VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

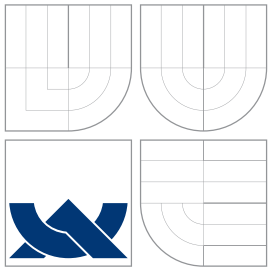
SEARCH IN SPEECH DATA

DIPLOMOVÁ PRÁCA
MASTER'S THESIS

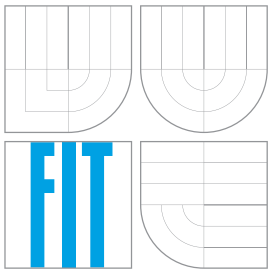
AUTOR PRÁCE
AUTHOR

MICHAL FAPŠO

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHĽADÁVANIE V ZÁZNAMOCH REČI

SEARCH IN SPEECH DATA

DIPLOMOVÁ PRÁCA
MASTER'S THESIS

AUTOR PRÁCE — MICHAL FAPŠO
AUTHOR

VEDÚCI PRÁCE — Ing. IGOR SZÖKE
SUPERVISOR

BRNO 2007

Abstrakt

Táto práca popisuje navrhnutý a implementovaný systém pre efektívne ukladanie, indexovanie a vyhľadávanie v rečových dokumentoch, ktorý využíva automatické rozpoznávanie reči. Keďže kvalita dnešných systémov pre rozpoznávanie reči nie je dostatočná na optimáne použitie v iných aplikáciach, je potrebné indexovať "mnohovýznamový" výstup rozpoznávača – grafy hypotéz. Potom je však nie je možné použiť štandardné metódy známe z textových vyhľadávacích systémov. Táto práca pojednáva o optimalizovanom systéme na indexovanie a efektívne vyhľadávanie v komplexných a objemných dátových štruktúrach, ktoré sú výstupom z rozpoznávača reči.

Kľúčové slová

Vyhľadávanie, vyhľadávanie v reči, vyhľadávanie v grafoch, indexovanie, indexovanie grafov

Abstract

This thesis describes a designed and implemented system for efficient storage, indexing and search in collections of spoken documents that takes advantage of automatic speech recognition. As the quality of current speech recognizers is not sufficient for a great deal of applications, it is necessary to index the ambiguous output of the recognition, i. e. the acyclic graphs of word hypotheses – recognition lattices. Then, it is not possible to directly apply the standard methods known from text–based systems. This paper discusses an optimized indexing system for efficient search in the complex and large data structures which are the output of the recognizer.

Keywords

Search, search in speech, search in graphs, indexing, indexing of graphs

Bibliography

Michal Fapšo: Search in speech data, master's thesis, Brno, FIT BUT in Brno, 2007

Search in speech data

Disclaimer

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

.....
Michal Fapšo
June 16, 2007

Acknowledgement

This work was partly supported by European projects AMIDA (IST-033812) and Caretaker (FP6-027231), by Grant Agency of Czech Republic under project No. 102/05/0278 and by Czech Ministry of Education under project No. MSM0021630528. The hardware used in this work was partially provided by CESNET under projects No. 119/2004, No. 162/2005 and No. 201/2006.

Contents

Chapter 1

Introduction

It is very likely that today's success of Google in text search will excite interest in search also in other media. Among these, search in speech is probably the most interesting, as most of the human–to–human communication is done by this modality. Although there is a plenty of audio records publicly available on the Internet, the only information we can directly get about them is a title or summary at best. But in case we are looking only for some specific information discussed for example on an one hour long meeting, we need to spend a lot of time listening to something that is not interesting, until we find what we are really looking for. In this and many other situations, a system capable of searching in speech data would be of great help. We can imagine another application for example in eLearning where students are able to search for any information they are interested in not only in written documents but also in video or audio lectures. This would significantly increase the usability of recorded lectures. In general, search in speech is necessary whenever we need to access information from multimedia records. Thus there is a plenty of other applications in call centres, meeting processing, multimedia data mining, security and defence, etc.

There are two different approaches to searching in speech data:

Spoken Term Detection (STD) addresses the problem of finding all of the occurrences of a specified term[1] in a given corpus of speech data.

Spoken Document Retrieval (SDR) aims at searching for a given term and returning the most relevant documents.

The goal of this thesis is to develop a Spoken Term Detection system capable of fast searching for a given term in large archives of audio data with high accuracy of search results and an ability to search for less common words like proper names and technical terms.

Spoken Term Detection is a complex process that needs to address the following points:

- conversion of speech to discrete symbols that can be indexed and searched – LVCSR[2] systems and phoneme recognizers can be used (Section 3.2).
- accounting for inherent errors of LVCSR and phoneme recognizer – this is usually solved by indexing and search in recognition lattices instead of 1–best output (Section 3.2).

[1] Term is a sequence of one or more words.
[2] LVCSR = Large Vocabulary Continuous Speech Recognition

- determining the confidence of a query – in this thesis done by evaluating the posterior probability (Section 3.3).
- capability of searching for less common words like proper names, technical terms or mispronounced words (Section 3.2).
- processing multi–word queries, both quoted (exact sequences of words) and unquoted (Section 3.4.2).
- providing an efficient and fast mechanism to obtain the search results in reasonable time even for huge amounts of data (Table 5.2).
- evaluation of results. In this thesis done by using NIST STD metrics (Chapter 5).

This master thesis is a continuation of the work described in my bachelor thesis entitled "Search Engine for Access to Information from Speech Recognition" [8], where several issues have not been addressed like phoneme search and OOV[3] handling.

This thesis is oriented towards indexing and search issues. For more information about LVCSR system and phoneme recognizer see [14, 11].

1.1 Thesis structure

Chapter 2 analyzes existing text and speech search systems, lists their advantages and disadvantages to provide a generalized view on searching and information retrieval.

Chapter 3 analyzes various possible approaches and shows solutions for the development of a speech search system. This chapter also covers the mechanism of searching for multi–word queries, phonetic keyword–spotting, the problem of assigning score to search results.

Chapter 4 describes the implemented system, the indexing engine , the sorter, the search engine with it's client–server architecture and a GUI for multimedia browsing with search capabilities.

Experiments and results are described in chapter 5. The system was evaluated on NIST Spoken Term Detection 2006 evaluations. Results from these evaluations together with an experiment on candidate verification are described and analyzed here. Also the size of indices is presented.

[3] OOV = Out Of Vocabulary (less common words like proper names and technical terms that are not in an LVCSR system dictionary)

Chapter 2

Search systems

Looking on the architecture of any search system, one can find out, that there are components commonly used among all kinds of search systems like indices, lexicon, data storages, etc. (Fig. 2.1). These components are usually independent on the type of data the system is searching in. On the other hand, there are also many optimizations which are data type dependent.

2.1 Existing text search systems

We can imagine the simplest case of a speech search system where a speech recognizer is used for generating automatic transcriptions from audio data to a simple text (1–best string). After the audio data are converted to the text, any text search system can be used. Although this approach leads to a higher miss rate, it is commonly used for it's simplicity. Especially when dealing with high quality audio records like from radio or television, where the speech recognizer's accuracy is quite high, this simple approach can be sufficient enough.

In a case of a speech recognizer with 100% accuracy, there would be no need to develop a speech search system, since any text search system could be used.

2.1.1 Google

A prototype of this system was developed on Standford University as a search system which makes use of maximum of information from hypertext documents [3]. Google is designed for an effective analysis and indexing of the web and for producing search results more relevant then other existing search systems. These systems depend mostly on a document term frequency. Google uses PageRank for an evaluation of a whole web where documents are connected by hyperlinks. This way relevant documents are discriminated from irrelevant ones. A hypertext document analysis is made as well to contribute to computation of document relevance.

Downloading of web pages is performed by several servers (crawlers). The URL server sends them lists of URL addresses to download. These pages are then sent to store servers, compressed and afterwards stored in a data repository. Each web page has assigned a unique number – docID – which is generated for each new downloaded web page. Indexing is performed by the indexer and the sorter. The indexer processes the data repository, decompresses documents and analyses them. Each document is converted to a set of word occurrences called hits. Each hit stores this information: word, position of the word in the document and an estimation of font size. The indexer distributes these hits to a set of

barrels creating a partially sorted forward index. The indexer also takes all links appearing in the document and stores all necessary information about them to the anchors file. This file contains enough information to determine where and whence each link points and also to get the link's title.

The URL resolver processes the anchors file and translates relative URL addresses to the absolute ones which are converted to a docID. It inserts the link title connected with a target docID to the forward index. It also generates a link database (docID $\leftrightarrow$ URL pairs). The link database is used for evaluation of PageRank score for all documents.

The sorter processes barrels which are sorted by docID and sorts them by wordID to generate an inverted index. The sorter also generates a list of all wordIDs with positions in the inverted index. An application called DumpLexicon processes the inverted index and the lexicon and generates a new lexicon for use in the searcher which is executed by a web server and uses the lexicon, the inverted index and the page relevance table for answering queries.

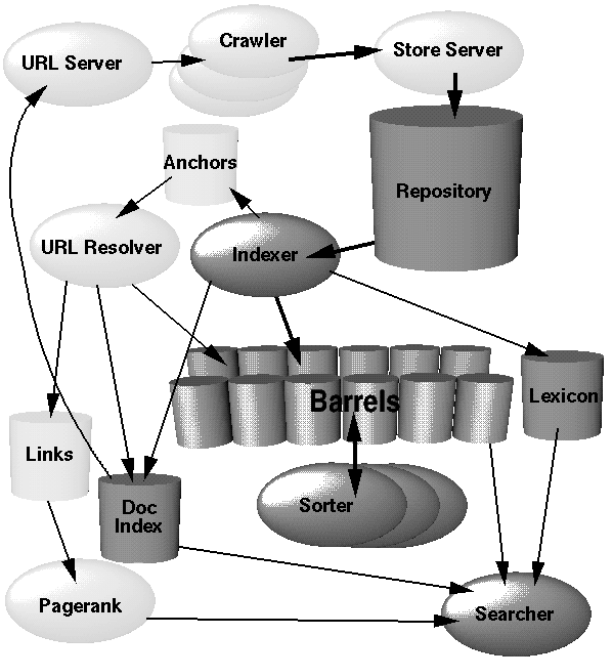


Figure 2.1: Google architecture overview. The emphasized system modules (repository, indexer, barrels, lexicon, document index, sorter and searcher) are common for all kinds of search systems

The Google search process:

1. A web server sends a request to index servers. The content of index servers is similar to an index in a book. It says which pages contain words of the given query.
2. The request is sent to document servers which contain stored documents. Short text segments describing the context of searched words are generated here.
3. Results are sent to the user through a web interface.

