Efficient and Robust Music Identification with Weighted Finite-State Transducers

Mehryar Mohri, Pedro Moreno, and Eugene Weinstein

***Abstract*—We present an approach to music identification based on weighted finite-state transducers and Gaussian mixture models, inspired by techniques used in large-vocabulary speech recognition. Our modeling approach is based on learning a set of elementary music sounds in a fully unsupervised manner. While the space of possible music sound sequences is very large, our method enables the construction of a compact and efficient representation for the song collection using finite-state transducers.**

This paper gives a novel and substantially faster algorithm for the construction of *factor transducers*, the key representation of song snippets supporting our music identification technique. The complexity of our algorithm is linear with respect to the size of the suffix automaton constructed. Our experiments further show that it helps speed up the construction of the weighted suffix automaton in our task by a factor of 17 with respect to our previous method using the intermediate steps of determinization and minimization. We show that, using these techniques, a large-scale music identification system can be constructed for a database of over $15\,000$ songs while achieving an identification accuracy of 99.4% on undistorted test data, and performing robustly in the presence of noise and distortions.

***Index Terms*—Music identification, content-based information retrieval, factor automata, suffix automata, finite-state transducers**

I. Introduction

AUTOMATIC identification of music has been the subject of several recent studies both in research [1]–[3] and industry [4]–[6]. Given a test recording of a few seconds, music identification systems attempt to find the matching reference recording in a large database of songs. This technology has numerous applications, including end-user content based search, radio broadcast monitoring by recording labels, and copyrighted material detection by audio and video content providers such as Google YouTube.

In a practical setting, the test recording provided to a music identification system is usually limited in length to a few seconds. Hence, a music identification system is tasked with not only picking the song in the database that the recording came from, but also aligning the test recording against a particular position in the reference recoding. In addition, the machinery used to record and/or transmit the query audio, such as a cell phone, is often of low quality. These challenges highlight the need for robust music identification systems. The approach described in this article has robustness as a central consideration, and we demonstrate that the performance of our system is robust to several different types of noise and distortions.

Much of the previous work on music identification (see [7] for a recent survey) is based on hashing of frequency-domain features. The features used vary from work to work. Haitsma et al. [1] used hand-crafted features of energy differences between Bark-scale cepstra. Ke et al. [2] used similar features, but selected them automatically using boosting. Covell et al. [5] further improved on Ke by using wavelet features. Casey et al. [6] used cepstral features in conjunction with Locality Sensitive Hashing (LSH) for nearest-neighbor retrieval for music identification and detection of cover songs and remixes. Hashing approaches index the feature vectors computed over all the songs in the database in a large hash table. During test-time, features computed over the test audio are used to retrieve from the table.

Hashing-based approaches are marked by two main limitations, the requirement to match a fingerprint exactly or almost exactly and the need for a disambiguation step to reject many false positive matches. Batlle et al [3] proposed to move away from hashing approaches by suggesting a system decoding MFCC features over the audio stream directly into a sequence of audio events, as in speech recognition. Each song is represented by a sequence of states in a hidden Markov model (HMM), where a state corresponds to an elementary music sound. However, the system looks only for atomic sound sequences of a particular length, presumably to control search complexity.

In this work, we present an alternative approach to music identification based on weighted finite-state transducers and Gaussian mixture models, inspired by techniques used in large-vocabulary speech recognition. The learning phase of our approach is based on an unsupervised training process yielding an inventory of music phoneme units similar to phonemes in speech and leading to a unique sequence of music units characterizing each song. The representation and algorithmic aspects of this approach are based on weighted finite-state transducers, more specifically *factor transducers*, which can be used to give a compact representation of all song snippets for a large database over $15\,000$ songs. This approach leads to a music identification system that achieves an identification accuracy of 99.4% on undistorted test data, and performs robustly in the presence of noise and distortions. It allows us to index music event sequences in an optimal and compact way and, as we shall later demonstrate, with very rare false positive matches.

A primary contribution of this paper is a novel and efficient algorithm for the construction of a weighted suffix or factor

Mehryar Mohri and Eugene Weinstein are with the Courant Institute of Mathematical Sciences, New York, NY USA, and Google Inc. e-mail: {mohri,eugenew}@cs.nyu.edu. Pedro Moreno is with Google Inc., New York, NY USA. e-mail: pedro@google.com.

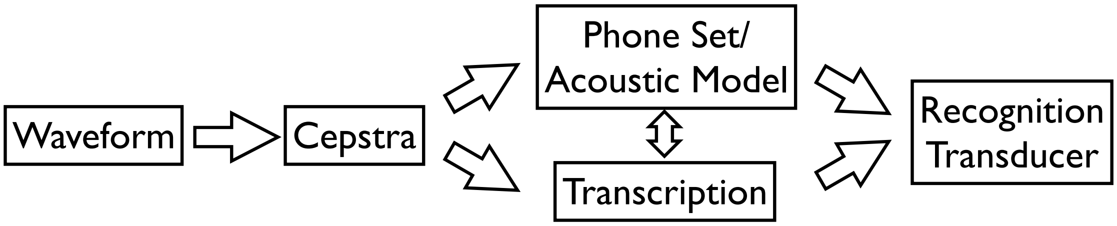


Fig. 1. Music Identification System Construction

automaton from an input acyclic weighted automaton. The complexity of the algorithm is linear in the size of the output suffix automaton. The algorithm is also straightforward to implement and is very efficient in practice. Our experiments show that this algorithm helps speed up the construction of the weighted suffix automaton by a factor of 17 over the previous algorithms for constructing an index of a music song collection.

The remainder of this paper is organized as follows. Section II presents an overview of our music identification approach including our acoustic modeling technique and the construction of the recognition transducer from a weighted factor automaton. This transducer is searched by our decoder to identify a test recording. In Section III, we present the essential properties of our weighted suffix and factor automaton and give a linear-time algorithm for its construction from an input weighted automaton. Section IV reports our experiments with this algorithm demonstrating that it is substantially faster than our previous construction method. We also present empirical results illustrating the robustness of our music identification system.

II. Music Identification with Weighted Finite-State Transducers

In our music identification approach, each song is represented by a distinct sequence of music sounds, called *music phonemes* in our work. Fig. I gives an architectural view of our system. Our system learns the set of music phonemes automatically from training data using an unsupervised algorithm. We also learn a unique sequence of music phonemes characterizing each song. The music identification problem is then reduced to a mapping of music phoneme sequences to songs. As in a speech recognition system, this mapping can be represented compactly with a finite-state transducer.

Specifically, a test audio snippet can be decoded into a music phoneme sequence using the Viterbi beam search algorithm. The transducer associates a weight to each pairing of a phoneme sequence with a song, and the search process approximates the most likely path through the transducer given the acoustic evidence [8].

However, our music song collection is not transcribed with reference music phoneme sequences, and hence the music phoneme inventory has to be learned simultaneously with the most likely transcription for each song. Also, the size of the transducer representing the entire song collection can be quite large. In addition, the requirement to identify song snippets (as opposed to entire songs) introduces additional algorithmic challenges. In the remainder of this section, we address these two problems.

A. Acoustic Modeling

Our acoustic modeling approach consists of jointly learning an inventory of music phonemes and the sequence of phonemes best representing each song. Cepstral features have recently been shown to be effective in the analysis of music [3], [9], [10], and in our work we also use mel-frequency cepstral coefficient (MFCC) features computed over the song audio. We use 100ms windows over the feature stream, and keep the first twelve coefficients, the energy, and their first and second derivatives to produce a 39-dimensional feature vector.

