N are variables which decrease monotonically in value until K reaches the desired number of states K, denoted by $\mathrm{K}_{\mathrm{target}}$ in the following for clarity.

One of the following atomic operations is performed at each iteration step to reduce the number of state vectors:

(a) Delete state. Delete a state and all nodes associated with it. All states which have nodes as immediate neighbours of a deleted node are updated (recomputed using (8)) so as to minimize the error in the neighbourhood of the deleted nodes. The other state vectors are kept fixed. The states containing temporally the first or the last node cannot be deleted.

The left panel of Fig. 3 illustrates a simple case of state deletion. In this example, the state six is being deleted and the node associated with it is marked with an asterisk. The states four and five which occur as immediate neighbours of the deleted node are updated in order to minimize the error over the time span that is marked with a solid line in the figure. The right panel of Fig. 3 illustrates a more general case of state deletion where the deleted state (six) has several associated nodes. The states occurring as neighbours to the deleted nodes (four, five, and seven) are updated. Note that one of the updated states (five) has several associated nodes (including the one at the time $t(n) = 70$).

Repeating only the *delete state* operation until $K = \mathrm{K}_{\mathrm{target}}$ results in a model where each state occurs exactly once (has one associated node) and state transitions occur only from a state to the next state. In order to introduce arbitrary state transitions, another atomic operation is needed.

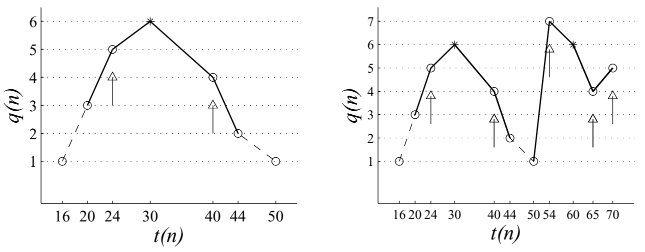

Figure 3: Left: A simple state deletion case. The node of the deleted state is marked with an asterisk and the nodes of the updated states are marked with arrows. Right: a more general case.

(b) Merge state. Merge two states k_1 and k_2. All nodes associated with either of the two states are associated with k_1 after the operation. An optimal state vector is computed for the merged state k_1 using (8) and the other state k_2 is deleted. As a result, the number of states is decreased by one.

The above atomic operations are performed one at a time to reduce the model order until the desired order $\mathrm{K}_{\mathrm{target}}$ is reached. Important in doing this is to choose such an atomic operation at each step that the resulting cost (added modeling error) is minimized.

Repeating the delete and merge operations one-by-one leads to a model with arbitrary state transitions. However, in practical experiments two additional atomic operations turned out to be useful. These are more "garbage collection" type of operations but make a substantial improvement in the actual modeling result.

(c) Delete node. It is useful to be able to remove individual nodes that are associated with states that have several nodes. The *delete node* operation deletes a node n associated with a state $q(n) = k$. After the deletion, the state k and the states occurring as immediate neighbours of the node n are updated (a maximum of three states). If a state has only one associated node, its deletion requires the *delete state* operation.

(d) Move node. Move an individual node one time index towards the past ($t(n) \leftarrow t(n) - 1$) or towards the future ($t(n) \leftarrow t(n) + 1$). The state $q(n)$ is updated after the operation.

3.3 Keeping track of the cost of the atomic operations

In order to keep track of the cost of different operations (and to choose the best at each step), the following data structures are introduced. Deletion and merging costs for the states $k = 1, \ldots, K$ are stored in the vectors $c_{\mathrm{sdel}}(k)$ and $c_{\mathrm{smerge}}(k)$. Along with the latter, a vector $p_{\mathrm{smerge}}(k)$ is needed where the optimal merge partner is stored for each k. Node deletion and node moving costs are stored in the vectors $c_{\mathrm{ndel}}(n)$ and $c_{\mathrm{nmove}}(n)$ for $n = 1, \ldots, N$. Along with the latter, a vector $d_{\mathrm{nmove}}(n)$ is needed which indicates whether a move towards the past or towards the future is better for the corresponding node. Initializing the cost vectors is rather straightforward since in the beginning, each state has only one associated node.

