Representing Musical Sounds With an Interpolating State Model

Anssi Klapuri, *Member, IEEE*, and Tuomas Virtanen, *Member, IEEE*

***Abstract*—A computationally efficient algorithm is proposed for modeling and representing time-varying musical sounds. The aim is to encode individual sounds and not the statistical properties of several sounds representing a certain class. 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 proposed interpolating state model is generic and can be used to represent any multidimensional data sequence. In this paper, it is applied to represent musical instrument sounds in a compact and accurate form. Simulation experiments were carried out which show that the proposed method clearly outperforms the conventional vector quantization approach where the acoustic feature data is k-means clustered and the feature vectors are replaced by the corresponding cluster centroids. The computational complexity of the proposed algorithm as a function of the input sequence length T is $O(T \log T)$.**

***Index Terms*—Acoustic signal processing, audio coding, discrete cosine transforms (DCTs), interpolation, vector quantization.**

I. Introduction

This paper proposes an interpolating state model and an algorithm for representing and coding time-varying musical sounds. In particular, we focus on modeling individual sounds, not the statistical properties of several sounds representing a certain class. This has applications in structured audio coding, sound synthesis, and music content analysis.

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. Such a model is, for example, the hidden Markov model (HMM), where a hidden state variable and state-conditional observation distributions determine the acoustic features emitted at each time instant [1]. The Gaussian mixture model (GMM), too, can be viewed as a state model, although no temporal continuity constraints are imposed [2]. The third and most straightforward example of this model class is the k-means clustering algorithm that assigns feature vectors into K subsets, and the cluster centroids can then be used as vector-quantized values in place of the original data [3]. Despite their shortcomings, these models and especially the HMM are widely used in classifying musical sounds [4] and to encode time series of feature vectors [5], [6].

Manuscript received January 10, 2009; revised November 24, 2009. Current version published February 10, 2010. This work was supported by the Academy of Finland under Project 5213462 (Finnish centre of Excellence program 2006–2011). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Bertrand David.

A. Klapuri was with the Department of Signal Processing, Tampere University of Technology, FI-33720 Tampere, Finland. He is now with the Department of Electronic Engineering, Queen Mary, University of London, London E1 4NS, U.K. (e-mail: anssi.klapuri@tut.fi).

T. Virtanen with the Department of Signal Processing, Tampere University of Technology, FI-33720 Tampere, Finland (e-mail: tuomas.virtanen@tut.fi).

Digital Object Identifier 10.1109/TASL.2010.2040781

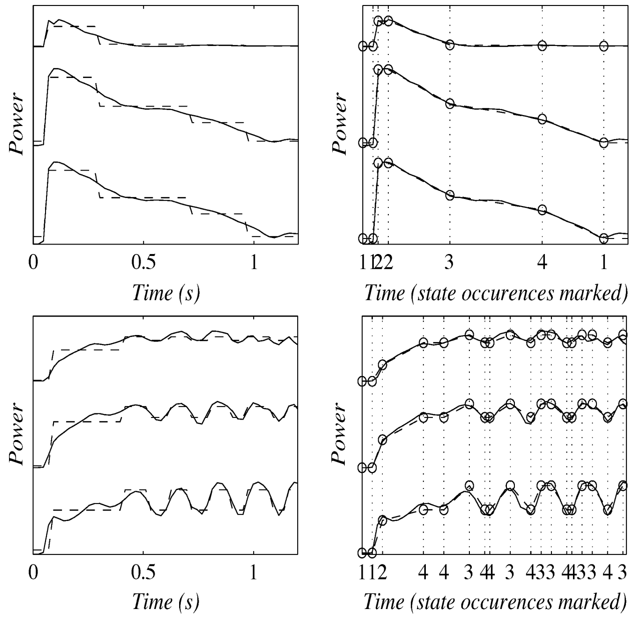

Fig. 1. Upper-left panel shows the power envelopes of a piano sound at three subbands (solid line) modeled with k-means clustering (dashed line). Upper-right panel shows the same envelopes modeled with an interpolating state model (dashed line connecting the circles). The vertical lines indicate state occurrences. Lower panels show a similar example for a violin sound.

Contrary to the above models, most musical sounds can be represented very efficiently by *interpolating* between spectra that are taken from appropriate temporal positions of the input signal. Several examples of this can be found in sound synthesis [7, p. 319–329]. As an example, consider the power envelopes of a piano sound at three subbands as illustrated in the upper 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. On the contrary, the proposed interpolating state model, illustrated in the upper right panel, achieves much better fit to the data. The amount of stored model parameters is the same as in the left panel (four different states). The occurrence times of the four states 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 lower panels show a similar example, where vector quantization and the interpolating state model are used to model a violin sound.

The limitations of, e.g., HMMs are well known in speech processing. Musical sounds are generally more slowly varying than speech signals and therefore the limitations of the conventional models are even earlier encountered. In speech recognition and speech synthesis, the most common way of addressing the dynamic aspects of speech is to include time-differentials of static features in the feature vector [8], [9]. More advanced techniques include triphone models, segment models, and trajectory models. In triphone models, the context-dependent nature of observations is taken into account by training a three-state HMM for each phoneme/context combination [10], [11]. In segment models, the unit being modeled consists of several consecutive frames, and the Markov assumption is made only between the segments [12]. Training algorithms for the distributions within segments have been presented for example in [13]. Trajectory models, in turn, involve time-varying parameters. For example, Sim and Gales proposed a model where the trajectory is a function of the acoustic observations and a set of centroids in the feature space [14]. Also, the use of the dynamic features can be formulated as a trajectory model [9]. Other modeling alternatives include switching linear dynamical systems which involve a continuous latent state variable [15], and buried HMMs which train a Bayesian network for a window of observations [16]. The data-driven interpolation model of Sun [17] is related to the model proposed here, but the author did not present an algorithm for estimating the state vectors (anchor points) of the model.

When it comes to representing individual musical sounds in a coding sense, a drawback of the segment models and trajectory models is that they are better suited for modeling the statistical properties of (speech) signals and they increase the model complexity, i.e., the amount of stored parameter data. Triphone models, in turn, are not always well matched to music signals, since the concept of a context is not equally well-defined in music, and musical sounds do not consist of enumerable phoneme-like units.

The proposed interpolating state model bears some resemblance to *splines*, functions that are defined piecewise by polynomials and are typically used for interpolating between given data points [18]. The proposed model can be viewed as a first degree (linear) spline, but a crucial difference here is that the anchor points are learned from the data and not given in advance. Linear regression represents another class of techniques related to the present method [19]. The main difference here is that the basis functions are learned from data and each data point is modeled as a weighted sum of only two basis functions, not all. Connections to sparse coding and other variants of linear models will be discussed in Section II.

In the following section, the interpolating state model illustrated in Fig. 1 will be defined more exactly. In Section III, a computationally efficient algorithm is proposed for estimating the model parameters. Whereas globally optimal parameters for this model are computationally intractable, the method finds a suboptimal solution. The time-complexity of the algorithm is $O(T\log(T)K^2D)$, 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. Some parts of this work have been previously published in the conference paper [20].

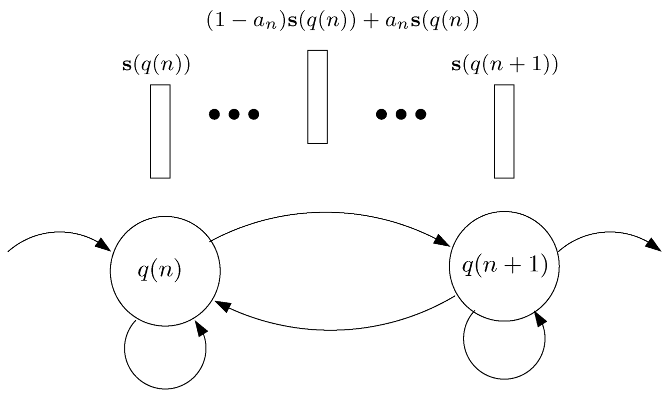

Fig. 2. Data vectors generated by the interpolating state model between two states $q(n)$ and $q(n+1)$. See text for details.