1) Model Initialization: A set of music phonemes is initially created by clustering segments of pseudo-stationary audio signal in all the songs. The song audio is segmented by sliding a window along the features and fitting a single diagonal covariance Gaussian model to each window. We compute the symmetrized KL divergence between the resulting probability distributions of all adjacent window pairs. The symmetrized KL divergence between two Gaussians $G_1 \sim N(\mu_1, \Sigma_1)$ and $G_2 \sim N(\mu_2, \Sigma_2)$ as used in this work is defined as double the sum of the non-symmetric KL divergences,

$$\begin{aligned}\mathrm{KL}_{sym}(G1, G2) &= 2\left(\mathrm{D}_{KL}(G_1 \| G2) + \mathrm{D}_{KL}(G_2 \| G_1)\right) \\ &= \mathrm{tr}\left(\Sigma_2 \Sigma_1^{-1}\right) + \mathrm{tr}\left(\Sigma_1 \Sigma_2^{-1}\right) \\ &\quad + (\mu_2 - \mu_1)^\top \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)(\mu_2 - \mu_1) \\ &\quad -2m \end{aligned} \tag{1}$$

where m is the dimensionality of the data. After smoothing the KL divergence signal, we hypothesize segment boundaries where the KL divergence between adjacent windows is large.

We then apply a clustering algorithm to the song segments to produce one cluster for each of k desired phonemes. Clustering is performed in two steps. First we apply hierarchical, or divisive, clustering in which all data points (hypothesized segments) are initially assigned to one cluster. The centroid (mean) of the cluster is then randomly perturbed in two opposite directions of maximum variance to make two new clusters. Points in the old cluster are reassigned the child cluster with higher likelihood [11]. This step ends when the desired number of clusters or music phonemes k is reached or the number of points remaining is too small to accommodate a split. In a second step we apply ordinary k-means clustering to refine the clusters until convergence is achieved. As in [11] we use maximum likelihood as an objective distance function rather than the more common Euclidean distance.

2) Model Training: The acoustic model for each of our k phonemes is initially a single Gaussian parametrized with the sufficient statistics of a single segment cluster obtained in the above initialization procedure. However, a single Gaussian is unlikely to accurately represent a complex music sound. In speech recognition, a common modeling technique is to use a mixture of Gaussians to represent speech phoneme acoustics in the cepstral feature space. Following this approach, we model music phonemes with Gaussian mixture models.

Since there are no reference transcriptions of the song database in terms of music sound units, we use an unsupervised learning approach similar to that of [3] in which the model representing each music phoneme and the transcriptions are inferred simultaneously. The training procedure repeats the following two steps until convergence is achieved:

- Apply Viterbi decoding using the current model and allowing any sequence of music phonemes (i.e., no language model) to find a transcription for each song.
- Refine the model using the standard expectation-maximization (EM) training algorithm using the current transcriptions as reference.

This process is similar to the standard acoustic model training algorithm for speech recognition with the exception that at each training iteration, a new transcription is obtained for each song in our database. This process is illustrated in Fig. 2. Note that since a full Viterbi search is performed at each iteration, the transcription as well as the alignment of phonemes to audio frames may change.

3) Measuring Convergence: In speech recognition, each utterance is usually labeled with a ground truth transcription that is fixed throughout the acoustic model training process. Convergence is typically evaluated by measuring the change in model likelihood from iteration to iteration. Since in our music identification scenario no such ground truth exists, to evaluate the convergence of our algorithm we measure how much the reference transcription changes with each iteration. To compare transcriptions we use the edit distance, here defined as the minimal number of insertions, substitutions, and deletions of music phonemes required to transform one transcription into another.

For a song set S let $t_i(s)$ be the transcription of song s at iteration i and $\mathrm{ED}(a, b)$ the edit distance of sequences a and b. At each iteration i, we compute the average edit distance per song

$$C_i = \frac{1}{|S|} \sum_{s \in S} \mathrm{ED}(t_i(s), t_{i-1}(s)) \qquad (2)$$

as our convergence measure.

Fig. 2 illustrates this situation by giving three example transcriptions assigned to the same song in consecutive acoustic model training iterations. We have $t_1(s)$ =mp2 mp5 mp86; $t_2(s)$ = mp2 mp43 mp22 mp86, and $t_3(s)$ = mp37 mp43 mp22 mp86. The edit distances computed here will be $\mathrm{ED}(t_1(s), t_2(s)) = 2$ and $\mathrm{ED}(t_2(s), t_3(s)) = 1$.

Fig. 3 illustrates how the edit distance changes during training for three phoneme inventory sizes. Note that, for example, with 1024 phonemes almost 900 edits on average occurred per song between the first and second round of training. Considering that the average transcription length is around 1700 phonemes, this means that only around half of the phonemes remained the same. In our experiments, convergence was exhibited after around twenty iterations. In the last few iterations of training, the average edit distance decreases considerably to around 300, meaning 5/6 of the phonemes remain the same from iteration to iteration. It is intuitive that the average edit distance achieved at convergence grows with the phoneme inventory size, since with a very large phoneme inventory many phonemes will be statistically very close. In the other extreme, with only one music phoneme, the transcription would never change at all.

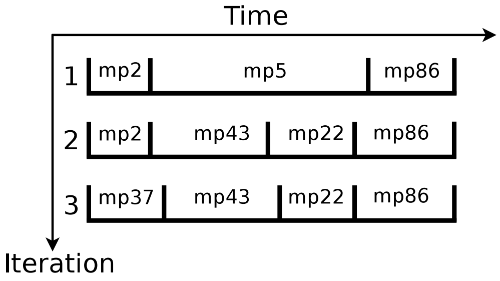


Fig. 2. An illustration of changing transcription and alignment for a particular song during the course of three iterations of acoustic model training. mp$x$ stands for music phoneme number x and the vertical bars represent the temporal boundaries between music phonemes.

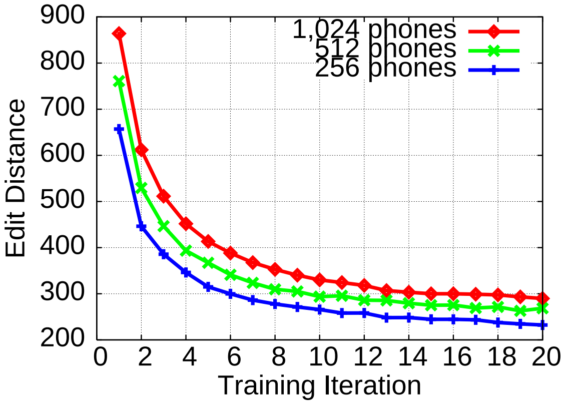


Fig. 3. Average edit distance per song vs. training iteration.

B. Automata Overview

Before we describe the transducer representation of our music collection, we briefly review the automata concepts relevant to this work. The devices of interest in this paper are weighted automata and transducers. For the purposes of this paper, weighted automata and transducers are defined as follows.

A *weighted automaton* is specified by an alphabet, a finite set of states, a finite set of transitions, an initial state, a set of final states, and a final weight function. Each transition associates pairs of states with a symbol and a weight. A *weighted finite-state transducer* is specified by input and

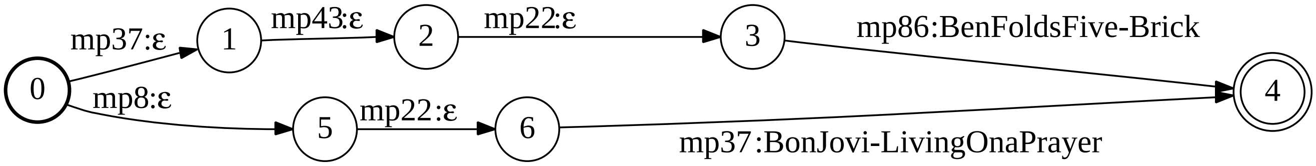


Fig. 4. Finite-state transducer T_0 mapping each song to its identifier. mp$x$ stands for music phoneme number x

output alphabets, a finite set of states, a finite set of transitions, an initial state, a set of final states, and a final weight function. Each transition associates pairs of states with an input symbol, an output symbol, and a weight.