During the iterative algorithm—that is, after performing any of the four atomic operations, we need to update the cost vectors for the states and nodes that have been affected by the operation. The vectors $c_{\mathrm{sdel}}(k)$ and $c_{\mathrm{smerge}}(k)$ have to be updated for the states that have associated nodes as immediate neighbours to the nodes of the states that were *updated* during the previous atomic operation. The vectors $c_{\mathrm{ndel}}(n)$ and $c_{\mathrm{nmove}}(n)$ have to be updated for the nodes that occur as such neighbours.

The costs are computed as follows:

(a) The costs $c_{\mathrm{sdel}}(k)$ are approximated by testing to delete the state k and all its associated nodes and by computing the resulting error (3) *without* performing any updates for the remaining states. This is compared to the error before the deletion to calculate the cost of the operation. It should be noted that this gives an upper bound for the error: if the deletion is later realized, the states occurring as neighbours of the deleted state are updated, leading to an added modeling error which is $\leq c_{\mathrm{sdel}}(k)$.

(b) The costs $c_{\mathrm{smerge}}(k)$ are calculated by computing Euclidean distance between state k and all the other state vectors, testing to merge the state k with M closest states,[1] one at the time, and by comparing the resulting error with that before merging. After finding the best merging partner for the state k, the corresponding cost is stored in $c_{\mathrm{smerge}}(k)$ and the partner in $p_{\mathrm{smerge}}(k)$.

(c) The costs $c_{\mathrm{ndel}}(n)$ are approximated by testing to delete the node n and by computing the error in (3) *without* performing any updates for the state vectors. This gives an upper bound for the cost (cf. (a) above).

(d) The costs $c_{\mathrm{nmove}}(n)$ are computed by testing to move the node n one step to both directions and by computing the resulting modeling error without any state updates. This gives an upper bound for the cost (cf. (a) above).

It should be noted that the operations (c) and (d) are performed only in the case that they *reduce* the modeling error. If this condition is not satisfied, the smallest value in the two vectors $c_{\mathrm{sdel}}(k)$ and $c_{\mathrm{smerge}}(k)$ is searched for, and the corresponding atomic operation is performed.

3.4 Computationally efficient implementation

The computational efficiency of the above-described algorithm is already practically applicable, but the method becomes quite slow when the input data consists of the order of thousands of data vectors. In this subsection, we describe a mechanism which reduces the time-complexity of the algorithm to $O(\mathrm{T}\log\mathrm{T})$. That is, the computation time as a function of the input data sequence length T is proportional to $\mathrm{T}\log\mathrm{T}$ when all other factors are kept fixed.

The basic idea of the computational improvement is to divide the input data sequence into temporally consecutive segments called *groups*. In the beginning, the number of the groups, G, is selected so that G is a power of two and each group g contains more than $\mathrm{K}_{\mathrm{target}}$ but no more than $2 \cdot \mathrm{K}_{\mathrm{target}}$ data vectors. The above-described iterative algorithm is then applied *within each group* so that merge partners have to come from within the same group and search for the optimal atomic operation is limited to the states and nodes within the group (State or node deletion at a group boundary still causes state updates in the neighbouring group as usual.) When all the groups have been processed so that they contain exactly $\mathrm{K}_{\mathrm{target}}$ states, pairs of neighbouring groups (0 and 1, 2 and 3, etc.) are joined to form temporally longer groups with exactly $2 \cdot \mathrm{K}_{\mathrm{target}}$ states. These are then processed until $K = \mathrm{K}_{\mathrm{target}}$ is again reached. The process is continued until all the data vectors are joined into a single group and it is processed to contain exactly $\mathrm{K}_{\mathrm{target}}$ states.