The proposed model is generic and can be used to approximate any multidimensional data sequences that can be effectively modeled by piece-wise linear regression between K anchor points (states) in the target space. In this paper, we apply the model to represent musical instruments in a compact and accurate form. An analysis–synthesis procedure to this end is described in Section IV. In addition to sound synthesis and coding, the proposed method has potential as intermediate data representation in acoustic signal analysis in general (see, for example, [21], where interpolating HMM was used for musical instrument recognition).

Simulation experiments were carried out which show that the proposed method clearly outperforms the conventional "vector quantization" approach where acoustic feature vectors are k-means clustered and the feature vectors are then replaced by the corresponding cluster centroids. The results are presented in Section V. Section VI summarizes the conclusions.

II. Model Formulation

The input data to be modeled is a sequence of feature vectors $\mathbf{x}(\tau)$ which have been extracted in successive time frames $\tau = 0, \ldots, T-1$ of an acoustic signal. The dimensionality of the feature vectors is denoted by D and a matrix $\mathbf{X} = [\mathbf{x}(0), \ldots, \mathbf{x}(T-1)]^\mathsf{T}$ of size $(T \times D)$ 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 $K \ll T$ states k, each of which corresponds to a distinct D-dimensional state vector $\mathbf{s}(k)$. For convenience, the state vectors are collected into a matrix $\mathbf{S} = [\mathbf{s}(0), \ldots, \mathbf{s}(K-1)]^\mathsf{T}$.

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. Fig. 2 illustrates 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 are called *nodes* where, momentarily, 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. The violin sound in Fig. 1, for example, loops around the states three and four during the sustained vibrato portion that extends over the latter half of the signal. Allowing the transitions back to the previous states greatly reduces the number of states needed for modeling.

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

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

$$\hat{\mathbf{x}}(\tau) = (1 - a_n(\tau))\mathbf{s}(q(n)) + a_n(\tau)\mathbf{s}(q(n+1)) \quad (1)$$

where

$$a_n(\tau) = \frac{\tau - t(n)}{t(n+1) - t(n)}. \quad (2)$$

It should be noted that the model is completely deterministic, contrary to, e.g., hidden Markov models or stochastic segment models [12]. No probability distributions are applied.

In a matrix form, the model (1)–(2) can be written as $\hat{\mathbf{X}} = \mathbf{AS}$, where the matrix $\mathbf{A}$ of size $(T \times K)$ contains the interpolation weights $a_n(\tau)$ and is restricted to have only two nonzero entries at each row. A similar linear representation is used in many machine learning algorithms, such as principal component analysis [22], independent component analysis [23], sparse coding [24], and non-negative matrix factorization [25]. However, the further modeling assumptions in these differ from each other. For example in sparse coding, $\mathbf{A}$ is assumed to be a sparse matrix, whereas in non-negative matrix factorization the entries in $\mathbf{A}$ and $\mathbf{S}$ are assumed to be non-negative. Finding a sparse representation given the model is a difficult optimization problem for which several greedy algorithms have been proposed, such as matching pursuit [26] and basis pursuit [27]. A generic algorithm for learning dictionary $\mathbf{S}$ for sparse representations has been proposed in [28]. In our method, the weights in $\mathbf{A}$ are sparse and restricted to have certain interpolation properties; therefore, we need a specific algorithm for the parameter estimation.

III. Algorithm for Parameter Estimation

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

$$e = \sum_{\tau=0}^{T-1} \sum_{d=0}^{D-1} (\mathbf{X}_{\tau,d} - \hat{\mathbf{X}}_{\tau,d})^2. \quad (3)$$

A basic version of the parameter estimation algorithm is first described in Sections III-A–III-D and then the efficient version in Section III-E.

A. Least-Squares 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 [29, Chap. 8].

The solution can be formulated in the present context as follows. Let $\mathbf{A}$ be a $(T \times K)$ matrix of weights where each row τ, $t(n) \leq \tau \leq t(n+1)$, contains only two nonzero values

$$\mathbf{A}_{\tau,k} = \begin{cases} 1 - a_n(\tau), & \text{for column } k = q(n) \\ a_n(\tau), & \text{for column } k = q(n+1) \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

Note that each time index τ is between two states or exactly at one state. As a consequence, the matrix $\mathbf{A}$ is sparse, with only one or two nonzero values at each row.

Then, the least squares solution for the state vectors is [29]:

$$\mathbf{S} = (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\mathbf{X}. \quad (5)$$

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 $\mathcal{K}$ the set of states that are being updated. First, all columns k for which $k \notin \mathcal{K}$ are removed from $\mathbf{A}$. Then a $(T \times D)$ matrix $\mathbf{Y}$ is defined where each row τ, $t(n) \leq \tau \leq t(n+1)$ is defined as

$$\mathbf{y}(\tau)^\mathsf{T} = \mathbf{x}(\tau)^\mathsf{T} - I(n)(1 - a_n(\tau))\mathbf{s}(q(n))^\mathsf{T} - I(n+1)a_n(\tau)\mathbf{s}(q(n+1))^\mathsf{T} \quad (6)$$

where the indicator function $I(n)$ is defined counter-intuitively as

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

The matrix $\mathbf{Y}$ represents a version of the input data sequence where the effect of the fixed states is compensated for.

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

$$\tilde{\mathbf{S}} = (\tilde{\mathbf{A}}^\mathsf{T}\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^\mathsf{T}\tilde{\mathbf{Y}} \quad (8)$$

where $\tilde{\mathbf{S}}$ contains only the states $k \in \mathcal{K}$. Note that the size of the square matrix $(\tilde{\mathbf{A}}^\mathsf{T}\tilde{\mathbf{A}})$ depends on the number of unknowns

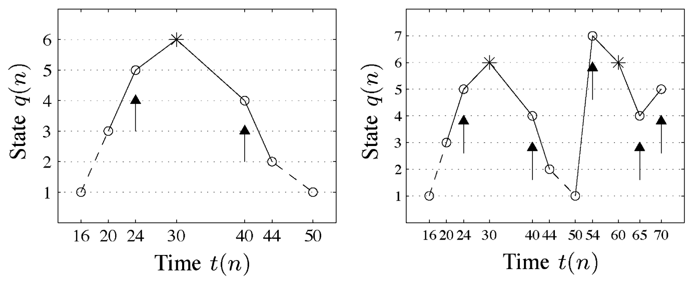

Fig. 3. Left panel shows a simple state deletion case. The node of the deleted state is marked with an asterisk and the nodes of the updated states are marked with arrows. Right panel shows a more general case.

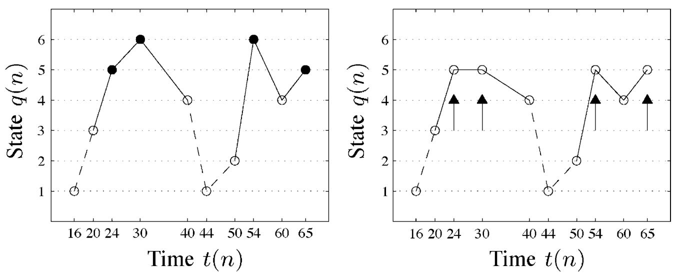

Fig. 4. Illustration of state merging. Left and right panels shows the situation before and after the merging, respectively. Filled circles indicate the nodes of the merged states and the arrows indicate the nodes of the updated state.