Unweighted automata and transducers are obtained by simply omitting the weights from their weighted counterparts. Thus, a transition no longer associates a weight with a pair of states but only an input and/or output label. For example the transducer T_0 in Fig. 4 consists of seven states and seven transitions, with 0 the initial state and 4 the sole final state. The symbol ϵ represents the empty string. Input and output labels are given as `input:output`. For example, one path through T_0 associates the input sequence mp37 mp43 mp22 mp86 with the output label `BenFoldsFive-Brick`.

The *semiring* over which an acceptor or transducer is defined specifies the weight set used and the algebraic operations for combining weights. One semiring used extensively in speech and text processing is the tropical semiring. In the tropical semiring, the total path along a given path is found by adding the weights of the transitions composing the path. If the weights are log-likelihoods, the total weight of a path is the total log-likelihood. The total weight assigned by the automaton to a string x is that of the minimum weight (maximum likelihood) path labeled with x. For weighted automata the weight is indicated as `input:output/weight`. If the weight is omitted, then it is zero (in the tropical semiring). For example, the acceptor in Fig. 6(b) has an accepting path (that leading from the initial to a final state) labeled with mp22 mp86 with a weight of 0 and an accepting path labeled with mp8 mp22 mp37 with a weight of 1.

An automaton is *deterministic* if at any state no two outgoing transitions share the same input label. A deterministic automaton is *minimal* if there is no equivalent deterministic automaton with a smaller number of states. An automaton is *epsilon-free* if it contains no epsilon transitions. An automaton that is epsilon-free and deterministic can be processed in a time-efficient manner. A minimal (deterministic) automaton is further optimally space-efficient. Accordingly, such an automaton is often referred to as *efficient.* In fact, it is optimal in the sense that the lookup time for a given string in the automaton is linear in the size of the string. As a result of the relatively recent introduction of new algorithms, such as weighted determinization, minimization, and epsilon removal [12], [13], automata have become a compelling formalism used extensively in a number of fields, including speech, image, and language processing.

C. Recognition Transducer

Given a set of songs S, the music identification task is to find the songs in S that contain a query song snippet x. In speech recognition, it is common to construct a weighted finite-state transducer specifying the mapping of phoneme sequences to word sequences, and to decode test audio using the Viterbi algorithm constrained by the transducer [8]. Our music identification system operates in a similar fashion, but the final output of the decoding process is a single song identifier. Hence, the recognition transducer must map any sequence of music phonemes appearing in the transcriptions found in the final iteration of training to the corresponding song identifiers.

Let Σ denote the set of music phonemes and let the set of music phoneme sequences describing m songs be $U = \{x_1, \ldots, x_m\}, x_i \in \Sigma^*$ for $i \in \{1, \ldots, m\}$. A *factor,* or *substring,* of a sequence $x \in \Sigma^*$ is a sequence of consecutive phonemes appearing in x. Thus, y is a factor of x iff there exists $u, v \in \Sigma^*$ such that $x = uyv$. In our experiments, $m = 15\,455$, $|\Sigma| = 1024$ and the average length of a transcription x_i is more than 1700. Thus, in the worst case, there can be as many as $15\,455 \times 1700^2 \approx 45 \times 10^9$ factors. The size of a naïve prefix-tree-based representation would thus be prohibitive, and hence we endeavor to represent the set of factors with a much more compact factor automaton.

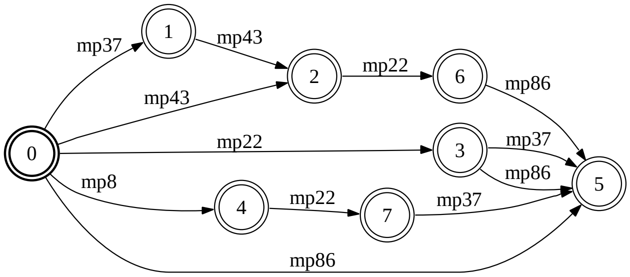


Fig. 5. Deterministic and minimal unweighted factor acceptor $F(A)$ for two songs.

1) Factor Automaton Construction: We denote by $F(A)$ the minimal deterministic automaton accepting the set of factors of a finite automaton A, that is the set of factors of the strings accepted by A. Similarly, we denote by $S(A)$ the minimal deterministic automaton accepting the set of suffixes of an automaton A. In the remainder of this section, we outline a method for constructing a factor automaton of an automaton using the general automata algorithms for weighted determinization and minimization. This method is described here to illustrate the concept of factor automata, as well as to give the context of the methods for constructing factor automata previously employed both in the present work and for other tasks (e.g., [14]). However, the novel suffix automaton algorithm given in Section III-B enjoys a linear complexity and thus is now the preferred method for this construction.

Let T_0 be the transducer mapping phoneme sequences to song identifiers before determinization and minimization. Fig. 4 shows T_0 when U is reduced to two short songs. Let A be the acceptor obtained by omitting the output labels of T_0. Intuitively, to accept any factor of A, we want to read strings in A, but starting and finishing at any pair of states linked by a sequence of transitions. We can accomplish this by creating "shortcut" ϵ-transitions from the initial state of A to all other states, making all states final, and applying ϵ-removal, determinization, and minimization to yield an efficient acceptor. This construction yields the factor automaton $F(A)$ (Fig. 5), but it does not allow us to output the song identifier associated with each factor.

2) The Algorithmic Challenge of Factor Automata: Constructing a compact and efficient factor automaton that retains the mapping between all possible music phoneme subsequences and the songs to which they correspond is non-trivial. The following intuitive, but naïve, solution illustrates this point. All accepting paths of the automaton A after the addition of ϵ-transitions, i.e. all factors, can be augmented with output labels corresponding to song identifiers. As a result, the matching song identifier is always obtained as an output when traversing a set of input music phonemes.

However, this approach immediately fails because factors with different output labels cannot be collapsed into the same path, and as a result upon determinization and minimization the resulting transducer is prohibitively larger than A. Thus, the crucial question about the problem of constructing a factor transducer for our music identification task is how to construct an automaton where states and transitions can be shared among paths belonging to different songs, while preserving the mapping between phoneme sequences and songs.

3) Using Weights to Represent Song Labels: Our approach for avoiding the combinatorial explosion just mentioned is to use weights, instead of output labels, to represent song identifiers. We create a compact weighted acceptor over the tropical semiring accepting the factors of U that associates the total weight s_x to each factor x. During the application of weighted determinization and minimization to construct a factor automaton, the song identifier is treated as a weight that can be distributed along a path. The property that the sum of the weights along the path labeled with x is s_x is preserved by these operations. As a result, paths belonging to transcription factors common to multiple songs can be collapsed and the mapping between factors to songs is preserved.

To construct the weighted factor automaton $F_w(A)$ from T_0 we

1) Drop the output labels to produce A.
2) Assign a numerical label to each song and augment each song's path in A with that label as a single weight (at the transition leaving the initial state).
3) Add ϵ-transitions from the initial state to each other state weighted with the song identifier corresponding to the path of the song to which the transition serves as a "shortcut." This produces the weighted acceptor $F_\epsilon(A)$ (Fig. 6(a)).
4) Apply epsilon removal, determinization, and minimization to produce the weighted acceptor $F_w(A)$ (Fig. 6(b)).