2.2 Existing speech search systems

Let us have a look on two commercial speech search systems. The first one – HP Speech Bot – is based on a word recognizer (LVCSR[1]) and the second one – Nexidia NEXminer Enterprise – is based on a phoneme recognizer. The differences between these two approaches as well as the problem of OOV handling will be described later in the chapter 3

2.2.1 HP Speech Bot

Is a system which processes a broadcasting of several radio and television stations which provide records on the internet. Each record is transcribed to a text in which users can search. To be able to process several broadcastings in real–time, the system runs on several computers [17].

- Advantages
  - High search speed since the system searches in a simple text.
  - Recognition of most of the radio and television records has very high accuracy (one speaker, high quality of the audio record, low SNR[2]).
- Disadvantages
  - No speaker segmentation. If there is a discussion, the system outputs only a simple string as if there was only one speaker speaking for all.
  - Recognition of less common words (proper names, etc.) is not possible since the system uses a lexicon. According to statistics, there are only 1.2% of OOV[3] words in indexed records. However, the OOV rate of user queries climbs up to 12% which has clearly an impaction on the performance of the system since a query with a word that is out of vocabulary do not return any valid hit.
  - The recognition system outputs only a 1–best string, so it is not possible to search for any misrecognized word.

2.2.2 Nexidia NEXminer Enterprise

Nexidia NEXminer offers a scalable solution for speech analysis and data mining from audio records (conversations with customers, etc.). It uses a phonetic approach with less than 400 phonemes to cover all languages and dialects in the world. The speed of records analysis is 50–times faster than real–time while only 10% of processor speed is used and the search speed can climb as high as 100,000–times faster than real–time. The system is used mostly for speech analysis and for many statistical techniques for knowledge management, emergency services, health services, government institutions, etc. [5, 9]

- Advantages
  - Searching for less common words, proper names, mispronounced words, technical and substandard words.

[1] LVCSR = Large Vocabulary Continuous Speech Recognition
[2] SNR = Signal to Noise Ratio
[3] OOV = Out of Vocabulary (a word that is not in a recognizer's lexicon)

  - Adding a support for a new language does not require a difficult creation of a lexicon.
  - Provides analytical tools for determining correlations between records.
  - Very fast processing of records (signal processing, recognition and indexing).

- Disadvantages
  - No language model (pure accoustic approach).
  - No context of search result can be obtained.
  - Lower accuracy for common words than LVCSR systems

It is not that straightforward to say which system is better since this decision depends on an application. The two systems described above represent two possible approaches, two possible recognition systems that are used for processing audio records and converting them to a representation feasible for indexing. These and other approaches will be described in chapter 3.

2.3 Text vs. speech search systems

There are several common signs and several differences between these two types of search systems.

The architecture of both of them usually consists of three main parts — an indexer, sorter and searcher. The *indexer* takes documents we need to search in, gathers indexing units (usually words) and stores them together with some metadata into a forward index. The records in the forward index are not sorted explicitly but in case of sequential adding of new documents to the index, we can assume that all records in the forward index are sorted by document and within a document by the order of occurrence of an indexed unit. It means that for one particular indexed unit, it's occurrences are spread among the whole index. The *sorter* takes the forward index and sorts it by the indexing units to create an inverted index. The inverted index is much more suitable for searching because it contains all occurrences of each indexed unit on one place. The *searcher* takes a query and searches for it in the inverted index.

The differences between text and speech search systems come from the different input data types. To index text data only a simple parser is needed to gather all words from the input document. Indexing speech data involves the use of a speech recognizer which can provide indexable representation of the speech signal (either text transcripts or graph of hypotheses). In case of using text transcripts as the input to the indexer, there is no difference between the text and speech search system. But if the indexer processes graphs of hypotheses, a specialized speech search system has to be used. It has to be able to deal with parallel hypotheses and it has to assign score to them as well. More about these speech search system's specific issues can be found in the next chapter.

Chapter 3

Analysis

The main goal of this master thesis is to develop a search system capable of indexing and search in speech data. In the previous chapter few existing search systems were described.

To transform a speech audio signal to a representation feasible for indexing a speech recognition system will be used. After the recognition phase an indexer will process the recognizer's output and store it in an optimized way to provide fast access to the data needed in the process of searching and to save the disk space as well. Another important thing is to assign score to hypotheses which should be uniform among various speech records.

3.1 Speech recognition systems overview

To be able to search in speech data it is needed to transform an audio signal to some other representation which is more feasible for indexing. For such purpose various speech recognition systems can be used. In general we can split them to the following categories:

Large Vocabulary Continuous Speech Recognition (LVCSR) system is based on an acoustic model a pronunciation dictionary and on a language model (Fig. 3.1). Acoustic models determine a likelihood of occurrence of the smallest acoustic units – phonemes. These models are trained on speech records tagged with human–made transcripts. To be able to output words a dictionary of pronunciation variants is used (for English it consists of about 50 to 100 thousand words). The language model assigns probabilities of word sequences: "running ahead" vs. "running a head". It is trained on large text corpora containing usually 10 to 1,000 mega–words. Because of the closed vocabulary the LVCSR system is unable to recognize the words which are not included in the vocabulary (out–of–vocabulary words). The most important part of the LVCSR system is a decoder which makes use of the acoustic models the dictionary and the language model to determine the optimal word sequences in the input signal. In this thesis the AMI LVCSR system [15, 7] will be used.

Subword–unit recognizer does not output words but smaller units like syllables or phonemes instead. It is more simple than the LVCSR system because it has a much smaller dictionary (in English there are $\sim$ 45 phonemes). A subword–unit recognizer also uses acoustic models and a phoneme language model in the way similar to the LVCSR system. The decoder examines various paths on which the language and acoustic probabilities are accumulated and it chooses the best path afterwards. In this thesis a phoneme recognizer based on LVCSR acoustic models was used. Only a word

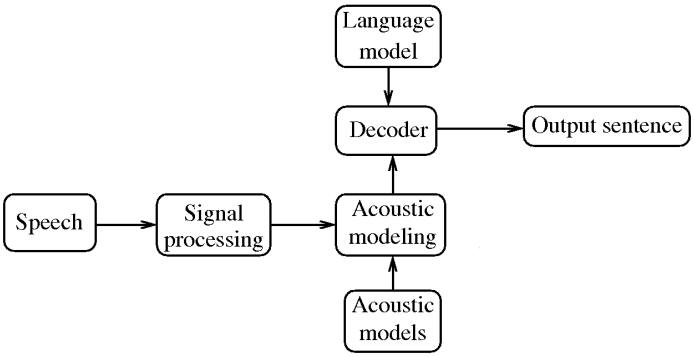


Figure 3.1: LVCSR system architecture

dictionary was omitted and the word language model was replaced by a phoneme language model.

Both the LVCSR and the phoneme recognizer can provide various possible output types:

N–best string represents the N most probable transcriptions of speech. Let's consider $N = 1$ (1–best string) as a special case of N–best string which represents the most probable transcription of the processed speech. In the best case it should look exactly like a transcript made by human. But in fact the real applications of speech recognizers are still far from this ideal case. Since the performance of a recognition system depends on various circumstances (noise in the background, microphone quality, sampling rate, ...), it is difficult to build a 100% accurate recognizer. Therefore, in the 1–best string there are various errors (insertions, deletions and substitutions) which lead to a lower STD performance [10]. However, the advantage of using the 1–best string output is in the possibility of using an existing text search system for searching in it.

For higher N the number of misses is decreasing. The fact that there are N parallel hypotheses ($N > 1$) at the same time with assigned scores requires more changes to the text search system than in the most simple $N = 1$ case.

Graph of hypotheses (a lattice) allows for more hypotheses in parallel. In comparison with the N–best string output a lattice can be stored more efficiently. It consists of nodes and links. Each node has a time mark and each link, representing a particular word hypothesis between two nodes, has a likelihood assigned. An LVCSR system assigns to all links in the lattice a language model likelihood and an acoustic likelihood while a phoneme recognizer outputs only an acoustic likelihood. Taking this into account a score for each link and for a sequence of links is needed to be computed (either in the indexing or in the search phase).

The experiments of Szöke et al. [16] have shown that in case the phoneme lattice is dense, it is sufficient to look for an exact match of the searched string and not to take into account substitution, insertion and deletion errors.

To decide which output type should be chosen for indexing the resulting index size and search speed is taken into account. To build a highly scalable speech search system lattices were chosen as the recognizer's output format since all other output types can be converted to a lattice structure.

3.2 Speech search systems overview

LVCSR–based search systems (indexing words) index an output of an LVCSR system. Although the LVCSR–based search systems are quite accurate, the restriction of using a closed vocabulary degrades them significantly. According to various statistics the user queries contain less common words (OOVs) in 10–50%. Since the recognizer is not able to recognize such words, the search engine can not search for them which leads to much lower retrieval performance.