(size of the set $\mathcal{K}$) and cannot be singular due to the way the matrix is composed.

B. 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 $[0, T-1]$. As testing all these configurations for error minimization is not feasible, an iterative algorithm is proposed which finds a suboptimal solution. The processing steps are as follows.

Initialization: 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 $K' = T$ states and $N' = T$ nodes with the state vectors $\mathbf{s}(k) = \mathbf{x}(k)$ for $k = 0, \ldots, K-1$, and the node sequence $t(n) = n$, $q(n) = n$ for $n = 0, \ldots, N-1$.

Iteration: 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.

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

1) *Delete state*. Delete a state and all nodes associated with it. All states which have nodes as immediate neighbors of a deleted node are updated [recomputed using (8)] so as to minimize the error in the neighborhood 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 neighbors 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 neighbors 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 $K' = K$ 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.

2) *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.

Fig. 4 illustrates the merge operation. The nodes associated with the two states being merged (five and six) are marked with black. After merging the states, the state five is updated so that the error is minimized over the solid line in the figure.

The above atomic operations are performed one at a time to reduce the model order until the desired order K 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 which can have arbitrary state transitions. In practical experiments, however, two additional atomic operations turned out to be useful. These make a substantial improvement in the actual modeling result.

3) *Delete node*. After several merge operations, 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 neighbors 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.

4) *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.

C. 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 $c_{\text{sdel}}(k)$ and $c_{\text{smerge}}(k)$. Along with the latter, a vector $p_{\text{smerge}}(k)$ is needed where the optimal merge partner is stored for each k. Node deletion and node moving costs are stored in the vectors $c_{\text{ndel}}(n)$ and $c_{\text{nmove}}(n)$ for $n = 1, \ldots, N'$. Along with the latter, a vector

$d_{\text{nmove}}(n)$ 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 $c_{\text{sdel}}(k)$ and $c_{\text{smerge}}(k)$ have to be updated for the states that have associated nodes as immediate neighbors to the nodes of the states that were *updated* during the previous atomic operation. The vectors $c_{\text{ndel}}(n)$ and $c_{\text{nmove}}(n)$ have to be updated for the nodes that occur as such neighbors.

The costs are computed as follows.

1) The costs $c_{\text{sdel}}(k)$ 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 neighbors of the deleted state are updated, leading to an added modeling error which is $\leq c_{\text{sdel}}(k)$.
2) The costs $c_{\text{smerge}}(k)$ are calculated by computing Euclidean distance between state k and all the other state vectors, testing to merge the state k with μ closest states,[1] one at the time, and by comparing the resulting error (3) with that before merging. After finding the best merging partner for the state k, the corresponding cost is stored in $c_{\text{smerge}}(k)$ and the partner in $p_{\text{smerge}}(k)$.
3) The costs $c_{\text{ndel}}(n)$ 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. 1) above).
4) The costs $c_{\text{nmove}}(n)$ are computed by testing to move the node n one step to both directions and by computing the resulting modeling error (3) without any state updates. This gives an upper bound for the cost (cf. 1) above).

At each step of the above iterative process, it is important to choose the atomic operation that leads to the smallest amount of added modeling error. This is done by keeping track of the above costs. *Delete node* and *move node* operations are performed only in the case that they *reduce* the modeling error.[2] If this condition is not satisfied, the smallest value in the two vectors $c_{\text{sdel}}(k)$ and $c_{\text{smerge}}(k)$ is searched for, and the corresponding atomic operation is performed.

D. Weighted Least Squares

In some applications, it is desirable to weight the modeling error in such a way that some data vectors are more important than others and therefore modeled more accurately. For example in perceptual audio coding, quiet parts of the time–frequency plane have to be modeled more accurately than the loud parts where errors are more effectively masked by the signal itself [30]. Such a weighting can be easily incorporated in the proposed method. Let us define a weight vector $\mathbf{w} = [w(0), \ldots, w(T-1)$ which determines the weight of the feature vectors $\mathbf{x}(0), \ldots, \mathbf{x}(T-1)$ so that the desired modeling error of the data vectors should be inversely proportional to the weights.

This objective is achieved by weighted least squares which requires only a few modifications to the above described algorithm. First, a weighted error function is defined as

$$e' = \sum_{\tau=0}^{T-1} \sum_{d=0}^{D-1} w(\tau)(\mathbf{X}_{\tau,d} - \hat{\mathbf{X}}_{\tau,d})^2. \tag{9}$$

To minimize the above error, weights are included in (8) to obtain

$$\tilde{\mathbf{S}} = (\tilde{\mathbf{A}}^{\mathsf{T}} \mathbf{W} \tilde{\mathbf{A}})^{-1} \tilde{\mathbf{A}}^{\mathsf{T}} \mathbf{W} \tilde{\mathbf{Y}} \tag{10}$$

where $\mathbf{W}$ is a square matrix containing the weights $w(\tau)$ on its diagonal. In practice, the weighting can be implemented by multiplying each row τ of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{Y}}$ by $\sqrt{w(\tau)}$ prior to the application of (8).

In calculating the costs $c_{\text{sdel}}(k), c_{\text{smerge}}(k)$, $c_{\text{ndel}}(n)$, and $c_{\text{nmove}}(n)$, the weighted error (9) is used instead of (3) to predict the cost of different operations.

In all other respects, the algorithm utilizing the weights remains the same as the one described in the preceding sections.

E. 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 $O(T \log T)$. That is, the computation time as a function of the input data sequence length T is proportional to $T \log T$ 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 K but no more than $2 \cdot K$ 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 neighboring group as usual.) When all the groups have been processed so that they contain exactly K states, pairs of neighboring groups (0 and 1; 2 and 3; etc.) are joined to form temporally longer groups with exactly $2 \cdot K$ states. These are then processed until $K' = K$ 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 K states.

The above procedure significantly improves the computational efficiency of the algorithm and has only a negligible effect on the resulting modeling error. An intuitive explanation for the computational improvement is that the number of states

[1]$\mu = 5$ is a free parameter of the algorithm.

[2]This is because model complexity is attributed only to states (desiring K states) and not for nodes. It is possible to attribute model complexity both to states and nodes and, at each step, to perform the atomic operation which yields the smallest ratio of cost to model complexity decrease.

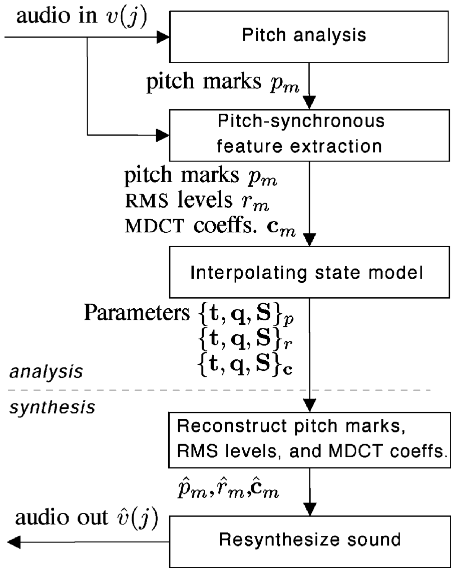
audio in $v(j)$
Pitch analysis
pitch marks p_m
Pitch-synchronous feature extraction
pitch marks p_m
RMS levels r_m
MDCT coeffs. $\mathbf{c}_m$
Interpolating state model
Parameters $\{\mathbf{t},\mathbf{q},\mathbf{S}\}_p$ $\{\mathbf{t},\mathbf{q},\mathbf{S}\}_r$ $\{\mathbf{t},\mathbf{q},\mathbf{S}\}_\mathbf{c}$
analysis
synthesis
Reconstruct pitch marks, RMS levels, and MDCT coeffs.
$\hat{p}_m, \hat{r}_m, \hat{\mathbf{c}}_m$
audio out $\hat{v}(j)$
Resynthesize sound

Fig. 5. Overview of the musical sound coding procedure.

within groups g is always within the limits K and $2 \cdot K$. As a consequence, the number of unknowns in (8) is limited.