The above procedure significantly improves the computational efficiency of the algorithm and has only a negligible effect on the the resulting modeling error. An intuitive explanation for the computational improvement is that the number of states within groups g is always within the limits $\mathrm{K}_{\mathrm{target}}$ and $2 \cdot \mathrm{K}_{\mathrm{target}}$. As a consequence, the number of unknowns in (8) is limited.

One more constraint has to be added to ensure the $O(\mathrm{T}\log\mathrm{T})$ complexity: the number of *delete node* and *move node* operations within each group has to be limited to $\mathrm{K}_{\mathrm{target}}$ during the life-time of the group. After reaching the limit, these operations are disabled. This has some minor effect on the accuracy of the resulting model.

The complexity of the method with respect to all the three important parameters is $O(\mathrm{T}\log\mathrm{T}\mathrm{K}^2\mathrm{D} + \mathrm{K}^3)$. However, since K is always less than or equal to T, only the first term is of practical importance. The complexity of the matrix inversion in (8) alone can be $O(\mathrm{K}^3)$. However, this worst-case (of the order K unknowns) can occur only T/K times during the runtime of the algorithm.

An intuitive explanation for the fact that the grouping mechanism does not significantly affect the modeling *accuracy* is that, since the complete data set has to be modeled with a total of $\mathrm{K}_{\mathrm{target}}$ states, this amount of states is enough for modeling each subset of the data. This observation was practically validated in simulations, as described in Sec. 4.

[1] $\mathrm{M} = 5$ is a free parameter of the algorithm.

4. SIMULATION RESULTS

Simulation experiments with musical instrument samples were carried out to validate the proposed method. Acoustic material consisted of samples from the McGill University Master Samples collection and of independent recordings for the acoustic guitar. There were altogether 32 different musical instruments, comprising brass and reed instruments, strings, flutes, mallet percussion instruments, and the piano. Different playing techniques were included where applicable (e.g. plucked and bowed violin). The total number of samples was 1666, with an average duration of 2.7 seconds (variance 3.2s). Each sample represented an individual musical note.

Feature vectors $\mathbf{x}(i)$ were extracted from the acoustic data as follows. First, discrete Fourier transforms were calculated in successive 23 ms time frames which were Hanning-windowed and overlapped 50%. In each frame, fifteen triangular-response bandpass filters were simulated that were uniformly distributed on the Mel frequency scale between 20 Hz and 10 kHz. The power of the signal at the subbands was calculated and subjected to a logarithmic compression to arrive at a perceptually meaningful representation of the spectrum. The compressed subband levels in time frame i were stored to the data vector $\mathbf{x}(i)$.

4.1 Reference method: k-means clustering

One important goal of this paper was to investigate if the described interpolating state model can achieve a better modeling accuracy than the conventional "vector quantization" approach (see Introduction) with the same model order. Among the latter approaches, we chose k-means clustering algorithm to act as a point of comparison for the proposed algorithm. K-means clustering partitions input data vectors into K clusters so as to minimize the sum of squared distances between the input data points and their corresponding cluster centroids $_k$: $e_{\text{kmeans}} = {}^{\text{K}}_{k=1} \; {}_{i \in S_k} |\mathbf{x}(i) - {}_k|^2$. The cluster centroids are then used in place of the input data vectors to approximate them. Here we used the implementation of the k-means clustering algorithm in the Matlab Statistical Toolbox.

Computational complexity of the k-means algorithm is $O(\text{TKDI})$, where T is the input data sequence length, K is the number of clusters, D is the dimensionality of the data, and I is the number of iterations performed during the clustering. In practice, the number of iterations required is directly proportional to T [6].

4.2 Results

The left panel of Fig. 4 shows the modeling error of the proposed method and that of k-means clustering as a function of the model order K. In the figure, the sum-of-squared-error criterion is used for both models and these are normalized by the sum of the squared input data. The absolute numerical range of the error is not very informative since the input data do not have a zero mean.

If not otherwise mentioned, all the results for the interpolating state model are computed using the computationally efficient version of the algorithm as outlined in Section 3.4.