Observe in Fig. 6(b) that the numerical labels 0 and 1 are assigned to song labels `BenFoldsFive-Brick` and `BonJovi-LivingOnaPrayer`, respectively. Notice that, for example, the factor `mp`22 `mp`37 is correctly assigned a weight of 1 by $F_w(A)$. Observe finally that the information about all the factors found in the original transducer T_0 (Fig. 4) and the songs they correspond to is preserved.

Finally, $F_w(A)$ is transformed into a song recognition transducer T by treating each output weight integer as a regular output symbol. Given a music phoneme sequence as input, the associated song identifier is obtained by summing the outputs yielded by T.

We have empirically verified the feasibility of this construction. For $15\,455$ songs, the total number of transitions of the transducer T is about 53.0M (million), only about 2.1 times that of the minimal deterministic transducer T_0 representing all full-song transcriptions. In Section III, we present the results of a careful analysis of the size of factor automata of automata and give a matching linear-time algorithm for their construction. These results suggest that our method can scale to a larger set of songs, e.g., several million songs.

III. Analysis and Construction of Suffix and Factor Automata

As discussed in Section IV-C, the above representation of music sequences with factor automata is empirically compact for our music collection of over $15\,000$ songs. To ensure the scalability of our approach to a larger set of songs, we wished to derive a bound on the size of the factor automata of automata. One quality of the music phoneme sequences considered in this as well as in many other applications is that the sequences do not share long suffixes. This motivated our analysis of the size of the factor automata with respect to the length of the common suffixes in the original automaton, formalized with the following definition.

Definition Let k be a positive integer. A finite automaton A is *k-suffix-unique* if no two strings accepted by A share a suffix of length k. A is said to be *suffix-unique* when it is k-suffix-unique with $k = 1$.

A. Bounds on the Size of Suffix and Factor Automata

The following two propositions give novel and substantially improved bounds on the size of the suffix and factor automata of A if A is suffix-unique and k-suffix-unique. The notation $|A|_Q$, $|A|_E$ and $|A|$ is used to refer to the number of states, transitions, and states and transitions combined, respectively, in the automaton A.

The factor automaton $F(A)$ can be obtained from the suffix automaton $S(A)$ by making all states final and applying minimization. Thus, $|F(A)| \leq |S(A)|$. The following bounds are stated as bounds on the size of the factor automaton $F(A)$, but they are actually proved as bounds on that of the suffix

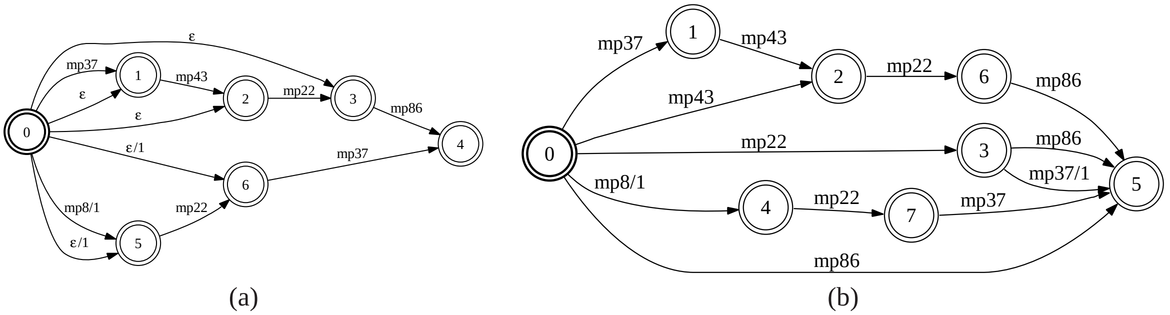


Fig. 6. (a) Factor acceptor $F_\epsilon(A)$ for two songs produced by adding weights and epsilon transitions to A. (b) Deterministic and minimal weighted factor acceptor $F_w(A)$ produced by optimizing $F_\epsilon(A)$.

automaton $S(A)$, and apply of course also to $F(A)$ by the size relationship just mentioned. The detailed proofs of the following result are given in [15], [16].

Proposition 1. *Let A be a suffix-unique deterministic and minimal automaton accepting strings of length more than three. Then, the number of states of the suffix or factor automaton of A is bounded as follows*

$$|F(A)|_Q \leq 2|A|_Q - 3. \tag{3}$$

Proposition 2. *Let A be a k-suffix-unique automaton accepting strings of length more than three and let n be the number of strings accepted by A. Then, the following bound holds for the suffix or factor automaton of A:*

$$|F(A)|_Q \leq 2|A_k|_Q + 2kn - 3. \tag{4}$$

where A_k is the part of the automaton of A obtained by removing the states and transitions of all suffixes of length k.

Corollary 1. *Let $U = \{x_1, \ldots, x_m\}$ be a set of strings of length more than three and let A be a prefix-tree representing U. Then, the number of states of the suffix or factor automaton of the strings of U is bounded as follows*

$$|F(U)|_Q \leq 2|A|_Q - 2. \tag{5}$$

B. Weighted Suffix Automaton Algorithm

In Section II-C we described a method for constructing a compact factor transducer T mapping music phoneme sequences to song identifiers by adding ϵ-transitions to A and applying weighted determinization and minimization; however, we stated that a more efficient method would be presented shortly. Indeed, the bounds in section III-A guarantee only a linear size increase from A to $S(A)$ and $F(A)$. However, the ϵ-removal and determinization algorithms used in this method have in general at least a quadratic complexity in the size of the input automaton. While the final result of the construction algorithm is guaranteed to be compact, the algorithms described thus far are not optimal.

This section describes a new linear-time algorithm for the construction of the suffix automaton $S(A)$ of a weighted suffix-unique input automaton A, or similarly the factor automaton $F(A)$ of A. Since $F(A)$ can be obtained from $S(A)$ by making all states of $S(A)$ final and applying a linear-time acyclic minimization algorithm [17], it suffices to describe a linear-time algorithm for the construction of $S(A)$. It is possible however to give a similar direct linear-time algorithm for the construction of $F(A)$. The algorithm given in this section holds over the tropical semiring, which is used in our music identification system; however, we conjecture that this algorithm can be generalized to arbitrary semirings. The algorithm is a generalization of the unweighted algorithm presented in [16].

```
CREATE-SUFFIX-AUTOMATON$(A, f)$
$S \leftarrow Q_S \leftarrow \{I\}$ $\triangleright$ initial state
$s[I] \leftarrow$ UNDEFINED; $l[I] \leftarrow 0; W[i] \leftarrow 0$
while $S \neq \emptyset$ do
    $p \leftarrow$ HEAD$(S)$
    for each $a$ such that $\delta_A(p, a) \neq$ UNDEFINED do
        if $\delta_A(p, a) \neq f$ then
            $Q_S \leftarrow Q_S \cup \{p\}$
            $l[q] \leftarrow l[p] + 1$
            SUFFIX-NEXT$(p, a, q)$
            ENQUEUE$(S, q)$
$Q_S \leftarrow Q_S \cup \{f\}$
for each state $p \in Q_A$ and $a \in \Sigma$ s.t. $\delta_A(p, a) = f$ do
    SUFFIX-NEXT$(p, a, f)$
    SUFFIX-FINAL$(f)$
for each $p \in F_A$ do
    SUFFIX-FINAL$(q)$
$\rho_S(I) \leftarrow \min_{p \in Q_S} W[p]$
return $S(A) = (Q_S, I, F_S, \delta_S)$
```

Fig. 7. Algorithm for the construction of the weighted suffix automaton of a suffix-unique automaton A.