One more constraint has to be added to ensure the $O(T \log T)$ complexity: the number of *delete node* and *move node* operations within each group has to be limited to K during the lifetime 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 $O(T \log(T) K^2 D)$. Detailed analysis of the complexity is presented in the Appendix.

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 K states, this amount of states is enough for modeling each subset of the data. This was experimentally verified in simulations.

IV. Application to the Representation of Musical Sounds

The proposed method was applied to represent musical instrument sounds in a compact and accurate form. Fig. 5 shows an overview of the analysis–synthesis procedure employed. It is based on pitch-synchronous processing and therefore involves the limitation that only one pitched sound is assumed to be active at each time in the input audio. In our experiments, isolated musical instrument sounds and sound sequences were used.

The first block in Fig. 5, pitch analysis, produces pitch marks which are time stamps located one pitch period apart from each other and aim to keep in synchrony with the time-varying pitch. This is followed by pitch-synchronous feature extraction, where the root mean square (rms) level and waveshape of the signal is measured in frames that extend over each two successive pitch periods. The waveshape is represented using modified discrete cosine transform (MDCT) described later. The proposed interpolating state model is then used to encode the time-varying pitch, rms level, and MDCT coefficients. At the synthesis stage, the three acoustic feature sequences are reconstructed from the model and used to resynthesize an approximation of the input audio signal. Accuracy of the method is measured by comparing the reconstructed signal $\hat{v}(j)$ with the original one, $v(j)$. These steps are now described in more detail.

Preliminary pitch analysis is carried out in 93-ms time frames which overlap 75%. We employed the method described in [31] which makes no assumptions about the sound source and can handle the wide pitch range of musical instruments. Based on the pitch estimates, pitch marks are generated that indicate individual pitch periods. While doing this, small corrections to the preliminary pitch estimates are made, in order to minimizing the mean-square error between the waveshapes of successive pitch periods. The pitch marks form a sequence $p_m, m = 1, \ldots, M$, where p_m and p_{m+1} delimit the mth pitch period in $v(j)$. Important for the quality of especially high-pitched sounds is that the pitch marks are actually real-valued (sub-sample precision). Further details of the pitch mark generation are beyond the scope of this paper.

For computational efficiency reasons explained later, each pitch period (interval between p_m and p_{m+1}) is uniformly resampled to consist of exactly L samples. The resampling is implemented simply by a linear interpolation of $v(j)$, since this is computationally efficient and the resulting nonoptimality in sound quality is negligible compared to the other modeling steps. The value of L was chosen according to the maximum period between successive pitch marks p_m. In the following, we use $v_m(\ell), \ell = 0, \ldots, 2L-1$ to denote one pitch-synchronous frame of the input signal. It consists of the resampled waveform covering two periods of the input signal between p_m and p_{m+2}.

The rms level of the signal in frame m is calculated as

$$r_m = \left[\frac{1}{2L} \sum_{\ell=0}^{2L-1} h(\ell) v_m(\ell)^2 \right]^{0.5} \tag{11}$$

where the window function $h(\ell) = \sin[\pi(\ell+1)/(L+1)]$ equals the square root of the Hann window.[3]

The waveshape (timbre) in frame m can be compactly represented using the MDCT. It transforms $2L$ real numbers $v_m(0), \ldots, v_m(2L-1)$, into L real numbers $c_m(0), \ldots, c_m(L-1)$ according to

$$c_m(d) = \frac{1}{r_m} \sum_{\ell=0}^{2L-1} h(\ell) v_m(\ell) \times \cos\left[\frac{\pi}{L} \left(\ell + \frac{1}{2} + \frac{L}{2} \right) \left(d + \frac{1}{2} \right) \right]. \tag{12}$$

Note the normalization by r_m above, which allows modeling the waveshape and the level separately. Often the waveshape remains the same although the level of a sound changes. Similarly to the fast Fourier transform, MDCT can be implemented in $O(L \log L)$ time. Due to the resampling, all the transform frames are of length $2L$ and therefore the cosine basis needs to be computed only once.

[3]Square root is applied because the same window function is employed again at the resynthesis stage. This enables perfect reconstruction when the modeling step is bypassed (except for small errors due to the resampling).

MDCT enables perfect reconstruction despite the $1:2$ data reduction in individual frames. This is possible because successive frames overlap 50% and the errors introduced in one frame are canceled by the errors introduced in the neighboring frame [32]. Another nice feature of the MDCT is energy compaction, meaning that the first few coefficients usually capture most of the energy. In practice, $D=50$ coefficients were found sufficient to encode the waveshape of musical sounds. For convenience, we use $\mathbf{c}_m=[c_m(0),\ldots,c_m(D-1)]^\mathsf{T}$ to denote the first D transform coefficients in frame m.

The three acoustic feature sequences, p_m, r_m, and $\mathbf{c}_m$, are each separately modeled using the proposed interpolating state model. In principle, the features in each frame could also be catenated into a single feature vector and the modeling be done jointly for it. However, there is no guarantee that the three features would correlate, and furthermore, from a perceptual point of view, it is more satisfying to model the three perceptual attributes of sound separately and to analyze the importance of each on the sound quality. This bears resemblance to source-filter modeling, where pitch and spectral shape are separately modeled, too [33]. Here, rms levels and MDCT coefficients are modeled directly, but pitch marks have to be preprocessed before applying the model. The most straightforward way of doing this would be to model the pitch period lengths $p_1-p_0,\ldots,p_M-p_{M-1}$, instead of the absolute time stamps p_m. Although this produces perceptually good results, a problem in doing this is that modeling errors cause the resynthesized signal to go out-of-sync compared to the input sound, which complicates the comparison of $\hat{v}(j)$ and $v(j)$. Therefore, when modeling individual musical sounds, the pitch marks are preprocessed as follows. A second-order polynomial is fitted to the sequence $p_m, m=0,\ldots,M-1$, the resulting approximation is subtracted from p_m to obtain $\tilde{p}_m$, and $\tilde{p}_m$ is then subjected to the modeling. The coefficients of the polynomial (that is, three real numbers) are stored along with the model parameters to enable reconstruction of p_m later on.[4]

Reconstruction of $\hat{p}_m, \hat{r}_m$, and $\hat{\mathbf{c}}_m$ from the model parameters is straightforward using (1)–(2). For $\hat{p}_m$, the above-mentioned polynomial has to be added after the reconstruction. Fig. 6 shows the three feature sequences $\tilde{p}_m, r_m$, and c_m, and the corresponding model outputs $\hat{p}_m$, $\hat{r}_m$, and $\hat{\mathbf{c}}_m$ for a flute sound with pitch 294 Hz.

Audio resynthesis using $\hat{p}_m, \hat{r}_m$, and $\hat{\mathbf{c}}_m$ is straightforward. Vector $\hat{\mathbf{c}}_m$ is inverse MDCT transformed, windowed with $h(\ell)$, and resampled to obtain the number of samples indicated by pitch marks $\hat{p}_m$ and $\hat{p}_{m+1}$. The resulting frame is weighted by $\hat{r}_m$ and added to the appropriate position of the output signal $\hat{v}(j)$.