The right panel of Fig. 4 shows the ratio of the modeling error of the proposed method to that of k-means clustering as a function of the model order K. The ratio is around 0.5 for all model orders, meaning that the proposed model leads to approximately twice smaller amount of modeling error than k-means clustering.

Figure 5 shows the ratio of the error of the proposed method to that of k-means clustering for individual instruments. The proposed method is particularly advantageous for freely decaying (plucked, struck) sounds and, on the other hand, sounds that exhibit only little temporal variation are well modeled by both methods.

Finally, the accuracy of the computationally efficient version of the proposed algorithm was compared to the baseline version described in Sections 3.2–3.3. It turned out that, on the average, the less efficient version of the method further reduces the amount of error by a modest factor of about 0.8. For different instruments, the factor varies between 0.7 and 0.9. However, the baseline method is substantially slower for large datasets and thus not very practical.

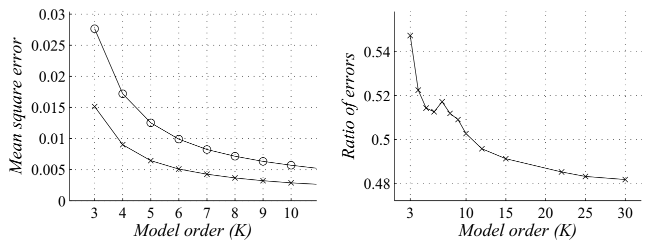

Figure 4: Left: Modeling error of the proposed interpolating state model (crosses) and that of the k-means clustering (circles). Right: ratio of the two errors as a function of the model order.

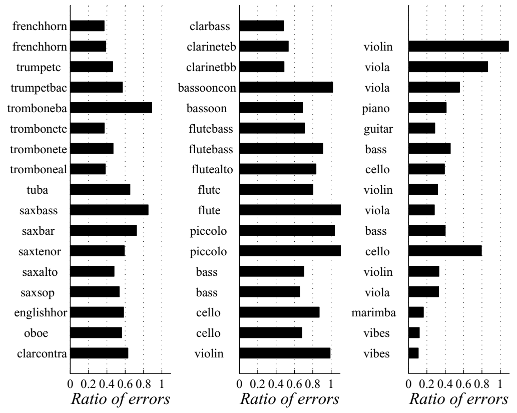

Figure 5: Ratio of the modeling error of the proposed method to that of k-means clustering for different musical instruments (K = 5).

5. CONCLUSIONS AND FUTURE WORK

A computationally efficient algorithm was presented for modeling musical sounds with an interpolating state model. In addition to proposing the estimation algorithm, one of the main interests of this this paper was to investigate if an interpolating state model leads to substantially better modeling accuracy than the conventional vector-quantization oriented (clustering) approaches. The simulation results showed that the proposed model approximately halves the modeling error of the k-means clustering method, thus motivating further development of algorithms for this model class. Future work will focus on statistical modeling of the parameter distributions and on using the model for sound source recognition.

REFERENCES

[1] M. Ostendorf, V. Digalakis, and O. A. Kimball. From HMMs to segment models: A unified view of stochastic modeling for speech reconition. *IEEE Trans. on Speech and Audio Processing*, 4:360–378, 1996.

[2] L. Deng, P. Kenny, Lennig M., and P. Mermelstein. Modeling acoustic transitions in speech by state-interpolation hidden Markov models. *IEEE Trans. on Signal Processing*, 40(2):265–271, 1992.

[3] A-V. I. Rosti. *Linear Gaussian models for speech recognition*. PhD thesis, Cambridge University, 2004.

[4] D. X. Sun. Statistical modeling of co-articulation in continuous speech based on data driven interpolation. In *Proc. IEEE Int'nal Conf. on Acoust., Speech and Signal Processing*, 1997.

[5] S. M. Kay. *Fundamentals of statistical signal processing: estimation theory*. Prentice-Hall, New Jersey, 1993.

[6] I. Davidson and A. Satyanarayana. Speeding up k-means clustering by bootstrap averaging. In *Proc. IEEE Data Mining Workshop on Clustering Large Data Sets*, Florida, 2003.