Figs. 7-9 give the pseudocode of the algorithm for constructing the suffix automaton $S(A) = (Q_S, I, F_S, \delta_S, \omega_S, \rho_S)$ of a suffix-unique automaton $A = (Q_A, I, F_A, \delta_A, \omega_A, \rho_A)$, where $\delta_S \colon Q_S \times \Sigma \mapsto Q_S$ denotes the partial transition function of $S(A)$ and likewise $\delta_A \colon Q_A \times \Sigma \mapsto Q_A$ that of A; $\omega_S \colon Q_S \times \Sigma \mapsto \mathbb{K}$ and $\omega_A \colon Q_A \times \Sigma \mapsto \mathbb{K}$ give the weight for each transition in $S(A)$ and A, respectively; and $\rho_S \colon F_S \mapsto \mathbb{K}$ and $\rho_A \colon F_A \mapsto \mathbb{K}$ are the final weight functions for $S(A)$ and A. f denotes the unique final state of A with no outgoing transitions. Let x be the longest string in $S(A)$ reaching the

```
SUFFIX-NEXT$(p, a, q)$
 1  $l[q] \leftarrow \max(l[p] + 1, l[q])$
 2  $W[q] \leftarrow W[p] + \omega_A(p, a)$
 3  while $p \neq I$ and $\delta_S(p, a) =$ UNDEFINED do
 4      $\delta_S(p, a) \leftarrow q$
 5      $\omega_S(p, a) \leftarrow W[q] - W[p]$
 6      $p \leftarrow s[p]$
 7  if $\delta_S(p, a) =$ UNDEFINED then
 8      $\delta_S(I, a) \leftarrow q$
 9      $\omega_S(I, a) \leftarrow W[q]$
10      $s[q] \leftarrow I$
11  elseif $l[p] + 1 = l[\delta_S(p, a)]$ and $\delta_S(p, a) \neq q$ then
12      $s[q] \leftarrow \delta_S(p, a)$
13  else $r \leftarrow q$
14      if $\delta_S(p, a) \neq q$ then
15          $r \leftarrow$ copy of $\delta_S(p, a)$ with same transitions
16          $Q_S \leftarrow Q_S \cup \{r\}$
17          $s[q] \leftarrow r$
18      $s[r] \leftarrow s[\delta_S(p, a)]$
19      $s[\delta_S(p, a)] \leftarrow r$
20      $W[r] \leftarrow W[p] + \omega_S(p, a)$
21      $l[r] \leftarrow l[p] + 1$
22      while $p \neq$ UNDEFINED and $l[\delta_S(p, a)] \geq l[r]$ do
23          $\delta_S(p, a) \leftarrow r$
24          $\omega_S(p, a) \leftarrow W[r] - W[p]$
25          $p \leftarrow s[p]$
```

Fig. 8. Subroutine of CREATE-SUFFIX-AUTOMATON processing a transition of A from state p to state q labeled with a.

```
SUFFIX-FINAL$(p)$
1  $m \leftarrow W[p]$
2  if $p \in F_S$ then
3      $p \leftarrow s[p]$
4  while $p \neq$ UNDEFINED and $p \notin F_S$ do
5      $F_S \leftarrow F_S \cup \{p\}$
6      $\rho_S(p) \leftarrow m - W[p]$
7      $p \leftarrow s[p]$
```

Fig. 9. Subroutine of CREATE-SUFFIX-AUTOMATON making all states on the suffix chain of p final.

state p and $u \in \Sigma^*$ the longest suffix of x reaching a distinct state p' in the automaton such that u is the longest string reaching p'. Then p' is referred to so the *suffix link* or *suffix pointer* of p.

The algorithm given here generalizes our previous linear-time unweighted suffix automaton construction algorithm [16] to the case where A is a weighted automaton. The unweighted algorithm is in turn a generalization to an input suffix-unique automaton of the standard construction for a single input string [18], [19]. Our presentation is similar to that of [18]. The algorithm maintains two arrays $s[q]$ and $l[q]$ for each state q of Q_S. $s[q]$ denotes the suffix pointer or failure state of q. $l[q]$ denotes the length of the longest path from the initial state to q in $S(A)$. l is used to determine the so-called *solid edges* or *transitions* in the construction of the suffix automaton. A transition (p, a, q) is solid if $l[p] + 1 = l[q]$, that is it is on a longest path from the initial state to q, otherwise, it is a shortcut transition.

We assume that for all $p \in Q_A$, $\rho_A(p) = 0$, since we may encode A to contain no final weights as follows: for any state p such that $\rho_A(p) = e$, we set $\rho_A(p) = 0$ and add a transition such that $\delta_A(p, \$) = f$ and $\omega_A(p, \$) = e$, where $\$$ is a unique encoding symbol for this transition. Decoding the resulting suffix automaton simply reverses this process. The weighted suffix automaton algorithm relies on the computation of $W[p]$, the forward potential of state p, i.e., the total weight of the path from I to p in A. The introduction of W yields a natural extension of our previous unweighted algorithm to the weighted case. This forward potential is computed as the automaton is traversed and is used to set weights as transitions are added to and redirected within $S(A)$. Throughout the algorithm, for any transition (p, a, q) in $S(A)$, we set $\omega_S(p, a) = W[q] - W[p]$ so that traversing a suffix in $S(A)$ yields the same weight as traversing the original string in A. As a result, any solid transition in $S(A)$ retains its weight from A.

S is a queue storing the set of states to be examined. The particular queue discipline of S does not affect the correctness of the algorithm, but we can assume it to be a FIFO order, which corresponds to a breadth-first search and thus admits a linear-time implementation. In each iteration of the loop of lines 3-10 in Fig. 7, a new state p is extracted from S. The processing of the transitions (p, a, f) with destination state f is delayed to a later stage (lines 12-14). This is because of the special property of f that it may not only admit different suffix pointers [16] but also different values of $l[f]$ and $W[f]$. Other transitions (p, a, q) are processed one at a time by creating, if necessary, the destination state q and adding it to Q_S, defining $l[q]$ and calling SUFFIX-NEXT(p, a, q).

The subroutine SUFFIX-NEXT processes each transition (p, a, q) of A. The loop of lines 3-6 inspects the iterated suffix pointers of p that do not have an outgoing transition labeled with a. It creates such transitions reaching q from all the iterated suffix pointers until the initial state or a state p' already admitting such a transition is reached. In the former case, the suffix pointer of q is set to be the initial state I and the transition (I, a, q) is created.

In the latter case, if the existing transition (p', a, q') is solid and $q' \neq q$, then the suffix pointer of q is simply set to be q' (line 12). Otherwise, if $q' \neq q$, a copy of the state q', r, with the same outgoing transitions is created (lines 15-16) and the suffix pointer of q is set to be r. The suffix pointer of q' is set to be r (line 19) and that of r is set to be $s[q']$ (18), and $l[r]$ set to $l[p] + 1$ (21). The transitions labeled with a leaving the iterated suffix pointers of p are inspected and redirected to r so long as they are non-solid transitions (lines 22-25).