V. Simulation Experiments

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 [34]. There were altogether 31 different musical instruments, comprising brass and reed instruments, strings, flutes, mallet percussion instruments, and the piano. These introduce several different sound production mechanisms and a variety of spectra. Different playing techniques were included where applicable (for example bowed, plucked, and martele playing of violin). The total number of samples was 1624, each sample representing an individual musical note. To give an appropriate weight to the beginning transients and other dynamic features, sounds longer than one second were truncated to 1.0 s. The signals were sampled at 44.1-kHz rate and 16-bit resolution.

[4]When processing a sequence of several sounds, the differences $p_1-p_0,\ldots,p_M-p_{M-1}$ have to be modeled, since in that case the pitch does not vary around a global mean that could be represented by the polynomial.

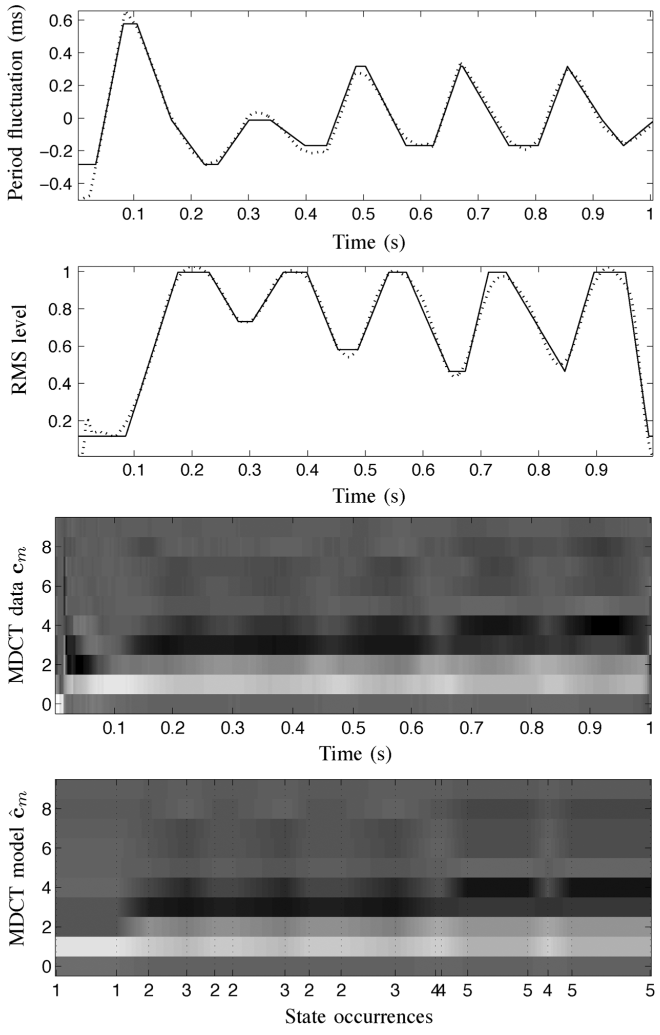

Fig. 6. Two upper panels correspond to pitch and rms level, respectively. Solid lines indicates the measured features and the dotted lines the values reconstructed from the model. The two lower panels show the measured and the reconstructed MDCT coefficients, respectively (only the first ten coefficients are shown).

A. 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 the k-means clustering algorithm [3], [35] to act as a point of comparison for the proposed algorithm.

The k-means algorithm partitions input data vectors into K clusters C_k so as to minimize the sum of squared distances between the input data points and their corresponding cluster centroids μ_k

$$e_{\mathrm{kmeans}} = \sum_{k=1}^{K} \sum_{\tau \in C_k} |\mathbf{x}(\tau) - \mu_k|^2. \tag{13}$$

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 algorithm in the Matlab Statistical Toolbox.

Weighted k-means is otherwise similar, but minimizes the error

$$e'_{\mathrm{kmeans}} = \sum_{k=1}^{K} \sum_{\tau \in C_k} w(\tau)|\mathbf{x}(\tau) - \mu_k|^2. \tag{14}$$

It is easy to see that k-means minimizes exactly the same error as the proposed method [compare the above error measures with (3) and (9)]. The weighted k-means algorithm was implemented by introducing the weights to the k-means implementation in the Matlab Statistics Toolbox.

Computational complexity of the k-means algorithm is $O(TKDI)$, 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 [3]. In practice, the number of iterations required is directly proportional to T [36] and therefore the resulting time complexity of K-means clustering is $O(T^2KD)$.

B. Results

Signal-to-noise ratio (SNR) was chosen as the measure of modeling quality because it is objective and easy to interpret. The SNR is defined as

$$\mathrm{SNR} = 10\log_{10} \frac{\sum_{\tau,d} \mathbf{X}_{\tau,d}^2}{\sum_{\tau,d}[\mathbf{X}_{\tau,d} - \hat{\mathbf{X}}_{\tau,d}]^2} \tag{15}$$

where the matrices $\mathbf{X}$ and $\hat{\mathbf{X}}$ represent the original data and the output of the model, respectively. All the results for the proposed method are computed using the efficient version of the algorithm as outlined in Section III-E.

For the sake of illustration, let us first study the modeling accuracy for a synthetic sinusoidal sweep, drawn with solid line in the left panels of Fig. 7. The dashed lines in the upper and lower left panel show the output of k-means and the proposed method, respectively, for $K = 4$. The right panel shows the SNRs for the proposed method and for k-means as a function of the model order K. The two models are shorthanded "IM" (interpolating model) and "VQ" (vector quantization), respectively. As can be seen, the proposed method outperforms k-means clearly for this one-dimensional and near-trivial data.

Fig. 8 shows results for realistic multidimensional data sequences. The data consists of the log-powers within 30 critical bands of hearing, calculated for musical instrument samples in successive 23-ms analysis frames that overlap 50%. This data was used in order to evaluate the performance of the proposed interpolating state model independently of the analysis–synthesis procedure described in Section IV. More exactly, 30

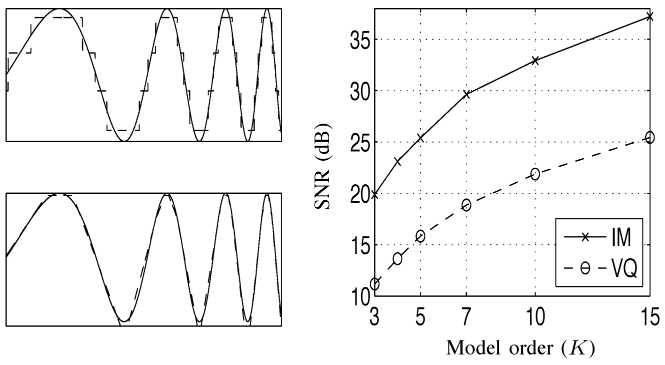

Fig. 7. Modeling a synthetic sinusoidal sweep. The left panels show the input signal (solid line) and the output of k-means (upper left) and the proposed method (lower left) with dashed line for $K = 4$. The right panel shows SNRs for the proposed method ("IM") and for k-means ("VQ") as a function of the model order K.

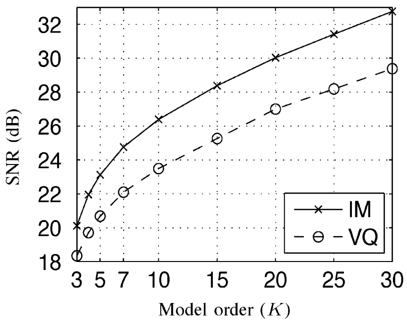

Fig. 8. Results for 30-dimensional feature sequences representing the log-powers withing critical bands of hearing. Average SNRs are shown for the proposed method ("IM") and for k-means ("VQ") as a function of the model order K.

triangular bandpass responses were uniformly distributed on the Mel-frequency scale between 40 Hz and 20 kHz, and the log-power within each band was calculated. The Mel scale is defined as $f_{\mathrm{Mel}} = 2595\log_{10}(1 + f_{\mathrm{Hz}}/700)$. This representation is well motivated from the perceptual viewpoint. The resulting 30-dimensional feature sequences were modeled and SNRs obtained for individual sounds were averaged.

As can be seen in Fig. 8, the proposed method outperforms k-means for all model orders K. The accuracy achieved by the proposed method with $K = 7$ is achieved by k-means with $K \approx 13$. For higher model orders, the difference between the proposed method and k-means is over 3 dB, meaning that the mean square error of the proposed method is only half of that of k-means.

We then turn to evaluating the proposed method from the viewpoint of audio coding using the analysis–synthesis procedure described in Section IV. Signal-to-noise ratio SNR_v for the resynthesized waveform is given by

$$\mathrm{SNR}_v = 10\log_{10} \frac{\sum_j v(j)^2}{\sum_j [v(j) - \hat{v}(j)]^2}. \tag{16}$$

To maximize SNR_v, pitch marks and MDCT data are modeled using the weighted models [see (10) and (14)], where the rms level $w(m) = r_m^2$ is used as the weight. The rms levels r_m themselves are modeled without weighting [(8) and (13)]. As an intuitive motivation for using the weighted models, imagine

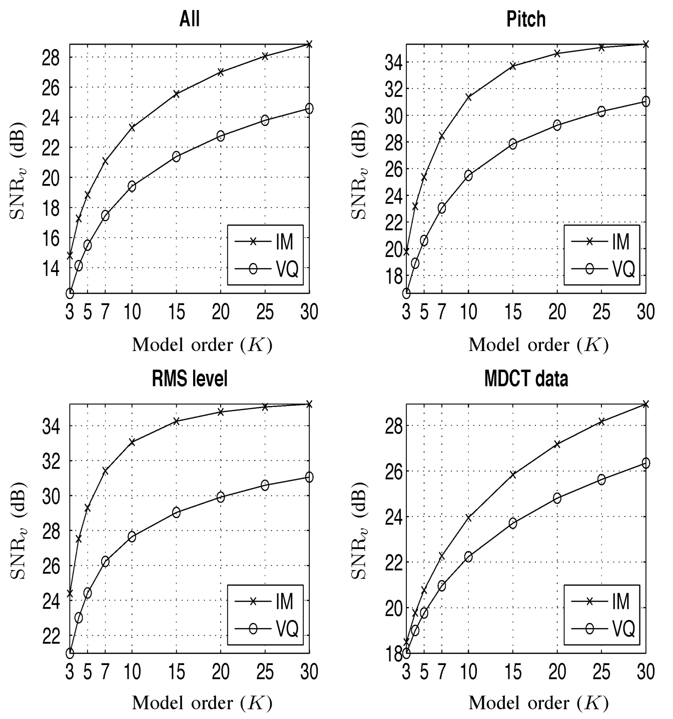

Fig. 9. Average SNRs for the proposed method ("IM") and for k-means ("VQ") as a function of the model order K. In the top left panel, all the three feature sequences were separately modeled. The remaining three panels show results for the cases where only one of the three features was modeled.

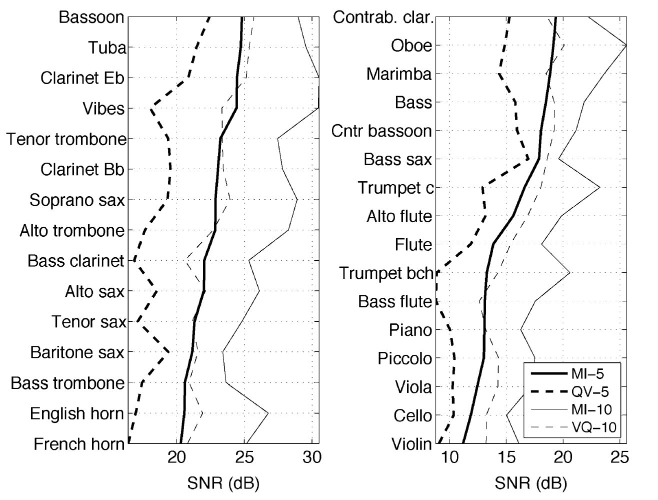

Fig. 10. Average SNRs for different instruments, when all the three features are modeled. Thick and thin solid lines indicate the proposed method ("MI") with $K = 5$ and $K = 10$, respectively. Thick and thin dashed lines indicate k-means ("VQ") with $K = 5$ and $K = 10$, respectively.

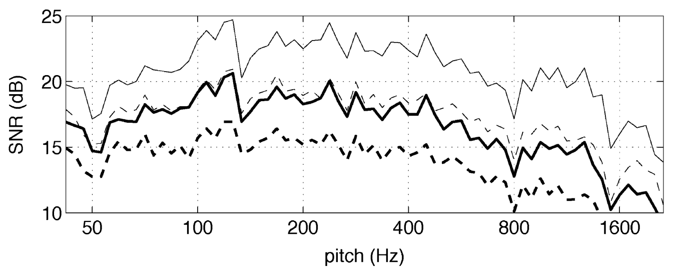

Fig. 11. Average SNRs as a function of the pitch of the modeled sounds. Thick and thin solid lines indicate the proposed method with $K = 5$ and $K = 10$, respectively. Thick and thin dashed lines indicate k-means with $K = 5$ and $K = 10$, respectively.

a situation where the target signal would contain a segment of low-level background noise. Without the weighting, the models would waste many parameters to represent the noise spectrum, not maximizing (15).

Fig. 9 shows the average SNRs for the proposed method and for k-means as a function of the model order K. The top left panel shows results in the case that all the three feature sequences are separately modeled. The remaining three panels show results in the case that only one of the three features is modeled, and for the other two, the modeling step is bypassed (equivalent to modeling them with infinite K).

In the case "All", where all the three features are modeled, the difference between the proposed method and k-means is over 3 dB for $K = 5$, and the difference between the two models increases slightly along with the model order. When modeling pitch or rms level alone, the difference between the proposed and the reference method is even larger. For MDCT data, however, neither of the two models achieves a good modeling accuracy at low model orders. Perceptually, thought, the difference between the proposed and the reference method is clear, since k-means vector quantization leads to abrupt changes in the sound spectrum which are clearly audible, whereas the proposed method naturally leads to smooth transitions between the states. When the modeling step is bypassed for all the three features, the SNR of the analysis–synthesis system is 36 dB on the average.

Fig. 10 shows the average SNRs for different musical instruments, when all the three features are modeled. The thick and thin solid lines show the results for the proposed method with $K = 5$ and $K = 10$, respectively, and the thick and thin dashed lines correspond to k-means with $K = 5$ and $K = 10$, respectively. For the sake of clarity, the instruments are sorted in descending SNR order according to the proposed method with $K = 5$. As can be seen, the proposed method with $K = 5$ achieves almost the same modeling accuracy as k-means with $K = 10$, since the corresponding to two lines (thick solid line and thin dashed line) run almost on top of each other in the middle-ground of the panels.

There is no single explaining factor for the fact that some instruments are modeled better than the others. For some instruments occupying the high pitch range, the pitch synchronization sometimes fails, making it more difficult to model the waveshape of the signals, since it appears different when shifted circularly. Piano is an interesting case, since its spectrum consists of partials that are not in perfect harmonic relationships and different partials modulate in an unconcerted manner, therefore the waveshape is constantly changing.