Subword unit–based search systems (indexing phonemes and/or syllables) index an output of a subword unit (phoneme) recognizer which in general has a lower accuracy than LVCSR systems. It does not have any dictionary and word language model which makes it's output robust for recognition of less common words. On the other hand this approach degrades the recognition accuracy.

To be able to search in an output of a subword unit–based recognizer another system capable of transcribing word queries to subword queries needs to be developed. It is called "grapheme to phoneme converter" (or grapheme to syllable) and it is dependent on the language in which the user enters a query. Another option is to use a dictionary similar to the one used in the LVCSR system to be able to transcribe at least the most frequent words to phonemes. Assume we have a grapheme to phoneme converter for the english language. If a user wants to search for the english word "actually" the system transcribes it to the phoneme string "ae k ch uw ax l iy" and then the system can search for it in the output of a phoneme recognizer. The grapheme to phoneme system is able to output several pronunciation variants of a given word and it allows us to search for any word (even for words that do not exist in the language and the system did not see them in the training phase).

Combined LVCSR and subword unit–based search systems take advantage of high accuracy of LVCSR–based search systems and the robustness of dealing with less common words of subword unit–based search systems. According to our preliminary experiments the optimal way to fuse these two systems is to search for common words using the LVCSR–based system and for OOV words using the subword unit–based system. This way the fused system will take the best of both subsystems and will have significantly better retrieval performance than any of the subsystems alone (Section 5.1.4).

Taking the described approaches into an account a decision to develop a combined LVCSR and subword unit–based speech search system was made during the work on this thesis.

3.3 Confidence measures

The purpose of a confidence measure is to estimate the reliability of a recognition result. Likelihoods (state, phone, word, etc.) generated by a standard recognizer are dependent on speaker, channel, environment, background noises etc. Confidence measure should suppress this dependency.

Confidence measure estimation algorithms can be classified into three major categories [4]:

Feature–based: These approaches assess the confidence on some selected features such as language model probability, acoustic score, word duration, number of phonemes, etc.

Explicit model–based: These approaches treat confidence measures as hypothesis testing problem and need to model extra alternative hypotheses. These techniques usually use some background models or anti–models. This approach is usually used for on–line processing.

Posterior probability–based: Conventional automatic speech recognition (ASR) algorithm uses the maximum a posterior (MAP) decision rule to find the most likely sequence of units $\hat{U}$ which achieves the maximum posterior probability $p(U|X)$ given any acoustic observation X

$$\hat{U} = arg \max_{U \in \Sigma} p(U|X), \tag{3.1}$$

where Σ is the set of all permissible sentences. This equation can be rewritten using Bayes formula to form:

$$\hat{U} = arg \max_{U \in \Sigma} \frac{p(X|U)p(U)}{p(X)}, \tag{3.2}$$

where $p(X|U)$ is the probability of observing X by assuming that U is the underlaying unit sequence for X and is modeled by GM/HMM. $p(U)$ is the prior probability of U and is modeled by language model LM. And finally $p(X)$ is the probability of observing X. Because $p(X)$ is constant across different units most ASR systems simply ignore it due to optimization.

The posterior probability estimated according to the standard Maximum a Posteriori (MAP) framework is a good candidate [12] for confidence measures. It has a good bounded range between 0 and 1 and a strong background model. This approach is used in the lattice–based keyword spotting.

In this thesis the posterior probability confidence measure will be used because the output of our recognizer are lattices so we can correctly compute a posterior probability of a word.

3.4 Computing posterior probability to hypotheses

Since there are many parallel hypotheses in a lattice, it is needed to assign a score to each hypotheses. The hypotheses on the best path in the lattice should have higher score then the other hypotheses. This score will be computed for each search result so it has to be comparable among all indexed lattices in the search system.

3.4.1 LVCSR hypotheses

LVCSR lattices (example in Fig. 3.2) contain nodes carrying word labels and arcs determining the timing and acoustic (L_a^{lvcsr}) and language model (L_l^{lvcsr}) likelihoods generated by an LVCSR decoder. Usually each speech record is first broken into segments (by speaker turn or voice activity detector) and each segment is represented by one lattice. The confidence of a keyword KW is given by

$$C^{lvcsr}(KW) = \frac{L_\alpha^{lvcsr}(KW) L^{lvcsr}(KW) L_\beta^{lvcsr}(KW)}{L_{best}^{lvcsr}}, \tag{3.3}$$

where the $L^{lvcsr}(KW) = L_a^{lvcsr}(KW)L_l^{lvcsr}(KW)$.

The forward likelihood $L_\alpha^{lvcsr}(KW)$ is the likelihood of the best path through lattice from the beginning of the lattice to the keyword and the backward likelihood $L_\beta^{lvcsr}(KW)$ is the likelihood of the best path from the keyword to the end of the lattice. For a node N these two likelihoods are computed by the standard Viterbi formulae:

$$L_\alpha^{lvcsr}(N) = L_a^{lvcsr}(N)L_l^{lvcsr}(N)\max_{N_P} L_\alpha^{lvcsr}(N_P) \tag{3.4}$$

$$L_\beta^{lvcsr}(N) = L_a^{lvcsr}(N)L_l^{lvcsr}(N)\max_{N_F} L_\beta^{lvcsr}(N_F) \tag{3.5}$$

where N_F is a set of nodes directly following node N (nodes N and N_F are connected by an arc) and N_P is a set of nodes directly preceding node N. The algorithm is initialized by setting $L_\alpha^{lvcsr}(first) = 1$ and $L_\beta^{lvcsr}(last) = 1$. The last likelihood we need in Eq. 3.3: $L_{best}^{lvcsr} = L_\alpha^{lvcsr}(last) = L_\beta^{lvcsr}(first)$ is the likelihood of the most probable path through the lattice.

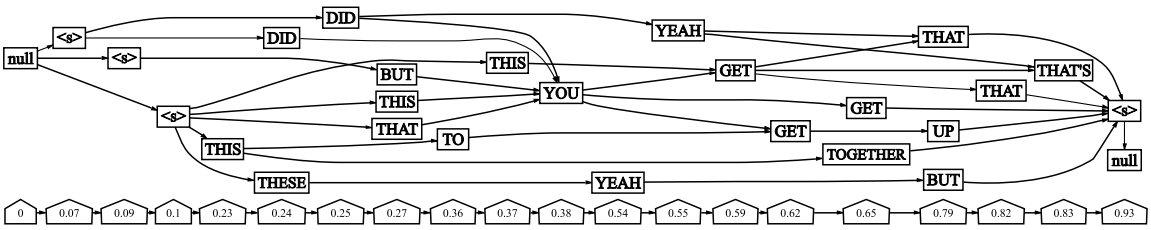


Figure 3.2: Example of a word lattice

3.4.2 LVCSR multi–word hypotheses

A usable spoken term detection system should support queries of type

`word1 word2 word3` and `"word1 word2 word3"`

with the former one representing finding words in random order with optional spaces in between (in opposite to text–search where we work within a document we specify a time–context) and the later one representing the exact match. Provided the query Q is found in the lattice we again need to evaluate its confidence $C(Q)$. Similarly to Eq. 3.3 this is done by evaluating the likelihood of the path with all the words w_i belonging to the query and dividing it by the likelihood of the optimal path:

$$C(Q) = \frac{L_{rest}\prod_i L(w_i)}{L_\alpha(last)}, \tag{3.6}$$

where L_{rest} is the likelihood of the "Viterbi glue": optimal path from the beginning of the lattice to $w_{earliest}$ connections between words, w_i (for unquoted query) and optimal path from w_{latest} to the end of the lattice. In other words L_{rest} represents everything except the searched words. $L_\alpha(last)$ is the likelihood of the best path in the lattice. We should note that each time we deviate the Viterbi path from the best one we loose some likelihood so that $C(Q)$ is upper–bounded by $\min_i C(w_i)$ — actually the confidence of the worst word in the query.

3.4.3 Phoneme hypotheses

To overcome the problem of a closed recognition vocabulary of LVCSR systems phoneme lattices (Fig. 3.3) are used for searching for out–of–vocabulary words.

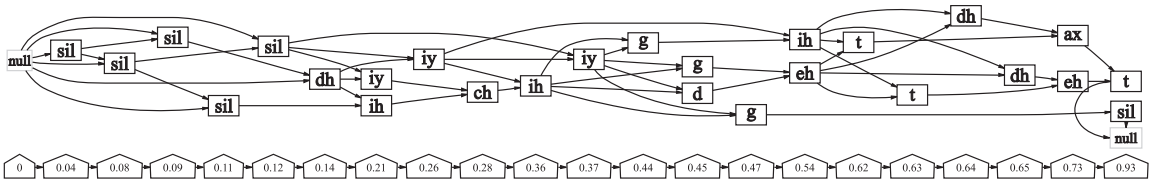


Figure 3.3: Example of a phoneme lattice

The confidence of keyword KW consisting of a string of phonemes $P_b \dots P_e$ is defined similarly as in Eq. 3.3 by:

$$C^{phn}(KW) = \frac{L_\alpha^{phn}(P_b) L_\beta^{phn}(P_e) \prod_{P \in P_b \dots P_e} L_a(P)}{L_\alpha^{phn}(last)}, \tag{3.7}$$

where $L_\alpha^{phn}(P_b)$ is the forward Viterbi likelihood from the beginning of lattice to phoneme P_b the product is the likelihood of the keyword and $L_\beta^{phn}(P_e)$ is the likelihood from the last phoneme till the end of the lattice. L_{best} is the likelihood of the optimal path.

Chapter 4

System architecture

The whole system consists of several modules as we can see on the figure 4.1. Audio data are processed by both LVCSR and Phoneme recognizers which produce data for the indexing engine. Index files are stored in the index database and lattices are stored in the description archive database. The search engine uses both databases to serve user's queries. There is also a multimedia databese from which a user can load audio and video data.