The subroutine SUFFIX-FINAL sets the finality and the final weight of states in $S(A)$. For any state p that is final in A, p and all the states found by following the chain of suffix pointers starting at p are made final in $S(A)$ in the loop of lines 4-7. The final weight of each state p' found by traversing

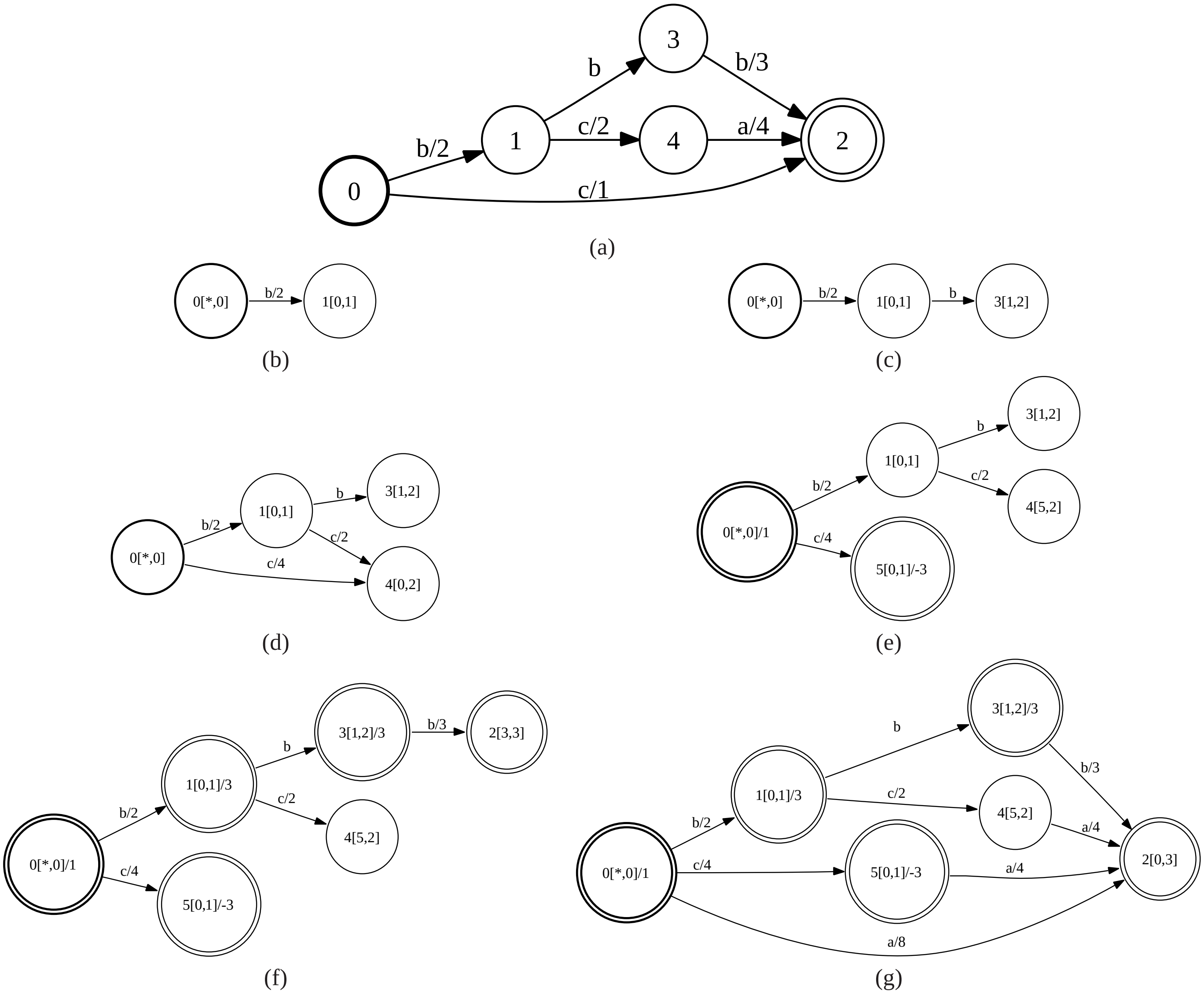


Fig. 10. Construction of the weighted suffix automaton using CREATE-SUFFIX-AUTOMATON. (a) Original automaton A. (b)-(g) Intermediate stages of the construction of $S(A)$. For each state $n[s,l]/w$, n is the state number, s is the suffix pointer of n, l is $l[n]$, and w is the final weight, if any.

the suffix pointer chain is set to $W[p] - W[p']$ (line 6).

We have implemented and tested the weighted suffix automaton algorithm just described. Fig. 10 illustrates the application of the algorithm to a particular weighted automaton. All intermediate stages of the construction of $S(A)$ are indicated, including $s[q]$, $W[q]$, and $l[q]$ for each state q.

Proposition 3. *Let A be a minimal deterministic suffix-unique automaton. Then, a call to* CREATE-SUFFIX-AUTOMATON(A, f) *constructs the suffix automaton of A, $S(A)$ in time linear in the size of $S(A)$, that is in $O(|S(A)|)$.*

Proof: The unweighted version of the suffix automaton construction algorithm is shown to have a complexity of $O(|S(A)|)$ in [16]. The total number of transitions added and redirected by the unweighted algorithm is of course also linear. In the weighted algorithm given in Figs. 7-9, transitions are added and redirected in the same way as in the unweighted algorithm, and weights are only adjusted when transitions are added or redirected (with the exception of the single initial weight adjustment in line 17 of CREATE-SUFFIX-AUTOMATON). Hence, the total number of weight adjustments is also linear. ■

IV. EXPERIMENTS

In the following, we discuss the experimental evaluation of our music identification system. The software tools used for acoustic modeling and runtime Viterbi decoding were those developed at Google for large-vocabulary speech recognition applications [20]. The algorithms for constructing the finite-state transducer representation of the song database were implemented in the OpenFst toolkit [21].

A. Music Detection

In a practical music identification system, a test recording may be provided that does not belong to a song in our database. Hence, an important task is music detection, or classifying songs as belonging to our database or not. To detect out-of-set songs, we use a single universal background acoustic model (UBM) generically representing all possible song sounds. This is similar to techniques used in speaker identification (e.g., [22]). The UBM is constructed by applying a divisive clustering algorithm to Gaussian mixtures across the GMMs of all the music phonemes, until the desired number of clusters/mixtures is yielded. We used a UBM with 16 clustered components.

To classify a song and in-set or out-of-set, we compute the log-likelihood of the best path in a Viterbi search through the regular song identification transducer and that given a trivial transducer allowing only the UBM. When the ratio of these two likelihoods is large, the test audio is accounted for much better by the in-set models than the generic model and hence it's more likely to have come from an in-set song, and vice versa. As a binary classification problem, this is a natural task for discriminative classifiers such as support-vector machines (SVMs) [23], [24]. The input to the SVM is a three-dimensional feature vector $[L_r, L_b, (L_r - L_b)]$ for each song snippet, where L_r and L_b are the log-likelihoods of the best path and background acoustic models, respectively. We used the `LIBSVM` implementation [25] with a radial basis function (RBF) kernel. The accuracy was measured using 10-fold cross-validation.

B. Detection and Identification Experiments

Our training data set consisted of $15\,455$ songs. The average song duration was 3.9 minutes, for a total of over 1000 hours of training audio. The test data consisted of 1762 in-set and 1856 out-of-set 10-second snippets drawn from 100 in-set and 100 out-of-set songs selected at random. The first and last 20 seconds of each song were omitted from the test data since they were more likely to consist of primarily silence or very quiet audio.