Fig. 11 shows the SNRs as a function of the pitch of the modeled sounds. Both models perform best at the middle pitch register. For low-pitched notes (<100 Hz), the SNR slightly decreases because the number of MDCT coefficients ($D = 50$) is no longer sufficient to represent the rich spectrum of low-pitched sounds. For high-pitched sounds (>400 Hz), the number of pitch periods M is larger and the modeled time sequences therefore longer. In some cases, the pitch marks go out-of-sync and the waveform starts to appear different.

C. Discussion

The k-means algorithm is just one way of performing vector quantization. It was chosen because it is widely used and minimizes exactly the same error measure as the proposed method. Another alternative would have been to employ the HMM and then use the state-conditional observations in place of the original feature vectors. However, HMM has temporal continuity constraints that do not help in improving the mean square error criterion. If they are omitted, HMM and k-means are basically equivalent as models.

When measuring audio quality, it is clear that the SNR is not perceptually the most accurate measure. SNR was chosen because it is easy to interpret and therefore makes the evaluations more transparent. In order to achieve perceptually more optimal coding result, a simple technique is to μ-law compress the MDCT coefficients and rms levels prior to the modeling, and then compensate for the compression in resynthesis. Since MDCT is a frequency transform, the compression results in minimizing the modeling error in the log magnitude spectrum, which allocates the error in a perceptually relevant manner. However, this makes it more difficult to interpret the results and listening tests should be arranged to evaluate the coding result.

Above, the problem was formulated as minimizing the modeling error for a given model order K. One standard way of choosing K automatically is to include a penalty term in the objective (3) to penalize excess number of anchor points [19]. A straightforward way of minimizing such an augmented cost is to first calculate the error e for different model orders and then choose K which minimizes the augmented cost.

The proposed method has not been used for audio classification so far. In principle, better modeling accuracy can lead to better recognition performance. However, since the model is completely deterministic, further statistical modeling of the parameter distributions is required to enable computing likelihoods that a certain feature sequence has been generated by a certain interpolating state model. Closely related to this is the ongoing work of Virtanen and Heittola on the interpolating HMM, which has produced promising results [21]. Furthermore, we will investigate the use of the proposed interpolating state model for the separation and recognition of sound sources in polyphonic sound mixtures in a manner similar to [37].

VI. Conclusion

A computationally efficient algorithm was described for representing musical sounds with an interpolating state model. In addition to proposing the estimation algorithm, one of the main interests of this paper was to investigate if an interpolating state model leads to substantially better modeling accuracy than the conventional vector-quantization (clustering) approaches. The simulation results show that the proposed model achieves a clearly better modeling accuracy than the k-means clustering method, thus motivating further development of algorithms for this model class.

In representing musical instrument sounds, the proposed model achieved on the average 3–4 dB better SNR than the k-means algorithm with a same model order K. Especially the pitch and rms level can be highly efficiently represented by the interpolating state model. For MDCT coefficients, neither of the models achieved a good accuracy at low model orders, but the proposed model outperformed the reference when the model order was increased.

Appendix A
Complexity Analysis

Computational complexity of the algorithm described in Section III-E is a result of several factors and a complete proof of the complexity of course requires a careful inspection of the C-language source code of the algorithm. Here we present only the main lines of the complexity analysis.

Let us first analyze the computational complexity of the individual atomic operations. In all the four operations, the time complexity is determined by the computation of (8), which consists of one matrix inverse and three matrix multiplications. Let us denote by K_{a} the number of states that are being updated and by T_{a} the number of time points (feature vectors) in the data that are affected by the updated states. Then the size of $\tilde{\mathbf{A}}$ in (8) is $(T_{\mathrm{a}} \times K_{\mathrm{a}})$ and the size of $\tilde{\mathbf{Y}}$ is $(T_{\mathrm{a}} \times D)$. Note that $\tilde{\mathbf{A}}$ is sparse due to the way it is constructed in (4) and therefore the complexity of the matrix multiplication $\tilde{\mathbf{A}}^{\mathsf{T}}\tilde{\mathbf{A}}$ is only $O(K_{\mathrm{a}}T_{\mathrm{a}})$. The complexity of the matrix inverse $(\tilde{\mathbf{A}}^{\mathsf{T}}\tilde{\mathbf{A}})^{-1}$ is $O(K_{\mathrm{a}}^3)$ and the complexities of the latter two matrix multiplications in (8) are $O(K_{\mathrm{a}}T_{\mathrm{a}})$ and $O(K_{\mathrm{a}}T_{\mathrm{a}}D)$, respectively. Omitting the insignificant terms, the overall time complexity of (8) is therefore $O(K_{\mathrm{a}}^3 + K_{\mathrm{a}}T_{\mathrm{a}}D)$.

Equation (8) has to be evaluated in all the four atomic operations, but in *merge state*, and *move node* operations, $K_{\mathrm{a}} = 1$ all the time, and in *delete node* operation, $K_{\mathrm{a}} = 3$. Therefore, the time complexity of (8) reduces to $O(T_{\mathrm{a}}D)$, which is also the overall complexity of these three operations. In *delete state* operation, K_{a} can take larger values and therefore the complexity of this operation remains $O(K_{\mathrm{a}}^3 + K_{\mathrm{a}}T_{\mathrm{a}}D)$.

After performing any of the four operations, we need to update the cost vectors for the states and nodes that were affected. However, only updating the merging costs $c_{\mathrm{smerge}}(k)$ require evaluating (8) and even in this case the state vector is calculated only for one state ($K_{\mathrm{a}} = 1$). It turns out that the costs $c_{\mathrm{sdel}}(k)$ and $c_{\mathrm{smerge}}(k)$ updates can be calculated in $O(T_{\mathrm{a}}D)$ time for one state k, and since there are of the order K_{a} states for which these costs have to be calculated, the complexity is $O(K_{\mathrm{a}}T_{\mathrm{a}}D)$. The complexity of updating the costs $c_{\mathrm{ndel}}(n)$ and $c_{\mathrm{nmove}}(n)$ for all the affected nodes is $O(T_{\mathrm{a}}D)$, where T_{a} denotes the number of data vectors that are affected by the updated nodes. Therefore, all the costs can be updated in $O(K_{\mathrm{a}}T_{\mathrm{a}}D)$ time after an individual atomic operation has been performed.

In summary, performing an atomic operation and the following cost updates takes computation time $O(K_{\mathrm{a}}^3 + K_{\mathrm{a}}T_{\mathrm{a}}D)$ in the worst case. In the following, we will consider the contribution of the two terms, K_{a}^3 and $K_{\mathrm{a}}T_{\mathrm{a}}D$, to the runtime of the algorithm separately.

Let us first calculate the complexity due to the term $K_{\mathrm{a}}T_{\mathrm{a}}D$ over the runtime of the algorithm, omitting the term K_{a}^3 which will be considered later. Throughout the runtime of the algorithm, $K \leq K_{\mathrm{a}} \leq 2K+2$, due to the way the groups are formed. The amount of potentially affected data vectors T_{a} varies, since the time span of individual groups and states increasing each time when neighboring groups are joined to form larger groups.

