Search Problems for Speech and Audio Sequences

by

Eugene Weinstein

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

September 2009

Mehryar Mohri—Advisor

To my family

Acknowledgments

To fully acknowledge the direct and indirect contributions of each one of my family, friends, and colleagues that has helped me move towards the successful completion of this thesis would surely require more pages than the technical content. The following is my attempt to thank those that have contributed the most, which is doubtlessly bound to result in an incomplete list.

Words cannot express my gratitude to my parents, Alla Bogomolnaya and Alexander Weinstein. By nurturing my curiosity from a very early age, my parents have instilled in me a passion for technology and its mathematical and scientific foundations. From this came my never-ending desire to tinker with computers, which eventually resulted in me pursuing my Ph.D. and completing this thesis.

Throughout the course of my graduate school experience I have benefited tremendously from the support and encouragement of my family and friends. I thank Wenjun Jing for being by my side and always believing in me throughout these four years. My sister, Ellen Weinstein, has been a daily source of support and encouragement. David Vener has offered me much-needed advice and help

in numerous tough situations. I thank also my step-parents and my extended family for taking care of me during this time.

My advisor, Mehryar Mohri, has taught me more than anybody other than my parents. Academically, I thank him for helping me develop a capacity and desire for scientific rigor, a thorough understanding of a number of theoretical and applied aspects of computer science, and an ability to evaluate critically my own work and that of others. Personally, I am indebted to him for pushing me to realize my potential as a researcher and for supporting me through the good and the difficult times of my Ph.D. years.

I also thank Pedro Moreno, my host at Google, for three years of fantastic collaboration. I have benefited immensely from his expertise in the foundations, design, and implementation of speech recognition systems. His guidance has helped me tremendously in the day-to-day grind of graduate student life.

I am also indebted to the other members of my thesis committee, Juan Pablo Bello, Ralph Grishman, and Dennis Shasha, who have taken the time to read my drafts and to give me thorough and helpful comments that have helped me significantly improve this thesis.

The nine years I spent at MIT before starting my Ph.D. gave me the solid foundation I needed to embark on a research career. Jim Glass advised me during my master's degree work and beyond, and I thank him for showing me the value and the excitement of research work. Anant Agarwal supervised me in my four years as a research staff member, and gave me an opportunity to work on a number of extremely exciting projects. Larry Rudolph's mentorship

helped me navigate the complexities of the academic world and to develop my own individual research mentality. Ken Steele and Jason Waterman were great collaborators in many efforts within the Oxygen project.

At NYU, I was fortunate to share an office with Ashish Rastogi, who not only helped me in many ways with my classes and research, but who also became a good friend. Afshin Rostamizadeh and Ameet Talwalkar were much farther away (next door), but nevertheless I will miss chatting with them about research and life during our coffee breaks.

At Google, I have enjoyed being part of an incredible team working on speech recognition products and research. Specifically, Cyril Allauzen, Michiel Bacchiani, Mike Cohen, Michael Riley, and Johan Schalkwyk all helped me substantially during the course of this work. In addition, Corinna Cortes was kind enough to take a personal interest in helping me navigate the corporate environment. I also thank Adam Berenzweig for providing the music collection on which the music identification experiments in this thesis were conducted.

I would also like to acknowledge my friends, who have all supported me tremendously throughout the course of this thesis. I was fortunate to have had the advice and support of Craig Springer and Igor Belakovskiy during the ups and downs of these four years. Jon Yi volunteered to read through my thesis draft and gave me very useful comments. While I cannot mention all my friends by name here, I am grateful to all of them for helping make this experience a fruitful and fun one. I was truly touched to see a number of my friends at my defense, and I thank them for having taken the time to attend.

Finally, I thank my running partner and furry companion, Max, for making my life more joyful over the past year.

Abstract

The modern proliferation of very large audio and video databases has created a need for effective methods of indexing and searching highly variable or uncertain data. Classical search and indexing algorithms deal with clean input sequences. However, an index created from speech or music transcriptions is marked with errors and uncertainties stemming from the use of imperfect statistical models in the transcription process. This thesis presents novel algorithms, analyses, and general techniques and tools for effective indexing and search that not only tolerate but exploit this uncertainty.

We have devised a new music identification technique in which each song is represented by a distinct sequence of music sounds, called "music phonemes." We learn the set of music phonemes, as well as a unique sequence of music phonemes characterizing each song, using an unsupervised algorithm. We also create a compact mapping of music phoneme sequences to songs. Using these techniques, we construct an efficient and robust large-scale music identification system.

We have further designed new algorithms for compact indexing of uncer-

tain inputs based on suffix and factor automata and given novel theoretical guarantees for their space requirements. We show that the suffix automaton or factor automaton of a set of strings U has at most $2Q - 2$ states, where Q is the number of nodes of a prefix-tree representing the strings in U. We also describe matching new linear-time algorithms for constructing the suffix automaton S or factor automaton F of U in time $O(|S|)$.

We have also defined a new quality measure for topic segmentation systems and designed a discriminative topic segmentation algorithm for speech inputs. The new quality measure improves on previously used criteria and is correlated with human judgment of topic-coherence. Our segmentation algorithm uses a novel general topical similarity score based on word co-occurrences. This new algorithm outperforms previous methods in experiments over speech and text streams. We further demonstrate that the performance of segmentation algorithms can be improved by using a lattice of competing hypotheses over the speech stream rather than just the one-best hypothesis as input.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The recent explosion in the quantities of multimedia data available on the Internet has precipitated the need for efficient and robust algorithms for indexing and searching such data. From campaign speeches to rock concerts, this audio and video content varies immensely in quality, purpose, and context. Accordingly, providing human access to these data in a valuable way has been a major challenge for both industry practitioners and academic researchers.

A rudimentary way of enabling search of audio content is by indexing the metadata associated with each particular recording. This can be compared roughly to indexing a large collection of PDF documents by merely using the filename of each document. It is clear that users will want to search the text found inside the PDF document in addition to the filename, so such an approach is unlikely to result in a satisfactory index. Analogously for audio collections, the user stands to benefit from an index of the content itself in

addition to that of just the metadata.

Meaningful search of audio content can be enabled by first transcribing the audio signal in terms of linguistic, acoustic, or conceptual units. These units can be words or phonemes in the case of speech, notes or chords for music, or other generic features computed over the audio signal to produce a fingerprint of the recording. We begin by motivating our foray into audio search with a theme that spans all the problems we shall be discussing in this thesis.

1.1 Motivation

As we shall see, the algorithms used to enable audio content search must overcome a primary challenge – that of uncertainty. Uncertainty can occur on the index creation and/or retrieval side, and the way in which it is managed directly affects the quality of the index. Correspondingly, this uncertainty leads to a number of challenging questions related to searching and indexing audio data. Here are some examples of such questions.

- Which units should be used to represent the audio recordings? For instance, can a polyphonic music sequence be represented with a single stream of notes? Can a one-best text hypothesis made by a speech recognizer be used to accurately index a collection of spoken audio recordings? Should the units be hand-crafted to correspond to some systematic representation of audio or should they be learned automatically?

- Which algorithms should be used to construct the index? Should this be

a deterministic process, i.e., computing frequency-domain features over an audio stream, or a probabilistic one, such as transcribing speech audio with the most likely word sequence according to a model, or a combination of both? If the transcription process is ambiguous (for example, the two best competing transcriptions for a single speech recording might be "recognize speech" and "wreck a nice beach"), what should be done differently in light of this ambiguity? In fact, can we leverage this ambiguity to discover an underlying structure to the audio data aggregated in our index, and thus improve the quality of our index?

- Which retrieval algorithm should be used? Even if the transcription or index construction algorithm is unambiguous, there is usually ambiguity exhibited on the retrieval side. For example, out of a potentially very large set of candidate audio recordings matching the search query, which should be selected and returned to the user? How do we disambiguate among multiple positive matches?

1.2 Summary of Contributions

In this work, we attempt to answer many of the above questions by presenting novel algorithms, analyses and techniques for enabling or improving audio search. As has been mentioned above, and as we shall see in detail over the length of this thesis, audio indexing is indexing in the presence of uncertainty. This directly leads to a number of research problems. In this thesis, we study

three such problems in depth.

- Music identification is content-based search of song databases. We have developed a new music identification approach, in which each song is represented by a distinct sequence of music sounds, called "music phonemes" in our work. 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. Finally, we propose a novel application of factor automata to map sequences of music phonemes to songs. Using these techniques, we construct a music identification system for a database of over 15,000 songs achieving an identification accuracy of 99.4% on undistorted test data, and performing robustly in the presence of noise and distortions.

- For the music search task above, as well as a number of other indexing scenarios, a crucial problem is constructing an efficient and scalable index of a set of strings. In the case of music search, the strings are sequences of music phonemes and the formalism used to model these sequences is a factor automaton. Suffix automata and factor automata are minimal deterministic automata representing the set of all suffixes or substrings of a set of strings. We present a novel analysis of the size of the suffix automaton or factor automaton of a set of strings. We show that the suffix automaton or factor automaton of a set of strings U has at most $2Q-2$ states, where Q is the number of nodes of a prefix-tree representing

the strings in U, a significant improvement over previous work. Our analysis suggests that the use of factor automata of automata can be practical for large-scale applications, a fact that is further supported by the results of our experiments applying factor automata in music identification. We also describe in detail a linear-time algorithm for constructing the suffix automaton S or factor automaton F of U in time $O(|S|)$. The linear size guarantee for the suffix and factor automata combined with this linear-time algorithm provides a solid and general algorithmic basis for the use of such automata for search tasks. For our music identification system, this novel algorithm helps speed up the construction of the weighted suffix automaton by a factor of 17 over the previous algorithms.

- Search of large collections of speech data can be enabled by transcribing the source audio in terms of linguistic units, such as phonemes or words. Discovering structural elements in the audio sequence can help us create an effective index as well as the improve the transcription quality. To this end, we make several contributions to the task of topic segmentation, or the automated discovery of topically-coherent segments in speech or text sequences. We give a new measure of topic segmentation quality that is more sound than previously used criteria, and is correlated with human judgment of topic-coherence. We then give two new topic segmentation algorithms which employ a new measure of text similarity based on word co-occurrence. Both algorithms function by finding

extrema in the similarity signal over the text, with the latter algorithm using a compact support-vector based description of a window of text or speech observations in word similarity space. In experiments over speech and text news streams, we show that our novel algorithm outperforms previous methods as measured both by our new quality measure, as well as the previously used evaluation techniques. We observe that topic segmentation of speech recognizer output is a more difficult problem than that of text streams; however, we demonstrate that by using a lattice of competing hypotheses rather than just the one-best hypothesis as input to the segmentation algorithm, the performance of the algorithm can be improved.

Chapter 2

Large-Scale Music Identification

2.1 Introduction

Automatic identification of music has been the subject of several recent studies both in research and industry. Music identification systems attempt to match a recording of a few seconds to 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 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. Our approach 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.

Since the size of the database is limited, another crucial task is *music detection*, that is determining if the recording supplied contains an in-set song. We demonstrate that our system performs accurately in the music detection task, even in the presence of noise or distortions sufficient to severely damage its performance in the identification task.

2.1.1 Previous Work

Much of the previous work on music identification (see [19] for a recent survey) is based on hashing of frequency-domain features. The features and hashing technologies used vary from work to work. Wang [80, 81] used energy peaks in the spectrogram of the recording to produce an audio signature to be hashed, and constructed a successful commercial music identification service that is still in use today. Haitsma et al. [34] used hand-crafted features of energy differences between Bark-scale cepstra. Ke et al. [43] used similar features, but selected them automatically using boosting. Covell et al. [24, 25] further improved on Ke by using wavelet features. Casey et al. [20] 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 [9] 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.

2.1.2 Summary of Contributions

In this chapter, 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 approach described here was published in [82, 58, 59]. 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 achieving an identification accuracy of 99.4% on undistorted test data, and performing robustly in the presence of noise and distortions. It allows us to index music event sequences in an optimal and compact way and with very rare false positive matches.

2.1.3 Outline

The remainder of this chapter is organized as follows. We begin with some notation and definitions of relevant automata concepts in Section 2.2. In Section 2.3, we describe our approach for automatically learning a set of elementary music sounds and Gaussian mixture acoustic models representing each sound. In Section 2.4, we cover our transducer representation for mapping music sound sequences to songs and the algorithms for constructing this transducer. Section 2.5 outlines some ways of improving the robustness of the system beyond the baseline system presented in the previous two sections. In Section 2.6, we describe our music detection algorithm using a universal background model (UBM) for music in conjunction with a support vector machine (SVM) classifier. In Section 2.7, we present our music identification and detection experiments on a database of 15,000 songs and in Section 2.8, we analyze the possibility of retrieving multiple matches for a given audio recording. We conclude in Section 2.9.

2.2 Preliminaries

We denote by Σ a finite alphabet. For a symbol $a \in \Sigma$, the notation a^*, a^+, and a^n denotes a repeated zero or more times, one or more times, and n times, respectively. This notation applies to Σ itself, that is Σ^*, Σ^+ and Σ^n denote zero or more, one or more, and n consecutive symbols from Σ, respectively. The length of a string $x \in \Sigma^*$ in number of symbols is denoted by $|x|$.

A number of the results presented in this thesis involve finite automata. Finite automata have been used in both theoretical and applied computer science from the very early days. For the purposes of this thesis, weighted automata and transducers shall be formally defined as follows. The symbol ϵ represents the empty string.

2.2.1 Weighted Finite Automata

Definition 2.1 *A* weighted automaton $(\Sigma, Q, E, I, F, \rho)$ *over the semiring* $(S, \oplus, \otimes, \overline{0}, \overline{1})$ *is specified by alphabet* Σ*, a finite set of states* Q*, a finite set of transitions* E*, a set of initial states* $I \in Q$*, a set of final states* $F \subseteq Q$*, and a final weight function* $\rho : F \mapsto S$*. The transition set* $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times S \times Q$ *associates pairs of states with a symbol in* $\Sigma \cup \{\epsilon\}$ *and a weight in* S*.*

Definition 2.2 *A* weighted finite state transducer $(\Sigma, \Delta, Q, E, I, F, \rho)$ *over the semiring* $(S, \oplus, \otimes, \overline{0}, \overline{1})$ *is specified by input and output alphabets* Σ *and* Δ*, respectively, a finite set of states* Q*, a finite set of transitions* E*, a set of initial states* $I \in Q$*, a set of final states* $F \subseteq Q$*, and a final weight function*

$\rho : F \mapsto S$. *The transition set* $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \Delta \cup \{\epsilon\} \times S \times Q$ *associates pairs of states with an input symbol, an output symbol, and a weight in* S.

An alternative notation for representing the set of transitions of a deterministic acceptor is to specify a partial transition function $\delta : Q \times \Sigma \cup \{\epsilon\} \mapsto Q$ and a weight function $\omega : Q \times \Sigma \cup \{\epsilon\} \mapsto S$ instead of explicitly through the set E. The corresponding notation applies also for deterministic transducers.

2.2.2 Weighted Automata and Semirings

The semiring $(S, \oplus, \otimes, \overline{0}, \overline{1})$ over which an acceptor or transducer is defined is a mathematical formalism that specifies the weight set used and the algebraic operations for combining weights. The operation $\oplus$ is used to combine the weights of multiple paths to obtain the total weight of a set of paths, and $\otimes$ is used to combine weights along a path in the computation of the total weight of the path. $\overline{0}$ and $\overline{1}$ are the identity elements for these operations, respectively.

Two semirings used extensively in fields such as speech and text processing are the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ and the log semiring $(\mathbb{R}_+ \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$, where $\oplus_{\log}$ is defined $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$. For many applications, these semirings are used when the weights are negative log-likelihoods. In both of these, the total weight along a given path is found by adding the weights of the transitions composing the path, corresponding to a product of the likelihoods. The two semirings differ in the way they combine weights across paths. In the log semiring, the total weight assigned

by the automaton to a string x is $\oplus_{\log}$ of the weights of all the paths in the automaton labeled with x, corresponding to the sum of the likelihoods of the paths. In contrast, in the tropical semiring, the total weight is that of the minimum weight (maximum likelihood) path labeled with x.

Given the above, it is clear that using the tropical semiring represents making the Viterbi approximation of the exact result, which is computed with the log semiring. This makes the tropical semiring a convenient and, indeed, invaluable tool for describing algorithms in speech and language processing [62], among many other fields. The tropical semiring is the one primarily used in this thesis.

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. More formally, an unweighted automaton is simply a weighted automaton over the Boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$.

2.2.3 Efficient Automata

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 automaton with a smaller number of states. An automaton is *ϵ-free* if it contains no ϵ-transitions. In order for search algorithms such as Viterbi decoding, among others, to function efficiently when processing a finite automaton, the automaton must be ϵ-free, deterministic and minimal.

Accordingly, such an automaton is often referred to as *efficient* or *optimal*. A deterministic and minimal automaton is, in fact, 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 ϵ-removal [54, 56], automata have become a compelling formalism used extensively in a number of fields, including speech, image, and language processing.

2.3 Acoustic Modeling

We begin the construction of our music identification system by applying an unsupervised algorithm to jointly learn 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 [9, 69, 50], 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.

2.3.1 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,

$$\begin{aligned} \mathrm{KL}_{sym}(G1, G2) &= 2\left(\mathrm{D}_{KL}(G_1 \| G2) + \mathrm{D}_{KL}(G_2 \| G_1)\right) \\ &= \mathrm{Tr}\left(\Sigma_2 \Sigma_1^{-1}\right) + \mathrm{Tr}\left(\Sigma_1 \Sigma_2^{-1}\right) \\ &\quad + (\mu_2 - \mu_1)^\top \left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)(\mu_2 - \mu_1) \\ &\quad - 2m \end{aligned} \tag{2.1}$$

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 above an experimentally determined threshold.

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 [8]. 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 [8] we use maximum likelihood as an objective distance function rather than the more common Euclidean distance.

2.3.2 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 initialization procedure just described. Since a single Gaussian is unlikely to accurately represent a complex music sound, we apply an iterative training algorithm to learn a standard Gaussian mixture acoustic model for each phoneme.

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 [9] in which the statistics 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 to find a transcription for each song.
- Refine the model with 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 Figure 2.1. 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.

2.3.3 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

$$C_i = \frac{1}{|S|} \sum_{s \in S} \mathrm{ED}(t_i(s), t_{i-1}(s)) \tag{2.2}$$

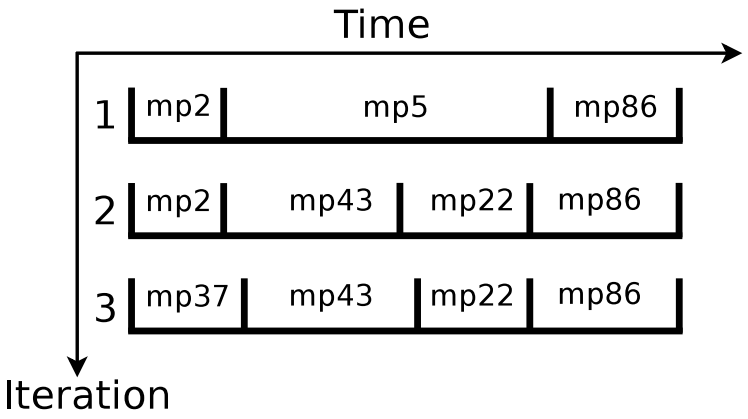


Figure 2.1: An illustration of changing transcription and alignment for a particular song during the course of three iterations of acoustic model training. mp$x$ stands for music phoneme number x and the vertical bars represent the temporal boundaries between music phonemes.

as our convergence measure.

Figure 2.1 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 $\mathrm{ED}(t_2(s), t_3(s)) = 1$.

Figure 2.2 shows how the edit distance changes during training for three phoneme inventory sizes. Note that, for example, with 1,024 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 1,700 phonemes, this means that only around half of the phonemes remained the same. In our experiments, convergence was exhibited after around twenty

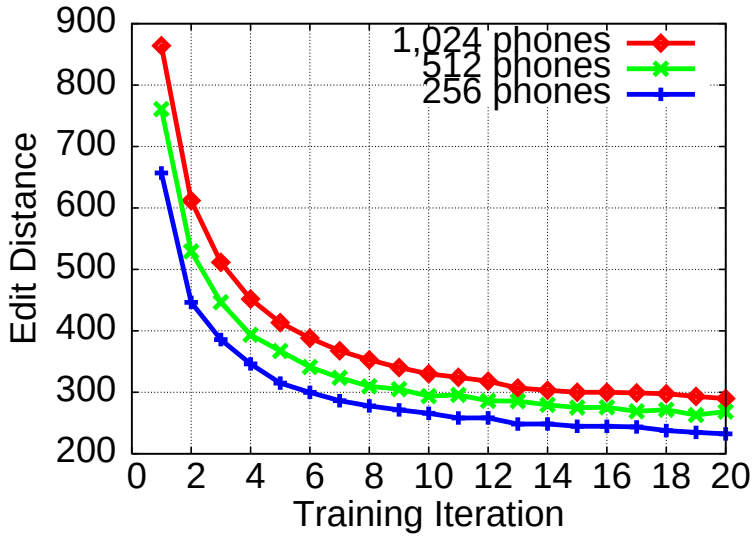


Figure 2.2: Average edit distance per song vs. training iteration.

iterations. In the last few iterations of training, the average edit distance decreases considerably to around 300, meaning around 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.

2.4 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 con-

struct 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 [62]. 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, $|\Sigma|$ = 1,024 and the average length of a transcription x_i is more than 1,700. Thus, in the worst case, there can be as many as $15{,}455 \times 1{,}700^2 \approx 45 \times 10^9$ 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.

2.4.1 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

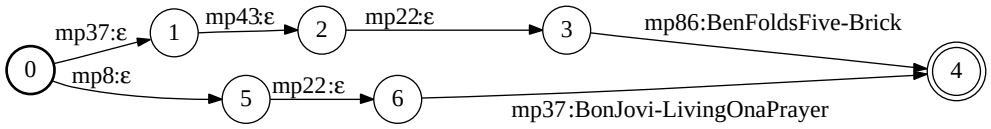


Figure 2.3: Finite-state transducer T_0 mapping each song to its identifier. $\texttt{mp}x$ stands for music phoneme number x.

of this section, we outline a method for constructing a factor automaton of an automaton using the general weighted determinization and minimization algorithms. 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., [6]). However, the novel suffix automaton algorithm we will give in Chapter 3 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. Figure 2.3 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 all factors of A, we want to accept all the symbol sequences labeling any path between any pair of paths in A. 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 $F(A)$ (Figure 2.4), but it does not allow us to output the song identifier associated with each factor.

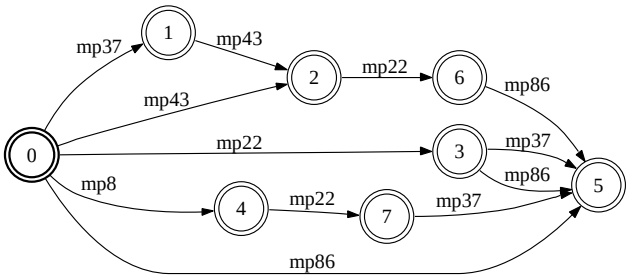


Figure 2.4: Deterministic and minimal unweighted factor acceptor $F(A)$ for two songs.

2.4.2 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 primary difficulty of constructing a factor transducer for music identification task is constructing an automaton where states and transitions can be shared among paths belonging to different songs, while preserving the mapping between phoneme sequences and songs.

2.4.3 Using Weights to Represent Song Labels

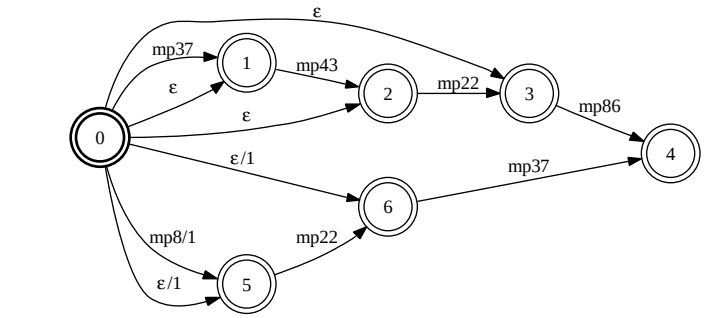
Our approach for avoiding the combinatorial explosion just mentioned is to use weights, instead of output labels, to represent song identifiers. We create a compact weighted acceptor over the tropical semiring accepting the factors of U that associates the total weight s_x to each factor x. During the 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 (Figure 2.3) we

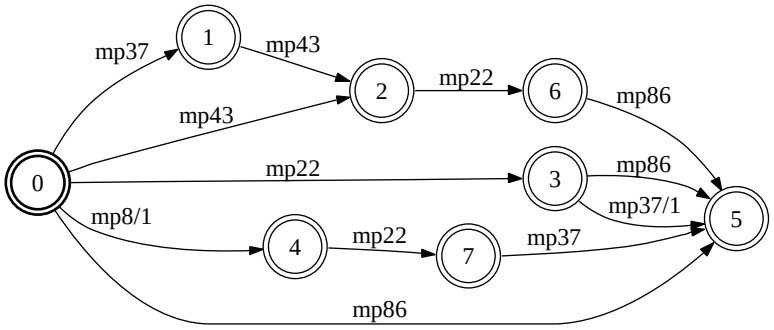
1. Drop the output labels to produce A.

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 $F_\epsilon(A)$ (Figure 2.5(a)).



(a)



(b)

Figure 2.5: (a) Factor acceptor $F_\epsilon(A)$ for two songs produced by adding weights and ϵ-transitions to A. (b) Deterministic and minimal weighted factor acceptor $F_w(A)$ produced by optimizing $F_\epsilon(A)$.

4. Apply ϵ-removal, determinization, and minimization to produce the weighted acceptor $F_w(A)$ (Figure 2.5(b)).

Recalling that a transition with no weight indicated has a weight of 0 in the tropical semiring, observe in Figure 2.5(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 `mp22 mp37` is correctly assigned a weight of 1 by $F_w(A)$. Observe finally that information about all the factors found in the original transducer T_0 (Figure 2.3) and their corresponding

songs is preserved.

The weights in the automaton $F_w(A)$ serve to represent the mapping from music phoneme sequences to songs, and hence do not correspond to likelihoods that can be used during the decoding process. Hence, $F_w(A)$ is transformed into an unweighted song recognition transducer T by treating each integer weight 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.

2.4.4 Weighted Transducer

As we have just mentioned, while the transducer T is constructed with the help of weights representing song identifiers, T itself is unweighted. This means that during decoding all transcription factors are equally likely. However, some phoneme sequences occur more frequently in our song collection and should thus be preferred during the Viterbi beam search. To accomplish this, we can view T as a weighted transducer in the *log semiring* (see Section 2.2) with zero weight on each path. The distribution of the weights along a given path can be changed (without affecting the total weight) by applying a weight pushing algorithm [56] to the transducer T. As a result, each prefix x of a path in T is weighted according to the frequency of snippets starting with x. Accordingly, more frequent factors are preferred during decoding. Our experiments show that this can result in a substantially higher accuracy for a given decoding speed.

2.4.5 Compact Representation

We have empirically verified the feasibility of the construction described in Section 2.4.3. 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 Chapter 3, 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.

2.5 Improving Robustness

When the recording quality of the test snippet is degraded, the recognized music phoneme sequence may contain errors. However, the transducer T, constructed as described above, accepts only factors of the exact transcriptions obtained during the last iteration of acoustic model training. To improve robustness, we may compose a transducer T_E with T that allows transcriptions with errors to also be accepted, resulting in the automaton $T \circ T_E$. One such corruption transducer T_E is the edit distance transducer, which associates a cost to each edit operation: deletion, insertion, or substitution [55]. In this case, the above composition allows edits to corrupt the input sequence x while penalizing any path allowing such corruptions in the Viterbi beam search algorithm. The costs may be determined analytically to reflect a desired set

of penalties, or may be learned to maximize identification accuracy.

Robustness can also be improved by including data reflecting the expected noise and distortion conditions in the acoustic model training process. The resulting models are then adapted to handle similar conditions in the test data.

2.6 Music Detection

Our music detection approach relies on the use of a universal background music phoneme model (UBM) model that generically represents all possible song sounds. This is similar to the techniques used in speaker identification (e.g., [65]). The UBM is constructed by combining the Gaussian mixture model components across all the music phonemes. We apply a divisive clustering algorithm to yield a desired number of mixture components.

To detect out-of-set songs, 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 that allows only the UBM. When the likelihood ratio of the two models is large, one can expect the song to be in the training set. However, a simple threshold on the likelihood ratio is not a powerful enough classifier for accurate detection. Instead, we use a discriminative method for out-of-set detection. We construct 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. These serve as the features for a support vector machine (SVM) classi-

fier [23, 79].

2.7 Experiments

In this section, we describe the experimental evaluation of our music identification and detection system in the context of a database of over 15,000 songs.

2.7.1 Experimental Setup

Our training data set consisted of 8,764 tracks from the `uspop` data set [11], as well as an additional 6,897 tracks downloaded from a self-publishing music web site. The collection included tracks from a total of 2,642 artists and spanned a number of non-classical genres including pop, rock, electronica, and jazz. A large fraction of the songs in the collection was from the nineties, but there was some coverage of the previous several decades, going as far back as the sixties, and some tracks recorded in the present decade. The average song duration was 3.9 minutes, for a total of over 1,000 hours of training audio.

The test data consisted of 1,762 in-set and 1,856 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 1,024 units, each Gaussian mixture acoustic model consisting of 16 mixture components. For the music detection exper-

iments, we also used a UBM with 16 components. All experiments run in 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 (i.e., 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 [9].

3. MP3-x: mp3 reencode at x kbps (using `lame`). This simulates compression or transmission at a lower bitrate.

For the detection experiments we used the `LIBSVM` implementation [21] with a radial basis function (RBF) kernel. The accuracy was measured using 10-fold cross-validation and a grid search for the values of γ in the RBF kernel and the trade-off parameter C of support vector machines [23, 79].

2.7.2 Results and Discussion

The identification and detection accuracy results are presented in Table 2.1. The identification performance is almost flawless on clean data. The addition

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 2.1: Identification accuracy rates under various test conditions

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.

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 speeding up or slowing down the audio. This shift results in a mismatch between the audio data and the acoustic models.

2.8 Factor Uniqueness Analysis

We observed that our identification system performs well when snippets of five seconds or longer are used. Indeed, there is almost no improvement when the snippet length increases from ten seconds to the full song. To further analyze this, we examined the sharing of factors across songs. Let two song transcriptions $x_1, x_2 \in S$ share a common factor $f \in \Sigma^*$ such that $x_1 = ufv$ and $x_2 = wfz$; $u, v, w, z \in \Sigma^*$. 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.

Figure 2.6 gives the number of factors occurring in more than one song over a range of lengths. This illustrates that some sharing of long elements is present, indicating similar music segments, or non-music elements such as silence, across songs. However, factor collisions decrease rapidly as the factor length increases. For example, we can see that for factor length of 50, only 256 out of the 24.4M existing factors 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. 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

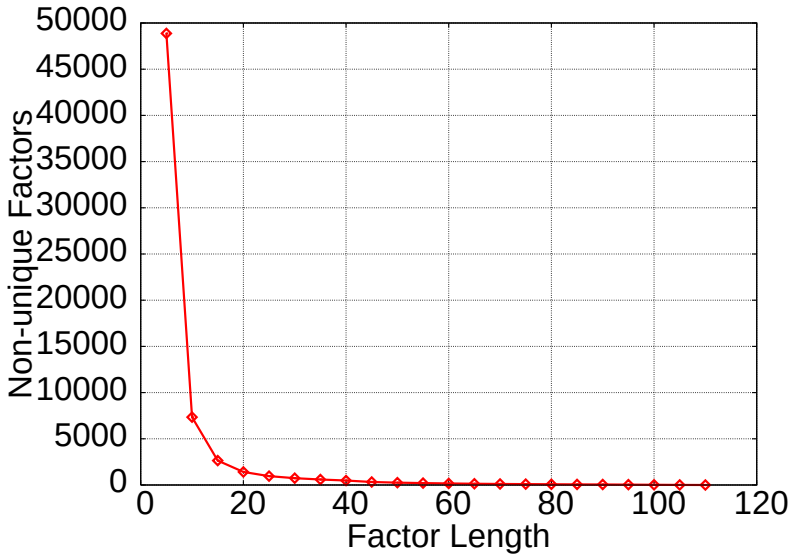


Figure 2.6: Number of factors occurring in more than one song in S for different factor lengths.

factors. This explains the fact that even a five-second snippet is sufficient for accurate identification.

2.9 Summary

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.

By making these observations about the compactness of our representation,

we have set the stage for the theoretical and algorithmic work we undertake in Chapter 3. There we will not only show that this representation is guaranteed to scale gracefully as the size of the music database grows but we will give novel algorithms that allow efficient construction of the type of music index described in this chapter.

Finally, we have provided an empirical analysis of factor uniqueness across songs. This analysis has verified that five-second or longer song snippets are sufficient for very infrequent factor collision and thus accurate identification.

Chapter 3

Theory and Algorithms for Suffix and Factor Automata

3.1 Introduction

Searching for patterns in massive collections of natural language texts, biological sequences, and other widely accessible digitized sequences is a problem of central importance in computer science. The problem has a variety of applications and has been extensively studied in the past [33, 27].

This chapter considers the problem of constructing a full index, or inverted file, to enable search of a collection of strings. When the number of strings is large, such as thousands or even millions, the set of strings can be compactly stored as an automaton, which also enables efficient implementations of search and other algorithms [6, 82]. In fact, in many contexts such as speech recogni-

tion or information extraction tasks, the entire set of strings is often directly given as an automaton. In these settings, the set of strings to be indexed is a collection of uncertain alternative representations of content such as speech or music transcriptions (see e.g., Section 4.4). Accordingly, each string in this set is often associated with a weight denoting a probability or a cost. The results in this chapter hold for both weighted and unweighted automata.

An efficient and compact data structure for representing a full index of a set of strings is a *suffix automaton*, a minimal deterministic automaton representing the set of all suffixes of a set of strings. Since a substring is a prefix of a suffix, a suffix automaton can be used to determine if a string x is a substring in time linear in its length $O(|x|)$, which is optimal. Additionally, just as suffix trees, suffix automata have other interesting properties in string-matching problems which make their use and construction attractive [33, 27]. Another similar data structure for representing a full index of a set of strings is a *factor automaton*, a minimal deterministic automaton representing the set of all factors or substrings of a set of strings. Factor automata offer the same optimal linear-time search property as suffix automata, and are strictly never larger.

3.1.1 Previous Work

The size of suffix and factor automata of a single string has been analyzed by Crochemore et al. [26] and Blumer et al. [14, 16]. In parallel results, these authors demonstrated that, remarkably, the size of the factor automaton of a

string x is linear. More precisely, the factor automaton for a string x of length more than three has at most $2|x| - 2$ states and $3|x| - 4$ transitions. These papers also gave matching on-line linear-time algorithms for constructing a factor automaton given a single string as input. Similar results were given for suffix automata, the minimal deterministic automata accepting exactly the set of suffixes of a string.

The construction and the size of the factor automaton of a finite set of strings $U = \{x_1, \ldots, x_m\}$ has also been previously studied [15, 16]. This work showed that an automaton accepting all factors of U can be constructed that has at most $2\|U\| - 1$ states and $3\|U\| - 3$ transitions, where $\|U\|$ is the sum of the lengths of all strings in U, that is $\|U\| = \sum_{i=1}^{m} |x_i|$.

3.1.2 Motivation

The original motivation for the work presented in this chapter was the design of the large-scale music identification system described in Chapter 2, where we represented our song database by a factor automaton. As mentioned in Section 2.4.5, the automaton representation for songs was empirically compact for our music collection of over 15,000 songs. Nonetheless, we wanted to ensure that our approach would scale to a larger set of songs, e.g., several million songs. Hence we set out to formally analyze the properties of suffix and factor automata of automata, give bounds on their size, and devise efficient algorithms for their construction. While our music identification system was the original motivation for this work, the results presented in this chapter are

general and applicable to a number of indexation tasks in a variety of fields.

3.1.3 Summary of Contributions

In this chapter, we prove a bound on the size of the suffix automaton or factor automaton of a set of strings that is significantly better than the bounds found in previous work. We show that the suffix automaton or factor automaton of a set of strings U has at most $2Q-2$ states, where Q is the number of nodes of a prefix-tree representation of the strings in U. The number of nodes Q can be dramatically smaller than $\|U\|$, the sum of the lengths of all strings. Thus, our space bound clearly improves on previous work [15]. More generally, we give novel bounds for the size of the suffix automaton or factor automaton of an acyclic finite automaton as a function of the size of the original automaton and the maximal length of a suffix shared by the strings accepted by the original automaton. This result can be compared to that of Inenaga et al. for compact directed acyclic word graphs whose complexity, $O(|\Sigma|Q)$, depends on the size of the alphabet [42].

Using our space bound analysis, we also give a simple algorithm for constructing the suffix automaton S or factor automaton F of U in time $O(|S|)$ from a prefix tree representing U. Our algorithm applies in fact to any weighted or unweighted input *suffix-unique automaton* and strictly generalizes the standard on-line construction of a suffix automaton for a single input string. The algorithm is easy to implement and is very efficient in practice. The linear size guarantee for the suffix and factor automata presented in this chapter,

combined with this linear-time algorithm provides a solid and general algorithmic basis for the use of such automata for search tasks. For the music identification system described in Chapter 2, this novel algorithm helps speed up the construction of the weighted suffix automaton by a factor of 17 over the previous algorithms.

The bounds and algorithms presented in this chapter were published in [57, 59, 60].

3.1.4 Outline

The remainder of the chapter is organized as follows. Section 3.2 introduces string and automata definitions and terminology used throughout the chapter. Section 3.3 develops the concept of suffix-uniqueness and some important properties of suffix automata. In Section 3.4, we continue our analysis, which culminates in new bounds on the size of the suffix automaton and factor automaton of an automaton. Section 3.5 gives a detailed description of a linear-time algorithm for the construction of the suffix automaton and factor automaton of a finite set of strings, or of any suffix-unique unweighted automaton, including pseudocode of the algorithm. In Section 3.6, we extend this algorithm to produce a weighted suffix or factor automaton of an input weighted suffix-unique automaton. Section 3.7 describes the use of factor automata in music identification and reports several empirical results related to their size. We conclude in Section 3.8.

3.2 Preliminaries

Section 2.2 reviews the fundamental definitions of finite automata and related concepts. In this section, we introduce some additional notation that will be required in this chapter.

A *factor*, or *substring*, of a string $x \in \Sigma^*$ is a sequence of symbols appearing consecutively in x. Thus, y is a factor of x iff there exist $u, v \in \Sigma^*$ such that $x = uyv$. A *suffix* of a string $x \in \Sigma^*$ is a factor that appears at the end of x. Put otherwise, y is a suffix of x iff there exists $u \in \Sigma^*$ such that $x = uy$. y is a *proper suffix* of x, denoted by $y <_s x$, iff y is a suffix of x and $|y| < |x|$. The notation $y \not<_s x$ denotes that y is not a suffix of x.

Analogously, y is a *prefix* of x iff there exists $u \in \Sigma^*$ such that $x = yu$. More generally, a factor, suffix, or prefix of a set of strings U or an automaton A, is a factor, suffix, or prefix of a string in U or a string accepted by A, respectively. The symbol ϵ represents the empty string. By convention, for any string $x \in \Sigma^*$, ϵ is always a prefix, suffix, and factor of x. An equivalence relation $\equiv$ on Σ is referred to as a *right-invariant equivalence relation* if for any $x, y, z \in \Sigma^*$, $x \equiv y \Rightarrow xz \equiv yz$. Finally, $1_{m=0}$ is the indicator function that is 1 when $m = 0$ and 0 otherwise.

3.3 Factors of a Finite Automaton

This section develops the notion of a suffix-unique automaton and covers some key properties of suffix automata, generalizing similar observations made by

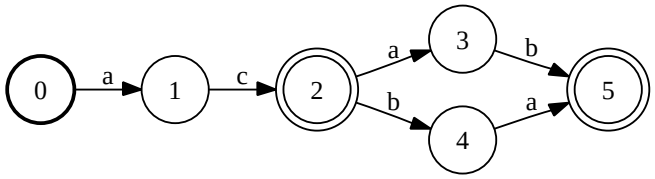


Figure 3.1: Finite automaton A accepting the strings $ac, acab, acba$.

Blumer et al. (1985) for a single string [14].

3.3.1 Suffix Uniqueness of Automata

In some applications such as music identification the strings considered may be long, e.g., sequences of music sounds, but with relatively short common suffixes. This motivates the following definition.

Definition 3.1 *Let k be a non-negative integer. We will say that 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 $k = 1$.*

Figure 3.1 gives an example of a simple automaton A accepting three strings ending in distinct symbols. Note that A is suffix-unique.

The main results of this chapter hold for suffix-unique automata, but we also present some results for the general case of arbitrary acyclic automata. Recall from Chapter 2 that we denote by $F(A)$ the minimal deterministic automaton accepting the set of factors of a finite automaton A. Similarly, we denote by $S(A)$ the minimal deterministic automaton accepting the set of suffixes of an automaton A.

3.3.2 Properties of Suffix-unique Automata

Definition 3.2 *Let A be a finite automaton. For any string $x \in \Sigma^*$, we define end-set(x) as the set of states of A reached by the paths in A that begin with x. We say that two strings x and y in Σ^* are equivalent and denote this by $x \equiv y$, when end-set(x) = end-set(y). This defines a right-invariant equivalence relation on Σ^*. We denote by $[x]$ the equivalence class of $x \in \Sigma^*$.*

Lemma 3.1 *Assume that A is suffix-unique. Then, a non-suffix factor x of the automaton A is the longest member of $[x]$ iff it is either a prefix of A, or both ax and bx are factors of A for distinct $a, b \in \Sigma$.*

Proof. Let x be a non-suffix factor of A. Clearly, if x is not a prefix, then there must be distinct a and b such that ax and bx are factors of A, otherwise $[x]$ would admit a longer member. Conversely, assume that ax and bx are both factors of A with $a \neq b$. Let y be the longest member of $[x]$. Let q be a state in *end-set*(x) = *end-set*(y). Since x is not a suffix, q is not a final state, and there exists a non-empty string z labeling a path from q to a final state. Since A is suffix-unique, both xz and yz are suffixes of the same string. Since y is the longest member of $[x]$, x must be a suffix of y. Since ax and bx are both factors of A with $a \neq b$, we must have $y = x$. Finally, if x is a prefix, then clearly it is the longest member of $[x]$. □

Proposition 3.1 *Assume that A is suffix-unique. Let $S_A = (Q_S, I_S, F_S, E_S)$ be the deterministic automaton whose states are the equivalence classes $Q_S =$*

$\{[x] \neq \emptyset \colon x \in \Sigma^*\}$, *its initial state* $I_S = \{[\epsilon]\}$, *its final states* $F_S = \{[x] \colon \textit{end-set}(x) \cap F_A \neq \emptyset\}$ *where* F_A *is the set of final states of* A, *and its transition set* $E_S = \{([x], a, [xa]) \colon [x], [xa] \in Q_S\}$. *Then,* S_A *is the minimal deterministic suffix automaton of* A: $S_A = S(A)$.

Proof. By construction, S_A is deterministic and accepts exactly the set of suffixes of A. Let $[x]$ and $[y]$ be two equivalent states of S_A. Then, for all $z \in \Sigma^*$, $[xz] \in F_S$ iff $[yz] \in F_S$, that is z is a suffix of A iff yz is a suffix of A. Since A is suffix-unique, this implies that either x is a suffix of y or vice-versa, and thus that $[x] = [y]$. Thus, S_A is minimal. □

In what follows, we will be interested in the case where the automaton A is acyclic. We denote by $|A|_Q$ the number of states of A, by $|A|_E$ the number of transitions of A, and by $|A|$ the *size of* A defined as the sum of the number of states and transitions of A.

3.3.3 Suffix and Factor Automaton Construction Using General Automata Algorithms

When A accepts only a single string, there are standard algorithms for constructing $S(A)$ and $F(A)$ from A in linear time [14, 26]. In the general case, $S(A)$ can be constructed from A as follows:

1. Add an ϵ-transition from the initial state of A to each state of A (Figure 3.2(a)).

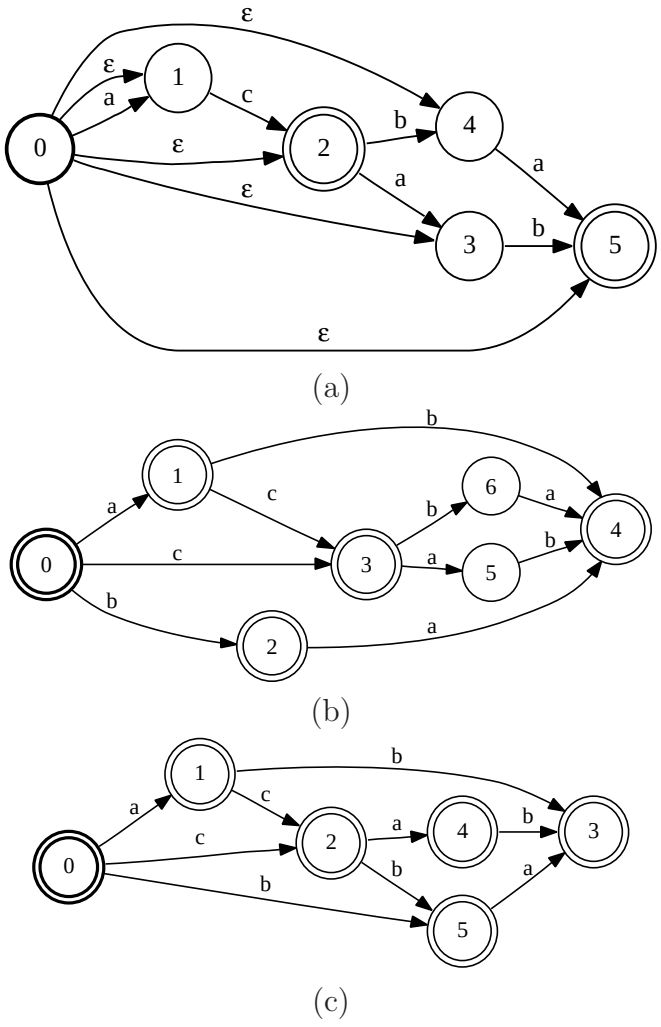


Figure 3.2: Construction of the suffix automaton using general automata algorithms. (a) The automaton A_ϵ produced by adding ϵ-transitions to automaton A of Figure 3.1. (b) Suffix automaton $S(A)$ of the automaton A produced by applying ϵ-removal, determinization and minimization to A_ϵ. (c) Factor automaton $F(A)$ produced from $S(A)$ by making every state final and minimizing.

2. Apply an ϵ-removal algorithm, followed by determinization and minimization (Figure 3.2(b)).

$F(A)$ (Figure 3.2(c)) can be obtained similarly by further making all states final before applying ϵ-removal, determinization, and minimization. It can also be obtained from $S(A)$ by making all states of $S(A)$ final and applying minimization. Note that in the example given in Figure 3.2, $|F(A)|_Q = |S(A)|_Q - 1$. This is because after state 6 of $S(A)$ is made final, it becomes equivalent to state 2, and the two states are combined during minimization to make state 5 of $F(A)$.

3.4 Space Bounds for Factor Automata

In this section, we derive new bounds on the size of $S(A)$ and $F(A)$ in the case of interest for our applications where A is an acyclic automaton, typically deterministic and minimal.

3.4.1 Size of Suffix and Factor Automata

When A represents a single string x, the size of the automata $S(A)$ and $F(A)$ can be proved to be linear in $|x|$. More precisely, the following bounds hold for $|S(A)|$ and $|F(A)|$ [26, 14]:

$$\begin{array}{rclrcl} |S(A)|_Q & \leq & 2|x| - 1 & |S(A)|_E & \leq & 3|x| - 4 \\ |F(A)|_Q & \leq & 2|x| - 2 & |F(A)|_E & \leq & 3|x| - 4. \end{array} \tag{3.1}$$

These bounds are tight for strings of length more than three. [15] gave similar results for the case of a set of strings U by showing that the size of the factor automaton $F(U)$ representing this set is bounded as follows

$$|F(U)|_Q \leq 2\|U\| - 1 \quad |F(U)|_E \leq 3\|U\|_E - 3, \tag{3.2}$$

where $\|U\|$ denotes the sum of the lengths of all strings in U.

In general, the size of an acyclic automaton A representing a finite set of strings U can be substantially smaller than $\|U\|$. In fact, $|A|$ can be exponentially smaller than $\|U\|$. Thus, we are interested in bounding the size of $S(A)$ or $F(A)$ in terms of the size of A, rather than the sum of the lengths of all strings accepted by A.

3.4.2 Suffix Subset Inclusion

Our size bound for suffix automata relies on an analysis of the connection between the suffix sets of $S(A)$ with the states of A. We formalize this relationship with the following notation. For any state q of $S(A)$, we denote by $\text{suff}(q)$ the set of strings labeling the paths from q to a final state. We also denote by $N(q)$ the set of states in A from which a path labeled with a non-empty string in $\text{suff}(q)$ reaches a final state. See Figure 3.3 for an illustration.

Lemma 3.2 *Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then*

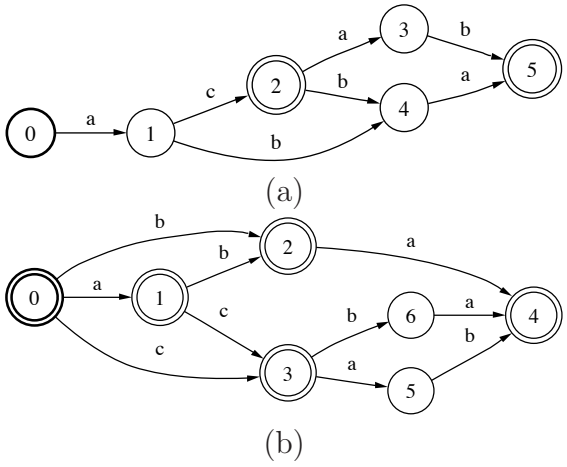


Figure 3.3: (a) An automaton A. (b) The corresponding suffix automaton $S(A)$. To illustrate the notation $\mathrm{suff}(q)$ and $N(q)$, note that starting from state 3 in $S(A)$ and reading the strings ab and ba we arrive at a final state. Hence, $\mathrm{suff}(3) = \{ab, ba\}$. The set of states in A from which ab and/or ba can be read to reach a final state is $N(3) = \{2, 1\}$.

$$\big(\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \quad \textit{and} \quad N(q) \subseteq N(q')\big) \textit{ or}$$
$$\big(\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \quad \textit{and} \quad N(q') \subseteq N(q)\big). \tag{3.3}$$

Proof. Since $S(A)$ is a minimal automaton, its states are accessible from the initial state. Let u be the label of a path from the initial I of $S(A)$ to q and similarly u' the label of a path from I to q'. By assumption, there exists $p \in N(q) \cap N(q')$. Thus, there exist non-empty strings $v \in \mathrm{suff}(q)$ and $v' \in \mathrm{suff}(q')$ such that both v and v' label paths from p to a final state. See Figure 3.4 for an illustration.

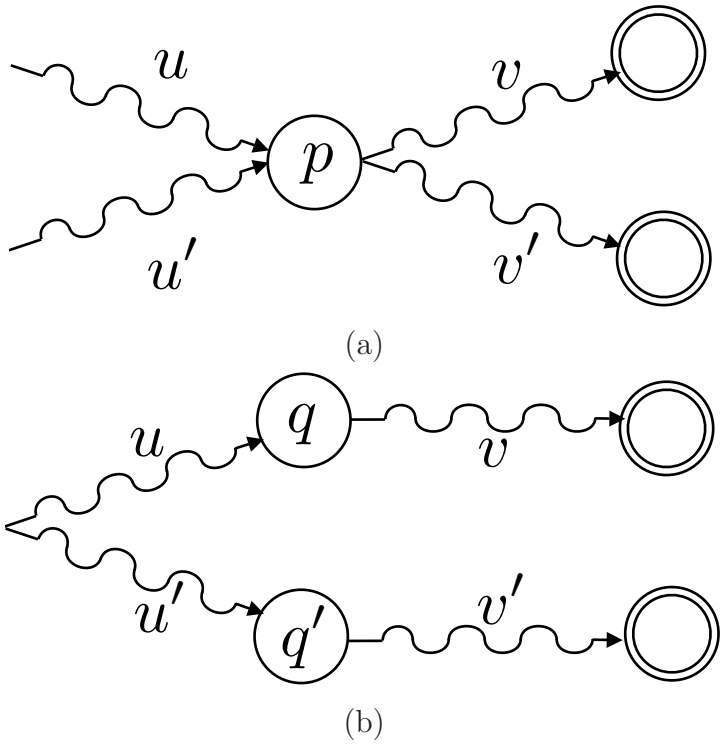


Figure 3.4: Illustration 1 of Lemma 3.2. (a) An automaton A with paths $uv, uv', u'v$ and $u'v'$ all going through state p. (b) The corresponding suffix automaton $S(A)$ where uv goes through q and $u'v'$ goes through q'.

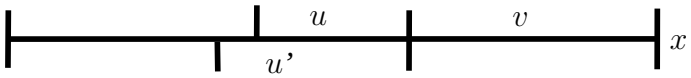


Figure 3.5: Illustration 2 of Lemma 3.2. uv and $u'v$ are suffixes of the same string x. Thus, u and u' are also suffixes of the same string. Thus, u is a suffix of u' or vice-versa.

By definition of u and u', both uv and $u'v'$ are suffixes of A. Since A is suffix-unique and v is non-empty, there exists a unique string accepted by A and ending with v. There exists also a unique string accepted by A and ending with uv. Thus, these two strings must coincide.

This implies that any string accepted by A and admitting v as suffix also admits uv as suffix. In particular, the label of any path from an initial state to p must admit u as suffix. Reasoning in the same way for v' we conclude that the label of any path from an initial state to p must also admit u' as suffix. Thus, u and u' are suffixes of the same string. Thus, u is a suffix of u' or vice-versa. Figure 3.5 illustrates this situation.

Assume without loss of generality that u is a suffix of u'. Then, for any string w, if $u'w$ is a suffix of A so is uw. Thus, $\mathrm{suff}(q') \subseteq \mathrm{suff}(q)$, which implies $N(q') \subseteq N(q)$. When u' is a suffix of u, we obtain similarly the other case of the statement of the lemma. □

Note that Lemma 3.2 holds even when A is a non-deterministic automaton.

Lemma 3.3 *Let A be a suffix-unique deterministic automaton and let q and q' be two distinct states of $S(A)$ such that $N(q) = N(q')$, then either q is a final state and q' is not, or q' is a final state and q is not.*

Proof. Assume that $N(q) = N(q')$. By Lemma 3.2, this implies $\text{suff}(q) = \text{suff}(q')$. Thus, the same non-empty strings label the paths from q to a final state or the paths from q' to a final state. Since $S(A)$ is a minimal automaton, the distinct states q and q' are not equivalent. Thus, one must admit an empty path to a final state and not the other. □

3.4.3 Main Size Bound

The following proposition extends the results of [15] which hold for a suffix automaton of a set of strings U, to the case where U is given as an automaton A.

Proposition 3.2 *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 automaton of A is bounded as follows*

$$|S(A)|_Q \leq 2|A|_Q - 3. \tag{3.4}$$

Proof. If the strings accepted by A are all of the form a^n, $S(A)$ can be derived from A simply by making all its states final and the bound is trivially achieved. In the remainder of the proof, we can thus assume that not all strings accepted by A are of this form.

Let F be the unique final state of $S(A)$ with no outgoing transitions. Lemmas 3.2-3.3 help define a tree T associated to all states of $S(A)$ other than F

by using the ordering:

$$N(q) \sqsubseteq N(q') \text{ iff } \begin{cases} N(q) \subset N(q') \text{ or} \\ N(q) = N(q') \text{ and } q' \text{ final}, q \text{ non-final.} \end{cases} \tag{3.5}$$

We will identify each node of T with its corresponding state in $S(A)$. By Proposition 3.1, each state q of $S(A)$ can also be identified with an equivalence class $[x]$. Let q be a state of $S(A)$ distinct from F, and let $[x]$ be its corresponding equivalence class. Observe that since A is suffix-unique, *end-set*(x) coincides with $N(q)$.

We will show that the number of nodes of T is at most $2|A|_Q - 4$, which will yield the desired bound on the number of states of $S(A)$. To do so, we bound separately the number of non-branching and branching nodes of T.

Non-Branching Nodes

Let q be a node of T and let $[x]$ be the corresponding equivalence class, with x its longest member. The children of q are the nodes corresponding to the equivalence classes $[ax]$ where $a \in \Sigma$ and ax is a factor of A.

By Lemma 3.1, if x is a non-suffix and non-prefix factor, then for some $a, b \in \Sigma$ such that $a \neq b$, there exist factors ax and bx. Thus, q admits at least two children corresponding to $[ax]$ and $[bx]$ and is thus a branching node. Thus non-branching nodes can only be either nodes q where x is a prefix, or those where x is a suffix, that is when q is a final state of $S(A)$.

Since the strings accepted by A are not all of the form a^n for some $a \in \Sigma$, the

empty prefix ϵ occurs at least in two distinct left contexts a and b with $a \neq b$. Thus, the prefix ϵ, which corresponds to the root of T, is necessarily branching. Also, let f be the unique final state of A with no outgoing transitions. The equivalence class of the longest factor ending in f, that is the longest string accepted by A, corresponds to the state F in $S(A)$, which is not included in the tree T. Thus, there are at most $|A|_Q - 2$ non-branching prefixes.

There can be at most one non-branching node for each string accepted by A. Let N_{str} denote the number of strings accepted by A, then, the number of non-branching nodes N_{nb} of T is at most $N_{nb} \leq |A|_Q - 2 + N_{str}$.

Branching Nodes

To bound the number of branching nodes N_b of T, observe that since A is suffix-unique, each string accepted by A must end with a distinct symbol a_i, $i = 1, \ldots, N_{str}$. Each a_i represents a distinct left context for the empty factor ϵ, thus the root node $[\epsilon]$ admits all $[a_i]$, $i = 1, \ldots, N_{str}$, as children. Let T_{a_i} represent the sub-tree rooted at $[a_i]$ and let n_{a_i} represent the number of leaves of T_{a_i}. Let a_j, $j = N_{str} + 1, \ldots, N_{str} + m$ denote the other children of the root and let T_{a_j} denote each of the corresponding sub-tree. See Figure 3.6 for an illustration. A tree with n_{a_i} leaves has fewer than n_{a_i} branching nodes. Thus, the number of branching nodes of T_{a_i} is at most $n_{a_i} - 1$. The total number of leaves of T is at most the number of disjoint subsets of Q excluding the initial state and f.

Note, however, that when the root node $[\epsilon]$ admits only $[a_i]$s, $i = 1, \ldots, N_{str}$,

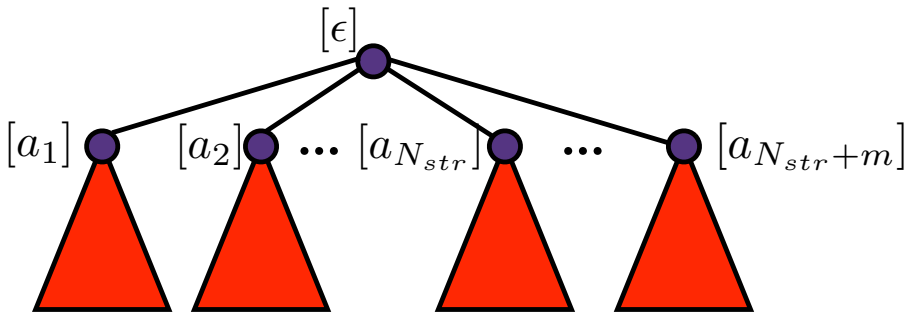


Figure 3.6: Branching nodes in the suffix class hierarchy. The root of the hierarchy is $[\epsilon]$, and its children are $[a_1], \ldots, [a_{N_{str}+m}]$. $[a_i]$, $i = 1, \ldots, N_{str}$ are the equivalence classes identified by the distinct final symbols of each string accepted by A. The other children of $[\epsilon]$ are denoted as $[a_j]$, $j = N_{str} + 1, \ldots, N_{str} + m$.

as children, that is when $m = 0$, then there is at least one a_i, say a_1, that is also a prefix of A since any other symbol would have been the root node's child. The node a_1 will then have also a child since it corresponds to a suffix or final state of $S(A)$. Thus, a_1 cannot be a leaf in that case. Thus, there are at most as many as $\sum_{i=1}^{N_{str}+m} n_{a_i} \leq |A|_Q - 2 - 1_{m=0}$ leaves and the total number of branching nodes of T, including the root is at most $N_b \leq \sum_{i=1}^{N_{str}+m}(n_{a_i} - 1) + 1 \leq |A|_Q - 2 - 1_{m=0} - (N_{str} + m) + 1 \leq |A|_Q - 2 - N_{str}$. The total number of nodes of the tree T is thus at most $N_{nb} + N_b \leq 2|A|_Q - 4$.

□

In the specific case where A represents a single string x, the bound of Proposition 3.2 matches that of [26] or [14] since $|A|_Q = |x| + 1$. The bound of Proposition 3.2 is tight for strings of length more than three and thus is also tight for automata accepting strings of length more than three. Note that the

automaton of Figure 3.1 is suffix-unique, deterministic, and minimal and has $|A|_Q = 6$ states. The number of states of the minimal suffix automaton of A (Figure 3.2(b)) is $|S(A)|_Q = 7 < 2|A|_Q - 3$.

3.4.4 Implications of Size Bound

Corollary 3.1 *Let A be a suffix-unique deterministic and minimal automaton accepting strings of length more than three. Then, the number of states of the factor automaton of A is bounded as follows*

$$|F(A)|_Q \leq 2|A|_Q - 3. \tag{3.6}$$

Proof. As mentioned in Section 3.3.3, a factor automaton $F(A)$ can be obtained from a suffix automaton $S(A)$ by making all states final and applying minimization. Thus, $|F(A)| \leq |S(A)|$. The result follows from Proposition 3.2. □

Blumer et al. (1987) showed that an automaton accepting all factors of a set of strings U has at most $2\|U\| - 1$ states, where $\|U\|$ is the sum of the lengths of all strings in U [15]. The following gives a significantly better bound on the size of the factor automaton of a set of strings U as a function of the number of nodes of a prefix-tree representing U, which is typically substantially smaller than $\|U\|$.

Corollary 3.2 *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 factor automaton $F(U)$ and that of the suffix automaton $S(U)$ of the strings of U are bounded as follows

$$|F(U)|_Q \leq 2|A|_Q - 2 \quad |S(U)|_Q \leq 2|A|_Q - 2. \tag{3.7}$$

Proof. Let B be a prefix-tree representing the set $U' = \{x_1\$_1, \ldots, x_m\$_m\}$, obtained by appending to each string of U a new symbol $\$_i$, $i = 1, \ldots, m$, to make their suffixes distinct and let B' be the automaton obtained by minimization of B. By construction, B has m more states than A, but since all final states of B are equivalent and merged after minimization, B' has at most one more state than A.

By construction, B' is a suffix-unique automaton and by Proposition 3.2, $|S(B')|_Q \leq 2|B'|_Q - 3$. Removing from $S(B')$ the transitions labeled with the extra symbols $\$_i$ and connecting the resulting automaton yields the minimal suffix automaton $S(U)$. In $S(B')$, there must be a final state reachable only by the transitions labeled with $\$_i$, which becomes non-accessible after removal of the extra symbols. Thus, $S(U)$ has at least one state less than $S(B')$, which gives:

$$|S(U)|_Q \leq |S(B')|_Q - 1 \leq 2|B'|_Q - 4 = 2|A|_Q - 2. \tag{3.8}$$

A similar bound holds for the factor automaton $F(U)$ following the argument given in the proof of Corollary 3.1. □

3.4.5 k-suffix-unique Bound

When A is k-suffix-unique with a relatively small k, as in our applications of interest, the following proposition provides a convenient bound on the size of the suffix automaton.

Proposition 3.3 *Let A be a k-suffix-unique deterministic 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 number of states of the suffix automaton of A:*

$$|S(A)|_Q \leq 2|A_k|_Q + 2kn - 3, \tag{3.9}$$

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

Proof. Let A be a k-suffix-unique deterministic automaton accepting strings of length more than three and let the alphabet Σ be augmented with n temporary symbols $\$_1, \ldots, \$_n$. By marking each string accepted by A with a distinct symbol $\$_i$, we can turn A into a suffix-unique deterministic automaton A'.

To do that, we first unfold all k-length suffixes of A. In the worst case, all these (distinct) suffixes were sharing the same $(k-1)$-length suffix. In this worst case, unfolding can increase the number of states of A by as many as $kn - n$ states. Marking the end of each suffix with a distinct \$-symbol further increases the size by n. The resulting automaton A' is deterministic and $|A'|_Q \leq |A_k|_Q + kn$. By Proposition 3.2, the size of the suffix automaton of

A' is bounded as follows: $|S(A')| \leq 2|A'| - 3$. Since transitions labeled with a \$-sign can only appear at the end of successful paths in $S(A')$, we can remove these transitions and make their origin state final, and minimize the resulting automaton to derive a deterministic automaton A'' accepting the set of suffixes of A. The statement of the proposition follows the fact that $|A''| \leq |S(A')|$. □

Since the size of $F(A)$ is always less than or equal to that of $S(A)$, we obtain directly the following result.

Corollary 3.3 *Let A be a k-suffix-unique automaton accepting strings of length more than three. Then, the following bound holds for the factor automaton of A:*

$$|F(A)|_Q \leq 2|A_k|_Q + 2kn - 3. \tag{3.10}$$

The bound given by the corollary is not tight for relatively small values of k in the sense that in practice, the size of the factor automaton does not depend on kn, the sum of the lengths of suffixes of length k, but rather on the number of states of A used for their representation, which for a minimal automaton can be substantially less. However, in the extreme case when k is large, e.g., when all strings are of the same length and k is as long as the length of the strings accepted by A, our bound coincides with that of [15].

3.5 Suffix Automaton Construction Algorithm

As mentioned in Sections 2.4.1 and 3.3.3, we may construct a factor automaton $F(A)$ by adding ϵ-transitions to A and applying determinization and minimization. The bounds in section 3.4 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. Thus, while the final result of the construction algorithm is guaranteed to be compact, the algorithms described thus far are inefficient at best and prohibitive at worst.

This section describes a linear-time algorithm for the construction of the suffix automaton $S(A)$ of an input suffix-unique automaton A, or similarly the factor automaton $F(A)$ of A. Since, as mentioned in Section 3.3.3, a factor automaton can be obtained from $S(A)$ by making all states of $S(A)$ final and applying a linear-time acyclic minimization algorithm [70], 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 $F(A)$.

Our algorithm relies on the following classical concept in the string matching literature.

Definition 3.3 *If a state q in the suffix automaton corresponds to the equivalence class $[x]$, then the* suffix link *(or* suffix transition *or* suffix pointer *or* failure state*) of p, denoted by $s[q]$, is the state corresponding to the equivalence*

class $[v]$*, where* v *is longest suffix of* x *such that* $v \not\equiv x$*.*

3.5.1 Algorithm Pseudocode and Description

Figures 3.7-3.9 give the pseudocode of the algorithm for constructing the suffix automaton $S(A) = (Q_S, I, F_S, \delta_S)$ of a suffix-unique automaton $A = (Q_A, I, F_A, \delta_A)$, where $\delta_S : Q_S \times \Sigma \mapsto Q_S$ denotes the partial transition function of $S(A)$ and likewise $\delta_A : Q_A \times \Sigma \mapsto Q_A$ that of A. As in the previous section, f denotes the final state of A with no outgoing transitions.

```
CREATE-SUFFIX-AUTOMATON(A, f)
 1  S ← Q_S ← {I}  ▷ initial state
 2  s[I] ← UNDEFINED; l[I] ← 0
 3  while S ≠ ∅ do
 4       p ← HEAD(S)
 5       for each a such that δ_A(p, a) ≠ UNDEFINED do
 6            if δ_A(p, a) ≠ f then
 7                 Q_S ← Q_S ∪ {q}
 8                 l[q] ← l[p] + 1
 9                 SUFFIX-NEXT(p, a, q)
10                 ENQUEUE(S, q)
11  Q_S ← Q_S ∪ {f}
12  for each state p ∈ Q_A and a ∈ Σ such that δ_A(p, a) = f do
13       SUFFIX-NEXT(p, a, f)
14       SUFFIX-FINAL(f)
15  for each p ∈ F_A do
16       SUFFIX-FINAL(p)
17  return S(A) = (Q_S, I, F_S, δ_S)
```

Figure 3.7: Algorithm for the construction of the suffix automaton of a suffix-unique automaton A.

```
SUFFIX-NEXT(p, a, q)
 1  l[q] ← max(l[p] + 1, l[q])
 2  while p ≠ I and δ_S(p, a) = UNDEFINED do
 3        δ_S(p, a) ← q
 4        p ← s[p]
 5  if δ_S(p, a) = UNDEFINED then
 6        δ_S(I, a) ← q
 7        s[q] ← I
 8  elseif l[p] + 1 = l[δ_S(p, a)] and δ_S(p, a) ≠ q then
 9        s[q] ← δ_S(p, a)
10  else r ← q
11        if δ_S(p, a) ≠ q then
12              r ← copy of δ_S(p, a)  ▷ new state with same transitions
13              Q_S ← Q_S ∪ {r}
14              s[q] ← r
15        s[r] ← s[δ_S(p, a)]
16        s[δ_S(p, a)] ← r
17        l[r] ← l[p] + 1
18        while p ≠ UNDEFINED and l[δ_S(p, a)] ≥ l[r] do
19              δ_S(p, a) ← r
20              p ← s[p]
```

Figure 3.8: Subroutine of CREATE-SUFFIX-AUTOMATON processing a transition of A from state p to state q labeled with a.

```
SUFFIX-FINAL(p)
1  if p ∈ F_S then
2        p ← s[p]
3  while p ≠ UNDEFINED and p ∉ F_S do
4        F_S ← F_S ∪ {p}
5        p ← s[p]
```

Figure 3.9: Subroutine of CREATE-SUFFIX-AUTOMATON making all states on the suffix chain of p final.

The algorithm is a generalization to an input suffix-unique automaton of the standard construction for an input string. Our presentation is similar to that of [26]. The algorithm maintains two values $s[q]$ and $l[q]$ for each state q of S_q. $s[q]$ denotes the suffix pointer 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 short-cut transition.

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 admits of course a linear-time implementation. In each iteration of the loop of lines 3-10 in Figure 3.7, 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 particular properties of f which, as discussed in the previous section, can be viewed as the child of different nodes of the tree T, and thus can admit different suffix links. 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) in a way that is very similar to the standard string suffix automaton construction. The loop of lines 2-4 inspects the iterated suffix pointers of p that do not already have an outgoing transition labeled with a. It further 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 9). Otherwise, if $q' \neq q$, a copy of the state q', r, with the same outgoing transitions is created (line 12) and the suffix pointer of q is set to be r. The suffix pointer of r is set to be $s[q']$ (line 15), that of q' is set to r (16), and $l[r]$ defined as $l[p] + 1$ (17). 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 18-20).

The subroutine SUFFIX-FINAL sets the finality 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 3-5.

Figure 3.10 illustrates the application of the algorithm to a particular suffix-unique automaton. All intermediate stages of the construction of $S(A)$ are indicated, including $s[q]$ and $l[q]$ for each state q.

Our algorithm can also be used straightforwardly to generate the *suffix oracle* of a set of strings, which has been shown to have various useful properties [4]. In the construction of the suffix oracle, no new state is created with respect to the input. The suffix oracle of A can thus be constructed in a similar way simply by replacing line 12 in Figure 3.8 by: $r \leftarrow \delta_S(p, a)$ and removing lines 15-17. This algorithm thus straightforwardly extends the construction of the suffix oracle to the case of suffix-unique input automata.

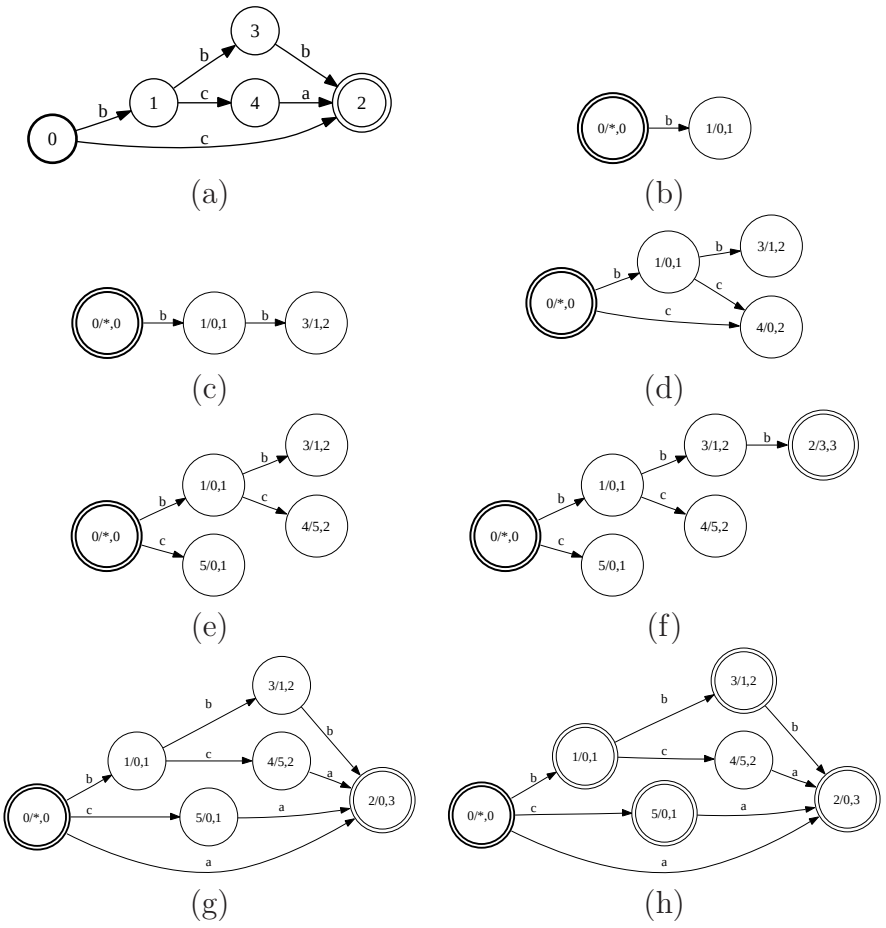


Figure 3.10: Construction of the suffix automaton using CREATE-SUFFIX-AUTOMATON. (a) Original automaton A. (b)-(h) Intermediate stages of the construction of $S(A)$. For each state $(n/s, l)$, n is the state number, s is the suffix pointer of n, and l is $l[n]$.

3.5.2 Algorithm Complexity

For the complexity result that follows, we will assume an efficient representation of the transition function such that an outgoing transition with a specific label can be found in constant time $O(1)$ at any state. Other authors are sometimes assuming instead an adjacency list representation and a binary search to find a transition at a given state, which costs $O(\min\{\log|\Sigma|, e_{\max}\})$ where $e_{\max}$ is the maximum outdegree [26, 27]. If one adopts that assumption, the complexity results we report as well as those of Blumer et al. [14, 15] should be multiplied by the factor $\min\{\log|\Sigma|, e_{\max}\}$.

We begin our analysis of the algorithm runtime by proving that the number of transitions redirected in the while loop of lines 18-20 of SUFFIX-NEXT is linear.

Transition Redirections

Our proof of the linearity of the number of transition redirections proceeds as follows. We first prove that a redirection requires the traversal of a particular combination of non-equivalent factors in A related to each other by a common suffix (Lemma 3.6). Using this fact, we show that a transition is redirected multiple times when we traverse a family of such factors in A (Corollary 3.4). Relying on the analysis for the suffix automaton of a single string [15, 16], we show that the number of redirections across all families in each linear chain of states in A is linear in the length of the chain. The total number of redirections is the sum of those across all the chains in A, which is linear in $|A|$ (Proposition

3.4).

Let A be the input automaton to CREATE-SUFFIX-AUTOMATON. We denote as equiv(p) the longest factor v of A such that $[v]$ is the equivalence class corresponding to state p in $S(A)$. Recall that the notation $u <_s v$ indicates u is a proper suffix of v, and $u \not<_s v$ indicates that u is not a suffix of v. A transition from state p to state q with the label a is denoted as (p, a, q), or if the destination state is not needed for the argument simply as (p, a). We also use the partial transition function of $S(A)$ to refer to transitions, as $\delta_S(s, a) = q$.

Lemma 3.4 *Let A be suffix-unique, and let $x \in \Sigma^*$ be the longest string in $[x]$. Then every member of $[x]$ is a suffix of x.*

Proof. Clearly x is a suffix of x. Let $y \in \Sigma^*$ be such that $y \neq x$ and $y \in [x]$. Let $z \in \Sigma^*$ be a string such that xz is a suffix of A. Then by definition of the equivalence relation on factors of A, yz is also a suffix of A. But by suffix-uniqueness of A, since xz and yz are both suffixes, then y is a suffix of x. □

For the following, note that by definition of the suffix link, $s[q] = p$ iff equiv(p) is the longest suffix of equiv(q) such that [equiv(p)] $\neq$ [equiv(q)].

Lemma 3.5 *If two transitions (s_1, a, m_1) and (s_2, a_1, m_2) are redirected during a single call of* SUFFIX-NEXT*, then $m_1 = m_2$.*

Proof. The proof follows by repeated applications of Proposition 3.1. Before the redirection, we have [equiv(s_1)a] $= m_1$ and [equiv(s_2)a] $= m_2$. After the

redirection we have $\delta_S(s_1, a) = \delta_S(s_2, a) = r$. This implies that $[\text{equiv}(s_1)a] = [\text{equiv}(s_2)a]$, and in turn that $m_1 = m_2$. $\square$

Now that we have established two facts that help us analyze redirections during the construction of the suffix automaton, we prove the following two lemmas which give the connection between redirections and the factors in A that cause them. Figure 3.11 gives an illustration of Lemma 3.6.

Lemma 3.6 *If a transition (s, a, m) is redirected, then A contains two distinct factors of the form cwa and dwa, where $c, d, a \in \Sigma$, $w \in \Sigma^*$, and $c \neq d$.*

Proof. By assumption, a transition is redirected. Hence, there is a call of SUFFIX-NEXT(p, a, q), such that before the call we have $\delta_S(s, a) = m$ and after the call $\delta_S(s, a) = r \neq m$. By Lemma 3.5, there is a single state m associated with all redirections made during this call of SUFFIX-NEXT.

After the redirection, we set $\delta_S(s, a) \leftarrow r$ (line 19), and hence we have $[\text{equiv}(s)a] = [\text{equiv}(r)]$. This by Lemma 3.4 implies that $\text{equiv}(s)a$ is a suffix of $\text{equiv}(r)$. Since we set $s[q] \leftarrow r$, and $s[m] \leftarrow r$, this implies that $\text{equiv}(r) <_s \text{equiv}(m)$ and $\text{equiv}(r) <_s \text{equiv}(q)$. Hence $\text{equiv}(s)a <_s \text{equiv}(m)$ and $\text{equiv}(s)a <_s \text{equiv}(q)$.

So far we know that $\text{equiv}(s)a$ is a suffix of $\text{equiv}(r)$, and a proper suffix of $\text{equiv}(m)$ and $\text{equiv}(q)$. Since $m \neq r$ then by Proposition 1, $\text{equiv}(m) \neq \text{equiv}(r)$. However, since by the pseudocode r becomes the suffix link of m, we have $\text{equiv}(r) <_s \text{equiv}(m)$. Since $s[q] = r$ and $s[m] = r$, m cannot be in

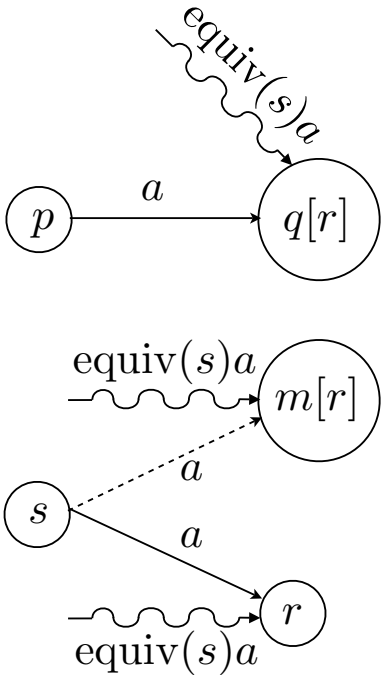


Figure 3.11: Illustration of Lemma 3.6. The dashed line indicates the redirected transition and the notation $q[r]$ indicates state q with suffix link r. equiv$(s)a$ is a suffix of equiv(q), equiv(m), and equiv(r) and hence r, m, and q are all reachable by solid paths ending with equiv$(s)a$.

the suffix link chain of q, otherwise either q has two suffix links, or m is in the suffix chain of r (meaning there is a cycle in the suffix chain), both of which contradict the definition of suffix link. Thus, equiv$(m) \not<_s$ equiv(q). But m and q do have a common suffix ending in a. In other words there exist w, v, c, and d defined as in the lemma statement such that $cwa <_s$ equiv(m) and $dwa <_s$ equiv(q) but $c \neq d$. □

Lemma 3.7 *If a transition (s, a) is redirected twice over all calls of* Suffix-Next*, then A contains factors evwa, cvwa, dwa traversed in that order, such*

that $w \in \Sigma^*$*;* $v \in \Sigma^+$*;* $c, d, e, a \in \Sigma$*;* $c \neq e$ *and* $d \not<_s v$*.*

Proof. By assumption the transition (s, a) is redirected twice. Let m be its destination before the redirections, and m' and m'' those after the first and second redirection, respectively. By the statement and proof of Lemma 3.6 applied to the first redirection, there must be $c, e \in \Sigma$ and $v' \in \Sigma^*$ such that $ev'a$ and $cv'a$ are factors of A, $ev'a <_s \text{equiv}(m)$ and $cv'a <_s \text{equiv}(m')$. After the redirection, we set $s[m] = m'$, meaning that $\text{equiv}(m') <_s \text{equiv}(m)$. Since the second redirection occurs, (s, a, m') is a non-solid transition after the first redirection, meaning that $|v'| > 0$. The second redirection from m' to m'' occurs, so again by Lemma 3.6 there must exist $v \in \Sigma^+$ and $w \in \Sigma^*$ such that $v' = vw$, $wa <_s \text{equiv}(m'')$ and a factor of the form dwa, where $d \in \Sigma$ and $d \not<_s v$, follows the factors already mentioned in the traversal of A. □

Lemma 3.8 *Let* (s, a) *be a transition redirected twice over all calls of* SUFFIX-NEXT*. If the intermediate factor* $cvwa$ *of Lemma 3.7 is removed from the traversal of* A*, then one redirection of* (s, a) *still occurs due to the remaining factors* $evwa$ *and* dwa*.*

Proof. By assumption (s, a) is redirected twice. By Lemma 3.7, the factors $evwa$, $cvwa$, and dwa exist in A. Let SUFFIX-NEXT(p, a, q) be the call on which the a of dwa of Lemma 3.7 is read. Then by the pseudocode, s is in the suffix link chain of p. Since after the first redirection we set $s[m] = m'$, $\text{equiv}(m') <_s \text{equiv}(m)$. Since the second redirection occurs, by the proof of Lemma 3.6, $\text{equiv}(s)a <_s \text{equiv}(m')$, and thus also $\text{equiv}(s)a <_s \text{equiv}(m)$.

Removing *cvwa* from the traversal does not change equiv(s), and hence s is still in the suffix chain of p, and (s, a) is still a non-solid transition. Hence, it is redirected once when *dwa* is traversed. □

Definition 3.4 *Let the* primary factor *and* secondary factor *of a redirection of transition* (s, a) *be the factors cwa and dwa of Lemma 3.6. Then we denote the set of all the primary and the last secondary factor of any redirections of the transition* (s, a) *during the traversal of* A *as the* redirection family $\Pi_{s,a}(A)$.

The next corollary follows from a repeated application of the argument in Lemma 3.7.

Corollary 3.4 *Let* (s, a) *be a redirection redirected* n *times during the construction of* $S(A)$. *Then the redirection family* $\Pi_{s,a}(A)$ *consists of the factors* $f_{n+1}, \ldots, f_1$, *traversed in that order where* $f_{n+1} = c_{n+1}v_nv_{n-1}\ldots v_1a$, *and for* $k \in \{1, \ldots, n\}$, $f_k = c_kv_kv_{k-1}v_{k-2}\ldots v_1a$, *where* $v_1 \in \Sigma^*$, $v_2, \ldots, v_n \in \Sigma^+$, $c_1 \ldots c_{n+1} \in \Sigma$ *such that for* k *in* $k \in \{1, \ldots, n\}$, $c_k \not\leq_s v_{k+1}$ *and* $c_{n+1} \neq c_n$.

The next corollary follows by an repeated application of the argument of Lemma 3.8 to the generalized form of a redirection family just given in Corollary 3.4.

Corollary 3.5 *Let* (s, a) *be a redirection redirected* n *times during the construction of* $S(A)$. *Then removing* k *factors in the redirection family* $\Pi_{s,a}(A)$ *from the traversal of* A *reduces the number of redirections of* (s, a) *to* $n - k$.

Define a *linear chain* of A to be a path between states b and e such that every state on the path has one outgoing transition, except for e which may have one or zero outgoing transitions. If B is a linear chain, then $\Pi_{s,a}(B) \subseteq \Pi_{s,a}(A)$ is the subset such that every member of $\Pi_{s,a}(B)$ finishes on a transition in B. Let $\Pi(B)$ be the set of all those factors belonging to redirection chains that cause redirections inside B: $\Pi(B) = \bigcup_{(s,a)\in A} \Pi_{s,a}(B)$. Finally let $r(B)$ be the total number of redirections made over all calls of SUFFIX-NEXT(p, a, q) such that $(p, a, q) \in B$.

Lemma 3.9 $r(B) = O(|B|)$.

Proof. Let str(B) be the string automaton accepting the single string read in A by traversing the linear chain B sequentially. We know from previous work [15, 16] that the number of redirections made during the construction of $S(\text{str}(B))$ is $O(|B|)$. By Corollary 3.5, the number of redirections made during the construction of $S(\text{str}(B))$ is exactly the same as that made during the construction of $S(A)$ while traversing transitions of B, with the possible exception of the first member of a given redirection family that is encountered while traversing B. Such a factor za can cause a redirection of transition (s, a) if it serves as a secondary factor to a primary factor traversed before B. In the construction of $S(\text{str}(B))$, this redirection would not occur since za is the first member of $\Pi_{s,a}(\text{str}(B))$. But every transition (s, a) redirected during the construction of $S(\text{str}(B))$ is associated uniquely with a redirection family $\Pi_{s,a}(A)$. Hence, there is at most one extra redirection per transi-

tion of $S(\text{str}(B))$, and thus the total number of extra redirections is at most $|S(\text{str}(B))|$. We know from the previously known size bounds for the suffix automaton of a single string [26, 14] that $|S(\text{str}(B))| = O(|\,\text{str}(B)|) = O(|B|)$. Hence $r(B) = O(|B|) + O(|B|) = O(|B|)$. □

Proposition 3.4 *The total number of transition redirections made during a call of* CREATE-SUFFIX-AUTOMATON(A) *is* $O(|A|)$.

Proof. By Lemma 3.9, the total number of redirections in any chain B across all families is $O(B)$. Thus, the total number of redirections over the entire traversal of A is $\sum_{B \in A} r(B) = \sum_{B \in A} O(|B|) = O(|A|)$. □

Final Complexity Result

Armed with the proof of the linearity of the number of redirections made during the construction of $S(A)$, we now prove the following final complexity result.

Proposition 3.5 *Let A be a minimal deterministic suffix-unique automaton. Then, the runtime complexity of algorithm* CREATE-SUFFIX-AUTOMATON(A, f) *is* $O(|S(A)|)$.

Proof. SUFFIX-NEXT is called at most once per transition, so the total number of calls of SUFFIX-NEXT is $O(|A|)$. Fix a transition (p, a, q) of A with $q \neq f$. The cost of the execution of the steps 1-20 by SUFFIX-NEXT is proportional to the total number of iterated suffix link traversals in the loop of

lines 2-4 and lines 18-20. Each iteration of lines 2-4 results in a new transition being created in $S(A)$, so the total number of loop iterations over all calls of SUFFIX-NEXT is $O(|S(A)|)$. By Proposition 3.4, the total number of iterations of the loop of lines 18-20 is $O(|A|)$. Thus, the total complexity is $O(|S(A)|)$. □

3.6 Weighted Suffix Automaton Algorithm

In the forthcoming, we generalize the algorithm presented in Section 3.5 to the weighted case. That is, we describe 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. Once again in the weighted case, 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, it suffices to describe a linear-time algorithm for the construction of $S(A)$.

The algorithm given in this section holds over the tropical semiring (see Section 2.2), which is used in our music identification system; however, we conjecture that this algorithm can be generalized to arbitrary semirings.

3.6.1 Algorithm Pseudocode and Description

Figures 3.12-3.14 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 the partial transition functions δ_S and δ_A

are as in Section 3.5; $\omega_S \colon Q_S \times \Sigma \mapsto \mathbb{K}$ 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.

```
WEIGHTED-CREATE-SUFFIX-AUTOMATON(A, f)
 1  S ← Q_S ← {I}  ▷ initial state
 2  s[I] ← UNDEFINED; l[I] ← 0; W[i] ← 0
 3  while S ≠ ∅ do
 4       p ← HEAD(S)
 5       for each a such that δ_A(p, a) ≠ UNDEFINED do
 6            if δ_A(p, a) ≠ f then
 7                 Q_S ← Q_S ∪ {q}
 8                 l[q] ← l[p] + 1
 9                 WEIGHTED-SUFFIX-NEXT(p, a, q)
10                 ENQUEUE(S, q)
11  Q_S ← Q_S ∪ {f}
12  for each state p ∈ Q_A and a ∈ Σ s.t. δ_A(p, a) = f do
13       WEIGHTED-SUFFIX-NEXT(p, a, f)
14       WEIGHTED-SUFFIX-FINAL(f)
15  for each p ∈ F_A do
16       WEIGHTED-SUFFIX-FINAL(p)
17  ρ_S(I) ← min_{p∈Q_S} W[p]
18  return S(A) = (Q_S, I, F_S, δ_S, ω_S, ρ_S)
```

Figure 3.12: Algorithm for the construction of the suffix automaton of a weighted suffix-unique automaton A.

In the following $s[q]$, $l[q]$, and S are as defined in Section 3.5. 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

WEIGHTED-SUFFIX-NEXT(p, a, q)

```
 1  l[q] ← max(l[p] + 1, l[q])
 2  W[q] ← W[p] + ω_A(p, a)
 3  while p ≠ I and δ_S(p, a) = UNDEFINED do
 4        δ_S(p, a) ← q
 5        ω_S(p, a) ← W[q] − W[p]
 6        p ← s[p]
 7  if δ_S(p, a) = UNDEFINED then
 8        δ_S(I, a) ← q
 9        ω_S(I, a) ← W[q]
10        s[q] ← I
11  elseif l[p] + 1 = l[δ_S(p, a)] and δ_S(p, a) ≠ q then
12        s[q] ← δ_S(p, a)
13  else r ← q
14        if δ_S(p, a) ≠ q then
15              r ← copy of δ_S(p, a)  ▷ new state with same transitions
16              Q_S ← Q_S ∪ {r}
17              s[q] ← r
18        s[r] ← s[δ_S(p, a)]
19        s[δ_S(p, a)] ← r
20        W[r] ← W[p] + ω_S(p, a)
21        l[r] ← l[p] + 1
22        while p ≠ UNDEFINED and l[δ_S(p, a)] ≥ l[r] do
23              δ_S(p, a) ← r
24              ω_S(p, a) ← W[r] − W[p]
25              p ← s[p]
```

Figure 3.13: Subroutine of WEIGHTED-CREATE-SUFFIX-AUTOMATON processing a transition of A from state p to state q labeled with a.

```
WEIGHTED-SUFFIX-FINAL(p)
 1  m ← W[p]
 2  if p ∈ F_S then
 3       p ← s[p]
 4  while p ≠ UNDEFINED and p ∉ F_S do
 5       F_S ← F_S ∪ {p}
 6       ρ_S(p) ← m − W[p]
 7       p ← s[p]
```

Figure 3.14: Subroutine of WEIGHTED-CREATE-SUFFIX-AUTOMATON making states on the suffix chain of p final and setting their final weights.

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.

The functions WEIGHTED-CREATE-SUFFIX-AUTOMATON, WEIGHTED-SUFFIX-NEXT, and WEIGHTED-SUFFIX-FINAL are weighted versions of their unweighted analogues, CREATE-SUFFIX-AUTOMATON, SUFFIX-NEXT, and SUFFIX-FINAL, respectively. In WEIGHTED-CREATE-SUFFIX-AUTOMATON,

the processing of the transitions (p, a, f) with destination state f is again delayed to a later stage (lines 12-14). This is because of the special property of f that it may admit not only different suffix pointers but also different values of $l[f]$ and $W[f]$.

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

Figure 3.15 illustrates the application of the algorithm to a 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.

3.6.2 Algorithm Complexity

Proposition 3.6 *Let A be a minimal deterministic suffix-unique automaton. Then, a call to* WEIGHTED-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 linear runtime complexity in Proposition 3.5. The total number of transitions added and redirected by the unweighted algorithm is of course also linear. In the weighted algorithm given in Figures 3.12-3.14,

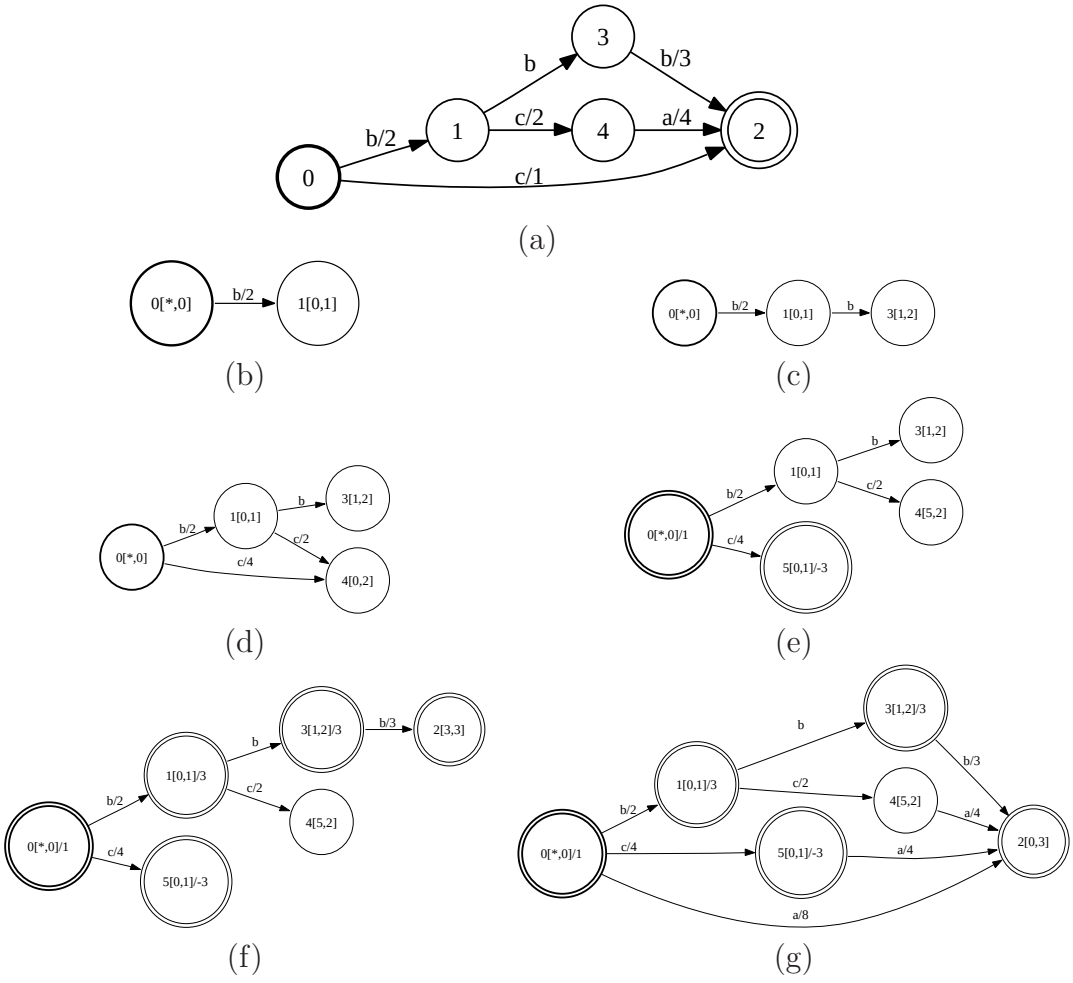


Figure 3.15: Construction of the suffix automaton using Weighted-Create-Suffix-Automaton. (a) Original automaton A. (b)-(g) Intermediate stages of the construction of $S(A)$. For each state $n[s, l]/w$, n is the state number, s is the suffix pointer of n, l is $l[n]$, and w is the final weight, if any.

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 WEIGHTED-CREATE-SUFFIX-AUTOMATON). Hence, the total number of weight adjustments is also linear. □

3.7 Experiments

We have conducted several empirical studies of the bounds and algorithms presented in this chapter. The results are presented in this section.

3.7.1 Factor Automata Size Bounds

We have verified the novel bounds on the size of factor and suffix automata given in Section 3.4 in the context of the music identification system described in Chapter 2. As mentioned in Section 2.4, remarkably, even in the case of 15,455 songs, the total number of transitions of $F_w(A)$ was 53.0M, which is only about 0.004% more than that of $F(A)$. We also have $|F(A)|_E \approx 2.1|A|_E$. As is illustrated in Figure 3.16, this multiplicative relationship is maintained as the song set size is varied between 1 and 15,455.

The k-suffix-unique bound given in Section 3.4.5 relies on the strings in our set U to have relatively short common suffixes. We tested this assumption in our music identification system. For the case of 15,455 songs, U is 45-suffix-unique, which is relatively short compared to an average string length of 1,700.

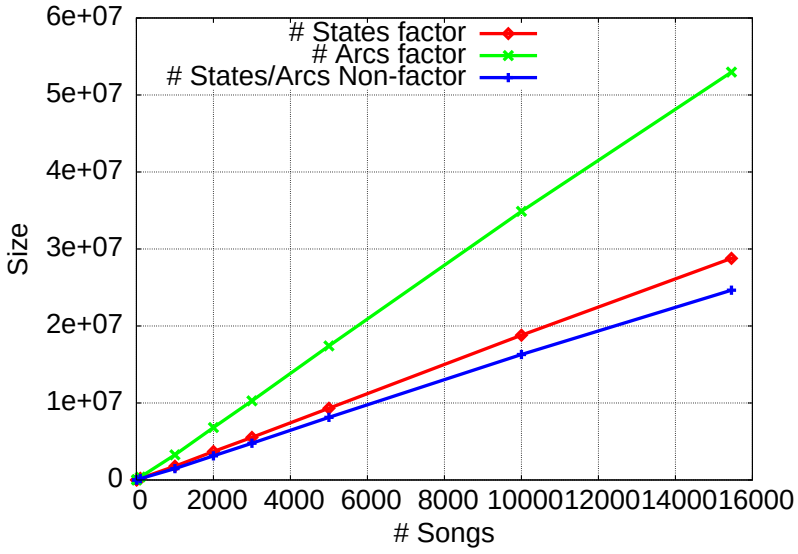


Figure 3.16: Comparison of automaton sizes for different numbers of songs. "#States/Arcs Non-factor" is the size of the automaton A accepting the entire song transcriptions. "# States factor" and "# Arcs factor" is the number of states and transitions in the weighted factor acceptor $F_w(A)$, respectively.

Figure 3.17 demonstrates that the number of suffix "collisions" drops rapidly as the suffix size is increased. For our music collection, the number of states of the weighted factor automaton $F_w(A)$ compares to the size of the input automaton A as

$$|F_w(A)|_Q \approx 28.8\text{M} \approx 1.2|A|_Q \tag{3.11}$$

We can thus conclude that the bound of Corollary 3.3 is verified in this empirical context.

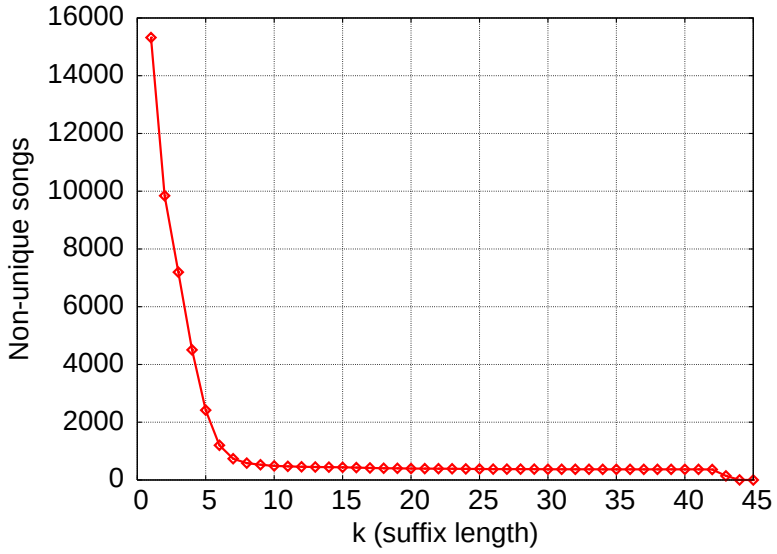


Figure 3.17: Number of strings in U for which the suffix of length k is also a suffix of another string in U.

3.7.2 Correctness Verification

As Section 3.3.3 describes, an alternative way of constructing suffix automata is by adding ϵ-transitions, followed by ϵ-removal, determinization, and minimization. As a result, the correctness of the output of the suffix automaton construction algorithms given in Sections 3.5 and 3.6 may be verified by testing equivalence of the resulting suffix automaton to that produced using these general algorithms.

Our implementation was in the OpenFst toolkit [7], which also provides implementations of the above general algorithms, as well as an equivalence checking algorithm. Hence, our test suite was constructed by repeating the following test scenario:

1. Generate a random suffix-unique string set U, along with random weights for the weighted case.

2. Construct the deterministic and minimal input automaton A accepting U.

3. Construct suffix automata $S_g(A)$ with the general algorithm and $S_n(A)$ with the new algorithm from Section 3.5 or 3.6.

4. Check that $S_g(A)$ and $S_n(A)$ are equivalent.

We ran this simulation continuously for several weeks on a cluster of 20 servers. No cases were found where our algorithms produced an incorrect result.

3.7.3 Algorithm Speed Improvement

We constructed the weighted factor automaton required for the music identification system of Chapter 2 with both the method of Section 3.3.3 and the new suffix automaton algorithm presented in this section. The new algorithm completed in 3,190 seconds, as compared to 56,058 seconds for the previous algorithm. Hence, a speedup of more that 17 times was exhibited in this empirical setting.

3.8 Summary

We have presented a novel analysis of the size of the suffix automaton and factor automaton of a set of strings represented by an automaton in terms of the size of the original automaton. Our analysis shows that suffix automata and factor automata can be practical for constructing an index of a large number of strings. As a result, factor automata of automata can be used to construct a useful and compact index for very large-scale search tasks.

We have further given unweighted and weighted versions of a linear-time algorithm for constructing the suffix automaton or factor automaton of a set of strings in time linear in the size of a prefix tree representing them, a drastic improvement on the previous method using the generic ϵ-removal and determinization algorithms. Our algorithm applies to any input suffix-unique automaton and generalizes the standard on-line construction of a suffix automaton for a single input string.

All the contributions presented in this chapter are generally applicable to a number of tasks where search of a collection of strings or sequences, possibly with associated costs or probabilities, is required. However, in this thesis, both the motivation for this work and its empirical validation have been in the context of the music identification system presented in Chapter 2. The size bounds presented ensure that our finite automaton representation will scale to even larger song collections. Specifically, the expected linear relationship between the number of songs and the size of the factor automaton

representing the song collection is exhibited consistently as the number of songs is increased. Moreover, the new algorithms given in this chapter ensure that a compact representation can be constructed efficiently. Indeed, we have confirmed that our algorithm results in more than a 17-fold speed improvement in the construction of the song factor automaton over the previously used algorithms.

Chapter 4

Topic Segmentation

4.1 Introduction

Search of large collections of audio sequences such as speech and music relies on a transcription of the source sequences in terms of intermediate units. As mentioned in Chapter 1, the wide variety of possible units includes linguistic primitives such as words or phonemes in the case of speech and elementary music sounds such as notes or the music phonemes of Chapter 2 in the case of music.

In order to enable effective indexation, and in turn, effective search, it is important to analyze the transcribed sequence to discover structural elements. For example, in the case of music phoneme transcription of Chapter 2, the construction of a factor automaton allows us to take advantage of phoneme sequences that appear in the transcription of multiple songs to construct an

efficient and accurate index.

Another type of structure that applies to audio collections is the presence of topics. Natural language streams, such as news broadcasts and telephone conversations, are marked with the presence of underlying topics. These topics influence the statistics of text or speech that is produced. As a result, learning to identify the topic underlying a given segment of speech or text, or to detect boundaries between topics is beneficial in a number of ways. For example, for a speech recording that is being transcribed, knowledge of the topic can be used to improve transcription quality through the use of a speech recognizer with a topic-dependent language model [83]. Topicality information can also be used to improve navigation of audio and video collections such as YouTube, by considering a common topic as a feature when creating links between items. Finally, in real-time speech recognition applications, topicality information can be used to improve both the quality of speech recognition and the dialogue path selected by the system [48, 72].

In this chapter, we primarily discuss topicality in text and speech. However, it should be noted that the notion of underlying topics or factors that account for observed content is applicable broadly to a number of fields beyond speech and text processing. Models traditionally used to detect and label topics in text and speech have been used successfully for tasks ranging from music similarity computation to image categorization [38, 49].

4.1.1 Previous Work

The previous work related to the material in this chapter spans several areas of natural language and speech processing, machine learning, and information retrieval.

Topic Models

Topic models or topic labeling algorithms assign a topic label sequence to a stream of text or speech. Much of the recent work on topic analysis has been focused on generative models. A good introduction to these models can be found in [75]. Let $V = \{w_1, w_2, \ldots, w_n\}$ be the vocabulary of n words. Then an *observation* a is an observed set of text or speech expressed through the empirical frequency, or expected count, $C_a(w_i)$ for each $w_i \in V$. In generative topic models, a sequence of word observations is explained by a latent sequence of topic labels. As a result, high-dimensional text can be described with a low-dimensional mixture of the topics learned. A simple generative formulation of a topic model is

$$z = \arg\max_z \Pr(z|a) = \arg\max_z \Pr(a|z)\Pr(z), \tag{4.1}$$

where a is the sequence of observed text, and z is the topic label assigned. The second equality follows by Bayes' rule and the realization that the prior over the observations $\Pr(a)$ does not change with respect to topic. Under such topic models, text is labeled by decoding a maximum *a posteriori* sequence

of topics accounting for the text. In these models, a is treated as a "bag of words," meaning the order of the words in the text or speech stream underlying a is generally not considered, merely the occurrence frequency of each word within a. In practice, a can be a sentence, a window of n words, an utterance, or a single word.

There has been a substantial body of work trying to discover the best model for the conditional distribution $\Pr(a|z)$. Initial work proposed using a corpus of data labeled with topics to construct a per-topic unigram language model [84]. An approach known as PLSA [40] decomposes the training observation sequence into topic-coherent documents, and decomposes the topic-dependent conditional topic model into a word-topic model and a document-topic model, $\Pr(a|z)$ and $\Pr(d|z)$, respectively. The latter likelihood defines a per-document weight on each topic, and thus each document can be viewed as being associated with a particular mixture of topics. One limitation of PLSA is that it is only able to predict document classes for text seen in the training data. A modified PLSA is used in [12] to allow for such generalization.

Latent Dirichlet Allocation (LDA) [13] is a truly generative topic model intended to overcome some of the main limitations of PLSA. LDA does not have a concept of documents, but views each observation as a mixture of topics where the mixture weights themselves are random variables. Namely, the formulation of Equation 4.1 is used, but the distributions $\Pr(w|z)$ and $\Pr(z)$ are modeled as multinomial distributions with Dirichlet priors. As a result, any topic mixture is possible, not just the one seen at training time. The training

algorithm involves finding the maximum-likelihood setting of the Dirichlet parameters directly with expectation-maximization (EM) or approximately with Gibbs sampling (see e.g., [75]).

While LDA has been a very popular topic model, its primary disadvantage is that it assigns topic labels to observations one at a time without considering the context of surrounding data in the observation stream. To rectify this, several generative formulations have been suggested for including the transition from topic to topic in the observation stream in the model. Hidden topic Markov models (HTMMs) [32] use an HMM structure where each state corresponds to a topic z, as in [84, 12]. Each topic is in turn associated with a standard LDA model. HTMM allows for a context-dependency model for topic-to-topic transitions to be learned at the same time as the topic model itself. Another recent work [68] provides a slightly different context dependency formulation in which each state in the Markov model accounts for the word observations with a mixture of topic models rather than a single topic assignment.

Topic Segmentation

Topic labeling algorithms are also topic segmentation algorithms because a topic assignment to a stream of text or speech also implies a topic-wise segmentation of the stream. However, generative topic analysis algorithms such as HTMM and PLSA attempt to model the distribution of words in a particular topic, the distribution of topic-to-topic transitions, and the global distribution

of topic labels. Certainly if one can accurately model the underlying topic sequence, one can also easily solve the problem of topic segmentation or any other related problem. However, our goal in this work is simpler – to arrive at the best topic-wise segmentation of a stream of text or speech, and we endeavor to create an algorithm specifically designed for this problem. A number of topic segmentation algorithms have been proposed in the literature, many in the context of the DARPA Topic Detection and Tracking (TDT) evaluation [2]. This evaluation challenged researchers to develop algorithms for three related segmentation tasks:

- New event detection – detect the onset of a new topic in a stream of news stories in an online fashion.
- Event detection – given a stream of news stories, determine all the topics present.
- Text segmentation – given a stream of text or speech, determine the boundaries between topic-coherent segments.

In this chapter, we shall target the last task specifically; however since all three tasks involve segmenting a stream of text or speech into topic-coherent blocks, the algorithms and quality measure we propose can be adapted to the other tasks as well.

An early topic segmentation algorithm is known as TextTiling [36]. This work proposed the idea of doing segmentation by computing word counts for a

sliding window over the input text. Text similarity is then evaluated between pairs of adjacent windows according to a cosine similarity measure,

$$\frac{\sum_{i=1}^{n} C_1(w_i)C_2(w_i)}{\sqrt{\sum_{i=1}^{n} C_1(w_i)^2 \sum_{i=1}^{n} C_2(w_i)^2}}. \tag{4.2}$$

where $C_1(w)$ and $C_2(w)$ are counts for word i in windows 1 and 2, respectively, and n is the vocabulary size. The segmentation is obtained by placing segment boundaries between window pairs that differ significantly according to this distance.

The cosine score is only one measure of text similarity, or *lexical cohesion* as it is frequently referred to in the computational linguistics literature. One disadvantage of using raw word counts is that this method assigns equal weights to all words. As a result, words that are naturally more prevalent in the corpus effectively receive a high weight in the computation of the cosine score. Several works in the TDT evaluation [17, 85, 3], especially those concerned with detecting new stories in a news stream, as well as more recent work [30] sought to bypass this limitation by using the term frequency–inverse document frequency (tf–idf) to weight each word's contribution to the similarity score. The tf–idf score of word w in document d is often defined as

$$\text{tf–idf}_{d,w} = C_d(w) \log \frac{N}{N_w}, \tag{4.3}$$

where $C_d(w)$ is the count of word w in document d, N is the total number of documents in the collection and N_w is the number of documents containing

the word w, though many variants of this weighting have been suggested [73].

While tf–idf has been a common theme in the topic segmentation literature the algorithms used have varied. Brants et al. [17] showed that computing the tf–idf-weighted Hellinger distance between adjacent stories and using a threshold to detect topic changes yields an improvement over the cosine distance baseline. For event detection, Yang et al. and Eichmann et al. [85, 29] used tf–idf variants in conjunction with clustering algorithms in which only sequentially adjacent story clusters in the news stream were allowed to merge.

As we shall discuss in Section 4.2.2, considering words in isolation for topic segmentation results in a natural limitation on the algorithms created. As demonstrated by Kozima in 1993 [45], word pair similarity can be an important source of information for topic segmentation. In this work, word pair similarity was computed from a semantic network constructed from a dictionary [46]. The authors passed a single window over the input text (a single short story) and calculated a lexical cohesion score by evaluating the similarity of word pairs found in the window. Low lexical cohesion scores were demonstrated to be correlated with ground truth topic boundaries as judged by human readers. A later work proposed predicting a topic boundary before any single word that scored poorly by dictionary-based similarity to the words seen since the previous boundary [47]. Evaluation was still on the same single short story, and correlation with human-placed boundaries was again observed. However, the segmentation algorithm just mentioned was marked with the use of numerous heuristics, which combined with the scarce evaluation put into question the

broad applicability of the approach.

Several works suggested that the topic segmentation problem can be viewed as a binary classification problem at every possible segment boundary. Maximum entropy models have been a popular classifier choice; however, the choice of features has varied. Beeferman et al. [10] used a combination of short-range language models and long-range dependencies between words in the text stream as input to the classifier. Reynar [71] used a number of features computed between adjacent windows of text, including word distribution similarity, word synonymy according to WordNet [53], and counts of hand-picked cue words. In a topic segmentation work over the TDT speech corpora, Dharanipragada et al. [28] used a combination of decision trees and maximum entropy models with features including n-gram counts in adjacent windows of text, as well as non-text events such as silence duration.

Novelty Detection

The term novelty detection generally stands for the task of detecting data points that stand out as outliers in an established pattern. There is a vast literature on novelty detection spanning many theoretical and applied fields. Novelty detection techniques have been used to find anomalies in data streams ranging from cell phone network usage [41] to mammogram detection [76, 39] to handwritten digit recognition [74, 39]. Several surveys of the novelty detection literature have been published [51, 52, 37].

Assuming the presence of outliers, many novelty detection approaches at-

tempt to learn a model of the data that reflects the true distribution more than the empirical distribution. However, as was pointed out by [77, 74], detecting outliers is a simpler problem than modeling the full distribution of the data. Hence, these authors suggested that to detect outliers, it suffices to give a boundary separating the bulk of the points and those classified as outliers without attempting to estimate the data distribution. Treating points lying on the decision boundary as support patterns, an outlier classifier of a compact form similar to that of the support vector machine [23, 79] can be given in the form of a sphere enclosing the non-outlier points [77] or as a hyperplane separating the non-outlier points from the origin [74].

4.1.2 Summary of Contributions

In this chapter, we describe our work on topic segmentation, defined here as automatic division of a stream of text or speech into topic-coherent segments. Throughout the work, we address specifically the case when the input to the topic segmentation algorithm is a speech audio sequence. We point out major limitations of the currently accepted topic segmentation quality measure known as CoAP, including the fact that it does not take into account the word content of the segments produced by the algorithms. We then introduce a general measure of text and speech similarity and give a topic segmentation quality measure incorporating this similarity score and overcoming many of the limitations of CoAP.

Speech-to-text transcription of audio streams is a process inherently marked

with errors and uncertainty, which results in difficulties for algorithms trying to discover topical structure. We create novel algorithms for topic segmentation that use word co-occurrence statistics to evaluate topic-coherence between pairs of adjacent windows over the speech or text stream and hypothesize segment boundaries at extrema in the similarity signal. Our algorithms move beyond cosine-similarity and other methods based purely on comparing word distributions. In our approach, we apply our general similarity measure to evaluate topic-coherence between pairs of adjacent windows over the speech or text stream. In this way, our topic segmentation algorithms are related to the music segmentation algorithm used for acoustic model initialization of Section 2.3.1.

We point out that similarity-based algorithms suffer from a vulnerability to off-topic word content in a generally topic-coherent observation stream introduced by speech recognition errors and/or filler text. To create a more robust algorithm, we employ the use of a compact support-vector based description of a window of text or speech observation in word similarity space. We prove that our text similarity measure is a positive definite symmetric (PDS) kernel, and thus that such a description can be found by solving a convex optimization problem. We demonstrate that, indeed, by comparing these compact feature-space descriptions of the window statistics rather than the window statistics themselves, we can create a more accurate topic segmentation algorithm.

In experiments over speech and text streams from the Topic Detection and Tracking (TDT) corpus, we show that our novel algorithm outperforms a mod-

ern generative learning technique, the hidden topic Markov model (HTMM) [32], as measured both by our new Topic Closeness Measure (TCM) as well as the previously used CoAP measure. In line with the overall theme of this thesis, we note that topic segmentation in the presence of uncertainty introduced by the use of the speech recognizer can be improved by using the text hypotheses competing with the single most likely one. Specifically, we demonstrate that information from two information sources derived from speech recognition word lattices can help improve topic segmentation over the one-best baseline.

Some of the results presented in this chapter were published in [61].

4.1.3 Outline

The remainder of the chapter is organized as follows. In Section 4.2, we review CoAP, point out its limitations, and develop a new topic segmentation measure based on a general formulation for computing similarity between streams of natural language text or speech. In Section 4.3, we give a general formulation of similarity-based topic segmentation algorithms, and suggest two new algorithms within this formulation. In Section 4.4, we report our technique for incorporating lattice information from a speech recognizer into topic segmentation algorithms. We present our empirical results in Section 4.5 and conclude in Section 4.6.

4.2 Topic Segmentation Quality

In this section we seek a new quality measure for topic segmentation algorithms. This measure should be applicable both to segmentations produced by topic models such as HTMM [32], which implicitly segment the text stream by assigning topic labels to successive segments, as well as explicit segmentation algorithms such as TextTiling [36]. It should also apply to text as well as speech input, and should reflect human intuition about what constitutes a good segmentation. The most popular quality measure used in past work is known as the Co-occurrence Agreement Probability, or CoAP.

4.2.1 Co-occurrence Agreement Probability

Let the input to a segmentation algorithm be a sequence of observations $T = (x_1, \ldots, x_m)$. We refer to the correct segmentation provided by human judges of topicality or some other oracle as the reference, and that provided by a topic segmentation algorithm to be evaluated as the hypothesis. CoAP [10] is broadly defined as:

$$P_D(\text{ref}, \text{hyp}) = \sum_{1 \le i \le j \le n} D(i,j) \left(\delta_{\text{ref}}(i,j) \overline{\oplus} \delta_{\text{hyp}}(i,j) \right), \tag{4.4}$$

where $D(i,j)$ is a distance probability distribution over observations i, j; δ_{ref} and δ_{hyp} are indicator functions that are one if observations i and j are in the same topic in the reference and hypothesis segmentations, respectively; and $\overline{\oplus}$ is the exclusive NOR operation ("both or neither"). In practice, CoAP scoring

is almost always reduced to a single sliding window of fixed size k over the observations, meaning that D is the distribution with its mass placed entirely on the event $\{i, j\colon j - i = k\}$. This form of CoAP, or more precisely the error according to this form of CoAP, is often referred to in the literature as P_k.

For example, consider the reference and hypothesis text segmentations given in Figure 4.1 and assume that $k = 2$. In the reference, sentence pairs $(1, 3)$, $(2, 4)$, $(4, 6)$ are in different segments, while $(3, 5)$ is in the same segment. In the hypothesis, $(1, 3)$, $(3, 5)$, $(4, 6)$ are in different segments, while $(2, 4)$ is in the same. Hence, the hypothesis segmentation contains one false positive segment boundary, $(3, 5)$ and one false negative, $(2, 4)$. Thus, out of four window positions, two are in agreement and two are not, hence this segmentation receives 50% according to CoAP with $k = 2$.

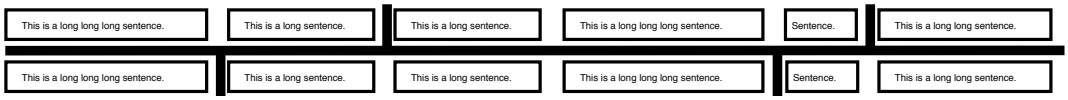


Figure 4.1: A comparison of a reference segmentation (top) and a hypothesis segmentation (bottom) for a stream of text sentences. Boxes indicate sentences and dark lines between boxes denote topic segment boundaries.

Various modifications of CoAP have been used in previous studies, including those assigning different weights to false positive and false negative segment boundaries (e.g., [64]). Another modification of CoAP known as WindowDiff proposed to compare the number of segment boundaries encountered within a window of size k rather than simply the presence or absence of one [66].

CoAP functions purely by analyzing the segmentation of the observations

into topic-coherent segments, without taking into account the content of the segments labeled as topic-coherent. As a result, every spurious or missing topic boundary is penalized equally without regard for the topics that it falsely separates or fails to correctly separate, thus yielding a measure that may not accurately reflect the quality of the segmentation. To illustrate this point, consider again the reference and hypothesis segmentations of Figure 4.1 with $k = 2$. CoAP penalizes the false negative boundary $(2, 4)$ and the false positive boundary $(3, 5)$ equally. However, suppose that the word distribution of the topic-coherent reference sentence range $[3, 5]$ is very close to that of sentence 2 and very far from that of sentence 6. Thus, while the hypothesis segmentation falsely includes both 2 and 6 in segments overlapping the true topic-coherent segment $[3, 5]$, the inclusion of 6 should be penalized more than the inclusion of 2. Our new measure presented in Section 4.2.3 formalizes this intuition.

Additionally, CoAP is dependent on the choice of window size k. WindowDiff does not suffer as much from this dependence, but a value for k must still be chosen and still affects the score [66]. Various heuristics exist for the choice of k. One idea used in previous work has been to set k equal to half the average reference segment length. Another popular setting is such that the score for degenerate segmentations (i.e., random segmentations, those that place every possible boundary, or those that place none at all) get a score of around 50%. This latter heuristic is the one used in the implementation of CoAP in this work.

Finally, by matching sentences i and j of Equation 4.4 between the refer-

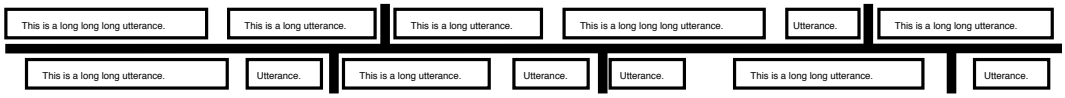


Figure 4.2: A comparison of a reference segmentation (top) and a hypothesis segmentation (bottom) for a stream of speech utterances. Boxes indicate utterances and dark lines between boxes denote topic segment boundaries. Note that the segmentation of speech into utterances is different in the reference and the hypothesis.

ence and hypothesis, CoAP implicitly requires that the reference and the hypothesis segmentations are obtained by placing boundaries in the same stream of text, or at least two streams of text where sentence i in the reference corresponds exactly to sentence i in the hypothesis. However, the hypothesis segmentation can be over the output of a speech recognizer, and hence the text can differ from that of the reference segmentation due to recognition errors, and may even be broken differently into utterances, as is illustrated in Figure 4.2. One way of handling this limitation used in previous work [64] and in this work is to align the reference text with the hypothesis text temporally. In this case, k becomes a temporal separation, i.e., number of seconds, rather than the sentence-wise separation of the text case. But even with this approach the behavior of the measure in certain cases is not well-defined. For example, in the reference segmentation of Figure 4.2, it is possible for the temporal CoAP window to be placed toward the end of segment 4 and not be aligned to any utterance in the hypothesis. This is handled by various heuristics, e.g., pick the utterance closest to the boundary (the one used in this work). Moreover,

in contrast to the text case, it is not clear how far apart successive windows should be, i.e., how to sample the distribution $D(i, j)$. In this work, we advanced the window k seconds after each comparison between the reference and hypothesis. For the above reasons, there exists a rather significant mismatch between the measure actually used for the text case and the speech case.

4.2.2 A General Measure of Topical Similarity

As we have discussed in detail above, CoAP is marked with various limitations, which we seek to address in the following. To formalize the intuition of considering not only the placement of segment boundaries but also the closeness of the segments they separate, we shall require the use of a similarity measure between segments of text or speech observations. One rudimentary similarity function is the cosine distance (Equation 4.2). As mentioned in Section 4.1.1, such dot-product based similarity functions are vulnerable to more frequent words being weighted more heavily in the similarity score, a limitation which may be rectified with the use of tf–idf word weighting [17].

Alternatively, the word frequency over the topic segment can be viewed as an unsmoothed unigram probability distribution of words in the segment. To evaluate the similarity of two segments of speech or text, we may compare their probability distributions. A number of probability distance functions can be used here, including the L_1 distance, L_2 distance, and the symmetrized relative entropy or KL-divergence.

However, the distance measures just mentioned are all limited in that they

are based on evaluating the divergence in probability assigned to a given word between the two distributions. For example, if the first segment being considered has many occurrences of "sport", then a segment making no mention of "sport" but mentioning "baseball" frequently would be assigned the same similarity score as a segment not mentioning anything relevant to sports at all.

Mutual Information for Words

Clearly, what is required is a measure of closeness between words. Various similarity measures have been suggested, such as those obtained by mining thesaurus or dictionary entries [63, 46, 71]. One powerful indicator of word similarity is co-occurrence in speech or text segments known to be topic-coherent. A measure that captures this intuition is *mutual information*.

Let V be the vocabulary, and $x, y \in V$ be two words. If T is a training corpus, then let $C_T(x, y)$, $C_T(x)$, and $C_T(y)$ be the empirical probabilities of x and y appearing together, and that of x and y appearing, in T, respectively. The pointwise mutual information (PMI) between x and y is then defined as

$$\mathrm{PMI}(x, y) = \log \frac{C_T(x, y)}{C_T(x) C_T(y)}, \tag{4.5}$$

The definition of "appearing together" can be interpreted to mean proximity in the word stream (see e.g., [22]). However, since topic segmentation is our task, we assume that our training corpus T is pre-segmented into topic-

coherent segments, and we say that x and y appear together when they appear in the same segment.

The logarithm in Equation 4.5 is customarily used due to connections with well-understood quantities in information theory, such as entropy. However, since logarithm is a monotone function, dropping it in the above formula does not change the ordering of word pairs and results in the similarity measure we shall define below being a positive definite symmetric (PDS) kernel. Thus, our similarity score between words is

$$\text{sim}(x, y) = \frac{C_T(x, y)}{C_T(x)C_T(y)}. \tag{4.6}$$

In past work [18, 78], some authors have referred to this quantity as *interest*, however, we shall refer to it simply as similarity.

Similarity Measure of Natural Language Texts

Our goal is to design a segmentation quality measure that penalizes segments spanning multiple topics while rewarding segments that respect topic boundaries. In the following measure, we match up segments between the reference and the hypothesis segmentation. Intuitively, those segments in the hypothesis that span multiple reference segments will likely get a low similarity score when compared to either reference segment, while hypothesis segments respecting reference segment boundaries will receive a high similarity score. Recall that we have set out with the goal of reducing the penalty of hypothesized segment

boundaries that do not exactly match reference boundaries if the segments hypothesized are close in content to the reference segments they overlap. The similarity score $\text{sim}(\cdot, \cdot)$ allows us to quantify this intuition.

If a and b are two segments of text or speech, let $C_a(w)$ and $C_b(w)$ be the empirical frequencies of word w in a and b, respectively, which when properly normalized (see Section 4.4) correspond to probability distributions over words. We will evaluate the total similarity of a pair of observations a and b as

$$K(a, b) = \sum_{w_1 \in a, w_2 \in b} C_a(w_1)\ C_b(w_2)\ \text{sim}(w_1, w_2) \tag{4.7}$$

Let the vocabulary be $V = \{w_1, w_2, \ldots, w_n\}$. Let A and B be the column vectors of empirical word frequencies such that $A_i = C_a(w_i)$ and $B_i = C_b(w_i)$ for $i = 1, \ldots, n$. Let $\mathbf{K}$ be the matrix such that $\mathbf{K}_{i,j} = \text{sim}(w_i, w_j)$. The similarity score can then be written as a matrix operation,

$$K(a, b) = A^\top \mathbf{K} B. \tag{4.8}$$

A normalization yields

$$K_{norm}(a, b) = \frac{A^\top \mathbf{K} B}{\sqrt{(A^\top \mathbf{K} A)(B^\top \mathbf{K} B)}}, \tag{4.9}$$

which ensures the score is in the range $[0, 1]$ and for any input a, the self-similarity is $K_{norm}(a, a) = 1$.

In the following, we show that the general measure of text similarity K_{norm} is a positive definite symmetric (PDS) kernel. This fact enables us to use K_{norm} to map word observations into a similarity feature space for the support vector based topic segmentation algorithm we shall present in Section 4.3.2.

Let $\mathbf{K}$ be as above a matrix of the similarity scores between words, $\mathbf{K}_{i,j} = \text{sim}(w_i, w_j)$. Then, we may make the following claim.

Proposition 4.1 *K_{norm} is a positive-definite symmetric (PDS) kernel.*

Proof. In the following, as in Equation 4.6, the empirical frequencies and expectations are computed over a training corpus T. For notational simplicity we omit the subscript T. Let 1_{w_i} be the indicator function of the event "w_i occurred" and let

$$\begin{aligned} \mathbf{K}_{ij} &= \frac{C(w_i, w_j)}{C(w_i)C(w_j)} \\ &= \frac{\mathbf{E}[1_{w_i} 1_{w_j}]}{\mathbf{E}[1_{w_i}]\,\mathbf{E}[1_{w_j}]} \\ &= \mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right]. \end{aligned} \tag{4.10}$$

Clearly K is symmetric. Recall that for two random variables X and Y, we have

$$\text{Cov}(X, Y) = \mathbf{E}[XY] - \mathbf{E}[X]\,\mathbf{E}[Y]. \tag{4.11}$$

Thus we have

$$\begin{aligned}\mathrm{Cov}\left(\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}, \frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right) &= \mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] - \mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\right]\mathbf{E}\left[\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] \\ &= \mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] - 1 \qquad (4.12)\end{aligned}$$

Next, recall that any covariance matrix is positive semidefinite, and thus for any $c_1, \ldots, c_m \in \mathbb{R}$,

$$\sum_{i,j=1}^{m} c_i c_j \,\mathrm{Cov}\left(\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}, \frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right) \geq 0. \qquad (4.13)$$

According to Equation 4.12, this can be rewritten as

$$\sum_{i,j=1}^{m} c_i c_j \left(\mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] - 1\right) \geq 0, \qquad (4.14)$$

that is

$$\sum_{i,j=1}^{m} c_i c_j \,\mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] - \sum_{i,j=1}^{m} c_i c_j \geq 0. \qquad (4.15)$$

Now, let $\mathbf{1}$ and C denote column vectors of size m such that $\mathbf{1}_i = 1$ and $C_i = c_i$ for $i = 1, \ldots, m$. Then,

$$\begin{aligned}\sum_{i,j=1}^{m} c_i c_j &= Tr(CC^\top \mathbf{1}\mathbf{1}^\top) \\ &= Tr(C^\top \mathbf{1}\mathbf{1}^\top C) \\ &= Tr((C^\top \mathbf{1})^2) \geq 0. \end{aligned} \tag{4.16}$$

Combining Equations 4.15 and 4.16, we get

$$\sum_{i,j=1}^{m} c_i c_j \, \mathbf{E}\left[\frac{1_{w_i}}{\mathbf{E}[1_{w_i}]}\frac{1_{w_j}}{\mathbf{E}[1_{w_j}]}\right] \geq \sum_{i,j=1}^{m} c_i c_j \geq 0. \tag{4.17}$$

This shows that the matrix $\mathbf{K}$ is positive semi-definite.

Now, if $K(a,b) = A^\top \mathbf{K} B$, where A denotes the column vector of the counts for the w_i's, $A = (C_a(w_1), \ldots, C_a(w_N))^\top$, and similarly with B, then

$$K(a,b) =< \mathbf{K}^{1/2} A, \mathbf{K}^{1/2} B >, \tag{4.18}$$

so, K is a PDS kernel. Normalization preserves PDS, so K_{norm} is also a PDS kernel. □

4.2.3 New Topic Segmentation Quality Measure

This property of K_{norm} just discussed enables us to map word observations into a similarity feature space for the support vector based topic segmentation algorithm we will present in Section 4.3.3. However, we first use this general measure of similarity for text to create a topic segmentation quality measure

that we call the Topic Closeness Measure (TCM). TCM overcomes the limitations of CoAP discussed in Section 4.2.1, and as we shall see in Section 4.5, correlates strongly with CoAP in empirical trials.

Let k and l be the number of segments in the reference and hypothesis segmentation, respectively. Additionally, let $r_1, \ldots, r_k$ and $h_1, \ldots, h_l$ be the segments in the reference and hypothesis segmentation, respectively. $Q(i, j)$ quantifies the overlap between the two segments i, j. In this work, $Q(i, j)$ is the indicator variable that is one when reference segment i overlaps with hypothesis segment j, and zero otherwise. However, various other functions can be used for Q, such as the duration of the overlap or the number of overlapping sentences or utterances. Similarly, other similarity scoring functions can be incorporated in place of K_{norm}. The topic closeness measure (TCM) between the reference segmentation R and the hypothesis segmentation H is defined as

$$\text{TCM}(R, H) = \frac{\sum_{i=1}^{k} \sum_{j=1}^{l} Q(i, j) K_{norm}\left(r_i, h_j\right)}{\sum_{i=1}^{k} \sum_{j=1}^{l} Q(i, j)}. \tag{4.19}$$

TCM is a more desirable measure of topic closeness than CoAP for a number of reasons. Like CoAP, TCM is in the range $[0, 1]$, and is symmetric in the sense the if the reference and hypothesis segmentations are exchanged the score is the same. However, unlike CoAP, through the use of a text similarity measure, TCM considers not only the placement of the topic boundaries but also the closeness of the content of the segments being separated by the boundaries. Additionally, the use of TCM is not dependent on the window

size parameter k used in previous measures. Finally, TCM is applicable in the same form to topic segmentations over text and speech streams.

TCM does consider the placement of topic boundaries, and accordingly, accomplishes the goal of CoAP – to penalize false positive and false negative segmentations. For example, adding a spurious boundary (i.e., one that separates two segments of the same topic) in a hypothesis segmentation would add one to l and would thus be penalized by the extra contribution to the normalization term $\sum_{i=1}^{k}\sum_{j=1}^{l} Q(i,j)$. Deleting a boundary between two different-topic segments is also penalized because the similarity score K_{norm} between the combined segment and the overlapping reference segments would be decreased.

4.3 Topic Segmentation Algorithms

As discussed in Section 4.1.1, there is a body of past work on topic segmentation algorithms using the general notion of similarity of adjacent text or speech windows. Various similarity measures are used by these algorithms, such as the cosine distance, possibly weighted by the tf–idf score. However, as we have pointed out in Section 4.2.2, these similarity measures are limited in that they consider only the frequencies of individual words, rather than those of related word pairs.

As in [47], the segmentation algorithm presented in this work is based on word-pair similarity. We adopt the general measure K_{norm} of Equation 4.9 as a method of evaluating similarity of two segments of text or speech.

In the following, we first suggest a straightforward and intuitive algorithm which places topic boundaries in the observation stream where K_{norm} between the two windows on either side of the boundary is small. As K_{norm} also underlies the TCM segmentation quality score presented in Section 4.2.3, this approach attempts to increase the TCM of the hypothesized segmentation. We then note that the performance of such an algorithm can be compromised by outlier sentences or utterances in the observation sequence, and suggest to use a compact support-vector based description to obtain a summary of a window of text or speech, producing a second, more robust, algorithm.

4.3.1 A General Description of Similarity-based Segmentation Algorithms

As we have already discussed in detail, topic segmentation algorithms here as well as in the literature compare adjacent windows of observations. Let the input be a sequence of observations $T = (x_1, \ldots, x_m)$. Then a topic segmentation algorithm must decide the set b of topic boundaries in T, that is the set of indices i such that x_i and x_{i+1} belong to different topics. For $i \in \{\delta, \ldots, m\}$, we will refer to a *window* of observations of size δ ending at i as the set $w_i = \{x_{i-\delta+1}, \ldots, x_i\}$. The windowing of an observation stream is illustrated in Figure 4.3. Let $s_i = s(w_i, w_{i+\delta})$ be either a similarity score or a distance between w_i and $w_{i+\delta}$. If s_i is a similarity score, we may hypothesize a segment boundary where the similarity signal dips below a global threshold to define

the boundary set $b = \{i : s_i < \theta\}$. Because the range of s on either side of a true boundary might vary, a more robust segmentation algorithm is to look for local extrema in s. This is accomplished by passing a window of size δ over s and hypothesizing boundaries where minima or maxima occur, depending on whether s is a similarity or distance score.

Figure 4.3: An illustration of windowing a stream of observations. Each square represents an observation, and the rectangle represents the current position of the window. To advance the window one position, the window is updated to add the observation marked with + and to remove that marked with −.

To denote the minimum and maximum of a range of s values, let $\text{rmax}(s, i, j) = \max(s_i, \ldots, s_j)$ and $\text{rmin}(s, i, j) = \min(s_i, \ldots, s_j)$. If s is a similarity score then we obtain a segmentation algorithm by detecting local minima in s, and applying the absolute threshold θ to each local minimum,

$$b = \{i : s_i < \theta \wedge s_i = \text{rmin}(s, i - \lfloor \delta/2 \rfloor, i + \lfloor \delta/2 \rfloor)\}. \tag{4.20}$$

If s is a distance function then we detect local maxima in s,

$$b = \{i : s_i > \theta \wedge s_i = \text{rmax}(s, i - \lfloor \delta/2 \rfloor, i + \lfloor \delta/2 \rfloor)\}. \tag{4.21}$$

Equations 4.20 and 4.21 yield a general formulation for topic segmentation algorithms based on distance or similarity functions. To arrive at the first algorithm used in this work, we let $s_i = K_{norm}(w_i, w_{i+\delta})$ and apply Equation 4.20. This simple search for local extrema, combined with the use of K_{norm} to evaluate similarity, results in a novel segmentation algorithm, to which we will refer as the similarity-based segmentation algorithm.

4.3.2 Support Vector Description of Speech and Text

Intuitively, a new topic occurs in the stream of text or speech when the word content changes drastically from one window of observations to the next. The topic segmentation algorithms based on similarity among word frequencies, such as that based on K_{norm} just mentioned, already reflect this intuition because two adjacent windows with significantly different word distributions would be assigned a low similarity. However, using the verbatim empirical word distribution within a window is problematic because there might be filler or off-topic content within a generally topic-coherent stream of observations. For example, consider the following two examples:

1. *A powerful explosion tore through a cafe frequented by Russian soldiers, south of Chechnya's capital Grozny, Sunday, killing at least eight people, including the owner of the cafe and four Russian soldiers. Russia's Interfax News Agency quotes security sources as saying Chechen rebels have claimed responsibility for the attack. That's our news summary till*

now. I'm David Coller, VOA News.

2. *the program one five emmys and carson was awarded the presidential medal of freedom and all the signed a law in nineteen ninety two more than fifteen million viewers tuned in to watch and say goodbye issue very hard.*

Snippet 1 is a human transcript and snippet 2 was produced by a speech recognizer. Both are derived from the TDT corpus. It is easy to see that the last two sentences of snippet 1 are off-topic. If the following news story also has a different topic, but also has similar filler text, this might increase the similarity of the two stories, thus increasing the likelihood of a segmentation error. Additionally, snippet 2 clearly contains off-topic content due to speech recognition errors[1].

To combat this drawback of computing similarity based on raw word statistics, we desire a compact description of the text or speech that is able to separate the distribution of the majority of the observations from the noise or outliers. The descriptors of Tax and Duin [77] attempt to find a sphere in feature space that encloses the true data from a given class but excludes outliers within that class and data from other classes. An alternative approach of Schölkopf et al. [74] posits this task as separating data from the origin in feature space, a problem that is equivalent to the spheres problem for many

[1]The reference transcription for the second snippet is: *The program won five Emmys and Carson was awarded the Presidential Medal of Freedom. When he signed off in 1992 more than 50 million viewers tuned in to watch him say good bye. I bid you a very heart [end of utterance] felt good night.*

kernels, including the one used in this work. This problem is often referred to as one-class classification, and because the problem formulation resembles that of support vector machines (SVM) [23, 79], often as the one-class SVM. We adopt the more intuitive spheres formulation.

More formally, given a set of observations $x_1, \ldots, x_m \in \mathcal{X}$, our task is to find a ball or sphere that, by enclosing the observations in feature space, represents a compact description of the data. We assume the existence of mapping of data observations into a feature space, $\Phi\colon \mathcal{X} \mapsto F$. This results in the existence of a kernel operating on a pair of observations, $K(x, y) = \Phi(x) \cdot \Phi(y)$. A sphere in feature space is then parametrized by a center $c \in F$ and radius $R \in \mathbb{R}$. Our optimization allows each observation x_i to lie outside the sphere by a distance ξ_i, at the cost of incurring a penalty in the objective function. This is illustrated in Figure 4.4. The optimization problem written in the form of [74] is

$$\begin{aligned} \min_{R\in\mathbb{R},\boldsymbol{\xi}\in\mathbb{R}^m,c\in F} \quad & R^2 + \tfrac{1}{\nu m}\textstyle\sum_i \xi_i \\ \text{subject to} \quad & \|\Phi(x_i) - c\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0 \text{ for } i \in [1, m]. \end{aligned} \tag{4.22}$$

The objective function attempts to keep the size of the sphere small, while reducing the total amount by which outlier observations violate the sphere constraint. The parameter ν controls the tradeoff between these two goals. Using standard optimization techniques, we may write the Lagrangian of this

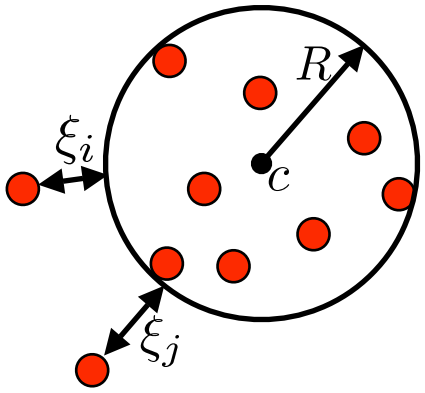


Figure 4.4: An illustration of the support vector data description algorithm in two-dimensional feature space. The sphere (c, R) found by the algorithm contains eight out of ten observations. The two observations x_i and x_j not contained by the sphere are incur a slack penalty of ξ_i and ξ_j, respectively. The three observations on the sphere boundary are the support patterns.

optimization problem utilizing Lagrangian variables $\alpha_i \geq 0, i \in [0, m]$. Solving for c, we obtain

$$c = \sum_i \alpha_i \Phi(x_i). \tag{4.23}$$

Substituting this back into the primal problem of Equation 4.22, we obtain the dual problem, in which the kernel k takes the place of dot products between training patterns. The dual problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \textstyle\sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{m} \alpha_i K(x_i, x_i) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \tfrac{1}{\nu m}, \quad \textstyle\sum_{i=1}^{m} \alpha_i = 1. \end{aligned} \tag{4.24}$$

By substitution into the Equation of a sphere in feature space, the classifier then takes the form

$$f(x) = \text{sgn}\left(R^2 - \sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_i, x_j) + 2\sum_{i=1}^{m} \alpha_i K(x_i, x) - K(x, x)\right) \quad (4.25)$$

The resulting data description is a combination of the support patterns $x_i : \alpha_i \neq 0$. The radius R may be recovered from the classifier as

$$R = \sqrt{\sum_{i,j=1}^{l} \alpha_i \alpha_j K(x_i, x_j) - 2\sum_{i=1}^{l} \alpha_i K(x_i, x_{sv}) + K(x_{sv}, x_{sv})}, \quad (4.26)$$

where x_{sv} is any support pattern.

Finally, we note that for any kernel K such that $K(x, x)$ is a constant, the sphere problem described above has the same solution as the separation from the origin problem known as the one-class SVM, as is pointed out in [74]. For our kernel K_{norm} of Equation 4.9, $K_{norm}(x, x) = 1$, thus the condition for equivalence is met and thus our geometric description may be viewed as an instance of either problem.

4.3.3 Sphere-based Topic Segmentation

The sphere-based support vector method yields a compact geometric description of text and speech segments. In contrast to using simple word frequencies,

this method is designed to be robust to the presence of outliers in the sentence or utterance stream. It also yields a natural geometric formulation for comparing two sets of observation streams, as follows.

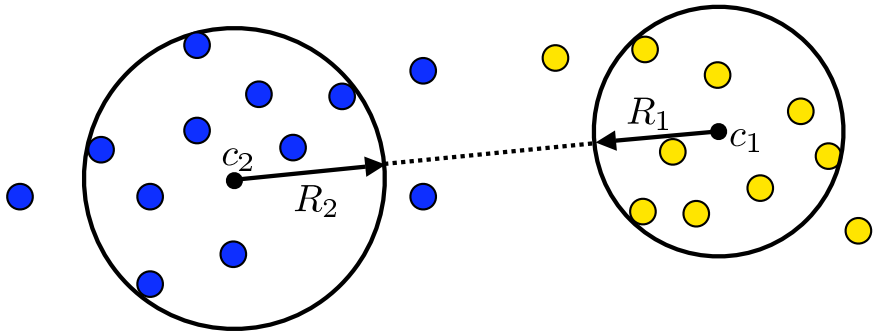


Figure 4.5: An illustration of two sets of observations being compared in feature space based on their sphere descriptors. The dashed line indicates the shortest distance between the two spheres.

We compute a distance between two sets of text or speech streams by calculating the geometric shortest distance in feature space between the two spheres representing them. This comparison is illustrated in Figure 4.5. Assume that we are comparing two windows of observations w_1 and w_2. Let $x_{1,1}, \ldots, x_{m_1,1}$ and $x_{1,2}, \ldots, x_{m_2,2}$ be the word frequency counts, and let $\alpha_{1,1}, \ldots, \alpha_{m_1,1}$ and $\alpha_{1,2}, \ldots, \alpha_{m_2,2}$ be the dual coefficients resulting from solving the optimization problem of Equation 4.24, for w_1 and w_2, respectively. Then the resulting support vector descriptions are represented by spheres (c_1, R_1) and (c_2, R_2) which are found through the application of Equations 4.23 and 4.26. Then, the distance between the centers of the spheres is the Euclidean distance in the feature space. Since the mapping $\Phi(\cdot)$ is implicitly expressed through the

kernel $K(\cdot,\cdot)$, the distance is also computed by using the kernel, as

$$\begin{aligned}\|c_1 - c_2\|^2 = \|c_1\|^2 + \|c_2\|^2 - 2\|c_1\|\|c_2\| = \\ \sum_{i,j=1}^{m_1} \alpha_{i,1}\alpha_{j,1}K(x_{i,1}, x_{j,1}) \\ + \sum_{i,j=1}^{m_2} \alpha_{i,2}\alpha_{j,2}K(x_{i,2}, x_{j,2}) \\ - 2\sum_{i=1}^{m_1}\sum_{j=1}^{m_2} \alpha_{i,1}\alpha_{j,2}K(x_{i,1}, x_{j,2}).\end{aligned} \tag{4.27}$$

The shortest distance between the spheres is simply obtained by subtracting the radii to obtain

$$\text{dist}(w_1, w_2) = \|c_1 - c_2\| - (R_1 + R_2). \tag{4.28}$$

Note that it is possible for the sphere descriptors to overlap, and in fact in practice this is frequently the case for adjacent windows of observations. In this case, the quantity $\text{dist}(w_1, w_2)$ will be negative. Hence, it does not always represent a geometric margin, but nevertheless it can be viewed as an an algebraic measure of separation even if it is negative.

By using the kernel K_{norm} of Equation 4.9, we map the observations in each window into a high-dimensional word similarity space. Distances between a pair of patterns in this space represent the divergence in similarity between word pairs across two patters computed according to the mutual information-

based word similarity score of Equation 4.6. As we have shown in Proposition 4.1, K_{norm} is a positive definite symmetric kernel, which guarantees the convexity of the dual optimization problem of Equation 4.24.

To construct our final discriminative topic segmentation algorithm, we simply set $s_i = \text{dist}(w_1, w_2)$, and hypothesize segment boundaries in the observation at local maxima in the s signal above the threshold θ, $b = \{i : s_i > \theta \wedge s_i = \text{rmax}(s, i - \lfloor \delta/2 \rfloor, i + \lfloor \delta/2 \rfloor)\}$. In the following, we will refer to this algorithm as the sphere-based or the support-vector topic segmentation algorithm.

4.4 Lattice-based Topic Analysis

There is a significant literature on topic analysis of spoken language (e.g., [84, 12]). However, the majority of these works use only the one-best recognition hypothesis as input to a topic labeling and/or segmentation algorithm. Since modern recognizers use the Viterbi sub-optimal beam search to approximate the most likely word sequence, there is always a beam of almost-as-likely hypotheses being considered. As a result, it is possible to produce a list, or more compactly, a graph, of the top hypotheses along with their likelihoods. Such a graph is known as a *lattice* (see Figure 4.6 for an illustration). Lattices can be represented with finite automata, which enables compactness, lookup efficiency, and easy implementation of necessary lattice manipulations with general automata algorithms [54]. A recent work [35] demonstrated an improvement using word and phoneme lattices for assigning topic labels to

pre-segmented utterances in isolation. In this work we focus exclusively on word lattices.

In our topic segmentation algorithms, we use two information sources derived from lattices, expected counts and confidence scores, as follows. The input to all the algorithms described in this paper is a sequence of bag-of-word observations. Let a be a set of observations and $f_a(w)$ the set of word weights to be computed. If no lattice information is available then a is represented by a bag of the l words appearing in its one-best hypothesis, $(w_1, \ldots, w_l)$. Then $f_a(w) = \sum_{i=1}^{l} 1_{w_i=w}$.

If a word lattice is available, we may compute for each word in the lattice a total posterior probability, or expected count, accumulated over all the paths that contain that word. If V is the vocabulary the expected count of the word w according to a stochastic lattice automaton A, i.e., that with the weights of all the paths summing to one, is

$$f_a(w) = \sum_{u \in V^*} |u|_w [\![A]\!](u), \tag{4.29}$$

where $|u|_w$ is the number of occurrences of word w in string u and $[\![A]\!](u)$ is the probability associated by A to string u. The set of expected counts associated with all the words found in the lattice can be computed efficiently [5]. We also compute word-level confidence scores for the one-best hypothesis using a logistic regression classifier. The classifier takes two features as input, the word expected counts just mentioned and a likelihood ratio between the standard

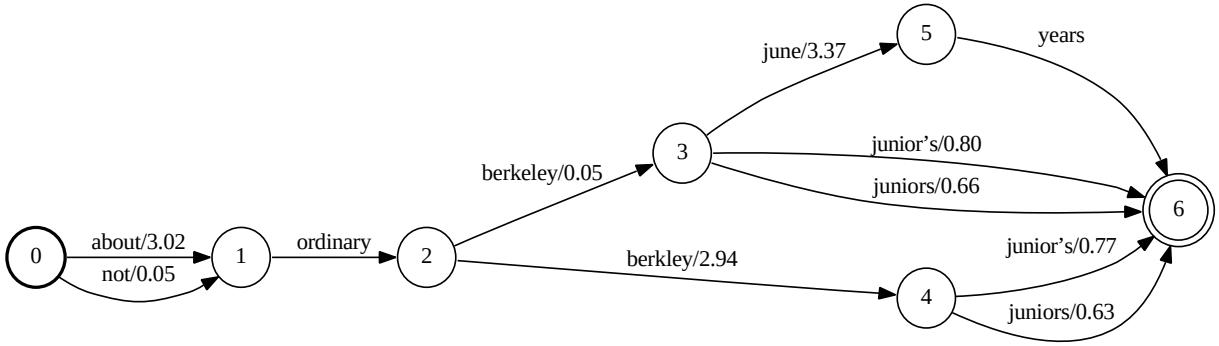


Figure 4.6: An example of a speech recognition word lattice. The weights indicate negative log-likelihoods.

recognizer with full context-dependent acoustic models and a simple recognizer with context-independent models. If each word w_i has an associated confidence $c(w_i)$, then

$$f_a(w) = \sum_{i=1}^{l} c(w_i) 1_{w_i=w}. \tag{4.30}$$

Finally, the word weights $f_a(w)$ are normalized to produce the counts that serve as input to the segmentation algorithm,

$$C_a(w) = \frac{f_a(w)}{\sum_{w \in V} f_a(w)}. \tag{4.31}$$

4.5 Experiments

Our experimental work has been in the context of the English portion of the TDT corpus of broadcast news speech and newspaper articles [44]. The speech corpus consisted of 447 news show recordings of 30-60 minutes per show, for a total corpus size of around 311 hours. For the experiments involving speech data, we used 41 and 69 shows picked from the Voice of America English News Program (VOA_ENG) and MS-NBC News With Brian Williams (MNB_NBW), containing 957 and 1,674 stories, for development and testing, respectively. The 337 shows from other sources were used for training. The training shows contained 6,310 stories, and were annotated with human segmentations and transcriptions for each story. Certain stories were also annotated by hand with topics such as "Earthquake in El Salvador," but these

labels were not used in the model training. For those experiments using text data, a total of 1,314 training news streams were used, including the human transcripts of the 337 news shows already mentioned as well as 977 non-speech news streams, from the New York Times (NYT_NYT) and the Associated Press (APW_ENG), with a total of 15,110 non-speech stories.

4.5.1 Text Similarity Evaluation

Figure 4.7 contains a listing of the 100 word pairs x, y with the top similarity score $\text{sim}(x, y)$, after the application of Porter stemming [67]. Note that many word pairs fit human intuition about words that are expected to occur together in topic-coherent documents (e.g., "banjo, bluegrass", "extraterrestri, ionospher", "cystic, fibrosi").

The general text similarity measure K_{norm} of Equation 4.9 is a key component underlying TCM. To evaluate it empirically, we computed K_{norm} between all pairs of stories with human topic labels. With 291 stories, there were 3,166 same-topic story pairs and 39,172 different-topic pairs in our experiment. The average pairwise similarity between different-topic story pairs was 0.2558 and that between same-topic story pairs was 0.7138, or around 2.8 times greater. This indicates that our text similarity measure is a good indicator of topical similarity between two segments of text or speech.

We also explored the correlation between K_{norm} and true segmentation boundaries by processing each show's transcription by sliding a window of $\delta = 6$ sentences along the text and accumulating the word frequencies within

krono,thirti 8801
darden,milken 4885
banjo,bluegrass 4314
kristol,luntz 3675
dakar,senegales 3373
jeanni,unz 3335
espi,rostenkowski 3308
biomass,goodwin 3308
fri,powder 3257
kinda,portia 3216
diari,portia 3101
extraterrestri,ionospher 3063
lamott,pastri 3040
pack,powder 2983
boyc,opal 2968
jeanni,opri 2916
broder,darden 2858
cystic,fibrosi 2858
comet,meteor 2770
quartet,thirti 2670
lift,powder 2614
menotti,solstic 2573
epilepsi,sinus 2573
powder,trail 2467
jerrold,kessel 2389
rushdi,salman 2378
basal,lesion 2370
biomass,turbin 2358
opal,superstit 2339
ionospher,telescop 2315
colosio,salina 2297
mack,purcel 2270
caramel,lamott 2237
bombai,fiftieth 2237
centimet,metric 2189
hydrogen,iceland 2167
boyc,superstit 2159
fanni,mae 2152
ionospher,piggyback 2144
elmo,homesick 2144
gal,portia 2144
powder,taho 2102
antiterrorist,nintendo 2075
krono,quartet 1993
minivan,windstar 1898
gobi,soya 1893
fri,pack 1888
eleph,fineman 1882
seneg,senegales 1765
jammu,shia 1715
koppel,nightlin 1686
dakar,seneg 1665
biomass,nutshel 1654
kurdish,kurdistan 1633
coughlan,goldberg 1625
fri,lift 1616
kurdistan,mule 1608
elmer,fiftieth 1608
casa,opal 1608
herbicid,soya 1602
dalai,lama 1589
hankin,solar 1527
fri,trail 1524
lift,pack 1496
boyc,casa 1484
aidid,somalia 1476
kurd,kurdistan 1474
limo,tranc 1470
acr,powder 1467
opium,poppi 1447
fiftieth,kilo 1429
aidid,allah 1429
brando,marlon 1413
pack,trail 1412
atkinson,foss 1357
fog,rosco 1349
bean,soya 1340
firfer,insulin 1319
crocodil,strieker 1319
akita,tanzanian 1286
menendez,mistrial 1286
archbishop,monsignor 1261
fri,taho 1260
coker,wilei 1257
altman,vultur 1247
krikalev,soyuz 1246
lift,trail 1238
consul,dominican 1233
diamet,ionospher 1225
gel,sinu 1225
opium,triangl 1212
atol,reef 1212
pack,taho 1202
halperin,repeal 1201
catalogu,constantin 1201
condom,ohh 1197
malaria,mosquito 1189
calori,litman 1187
chile,mercosur 1180
steroid,vial 1177

Figure 4.7: The 100 most similar word pairs in our training corpus. For each word pair x, y, we list the Porter stemmed words along with the score $\text{sim}(x, y)$ obtained in our training process.

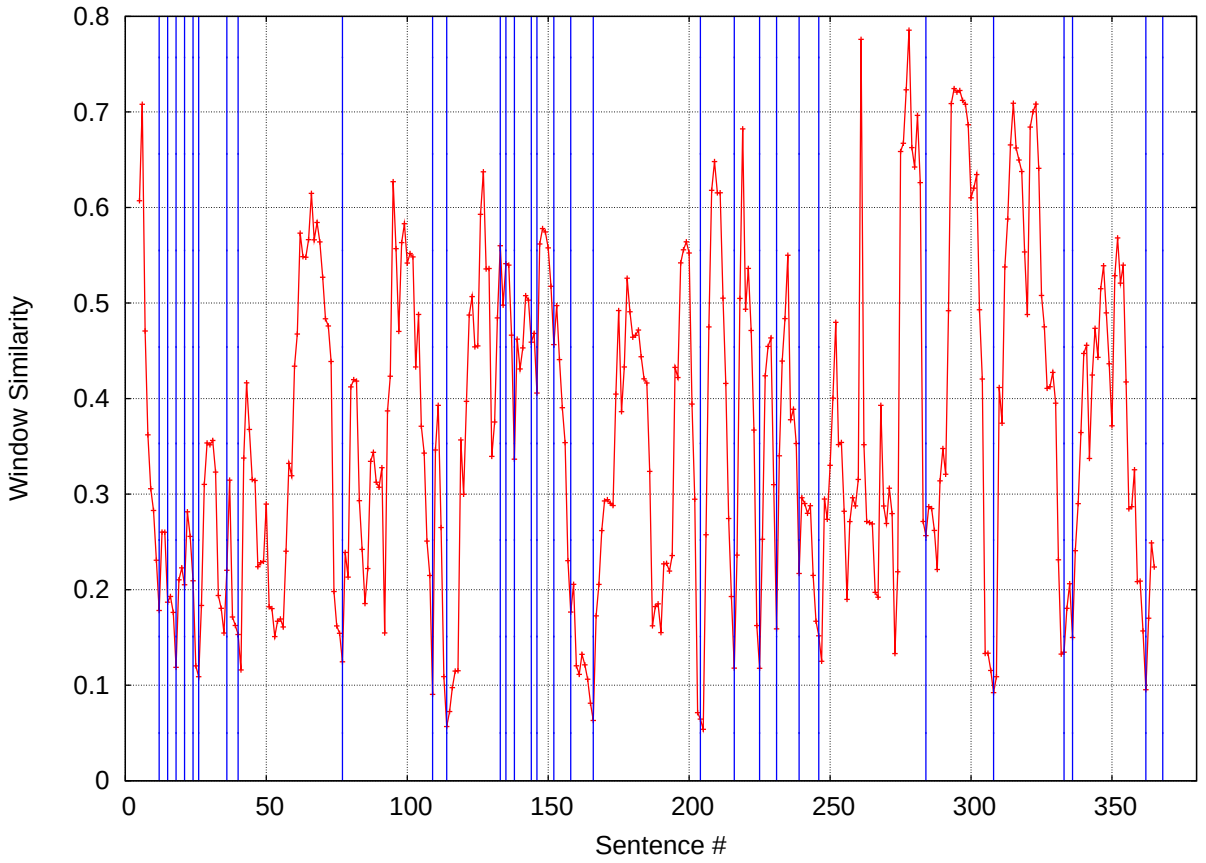


Figure 4.8: The window distance $K_{norm}(w_t, w_{t+\delta})$ for a representative show. The vertical lines are true story boundaries from the human-labeled corpus. A line at sentence t means that sentence $t+1$ is from a new story.

each window. For each sentence t, let w_t be the window ending at sentence t. Figure 4.8 displays the plot of $K_{norm}(w_t, w_{t+\delta})$ for a representative show. As this figure illustrates, true topic boundaries are extremely well correlated with local minima in the similarity score. Similar trends are observed with other shows in the corpus.

4.5.2 Topic Segmentation Evaluation

For the speech experiments, the audio for each show was first automatically segmented into utterances, while removing most non-speech audio, such as music and silence [1]. Each utterance was transcribed using the Google large-vocabulary continuous speech recognizer. This recognizer (the baseline system of [1]) used standard PLP cepstral features, a vocabulary of about 71K words, GMM-based triphone HMM acoustic models, and smoothed 4-gram language models pruned to about 8M n-grams. Both the acoustic and language models were trained on standard Broadcast News (BN) corpora. The word error rate of this recognizer on the 1997 BN evaluation set was 17.7%.

The vocabulary for all algorithms consisted of a subset of 8,821 words. This was constructed by starting with the set of words seen in the recognizer transcription of the training data, applying Porter stemming [67], removing a stoplist of function and other words not likely to indicate any topic, and keeping only those words occurring more than five times. The HTMM was trained with an Expectation Maximization (EM) algorithm initialized with random values for model parameters. Our HTMM had 20 topics and hyper-

parameters α and β set as in [75]. To minimize the possibility of a particular randomization overfitting the test data, we ran 20 trials of model training and testing and picked the model that had the best segmentation quality on the development data set.

For the algorithms based on K_{norm}, the parameter set included the threshold θ and the window size δ for both algorithms. The sphere-distance based algorithm was additionally parametrized by the regularization tradeoff parameter ν. For both algorithms, we performed parameter selection on the development data set. Since these two algorithms rely on the use of the co-occurrence based word similarity score, their training consists of calculating word and word pair frequencies over a corpus of text segmented into topic-coherent chunks. The input to the training stage of all the segmentation algorithms were human transcriptions of the speech news broadcasts as well as non-speech news sources, as detailed above.

Table 4.1 gives segmentation quality scores for degenerate segmentations with boundaries decided by a fair coin toss (Random), all possible boundaries (Full), and no boundaries at all (None).

We trained two separate HTMMs, the first using reference text as training data, and the second using the one-best transcription of the training data. Since these two algorithms rely on the use of the word similarity score of Equation 4.6, their training consists of calculating word and word pair frequencies over a corpus of text segmented into topic-coherent segments. The input to this training procedure were human transcriptions of the speech news

Table 4.1: CoAP and TCM measured on degenerate segmentations.

Input Type	**CoAP**	**TCM**
Text Random	50.4%	58.4%
Text Full	50.4%	51.8%
Text None	49.6%	56.2%
One-best Random	50.8%	48.8%
One-best Full	51.0%	43.0%
One-best None	49.1%	52.9%

Table 4.2: Topic segmentation quality as measured with CoAP and TCM.

Input Type	**Algorithm**	**Quality Measure**	
		CoAP	**TCM**
Text	HTMM	66.9%	72.6%
	Sim	72.0%	75.0%
	SV	**76.6%**	**77.7%**
One-best	HTMM	65.0%	61.5%
	Sim	60.4%	62.8%
	SV	**68.6%**	**66.0%**
Counts	HTMM	65.5%	62.4%
	Sim	59.4%	63.4%
	SV	**68.5%**	**66.5%**
Confidence	HTMM	68.3%	64.2%
	Sim	59.7%	63.8%
	SV	**69.2%**	**66.8%**

Table 4.3: Topic segmentation quality for HTMM when trained on speech data.

Input Type	**CoAP**	**TCM**
One-best	67.3%	62.8%
Counts	69.7%	64.1%
Confidence	68.8%	64.9%

broadcasts as well as non-speech news sources, as detailed above. Since reference segmentations into topic-coherent segments of our speech transcriptions of the training data were not available, we did not train these algorithms on speech data. However, testing was conducted on speech and text sources across all conditions and algorithms. The results for the HTMM trained on speech data are given in Table 4.3, while the results for all three algorithms trained on text data are given in Table 4.2. The results for the similarity-based segmentation algorithm of Section 4.3.1 and the support-vector based algorithm of Section 4.3.3 are denoted as Sim and SV, respectively. Across all algorithms, we tested on the reference text (Text), as well as three modifications of speech transcriptions, one-best transcriptions with frequencies as weights (One-best), one-best transcriptions weighted with confidence scores (Confidence), and words weighted with lattice expected counts (Counts).

4.5.3 Discussion

In the following, all comparisons are made in terms of relative error improvements. TCM error and CoAP error are both defined as $1-\text{TCM}$ and $1-\text{CoAP}$, respectively. We make the following observations about the results just mentioned.

1. Segmentations produced by all three segmentation algorithms significantly outperform degenerate segmentations by both measures.

2. The similarity-based segmentation algorithm does not show a consistent

improvement over HTMM in the cases where the inputs are derived from speech recognition data. However, for text test data, this algorithm outperforms HTMM. This results verifies empirically that the variability in the word distribution introduced by speech recognition errors is a challenge for similarity-based segmentation algorithms.

3. While the performance of the similarity-based algorithm degrades in the presence of speech recognition errors, the SV algorithm outperforms HTMM and Sim significantly across all test conditions by both measures. For example, compared to HTMM, segmentation error is reduced by 29.3% and 18.6% when we test on text data and by 10.3% and 11.7% when we test on one-best speech data, with CoAP and TCM, respectively. The fact that the SV algorithm beats HTMM substantially while the Sim algorithm fails to do so serves as strong empirical evidence that SV yields an effective geometric description of text and speech observations. This geometric formulation allows the SV algorithm to overcome noise introduced by speech recognition errors and outliers in the observation stream while capturing the true topicality information of the window being considered.

4. Improvements are additionally demonstrated by using lattice information in segmentation algorithms. For example, for HTMM, lattice counts yield a 2.3% and 3.5% relative improvement with text and speech training, respectively, in TCM error compared to the one-best baseline, and

1.4% and 7.3% in terms of CoAP. Confidence scores also yield improvements with both measures, 7.0% and 5.6% relative by TCM and 9.4% and 4.6% by CoAP. The SV algorithm also achieves improvements when confidence scores are used, of 1.9% and 2.4% by CoAP and TCM over the one-best baseline. In other cases, while both the SV and Sim algorithm consistently exhibit an improved TCM score over the one-best case when lattice information is used, this is not the case when the quality is measured with CoAP. This can possibly be attributed to the inconsistency in the definition of the CoAP measure in the case of speech transcripts (see the next item and Section 4.2.1). An alternative explanation is that the Sim algorithm lacks the robustness to handle the errors in the alternative hypotheses considered when lattice information is used. However, since both the HTMM and SV algorithms improve consistently when lattice-based confidence scores are used, we conclude that lattice information is a helpful feature for topic segmentation of speech recognition transcripts.

5. TCM is an effective measure of topic segmentation quality. Qualitatively, its output is generally correlated with that of CoAP. One interesting comparison to make is that between the Text case and the One-best case. Intuitively, we can expect topic segmentation on the reference transcriptions to be a much easier task than that on the output of a speech recognizer, due to the transcription errors present in the latter. Indeed, for HTMMs, error reductions from One-best to Text are achieved, but

5.4% as measured by CoAP, and 28.8% by TCM. For SV, the reduction is 25.5% by CoAP and 34.4% by TCM. This asymmetry can possibly be attributed to the mismatch between the CoAP used for text and that used for speech mentioned in Section 4.2.1. Past experiments (e.g., [28]) have observed that only small degradations in topic segmentation quality are attributed to speech recognition errors. However, since these trials were evaluated with CoAP-style quality measures, this suggests that the quality degradations may have been understated.

4.6 Summary

In this chapter, we have made several contributions to topic analysis of streams of speech and text. The first is to give a new measure of topic segmentation quality that overcomes major limitations of past evaluation techniques. Unlike previous quality measures, TCM applies generally to both speech and text sources, does not depend on a fixed window size, and considers similarity between segments labeled as topic-coherent, rather than simply the presence or absence of a segment boundary. In empirical trials, TCM is correlated with the previous measure. Additionally, the general text similarity measure underlying TCM is empirically correlated with ground truth topic boundaries and topic assignment in a human-labeled corpus of news text and speech.

We have further outlined a general formulation for distance-based topic segmentation algorithms which builds upon and generalizes previous work in

the topic segmentation literature. We have proposed two new topic segmentation algorithms based on our general measure of topical similarity. The first algorithm functions by comparing adjacent observation windows according to a similarity measure for words trained on co-occurrence statistics. The second is based on comparing compact geometric descriptions of the adjacent windows in topic similarity feature space. We have demonstrated both algorithms to be empirically effective, with the geometric description lending robustness to the latter algorithm in the presence of noise introduced by off-topic content and speech recognition errors. This algorithm significantly and consistently surpasses in quality the segmentation produced by a hidden topic Markov model (HTMM). This result holds when segmentation quality is measured by TCM as well as by the previously used CoAP measure.

Finally, we have demonstrated that in the presence of uncertainty resulting from the use of a speech recognizer, topic segmentation algorithms can be improved by using recognition hypotheses other than that receiving the highest likelihood. We have used two information sources derived from lattices, word frequencies or counts and word confidence scores, as weights on words found in the observation stream, and have observed improvements with both.

Chapter 5

Conclusion

In this thesis, we have studied the problem of searching very large audio collections. Reflecting the difficulty of this task is the diversity of the problems encountered in this field, and in this thesis. Not only have the problems we have addressed been broad in scope, but also in nature, ranging from theoretical to algorithmic to applied. The main contributions of this thesis have been

1. A large-scale, robust, and scalable music identification system based on weighted finite-state transducers.

2. A new, substantially improved bound on the size of suffix and factor automata in terms of the size of the input automaton.

3. A matching new linear-time algorithm for the construction of suffix and factor automata.

4. A new topic segmentation quality measure that considers the closeness in content of the segments labeled as topic-coherent.

5. A new discriminative topic segmentation algorithm able to remove outliers in the presence of noise and off-topic content.

As we have discussed, all the problems we have considered are related in that they share the common challenge of uncertainty. In music identification, both the transcription of a song in terms of music sound units and the identity of a song matching a given query is uncertain. Our music phoneme selection algorithm has allowed us to learn a set of music sounds in an unsupervised way. In addition, our index implementation based on Gaussian mixture acoustic models and weighted finite-state transducers has allowed us to efficiently and robustly match the query audio to our database.

Crucial to many indexing tasks, especially those involving uncertain data, is constructing a compact data structure that is efficient to query. Suffix and factor automata meet the requirements for compactness and efficiency, as they are minimal automata and are optimal in query complexity. We have presented new bounds that guarantee that a suffix automaton or factor automaton never has more than twice the number of states than its suffix-unique input automaton. We have also given a new linear-time algorithm for constructing suffix and factor automata that substantially improves the efficiency of indexing uncertain data.

Search and indexing of spoken language is marked with uncertainty stemming from the error-prone nature of speech-to-text transcription. However, index quality and effectiveness can be improved by segmenting the speech stream by topic. Through our topic segmentation quality measure, we have provided researchers and practitioners with a new tool for evaluating the success of their segmentation algorithms. Furthermore, we have designed a new segmentation algorithm able to function well in the presence of transcription errors and other off-topic content occurring naturally in spoken language.

5.1 Future Work

5.1.1 Music Identification

In our description of previous music identification systems, we noted that many previous approaches have been based on hashing individual feature vectors computed over the music sequence. In contrast, our music identification approach builds a description of each song in terms of automatically-learned elementary music sounds, leading to a number of interesting analysis opportunities. One aspect of music phonemes that should be explored in more detail is the connection between each music phoneme and the sound it represents. Is it the case that a given music phoneme corresponds to the onset of a drum beat, or strong vocals, or just plain silence? Further research in this direction may help us discover the "correct" music phoneme inventory size to use in a music identification system.

By describing each song as a sequence of elementary music sounds, our music identification approach gives us a tool for in-depth analysis of musical structure. We began to explore this avenue of research in Section 2.8 by counting the number of song collisions for different transcription factor lengths. However, much more exploration can be done, for instance to consider what it means when two songs have a collision of a long factor – have we discovered an instance of plagiarism, or a cover song? Is looking for long factors repeated several times within a song an effective algorithm for chorus detection?

The new size bounds presented in Chapter 3 guarantee that the size of our index scales gracefully as the database size increases. However, one additional important question is whether the query time will also scale gracefully. We have observed that our index of over 15,000 songs can be queried robustly in faster than real-time. This is facilitated by our use of deterministic and minimal factor automata, which are optimal data structures in the sense that the query time is linear in the size of the query. However, in our music identification system, many competing music phoneme transcriptions are considered simultaneously by the Viterbi decoder. Increasing the song collection size substantially may increase the number of paths being considered at any given point during the decoding of a test audio snippet, hence slowing down the decoding process. Since in such a search the vast majority of the paths would be pruned away due to poor match of the acoustic models to the observations, there is reason to be optimistic that this slowdown would not be large. However, further exploration of the precise slowdown encountered would be

beneficial.

In general, even given the compactness guarantees given in this thesis, for extremely large song collections the index representation may be difficult to construct and/or search on a single machine. In these cases, it may be necessary to split the song database into several parts that can be queried in parallel. The retrieval process will then require an additional step to pick the best match among those returned. The design of such a disambiguation step would be facilitated by the availability of general automata operations, which allow straightforward manipulation of sets of alternative hypotheses. However, more work should be done to explore the specific challenges of such a system.

5.1.2 Suffix and Factor Automata

As we have mentioned, the bound on the size of the suffix and factor automata given in Corollary 3.3 is not tight in the sense that in practice, shared suffixes of length k are not expanded to produce kn states in $S(A)$ or $F(A)$. An interesting avenue of future research would be to analyze in finer detail the impact of suffix sharing on the size of the factor and suffix automata.

On the algorithmic side, as we have noted the weighted suffix automaton construction algorithm of Section 3.6 functions in the tropical semiring. We have also conjectured that this algorithm can be extended to arbitrary semirings. Proving this conjecture true would allow the generalization of this algorithms to new tasks and strengthen the algorithmic foundation of suffix and factor automata.

5.1.3 Topic Segmentation

The form of the new Topic Closeness Measure of Section 4.2.3 is extremely general in that it allows the use of different overlap scores $Q(i, j)$, for which in this work we have used a simple indicator function; as well that of the text or speech similarity measure, for which we have used K_{norm}. Future work should explore other choices. For example, $Q(i, j)$ could measure the degree of overlap in terms of time or number of words. K_{norm} could be replaced with simpler similarity scores, such as the cosine distance, or more complex ones, such as ones based on language models trained over large corpora.

The sphere-based data descriptions of Section 4.3.2 are in practice effective at reducing the effect of noise on segmentation quality, and are associated with theoretical generalization performance guarantees. However, we have not analyzed the generalization performance for the sphere-based segmentation algorithm as a whole. Exploring this would be a valuable avenue of future research.

The topic segmentation algorithms presented in this thesis can be used as an initial stage in producing a topic-wise labeling of the observation stream. After segmentation, a topic classification algorithm such as that of [35] may be used to produce a sequence of labels. This choice between pre-segmenting an observation stream before labeling the individual segments and assigning a label sequence directly to the observations presents itself in speech recognition [31], among other tasks. In our work, we have evaluated the performance of our algorithm in the topic segmentation task; however, a hybrid segmen-

tation/classification algorithm of the type just mentioned can be compared directly to topic models such as HTMM in the topic labeling task. In addition, as we have mentioned in Section 4.1.2, the general threshold-based topic segmentation algorithm in this thesis is similar to the algorithm used to pre-segment songs for acoustic model training for the music identification task. Thus, for both topic and music analysis, it would be beneficial to explore hybrid segmentation/classification approaches.

Bibliography

[1] Christopher Alberti, Michiel Bacchiani, Ari Bezman, Ciprian Chelba, Anastassia Drofa, Hank Liao, Pedro Moreno, Ted Power, Arnaud Sahuguet, Maria Shugrina, and Olivier Siohan. An audio indexing system for election video material. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4873–4876, Taipei, Taiwan, 2009.

[2] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study: Final report. In *DARPA Broadcast News Transcription and Understanding Workshop*, pages 194–218, San Francisco, California, USA, February 1998. Morgan Kaufmann Publishers, Inc.

[3] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *Conference on Research and Development in Information Retrieval (SIGIR)*, pages 37–45, Melbourne, Australia, 1998.

[4] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient experimental string matching by weak factor recognition. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 51–72, London, UK, 2001. Springer-Verlag.

[5] Cyril Allauzen, Mehryar Mohri, and Brian Roark. Generalized algorithms for constructing statistical language models. In *Meeting of the Association for Computational Linguistics*, pages 40–47, Sapporo, Japan, 2003.

[6] Cyril Allauzen, Mehryar Mohri, and Murat Saraclar. General indexation of weighted automata - application to spoken utterance retrieval. In *Meeting of the Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT/NAACL), Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval.*, pages 33–40, Boston, Massachusetts, May 2004.

[7] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: a general and efficient weighted finite-state transducer library. In *Conference on Implementation and Application of Automata (CIAA)*, pages 110–121, Prague, Czech Republic, July 2007.

[8] Michiel Bacchiani and Mari Ostendorf. Joint lexicon, acoustic unit inventory and model design. *Speech Communication*, 29:99–114, November 1999.

[9] Eloi Batlle, Jaume Masip, and Enric Guaus. Automatic song identification in noisy broadcast audio. In *International Conference on Signal and Image Processing*, Kauai, Hawaii, USA, 2002.

[10] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1-3):177–210, 1999.

[11] Adam Berenzweig, Beth Logan, Daniel P.W. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[12] David M. Blei and Pedro J. Moreno. Topic segmentation with an aspect hidden Markov model. In *Conference on Research and Development in Information Retrieval (SIGIR)*, pages 343–348. ACM Press, 2001.

[13] David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, January 2003.

[14] Anselm Blumer, Janet Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel I. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[15] Anselm Blumer, Janet Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–589, 1987.

[16] Janet A. Blumer. *Algorithms for the directed acyclic word graph and related structures*. PhD thesis, Denver University, 1985.

[17] Thorsten Brants, Francine Chen, and Ayman Farahat. A system for new event detection. In *Conference on Research and Development in Information Retrieval (SIGIR)*, pages 330–337, Toronto, Canada, 2003.

[18] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA, 1997.

[19] Pedro Cano, Eloi Batlle, Ton Kalker, and Jaap Haitsma. A review of audio fingerprinting. *Journal of VLSI Signal Processing Systems*, 41:271–284, 2005.

[20] Michael Casey, Christophe Rhodes, and Malcolm Slaney. Analysis of minimum distances in high-dimensional musical spaces. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(5):1015–1028, July 2008.

[21] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.

[22] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

[23] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[24] Michelle Covell and Shumeet Baluja. Audio fingerprinting: Combining computer vision & data stream processing. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 213–216, Honolulu, Hawaii, 2007.

[25] Michelle Covell and Shumeet Baluja. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition*, 41(11):3467–3480, November 2008.

[26] Maxime Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86, 1986.

[27] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2002.

[28] Satya Dharanipragada, Martin Franz, J. Scott McCarley, K. Papineni, Salim Roukos, T. Ward, and W.-J. Zhu. Statistical methods for topic segmentation. In *International Conference on Speech and Language Processing (ICSLP)*, pages 516–519, Beijing, China, 2000.

[29] David Eichmann and Padmini Srinivasan. A cluster-based approach to broadcast news. In James Allan, editor, *Topic detection and tracking: event-based information organization*, pages 149–174. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[30] Michel Galley, Kathleen McKeown, Eric Fosler-Lussier, and Hongyan Jing. Discourse segmentation of multi-party conversation. In *Meeting of the Association for Computational Linguistics*, pages 562–569, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[31] James Glass. A probabilistic framework for segment-based speech recognition. *Computer, Speech, and Language*, 17:137–152, 2003.

[32] Amit Gruber, Michal Rosen-Zvi, and Yair Weiss. Hidden topic Markov models. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, San Juan, Puerto Rico, 2007.

[33] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge, UK, 1997.

[34] Jaap Haitsma, Ton Kalker, and Job Oostveen. Robust audio hashing for content identification. In *Content-Based Multimedia Indexing (CBMI)*, Brescia, Italy, September 2001.

[35] Timothy J. Hazen and Anna Margolis. Discriminative feature weighting using MCE training for topic identification of spoken audio recordings. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4965–4968, Las Vegas, Nevada, USA, 2008.

[36] Marti A. Hearst. TextTiling: segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[37] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[38] Matthew Hoffman, David Blei, and Perry Cook. Content-based musical similarity computation using the hierarchical dirichlet process. In *International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, Pennsylvania, USA, September 2008.

[39] Heiko Hoffmann. Kernel PCA for novelty detection. *Pattern Recognition*, 40(3):863–874, 2007.

[40] Thomas Hofmann. Probabilistic latent semantic indexing. In *Conference on Research and Development in Information Retrieval (SIGIR)*, pages 50–57, Berkeley, California, USA, 1999.

[41] Jaakko Hollmn and Volker Tresp. Call-based fraud detection in mobile communication networks using a hierarchical regime-switching model. In *Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 889–895. MIT Press, 1999.

[42] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, 146(2):156–179, 2005.

[43] Yan Ke, Derek Hoiem, and Rahul Sukthankar. Computer vision for music identification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 597–604, San Diego, California, USA, June 2005.

[44] Junbo Kong and David Graff. TDT4 Multilingual Broadcast News Speech Corpus. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2005S11, 2005.

[45] Hideki Kozima. Text segmentation based on similarity between words. In *Meeting of the Association for Computational Linguistics*, pages 286–288, Columbus, Ohio, USA, 1993.

[46] Hideki Kozima and Teiji Furugori. Similarity between words computed by spreading activation on an English dictionary. In *Conference on European chapter of the Association for Computational Linguistics*, pages 232–239, Utrecht, The Netherlands, 1993.

[47] Hideki Kozima and Akira Ito. A scene-based model of word prediction. In *International Conference on New Methods in Language Processing (NEMLAP)*, pages 110–120, Ankara, Turkey, 1996.

[48] Ian R. Lane, Tatsuya Kawahara, Tomoko Matsui, and Satoshi Nakamura. Dialogue speech recognition by combining hierarchical topic classification and language model switching. *IEICE - Transactions on Information and Systems*, E88-D(3):446–454, 2005.

[49] David Liu and Tsuhan Chen. Unsupervised image categorization and object localization using topic models and correspondences between images. In *International Conference on Computer Vision (ICCV)*, Rio De Janeiro, Brazil, October 2007.

[50] Beth Logan and Ariel Salomon. A music similarity function based on signal analysis. In *International Conference on Multimedia and Expo (ICME)*, pages 745–748, Tokyo, Japan, August 2001.

[51] Markos Markou and Sameer Singh. Novelty detection: a review - part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.

[52] Markos Markou and Sameer Singh. Novelty detection: a review - part 2: neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003.

[53] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Five papers on WordNet. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*. MIT Press, May 1998.

[54] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[55] Mehryar Mohri. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(6):957–982, 2003.

[56] Mehryar Mohri. Statistical Natural Language Processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[57] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. Factor automata of automata and applications. In *International Conference on Implementation and Application of Automata (CIAA)*, Prague, Czech Republic, July 2007.

[58] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. Robust music identification, detection, and analysis. In *International Conference on Music Information Retrieval (ISMIR)*, Vienna, Austria, September 2007.

[59] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. Efficient and robust music identification with weighted finite-state transducers. *IEEE Transactions on Audio, Speech, and Language Processing*, 2009.

[60] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. General suffix automaton construction algorithm and space bounds. *Theoretical Computer Science*, 410(37), September 2009.

[61] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. A new quality measure for topic segmentation of text and speech. In *Conference of the International Speech Communication Association (Interspeech)*, pages 2743–2746, Brighton, UK, 2009.

[62] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[63] Jane Morris and Graeme Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1):21–48, 1991.

[64] NIST. Topic Detection and Tracking Phase 2 (TDT2) Evaluation Plan. http://www.nist.gov/speech/tests/tdt/1998/doc/tdt2.eval.plan.98.v3.5.pdf, 1998.

[65] Alex Park and Timothy J. Hazen. ASR dependent techniques for speaker identification. In *International Conference on Spoken Language Processing (ICSLP)*, pages 1337–1340, Denver, Colorado, USA, September 2002.

[66] Lev Pevzner and Marti A. Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, 2002.

[67] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[68] Matthew Purver, Thomas L. Griffiths, Konrad P. Körding, and Joshua B. Tenenbaum. Unsupervised topic modelling for multi-party spoken discourse. In *International Conference on Computational Linguistics and meeting of the Association for Computational Linguistics*, pages 17–24, Sydney, Australia, 2006.

[69] David Pye. Content-based methods for the management of digital music. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2437–2440, Istanbul, Turkey, June 2000.

[70] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189, 1992.

[71] Jeffrey C. Reynar. Statistical models for topic segmentation. In *Meeting of the Association for Computational Linguistics*, pages 357–364, College Park, Maryland, USA, 1999.

[72] G. Riccardi, A. Gorin, A. Ljolje, and M. Riley. A spoken language system for automated call routing. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1143–1146, Munich, Germany, 1997.

[73] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[74] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 1999.

[75] Mark Steyvers and Tom Griffiths. Probabilistic topic models. In Thomas K. Landauer, Danielle S. McNamara, Simon Dennis, and Walter

Kintsch, editors, *Handbook of Latent Semantic Analysis*, pages 427–448. Routledge, 2007.

[76] L. Tarassenko, P. Hayton, N. Cerneaz, and M. Brady. Novelty detection for the identification of masses in mammograms. In *International Conference on Artificial Neural Networks (ICANN)*, pages 442–447, Paris, France, 1995.

[77] David M. J. Tax and Robert P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11-13):1191–1199, 1999.

[78] Peter D. Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *European Conference on Machine Learning (ECML)*, pages 491–502, Freiburg, Germany, 2001.

[79] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[80] Avery L. Wang. An industrial-strength audio search algorithm. In *International Conference on Music Information Retrieval (ISMIR)*, Washington, DC, USA, October 2003.

[81] Avery L. Wang. The Shazam music recognition service. *Communications of the ACM*, 49(8):44–48, August 2006.

[82] Eugene Weinstein and Pedro Moreno. Music identification with weighted finite-state transducers. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 689–692, Honolulu, Hawaii, USA, 2007.

[83] Jun Wu and Sanjeev Khudanpur. Building a topic-dependent maximum entropy model for very large corpora. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 777–780, Orlando, Florida, USA, 2002.

[84] J.P. Yamron, I. Carp, L. Gillick, S. Lowe, and P. van Mulbregt. Event tracking and text segmentation via hidden Markov models. In *Automatic Speech Recognition and Understanding (ASRU)*, pages 519–526, Santa Barbara, California, USA, 1997.

[85] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Conference on Research and Development in Information Retrieval (SIGIR)*, pages 28–36, Melbourne, Australia, 1998.