Our music phoneme inventory size was 1024 units because it was convenient for the divisive clustering algorithm for the number of phonemes to be a power of two, and also because an inventory of this size produced good results. Each phoneme's acoustic model consisted of 16 mixture components. All experiments run faster than real time: for instance with a Viterbi search beam width of 12, the runtime is 0.48 of real time (meaning a song snippet of m seconds can be processed in $0.48m$ seconds). We tested the robustness of our system by applying the following transformations to the audio snippets:

1) WNoise-x: additive white noise (using `sox`). Since white noise is a consistently broadband signal, this simulates harsh noise. x is the noise amplitude compared to saturation (e.g., WNoise-0.01 is 0.01 of saturation).
2) Speed-x: speed up or slow down by factor of x (using `sox`). Radio stations frequently speed up or slow down songs in order to produce more appealing sound [3].
3) MP3-x: mp3 reencode at x kbps (using `lame`). This simulates compression or transmission at a lower bitrate.

The identification and detection accuracy results are presented in Table I, showing almost perfect identification accuracy on clean data. The addition of white noise degrades the accuracy when the mixing level of the noise is increased. This is to be expected as the higher mixing levels result in a low signal-to-noise ratio (SNR). The inclusion of noisy data in the acoustic model training process slightly improves identification quality – for instance, in the WNoise-0.01 experiment, the accuracy improves from 85.5% to 88.4%. Slight variations in playback speed are handled quite well by our system (high 90's); however, major variations such as 0.9x and 1.1x cause the accuracy to degrade into the 40's. MP3 recompression at low bitrates is handled well by our system.

Condition	Identification Accuracy	Detection Accuracy
Clean	99.4%	96.9%
WNoise-0.001 (44.0 dB SNR)	98.5%	96.8%
WNoise-0.01 (24.8 dB SNR)	85.5%	94.5%
WNoise-0.05 (10.4 dB SNR)	39.0%	93.2%
WNoise-0.1 (5.9 dB SNR)	11.1%	93.5%
Speed-0.98	96.8%	96.0%
Speed-1.02	98.4%	96.4%
Speed-0.9	45.7%	85.8%
Speed-1.1	43.2%	87.7%
MP3-64	98.1%	96.6%
MP3-32	95.5%	95.3%

TABLE I
IDENTIFICATION ACCURACY RATES UNDER VARIOUS TEST CONDITIONS

The detection performance of our system is in the 90's for all conditions except the 10% speedup and slowdown. This is most likely due to the spectral shift introduced by the speed alteration technique. This shift results in a mismatch between the audio data and the acoustic models. We believe that a time scaling method that maintains spectral characteristics would be handled better by our acoustic models.

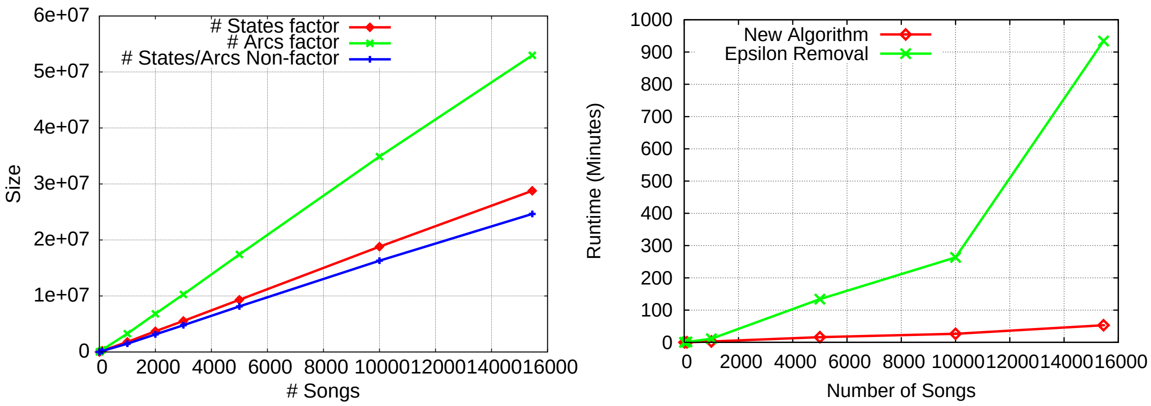
Direct comparisons to previous results are difficult because it is usually not possible for researchers to share music collections. However, anecdotally we can see that our system performs comparably to or better than some of the other systems in the literature. For example, [5] achieves perfect identification accuracy with a database of $10\,000$ songs on clean ten-second snippets but 80.3% and 93.7% accuracy on test conditions comparable to our Speed-1.02 and Speed-0.98, respectively.

C. Automata Size

Fig. 6(b) shows the weighted automaton $F_w(A)$ corresponding to the unweighted automaton $F(A)$ of Fig. 6(a). Note that $F_w(A)$ is no larger than $F(A)$. Remarkably, even in the case of $15\,455$ songs, the total number of transitions of $F_w(A)$ is 53.0M, only about 0.004% more than $F(A)$. We also have $|F(A)|_E \approx 2.1|A|_E$. As is illustrated in Fig. 11(a), this multiplicative relationship is maintained as the song set size is varied between 1 and $15\,455$. We have $|F_w(A)|_Q \approx 28.8M \approx 1.2|A|_Q$, meaning the bound of Proposition 2 is verified in this empirical context.

D. Suffix Automaton Algorithm Experiments

As previously mentioned, the method of Section II-C for constructing a compact factor transducer by adding ϵ-



(a) (b)

Fig. 11. (a) Comparison of automaton sizes for different numbers of songs. "#States/Arcs Non-factor" is the size of the automaton A accepting the entire song transcriptions. "# States factor" and "# Arcs factor" is the number of states and transitions in the weighted factor acceptor $F_w(A)$, respectively. (b) Runtime speeds for constructing $S(A)$ with ϵ-removal and the new suffix automaton algorithm.

transitions to A and applying weighted ϵ-removal, determinization, and minimization has at least a quadratic worst-case complexity. However, the novel weighted suffix automaton algorithm given in Section III-B can be used to construct the factor transducer T needed for the music identification system in linear time. As discussed in III-B, since acyclic automata can be minimized in linear time, the complexity advantage of the algorithm is demonstrated by applying the novel algorithm in place of ϵ-removal, determinization minimization. This algorithm operates on suffix-unique automata, and the automaton A representing the song transitions can easily be made suffix-unique by appending a unique symbol $\#_i$ to the end of each song transcription x_i. These placeholder symbols are ignored during the decoding process of the song identification algorithm.

Fig. 11(b) gives a comparison of the runtimes of both algorithms for varying sizes of our song set. When constructing a suffix automaton representing the entire collection of 15 455 songs, the new algorithm of section III-B runs in around 53 minutes, as compared to 934 minutes for the old algorithm using ϵ-removal and determinization. Furthermore, a clear non-linear runtime increase is exhibited by the ϵ-removal algorithm as the size of the song collection is increased.

E. Factor Uniqueness Analysis

We observed that our identification system performs well when test snippets of five seconds or longer are used. In fact, the accuracy is almost the same for ten-second snippets as when the full audio of the song is used. This encouraged us to analyze the sharing and repetition of similar audio segments among songs in our collection. A benefit of our music phoneme representation is that it reduces the task of locating audio similarity to that of finding repeated factors of the song transcriptions. More precisely, let two song transcriptions $x_1, x_2 \in U$ share a common factor $f \in \Sigma^*$ such that $x_1 = ufv$ and $x_2 = afc$; $u, v, a, c \in \Sigma^*$. Then the sections in these two songs transcribed by f are similar. Further, if a song x_1 has a repeated factor $f \in \Sigma^*$ such that $x_1 = ufvfw$; $u, v, w \in \Sigma^*$, then x_1 has two similar audio segments. If $|f|$ is large, then it is unlikely that the sharing of f is coincidental, and likely represents a repeated structural element in the song.

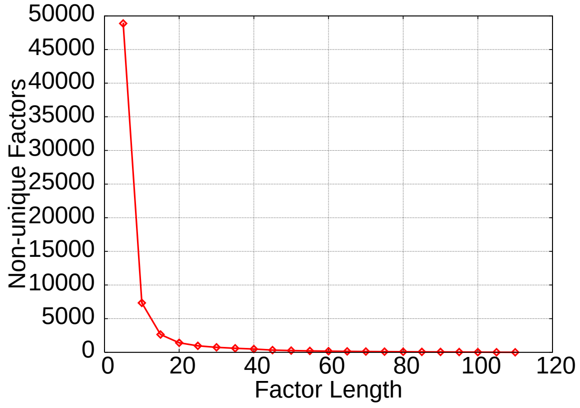


Fig. 12. Number of factors occurring in more than one song in S for different factor lengths.

Fig. 12 gives the number of non-unique factors over a range of lengths. This illustrates that some sharing of long elements is present, indicating similar music segments across songs. However, factor collisions decrease rapidly as the factor length increases. For example, out of the 24.4M existing factors of length 50, only 256 appear in more than one song. Considering that the average duration of a music phoneme in our experiments is around 200ms, a factor length of 50 corresponds to around ten seconds of audio, and in fact it is quite likely that these colliding ten-second snippets consist of primarily silence. This validates our initial estimate that ten seconds of music are sufficient to uniquely map the audio to a song in our database. In fact, even with factor length of 25 music phonemes, there are only 962 non-unique factors out of 23.9M total factors. This explains the fact that even a five-second snippet is sufficient for accurate identification.

V. Conclusion

We have described a music identification system based on Gaussian mixture models and weighted finite-state transducers and have shown it to be effective for identifying and detecting songs in the presence of noise and other distortions. The compact representation of the mapping of music phonemes to songs based on transducers allows for efficient decoding and high accuracy, even in the presence of noise and distortions.

We have given a novel algorithm for weighted suffix and factor automaton construction, which has a linear-time worst case complexity, a drastic improvement on the previous method using the generic ϵ-removal and determinization algorithms. This algorithm is a natural and essential extension of our previous unweighted algorithm [16] and matches our previous results guaranteeing the compactness of suffix and factor automata of automata. In this work we have applied this algorithm to our music identification system, and indeed in this setting it has exhibited an over 17-fold speedup over the previous method. Furthermore, this contribution is general and applicable to a number of other tasks where indexation of strings or sequences is required.

ACKNOWLEDGMENT

The authors thank C. Allauzen, M. Bacchiani, A. Berenzweig, M. Cohen, M. Riley, and J. Schalkwyk, for their help, advice, and support. The work of M. Mohri and E. Weinstein was partially supported by the New York State Office of Science Technology and Academic Research (NYSTAR).

REFERENCES

[1] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *Content-Based Multimedia Indexing (CBMI)*, Brescia, Italy, September 2001.

[2] Y. Ke, D. Hoiem, and R. Sukthankar, "Computer vision for music identification," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, June 2005, pp. 597–604.

[3] E. Batlle, J. Masip, and E. Guaus, "Automatic song identification in noisy broadcast audio," in *IASTED International Conference on Signal and Image Processing*, Kauai, Hawaii, 2002.

[4] A. L. Wang, "An industrial-strength audio search algorithm," in *International Conference on Music Information Retrieval (ISMIR)*, Washington, DC, October 2003.

[5] M. Covell and S. Baluja, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern Recognition*, vol. 41, November 2008.

[6] M. Casey, C. Rhodes, and M. Slaney, "Analysis of minimum distances in high-dimensional musical spaces," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 5, pp. 1015–1028, July 2008.

[7] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *Journal of VLSI Signal Processing Systems*, vol. 41, pp. 271–284, 2005.

[8] M. Mohri, F. C. N. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

[9] D. Pye, "Content-based methods for the management of digital music," in *ICASSP*, Istanbul, Turkey, June 2000, pp. 2437–2440.

[10] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *IEEE International Conference on Multimedia and Expo (ICME)*, Tokyo, Japan, August 2001.

[11] M.Bacchiani and M. Ostendorf, "Joint lexicon, acoustic unit inventory and model design," *Speech Communication*, vol. 29, pp. 99–114, November 1999.

[12] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[13] ——, "Statistical natural language processing," in *Applied Combinatorics on Words*, M. Lothaire, Ed. Cambridge University Press, 2005.

[14] C. Allauzen, M. Mohri, and M. Saraclar, "General indexation of weighted automata - application to spoken utterance retrieval," in *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2004), Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval.*, Boston, Massachusetts, May 2004, pp. 33–40.

[15] M. Mohri, P. Moreno, and E. Weinstein, "Factor automata of automata and applications," in *International Conference on Implementation and Application of Automata (CIAA)*, Prague, Czech Republic, July 2007.

[16] ——, "General suffix automaton construction algorithm and space bounds," *Theoretical Computer Science, To Appear*.

[17] D. Revuz, "Minimisation of acyclic deterministic automata in linear time," *Theoretical Computer Science*, vol. 92, pp. 181–189, 1992.

[18] M. Crochemore, "Transducers and repetitions," *Theoretical Computer Science*, vol. 45, no. 1, pp. 63–86, 1986.

[19] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. Chen, and J. Seiferas, "The smallest automaton recognizing the subwords of a text," *Theoretical Computer Science*, vol. 40, pp. 31–55, 1985.

[20] C. Alberti, M. Bacchiani, A. Bezman, C. Chelba, A. Drofa, H. Liao, P. Moreno, T. Power, A. Sahuguet, M. Shugrina, and O. Siohan, "An audio indexing system for election video material," in *ICASSP*, Taipei, Taiwan, 2009.

[21] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: a general and efficient weighted finite-state transducer library," in *12th International Conference on Implementation and Application of Automata (CIAA)*, Prague, Czech Republic, July 2007.

[22] A. Park and T. Hazen, "ASR dependent techniques for speaker identification," in *International Conference on Spoken Language Processing (ICSLP)*, Denver, Colorado, USA, September 2002.

[23] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[24] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[25] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

Mehryar Mohri Mehryar Mohri is a Professor of Computer Science at the Courant Institute and a Research consultant at Google Research. His current topics of interest are machine learning, theory and algorithms, text and speech processing, and computational biology. Prior to his current positions, he worked for about ten years at AT&T Labs - Research, formerly AT&T Bell Labs (1995-2004), where he served as the Head of the Speech Algorithms Department and as a Technology Leader, overseeing research projects in machine learning, text and speech processing, and the design of general algorithms.

Pedro Moreno Pedro J. Moreno is a research scientist at Google Inc. working in the New York office. His research interests are speech and multimedia indexing and retrieval, speech and speaker recognition and applications of machine learning to multimedia. Before joining Google Dr. Moreno worked in the areas of text processing, image classification, bioinformatics and robust speech recognition at HP labs where he was one of the lead researchers developing SpeechBot, one of the first audio search engines freely available on the web. He received a Ph.D. in electrical and computer engineering from Carnegie Mellon University and was a former Fullbright scholar.

Eugene Weinstein Eugene Weinstein is a Ph.D. candidate in the Computer Science Department of the Courant Institute at NYU, and a research intern at Google New York. His current interests are in machine learning, speech and music recognition, automata theory, and natural language processing. His dissertation research, which combines elements of each of these, is focused on enabling search of large collections of audio data. He received his M.Eng. and B.S. degree in Computer Science from MIT in 2001 and 2000, respectively.