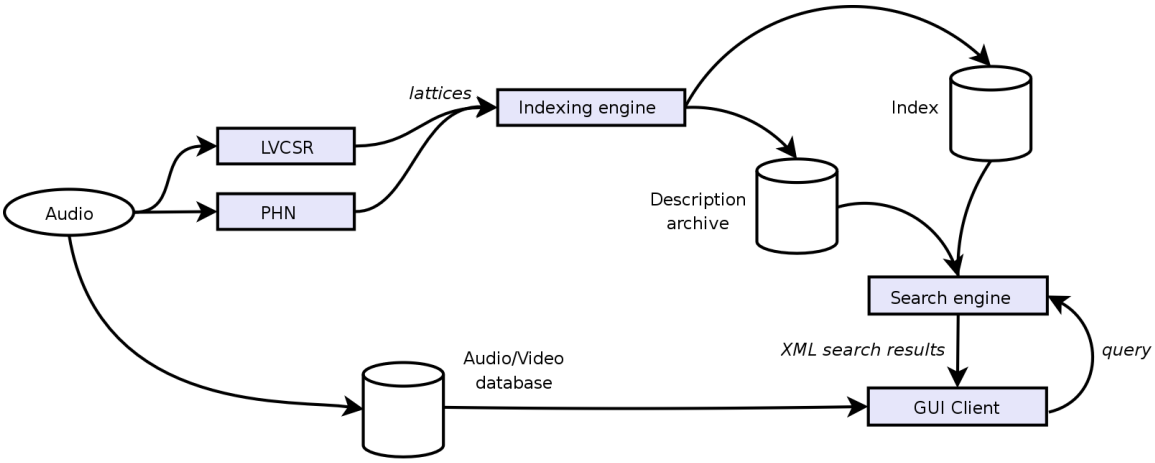


Figure 4.1: The overall design of the speech search engine.

4.1 Indexer's input — lattices

Word and phoneme lattices generated by LVCSR and phoneme recognizers represent the input for the indexing engine. The lattices (see example in Fig. 3.2 and 3.3) as an output of speech recognizers are stored in the standard lattice format (SLF) [6].

A lattice stored in SLF format consists of optional header information followed by a sequence of node definitions and a sequence of link (arc) definitions. Each node and link has an assigned ID.

Each link represents a word (or phoneme) instance occurring between two nodes. Because all the links ending in one node have the same label (the node effectively represents

the end of several word instances) the labels can be moved from each link to it's ending node.

Each node is labelled with a word (or phoneme) hypothesis and with a time. Each link has a start and end node number acoustic score and a language model score.

4.2 Indexer

The indexing mechanism (Fig. 4.2) consists of three main phases:

1. creating the lexicon
2. analyzing, storing and indexing lattices
3. creating the inverted index

The advantage of splitting the indexing mechanism into three phases is that the second phase (analyzing, storing and indexing lattices), which is the most resource demanding, can be run in parallel on several computers. Each parallel process creates it's own forward index. These indices are then merged together and sorted in the third phase to create the inverted index.

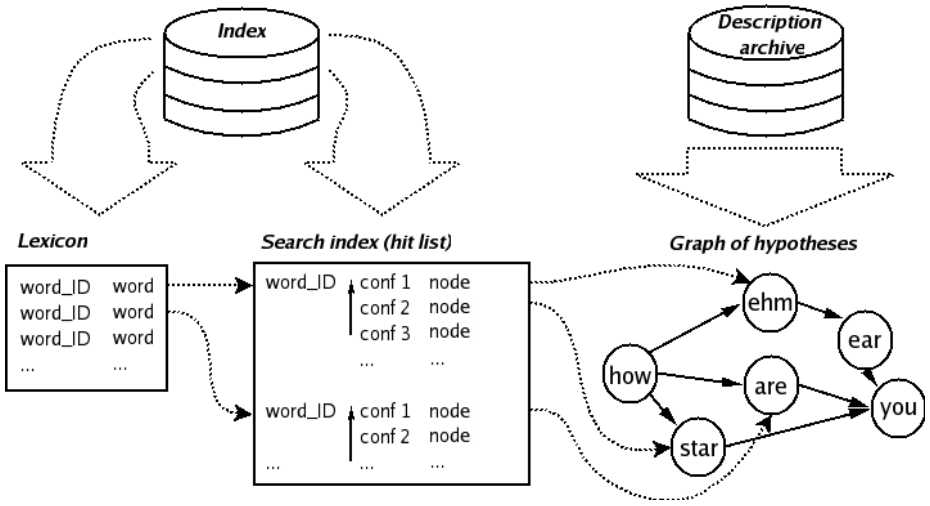


Figure 4.2: Simplified index structure

4.2.1 Creating lexicon

The lexicon provides a mapping from word to a unique number (ID) and vice versa. In general word IDs (numbers) need less space than string representation of words. Therefore, storing word IDs instead of words themselves in the search index keeps the length of records fixed, saves the used disk space and the time for comparing words.

In the English LVCSR dictionary there are usually some 50.000 – 100.000 words. But in the phoneme dictionary there are only about 45 phonemes. To keep the index structures balanced the number of phoneme indexing units should be closer to the number of words. According to Ng et al. [13] the optimal way of dealing with phonemes is to index them

as overlapping 3–grams. This way the number of phoneme indexing units grows up to $45^3 = 91125$ which is similar to the number of word units. Therefore, in the process of analysing phoneme lattices the sequences of 3 links (3 phonemes) are indexed instead of simple links in word lattices.

There are two ways how to create a lexicon for LVCSR search:

- The indexer can process all lattices produced by the recognizer, parse all words and create a lexicon.
- The LVCSR dictionary with pronunciation variants can be directly transformed to the search system's lexicon.

To create a lexicon for phoneme search a list of phonemes is needed. Taking this list a list of all combinations of phoneme 3–grams is generated. The phoneme search lexicon is then created from the list of phonemes and the list of 3–grams. Labels in phoneme lattices are represented by phoneme uni–grams and records in the phoneme search index contain phoneme 3–grams.

The lexicon (both LVCSR and phoneme) consists of three files:

List of units stores a list of records consisting of a word, phoneme or phoneme 3–gram's ID and it's string representation.

Alphabet index is a list of pointers to the list of units sorted in alphabetical order. It is used for fast mapping from unit's string label to it's ID.

ID index is a list of pointers to the list of units sorted by word/phoneme ID. It is used for fast mapping from unit's ID to it's string label.

4.2.2 Analyzing, storing and indexing lattices

Before a speech document is processed by a speech recognition system it is segmented using a voice–activity detector (VAD) to speech and non–speech segments. The recognizer then processes each speech segment separately and for each segment it creates one lattice. The time information on nodes in each lattice is relative to the start of it's segment. This information is recorded as the start frame number (a frame is usually 10ms long) in the lattice's filename. However, the indexer is able to concatenate lattices of the same document to create one lattice per speech document. This is useful in the searching phase when a search result traverses through two (or theoretically even more) lattices. For example if someone talks in the speech record and makes a few seconds long pause it will be detected as a silence and two segments (and lattices) will be created. Then in the candidate verification step in the searching phase (Section 4.4) it will not be possible to verify a multi–word term beginning in the first lattice and ending in the second one. Using the lattice–concatenation feature overcomes this problem.

In the phase of indexing lattices each lattice stored in SLF file generated by the recognizer is parsed and loaded into the indexer's internal structures. The time information on nodes is converted from relative to absolute using the start frame number in lattice's filename.

Next it is needed to compute α and β likelihoods and afterwards a confidence score using the Viterbi formulae for each word in case of indexing LVCSR lattices (Section 3.4.1) and for each 3–gram in case of indexing phoneme lattices (Section 3.4.3). To be able to use

dynamic programming techniques in the Viterbi algorithm nodes in the lattice are sorted in topological order so that each node appears in the order after all of its predecessors.

Since there are parallel hypotheses in lattices, it is quite common that there appear several overlapping instances of the same word or phoneme 3–gram at the same time. Although each of them has a different time alignment and confidence, it is not necessary to store them all in the index. In the indexer there is a stack mechanism implemented which takes care about storing only one instance of overlapping hypotheses with the best confidence score and outer time boundaries (Fig. 4.3).

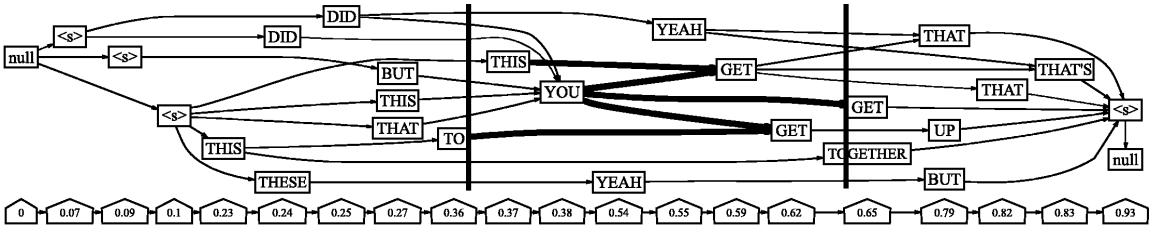


Figure 4.3: There are 5 instances of the word "GET" (horizontal bold lines) in parallel. However, the indexer stores only one record to the inverted index which represents all 5 occurrences. It assigns to this record the best confidence score and the outer time borders (vertical bold lines) of these overlapping parallel hypotheses.

The main purpose of the indexer is to generate a forward index which consists of all hypotheses occurring in lattices. Each record in the forward index has a fixed length and it consists of a word or phoneme 3–gram ID, confidence, time and document ID.

To be able to assign correct confidence scores to multi–word queries and to be able to extract the context around a found search term lattices have to be available in the search phase. To keep the process of searching as fast as possible various optimizations in the lattice storing structures have been done. Each stored lattice consists of several files:

Nodes list stores the information about nodes in the lattice: node ID, word or phoneme ID, time and α and β likelihoods. Since each record in the node list has a fixed length and the records are sorted by node ID, it is possible to directly address any record by it's node ID.

Links list stores all forward and backward links for each node in the lattice. In the header part of each node there is a node ID, the number of forward links and the number of backward links. Then a list of forward and backward links follows with a confidence score and a target node ID of each link. For a forward link the target node is the node closer to the end of the lattice and for the backward link it is the one closer to the beginning. Since there is a non–uniform number of links for each node the records in the links list have variable length. Therefore another indexing structure is needed.

Links index consists of pointers to the records in the links list and provides a fast access to the records in it. The pointers are sorted by node ID so it is possible to directly address a pointer to links of any node by it's node ID.

Time index is especially needed for huge lattices. When a lattice is loaded in the process of candidate verification in the search phase the time index makes it possible to load only a small part of a lattice around the found term. During the indexing of a lattice

the time index is filled at a specified sampling rate with the records carrying time and the first node ID occurring at that time. Then in the searching phase it is only a matter of selecting the outer time boundaries around the found term and the time index provides the corresponding node IDs. When the first and last node ID is known the loading of a part of lattice bounded by these nodes is straightforward.

4.3 Creating the inverted index

Each speech document (record) can be represented by several huge lattices (depending on number of recorded channels or speakers). The inverted index tells us in which lattice the keyword appears and what is it's time and confidence score in this particular lattice.

As it was mentioned in the section 2.3 the records in the forward index are sorted by document (in the order in which documents were indexed) and within a document by the order of occurrence of an indexed unit in the lattice.

The forward index itself is, apparently, not very useful for searching for a particular word in all indexed data because it would be necessary to go through the hit list sequentially to select all matching words. Therefore an inverted index is created which has the same structure as the forward index but is sorted by words (or word IDs in our case). It means that all the occurrences of a particular word are stored at one place. This is mainly important to reduce the disk I/O operations as much as possible since they represent the bottle–neck of each search system [3]. There is also a mechanism for mapping any word from lexicon to the beginning of the corresponding list in the inverted index.

A results list returned to a user should be sorted by result score. Therefore a hit list for each word is sorted by confidence.

In case of searching for one simple word results can be directly obtained from the inverted index. But searching for a phrase is more difficult. To be able to search for phrases efficiently another index sorted by document ID and time is needed as well. More details will be explained in the section 4.4.

So the sorter takes the forward index, sorts it by word ID (or phoneme 3–gram ID) and for each word ID it sorts it again by confidence of it's occurrences. Then this index consisting of records which contain word ID, confidence, document ID, start time and end time is stored. Afterwards the records for each word are sequentially tagged with numbers (starting from 0 and resetting the counter for each word). Records for each word are then sorted by document ID and within each document they are sorted by the start time. This index is then stored but it's records contain only word ID and the sequence numbers (pointers to the index sorted by words and confidence)

4.4 Searcher

The searcher uses the inverted index and stored lattices to answer user's queries as fast as possible.

4.4.1 Searching using the inverted index

Searching for one word consists only from jumping right to the beginning of it's hit list in the inverted index and selecting first N (N =number of desired search results) occurrences. Since the inverted index is sorted for each word by the confidence of it's occurrences it is not needed to reorder the results list.

Processing of a multi–word query involves the following steps:

1. In case of phonetic search the user's query (term) has to be converted to a phoneme string using an automatic G2P (grapheme–to–phoneme) system. Since there are overlapped 3–grams in the inverted index, the phoneme string has to be split to a train of overlapped 3–grams as well. Then the searcher deals with the query of overlapping phoneme 3–grams as if they were words in a quoted query (phrase).

   Terms shorter than 3 phonemes (in total) are not searched and are dropped.

2. Occurrences of all words are retrieved from the inverted index sorted by word ID, document ID and time.

3. Based on frequencies of words the least frequent one from the query w_{lf} is selected.

4. For each occurrence of w_{lf} a recursion algorithm is applied on the left and on the right word beside w_{lf}. The recursion algorithm uses a binary search algorithm to find the occurrence(s) which are close to the current occurrence of w_{lf} and verifies if they are within the specified time interval from w_{lf}. For non–quoted queries this time interval is usually longer than for the quoted queries. In case of overlapping phoneme 3–grams the occurrences even have to be overlapped. In this process of selecting occurrences the way of indexing overlapping hypotheses (Fig. 4.3) has to be taken into account.

   If there are more words in the query the recursion algorithm applied to the left word will move on to the left word and the one applied to the right word will move on to the right word.

   If the time constraints are satisfied for the whole query for some selected sequence of occurrences then this sequence is appended to a list of candidates.

5. The candidates list is then sorted by the upper–bound of query confidence as described in the section 3.4.2. The list is then limited to the pre–determined number of candidates (usually 10).

6. Now there are 2 options. Either the search engine sends the candidates list to the user as results and the searching is done or the candidates are verified in the corresponding lattices.

4.4.2 Verification of candidates

All previous search steps were based on information stored in the inverted index (and other index structures). Since there is no information about links transitions in the inverted index, it happens that in case of a multi–word phrase query there are candidates which do not represent any existing path in the corresponding lattice traversing through all the words from the phrase. Therefore the candidate verification algorithm was implemented in the searcher to verify the existence of the valid path in the lattice itself (see the experiment in section 5.2).

If we want to perform the verification step as well, the system takes the list of candidates from the previous search step and evaluates the confidence score correctly for each candidate in it's corresponding lattice using Eq. 3.6. While looking for the "Viterbi glue" the Viterbi algorithm is extended before and after the part of lattice containing Q in order to obtain the left and right contexts. Of course, the context can be retrieved only from lattices. Therefore it is possible only if the verification step is performed.

Because lattices can be quite huge, a time index is used for loading only a small part of the corresponding lattice in the verification step. This way the speed of verification does not suffer from huge lattices.

After the part of lattice containing the candidate is loaded into the memory a simple decoder is run. It traverses through the nodes (which are topologically ordered) and when it finds a node with the label equal to the first word (or phoneme) in the query it sends a token through all it's outgoing links to the successor nodes. As the token traverses through a link AB (from node A to node B) a confidence of the traversed path from the node where the token has started is computed by the following equation (log–likelihoods are used instead of likelihoods and therefore multiplication is replaced by sum and division is replaced by subtraction which leads to faster computation):

$$C_{path} + C_{newlink}(AB) = C_{path} - L_\beta(A) + L_\alpha(last) + L(AB) + L_\beta(B) - L_\alpha(last) \quad (4.1)$$

where $C_{path} - L_\beta(A) + L_\alpha(last)$ is the confidence of the traversed path without the right side (see Eq. 3.3), $L(AB) + L_\beta(B)$ is the right side of the path with the new link AB and $-L_\alpha(last)$ is there for normalization of the confidence over the whole lattice. C_{path} is initialized to the confidence of the first link on the token's path. A confidence of all links in the lattice $C(KW)$ is stored in the lattice's links list. $L_\beta()$ and L_α are stored in the nodes list of the lattice. Since

$$C(AB) = L_\alpha(A) + L(AB) + L_\beta(B) - L_\alpha(last) \quad (4.2)$$

$L(AB)$ can be expressed by:

$$L(AB) = C(AB) - L_\alpha(A) - L_\beta(B) + L_\alpha(last) \quad (4.3)$$

and the equation 4.1 can be rewritten as:

$$C_{path} + C_{newlink}(AB) = C_{path} - L_\beta(A) + L_\alpha(last) + C(AB) - L_\alpha(A) \quad (4.4)$$

This way a confidence of each token is computed. After the tokens reach the end of the loaded part of lattice the ones which do not satisfy the query are filtered out and the best token is chosen from the rest. This token keeps the confidence of the whole query occurrence the start time, the end time and the nodes history (visited nodes). To extract the context around the query occurrence is a simple matter of traversing the lattice from the first node on the best token's path to the beginning of the lattice and from the last node on the best token's path to the end of the lattice by choosing the links with the best confidence. These traversed links represent the context surrounding the occurrence of the query.

It might happen that in the candidate list there are few candidates close to each other and for all of them the same part of lattice is loaded and used for token passing. This could lead to a problem of preference of one token for all these candidates among the other tokens. Therefore the same occurrence would be returned as the result of token passing and although the other candidates might have only a little bit lower confidence they would be suppressed by the best one. This problem is avoided by starting a token only from within the time boundaries defined by each candidate.

4.4.3 Combining LVCSR and phoneme search

Each word in a query is either in–vocabulary (IV) or out–of–vocabulary (OOV) according to it's presence in the LVCSR dictionary.

The query is split to sequences of IV and OOV words. Each sequence represents a new query for which a list of candidates is obtained as it is described in the section 4.4.1.

Then the system deals with the candidates list of each sequence in the same way as with occurrences retrieved from the inverted index and the searching continues with the step nr. 3 in the section 4.4.1. But the verification step is different. Each candidate is again split to IV and OOV sequences which are verified in the corresponding lattices (IV sequences in LVCSR and OOV sequences in phoneme lattices)

If all sequences were successfully verified, the time and score is produced. Score is computed as a sum of IV (LVCSR) sequences and OOV (PHN) sequences.

4.5 GUI Client

The search engine can run in two modes. Either as a standalone application or as a non–blocking server. In the standalone mode the query is passed as a command line argument. In the server mode a client sends the query to the server address (defined by IP and port) and the server sends back search results in the form of an XML string. For communicating with the server any simple telnet–like application can be used.

To provide more interaction for the user a modular multimedia browser with the capability to communicate with the search engine (Fig. 4.4) was developed. The search plug–in communicates with a server and provides browsable results to a user in a google–like way.

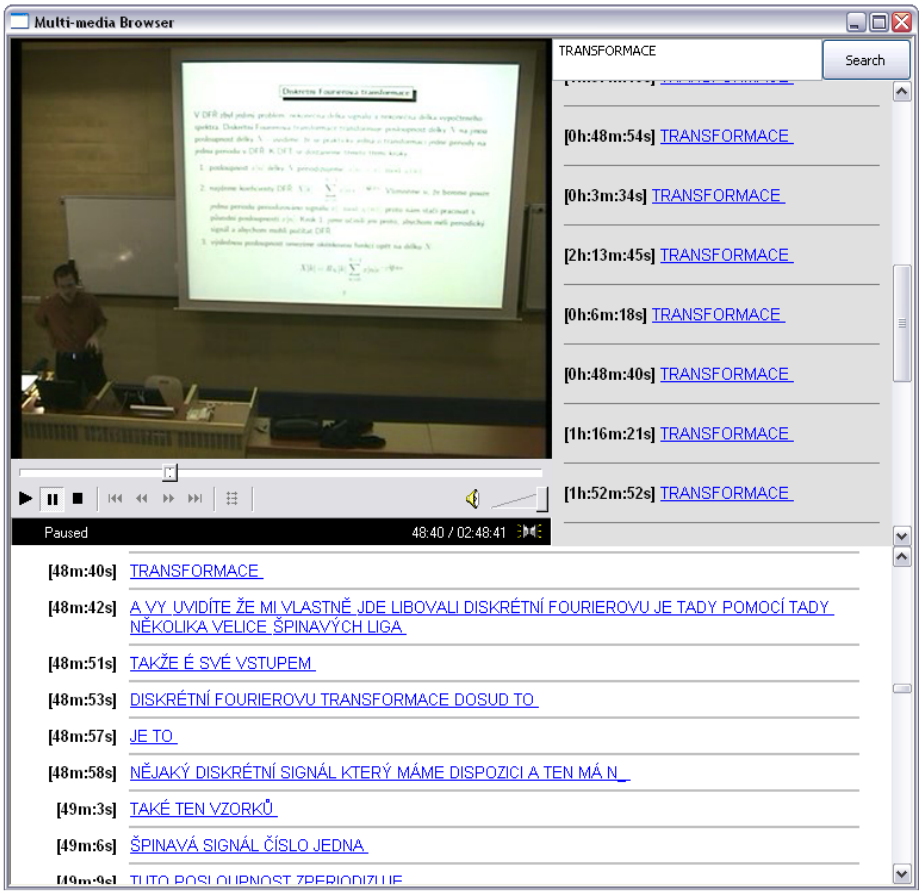


Figure 4.4: P.R.A.S.E. Multimedia browser showing the search results for "TRANSFORMACE" in the Signals and systems lecture

Chapter 5

Experiments and results

In this chapter the results of the 2006 Spoken Term Detection evaulations organized by NIST are presented as well as the description of the submitted system. At the end of this chapter, some interesting results on candidate verification experiment in lattices are presented.

5.1 NIST Spoken Term Detection evaluations

The National Institute of Standards and Technology (NIST) has created an evaluation initiative to facilitate research and development of technology for retrieving information from archives of speech data. This initiative is called Spoken Term Detection (STD) and is structured as a collaborative research activity that is intended to foster technical progress in STD [2].

The STD task is to find all of the occurrences of a specified "term" in a given corpus of speech data. For the STD task, a term is a sequence of one or more words.

The evaluation is intended to help develop technology for rapidly searching very large quantities of audio data. Although the evaluation actually uses only modest amounts of data, it is structured to simulate the very large data situation and to make it possible to extrapolate the speed measurements to much larger data sets. Therefore, systems must be implemented in two phases: indexing and search. In the indexing phase, the system must process the speech data without knowledge of the terms. In the search phase, the system uses the terms, the index, and optionally the audio to detect term occurrences.

For each term supplied to the system, all of the occurrences of that term in the test corpus are to be found and statistics for each found occurrence are to be output. For each found occurrence of the given term, the system is to output a record that includes

- the location of the term in the audio recording
- a score indicating how likely the term exists with more positive values indicating more likely occurrences
- a hard (binary) decision as to whether the detection is correct

A system output will be considered correct if the term appears in the transcript as an exact match (disregarding case) and if the time of the occurrence corresponds to that of the matching transcript. The score for each term occurrence can be of any scale (NIST

recommends a log likelihood[1]). However, since the scores will be used to derive term–pooled Decision Error Tradeoff (DET) curves, scores across terms must be commensurate to ensure minimum DET curves

Two data sets were provided in the '06 evaluations: a development set ("DevSet") used to aid the research, and an evaluation set ("EvalSet") that was supplied at the beginning of the formal evaluation.

The development and evaluation corpora included three languages and three source types.

- The three languages will be Arabic (Modern Standard and Levantine), Chinese (Mandarin), and English (American).
- The three source types will be Conversational Telephone Speech (CTS), Broadcast News (BNews), and Conference Room (CONFMTG) meetings i.e., goal oriented, small group, roundtable meetings.

The system described in this thesis has participated in the English and Arabic task. The search system was the same for both tasks, only the recognizers (dictionaries) were different.

	Arabic	**English**
Broadcast News	$\sim$ 1 hour	$\sim$ 3 hours
Telephone Conversations	$\sim$ 1 hour	$\sim$ 3 hours
Roundtable Meetings	No	$\sim$ 2 hours

Table 5.1: Language/Source Type pairs to be tested and the durations of indexed audio for both the DevSet and EvalSet

5.1.1 DET curve

Detection Error Tradeoff (DET) curves project the dependency between miss probability (P_{Miss}) versus false alarm probability (P_{FA}). Miss and false alarm probabilities are functions of the detection threshold θ, and are computed separately for each search term:

$$P_{Miss}(term, \theta) = 1 - N_{correct}(term, \theta)/N_{true}(term) \tag{5.1}$$

$$P_{FA}(term, \theta) = N_{spurious}(term, \theta)/N_{NT}(term) \tag{5.2}$$

where $N_{correct}(term, \theta)$ is the number of correct (true) detections of term with a score greater than or equal to θ. $N_{spurious}(term, \theta)$ is the number of spurious (incorrect) detections of term with a score greater than or equal to θ. $N_{true}(term)$ is the true number of occurrences of term in the corpus, $N_{NT}(term)$ is the number of opportunities for incorrect detection of term in the corpus (= "Non-Target" term trials).

Since there is no discrete specification of "trials", the number of Non-Target trials for a term, $N_{NT}(term)$, is defined somewhat arbitrarily to be proportional to the number of seconds of speech in the data under test. Specifically:

$$N_{NT}(term) = n_{tps}.T_{speech} - N_{true}(term) \tag{5.3}$$

[1]The log likelihood, with base e, is suggested, so that the system may be evaluated in a variety of application scenarios that exhibit different prior probabilities

where n_{tps} is the number of trials per second of speech (n_{tps} is set arbitrarily to 1), and T_{speech} is the total amount of speech in the test data (in seconds).

P_{Miss} and P_{FA} is computed separately for each term and then averaged over the selected terms, giving equal weight to each search term:

$$P_{Miss}(\theta) = \underset{term}{average} P_{Miss}(term, \theta) \tag{5.4}$$

$$P_{FA}(\theta) = \underset{term}{average} P_{FA}(term, \theta) \tag{5.5}$$

P_{Miss} and P_{FA} are averaged over only those terms with a nonzero number of true occurrences in the test data, so that P_{Miss} is defined. DET curves are computed as a function of language and source type as well as for various selections of data and terms.

5.1.2 Term weighted value

Term–weighted value ($value_T$) is computed by first computing the miss and false alarm probabilities for each term separately, then using these and an (arbitrarily chosen) prior probability to compute term–specific values, and finally averaging these term–specific values over all terms to produce an overall system value:

$$value_T(\theta) = 1 - \underset{term}{average}(P_{Miss}(term, \theta) + \beta . P_{FA}(term, \theta)) \tag{5.6}$$

where $\beta = \frac{C}{V}.(Pr_{term}^{-1} - 1)$ and θ is the detection threshold.

For the current evaluation, the cost/value ratio, C/V, is 0.1, and the prior probability of a term, Pr_{term}, will be 10^{-4}.

The maximum possible Value is 100 percent, corresponding to "perfect" system output: no misses and no false alarms. Note that the Value of a system that outputs nothing is zero. Note also that negative Values are possible.

5.1.3 Submitted system

The submitted system has a schema shown on the Fig. 5.1.

1. The developement data were segmented and used for estimation of diarization coeficients (speaker segmentation). This was done at TNO by David van Leeuwen.

2. The segmented data were passed to the LVCSR/phoneme recognizer (description of this system is over the scope of this thesis). Lattices were the output.

3. The lattices and reference transcriptions were used for estimation of normalization coefficients (this is also over the scope of this thesis).

4. The evaluation data were segmented and diarized, then passed through the recognizers.

5. The output lattices were pruned to keep their size small. Then they were indexed and the given terms were searched.

6. Finally the confidences of the search results were normalized. These normalized results were sent to NIST for scoring.

Estimation of normalization coefficients

These coefficients are necessary to normalize scores when computing the confidence of a query result. The coefficients are additive constants (correcting factors) for per phoneme, per frame and per term correction. They were estimated using lattices and word/phoneme dictionary.

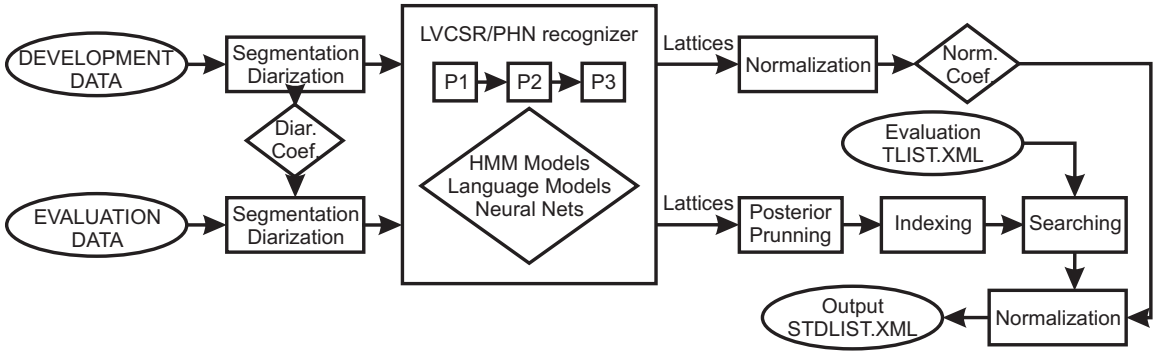


Figure 5.1: Data flow

5.1.4 Results

The table 5.2 shows us time and resources needed for data processing and search. Although the disk space needed to store lattices is quite huge, the search time is still quite low.

Index size	688.5781 MB/HS (Megabytes / Hours of Speech)
Indexing time	126.7596 HP/HS (Processing Hours / Hours of Speech)
Search speed	0.0383 sec.P/HS (Processing Seconds / Hours of Speech)
Index Memory Usage	2180.9120 MB
Search Memory Usage	24.1932 MB

Table 5.2: System performance statistics

The system was designed to achieve the highest possible term weighted value but it could be easily modified so that it would need lower system resources. Lattices could be pruned with a higher posterior pruning factor and also the recognition system could be simplified. These modifications would lead to lower value but also lower resource consumption and faster audio data processing.

The speed of search does not grow linearly with the indexed data size since various index structures and binary search algorithms are used.

Since there were three different tasks, the result values (ATWV[2] and MTWV[3]) are evaluated on each task independently. Table 5.3 shows us how the value depends on a task. The value is highest on broadcast news because of the high quality records and low SNR[4]. The value on telephone speech records is a bit lower and on meetings the value drops down significantly because of lower quality of records, higher SNR, mispronounced words, etc.

[2]ATWV = Actual Term Weighted Value
[3]MTWV = Maximum Term Weighted Value (the highest possible value which the system could have)
[4]SNR = Signal to Noise Ratio

Now let us have a look on columns "LVCSR" and "Phoneme" in the table 5.3. Here we can see a higher value for the LVCSR based system, but if we compare it's value with the fused system (the column "LVCSR + Phoneme"), we can see that the fused system is better. It is because of less common words which are not in the LVCSR dictionary (OOV[5]). The LVCSR based system is not able to find any OOV, but the phoneme based system does not make any difference in searching for common words or OOVs. We can also see a smaller change for the CTS and MTG data set then for the BCN data set. This is caused by the number of OOVs in the list of terms the system searched for. In BCN there were some 30 OOVs, in CTS there were only 5 and in MTG set only 2 OOVs.

	LVCSR		**Phoneme**		**LVCSR + Phoneme**	
Task	**ATWV**	**MTWV**	**ATWV**	**MTWV**	**ATWV**	**MTWV**
BCN	0.6278	0.6305	0.3571	0.3625	0.6541	0.6558
CTS	0.5186	0.5301	0.2977	0.3106	0.5235	0.5344
MTG	0.0463	0.0695	0.0078	0.0540	0.0549	0.0731

Table 5.3: Actual and Maximum Term Weighted Value on all three tasks using LVCSR, Phoneme and the fused LVCSR + phoneme systems

Now let us look at the value only on terms which contain at least one OOV. The column "Found OOVs" in the table 5.4 shows us the number of such terms in a particular set. If there is any in–vocabulary word in a term, it is searched using the LVCSR system. Only OOVs in the term are searched using the phoneme system. Since LVCSR has higher accurracy than the phoneme system, this combined approach should lead to higher OOV accurracy. However, we can not directly compare results of the fused system and the phoneme one. For the phoneme system, each term is an OOV. For the eval data, we do not have reference transcripts, thus we can not score the results ourselves. For the phoneme system, we have the value only for all terms (table 5.3) which can be taken as an average value for one term. For both the BCN and MTG tasks, the value of the fused system is better on OOVs then the corresponding "average" value of the phoneme system. For the CTS task, the situation is reversed. This can be caused by incorrect normalization for combined LVCSR + phoneme results. Another possibility is that the value for the phoneme system for the same terms is low as well.

Task	**Found OOVs**	**ATWV**	**MTWV**
BCN	48	0.3757	0.4094
CTS	23	0.1681	0.1835
MTG	24	0.2584	0.3084

Table 5.4: Actual and Maximum Term Weight Value on OOV queries on all three tasks using the fused LVCSR + phoneme system. Out of 1100 terms in the eval term list (for all tasks: BCN, CTS and MTG), there were 80 terms consisting of at least one OOV.

[5]OOV = Out Of Vocabulary

5.2 Verification in lattices

In the section 4.4, there were described 2 possible searching modes:

1. without verification step – based only on the information retrieved from the inverted index.

2. with verification step – like in the previous mode, but after the candidate list is retrieved from the inverted index, each candidate is verified in the corresponding lattices.

As we can see on the table 5.5, the disk space needed to store inverted indices is significantly lower than the space needed to store lattices. The reason for this fact is, that there is only an incomplete information about links and their interconnections in the inverted index. The difference of ATWV evaluated on the merged LVCSR + phoneme system with the verification step is only 0.014 higher than without candidates verification. This leads to a question, whether is the verification step worth the disk space it needs for storing lattices. This, of course depends on the application, the system is used for. However, the verification step is useful for experimental purposes to get the same search results as from the common lattice traversing keyword–spotting system [10] without any indexing mechanism.

	LVCSR		**Phoneme**		**LVCSR + Phoneme**	
	verif	**no verif**	**verif**	**no verif**	**verif**	**no verif**
size [MB]	395.8	7.8	1319	235	1716	242.8
ATWV	0.669	0.667	0.396	0.377	0.7020	0.6880

Table 5.5: Comparison of the system with and without the verification step on the broadcast news developement data.

5.3 Disk space

Table 5.6 shows disk sizes of word and phoneme lattices and indices. These sizes depend mainly on the density of lattices generated by the recognition system. The lattice posterior pruning factor is used for pruning lattices with the SRI Lattice Tool [1]. It prunes nodes with posterior less than P times the highest posterior path (P is the posterior pruning factor). The higher the pruning factor, the smaller are lattices. The results of prunning lattices using various posterior prunning factors are shown on Fig. 5.4. Pruning factors shown in the table were tuned to get the best possible retrieval performance with meaningful size of lattices. However, the size can still be smaller using the higher pruning factor at the expense of loosing some hits. The difference between the size of word and phoneme lattices is caused by the difference in length of a phoneme in comparison with the length of a word. This leads to higher number of links in phoneme lattices.

lattices type	lattice posterior pruning factor	index size [MB]	size of lattices [MB]
LVCSR	0.00001	9	121
Phoneme	0.001	180	817

Table 5.6: Disk space needed to store 8 hours of recognized speech for searching.

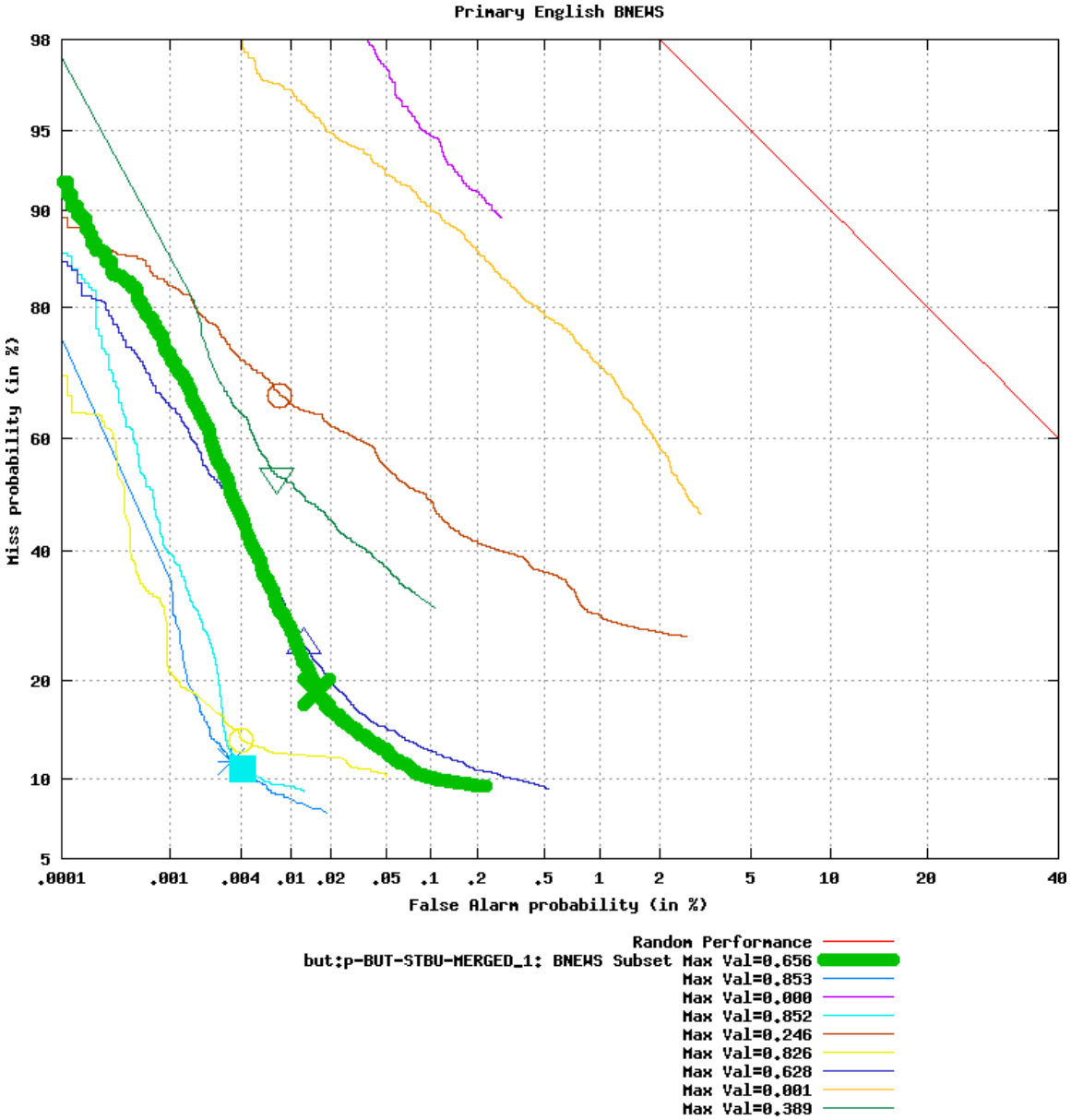


Figure 5.2: DET curves for competing English systems of all participated sites on the broadcast news task. The described system is the fourth curve (the bold one).

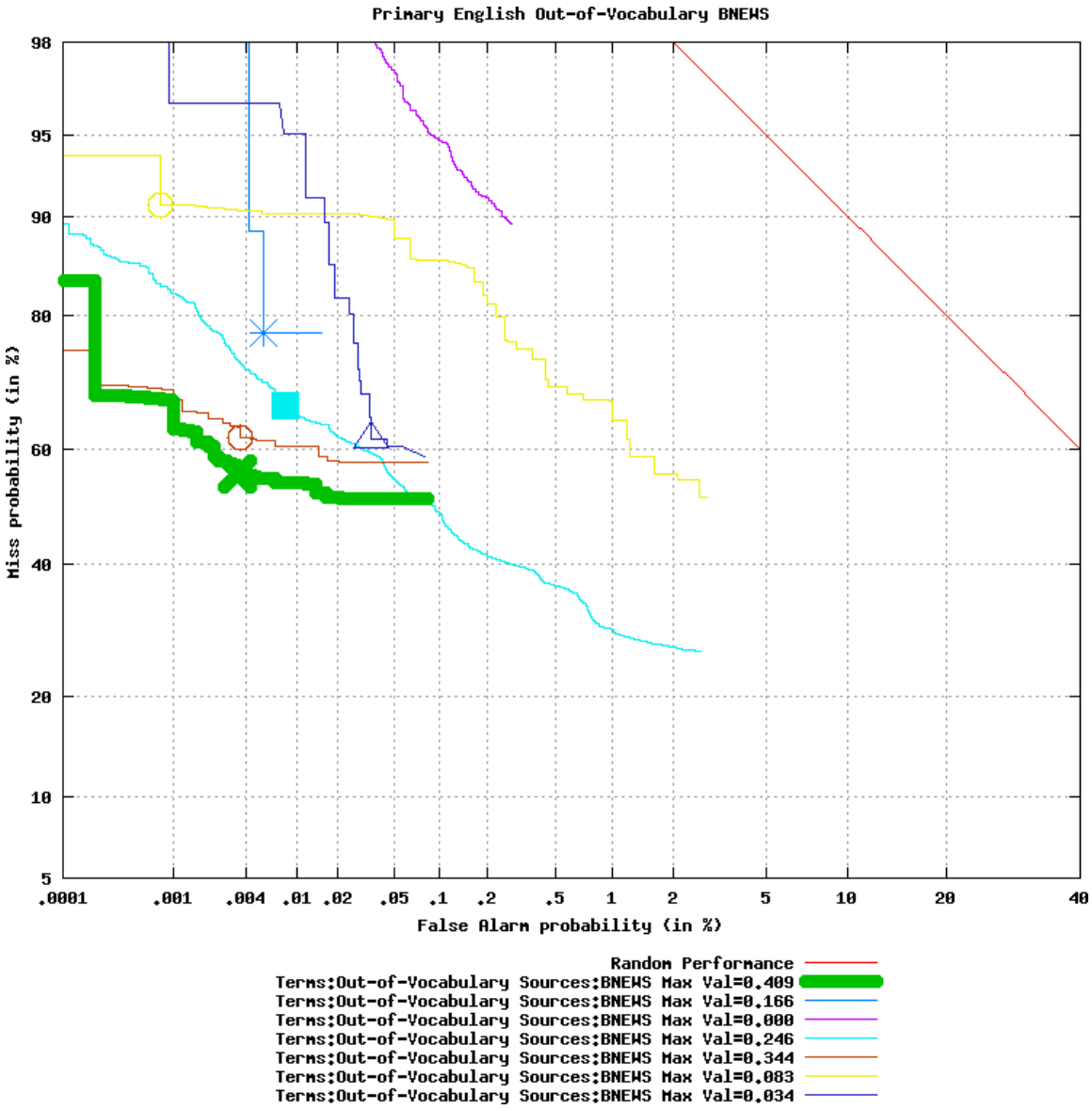


Figure 5.3: DET curves of competing English systems of all participated sites on the broadcast news task taking into account only those terms which consist of one or more OOVs. The described system is the first curve (the bold one).

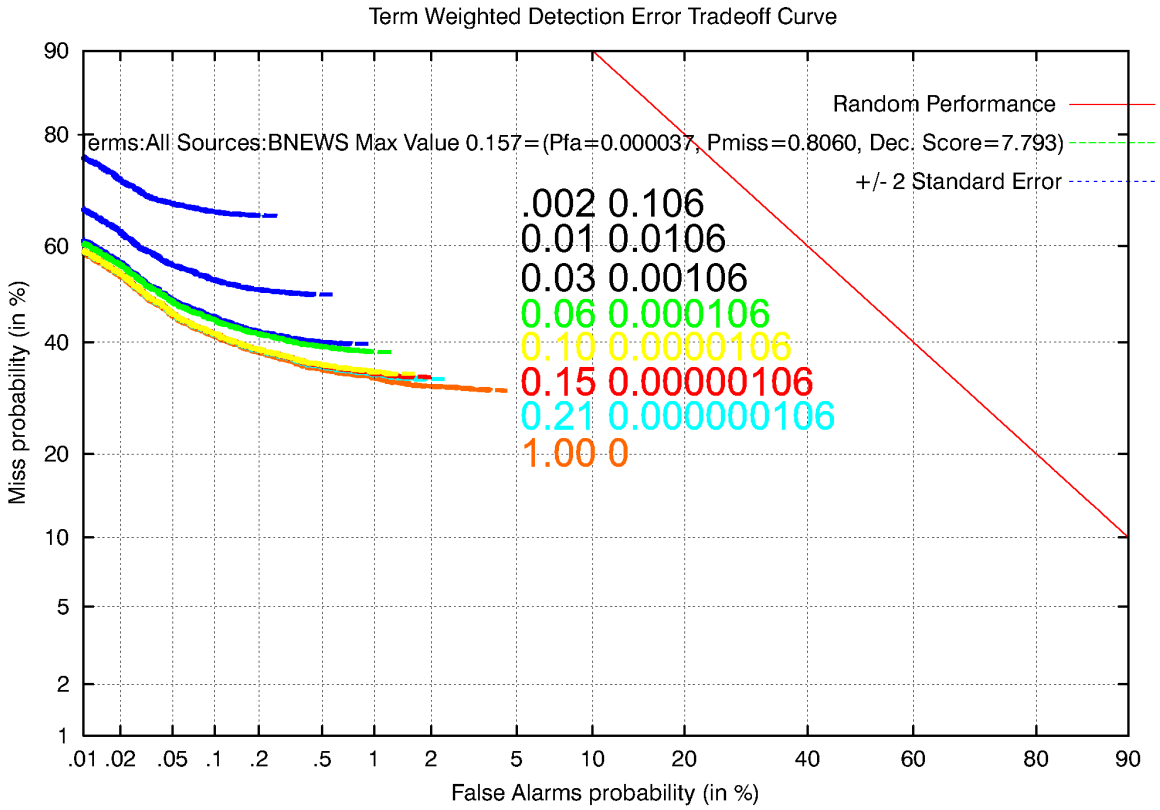


Figure 5.4: DET curves showing the result of posterior prunning. The relative lattice sizes are in the first column and the posterior prunning factors are in the second column.

Chapter 6

Conclusion

The goal of this thesis was to develop a Spoken Term Detection system capable of fast searching for a given term in large archives of audio data with high accuracy of search results and an ability to search for less common words like proper names and technical terms.

The system was developed and evaluated in the Spoken Term Detection Evaluations (STD) in 2006 organized by NIST with very good results. The search system was build in a way similar to Google but some structures had to be adapted to speech indexing and search. Since the system indexes lattices (graph of parallel hypotheses), it is able to search even for hypotheses with lower probability. This way a lower miss probability is achieved.

Various approaches to speech search systems were discussed in this thesis. The developed search system takes the best of LVCSR and phoneme based systems to retain high accuracy on common words (using LVCSR) and also to provide a search for less common words such as proper names, technical terms and misspelled words. The fusion of an LVCSR and a phoneme based system leads to very good results for less common words (OOV[1]) in the STD evaluations.

6.1 Future work

Although the system works quite fast, more optimizations will be investigated like using confusion networks instead of lattices, automatic selection of indexed N–grams, on–line index update, etc. The improvement will be evaluated on the next NIST Spoken Term Detection Evaluations.

[1] OOV = Out Of Vocabulary (less common words like proper names and technical terms that are not in an LVCSR system's dictionary)

Bibliography

[1] SRI Lattice Tool. `http://www.speech.sri.com/projects/srilm/`.

[2] NIST Spoken Term Detection Evaluations. `http://www.nist.gov/speech/tests/std/`, 2006.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[4] T. H. Chen, B. