INDEXATION, RETRIEVAL & DECISION TECHNIQUES FOR SPOKEN TERM DETECTION

by

Doğan Can

B. S., Electrical and Electronics Engineering, Boğaziçi University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2010

ACKNOWLEDGEMENTS

This research was supported in part by TÜBİTAK Project No: 105E102 and TÜBİTAK BİDEB 2210.

Over the time I worked on this thesis, I was very fortunate to have the help and support of many wonderful people. I am grateful to all of the following individuals and to those who somehow contributed to this work but I forgot to mention here.

My advisor and mentor Murat Saraçlar, for giving me the privilege of being his student, for his invaluable guidance, ideas, never-ending enthusiasm, support and close interest in this work. I was very lucky to have such a wonderful person as my supervisor.

My professor and friend Yağmur Denizhan, for all the "Chaos" and all the "etc." evenings, for everything she taught me and especially for the mind-bending ideas and thought provoking discussions which one way or another redefined who I am.

My professor and mentor Kıvanç Mıhçak, for all the occasional chats and discussions on everything academic and non-academic and for everything I learned in his excellent lectures.

My colleagues at BÜSİM, Ebru Arısoy, Sinan Yıldırım, Haşim Sak, Temuçin Som, Erinç Dikici, Sıddıka Parlak Polatkan, Ekin Şahin, Özgür Dalkılıç, Oya Çeliktutan, Arman Savran and Hatice Çınar, for their friendship, support and sincere help.

My dear friend Çiğdem Aksu, for all the good times and the good laughs, especially for the never-ending chats on books, music, cinema and so on.

My *alter ego* Kamil Şenel, for the joy he brought into my life, for being there whenever I was in need and for putting up with me over the course of our friendship. Quoting from a very wise man: "I define friendship over our relation". Indeed...

My significant other Esin Demirci, for her endless love, patience and care, for her generous support in writing this thesis. I must be a very lucky man to have her in my life.

My sister Güneş Can, for being the greatest sister and listener of all times, especially for reminding me the dust I swept under the rug. My parents Hasan and Yüksel Can, for their endless love and support. This thesis is their achievement more than mine.

ABSTRACT

INDEXATION, RETRIEVAL & DECISION TECHNIQUES FOR SPOKEN TERM DETECTION

Speech Retrieval (SR) systems aim to provide access to large multimedia archives that include a vast amount of spoken media like lecture videos, podcasts, news clips and audio books. To that end, SR integrates two well studied fields: Automatic Speech Recognition (ASR) and Information Retrieval (IR). In an ideal setup where ASR transcripts are on a par with manual transcripts, SR is nothing more than classical text retrieval applied on ASR output. However, ASR technology is far from that point when it comes to heterogeneous stacks of unconstrained, unorganized audio recorded in uncontrolled environments. Considering the domain of interest to the end-user – think of databases like "YouTube" –, it becomes immediately obvious that relying entirely on ASR transcripts is a not an option for SR.

To minimize the effect of recognition errors, most SR systems are built upon ASR lattices where the oracle word error rates are much lower. In these systems, it is possible to retrieve overlapping hits for different queries since the index takes many alternative transcriptions into consideration for each spoken segment in the database. As a result, it becomes possible to retrieve matches that are omitted in the best hypotheses. However, this approach alone does not meet the open-vocabulary search objective held by most SR systems since after all we are limited to ASR vocabulary during retrieval. Utilizing sub-word (phone, graphone, morpheme) transcripts, or sub-word lattices for that matter, projects the word-level index/search/decide problem to a finer grained space where sub-word strings are now the object of search. In this sub-word universe, retrieval is partly freed from the chains of system vocabulary and we can retrieve out-of-vocabulary (OOV) query terms simply by searching the sub-word level ASR outputs.

Lattice indexing and sub-word methods improve recall but they also stress the ranking/decision process by matching segments irrelevant to the query. As the decision threshold is lowered to retrieve more, a large number of false alarms come into play as a combined effect of lattices and sub-words. For that matter, it is increasingly important to develop effective decision strategies which provide better discrimination between actual hits and false alarms.

Spoken Term Detection (STD) is a relatively new SR task which aims to locate exact matches to a given query term – a sequence of words in text form – in a large spoken database. In this thesis, we look for high-performing, low cost, efficient and reliable solutions to the various challenges of the STD task. Our methods include novel techniques for indexing ASR lattices, retrieving OOV words and ranking/thresholding candidate results in a general, efficient and mathematically sound retrieval framework.

ÖZET

KONUŞULAN TERİMLERİ SAPTAMAK İÇİN DİZİNLEME, GERİ GETİRİM VE KARAR TEKNİKLERİ

Konuşma geri getirim sistemleri, otomatik konuşma tanıma ve bilgi geri getirim teknolojilerini bir araya getirerek konuşma içeriği zengin çokluortam (ders videoları, haber klipleri, sesli kitaplar, vb.) arşivlerine erişim sağlamayı hedefler. Otomatik konuşma tanıma vasıtasıyla elde edilen metinler aslına sadık olduğunda, konuşma geri getirimi bu metinler üzerinde klasik geri getirim tekniklerinin uygulanmasından ibarettir. Ancak günümüzün otomatik konuşma tanıma teknolojisi iş dağarcık sınırı olmayan, düzensiz, karma ses kayıtlarını yazılandırmaya geldiğinde yüksek kalitede çıktı üretmekten oldukça uzaktır. Son kullanıcının ilgi alanları göz önüne alındığında - örneğin "YouTube" arşivi - sadece otomatik metinleri kullanarak makul düzeyde konuşma geri getirimi sağlamak pek de mümkün görünmemektedir.

Pek cok konuşma geri getirim sistemi, konuşma tanıma hatalarının etkisini azaltmak amacıyla tanıma örülerinden faydalanır. Dizinleme esnasında veritabanındaki her sözce için pek çok alternatif söz dizisi dikkate alınır. Bu sayede, en iyi hipotezlerin dışarıda bıraktığı kimi söz dizilerini geri getirmek mümkün olmaktadır. Ancak, tanıma örüleri de sistem dağarcığı ile sınırlı olduğundan, bu yöntem sisteme açık dağarcıklı sorgulama yapabilme özelliğini kazandırmaz. Ses, hece, morfem gibi kelime altı birimlerden oluşan tanıma örülerinin kullanımı, dizinleme, arama ve karar problemlerinin daha yüksek çözünürlükteki bir düzleme taşınmasını sağlar. Bu düzlemde, dizinleme ve sorgulama işlemleri kelime altı söz dizileri vasıtasıyla yapılır. Bu sayede, geri getirim kısmen de olsa sistem dağarcığının kısıtlarından kurtulur ve dağarcık dışı kelimeleri arayabilmenin yolu açılmış olur.

Örü dizinleme ve kelime altı yöntemler geri getirilen sonuç miktarını ciddi oranda

arttırırken, sezim sürecini önemli ölçüde zorlaştırmaktadır. Kabul eşiği daha fazla sonuç elde etmek adına düşürüldükçe, örülerin ve kelime altı birimlerin ortaklaşa sonucu olan büyük miktardaki hatalı sonuç eşiği geçerek son kullanıcıya ulaşır. Bu nedenle, örülerin ve kelime altı birimlerin kullanıldığı sistemlerde, doğru ve hatalı sonuçlar arasında iyi ayrım yapabilen yöntemler çok daha önemli hale gelir.

Konuşulan terimlerin saptanması büyük bir veritabanı içerisinde sorgu terimiyle birebir örtüşen kısımları bulmayı hedefler. Sorgu terimi, tıpkı arama motorlarında olduğu gibi metin formundaki bir sözcük dizisidir. Bu çalışmada, konuşulan terimlerin saptanması dahilinde karşımıza çıkan problemlere yüksek performanslı, düşük maliyetli ve güvenilir çözümler getirmeyi amaçlıyoruz. Geliştirdiğimiz yöntemler genel, verimli ve matematiksel açıdan ayakları yere basan bir geri getirim platformu dahilinde otomatik konuşma tanıma örülerinin dizinlenmesi, dağarcık dışı kelimelerin geri getirimi ve aday sonuçların eşiklenmesi için çözümler sunmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

$[\![A]\!](x)$	The weight associated by acceptor A to the input string x
$\mathbf{A}$	Acoustic feature vector sequence
A	Weighted finite-state acceptor
$A(Q_k)$	Number of retrieved segments for the query term Q_k
B	Weighted finite-state transducer obtained after preprocessing
$\hat{C}_{\mathrm{FA}}(Q_k)$	Expected cost of a false alarm for the query term Q_k
$C(Q_k)$	Number of correctly retrieved segments for the query term Q_k
C_{ij}	Cost of choosing hypothesis H_i when H_j is true
$c_{k,n},\ n = 1, \ldots, N_k$	Candidate results for the k^{th} query term
$C_{x,y}(u_i)$	Number of occurrences of a factor-pair (x, y) in utterance u_i
$C_x(u_i)$	Number of occurrences of a factor x in utterance u_i
d	Acoustic feature dimension
$E_{x,y}(u_i)$	End time of factor-pair (x, y) in utterance u_i
E	Set of arcs
F	Set of final states
$F(A)$	Factor automaton of an acceptor A
$F(u)$	Factor automaton of a string u
H_0	Hypothesis 0 (false alarm)
H_1	Hypothesis 1 (hit)
I	Set of initial states
$\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$	Product semiring obtained from $\mathcal{L}$, $\mathcal{T}$ and $\mathcal{T}'$
$\mathcal{L}$	Log semiring
L	ASR lattice
$L(y)$	Likelihood ratio
$L_i(q)$	Timing of state q in B_i
M	Weighted finite-state automaton
m	ASR unit sequence length
N_Q	Number of query terms
$\mathcal{P}^*$	Phonetic language

p	Exponential mixture model density
P_0	False alarm score distribution
P_1	Hit score distribution
$p_j,\ j = 0, 1$	Density of distribution P_j
$P_n(Q_l)$	n-best L2S pronunciations for the query Q_l
P_{FA}	False Alarm probability
P_{Miss}	Miss probability
Q	Set of states
$Q_k,\ k = 1, \ldots, N_Q$	k^{th} query term
Q_l	Query as a string of letters
Q_p	Query as a string of phones
$\hat{R}(Q_k)$	Expected count of the query term Q_k in the database
$\mathcal{R}$	Real semiring
R	Reference transcription
$r(\delta)$	Bayes risk incurred by decision rule δ
$R(Q_k)$	Number of segments related to the query term Q_k
$R_j(\delta)$	Conditional risk for H_j incurred by decision rule δ
$\mathcal{S}$	SR vocabulary
$\mathcal{S}^*$	SR language
$S_{x,y}(u_i)$	Start time of factor-pair (x, y) in utterance u_i
$[\![T]\!](x, y)$	The weight associated by transducer T to the input-output string pair (x, y)
$\mathcal{T} * \mathcal{T} * \mathcal{T}$	Lexicographic semiring obtained from three tropical semirings
$\mathcal{T}$	Tropical semiring with min convention
$\mathcal{T}'$	Tropical semiring with max convention
T	Weighted finite-state transducer
t	Acoustic sequence length
t_s, t_e	Start time, end time
T_{Speech}	Total amount of speech in the database
u	Utterance
$\hat{V}$	Most probable ASR hypothesis
$\hat{V}_{\text{Hit}}(Q_k)$	Expected value of a hit for the query term Q_k

$\mathcal{V}$	ASR vocabulary
$\mathcal{V}^*$	ASR language
x	Input symbol sequence
$Y \sim P$	Y has distribution P
Y	Random variable representing a candidate score
$y_{k,n},\ n = 1, \ldots, N_k$	Candidate scores for the k^{th} query term
α	Relative cost of false alarms with respect to hits
β	TWV parameter
Δ	Output alphabet
δ	Decision rule
γ	L2S score scaling parameter
λ	Initial weight function
$\lambda_j,\ j = 0, 1$	Exponential distribution parameter for P_j
Π	A set of paths in a finite-state automaton
π	A path in a finite-state automaton
π_0	Prior probability of H_0 (having a false alarm)
π_1	Prior probability of H_1 (having a hit)
ρ	Final weight function
Σ	Input alphabet
$\hat{\theta}(Q_k)$	Term specific detection threshold for the query term Q_k
θ	Detection threshold
ASR	Automatic Speech Recognition
ATWV	Actual Term Weighted Value
BUTBN	Boğaziçi University Turkish Broadcast News Database
CN	Confusion Network
DET	Detection Error Tradeoff
EM	Expectation-Maximization
EMM	Exponential Mixture Model
GT	Global Thresholding
IR	Information Retrieval

IV	In-Vocabulary
L2S	Letter-to-Sound
LVCSR	Large Vocabulary Continuous Speech Recogntion
MDL	Minimum Description Length
MSTD	Multilingual Spoken Term Detection
MTWV	Maximum Term Weighted Value
OOV	Out-Of-Vocabulary
PR	Precision-Recall
ROC	Receiver Operating Characteristics
SD-TST	Score Distribution Based Term Specific Thresholding
SDR	Spoken Document Retrieval
SR	Speech Retrieval
STD	Spoken Term Detection
SUR	Spoken Utterance Retrieval
TST	Term Specific Thresholding
TWV	Term Weighted Value
TWV-TST	Term Weighted Value Based Term Specific Thresholding
VSM	Vector Space Model
WER	Word Error Rate
WFSA	Weighted Finite-State Acceptor
WFST	Weighted Finite-State Transducer

1. INTRODUCTION

The ever-increasing availability of vast multimedia archives calls for solutions to efficiently index and search this data. To this end, speech retrieval (SR) is a key technology which integrates automatic speech recognition (ASR) and information retrieval (IR) to provide large scale access to spoken content. In an ideal setup, the ASR component converts speech to text and text retrieval methods are applied on the recognition output. Unfortunately, state-of-the-art ASR systems are far from being reliable when it comes to transcribing unconstrained speech audio recorded in uncontrolled environments. Considering the heterogeneous nature of large spoken databases, it is no surprise SR research is centered around retrieval schemes which compensate ASR deficiencies.

Most SR applications are designed to imitate real-world problems where the end-user would like to perform open-vocabulary search over a large collection of spoken documents in a matter of seconds. Therefore, in these applications, speech corpora must be indexed prior to search without the advance knowledge of query terms. This is a challenging task. In text retrieval, the corpus is unambiguous in the sense that whether a particular word occupies a particular position in a document is known. In SR, on the other hand, the corpus to be indexed is the output of the ASR component, thus it is inherently ambiguous. Unlike text indexers, speech indexers have to deal with the fact that any query word may occur anywhere in a given corpus of spoken documents.

Lattices are graph structures that store weighted hypotheses in a compact form. Indexing ASR lattices, instead of the best ASR hypothesis, is a widely used SR method which partly compensates for the negative effects of recognition errors. In this method, spoken documents are divided into short segments called utterances. For each utterance an ASR lattice, representing ASR hypotheses, is generated. Then a probability is assigned to whether a word occupies a particular position in an utterance given the ASR lattice produced for that segment. The indexer uses these probabilities to create

soft (probabilistic) entries that are stored in an inverted index. During search, soft entries allow multiple words to occupy the same position and provide significant gains in the retrieval of in-vocabulary (IV) query words (words from the ASR vocabulary).

Since ASR lattices carry a large amount of connectivity information, most researchers argue them to be redundant for SR applications. A common SR approach is to use simpler structures, like confusion networks and position specific posterior lattices, that approximate raw lattices at the cost of decreased detection accuracy. These structures project the complex network of connections in the original lattice to a strictly linear representation that naturally lends itself to off-the-shelf text indexers.

Exact inversion of ASR lattices into an index is a understudied SR approach even though it promises higher detection accuracy since the index stores the exact connectivity information found in lattices. Most SR researchers favor approximate representations since inverted indexes derived from them are much smaller, and of course since they play well with the existing text retrieval technology. In tasks like spoken document retrieval (SDR), where both the query and the documents contain a large amount of redundancy for retrieval, it is hard to argue against the value of approximate structures.

In other tasks, however, keeping the connectivity information may play a crucial role. Spoken utterance retrieval and spoken term detection are two such tasks that aim to find respectively the utterances and the time intervals in those utterances which contain the exact sequence of words given as the query. In these tasks, approximate approaches carry the potential risk of degrading detection performance since they discard actual connectivity information and replace it with connections that potentially do not exist in the original lattices.

Open vocabulary search is a demanding criterion that complicates the SR problem altogether. Since the query terms may well be beyond the coverage of ASR vocabulary, word level methods are usually of no remedy. Sub-word (phone, morpheme, etc.) indexing, another well-studied SR method, tackles the recognition errors at the sub-

word level. In this approach, the indexer uses sub-word level transcriptions to create sub-word level entries that are then compiled into an inverted index. The retrieval is performed by converting the query to its sub-word representation and searching the index for matching sequences. The major offering of this method is a means for the retrieval of out-of-vocabulary (OOV) query words which do not occur is ASR hypotheses. In most SR systems, sub-word indexing is generally combined with lattice indexing to get even better results.

While sub-word indexing provides a method to retrieve OOV query words given a sub-word level representation for the query, it does not answer how to obtain these representations. Most SR systems tackle this problem at the retrieval stage by employing pronunciation models. However, it is not an easy task to generate proper pronunciations for words that haven't been seen before. For that reason, most SR systems utilize inexact matching methods to handle the discrepancies between the actual and hypothesized pronunciations.

As with any IR application, keeping a balance between recall and precision is essential in SR tasks. All above mentioned methods have the common goal of increasing recall rates in SR applications. For that purpose, they relax the conditions under which a trial is accepted as a match to the query. In this relaxed setup, higher recall rates usually translate into lower precision figures which call for effective methods to discriminate actual hits from false positives. On that account, ranking retrieval results with respect to a relevance criterion and filtering out potential false positives constitutes an indispensable ingredient of any SR system.

This work concentrates on a specific SR application known as spoken term detection (STD). STD is similar to the spoken utterance retrieval (SUR) task in that it aims to locate exact matches to a given query term – a sequence of words in text form. Unlike SUR, which aims to locate the utterances containing the query, STD task looks for exact locations in these utterances.

In the following chapters, we present novel indexation, retrieval and detection

methods for the STD task. Chapter 2 gives an overview of related ASR and IR concepts, and introduces the STD task. Then in Chapter 3, we detail the corpora and ASR systems used in this work. The main contributions of this work are presented in Chapters 4, 5 and 6. In Chapter 4, we develop a general lattice indexation/search framework for SR applications with STD task in mind. We show that proposed architecture is a search-time optimal solution to the problem of inverting ASR lattices for the STD task. Then in Chapter 5, we consider the problem of retrieving out-of-vocabulary STD queries without reliable pronunciations at hand. We develop a new retrieval strategy which incorporates weighted pronunciation hypotheses. Chapter 6 considers the thresholding problem in STD and develops a query adaptive detection technique which exploits relevance score distributions to estimate optimal term specific thresholds. Finally in Chapter 7, we give our conclusions and discuss future directions.

The main contributions of this thesis are:

(i) the development of an efficient, versatile and mathematically sound STD framework,
(ii) a query expansion technique which leverages the versatile search mechanism of our STD framework to retrieve out-of-vocabulary query terms,
(iii) a detection technique to classify the putative term occurrences retrieved from the index considering the occurrence score statistics.

2. BACKGROUND

Speech Retrieval (SR) systems aim to provide access to large multimedia archives that include a great amount of spoken media like lecture videos, podcasts, news clips and audio books. To that end, SR integrates two well studied fields: ASR and IR. In an ideal setup where ASR transcripts are on a par with manual transcripts, SR is nothing more than classical text retrieval applied on ASR output. However, ASR technology is far from that point when it comes to heterogeneous stacks of unconstrained, unorganized audio recorded in uncontrolled environments. Even moderately constrained recordings can easily give word error rates in the 30-50 per cent range with state-of-the-art LVCSR engines. Considering the domain of interest to the end-user – think of databases like "YouTube" –, it becomes immediately obvious that relying entirely on ASR transcripts is a not an option for SR.

To minimize the effect of recognition errors, most SR systems are built upon ASR lattices where the oracle word error rates are much lower. In these systems, it is possible to retrieve overlapping hits for different queries since the index takes many alternative transcriptions into consideration for each spoken segment in the database. As a result, it becomes possible to retrieve matches that are omitted in the transcripts of best hypotheses. However, this approach alone does not meet the open-vocabulary search objective held by most SR systems since after all we are limited to ASR vocabulary during retrieval. Utilizing sub-word (phone, graphone, morpheme) transcripts, or sub-word lattices for that matter, projects the word-level index/search/decide problem to a finer grained plane where sub-word strings are now the object of search. In this sub-word universe, retrieval is partly freed from the chains of ASR vocabulary and we have a means of retrieving OOV query terms by searching the sub-word strings in the ASR outputs.

Lattice indexing and sub-word methods improve recall but they also stress the ranking/decision process by matching segments irrelevant to the query. As the decision threshold is lowered to retrieve more, a large number of false alarms come into play as a

combined effect of lattices and sub-words. For that matter, it is increasingly important to develop effective detection strategies which provide better discrimination between actual hits and false alarms.

In this thesis, we look for high-performing, low cost, efficient and reliable methods to develop a viable solution to the SR problem known as spoken term detection (STD). Our methods include techniques for indexing ASR lattices, retrieving OOV words and ranking/thresholding candidate results in a general, extendible, efficient framework.

In this chapter, we review relevant concepts from ASR and IR fields, and present a survey on how these two fields are brought together by SR researchers. Then we introduce the STD task along with the evaluation methods used in assessment of STD system performance.

2.1. Automatic Speech Recognition

Automatic speech recognition aims to map acoustic speech signals to sequences of words. Current ASR systems are able to handle many constrained tasks like voice dialing, medical dictation and simple human-machine interaction applications such as voiced control interfaces employed smart home projects. When the tasks get more involved by relaxing vocabulary, speaker and environment constraints, the reliability of ASR outputs degrades in accordance.

Speech recognition problem can be discussed in many dimensions such as vocabulary size, continuity and speaker dependency. Large vocabulary continuous speech recognition (LVCSR) is a major field, where *large vocabulary* implies a vocabulary larger than 5000 words and *continuous* stands for no pause between words (unlike isolated word recognition, where each word is followed by a pause). Speaker dependent systems are optimized to recognize speech from a particular speaker whereas speaker independent ones are aimed at recognizing speech from any speaker. Similarly gender- and age-dependent models improve performance by limiting acoustic variability.

In the following, we give a mathematical description of statistical speech recognition approach, introduce ASR concepts relevant to SR, and outline common ASR techniques to improve SR performance.

2.1.1. Statistical Formulation of Automatic Speech Recognition

Statistical ASR systems aim to find the most likely unit (word or sub-word) sequence given an acoustic input signal (See [1, 2] for an introduction to ASR). The acoustic signal is processed by the ASR front-end and converted into a sequence of acoustic feature vectors $\mathbf{A}$. Given $\mathbf{A}$ and the ASR vocabulary $\mathcal{V}$, the ASR system looks for the unit sequence $\hat{V}$ among all unit sequences $\mathcal{V}^*$, satisfying

$$\hat{V} = \underset{V \in \mathcal{V}^*}{\operatorname{argmax}} P(V|\mathbf{A}) = \underset{V \in \mathcal{V}^*}{\operatorname{argmax}} P(\mathbf{A}|V)P(V)$$

where $P(V)$ and $P(V|\mathbf{A})$ are respectively the prior and the posterior probabilities (given acoustics $\mathbf{A}$) of the unit sequence V, and $P(\mathbf{A}|V)$ is the likelihood of observing the acoustic sequence $\mathbf{A}$ when the underlying unit sequence is V. These probabilities are computed using parametric models whose parameters are estimated from text and manually transcribed audio corpora. Actual recognition is performed by a decoder which conducts search over the space defined by the parametric models.

It should be noted that both $\mathbf{A}$ and V are variable length sequences:

$$\begin{aligned} \mathbf{A} &= [\mathbf{a}_1, \ldots, \mathbf{a}_t] \quad \in \mathbb{R}^{d \times t}, \\ V &= [v_1, \ldots, v_m] \quad \in \mathcal{V}^*, \end{aligned}$$

where $t,\ d,\ m \in \mathbb{N}_+$ are respectively acoustic sequence length, acoustic feature dimension and unit sequence length. Therefore, ASR models should account for any pair of sequences $(\mathbf{A}, V)$. Furthermore, the decoder should perform the search over all possible unit sequences.

The major components of an ASR system can be summarized as follows:

- *Front-end* converts acoustic signal into $\mathbf{A}$.
- *Acoustic model* gives an estimate of $P(\mathbf{A}|V)$.
- *Language model* gives an estimate of $P(V)$.
- *Decoder* finds $\hat{V}$ given the models and the acoustics.

This thesis is about post-ASR STD methods which use ASR outputs but do not specifically try to improve ASR performance. We are not concerned with how the front-end signal processing is performed, acoustic and language models are trained, and ASR outputs are produced. Therefore, in this section we will only talk about ASR lattices, sub-word language models and other relevant concepts which are utilized in our STD setup.

2.1.2. ASR Lattices

An ASR decoder searches through the network of all possible word sequences determined by the language model and looks for the paths that best match the acoustics given the acoustic model. An ASR transcript is the single best path in this network for the acoustic sequence at hand. It is possible to gather a collection of best performing paths for an acoustic sequence and represent them in compact form known as a lattice. Simply put, an ASR lattice represents weighted recognition hypotheses (including the best one) for an acoustic sequence. Figure 2.1 gives an example ASR lattice represented by a Weighted Finite-State Acceptor (WFSA). In this example lattice, arcs carry (word label / score) pairs, and each utterance hypothesis is a path from the initial state "0" to the final state "4". A (hypothesis / score) pair is obtained by concatenating the arc labels, and multiplying the arc scores and the final score on a path. Best hypothesis pair is (iyi gUnler / $0.730 \times 0.975 \times 1$).

Compared to the single best path, a lattice better represents the acoustic sequence since it basically is an approximate probability distribution over likely strings. As an example, let's assume that we have two utterances u_1, u_2, and that we obtained two acoustic sequences $\mathbf{A}_1$, $\mathbf{A}_2$, two lattices L_1, L_2, and two transcripts (best paths) $\hat{V}_1$, $\hat{V}_2$ for these utterances using an ASR system. For the moment, assume that we do not

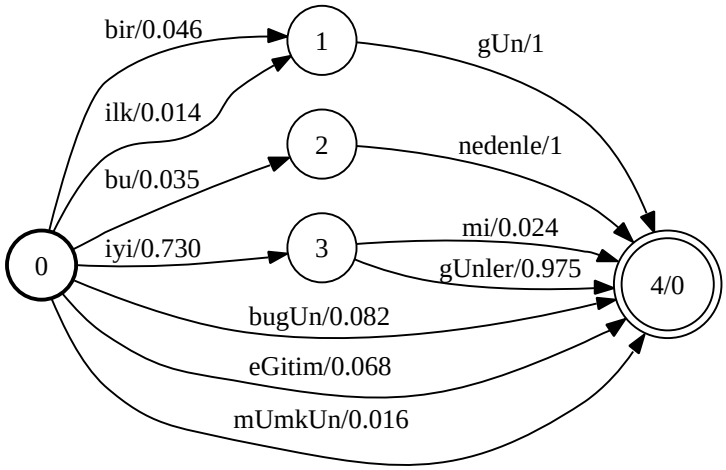


Figure 2.1. An example lattice

have reference transcripts at hand. If we would like to measure how similar the two utterances are – both acoustic and language similarity –, we have many options like checking

- the similarity of actual acoustic sequences $\mathbf{A}_1$ and $\mathbf{A}_2$, i.e. dynamic time warping based methods,
- the similarity between $\hat{V}_1$ and $\hat{V}_2$, i.e. minimum edit distance,
- or the similarity between L_1 and L_2, i.e. minimum edit distance in expected sense or Kullback-Leibler distance.

Among these, lattice methods are the most robust since they account for the acoustic and language variability between the two utterances. If we were to have the reference transcripts R_1, R_2 as well, we could have done other things like measuring the similarities between reference transcripts and utterances. Actually, this is the very problem we face in speech retrieval. Given a query term, go find the utterances most similar to it. Similar to above discussion, we can check

- the similarity between R_j and $\hat{V}_j$, $j = 1, 2$,
- or the similarity between R_j and L_j, $j = 1, 2$.

Since lattices are better representatives than single best paths, in the second strategy

we have a higher chance of finding a match to the queries, reference transcripts in this case.

2.1.3. Confusion Networks

Confusion networks (CNs) or sausages [3] are approximate lattice representations with a strictly linear structure. They are strings of confusion sets where each set comprises of aligned word (or sub-word) hypotheses. They are usually derived from lattices by clustering arcs with similar timing to form a confusion set. We should note that arcs in confusion networks usually carry posterior probabilities derived from confusion sets whereas lattice arcs usually carry likelihood scores. Figure 2.2 gives an example confusion network derived from the lattice in Figure 2.1.

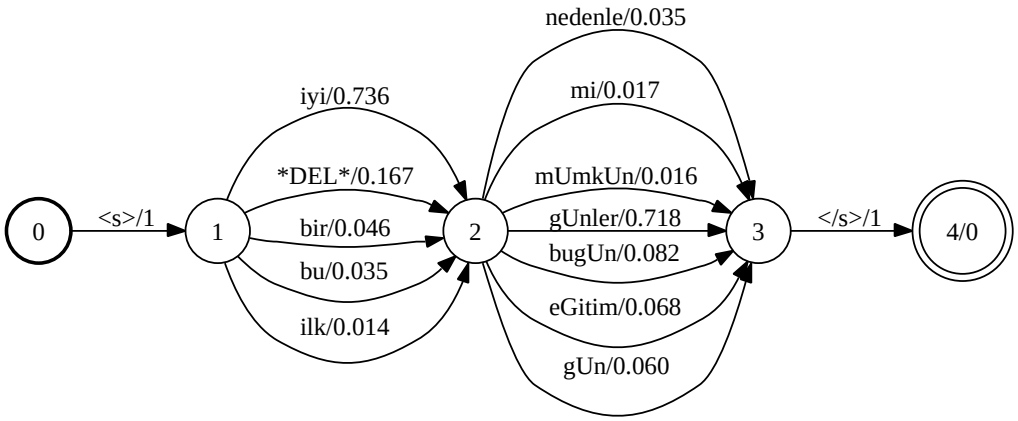


Figure 2.2. An example confusion network

In this example confusion network, utterance hypotheses are given by the paths from the initial state to the final state. Every such path includes only one arc from each confusion set. Best (hypothesis / score) pair is found by selecting the best performing arc in each set, in this case (iyi gUnler / $0.736 \times 0.718 \times 1$). Note that the best hypothesis is the same as the one obtained from the original lattice, however we have a different score this time. Not only that, but also the best hypothesis could have been different from what is obtained from the lattice. Best path in a confusion network is the one which minimizes the word error rate whereas best path in a lattice is the one that minimizes the sentence error rate.

One further discrepancy is that confusion networks are not equivalent to original lattices in the sense that they have extra paths that are not there in the original lattices and lack some of the paths that are in the original lattices. For instance, the original lattice in Figure 2.1 has no counterpart of the confusion network path (bir eGitim / 0.046×0.068) in Figure 2.2. This fact may be an advantage or a disadvantage depending on the application, i.e. lower word error rate in ASR, higher recall and false alarm rates in SR.

2.1.4. Automatic Speech Recognition Evaluation

The standard evaluation metric for speech recognition accuracy is called Word Error Rate (WER). It is computed using the formula:

$$\text{WER} = 100 \cdot \frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Number of words in reference}} \tag{2.1}$$

where the nominator represents the word-level minimum edit distance between the hypothesis and reference transcriptions. Oracle WER is defined for lattices and it stands for the WER of the hypothesis closest to the reference in the minimum edit distance sense.

2.1.5. Sub-word Language Modeling for Speech Retrieval

Let $\mathcal{S}$ and $\mathcal{S}^*$ denote the vocabulary and the target language of an SR system respectively. Note that $\mathcal{S}^*$ is not necessarily limited to the coverage of the ASR language $\mathcal{V}^*$ and in general ASR language (the set of phrases that can be output by ASR) is a subset of the SR language (the set of queries). Since ASR systems can not recognize $S \in \mathcal{S}^* \setminus \mathcal{V}^*$, the unit sequences that lie outside the ASR language $\mathcal{V}^*$, these sequences can not be retrieved without transducing the ASR outputs and the query terms to another language $\mathcal{P}^*$, i.e. phonetic sequences, where the transduced sequences can be matched. Word based ASR systems typically necessitate this approach since many query words tend to be out of ASR vocabulary. Even after transduction, certain query terms may still lie outside the coverage of ASR language.

Turkish SR is a very good example of the above situation since the set of possible queries comprise a much larger set compared to any word based language defined by an LVCSR vocabulary. Of course the main implication of this fact is that Turkish ASR will be poor when the recognition units are words since it is possible to obtain an unlimited set of valid Turkish words by adding consecutive suffixes to a single root. An example word is shown below:

```
el + im + de + ki + ler + den :  elimdekilerden
```

This word comprises of a root and 5 suffixes, and means "(one) of those in my hand".

In this context, sub-word based ASR promises better SR performance by availing the possibility of covering more query terms with or without further transduction. For agglutinative languages, like Turkish and Korean, sub-word language models are the primary practice in ASR whether or not SR is in the picture. In other languages, like English, where medium sized ASR vocabularies can cover most of the language, sub-word language models are still preferable in SR applications since OOV words, like named entities and foreign words, comprise a significant portion of the SR queries.

Some sub-word units commonly utilized in ASR are listed below:

(i) Linguistic Units
- *Phones* are the smallest identifiable units found in a stream of speech.
- *Syllables* are sound units composed of a vowel (at the center) and consonants around it.
- *Morphemes* are grammatical sub-units (stem, prefix and suffix) of a word

(ii) Non-linguistic Units
- *Morphs* are letter groups. They are typically derived from large text corpora by specifying a vocabulary size and employing the Minimum Description Length (MDL) principle.
- *Particles* are syllable-like data-driven units determined by maximizing the

leaving-one-out likelihood of a particle based bigram language model [4].

Most SR systems utilize morphs and particles since they are data-driven, language independent forms. Phonetic decoding, on the other hand, constitutes another well studied SR practice in which optimizing the retrieval performance is aimed directly at the ASR stage.

2.2. Information Retrieval

Information Retrieval (IR) aims to provide access to documents relevant to a user query. Traditionally, information retrieval is performed on text data. Recent increase in the amount of online multimedia documents initiates research for efficient indexing and search of these data as well. In this section we review text retrieval concepts and methods.

The main component of any IR system is an inverted index. Index is a compact structure which stores all the relevant information about a collection of documents. We may think of documents as a mapping from document IDs to the words enclosed by each of them. Logically index provides an inverse mapping between words and documents. Consider the "Google" index or the index of a textbook as an example. Index entries provide the location information of a selected subset of strings contained in the database. In the case of a search engine the database is that of web pages and in the case of a textbook it is the book itself. Looking at the textbook index or querying the search engine, we can learn the pages or web pages that includes the term we are looking for. This frees us from going all over the book or a directory of web pages to find relevant materials. Since the search is done over the index, instead of the document collection, search time is less dependent on the size of the collection.

Index construction method depends on the retrieval model we use. In the case of a Boolean Model, search engine returns documents that match the query term exactly, hence the index needs to store all connectivity information present in the documents, i.e. documents are treated as strings of words. On the other hand, with the well known

Vector Space Model (VSM), search engine returns documents relevant to a query term even if there is only a partial match to the query. Most text search engines follow a vector space model to find documents similar to the query and rank their results according to an estimated relevance score [5, 6]. Under the VSM model, the index does not need to store all of the connectivity information present in the documents, i.e. documents are treated as bags of words. Statistical Language Modeling is another retrieval model with partial matching. In this case, a language model is built for each document in the collection and the document relevance score and ranking is given by the probability of producing the query at hand using the language model in question [7]. *Ad hoc retrieval* is a common type of text retrieval. In this scheme, indexed documents remain relatively unchanged while the users submit various queries to the system.

Apart from retrieval models, many additional techniques are used for increasing the IR efficiency. *Query expansion* is the technique of augmenting a query with words or phrases similar to the query in meaning or lexical form, i.e. synonyms, morphological variants. *Document expansion* is more or less doing the same thing as query expansion on the documents to be indexed. Query and document expansion techniques are widely used to enrich the query and document content so that a previously unnoticed relevance can be inferred. These techniques aim to reduce the mismatch between the queries and the documents in the collection.

Stemming is another widely used method which relies on the fact that relevance information is largely contained in word stems. Stemming is even more important in the case of inflectional languages since a common stem may generate a plethora of inflectional word forms most of which are related to the same concept. Indexing stems and searching the stems of query words on this index usually gives a significant increase in the quality and quantity of search results. Indexing the first few characters of each word is a very simple stemming technique which was shown to provide significant improvement in retrieval results [8].

Reducing the dimensionality of the VSM term document matrix leads to a grouping of related terms and is a good method of reducing the query-document mismatch.

Latent semantic indexing is a popular dimensionality reduction technique which uses singular value decomposition to find best low rank approximation to the original term document matrix in the Frobenius norm sense. This reduction projects the terms to another space where semantically similar terms are clustered. It is possible to use other dimensionality reduction methods, such as nonnegative matrix factorization, to achieve the same goal [9].

Using stop lists is another widely used technique which excludes common words from indexing and retrieval. Since these words usually lack discriminating information, their exclusion both reduces the index size [10] and improves the ranking of results. Index of a textbook is a classical example where common words are removed to save space and ease visual search without disrupting the index quality.

2.2.1. Evaluation of IR Systems

2.2.1.1. Precision-Recall (PR) Curves. Precision and recall are the most popular IR evaluation metrics. Given a set of queries $Q_k,\ k = 1, \ldots, N_Q$, let

$R(Q_k)$ be the number of segments in the collection that are related to the query Q_k,
$A(Q_k)$ be the total number of retrieved segments and
$C(Q_k)$ be the number of correctly retrieved segments.

Then,

$$\text{Precision} = \frac{1}{N_Q}\sum_{i=1}^{N_Q}\frac{C(Q_k)}{A(Q_k)}, \qquad \text{Recall} = \frac{1}{N_Q}\sum_{k=1}^{N_Q}\frac{C(Q_k)}{R(Q_k)}. \tag{2.2}$$

There is always a trade-off between precision and recall figures of an IR system. Recall is usually low at the high precision region (best scoring segments) since many relevant but low scored segments might be missed. Likewise, precision is low at the high recall region since many irrelevant segments are included. Typically, the relation between precision and recall of an IR system is analyzed over precision-recall curves

which depict this trade-off.

2.2.1.2. Term Weighted Value. A novel metric to evaluate STD systems is the "Term Weighted Value (TWV)" which is defined in NIST STD 2006 Evaluation Plan [11] as:

$$\mathrm{TWV}(\theta) = 1 - \frac{1}{N_Q}\sum_{k=1}^{N_Q}\{P_{\mathrm{Miss}}(Q_k,\theta) + \beta.P_{\mathrm{FA}}(Q_k,\theta)\} \tag{2.3}$$

where θ is the detection threshold and β is a user defined parameter to adjust the trade-off between miss and false alarm costs. P_{Miss} and P_{FA} are calculated as:

$$\mathrm{P_{Miss}}(Q_k,\theta) = 1 - \frac{C(Q_k,\theta)}{R(Q_k)}, \qquad \mathrm{P_{FA}(Q_k},\theta) = \frac{A(Q_k,\theta) - C(Q_k,\theta)}{T_{\mathrm{Speech}} - C(Q_k,\theta)} \tag{2.4}$$

where T_{Speech} is the total amount of speech. TWV is calculated for all possible θ values and the maximum is called the Maximum Term Weighted Value (MTWV). When the system utilizes term specific thresholds, Actual Term Weighted Value (ATWV) attained by the system is calculated as:

$$\mathrm{ATWV} = 1 - \frac{1}{N_Q}\sum_{k=1}^{N_Q}\{P_{\mathrm{Miss}}(Q_k,\theta(Q_k)) + \beta.P_{\mathrm{FA}}(Q_k,\theta(Q_k))\}. \tag{2.5}$$

2.3. Integration of ASR and IR: Speech Retrieval

Text information retrieval is a well studied problem. It is the primary concern of current web search technology. Speech Retrieval (SR), on the other hand, is an emerging technology that has come into the spotlight with the increase of multimedia documents available on the world wide web. Spoken Document Retrieval, Spoken Term Detection and Spoken Utterance Retrieval are different SR applications which all aim to provide access to large spoken archives.

2.3.1. Spoken Document Retrieval

Spoken Document Retrieval (SDR) is the content based retrieval of spoken audio, which brings the speech recognition and information retrieval research together to access spoken information. It is the speech counterpart of traditional text retrieval. The main purpose and challenges of the SDR task differ from STD. For SDR, content of the document is more important than exact term matching. Unlike STD, SDR is less affected by OOV words. In SDR, it may be possible to find a relevant document, even if the query is OOV, using the other query words or clustering. However this is not possible in STD, unless additional methods are used.

Several research groups developed spoken document retrieval systems for leveraging access to voicemail messages, broadcast news, meetings, lecture recordings, etc. A prime example is BBN Technologies' "Rough'n'Ready" system which facilitates the retrieval of broadcast news. It is an Hidden Markov Model based IR system which incorporates several technologies such as LVCSR, speaker segmentation, name spotting, topic segmentation and classification [12]. It does not utilize video cues for retrieval; it is merely based on speech modality. "SpeechFind" [13] is a publicly available SDR system facilitating search over the National Gallery of the Spoken Word [14]. It makes use of the standard vector space model incorporating the metadata as well as automatic transcripts into the retrieval.

2.3.2. Spoken Term Detection

Spoken term detection (STD) is a key information retrieval technology which aims open vocabulary search over large collections of spoken documents. In the STD 2006 Evaluation Plan [11], NIST defines the STD task as finding all of the occurrences of each given *term* – a sequence of words consecutively spoken – in a large collection of spoken segments. Unlike SDR, it aims exact term matching.

From the IR perspective, STD is an ad-hoc retrieval application: the user submits a query and the search engine lists the results. Each result includes a segment ID, a

time interval (start-end time pair), and a decision (relevance) score (typically posterior probability). STD is intended for monitoring and detection applications where the topic or context of a result are not major concerns. IR techniques like stemming, clustering or stop-word elimination usually conflict with the exact match objective of STD. It is possible to evaluate STD systems with IR evaluation metrics though.

The NIST STD 2006 Evaluation initiated the STD track in three languages: Arabic, English and Mandarin. The participating systems were evaluated based on not only detection performance but also indexing and search time, index size and memory consumption. The SRI/OGI system achieved one of the best scores using a word+graphone system [15]. BBN system achieved the maximum accuracy in continuous telephone speech domain using an index built from word lattices and approximate phonetic transcripts [16].

The major challenge faced by STD is the lack of reliable transcriptions, an issue that becomes even more pronounced with heterogeneous, multilingual archives. The classical STD approach consists of converting the speech to word transcripts using LVCSR tools and extending classical IR techniques to word transcripts. However, recognition results are erroneous, many reference words are replaced in the output transcripts by alternatives that are probable, given the acoustic and language models of the ASR system.

Using alternative ASR hypotheses, in addition to the best one, is a useful approach to deal with the recognition errors. These alternative hypotheses can be in the form of lattices or approximate structures like confusion networks [3] and position specific posterior lattices [17]. In this context, lattice indexing [18–21] provides a means of reducing the effect of recognition errors by incorporating alternative transcriptions in a probabilistic framework. It is reported in [22] that, confusion networks perform better than lattices, in addition to their advantage of having smaller index sizes. A robustness analysis of lattice indexing methods is given in [23].

A significant drawback of word-based indexing methods is that search on queries

containing out-of-vocabulary (OOV) terms will not return any result. Considering the fact that many STD queries consist of rare named entities and foreign words, retrieval performance is highly dependent on the coverage of ASR vocabulary. It has been experimentally observed that over 10 per cent of user queries can contain OOV terms [24], as queries often relate to named entities that typically have a poor coverage in the ASR vocabulary. Developers of Speechbot [25] report that the percentage of OOV queries submitted to a real engine is about 13 per cent [26]. In many applications, the OOV rate may get worse over time unless the recognizer's vocabulary is periodically updated.

An approach for solving the OOV issue consists of converting the speech to phonetic transcripts and representing the query as a sequence of phones. Such transcripts can be generated by expanding the word transcripts into phones using the pronunciation dictionary of the ASR system. Another way is to use sub-word (phone, syllable, or word-fragment) based language models. The retrieval is based on searching the sequence of sub-words representing the query in the sub-word transcripts. During 1990s NIST TREC Spoken Document Retrieval tracks fostered speech retrieval research as described in [27]. Popular approaches are: search on sub-word decoding [18, 28, 29] or search on the sub-word representation of word decoding enhanced with phone confusion probabilities and approximate similarity measures for search [30].

OOV issue was also tackled by the IR technique of query expansion. In classical text IR, query expansion is based on expanding the query by adding additional words using techniques like relevance feedback, finding synonyms of query terms, finding all of the various morphological forms of the query terms and fixing spelling errors. Phonetic query expansion has been used for Chinese spoken document retrieval on syllable-based transcripts using syllable-syllable confusions from the ASR [31].

To emphasize the effect of OOV queries, OOV percentage is artificially increased to 50 per cent in [26]. Word-based, phoneme-based and particle-based systems are compared. Hybrids of word and sub-word systems are shown to demonstrate the best performance. The effects of query expansion on a hybrid system was evaluated in

[32]. An OOV query is expanded to in-vocabulary (IV) queries based on acoustic confusability and language model scores. This method provides an improvement of one per cent in average precision. Several other studies show the superiority of hybrid systems. In [19], phone and word cascades are shown to be useful. In [29] the word and phone indexes are directly combined using time stamps and hybrid queries are searched on the combined index.

In [33], a two stage method is employed to cut down the search time. First, a set of relevant documents are retrieved via VSM. Next, the query is located only in this set of documents. They vary the size of the set with a pruning threshold in VSM stage and analyze the degrade in detection performance with increasing speed.

2.3.3. Spoken Utterance Retrieval

Spoken Utterance Retrieval (SUR) is similar to STD in many ways. The only difference between the two tasks is that SUR aims to locate the utterances that are most similar to a query term while STD aims to locate exact intervals in the database that match the query term. For SUR, the optimal metric is how many times the query term is seen in an utterance, thus it picks those utterances which give the highest expected count of the query term. Expected count is a mere generalization of the classical term frequency metric used in text retrieval. In STD, expected term counts is of secondary importance behind the posterior scores of term occurrences. Although it is possible to use SUR systems to achieve the STD task [34], it is a suboptimal strategy as we discuss in the coming chapters.

3. DATA AND SYSTEM ARCHITECTURE

In this study, we present results on four different STD systems, two of them in Turkish and the rest in English. All these systems integrate IBM's Attila Speech Recognition Toolkit [35] and the STD tools we built over the last two years. This chapter details each system describing the ASR training and STD experimentation data used in each one. In Section 3.1 we give the architecture common to all STD systems presented throughout the thesis. Section 3.2 explains the Turkish systems built here at Bogazici University along with the data we used. Finally in Section 3.3 we give an overview of the English systems utilizing ASR engines built by our associates at IBM. We should note that we do not have direct access to the data and ASR systems in English.

3.1. STD Architecture

Our STD system comprise of five operational blocks: ASR, Indexer, Retriever, Detector and User Interface. Figure 3.1 gives the high-level description of our STD workflow. The system has two modes of operation: offline and online. In the offline mode, the ASR engine converts the speech database into recognition lattices and the Indexer inverts these lattices into an efficient search index. In the online mode, the User Interface takes the query from the user in text form and sends it to the Retriever which performs the actual search on the index. The Retriever first processes the user query to derive the actual search query and then retrieves a ranked list of scored term occurrences. Depending on the indexing units (words, subwords, phones, etc.) and the query itself, the search query may be quite different from the user query, i.e. out-of-vocabulary words replaced with phone strings. The Detector gets the ranked list of putative query term occurrences and decides which ones will be returned to the user. Finally the User Interface provides links to relevant segments in the database. Other than the described architecture, we also have a batch retrieval system used for evaluating the system.

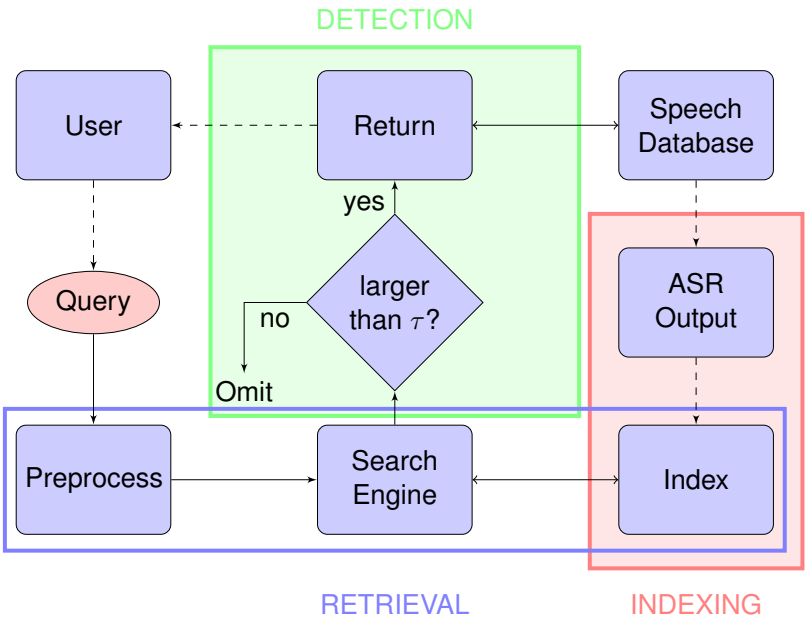


Figure 3.1. STD System Architecture

3.2. Turkish STD Systems

3.2.1. Boğaziçi University Turkish Broadcast News (BUTBN) Database

Boğaziçi University Speech Processing Group has been collecting a large database of Turkish Broadcast News since 2006. Currently, BUTBN database includes 350 hours of manually transcribed speech data collected from one radio (VoA) and four TV channels (CNN Türk, NTV, TRT1, TRT2). Transcriptions follow a common format adapted from Hub4 BN transcription guidelines and are annotated with topic, speaker and background information. In this study we used various subsets of the BUTBN database for building ASR systems and performing STD experiments. Table 3.1 gives the breakdown of current BUTBN database with respect to Hub4 classes: (f0) clean speech, (f1) spontaneous speech, (f2) telephone speech, (f3) background music, (f4) degraded acoustic conditions, and (fx) other.

T1, T2, H and R are mixed data sets that include news programs from all channels under different acoustic conditions. HI is a rather clean data set which includes recordings from the TRT2 News for the Hearing Impaired.

Table 3.1. Breakdown of BUTBN Database (in hours) w.r.t. Hub4 Acoustic Classes

Subset	Use	f0	f1	f2	f3	f4	fx	Total
T1	Training	65.7	15.5	8.3	19.4	71.9	3.2	184.0
H	Held-out	1.1	0.1	0.1	0.5	1.3	0.0	3.1
R	Retrieval	55.2	9.6	13.7	16.7	65.5	2.6	163.3
T2	Training	37.2	10.2	2.8	9.4	39.0	2.0	100.6
HI	Retrieval	11.6	0.0	0.0	0.0	0.0	0.0	11.6
All	-	122.0	25.2	22.1	36.6	138.7	5.8	350.4

3.2.2. Primary Turkish STD System

Primary system utilizes the T1, H and R subsets of BUTBN database for ASR training, ASR optimization and STD experiments respectively. This system is meant to mimic a realistic scenario where a large database of spoken documents is indexed and searched. 163 hours of speech constitutes a fairly large evaluation set for speech retrieval experiments.

Primary LVCSR engine was built with IBM Attila toolkit using the T1 subset. It is a word-based system with a vocabulary of 200K words. The language model uses the manual transcripts of T1 subset and a large text corpus of size 184M words [36]. It is the same as the 200K word-based language model presented in [37]. The LVCSR system's WER on the H and R subsets were 25.9 per cent and 29.9 per cent respectively.

In STD experiments on this setup, we use the R-IV query set which was selected from the reference transcriptions of the R subset. R-IV query terms are confined to the ASR vocabulary. Table 3.2 gives the decomposition of R-IV query set with respect to query length.

Table 3.2. R-IV Query Set Decomposition

Query Length	Number
1-word	2312
2-word	1725
3-word	256
4-word	115
Any	4408

3.2.3. Secondary Turkish STD System

Secondary system utilizes the T2 and HI subsets for ASR training and STD experiments respectively. In this setup, we did not utilize a held-out set for ASR optimization. This STD setup is fairly small compared to the primary one. It was built as a test system, not as an actual evaluation system, during the development of our STD tools.

Secondary LVCSR engine was built with IBM Attila toolkit using the T2 subset. It is a word-based system with a vocabulary of 50K words. The language model uses the manual transcripts of T2 subset and a large text corpus of size 184M words [36]. It is the same as the 50K word-based language model presented in [37]. The LVCSR system's WER on the HI subset was 18.7 per cent.

STD experiments on this setup uses the HI-IV query set which includes all 10229 words common to the system vocabulary and the reference transcriptions of the HI subset.

3.3. English STD Systems

3.3.1. Primary English STD System

This system was built during 2008 CLSP Summer Workshop by the Multilingual Spoken Term Detection (MSTD) group. I was responsible for the development and implementation of the STD component while the data collection, LVCSR development and the query selection were handled by the rest of the group.

In the workshop, our goal was to address out-of-vocabulary (OOV) pronunciation validation using speech in a variety of applications (recognition, retrieval, synthesis) for a variety of types of OOVs (names, places, rare/foreign words). To this end, we selected speech from English broadcast news and 1290 OOV words to be used as queries. Throughout the thesis, we refer to this query set as MSTD-OOV. The OOVs were selected with a minimum of five acoustic instances and four phones per word, and common English words were filtered out to obtain meaningful queries, e.g. NATALIE, PUTIN, QAEDA, HOLLOWAY. Once selected, these queries were removed from the recognizer's vocabulary and all speech utterances containing them were removed from ASR training.

The LVCSR system was built using the IBM Attila toolkit. Acoustic models were trained on 300 hours of HUB4 data excluding utterances containing the selected OOV words. Excluded utterances (around 100 hours) were used as the test set for STD experiments. The language model for the LVCSR system was trained on 400M words from various text sources. The LVCSR system's WER on a standard broadcast news test set RT04 was 19.4 per cent.

3.3.2. Secondary English STD System

This system was also built during the 2008 CLSP Summer Workshop. It uses standard data sets from NIST's 2006 Spoken Term Detection Evaluation [11]. Experiments utilize the three-hour-long Broadcast News subset of STDDEV06 data set and

the DRYRUN06 query set which includes 1100 terms. The LVCSR system is the one used by IBM during the 2006 NIST STD evaluations [29]. Architectural details of the IBM research prototype ASR system can be found in [35]. The LVCSR system's WER on STDDEV06 data set was 12.7 per cent.

4. LATTICE INDEXATION/SEARCH FRAMEWORK

This chapter deals with the problem of constructing an inverted index from a collection of raw ASR lattices. As we motivated in the introduction, indexing ASR lattices (word or sub-word level) is not a straightforward task. Our work extends the general weighted automata indexation/search framework developed for the SUR task [19] to include timing (hence proximity) information inside the index. In Section 4.1 we give the terminology and definitions used throughout the chapter. Section 4.2 recaps the general weighted automata indexation/search framework [19] and proceeds with how it can used for the STD task. We first outline the two-stage STD approach of [20] and lay out its shortcomings. Then we present the early approach we took to solve them along with a critique of how well we did. This solution had been developed before and during the 2008 CLSP Summer Workshop and was published in [21, 38]. Section 4.4 presents a new architecture that is search-time optimal for the STD task.

4.1. Preliminaries

In the coming sections, we will be using the algebra of graphs to formulate our STD framework. More specifically, we will develop an inverted index in weighted finite state transducer form which operates over the cartesian product of three tropical semirings under a lexicographic ordering. To that end, we will deal with semirings, partial and total ordering of sets and weighted finite state automata as mathematical objects. Since our STD framework is a mere generalization/modification of the factor automaton and factor transducer concepts, it is also desirable to have an understanding of those machines before following the mathematical development in Sections 4.2 and 4.4. In the following section, we recap relevant concepts from the graph algebra as well as those from string and automata theory.

4.1.1. Semirings

Definition 1 *A monoid is a triple* $(\mathbb{K}, \otimes, \overline{1})$ *where* $\otimes$ *is a closed associative binary operator on the set* $\mathbb{K}$*, and* $\overline{1}$ *is the identity element for* $\otimes$*, i.e.* $\forall a \in \mathbb{K}$*,* $a \otimes \overline{1} = \overline{1} \otimes a = a$*. A monoid is commutative if* $\otimes$ *is commutative.*

Definition 2 *A semiring is a 5-tuple* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *such that*

(i) $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid,*
(ii) $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid,*
(iii) $\otimes$ *distributes over* $\oplus$*:* $\forall a, b, c \in \mathbb{K}$*,*

$$\begin{aligned} (a \oplus b) \otimes c &= (a \otimes c) \oplus (b \otimes c), \\ c \otimes (a \oplus b) &= (c \otimes a) \oplus (c \otimes b). \end{aligned}$$

(iv) $\overline{0}$ *is an annihilator for* $\otimes$*:* $\forall a \in \mathbb{K}$*,* $a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$*.*

Definition 3 *A semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is* idempotent *if* $\forall a \in \mathbb{K}, a \oplus a = a$*.*

Lemma 1 *If* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is an idempotent semiring, then the relation* $\leq$ *defined by*

$$\forall a, b \in \mathbb{K} : \ (a \leq b) \Leftrightarrow (a \oplus b = a)$$

is a partial order over $\mathbb{K}$*, called the* natural order [39] *over* $\mathbb{K}$*. It is a total order if and only if the semiring has the path property:* $\forall a, b \in \mathbb{K}$*,* $a \oplus b = a$ *or* $a \oplus b = b$*.*

Definition 4 *An idempotent semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is* totally-ordered *if its natural order is a total ordering.*

Definition 5 *Let* $\mathcal{K} = (\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *and* $\leq$ *respectively be a semiring and a partial-*

ordering on $\mathbb{K}$. $\leq$ *is* monotonic *with respect to* $\mathcal{K}$ *if for all* $a, b, c \in \mathbb{K}$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c),$$
$$(a \leq b) \Rightarrow (c \otimes a \leq c \otimes b).$$

$\leq$ *is* negative *if and only if* $\overline{1} \leq \overline{0}$.

Lemma 2 *If* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *be an idempotent semiring, then its natural order is monotonic.*

In speech processing, two semirings are of particular importance. The *log semiring* is defined as

$$\mathcal{L} = (\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$$

where for all $a, b \in \mathbb{R} \cup \{\infty\}$

$$a \oplus_{\log} b = -\log(e^{-a} + e^{-b})$$

along with the conventions $e^{-\infty} = 0$ and $-\log(0) = \infty$. $\mathcal{L}$ is isomorphic to the familiar real or probability semiring $\mathcal{R} = (\mathbb{R}_+, +, \times, 0, 1)$ via a log morphism. The *tropical semiring* is usually defined as

$$\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$$

where the min-convention is used for the tropical addition. Note that $\mathcal{T}$ is idempotent and the natural order over $\mathcal{T}$ is a total order, the usual order of real numbers [39]:

$$\forall a, b \in \mathbb{R}_+ \cup \{\infty\}, (\min\{a, b\} = a) \Leftrightarrow (a \leq b).$$

Alternatively one can use the max-convention to obtain another idempotent semiring

$$\mathcal{T}' = (\mathbb{R}_+ \cup \{-\infty\}, \max, +, -\infty, 0)$$

over which the natural order is another total order, this time the reverse order of real numbers.

Definition 6 *For two partially-ordered semirings* $\mathcal{A} = (\mathbb{A}, \oplus_{\mathbb{A}}, \otimes_{\mathbb{A}}, \overline{0}_{\mathbb{A}}, \overline{1}_{\mathbb{A}})$ *and* $\mathcal{B} = (\mathbb{B}, \oplus_{\mathbb{B}}, \otimes_{\mathbb{B}}, \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{B}})$*, the* product semiring *over* $\mathbb{A} \times \mathbb{B}$ *is defined as*

$$\mathcal{A} \times \mathcal{B} = (\mathbb{A} \times \mathbb{B}, \oplus_{\times}, \otimes_{\times}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}})$$

where for all $a_1, a_2 \in \mathbb{A}$*,* $b_1, b_2 \in \mathbb{B}$*:* $\oplus_{\times}$ *and* $\otimes_{\times}$ *are component-wise operators*

$$\begin{aligned} (a_1, b_1) \oplus_{\times} (a_2, b_2) &= (a_1 \oplus_{\mathbb{A}} a_2, b_1 \oplus_{\mathbb{B}} b_2), \\ (a_1, b_1) \otimes_{\times} (a_2, b_2) &= (a_1 \otimes_{\mathbb{A}} a_2, b_1 \otimes_{\mathbb{B}} b_2). \end{aligned}$$

Note that the natural order over $\mathcal{A} \times \mathcal{B}$, given by

$$((a_1, b_1) \leq_{\times} (a_2, b_2)) \Leftrightarrow (a_1 \oplus_{\mathbb{A}} a_2 = a_1, b_1 \oplus_{\mathbb{B}} b_2 = b_1),$$

defines a partial order, known as product order, even if $\mathcal{A}$ and $\mathcal{B}$ are totally-ordered.

Definition 7 *For two partially-ordered semirings* $\mathcal{A} = (\mathbb{A}, \oplus_{\mathbb{A}}, \otimes_{\mathbb{A}}, \overline{0}_{\mathbb{A}}, \overline{1}_{\mathbb{A}})$ *and* $\mathcal{B} = (\mathbb{B}, \oplus_{\mathbb{B}}, \otimes_{\mathbb{B}}, \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{B}})$*, the* lexicographic semiring *over* $\mathbb{A} \times \mathbb{B}$ *is defined as*

$$\mathcal{A} * \mathcal{B} = (\mathbb{A} \times \mathbb{B}, \oplus_{*}, \otimes_{*}, \overline{0}_{\mathbb{A}} \times \overline{0}_{\mathbb{B}}, \overline{1}_{\mathbb{A}} \times \overline{1}_{\mathbb{B}})$$

where for all $a_1, a_2 \in \mathbb{A}$, $b_1, b_2 \in \mathbb{B}$: $\oplus_*$ *is a lexicographic priority operator*

$$(a_1, b_1) \oplus_* (a_2, b_2) \quad = \quad \begin{cases} (a_1, b_1 \oplus_{\mathbb{B}} b_2) & a_1 = a_2 \\ (a_1, b_1) & a_1 = a_1 \oplus_{\mathbb{A}} a_2 \neq a_2 \\ (a_2, b_2) & a_1 \neq a_1 \oplus_{\mathbb{A}} a_2 = a_2 \end{cases}$$

and $\otimes_*$ *is a component-wise multiplication operator*

$$(a_1, b_1) \otimes_* (a_2, b_2) = (a_1 \otimes_{\mathbb{A}} a_2, b_1 \otimes_{\mathbb{B}} b_2).$$

Unlike $\mathcal{A} \times \mathcal{B}$, the natural order over $\mathcal{A} * \mathcal{B}$, given by

$$((a_1, b_1) \leq_* (a_2, b_2)) \Leftrightarrow \begin{array}{c} (a_1 = a_1 \oplus_{\mathbb{A}} a_2 \neq a_2) \\ \text{or} \\ (a_1 = a_2 \text{ and } b_1 = b_1 \oplus_{\mathbb{B}} b_2) \end{array},$$

defines a total order, known as the lexicographic order, when $\mathcal{A}$ and $\mathcal{B}$ are totally-ordered.

More generally, one can define the product and lexicographic orders (or semirings) on the Cartesian product of n ordered sets. Suppose $\{\mathbb{A}_1, \mathbb{A}_2, \cdots, \mathbb{A}_n\}$ is an n-tuple of sets, with respective strict total orderings $\{<_1, <_2, \cdots, <_n\}$. The product order $<_\times$ of $\{\mathbb{A}_1, \mathbb{A}_2, \cdots, \mathbb{A}_n\}$:

$$(a_1, a_2, \ldots, a_n) <_\times (b_1, b_2, \ldots, b_n) \iff (a_i <_i b_i) \ \ \forall\ i \leq m.$$

Similarly, the lexicographic order $<_*$ of $\{\mathbb{A}_1, \mathbb{A}_2, \cdots, \mathbb{A}_n\}$:

$$\begin{array}{c} (a_1, a_2, \ldots, a_n) <_* (b_1, b_2, \ldots, b_n) \\ \iff \exists\ m > 0,\ \forall\ i < m \quad (a_i = b_i) \wedge (a_m <_m b_m). \end{array}$$

That is, for one of the terms $a_m <_m b_m$ and all the preceding terms are equal.

We should also note that product (or lexicographic) semiring on $\{\mathbb{A}_1, \mathbb{A}_2, \cdots, \mathbb{A}_n\}$ can be recursively defined using the associativity of $\times$ (or $*$) operator:

$$\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n = ((\cdots(\mathcal{A}_1 \times \mathcal{A}_2) \times \cdots) \times \mathcal{A}_n).$$

4.1.2. Weighted Finite-State Automata

A finite-state transducer [40] is a finite automaton whose state transitions (arcs) are labeled with both input and output symbols. Therefore, a path through the transducer encodes a mapping from an input symbol sequence to an output symbol sequence. A weighted finite-state transducer puts weights on arcs in addition to the input and output symbols. Weights may encode probabilities, durations, penalties, or any other quantity that accumulates along paths to compute the overall weight of mapping an input sequence to an output sequence.

Definition 8 *A weighted finite-state transducer T over a semiring $\mathbb{K}$ is an 8-tuple $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ where: Σ is the finite input alphabet of the transducer; Δ is the finite output alphabet; Q is a finite set of states; $I \subseteq Q$ the set of initial states; $F \subseteq Q$ the set of final states; $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$ a finite set of arcs; $\lambda : I \to \mathbb{K}$ the initial weight function; and $\rho : F \to \mathbb{K}$ the final weight function mapping F to $\mathbb{K}$.*

A *weighted finite-state acceptor* $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ is defined in a similar way by simply omitting the output labels. The size of an automaton[1] M is defined as $|M| = |Q| + |E|$.

[1]In finite automata literature, the terms *automaton* and *acceptor* are often used interchangeably. Throughout this thesis, we use the term *automaton* when we do not differentiate between a transducer and an acceptor.

Given an arc $e \in E$, we denote by:

$$\begin{array}{rl} i[e] & \text{its input label,} \\ o[e] & \text{its output label,} \\ w[e] & \text{its weight,} \\ p[e] & \text{its origin or previous state,} \\ n[e] & \text{its destination state or next state.} \end{array}$$

A path $\pi = e_1 \cdots e_k$ is an element of E^* with consecutive arcs satisfying:

$$n[e_{i-1}] = p[e_i], \quad i = 2, \ldots, k.$$

We extend n and p to paths by setting:

$$n[\pi] = n[e_k] \quad \text{and} \quad p[\pi] = p[e_1].$$

A cycle π is a path whose origin and destination states coincide such that $n[\pi] = p[\pi]$. We denote by $P(q, q')$ the set of paths from q to q', by $P(q, x, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and by $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$ and output label $y \in \Delta^*$. These definitions can be extended to subsets $R, R' \in Q$, by:

$$P(R, x, R') = \bigcup_{q \in R, q' \in R'} P(q, x, q').$$

The labeling functions i and o and the weight function w can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent arcs, and the weight of a path as the $\otimes$-product of the weights of its constituent arcs:

$$\begin{aligned} i[\pi] &= i[e_1] \ldots i[e_k], \\ o[\pi] &= o[e_1] \ldots o[e_k], \end{aligned}$$

$$w[\pi] \quad = \quad w[e_1] \otimes \ldots \otimes w[e_k].$$

We also extend w to any finite set of paths Π by setting:

$$w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi].$$

A successful path in an automaton M is a path from an initial state to a final state. A symbol sequence x is accepted by M if there exists a successful path π labeled with x on the input side. M is *unambiguous*, if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x on the input side. Thus, an unambiguous transducer defines a function.

The weight associated by a transducer T to any input-output string pair $(x, y) \in \Sigma^* \times \Delta^*$ is given by:

$$[\![T]\!](x, y) = \bigoplus_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

and $[\![T]\!](x, y)$ is defined to be $\overline{0}$ when $P(I, x, y, F) = \emptyset$. Similarly, the weight associated by an acceptor A to any input string $x \in \Sigma^*$ is given by:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

and $[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$.

Note that the second operation of the tropical semiring and the log semiring as well as their identity elements are identical. Thus the path weights in an automaton M over the tropical semiring do not change if M is viewed as an automaton over the log semiring and vice-versa. Log semiring ensures that weights of identically labeled paths in an ambiguous automaton are merged with the $\oplus_{\log}$ operation during weighted determinization and minimization operations [41]. Tropical semiring, on the

other hand, is idempotent and the natural order associated with it is a total order. Natural order is the default total order (under the requirements of section 4.1.1) used by the shortest path and pruning algorithms operating on weighted automata over an idempotent semiring. It is the natural one to use given that it generally needs to be total, monotonic and negative: total so that all weights can be compared, monotonic so there is a practical algorithm, and negative so that the “free" weight $\overline{1}$ is preferred to the “disallowed" weight $\overline{0}$ [42].

4.1.3. Factor Automata

Definition 9 *Given two strings u and v in Σ^*, v is a factor (substring) of u if $u = xvy$ for some x and y in Σ^*. More generally, v is a factor of $L \subseteq \Sigma^*$ if v is a factor of some $u \in L$. The factor automaton $F(u)$ of a string u is the minimal deterministic finite-state acceptor recognizing exactly the set of factors of u.*

$F(u)$ can be built in linear time and its size is linear in $|u|$ [43, 44] where $|x|$ denotes the length of a string $x \in \Sigma^*$. Similarly, we denote by $F(A)$

the minimal deterministic acceptor recognizing the set of factors of a finite acceptor A, that is the set of factors of the strings recognized by A. A recent work [45] showed that the size of factor automaton $F(A)$ is linear in the size of A and provided an algorithm for the construction of $F(A)$ in linear time.

4.2. Factor Transducer of Weighted Automata

General indexation of weighted automata provides an efficient means of indexing speech utterances based on the within utterance expected counts of factors seen in the data [19]. This section recaps the general weighted automata indexation algorithm which aims to construct an efficient index for a large set of speech utterances.

Assume that for each speech utterance u_i in the data-set considered, $i = 1, \ldots, n$, a weighted automaton A_i over the log semiring with alphabet Σ, i.e. phone or word

lattice output by ASR, is given. The problem consists of creating an index that can be used for direct search of any factor of any string accepted by these automata. Note that this problem crucially differs from classical text indexation problems in that the input data is uncertain. The algorithm given in this section makes use of the weights associated to each string by the input automata.

In the most basic form, this algorithm leads to an index represented as a weighted finite state transducer (WFST) where each factor x leads to a successful path over the input labels for each utterance that particular substring was observed. Output labels of these paths carry the utterance indices $i \in \{1, \ldots, n\}$, while path weights give the negative log of within utterance expected counts. The index construction algorithm is simple and based on general weighted automata and transducer algorithms. It can be viewed as a generalization of the notion of factor automaton which is a classical representation used in text indexation [44, 46]. In the following subsections we describe the consecutive stages of the algorithm.

4.2.1. Preprocessing

When the automata A_i are word or phone lattices output by a speech recognition or other natural language processing system, the path weights correspond to joint probabilities. We can apply to A_i a general weight-pushing algorithm in the log semiring [41] which converts these weights into the desired ($-\log$ of) posterior probabilities. More generally, the path weights in the resulting automata can be interpreted as log-likelihoods. We denote by P_i the corresponding probability distribution. When the input automaton A_i is acyclic, the complexity of the weight-pushing algorithm is linear in its size ($O(|A_i|)$). Figures 4.1(b)(d) illustrate the application of weight pushing algorithm to the automata of Figures 4.1(a)(c).

4.2.2. Index Construction

Let $B_i = (\Sigma, Q_i, I_i, F_i, E_i, \lambda_i, \rho_i)$ denote an acceptor over $\mathcal{L}$ obtained by the application of the weight pushing algorithm to the automaton A_i. The weight associated

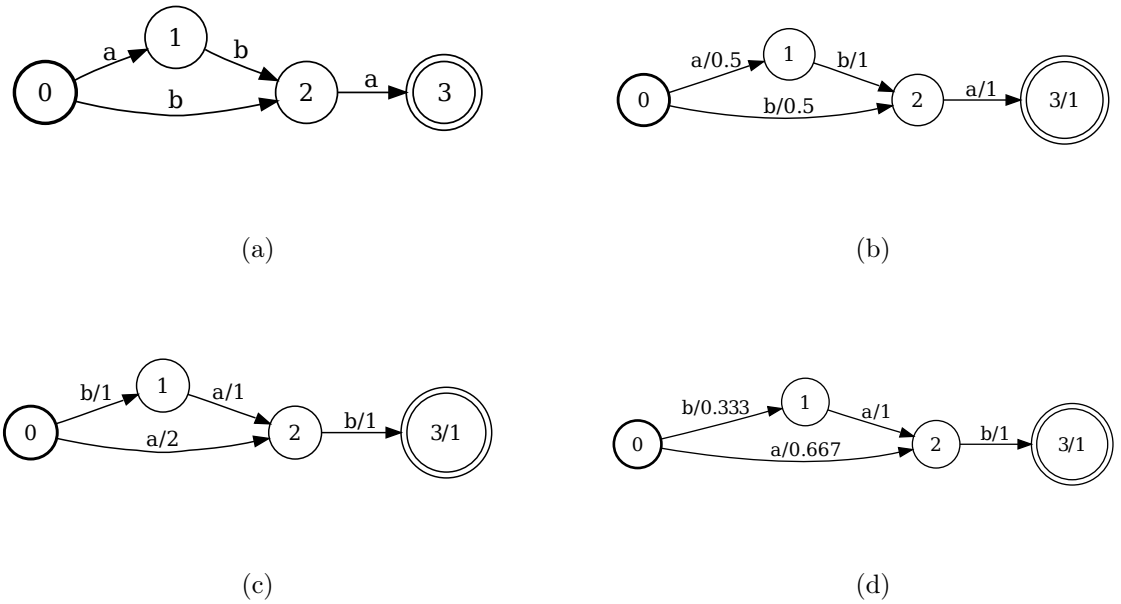


Figure 4.1. Weighted automata over the real semiring $\mathcal{R}$ (a) A_1 and (c) A_2. (b) B_1 and (d) B_2 over the real semiring obtained by applying weight pushing to A_1 and A_2 respectively.

by B_i to each string it accepts can be interpreted as the log-likelihood of that string for the utterance u_i given the models used to generate the automata. More generally, B_i defines a probability distribution P_i over all strings $x \in \Sigma^*$ which is just the sum of the probability of all paths of B_i in which x appears. For each state $q \in Q_i$, we denote by $d[q]$ the shortest distance from I_i to q (or $-\log$ of the forward probability) and by $f[q]$ the shortest distance from q to F_i (or $-\log$ of the backward probability):

$$d[q] = \bigoplus_{\pi \in P(I_i, q)}^{\log} (\lambda_i(p[\pi]) + w[\pi]) \tag{4.1}$$

$$f[q] = \bigoplus_{\pi \in P(q, F_i)}^{\log} (w[\pi] + \rho_i(n[\pi])) \tag{4.2}$$

The shortest distances $d[q]$ and $f[q]$ can be computed for all states $q \in Q_i$ in linear time ($O(|B_i|)$) when B_i is acyclic [39]. Let Π_i denote the set of paths in B_i and $C_x(u_i)$

denote the number of occurrences of a factor x in u_i. Then,

$$-\log(E_{P_i}[C_x(u_i)]) = \bigoplus_{\substack{i[\pi]=x \\ \pi\in\Pi_i}}^{\log} d[p[\pi]] + w[\pi] + f[n[\pi]] \tag{4.3}$$

gives the expected count of having the factor-pair x in u_i.

From the weighted acceptor B_i over $\mathcal{L}$, one can derive a timed factor transducer T_i over $\mathcal{T}$ in two steps:

(i) Factor Selection. In the general case we select all of the factors in the following way:
- Replace each arc $(p, a, w, q) \in Q_i \times \Sigma \times \mathbb{R} \times Q_i$ with $(p, a, a, w, q) \in Q_i \times \Sigma \times \Sigma \times \mathbb{R} \times Q_i$;
- Create a new state $s \notin Q_i$ and make s the unique initial state;
- Create a new state $e \notin Q_i$ and make e the unique final state;
- Create a new arc $(s, \epsilon, \epsilon, d[q], q)$ for each state $q \in Q_i$;
- Create a new arc $(q, \epsilon, i, f[q], e)$ for each state $q \in Q_i$;

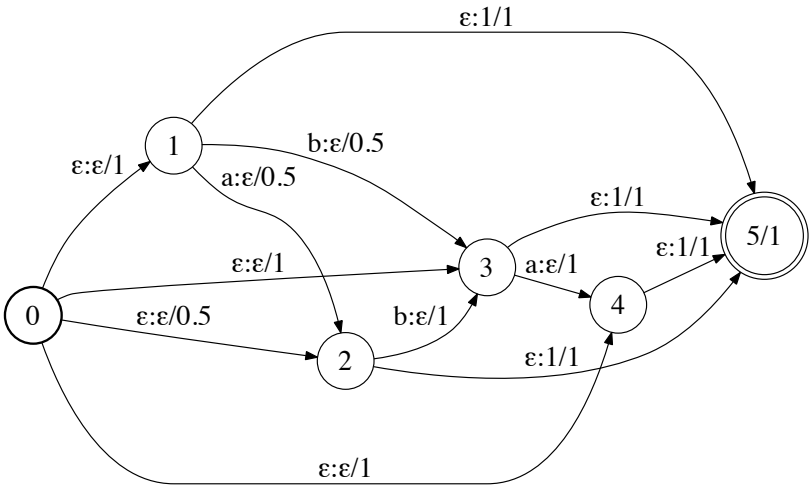
(ii) Optimization. The resulting transducer can be optimized by applying weighted ϵ-removal, determinization, and minimization over the $\mathcal{L}$ semiring by viewing it as an acceptor, i.e. input-output labels are encoded as a single label. Final result is obtained by mapping each arc weight:

$$w \in \mathcal{L} \rightarrow w \in \mathcal{T}.$$

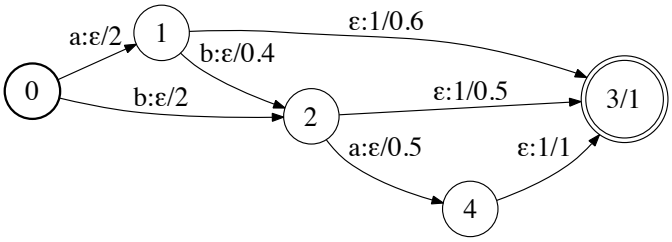
It is clear from Equation 4.3 that for any factor $x \in \Sigma^*$:

$$[\![T_i]\!](x, i) = -\log(E_{P_i}[C_x]) \in \mathcal{T} \tag{4.4}$$

Figures 4.2(a)(b) illustrate the construction of the weighted factor transducer from a weighted automaton.



(a)



(b)

Figure 4.2. Construction of factor transducer T_1 over the real semiring $\mathcal{R}$ from the weighted automaton B_1 given in Figure 4.1(b): (a) intermediary result after factor selection and (b) resulting transducer T_1 over the real semiring after optimization.

The factor transducer T of the entire data-set is constructed by

- taking the union U of individual timed factor transducers built for each utterance

$$U = \bigcup_i T_i,\ i = 1, \ldots, n;$$

- applying weighted ϵ-removal and determinization over the $\mathcal{T}$ semiring to the acceptor obtained by encoding the input-output labels of U as a single label;
- defining T as the transducer obtained by sorting U on the input side [42] after decoding its labels;

Figure 4.3 illustrates the construction and optimization of full factor transducer T.

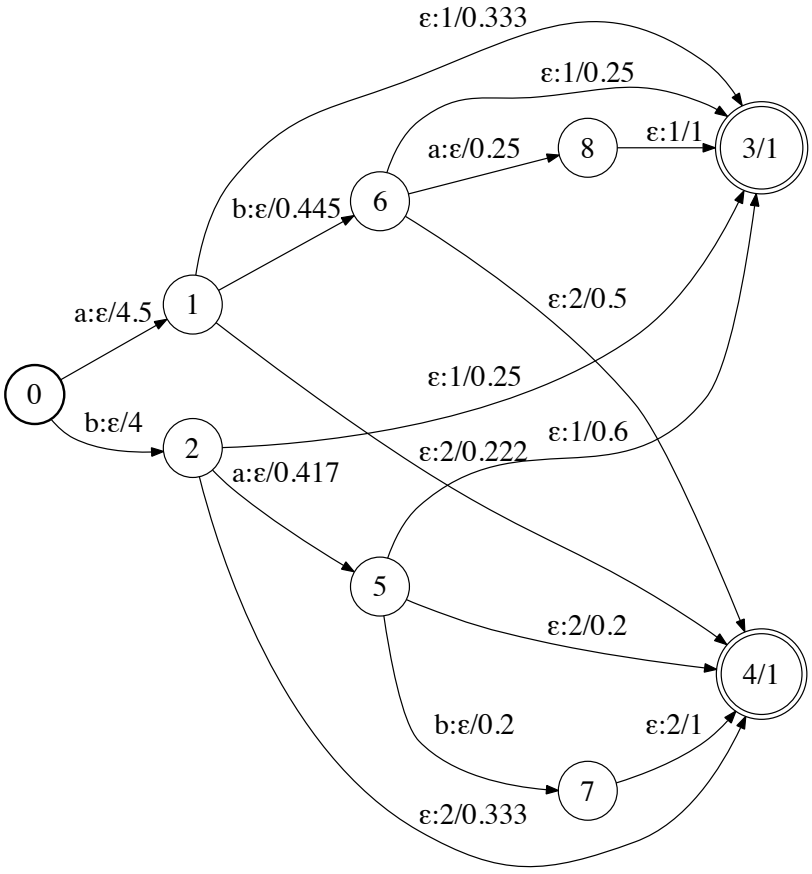


Figure 4.3. Factor transducer T over the real semiring $\mathcal{R}$ obtained from the weighted automata B_1 and B_2 given in Figures 4.1(b)(d).

4.2.3. Search

As far as the SUR task is concerned, the full index represented by the weighted finite-state transducer T is optimal. Indeed, T contains no arcs with input ϵ other than the final transitions labeled with an output index and it is deterministic. Thus, the set of indices I_x of the weighted automata containing a factor x can be obtained in $O(|x|+|I_x|)$ by reading in T the unique path with input label x and then the transitions with input ϵ and output a distinct index label.

The user query is typically an unweighted string, but it can be given as an arbitrary weighted automaton X. This covers the case of Boolean queries or regular

expressions which can be compiled into automata. The response R to a query X is another automaton obtained by

- composing X with T on the input side [47] and projecting the resulting transducer onto its output labels

$$P = \Pi_2(X \circ T);$$

- removing the ϵ transitions and finally sorting P with the shortest-path algorithm to get R.

Each successful path π in R carries an automaton identifier on its label $i[\pi]$, and a $-\log$ expected count on its weight $w[\pi] \in \mathcal{T}$. A simple traversal over R in the arc order gives results sorted with the natural order of $\mathcal{T}$, in other words results sorted with respect to decreasing expected count of query X in each automaton $A_i,\ i \in I_X$. This is an ideal result for the SUR task where the aim is to return (utterance ID, relevance score) pairs for each query. Expected count of a query term within a particular utterance is a direct relevance metric for the SUR task, that can be used to rank the results obtained from the index. Note that R can be pruned before traversal to retain the most likely responses only. The pruning threshold may be varied to achieve different operating points.

4.3. Spoken Term Detection with Factor Transducer

Even though the factor transducer of weighted automata was designed with the SUR task in mind, it can be used in other SR applications as well. An example is the two-stage STD system of [20] which we explain below.

4.3.1. Two-stage Spoken Term Detection with Factor Transducer

In this setup actual utterances retrieved with the factor transducer are aligned with the query automaton to retrieve the timing of query words since the index provides

the utterance information only. A variation can be to search the query labels directly in the automata obtained from utterances.

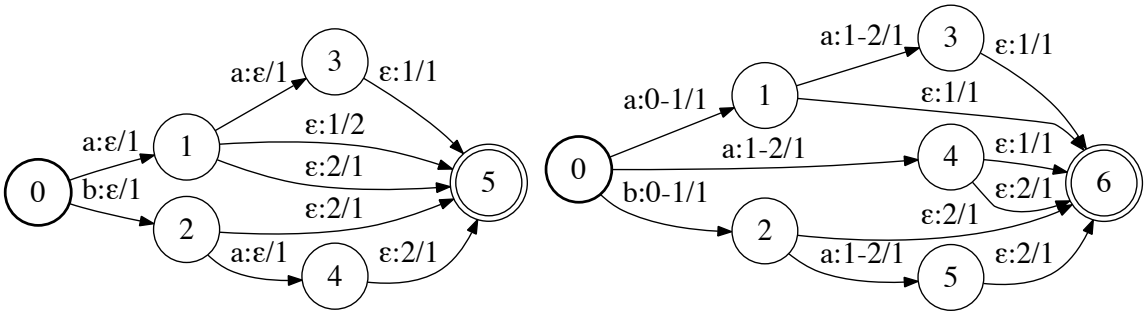
Prominent complication of the two-stage STD setup is that it requires a time-costly alignment operation which is performed online, a deal-breaker on large archives. Furthermore, the index stores utterance-wide expected counts which are not direct relevance measures for STD since a query term may appear more than once in an utterance. Let an *occurrence* represent a path labeled with the query term in an automaton, and a *cluster* all such paths with overlapping time-alignments. Each time a term occurs within an utterance, it will contribute to the utterance-wide expected count and the contribution of distinct clusters will be lost. Since the index provides neither the individual contribution of each cluster to the expected count nor the number of clusters, both of these parameters have to be estimated in the second stage which in turn decreases the detection performance.

4.3.2. Spoken Term Detection with Modified Factor Transducer

To overcome the drawbacks of two-stage STD strategy, we extend the factor transducer structure by embedding the time-alignment information of factors into the output labels. This modified structure is a generalization of the factor transducer to meet the extra requirements of the STD task.

Construction of the modified factor transducer requires processing the time alignment information of factors since every distinct alignment will lead to another index entry which means factors with slightly off time-alignments will be indexed separately. Note that this is a concern only if we are indexing lattices, consensus networks or single-best transcriptions do not have such a problem by construction. Also note that no such preprocessing was required for the original structure since all occurrences in an automaton would have to be combined anyway. To alleviate the time-alignment issue, we cluster the factor occurrences prior to indexing. Desired behavior is achieved by assigning the same time-alignment label (start-end) to all occurrences in a cluster. Figure 4.4 illustrates the differences between the factor transducer and the modified

factor transducer for a simple database consisting of two strings: "a_{0-1} a_{1-2}" and "b_{0-1} a_{1-2}". While original factor transducer construction algorithm does not differentiate between the occurrences in an automaton and combines all of them, the modified algorithm classifies them into distinct clusters and combines those that fall into the same cluster.



(a) Factor Transducer (b) Modified Factor Transducer

Figure 4.4. Comparison of the factor transducer structures over the real semiring $\mathcal{R}$. (a) Factor transducer omits the time alignment information. (b) Modified factor transducer stores time alignment in the output labels.

Main advantage of the modified index is that second stage of the previous setup is no longer required since the new index already provides all the information we need for an actual hit: the utterance id and start-end times. Eliminating the second stage significantly improves the search time since time-alignment of utterances takes much more time compared to their retrieval from the index. Another advantage of the new structure is that it distributes the total expected count among clusters, thus the hits can now be ranked based on their posterior probability scores. To be more precise, assume we have a path π in the modified index with a particular factor x on the input labels $i[\pi] = x$. Weight of this path $w[\pi]$ corresponds to the posterior probability of x given the lattice and the time interval indicated by the path output labels $o[\pi]$. The modified index provides posterior probabilities instead of expected counts.

On the other hand, embedding time-alignment information leads to a larger index which is no longer search-time optimal. Common paths among different automata are largely reduced due to the mismatch between time-alignment labels and this in turn degrades the effectiveness of the weighted automata optimization. To control the

index size, we quantize the time-alignments (25ms steps) during preprocessing without altering the STD performance.

4.4. Timed Factor Transducer of Weighted Automata

This section presents an algorithm for the construction of an efficient timed index for a large set of speech utterances. We propose a new factor transducer structure, which stores the timing information on the arc weights, solving the optimality issues associated with the structure presented in Section 4.3.2.

We assume that for each speech utterance u_i of the data-set considered, $i = 1, \ldots, n$, a weighted automaton A_i over the log semiring with alphabet Σ, i.e. phone or word lattice output by ASR, and a list L_i of state potentials, i.e. timing of lattice nodes, are given. The problem consists of creating a timed index that can be used for direct search of any factor of any string accepted by these automata.

The main idea is that the timed index can be represented by a weighted finite-state transducer T mapping each factor x to the set of automata in which x appears, start-end times of the intervals where x appears in each automaton and the negative log of the posterior probabilities of x actually occurring in each automaton in the corresponding time interval. It can be viewed as a generalization of the procedure given in Section 4.2. Following subsections detail the consecutive stages of the algorithm.

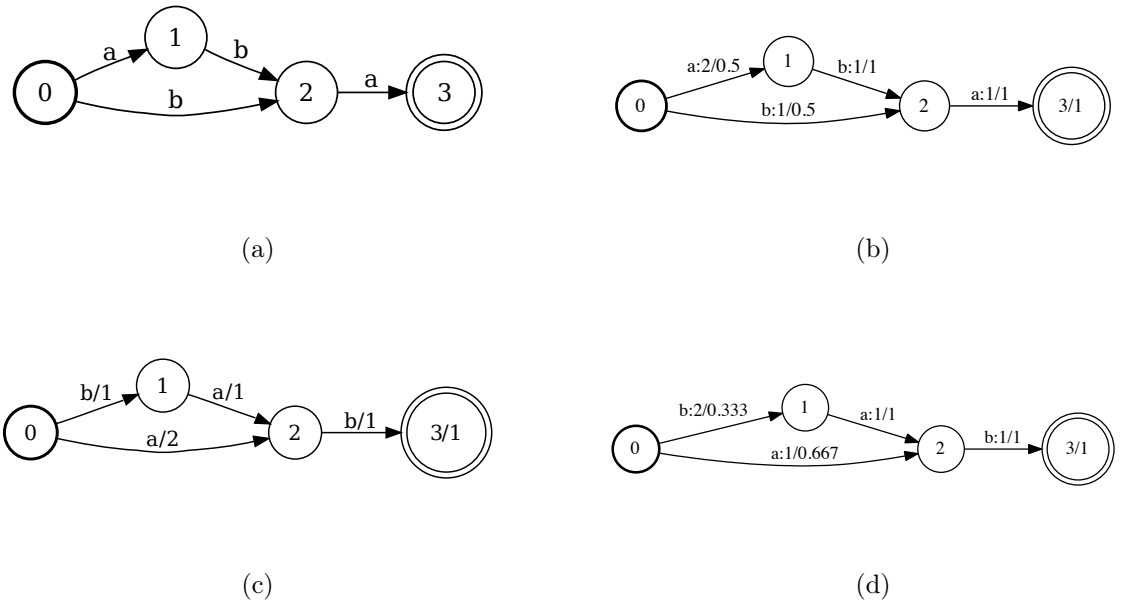
4.4.1. Preprocessing

As we explained in Section 4.2.1, we first perform weight-pushing in the log semiring on A_i, $i = 1, \ldots, n$. The generic algorithm given in Section 4.2 does not make a discrimination between non-overlapping occurrences of a factor in an utterance. While this is the desired behavior for the SUR problem, in the case of STD we would like to keep separate indices for non-overlapping occurrences as we previously discussed in Section 4.3.1. When the lists L_i are state potentials giving the timing information of lattice nodes, this separation can be achieved by clustering the arcs with the same

input label and overlapping time-spans. To achieve this, we use a single-pass clustering algorithm similar to the WFST encoding/decoding operations [42]. In effect, we convert the input automaton to a transducer where each arc carries a cluster identifier on the output label.