- In the beginning, there are $G \approx T/K$ groups and each group requires of the order K operations to reach $K' = K$ within the group. At this stage, the time span of an individual group is $T_{\mathrm{a}} \leq 2K$ and therefore the complexity of one atomic operation is $O(K_{\mathrm{a}}T_{\mathrm{a}}D) = O(K^2D)$ (omitting the term K_{a}^3 which will be considered later). The time complexity of processing all the groups is $O(G \times K \times K^2 D) = O(TK^2D)$
- After joining pairs of neighboring groups, there are $G \approx T/(2K)$ groups and $T_{\mathrm{a}} \leq 4K$. It is easy to see that the time complexity of processing all the groups is again $O(TK^2D)$.
- In the end, there is only one group, which requires or the order K operations to reach $K' = K$. Now $T_{\mathrm{a}} \leq T$, and therefore the time complexity required to process the group is $O(TK^2D)$.

In the above list, the event of joining neighboring groups takes place of the order $\log_2(T/K) = (\log T - \mathrm{const.})$ times and the processing of each group configuration takes time that is proportional to $O(TK^2D)$. Therefore, the overall complexity of the above processing is $O(T\log(T)K^2D)$.

Next, let us consider the term K_{a}^3 which was omitted above. The term is due to the matrix inversion in (8) and only concerns the *delete state* operation as described above. Important to note is that the number of updated states K_{a} after each state deletion depends on the number of nodes associated with the deleted state. Since new nodes are not generated during the runtime of the algorithm, $K_{\mathrm{a}} \approx K$ can occur in (8) only of the order T/K times before we run out of nodes. Therefore, the time complexity of the matrix inversion in (8) over the entire runtime of the algorithm in the worst case is $O(K^3 + T/K \times K^3) = O(TK^2)$, since $K \leq T$. Therefore, the time complexity of the entire algorithm remains $O(T\log(T)K^2D)$.

REFERENCES

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–289, Feb. 1989.

[2] J. Bilmes, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," Int. Comput. Sci. Inst., Berkeley, CA, Tech. Rep., 1998.

[3] S. P. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[4] P. Herrera-Boyer, A. Klapuri, and M. Davy, "Automatic classification of pitched musical instrument sounds," in *Signal Processing Methods for Music Transcription*, A. Klapuri and M. Davy, Eds. New York: Springer, 2006, pp. 163–200.

[5] M. Levy and M. Sandler, "Structural segmentation of musical audio by constrained clustering," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 16, no. 2, pp. 318–326, Feb. 2008.

[6] M. Casey, "General sound classification and similarity in MPEG-7," *Organized Sound*, vol. 6, no. 2, pp. 153–164, 2002.

[7] C. Roads, *The Computer Music Tutorial*. Cambridge, MA: MIT Press, 1996.

[8] S. Furui, "Speaker-independent isolated work recognition using dynamic features of speech spectrum," *IEEE Trans. Acoust., Speech Signal Process.*, vol. ASSP-34, no. 1, pp. 52–59, Feb. 1986.

[9] H. Zen, K. Tokuda, and T. Kitamura, "Reformulating the HMM as a trajectory model by imposing explicit relationships between static and dynamic feature vector sequences," *Comput. Speech Lang.*, vol. 21, no. 1, pp. 153–173, 2006.

[10] S. Young, D. Kershaw, J. Odell, O. D., V. Valtchev, and W. P., *The HTK Book*. Cambridge, U.K.: Cambridge Univ., 2000.

[11] M.-Y. Hwang and X. Huang, "Shared-distribution hidden Markov models for speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 1, no. 4, pp. 414–420, Jul. 1993.

[12] M. Ostendorf, V. Digalakis, and O. A. Kimball, "From HMMs to segment models: A unified view of stochastic modeling for speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 4, no. 5, pp. 360–378, Sep. 1996.

[13] W. J. Holmes and M. J. Russell, "Probabilistic-trajectory segmental HMMs," *Comput. Speech Lang.*, vol. 13, no. 1, pp. 3–37, 1999.

[14] K. Sim and M. Gales, "Discriminative semi-parametric trajectory model for speech recognition," *Comput. Speech Lang.*, vol. 21, no. 4, pp. 669–687, 2007.

[15] A.-V. I. Rosti, *Linear Gaussian Models for Speech Recognition*. Cambridge, U.K.: Cambridge Univ., 2004, Ph.D. dissertation.

[16] J. A. Bilmes, "Buried markov models: A graphical-modeling approach to automatic speech recognition," *Comput. Speech Lang.*, vol. 17, no. 2–3, pp. 213–231, 2003.

[17] D. X. Sun, "Statistical modeling of co-articulation in continuous speech based on data driven interpolation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1997, pp. 1751–1754.

[18] G. Wahba, *Spline Models for Observational Data*. Philadelphia, PA: Soc. for Industrial and Applied Math., 1990.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[20] A. Klapuri, T. Virtanen, and M. Heln, "Modeling musical sounds with an interpolating state model," in *Proc. Eur. Signal Process. Conf.*, Antalya, Turkey, 2005, pp. 1577–1580.

[21] T. Virtanen and T. Heittola, "Interpolating hidden Markov model and its application to automatic instrument recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Taipei, Taiwan, 2009, pp. 49–52.

[22] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer, 2002.

[23] A. Hyvrinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. New York: Wiley, 2001.

[24] B. A. Olshausen and D. F. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?," *Vis. Res.*, vol. 37, pp. 3311–3325, 1997.

[25] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Neural Inf. Process. Syst.*, Denver, CO, 2000, pp. 556–562.

[26] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[27] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[28] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 51, no. 11, pp. 4311–4322, Nov. 2006.

[29] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[30] J. Nikunen and T. Virtanen, "Noise-to-mask ratio minimization by weighted non-negative matrix factorization algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Dallas, TX, 2010, to be published.

[31] K. A. , "Multiple fundamental frequency estimation by summing harmonic amplitudes," in *Proc. 7th Int. Conf. Music Inf. Retrieval*, Victoria, BC, Canada, Oct. 2006.

[32] H. S. Malvar, "Lapped transforms for efficient transform/subband coding," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 6, pp. 969–978, Jun. 1990.

[33] A. Klapuri, "Analysis of musical instrument sounds by source-filter-decay model," in *Proc. IEEE Int. Conf. Audio, Speech, Signal Process.*, Honolulu, HI, 2007, pp. 53–56.

[34] F. Opolko and J. Wapnick, *McGill University Master Samples*. Montreal, QC, Canada: McGill Univ., 1987, Tech. Rep..

[35] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.

[36] I. Davidson and A. Satyanarayana, "Speeding up k-means clustering by bootstrap averaging," in *Proc. IEEE Data Mining Workshop Clustering Large Data Sets*, Miami, FL, 2003, p. .

[37] T. Heittola, A. Klapuri, and T. Virtanen, "Musical instrument recognition in polyphonic audio using source-filter model for sound separation," in *Proc. 10th Int. Soc. Music Inf. Retrieval Conf.*, Kobe, Japan, 2009.

Anssi Klapuri (M'06) received the M.Sc. and Ph.D. degrees in information technology from Tampere University of Technology (TUT), Tampere, Finland, in 1998 and 2004, respectively.

He visited as a Post-Doctoral Researcher at the Ecole Centrale de Lille, France, and Cambridge University, Cambridge, U.K., in 2005 and 2006, respectively. He worked until 2009 as a Professor at TUT. In December 2009, he joined Queen Mary, University of London, London, U.K., as a Lecturer in sound and music processing. His research interests include audio signal processing, auditory modeling, and machine learning.

Tuomas Virtanen (M'07) received the M.Sc. and Doctor of Science degrees in information technology from the Tampere University of Technology (TUT), Tampere, Finland, in 2001 and 2006, respectively.

He is currently working as a Senior Researcher in the Department of Signal Processing, TUT. He has also been working as a Research Associate at Cambridge University Engineering Department, Cambridge, U.K. His research interests include content analysis of audio signals, sound source separation, noise-robust automatic speech recognition, and machine learning.