Formally, we traverse the input automaton (typically in topological order) and form a hash table H_1 mapping (input label, start time, end time) triplets to positions in l which is a list of cluster identifiers, and another hash table H_2 mapping input labels to the number of non-overlapping clusters seen for that label. For each arc e we traverse, we form the triplet $(i[e], L_i[p[e]], L_i[n[e]])$ and check if another triplet (i, t_s, t_e) with the same label ($i = i[e]$), and overlapping time-span ($[t_s, t_e] \cap [L_i[p[e]], L_i[n[e]]] \neq \emptyset$) already exists in H_1. If so, we assign $o[e] = l[H_1(i, t_s, t_e)]$. If not, we increment $H_2(i[e])$, the number of clusters seen for that label; push $H_2(i[e])$ to the back of l; add H_1 a new entry mapping $(i[e], L_i[p[e]], L_i[n[e]])$ to the last position in l; and finally assign $o[e] = H_2(i[e])$. At the end of traversal, all overlapping arcs with the same input label carry the same cluster identifier on the output label. In other words, each (input label, output label) pair in the output transducer designates an arc cluster. Figures 4.5(b)(d) illustrate the application of preprocessing algorithm to the automata of Figures 4.5(a)(c). Note that cluster identifiers are in reverse order.

Note that the output of this clustering algorithm depends on the traversal order. Suppose the input automaton contains two adjacent arcs with the same input label, i.e. a reduplication. If there happens to be another arc with the same input label and a time-span overlapping both of the former arcs, then depending on the traversal order, the algorithm might form a single cluster or two clusters from these arcs. Reduplication problem can be solved with a two-pass algorithm which first identifies the largest set of non-overlapping arcs and then classifies the rest according to maximal overlap. Although not an exact solution, single-pass clustering over a topologically ordered automaton works very well in practice. The upside is that our algorithm can be applied online while traversing the input automatons for indexing.



(a) (b)

(c) (d)

Figure 4.5. Weighted automata over the real semiring $\mathcal{R}$ (a) A_1 and (c) A_2 (time alignments $L_1 = [0, 1, 2]$ and $L_2 = [0, 1, 2]$). (b) B_1 and (d) B_2 over the real semiring obtained by applying weight pushing to A_1 and A_2 respectively.

4.4.2. Index Construction

Let $B_i = (\Sigma, \Delta, Q_i, I_i, F_i, E_i, \lambda_i, \rho_i)$ denote a transducer over $\mathcal{L}$ obtained by the application of the weight pushing and clustering algorithms to the automaton A_i. The weight associated by B_i to each string it accepts can be interpreted as the log-likelihood of that string for the utterance u_i given the models used to generate the automata. The output string associated by B_i to each string it accepts gives the string of cluster identifiers. More generally, B_i defines a probability distribution P_i over all string pairs $(x, y) \in \Sigma^* \times \Delta^*$ which is just the sum of the probability of all paths of B_i in which (x, y) appears. Let Π_i denote the set of paths in B_i and $C_{x,y}(u_i)$ denote the number of occurrences of a factor-pair (x, y) in u_i. Then,

$$-\log(E_{P_i}[C_{x,y}(u_i)]) = \bigoplus_{\substack{i[\pi]=x,\ o[\pi]=y \\ \pi \in \Pi_i}}^{\log} d[p[\pi]] + w[\pi] + f[n[\pi]] \tag{4.5}$$

gives the expected count or equivalently posterior probability of having the factor-pair (x, y) in u_i where $d[q]$ and $f[q]$ are as defined in Equations 4.1 and 4.2. $E_{P_i}[C_{x,y}(u_i)]$ is a proper posterior probability even when there are multiple occurrences of a factor x in u_i since y will be different for non-overlapping occurrences of x. Similarly, let $L_i[q]$ denote the timing of state $q \in Q_i$ and $S_{x,y}(u_i)$ and $E_{x,y}(u_i)$ denote the ensemble boundaries (start and end times respectively) of the (x, y) cluster in u_i. Then,

$$S_{x,y}(u_i) = \min_{\substack{i[\pi]=x,\ o[\pi]=y \\ \pi \in \Pi_i}} L_i[p[\pi]], \tag{4.6}$$

$$E_{x,y}(u_i) = \max_{\substack{i[\pi]=x,\ o[\pi]=y \\ \pi \in \Pi_i}} L_i[n[\pi]]. \tag{4.7}$$

From the weighted transducer B_i over $\mathcal{L}$ and L_i, one can derive a timed factor transducer T_i over $\mathcal{T} * \mathcal{T} * \mathcal{T}$ in two steps:

(i) Factor Selection. In the general case we select all of the factors in the following way:

- Map each arc weight (see section 4.1.1):

$$w \in \mathcal{L} \rightarrow \{w, \overline{1}, \overline{1}\} \in \mathcal{L} \times \mathcal{T} \times \mathcal{T}';$$

- Create a new state $s \notin Q_i$ and make s the unique initial state;
- Create a new state $e \notin Q_i$ and make e the unique final state;
- Create a new arc $(s, \epsilon, \epsilon, \{d[q], L_i[q], \overline{1}\}, q)$ for each state $q \in Q_i$;
- Create a new arc $(q, \epsilon, i, \{f[q], \overline{1}, L_i[q]\}, e)$ for each state $q \in Q_i$;

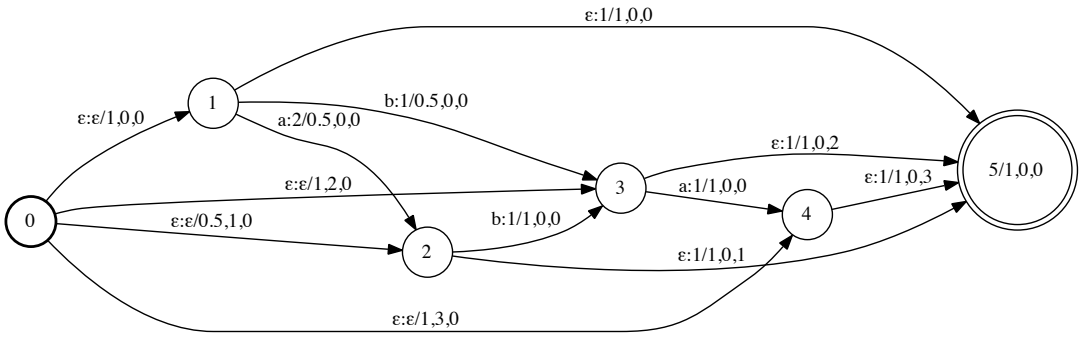
(ii) Optimization. The resulting transducer can be optimized by applying weighted ϵ-removal, determinization, and minimization over the $\mathcal{L} \times \mathcal{T} \times \mathcal{T}'$ semiring by viewing it as an acceptor, i.e. input-output labels are encoded as a single label. Final result is obtained by mapping each arc weight:

$$\{w_1, w_2, w_3\} \in \mathcal{L} \times \mathcal{T} \times \mathcal{T}' \rightarrow \{w_1, w_2, w_3\} \in \mathcal{T} * \mathcal{T} * \mathcal{T}.$$

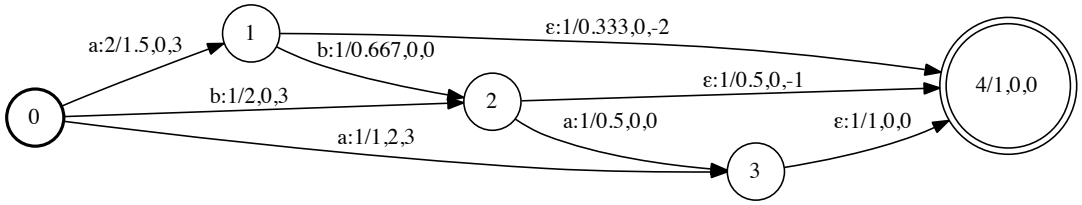
During optimization paths carrying the same factor-pair, in other words overlapping occurrences, are merged by adding the posterior probabilities in $\mathcal{L}$, start times in $\mathcal{T}$ and end times in $\mathcal{T}'$. It is clear from Equations 4.5, 4.6, 4.7 that for any factor $(x, y) \in \Sigma^* \times \Delta^*$:

$$[\![T_i]\!](x, y \cdot i) = \{-\log(E_{P_i}[C_{x,y}]), S_{x,y}, E_{x,y}\} \in \mathcal{T} * \mathcal{T} * \mathcal{T} \tag{4.8}$$

where $y \cdot i$ denotes the concatenation of the string of cluster identifiers y and the automaton identifier i. Figures 4.6(a)(b) illustrate the construction of the timed factor transducer from a weighted automaton.



(a)



(b)

Figure 4.6. Construction of timed factor transducer T_1 from the weighted automaton B_1 given in Figure 4.5(b) and $L_1 = [0, 1, 2]$: (a) intermediary result after factor selection and (b) resulting transducer T_1 after optimization over the $\mathcal{R} \times \mathcal{T} \times \mathcal{T}'$ semiring.

In general, the timed factor transducer T_i is not deterministic. However it is non-deterministic, hence ambiguous, only if there are non-overlapping occurrences of a factor in A_i. Even when this is the case, the amount of non-determinism present in T_i is negligible. Thus, we may safely assume that T_i will be a deterministic, hence unambiguous machine. When viewed as an acceptor by encoding input-output labels as a single label, T_i is always unambiguous. Note that the natural order associated with $\mathcal{T} * \mathcal{T} * \mathcal{T}$, being a total order, supports shortest-path and pruning operations.

The timed factor transducer T of the entire data-set is constructed by

- taking the union U of individual timed factor transducers built for each utterance

$$U = \bigcup_i T_i,\ i = 1, \ldots, n;$$

- encoding the input-output labels of U as a single label and applying weighted ϵ-removal, determinization, and minimization over the $\mathcal{T} * \mathcal{T} * \mathcal{T}$ semiring;
- decoding the labels of U and sorting the arcs with respect to input labels [42];
- and defining T as the transducer obtained by removing all output labels (cluster identifiers) except for those (automaton identifiers) on the arcs going into the unique final state;

We should note that there is no successful path shared between $T_i,\ i = 1, \ldots, n$ since each T_i carries a unique automaton identifier on the output labels of transitions into the unique final state. Therefore we are free to optimize U over any semiring on $\mathbb{R}^3$. However, $\mathcal{T} * \mathcal{T} * \mathcal{T}$, being a total order, gives us the opportunity to perform pruning operations before or during the optimization. Figure 4.7 illustrates the construction and optimization of full timed factor transducer T.

4.4.3. Search

Unlike T_i, T is always ambiguous since the same factor appears in many different utterances. However, this ambiguity does not hurt the optimality of the index since 1)

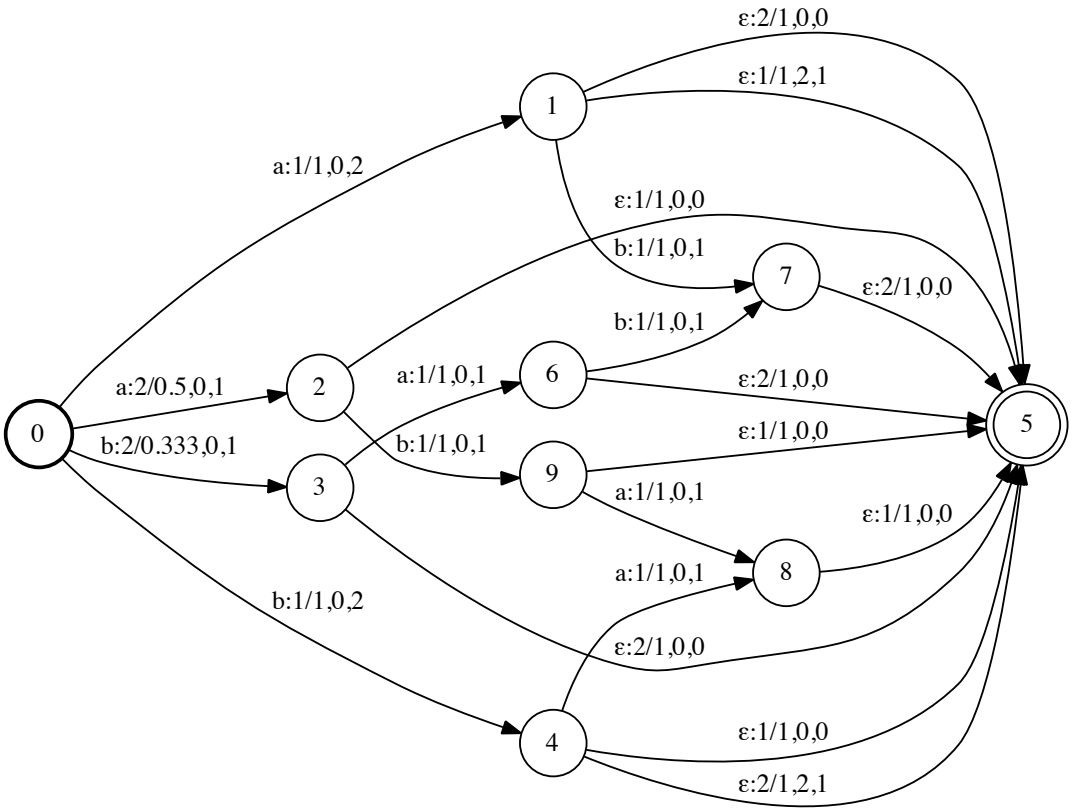


Figure 4.7. Timed Factor transducer T over the $\mathcal{R} \times \mathcal{T} \times \mathcal{T}'$ semiring obtained from the weighted automata B_1 and B_2 given in Figures 4.5(b)(d) along with $L_1 = [0, 1, 2]$ and $L_2 = [0, 1, 2]$.

T contains no arcs with input ϵ other than the final transitions which have automaton identifiers on the output and 2) T is "almost deterministic". Let $M_i(x),\ i = 1, \ldots, n$ represent the number of clusters of a factor x in A_i, i.e. $M_1(x) = 1$ if all occurrences of x in A_1 overlap forming a single cluster. Then, T has $N = \max_i M_i(x)$ paths, labeled with x on the input side, that originate from the unique initial state and terminate at M distinct penultimate (non-final) states. These M states are connected to the unique final state with a total of $M = \sum_i M_i(x)$ arcs that have ϵ labels on the input and automaton identifiers on the output side. Furthermore since the arcs of T are sorted on the input side, the set of paths carrying the label x can be traversed at $O(N|x| + M)$ in the worst case scenario. In the other limit, given that all individual factor transducers are deterministic, i.e. $M_i(x) \leq 1,\ i = 1, \ldots, n$ for any factor x, then the search

complexity for any query x is $O(|x|+M)$ which means T is deterministic as well (except for the final transitions). We say T is almost deterministic since $M_i(x) \leq 1,\ i = 1, \ldots, n$ for almost all factors. We should also note that $O(N|x| + M)$ is an upper bound on the search complexity and the actual complexity is close to $O(|x| + M)$.

Given a query automaton Q, the response automaton R is obtained with the procedure explained in Section 4.2.3. R is a simple acceptor which contains M arcs between the initial state and final states. Each successful path π in R carries an automaton identifier on its label $i[\pi]$, and a ($-\log$ posterior probability, start time, end time) triplet on its weight $w[\pi] \in \mathcal{T} * \mathcal{T} * \mathcal{T}$. A simple traversal over R in the arc order gives results sorted with the natural order of $\mathcal{T} * \mathcal{T} * \mathcal{T}$. Note that R can be pruned before traversal to retain the most likely responses only. The pruning threshold may be varied to achieve different operating points.

4.5. Experiments

In this section, we provide experiments comparing the three STD schemes (Two-Stage Retrieval with Factor Transducer, Retrieval with Modified Factor Transducer, Retrieval with Timed Factor Transducer) in terms of index size and total search time. We also analyze the change of search time as the query length changes (only for the last two schemes). Experiments in this section utilize the BUTBN-R and STDDEV06 data sets, R-IV and DRYRUN06 query sets.

ASR engines make use of hyper-parameters to determine how much of the search network will be output as the recognition lattice. For the experiments of this section, we first extracted a fairly large word lattice (five back-pointers per word trace) for each utterance. Then, we pruned the raw lattices with different beam widths and conducted the same set of experiments for each width. Following subsections compares the results in two dimensions. On the one dimension there is the lattice beam width and on the other the three retrieval schemes we investigated.

4.5.1. Baseline Detection Performance

This subsection gives the baseline detection performances of the three systems considered. As seen in Figure 4.8 and Figure 4.9, there is no significant difference between the systems as far as the term detection performance is concerned. Due to the minor variation between the system implementations, ATWV curves do not exactly match. It is evident from the figures that ATWV values converge around a beam width of four or five. Increasing lattice beam further these values does not provide a significant gain in ATWV. This is an expected behavior - and is in accordance with previous literature [34] - since the redundancy in lattices increase with the lattice beam width.

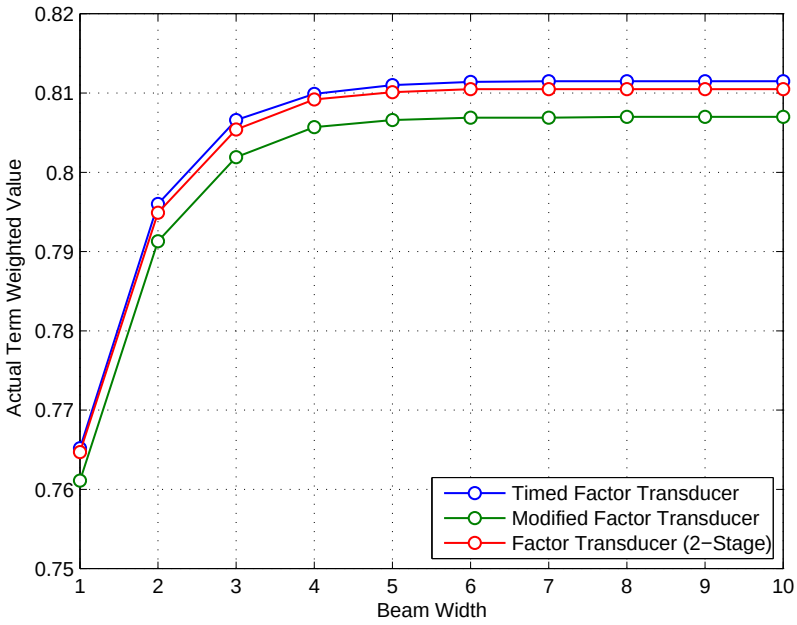


Figure 4.8. ATWV vs. lattice beam (BUTBN-R Data Set & R-IV Query Set)

4.5.2. Index Size

In retrieval applications, size of the inverted index is an important application concern. Preferably, the index size should be much smaller than the actual data size. The BUTBN-R data set includes 163 hours of speech and occupies around 25 GBs of disk space in 16KHz PCM form. The total size of raw ASR lattices extracted from BUTBN-R data set is around 2GBs. STDDEV06 data set includes three hours of

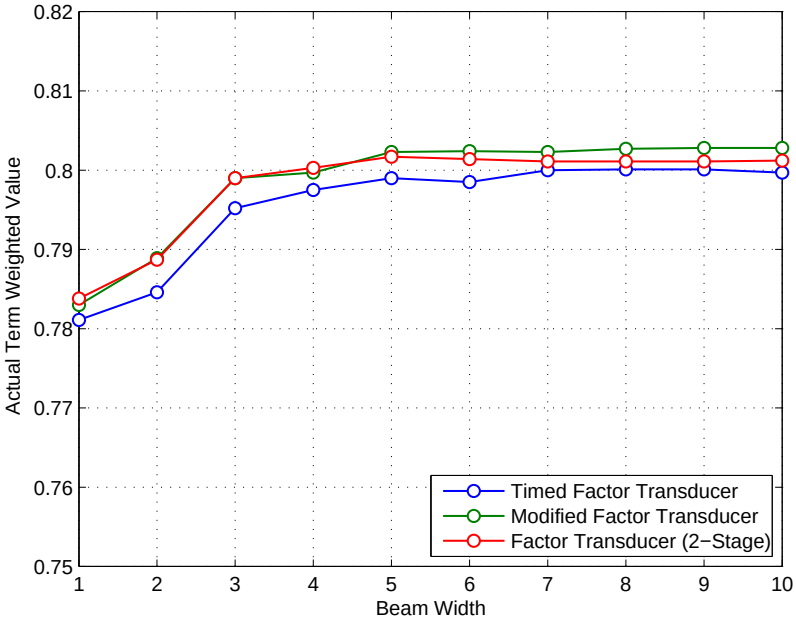


Figure 4.9. ATWV vs. lattice beam (STDDEV06 Data Set & DRYRUN06 Query Set)

speech and occupies 500 MBs of disk space in 16KHz PCM form.

The size of an STD index, or any index for that matter, heavily depends on the amount and type of data indexed. In our case, the data size depends on the total amount of speech and the beam width of the ASR lattices used in indexation. Figure 4.10 and Figure 4.11 show the change of index size as the lattice beam increases.

Modified factor transducer has the smallest size at all beam widths. This may appear rather unexpected since the factor transducer carries much less information compared to the modified one. However, modified factor transducer is not deterministic like the other two since arc output labels of this machine carry time alignment information of corresponding arc input labels. Before the final determinization step of the index construction algorithm, there are much less common paths in the union of modified factor transducers compared the unions of other index structures. This leads to a smaller index (in a counter-intuitive way) since the lattice-like structure of the index is largely preserved after determinization. On the other hand, factor transducer and timed factor transducer turn into tree-like structures during determinization. The size difference between these two machines stems from the extra information stored in

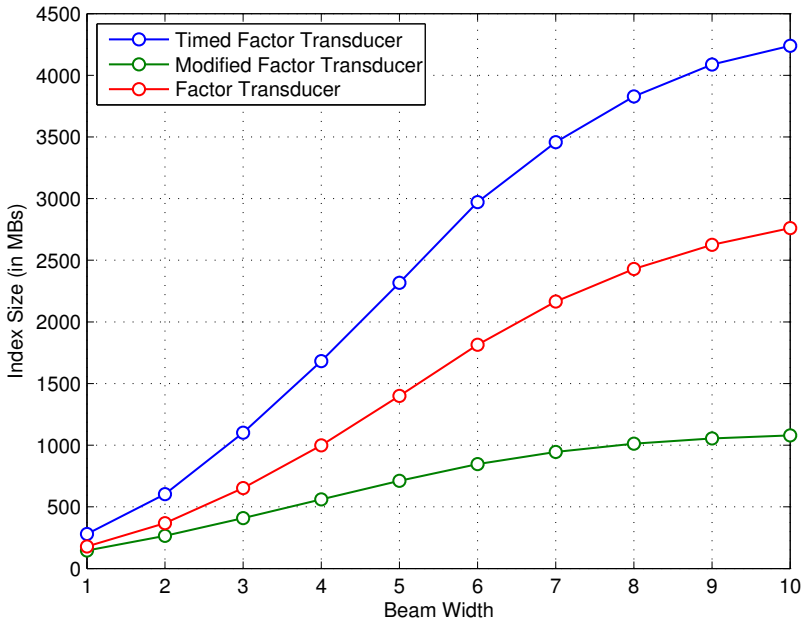


Figure 4.10. Index size vs. lattice beam (BUTBN-R Data Set)

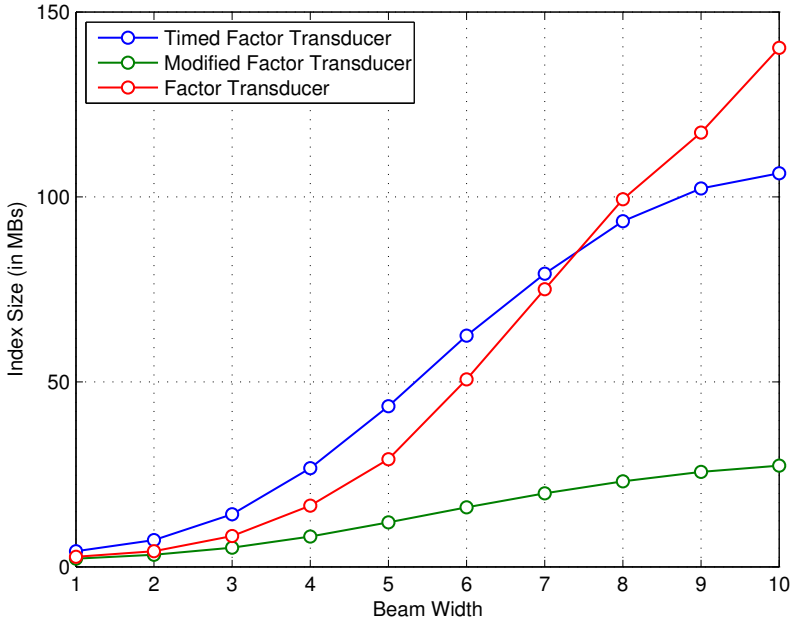


Figure 4.11. Index size vs. lattice beam (STDDEV06 Data Set)

the timed index - not from the number of states or the number of arcs which are similar for both structures. Factor transducers obtained from STDDEV06 data set exhibit a problematic behavior after a beam width of 5. This is probably due to a problem in the ASR lattices. Considering the ATWV performance given in Figure 4.8 along with the

index size, we can conclude that a beam width of 4 is ideal for actual system operation.

4.5.3. Search Time

Probably the most important concern in a retrieval application is the search time since search is performed online unlike indexation. Figure 4.12 and Figure 4.13 give the change of per result average search time as the lattice beam increases. Performing a search on the factor transducer is actually faster than the other schemes, however obtaining time alignment information online is a runtime expensive task with a complexity linear in the number of results. Due to the costly second stage, using the factor transducer in STD task results in two to three orders of magnitude slower search. The search time difference between the 2-stage retrieval and the others becomes larger as the beam width increases since the number of putative results increases along with the beam.

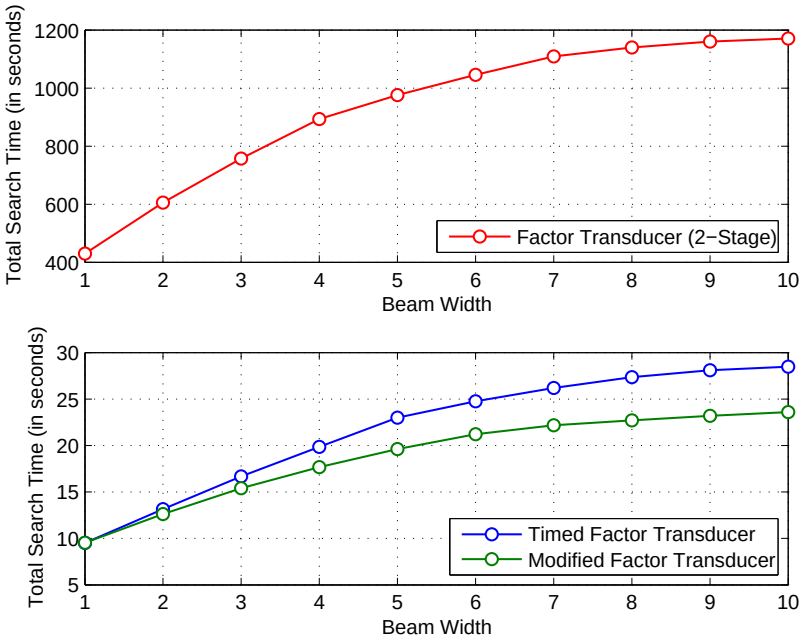


Figure 4.12. Total search time vs. lattice beam (BUTBN-R Data Set & R-IV Query Set)

Modified and timed factor transducers give similar search time performances as far as the whole query sets are concerned. Since our query sets are heavily populated with one-word query terms, this behavior is not surprising. While the timed factor

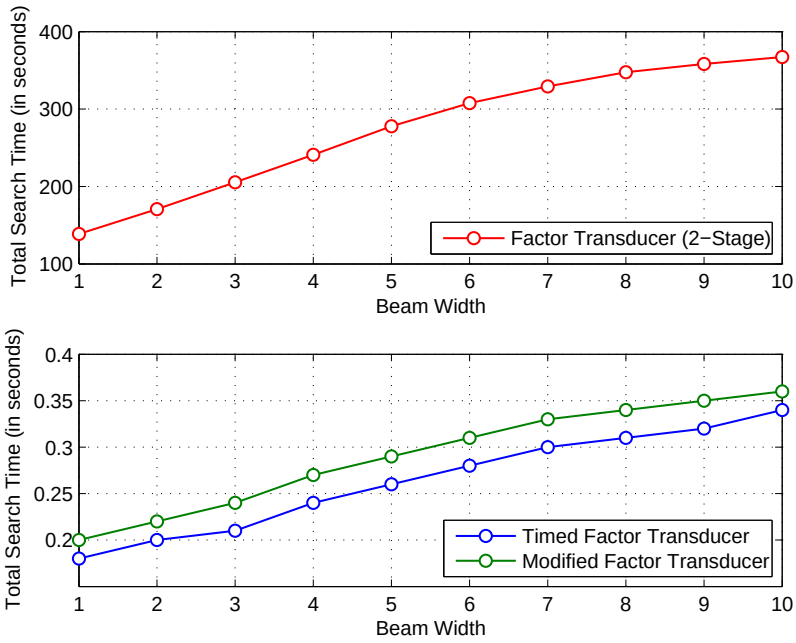


Figure 4.13. Total search time vs. lattice beam (STDDEV06 Data Set & DRYRUN06 Query Set)

transducer is "almost deterministic", this feature is not fully utilized when the queries are short. Even though the modified factor transducer is not deterministic, it has a much smaller size, hence we obtain comparable search times. On the other hand, we know that average search speed is linear in the sum of query length and the number of results for the timed factor transducer while it is linear in the product of query length and the number of results in the case of modified factor transducer. Figure 4.14 gives a decomposition of per query average search times over the BUTBN-R data set with respect to the query length.

First thing to notice about the plots in Figure 4.14 is that modified transducer is faster than the timed transducer only when the query is a single word. As the query length gets longer, timed index becomes much faster than the other due to its "almost deterministic" structure. In the modified index, every distinct time alignment of the query words leads to a new path to be traversed. Thus each path matching the beginning of the query has to be traversed until a mismatch is found to determine the successful paths matching the whole query. On the other hand, in the timed index, with a very high probability only one arc - hence only one path - needs to be traversed

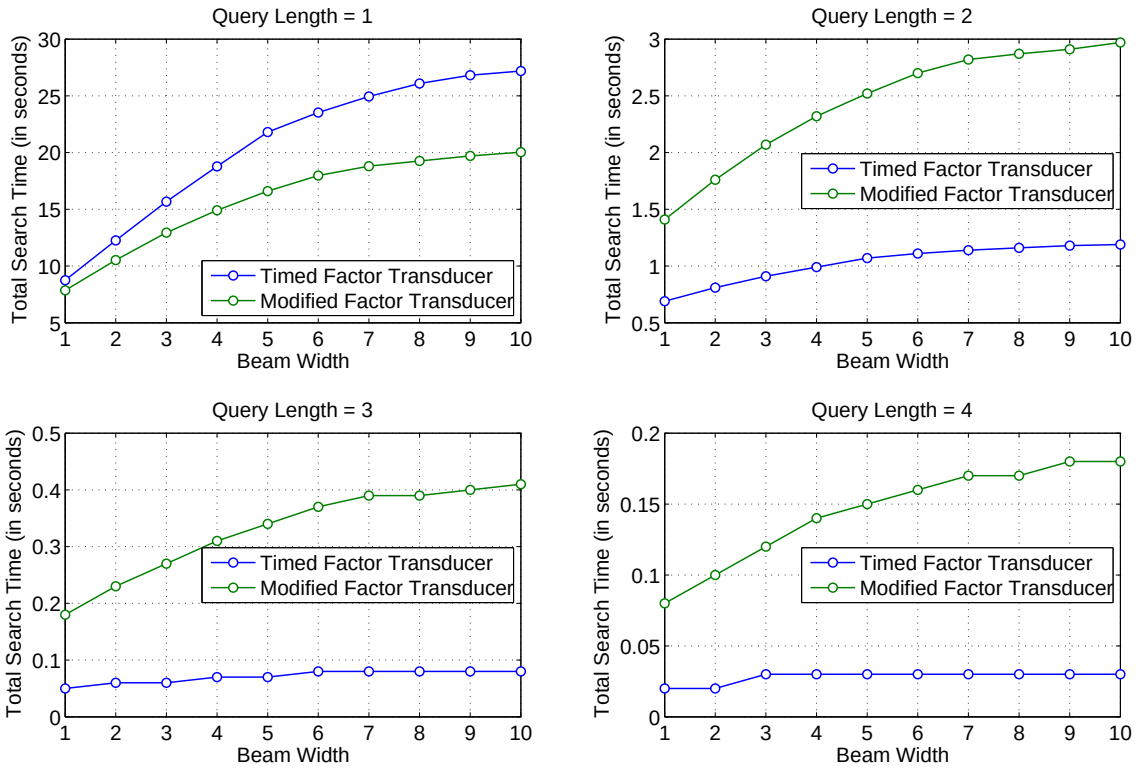


Figure 4.14. Decomposition of per query average search times with respect to query length (BUTBN-R Data Set & R-IV Query Set)

at each state until the single partial path matching the query is found. After that, results are read from the final transitions leaving the single destination state of that partial path.

Figure 4.15 gives the change of per result average search time as query length increases. This graph clearly demonstrates the search time advantage of timed factor transducer construct. As proposed in the search time analysis of Section 4.4.3, timed factor transducer has a search time complexity almost linear in the length of query. This linearity derives from the near deterministic nature of the index.

Experimental results given in this subsection somewhat underemphasize the search time performance of timed factor transducer since we are indexing word lattices and searching for rather short query terms. In the case of sub-word (morpheme, phone, etc.) indexing, average query term length significantly increases since each query word

Figure 4.15. Per result average search time vs. query length (BUTBN-R Data Set & R-IV Query Set)

is split into a string of sub-words for search. In such a scenario, a deterministic index would be much more crucial and timed factor transducer would dramatically outperform the modified one.

5. PHONETIC QUERY EXPANSION FOR OUT-OF-VOCABULARY QUERY RETRIEVAL

This chapter deals with the problem of performing phonetic search over a phonetic index to retrieve out-of-vocabulary STD queries. The work presented in this chapter was done during the 2008 CLSP Summer Workshop (WS08) and its aftermath and was published in [21, 38]. Even though I, myself, have performed the experiments and obtained the results mentioned in this chapter, the methodology and implementation is a collective effort which I cannot take full credit for. I would like to thank all members of the "Multilingual Spoken Term Detection: Finding and Testing New Pronunciations" group of WS08, especially Erica Cooper, Bhuvana Ramabhadran, Abhinav Sethy and of course my supervisor Murat Saraçlar. In the following sections, we describe the query expansion technique used for the retrieval of OOV terms and present the experimental results.

5.1. Query Forming and Expansion for Phonetic Search

When using a WFST-based phonetic index as in Chapter 4, the textual representation of a query needs to be converted into a phone sequence or more generally into a WFST representing the pronunciation of the query. For OOV queries, this conversion is achieved using a letter-to-sound (L2S) system. In this study, the L2S system is a WFST representing an n-gram model over (letter, phone) pairs that are obtained after an alignment step. Instead of simply taking the most likely output of the L2S system, we investigate using multiple pronunciations for each query.

Assume that we are searching for a letter string Q_l represented by an automaton. We find the n-best phone-strings $P_n(Q_l)$ by composing the automaton representing Q_l with the L2S transducer and taking n-best paths of the result. Then the posterior

probability of finding Q_l in lattice L within time interval $[t_s, t_e]$ can be written as

$$P(Q_l|L,[t_s,t_e]) = \sum_{Q_p \in P_n(Q_l)} \tilde{P}(Q_l|Q_p) P(Q_p|L,[t_s,t_e])$$

where $P(Q_p|L,[t_s,t_e])$ is the posterior probability supplied by the phonetic index and $\tilde{P}(Q_l|Q_p)$ is the posterior probability derived from L2S scores.

Composing an OOV query term with the L2S transducer returns a huge number of pronunciations. As demonstrated above, we retain only the n-best pronunciations for search to control the false alarm rates. To obtain the conditional probabilities $\tilde{P}(Q_l|Q_p)$, we perform a normalization operation on the retained pronunciations which can be expressed as

$$\tilde{P}(Q_l|Q_p) = \frac{P^\gamma(Q_l,Q_p)}{\sum_{Q_p \in P_n(Q_l)} P^\gamma(Q_l,Q_p)}$$

where $P(Q_l,Q_p)$ is the joint score supplied by the L2S transducer and γ is a scaling parameter.

Most of the time, retained pronunciations are such that a few dominate the rest in terms of likelihood scores, a situation which becomes even more pronounced as the query length increases. Thus, using raw L2S scores ($\gamma = 1$) is not a good idea since most of the time best pronunciation takes almost all of the posterior probability leaving the rest out of the picture.

A quick and dirty solution is to remove the pronunciation scores instead of scaling them. This corresponds to selecting $\gamma = 0$ which assigns the same posterior probability $\tilde{P}(Q_l|Q_p)$ to all pronunciations:

$$\tilde{P}(Q_l|Q_p) = \frac{1}{|P_n(Q_l)|}, \text{ for each } Q_p \in P_n(Q_l).$$

Unfortunately, this method is likely to boost false alarm rates since it does not make

any distinction among pronunciations. The challenge is to find a good query-adaptive scaling parameter which will dampen the large scale difference among L2S scores.

In our experiments we selected $\gamma = 1/|Q_l|$ which scales the log likelihood scores by dividing them with the “length of the letter string". This way, pronunciations for longer queries are dampened more. Another possibility is to select $\gamma = 1/|Q_p|$, which does the same with the “length of the phone string". Section 5.2.2 presents a comparison between removing pronunciation scores and scaling them with our method.

5.2. Experiments

Experiment given in this section use the MSTD data set and the MSTD-OOV query set. The gold standard experiments were conducted using the reference lexicon pronunciations for the query terms. The L2S system was trained using the reference pronunciations of words in the LVCSR vocabulary. This system was then used to generate multiple pronunciations for the OOV query words.

5.2.1. Reference Lexicon (Reflex) Pronunciations

For the gold standard experiments, we used the reference pronunciations to search for OOV queries in various indexes. The indexes were obtained from word and sub-word (fragment) based LVCSR systems. The output of the LVCSR systems were in the form of 1-best transcripts, consensus networks, and lattices. The results are presented in Table 5.1. Best performance in terms of Actual Term Weighted Value (See Section 2.2.1.2) is given by the phonetic index obtained from sub-word lattices.

5.2.2. L2S Pronunciations

For the L2S experiments, we investigated varying the number of pronunciations for each query for two scenarios and different indexes. The first scenario considered each pronunciation equally likely (unweighted pronunciations) whereas the second one made use of properly normalized L2S probabilities (weighted pronunciations). The results

Table 5.1. Gold Standard (Reflex) Results

Data Set	P_{FA} (%)	P_{Miss} (%)	ATWV
Word One-best	.001	77.0	.215
Word Consensus Nets	.002	68.7	.294
Word Lattices	.002	65.7	.322
Fragment One-best	.001	68.0	.306
Fragment Consensus Nets	.003	58.4	.390
Fragment Lattices	.003	48.5	.484

are presented in Figure 5.1 and summarized in Table 5.2. In the first scenario, the performance peaks at three pronunciations per query. This behavior can be explained by the boost in false alarms as a result of adding pronunciations that match words in ASR vocabulary. In the second scenario, adding more pronunciations does not degrade the performance – thanks to weighting – and a significant improvement is obtained. Best results of the second scenario are even better than the reflex results.

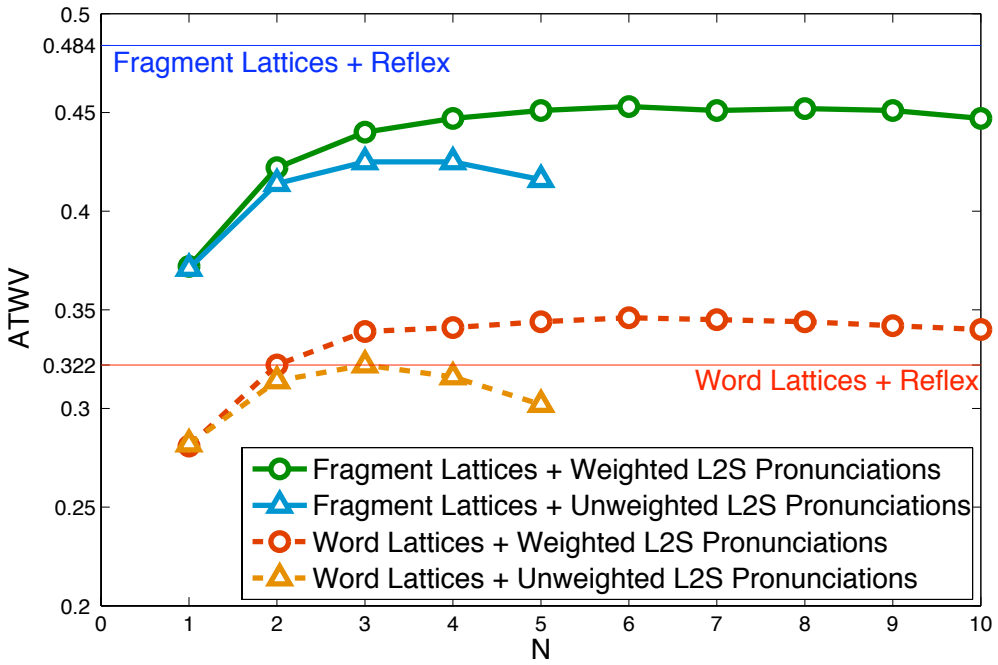


Figure 5.1. ATWV vs N-best L2S Pronunciations

Table 5.2. Best Performing N-best L2S Pronunciations

Data Set	L2S	Best	P_{FA} (%)	P_{Miss} (%)	ATWV
Word One-best	-	1	.001	79.6	.190
	Weighted	6	.004	73.0	.233
Word Lattices	-	1	.002	69.8	.281
	Unweighted	3	.005	62.5	.322
	Weighted	6	.005	60.6	.346
Fragment One-best	-	1	.001	75.7	.229
	Weighted	10	.005	66.2	.286
Fragment Lattices	-	1	.003	59.7	.372
	Unweighted	3	.006	51.2	.425
	Weighted	6	.006	48.7	.453

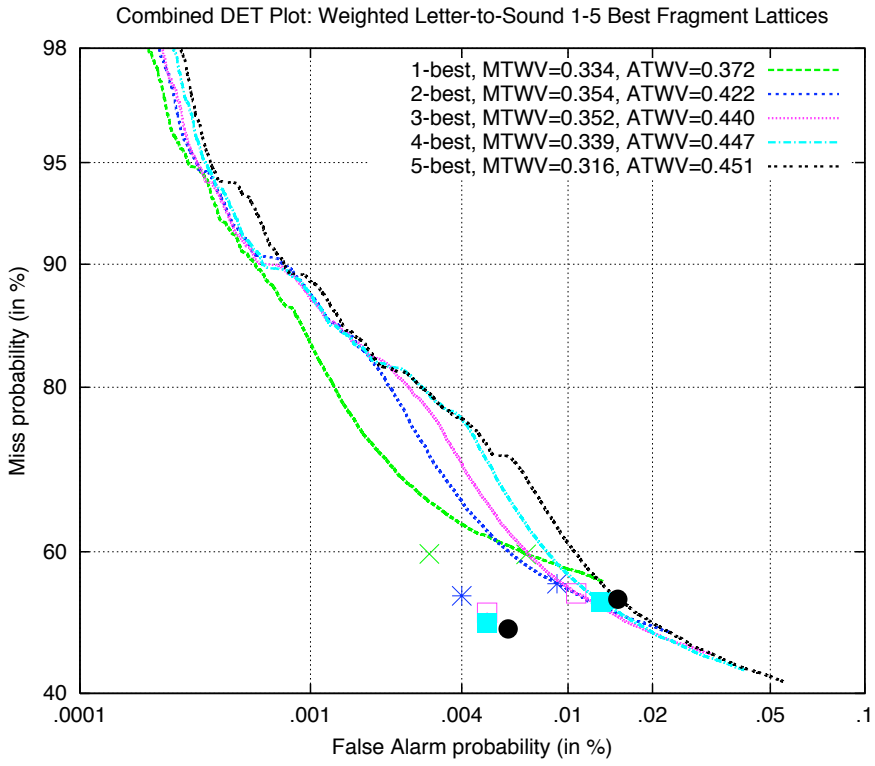


Figure 5.2. Combined DET plot for weighted L2S pronunciations

The combined DET plot drawn from the results of experiments utilizing weighted L2S pronunciations and the phonetic index obtained from fragment lattices, is presented in Figure 5.2. The single dots indicate MTWV (using a single global threshold) and ATWV (using term specific thresholds [16]) points.

6. EXPLOITING SCORE DISTRIBUTIONS FOR THRESHOLDING

Most STD systems, including ours, comprise of four consecutive stages:

(i) **Recognition**. For each speech utterance in the archive, a recognition lattice is generated using an LVCSR system.
(ii) **Indexation**. These lattices are indexed for efficient retrieval.
(iii) **Retrieval**. For each query $Q_k, k = 1, \ldots, N_Q$ presented to the system, a set of candidates $c_{k,n},\ n = 1, \ldots, N_k$ ranked by posterior probabilities $y_{k,n}$ are retrieved from the index.
(iv) **Decision**. These posterior probabilities are compared to a threshold *theta* to decide which candidates should be returned.

This chapter focuses on the decision stage of an STD system. We propose a term specific thresholding (TST) method, published in [48], that uses per query posterior score distributions to select candidates to be returned. In experiments, we use a word-level index obtained from ASR lattices and search for IV query terms. We show that our method is comparable to state-of-the-art TST methods.

6.1. Term Specific Thresholding for STD

STD systems return a relevance score for each match to the query term found during retrieval. These matches are candidate results out of which actual results are selected at the decision stage. The relevance scores are used for making a decision as to which candidates should be returned as well as ranking returned results. Various operating points can be obtained by changing the cost scheme decision algorithm works under.

Since STD looks for exact matches to the query, posterior score is usually the most effective relevance metric for decision. Most STD systems [16] employ simple

thresholding over the candidate posteriors to decide which candidates will be returned. More involved decision algorithms, like the neural network approach of [15], utilize various relevance metrics in a classification framework.

In the following, we propose a new TST technique for the STD task. We also review other well-known thresholding methods employed in STD and explain how our method fits into the picture.

6.1.1. Global Thresholding (GT)

The simplest thresholding strategy is to set a global threshold value θ for all query terms and accept candidates satisfying $y_{k,n} > \theta$. In this approach, each candidate retrieved for a query term is thresholded separately without taking other candidates into account. Different precision recall points are achieved by changing θ. In our setup θ is changed in $[0, 1]$ range since relevance scores are proper posterior probabilities. Choosing $\theta = 0$ accepts all candidates while $\theta = 1$ rejects them all. Since all candidates are processed separately, there is no term specific behavior in global thresholding. Hence global thresholding gives poor performance compared to term specific methods.

6.1.2. Term Weighted Value Based Term Specific Thresholding (TWV-TST)

A recent study [16] proposed a new detection approach which determines term specific thresholds that optimize the STD evaluation metric known as Term-Weighted Value (TWV). NIST defines the TWV metric as:

$$\mathrm{TWV}(\theta) = 1 - \frac{1}{N_Q} \sum_{k=1}^{N_Q} \{P_{\mathrm{Miss}}(Q_k, \theta) + \beta . P_{\mathrm{FA}}(Q_k, \theta)\}$$

where θ is the detection threshold, β is a user defined parameter to adjust the trade-off between miss and false alarm costs. P_{Miss} and P_{FA} are calculated as:

$$\mathrm{P}_{\text{Miss}}(Q_k,\theta) = 1 - \frac{C(Q_k,\theta)}{R(Q_k)}, \qquad \mathrm{P}_{\text{FA}}(\mathrm{Q_k},\theta) = \frac{A(Q_k,\theta) - C(Q_k,\theta)}{T_{\text{Speech}} - C(Q_k,\theta)}$$

where T_{Speech} is the total amount of speech in the database.

In this approach, all candidates retrieved for a query term Q_k are pooled and $\hat{R}(Q_k)$, expected count of that query term in the archive, is estimated by summing candidate scores. Then for each query term, an optimal threshold is determined using the relations

$$\hat{V}_{\text{Hit}}(Q_k) = \frac{1}{\hat{R}(Q_k)}, \quad \hat{C}_{\text{FA}}(Q_k) = \frac{\beta}{T_{\text{Speech}} - \hat{R}(Q_k)}, \quad \hat{\theta}(Q_k) = \frac{\hat{C}_{\text{FA}}(Q_k)}{\hat{C}_{\text{FA}}(Q_k) + \hat{V}_{\text{hit}}(Q_k)},$$

where $\hat{V}_{\text{Hit}}(Q_k)$ is the expected value of a hit, $\hat{C}_{\text{FA}}(Q_k)$ is the expected cost of a false alarm, and $\hat{\theta}(Q_k)$ is the optimal threshold maximizing TWV in the expected sense.

In the given setup, different precision recall points are achieved by changing β, i.e. increasing β gives a larger false alarm cost and hence the threshold moves up. Note that the change of β moves $\hat{\theta}(Q_k)$ in the interval $[0, 1]$. In TWV-TST, sum of candidate scores gives a sufficient statistic for determining the threshold. In the next section, we incorporate other statistics of candidate scores into the decision.

6.1.3. Score Distribution Based Term Specific Thresholding (SD-TST)

6.1.3.1. Decision Framework. Retrieval stage of an STD system provides a collection of scored candidates that belong to two classes: false alarms and hits. Selection of the candidates to be returned can be cast as a simple hypothesis-testing problem given that we have:

- two probability distributions P_0 and P_1 giving the score distributions of false alarms and hits respectively,

- prior probabilities π_0 and π_1 of having a candidate from each class,
- and the costs C_{ij} incurred by choosing hypothesis H_i when H_j is true.

In this setup, we assume that a candidate score y is a realization of a random variable Y and that there are two possible hypotheses

$$\begin{aligned} H_0 : &\quad Y \sim P_0 \\ H_1 : &\quad Y \sim P_1 \end{aligned}$$

where the notation "$Y \sim P$" denotes the condition "Y has distribution P". Finally, the decision is rule δ is a function on $[0, 1]$ given by

$$\delta(y) = \begin{cases} 1 & \text{if } y > \theta \\ 0 & \text{if } y \leq \theta \end{cases}$$

so that the value of δ for a given $y \in [0, 1]$ is the index of the hypothesis accepted by δ.

Our problem is now transformed into choosing θ so that δ is a Bayes rule. With that in mind, we assign the following costs to our decisions:

$$C = \begin{bmatrix} 0 & 1 \\ \alpha & 0 \end{bmatrix}$$

where α is a user defined parameter specifying the cost of false alarms relative to hits. We can now define the conditional risk $R_j(\delta)$ for each hypothesis as the expected cost incurred by decision rule δ when that hypothesis is true:

$$\begin{aligned} R_0(\delta) &= \alpha P_0(y > \theta), \\ R_1(\delta) &= P_1(y \leq \theta), \end{aligned}$$

For given priors, the Bayes risk incurred by decision rule δ is given by

$$\begin{aligned} r(\delta) &= \pi_0 R_0(\delta) + \pi_1 R_1(\delta) \\ &= \pi_0 \alpha P_0(y > \theta) + \pi_1 P_1(y \leq \theta). \end{aligned}$$

Now the Bayes rule for H_0 versus H_1 is the one that minimizes, over all decision rules, the Bayes risk. We can rewrite the Bayes risk as

$$r(\delta) = \pi_1 + \pi_0 \alpha P_0(y > \theta) - \pi_1 P_1(y > \theta).$$

using the fact that $P_1(y > \theta) = 1 - P_1(y \leq \theta)$. Assuming that P_j has density p_j for $j = 0, 1$, we can write

$$r(\delta) = \pi_1 + \int_\theta^1 [\pi_0 \alpha p_0(y) - \pi_1 p_1(y)] dy.$$

$r(\delta)$ is a minimum over all $(\theta, 1]$ if we choose the acceptance region as

$$\begin{aligned} (\theta, 1] &= \{y \in [0,1] \mid \pi_0 \alpha p_0(y) - \pi_1 p_1(y) \leq 0\} \\ &= \left\{ y \in [0,1] \;\middle|\; \frac{p_0(y)}{p_1(y)} \leq \frac{\pi_1}{\pi_0 \alpha} \right\}. \end{aligned}$$

This acceptance region can be reformulated as

$$(\theta, 1] = \{y \in [0,1] \mid L(y) > \tau\} \tag{6.1}$$

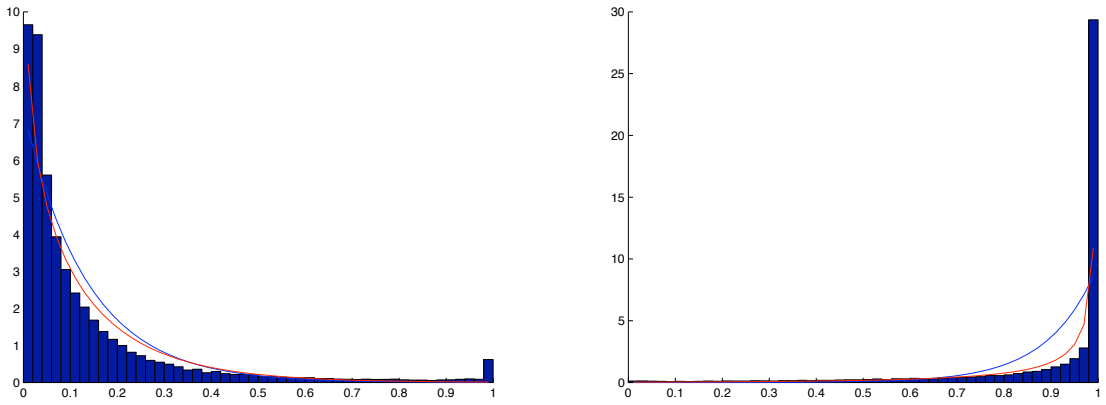
where

$$L(y) = \frac{p_1(y)}{p_0(y)}, \quad \tau = \frac{\pi_0 \alpha}{\pi_1}. \tag{6.2}$$

$L(y)$ is the familiar likelihood-ratio statistic between H_0 and H_1.

6.1.3.2. Class Distributions. We developed the Bayes optimal decision rule in the previous section. Now what remains is to find parametric models for each class distribution. This section tells the story of how we found them.

Histograms give a good idea about distributions given you have enough data. With that in mind, we pooled all candidates retrieved for all query terms, labeled each one as a hit or a false alarm, and derived a normalized histogram for each class. These histograms are given in Figure 6.1. As evident from the graphs, both classes follow exponential-like distributions. To support our intuition, we then estimated the parameters of exponential and gamma distributions using the class pools. These distributions overlay the class histograms in Figure 6.1. A closer look at these graphs tells us that parametric forms very well fit the distribution of false alarms. Unfortunately, this is not true for hits.



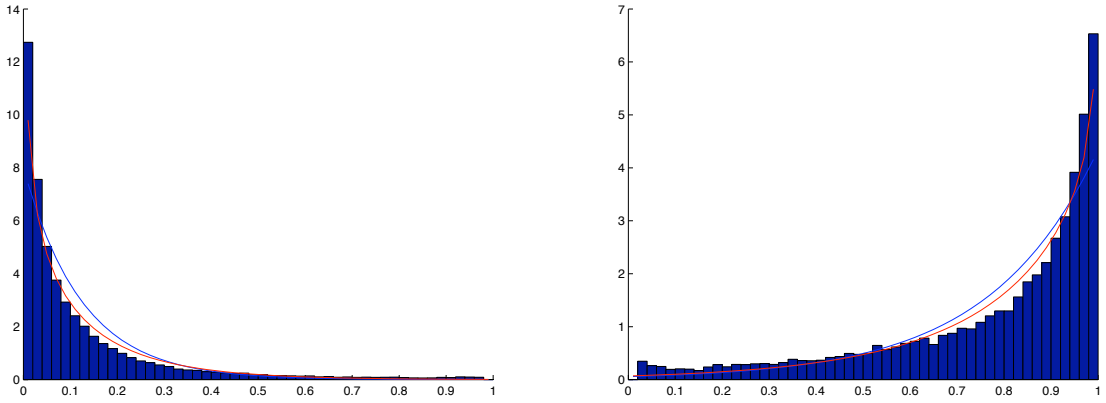
(a) (b)

Figure 6.1. Normalized posterior score histograms of (a) false alarms and (b) hits for the pool of all query terms (exponential (blue) and gamma (red) distribution estimates overlaid)

There is a curious jump in the distribution of hits near the right end, thus parametric models do not fit the distribution. When we investigated why there is such a jump in the distribution of hits while it is all fine with false alarms, we noticed the asymmetry of our retrieval system. Due to the nature of the index, we never retrieve a candidate with a posterior score "0". On the other hand, we receive a good deal of

candidates with posterior score ‘1’, hence the jump.

Solving the mystery of hits, we developed a new idea to fit parametric models to the data: “leave out all candidates with extreme scores (≈ 0 or ≈ 1)". Then we repeated the experiment described in Figure 6.1 with the new set of candidates. Results are given in Figure 6.2. This time parametric forms very well fit the distributions of both classes.



(a) (b)

Figure 6.2. Normalized posterior score histograms of (a) false alarms and (b) hits for the pool of all query terms when extreme scores are removed (exponential (blue) and gamma (red) distribution estimates overlaid)

There is another problem though. The fact that a parametric form fits the distribution of all candidates in a class pool does not necessarily mean that the same form fits the distribution of candidates in a class retrieved for a single query term. Figure 6.3 gives the posterior score histograms of each class for an example query term. These graphs corroborate the observation on pools that both classes follow exponential-like distributions.

Note that the gamma form gives a better fit compared to the exponential form. However, estimates of gamma distribution parameters do not have closed form expressions like the exponential distribution. In this work, we decided to utilize the exponential form to model the distribution of each class. Parametric forms for the

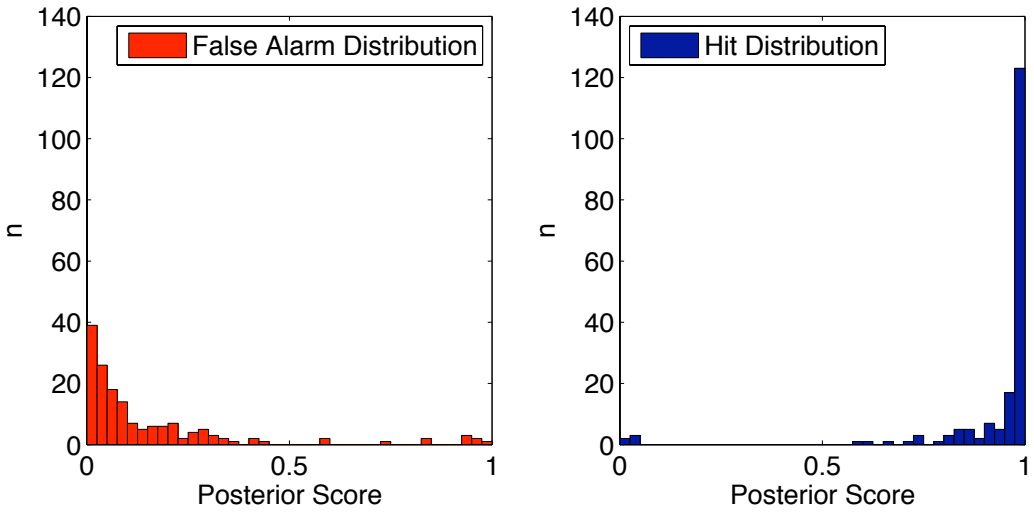


Figure 6.3. The posterior score histograms of both classes for an example query term.

densities p_0 and p_1 of the previous section are thus given as:

$$p_0(y) = \lambda_0 e^{-\lambda_0 y}, \tag{6.3}$$

$$p_1(y) = \lambda_1 e^{-\lambda_1 (1-y)}. \tag{6.4}$$

6.1.3.3. Parameter Estimation. Now that we have parametric forms for each class, our problem reduces to estimating these parameters λ_j for $j = 0, 1$, along with the prior probabilities π_j for $j = 0, 1$. However, this is not a straightforward task. When the system returns candidates for a query term, we do not know which one belongs to which class – we would not have a decision stage if we had that knowledge –, so we use a mixture p of the two exponentials p_0 and p_1 to model the distribution of all candidate scores retrieved for a query term:

$$p(y) = \pi_0 p_0(y) + \pi_1 p_1(y).$$

where π_0 and π_1 are mixture weights. Given the candidate scores ($y_{k,n}$ for $n = 1, \ldots, N_k$) of a query term Q_k, the exponential mixture model (EMM) parameters ($\lambda_0(Q_k)$, $\lambda_1(Q_k)$ and $\pi_0(Q_k)$) are estimated in an unsupervised manner using the Expectation-Maximization (EM) algorithm. Each iteration consists of first comput-

ing the posterior of each class

$$P(j|y_{k,n}) = \frac{\hat{\pi}_j(Q_k)p_j(y_{k,n})}{p(y_{k,n})} \quad j = 1, 2, \quad n = 1, \ldots, N_k$$

and then updating

$$\begin{aligned}
\hat{\lambda}_0(Q_k) &= \frac{\sum_n P(0|y_{k,n})}{\sum_n P(0|y_{k,n})y_{k,n}}, \\
\hat{\lambda}_1(Q_k) &= \frac{\sum_n P(1|y_{k,n})}{\sum_n P(1|y_{k,n})(1-y_{k,n})}, \\
\hat{\pi}_j(Q_k) &= \frac{1}{N_k}\sum_n P(j|y_{k,n}).
\end{aligned}$$

After a sufficient number of iterations (typically two or three), we assume that each mixture component represents a class and mixture weights correspond to class priors. Figure 6.4 shows the normalized histogram of posterior scores and the EM estimate given by our method for an example query.

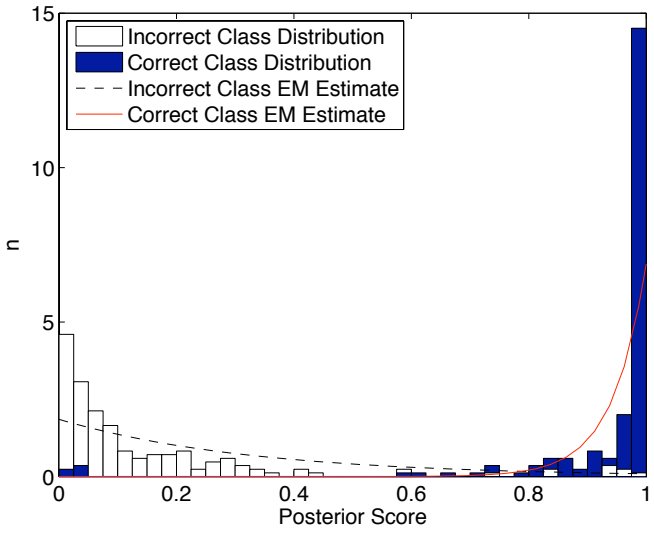


Figure 6.4. The normalized histogram of posterior scores and the EM estimates for false alarms and hits for an example query.

6.1.3.4. Bayes Optimal Threshold. In the light of Equations 6.2, 6.3 and 6.4:

$$L(y) = \frac{\lambda_1 e^{-\lambda_1(1-y)}}{\lambda_0 e^{-\lambda_0 y}} = \frac{\lambda_1}{\lambda_0} e^{-\lambda_1 + (\lambda_0+\lambda_1)y}. \tag{6.5}$$

Then using Equation 6.1, 6.2 and 6.5, the acceptance region is given as

$$\begin{aligned}(\theta, 1] &= \left\{ y \in [0,1] \mid \frac{\lambda_1}{\lambda_0} e^{-\lambda_1 + (\lambda_0 + \lambda_1) y} > \frac{\pi_0 \alpha}{\pi_1} \right\} \\ &= \left\{ y \in [0,1] \mid y > \frac{\lambda_1 + \log(\lambda_0/\lambda_1) + \log(\pi_0/\pi_1) + \log \alpha}{\lambda_0 + \lambda_1} \right\}.\end{aligned}$$

Hence, in the given setting, Bayes optimal term specific threshold values are given as

$$\hat{\theta}(Q_k) = \frac{\hat{\lambda}_1(Q_k) + \log(\hat{\lambda}_0(Q_k)/\hat{\lambda}_1(Q_k)) + \log(\hat{\pi}_0(Q_k)/\hat{\pi}_1(Q_k)) + \log \alpha}{\hat{\lambda}_0(Q_k) + \hat{\lambda}_1(Q_k)}, \quad k = 1, \ldots, Q. \tag{6.6}$$

In the proposed method α plays the same role β does in TWV-TST, and allows us to control the decision threshold for different cost settings. Note that this method utilizes the likelihood ratio statistic along with class priors to make a Bayes optimal decision under the cost setting determined by α. Hence, as long as we have reasonable estimates, this method is likely to beat other approaches.

6.2. Experiments

In this section, we give the detection results on BUTBN-R and BUTBN-HI data sets. We provide experiments comparing Global Thresholding, Term Weighted Value Based Term Specific Thresholding and Score Distribution Based Term Specific Thresholding methods.

6.2.1. Global Thresholding

This subsection presents the results of global thresholding on both data sets. We used NIST's STDEval suite to obtain the Detection Error Tradeoff (DET) curves given in the following. Figure 6.5 and Figure 6.6 give the DET curves evaluating the results on BUTBN-R and BUTBN-HI data sets obtained by using the timed factor transducer and a lattice beam of four. The points marked on the DET curves indicate the operating points maximizing the TWV metric.

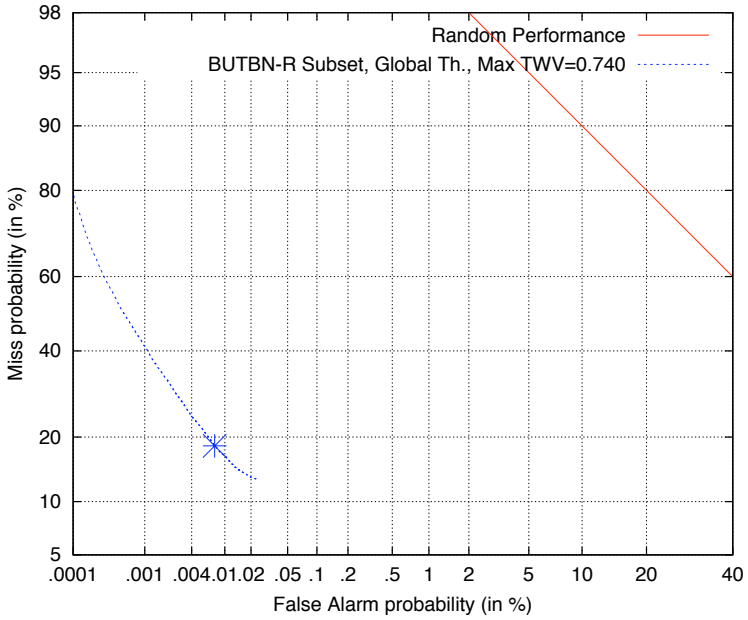


Figure 6.5. DET curve (BUTBN-R set, timed factor transducer, beam = 4)

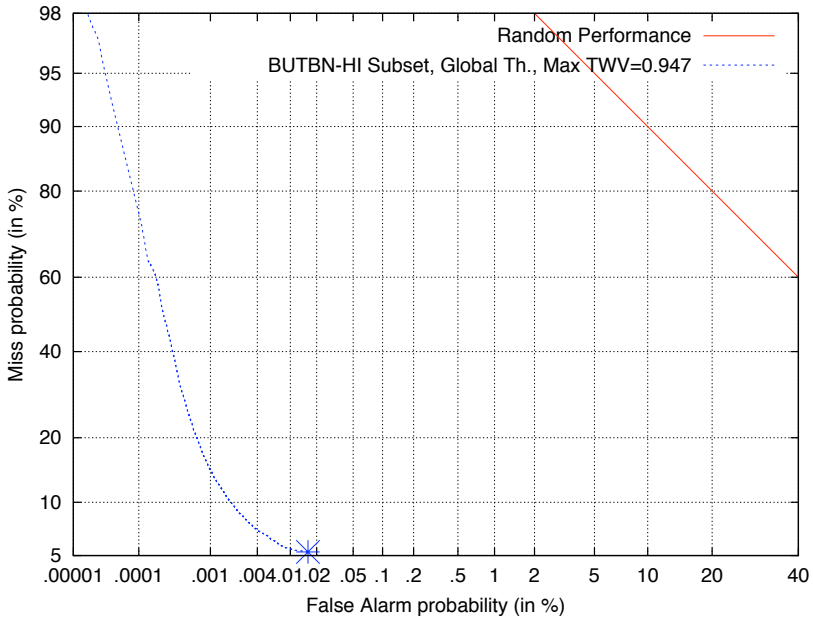


Figure 6.6. DET curve (BUTBN-HI set, timed factor transducer, beam = 4)

6.2.2. Term Specific Thresholding

This section compares the TST methods. We provide Precision Recall (PR) and Receiver Operating Characteristics (ROC) curves for comparison. Plots also include GT results as a baseline. Figure 6.7 and Figure 6.8 respectively give the PR curves evaluating the results on the BUTBN-R and BUTBN-HI data sets obtained by using the timed factor transducer and a lattice beam of 4.

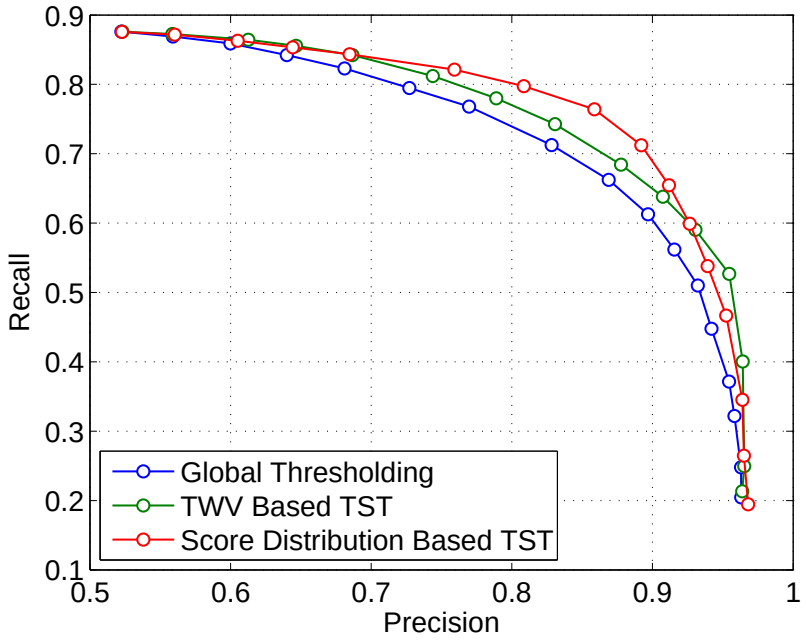


Figure 6.7. PR curves (BUTBN-R set, timed factor transducer, beam = 4)

Figure 6.9 and Figure 6.10 respectively give the ROC curves evaluating the results on BUTBN-R and BUTBN-HI data sets obtained by using the timed factor transducer and a lattice beam of four.

As evident from the figures in this subsection, TST methods outperform GT method at all times. Looking at the PR curves, we see that SD-TST curve stays above the TWV-TST curve over a large interval of precision values for both data sets. On the contrary, ROC curves demonstrate that TWV-TST outperforms SD-TST over the entire region of interest. This rather counter-intuitive result can be explained by recalling the objectives of both methods. TWV-TST aims to maximize the TWV metric given in Section 2.2.1.2. This metric is directly determined from the P_{FA} and

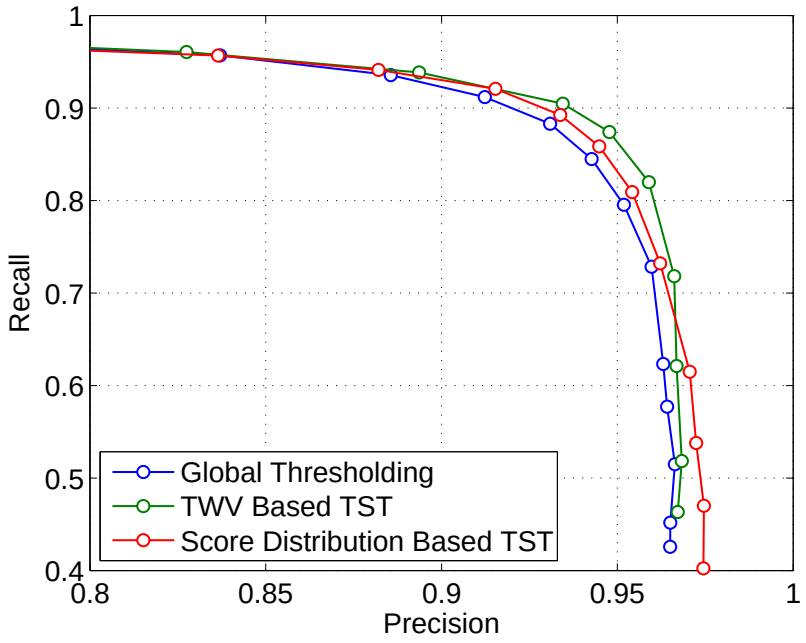


Figure 6.8. PR curves (BUTBN-HI set, timed factor transducer, beam = 4)

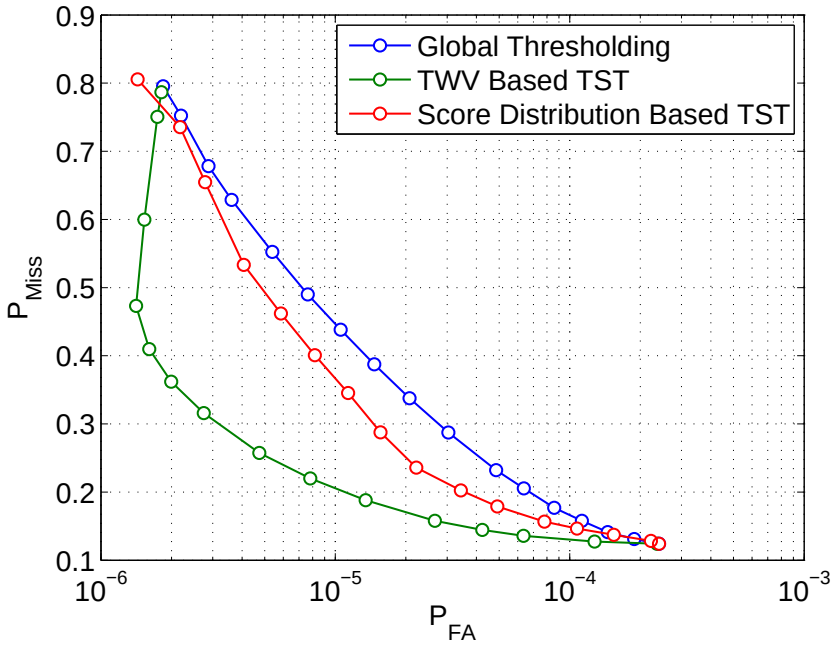


Figure 6.9. ROC curves (BUTBN-R set, timed factor transducer, beam = 4)

P_{Miss} figures achieved by the system and the β parameter. TWV-TST method first estimates the count of matches to each query and then calculates the optimal term specific threshold that will provide the best P_{FA}-P_{Miss} combination. Since this method directly takes into account the P_{FA} and P_{Miss} figures, it is quite plausible that it gives

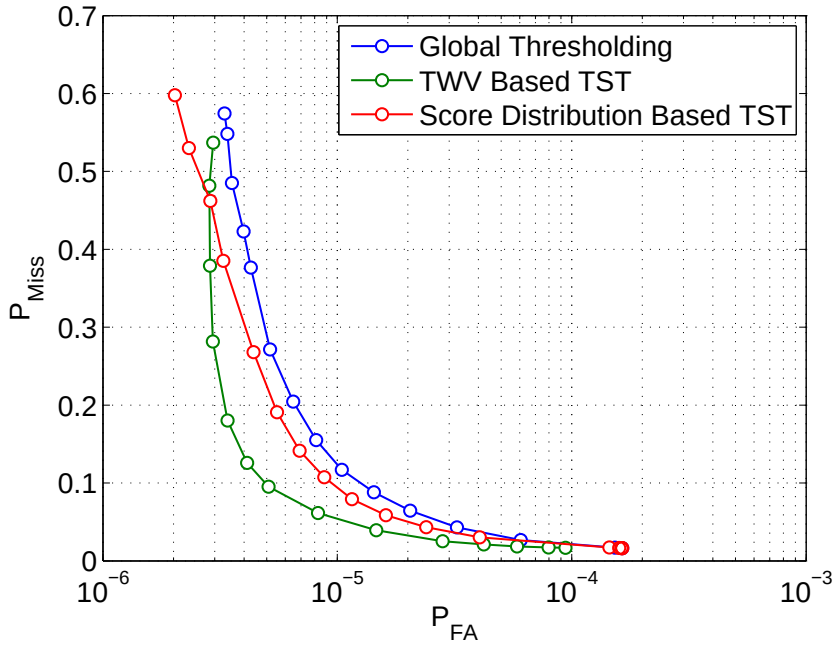


Figure 6.10. ROC curves (BUTBN-HI set, timed factor transducer, beam = 4)

the best performance under the ROC curve. On the other hand, SD-TST tries to find the optimal term specific threshold which separates the putative results in the best way possible. This is not directly related to minimizing P_{FA} as defined in Section 2.2.1.2. A good separation between putative term results does not necessarily coincide with the optimal false alarm rate. On the other hand, SD-TST method is directly related to the Precision and Recall figures since Precision is the measure of how good the separation is. Long story short, both TST methods have distinct advantages, their relative value depends on the particular application at hand.

SD-TST performs better on the BUTBN-R data set. There are two factors contributing to the difference between the sets. First, BUTBN-HI set is quite small compared to BUTBN-R, thus SD-TST estimates are quite poor for a large fraction of queries due to data sparsity issues. Second, BUTBN-HI is a fairly clean set with low WERs and high discrimination between recognition alternatives. Consequently, there is not as much improvement opportunity as the BUTBN-R set. Still, the SD-TST performance on BUTBN-HI set is comparable to the TWV-TST performance over the entire region of interest. As a matter of fact, SD-TST performs better in the high precision region which might be valuable for some STD applications like the sign dictionary [48].

7. CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis, we addressed the challenges of the STD task and developed novel techniques to deal with the indexation, OOV retrieval and detection problems. We developed four different STD systems - two of them in Turkish and the rest in English - to evaluate the effectiveness of our proposed methods. Following sections summarize our conclusions and give future directions for research.

7.1. Indexation and Search

Indexing ASR lattices (word or sub-word level) for STD is not a straightforward task. Over the last two years, we developed three different strategies to handle this problem. Our first approach was to employ a two-stage retrieval scheme where STD is achieved by first performing SUR and then aligning the query with the retrieved utterance lattices. Later, we developed single-stage schemes which modified the SUR indexation method by augmenting the index with timing information. As demonstrated by the comparisons given in Chapter 4, single-stage methods significantly improve the search-time over the two-stage method. We further analyzed the differences between the single-stage methods - both theoretically and experimentally - and showed that timed factor transducer significantly outperforms modified factor transducer as the query length increases. This fact becomes even more valuable in the case of sub-word indexing since sub-word query strings are usually much longer.

Since the timed factor transducer inherently stores proximity information (by means of time stamps), it can be utilized in other SR applications like SDR. Furthermore, since the query can be any weighted automaton, we can search for complex relations between query words without changing the index. Any finite state relation, e.g. a regular expression, can be compiled into a query automaton and retrieved from the index. We gave an example to this type of search in Chapter 5 where we compiled weighted pronunciation alternatives into a query automaton to search the index for the OOV term occurrences. Another possibility is to relax the exact string match objec-

tive and allow for gaps between query terms. Although this is not expected in STD, it might be useful in other speech retrieval applications. Implementing such a search is trivial in our framework since we can easily modify the query automaton in such a way that an arbitrary number of words can be inserted between actual query terms. Searching for arbitrary permutations of query words, even allowing the insertion of other words in between these permutations, is yet another trivial extension which can be achieved without changing the index.

In this study, we have not looked into reducing the information stored inside the index. The indexes we built for STD store all substrings seen in the lattices. We can significantly reduce the index size by applying factor selection methods if there are certain constraints on the queries that will be submitted to the system. A typical example is the length restriction filter which omits factors longer than a threshold value during the indexation. Another method is to apply pruning on the index to remove unlikely paths.

7.2. OOV Retrieval

In Chapter 5, we developed a phonetic query expansion technique to retrieve out-of-vocabulary (OOV) query words and showed that it can be as effective as knowing the actual OOV pronunciations. We performed experiments with various phonetic indexes generated from words and sub-words using lattices and confusion networks. Our experiments showed that sub-words indexes perform much better in the case of OOV queries. We also observed that lattice indexes outperform confusion network indexes as far as retrieval performance is concerned. L2S experiments showed that using multiple pronunciations significantly improve OOV retrieval, particularly when the alternatives are properly weighted.

The biggest problem of the L2S method is the boost in false alarm rates. We suppressed the false alarms by weighting alternative pronunciations with properly scaled L2S likelihoods. Another possibility is to remove pronunciations matching the words in the system vocabulary. It is also possible to use a language model or other contex-

tual information to detect false alarms. An example is to use web resources to build contextual information about the OOV word at hand. Using a web search engine, we might derive a simple language model representing the OOV context and classify our search results with this model.

7.3. Detection

In Chapter 6, we proposed a term specific thresholding technique which utilizes score statistics to find an optimal threshold value for each query and showed that it outperforms TWV-TST in terms of precision-recall performance over a wide range of precision values. This is an expected result since our detection setup is directly related to the precision-recall figures. Our method is inferior to TWV-TST as far as the DET (or ROC) curves are concerned. Again, this is an expected result since our method does not try and optimize the false alarm rate defined in Section 2.2.1.2 like the TWV-TST method.

Our current implementation of the proposed method does not make use of any training data to estimate the initial parameters for the EM algorithm. Instead, it relies on some loose assumptions about the initial parameters of the likelihood functions and uses uninformative prior distributions. Furthermore, our assumptions about the parametric form of the likelihood function may not be valid at all times. Maximizing the likelihood with mismatched models degrades the performance even when the initial parameters are close to the optimal values. We have plans to utilize other parametric forms to better model the posterior score distributions. Another problem of the proposed method is the lack of sufficient data for robust parameter estimation. Maximum likelihood estimation is prone to overtraining with sparse data. Bayesian methods can be used to introduce priors on the model parameters in order to make the estimation step more robust.

REFERENCES

1. Rabiner, L. and B. H. Juang, *Fundamentals of speech recognition*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1993.

2. Jelinek, F., *Statistical methods for speech recognition*, MIT Press, Cambridge, MA, 1997.

3. Mangu, L., E. Brill, and A. Stolcke, "Finding consensus in speech recognition: word error minimization and other applications of confusion networks", *Computer Speech & Language*, Vol. 14, No. 4, pp. 373–400, 2000.

4. Whittaker, E. W. D., J. M. Van Thong, and P. J. Moreno, "Vocabulary independent speech recognition using particles", *IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 315–318, 2001.

5. Baeza-Yates, R. A. and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.

6. Jurafsky, D. and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice Hall, Englewood Cliffs, New Jersey, 2000.

7. Ponte, J. M. and W. B. Croft, "A language modeling approach to information retrieval", *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 275–281, ACM, New York, NY, 1998.

8. Can, F., S. Koçberber, E. Balçık, C. Kaynak, and H. C. Öcalan, "Information Retrieval on Turkish Texts", *Journal of the American Society for Information Science and Technology*, Vol. 59, No. 3, pp. 407–421, February 2008.

9. Mølgaard, L. L., K. W. Jørgensen, and L. K. Hansen, "Castsearch - Context Based Spoken Document Retrieval", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 4, pp. 93–96, April 2007.

10. Manning, C. and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA, 1999.

11. NIST, "The Spoken Term Detection (STD) 2006 Evaluation Plan", 2006, `http://www.itl.nist.gov/iad/mig/tests/std/`.

12. Makhoul, J., F. Kubala, T. Leek, D. Liu, L. Nguyen, R. Schwartz, and A. Srivastava, "Speech and language technologies for audio indexing and retrieval", *Proceedings of the IEEE*, Vol. 88, No. 8, pp. 1338–1353, Aug 2000.

13. Hansen, J. H. L., R. Huang, B. Zhou, M. S. Seadle, J. R. Deller, A. Gurijala, M. Kurimo, and P. Angkititrakul, "SpeechFind: Advances in Spoken Document Retrieval for a National Gallery of the Spoken Word", *IEEE Transactions on Speech and Audio Processing*, Vol. 13, No. 5-1, pp. 712–730, 2005.

14. "The National Gallery of the Spoken Word", `http://www.ngsw.org`.

15. Vergyri, D., I. Shafran, A. Stolcke, R. R. Gadde, M. Akbacak, B. Roark, and W. Wang, "The SRI/OGI 2006 Spoken Term Detection System", *Proceedings of the Interspeech/Eurospeech*, pp. 2393–2396, August 2007.

16. Miller, D. R. H., M. Kleber, C. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and Accurate Spoken Term Detection", *Proceedings of the Interspeech/Eurospeech*, pp. 314–317, August 2007.

17. Chelba, C. and A. Acero, "Position specific posterior lattices for indexing speech", *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 443–450, Association for Computational Linguistics, Morristown, NJ, USA, 2005.

18. Saraclar, M. and R. Sproat, "Lattice-Based Search for Spoken Utterance Retrieval", *Proceedings of Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 129–136, Association for Computational Linguistics, Morristown, NJ, May 2004.

19. Allauzen, C., M. Mohri, and M. Saraclar, "General Indexation of Weighted Automata: Application to Spoken Utterance Retrieval", *Proceedings of Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 33–40, Association for Computational Linguistics, Morristown, NJ, May 2004.

20. Parlak, S. and M. Saraclar, "Spoken term detection for Turkish Broadcast News", *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5244–5247, April 2008.

21. Can, D., E. Cooper, A. Sethy, C. White, B. Ramabhadran, and M. Saraclar, "Effect of pronunciations on OOV queries in spoken term detection", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 3957–3960, IEEE Computer Society, Los Alamitos, CA, 2009.

22. Hori, T., I. Hetherington, T. Hazen, and J. R. Glass, "Open-Vocabulary Spoken Utterance Retrieval Using Confusion Networks", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 73–76, April 2007.

23. Pan, Y. C., H. L. Chang, and L. S. Lee, "Robustness analysis on lattice-based speech indexing approaches with respect to varying recognition accuracies by refined simulations", *IEEE Spoken Language Technology Workshop*, pp. 289–292, Dec. 2008.

24. Logan, B. and J. V. Thong, "Confusion-based query expansion for OOV words in Spoken Document Retreival", *Proceedings of the 7th International Conference on Spoken Language Processing*, pp. 1997–2000, 2002.

25. Thong, J. M. V., P. Moreno, B. Logan, B. Fidler, K. Maffey, and M. Moores, "Speechbot: an experimental speech-based search engine formultimedia content on the web", *IEEE Transactions on Multimedia*, Vol. 4, No. 1, pp. 88–96, March 2002.

26. Logan, B., P. Moreno, and O. Deshmukh, "Word and sub-word indexing approaches for reducing the effects of OOV queries on spoken audio", *Proceedings of the second international conference on Human Language Technology Research*, pp. 31–35, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2002.

27. Garofolo, J. S., C. G. P. Auzanne, and E. M. Voorhees, "The TREC Spoken Document Retrieval Track: A Success Story", *Proceedings of the Text Retrieval Conference (TREC) 8*, pp. 107–130, 2000.

28. Siohan, O. and M. Bacchiani, "Fast vocabulary independent audio search using path based graph indexing", *Proceedings of Interspeech/Eurospeech*, pp. 53–56, 2005.

29. Mamou, J., B. Ramabhadran, and O. Siohan, "Vocabulary independent spoken term detection", *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 615–622, ACM, New York, NY, 2007.

30. Chaudhari, U. and M. Picheny, "Improvements in Phone Based Audio Search via Constrained Match with High Order Confusion Estimates", *IEEE Workshop on Automatic Speech Recognition Understanding*, pp. 665–670, Dec. 2007.

31. Li, Y. C., W. K. Lo, H. M. Meng, and P. C. Ching, "Query expansion using phonetic confusions for Chinese spoken document retrieval", *IRAL '00: Proceedings of the fifth international workshop on on Information retrieval with Asian languages*, pp. 89–93, ACM, New York, NY, 2000.

32. Logan, B., J.-M. V. Thong, and P. J. Moreno, "Approaches to reduce the effects of OOV queries on indexed spoken audio", *IEEE Transactions on Multimedia*, Vol. 7, No. 5, pp. 899–906, 2005.

33. Matthews, B., U. Chaudhari, and B. Ramabhadran, "Fast Audio Search Using Vector Space Modeling", *IEEE Workshop on Automatic Speech Recognition Understanding*, pp. 641–646, December 2007.

34. Parlak, S., "Speech Retrieval for Turkish Broadcast News", MS Thesis, Boğaziçi University, 2008.

35. Soltau, H., B. Kingsbury, L. Mangu, D. Povey, G. Saon, and G. Zweig, "The IBM 2004 conversational telephony system for rich transcription", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 205–208, 2005.

36. Sak, H., T. Güngör, and M. Saraçlar, "Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus", *GoTAL 2008*, Vol. 5221 of *LNCS*, pp. 417–427, Springer, 2008.

37. Arısoy, E., D. Can, H. Sak, S. Parlak, and M. Saraçlar, "Turkish Broadcast News Transcription and Retrieval", *IEEE Transactions on Speech and Audio Processing*, Vol. 12, No. 2, pp. 291–301, June 2009.

38. Can, D., E. Cooper, A. Ghoshal, M. Jansche, S. Khudanpur, B. Ramabhadran, M. Riley, M. Saraclar, A. Sethy, M. Ulinski, and C. White, "Web derived pronunciations for spoken term detection", *SIGIR '09: Proceedings of the 32th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 83–90, 2009.

39. Mohri, M., "Semiring Frameworks and Algorithms for Shortest-Distance Problems", *Journal of Automata, Languages and Combinatorics*, Vol. 7, No. 3, pp. 321–350, 2002.

40. Mohri, M., F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition", *Computer Speech & Language*, Vol. 16, No. 1, pp. 69–88, 2002.

41. Mohri, M., "Finite-State Transducers in Language and Speech Processing", *Computational Linguistics*, Vol. 23, No. 2, pp. 269–311, 1997.

42. Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A General and Efficient Weighted Finite-State Transducer Library", *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, Vol. 4783 of *Lecture Notes in Computer Science*, pp. 11–23, Springer, 2007, `http://www.openfst.org`.

43. Blumer, A., J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas, "The smallest automaton recognising the subwords of a text", *Theoretical Computer Science*, Vol. 40, pp. 31–55, 1985.

44. Crochemore, M., "Transducers and repetitions", *Theoretical Computer Science*, Vol. 45, No. 1, pp. 63–86, 1986.

45. Mohri, M., P. Moreno, and E. Weinstein, "General suffix automaton construction algorithm and space bounds", *Theoretical Computer Science*, Vol. 410, No. 37, pp. 3553 – 3562, 2009.

46. Blumer, A., J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht, "Complete inverted files for efficient text retrieval and analysis", *Journal of the ACM*, Vol. 34, No. 3, pp. 578–595, 1987, July.

47. Mohri, M., F. C. N. Pereira, and M. Riley, "Weighted Automata in Text and Speech Processing", *Proceedings of ECAI, Workshop on Extended Finite State Models of Language*, 1996.

48. Can, D. and M. Saraclar, "Score Distribution Based Term Specific Thresholding for Spoken Term Detection", *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pp. 269–272, Association for Computational Linguistics, Boulder, Colorado, June 2009.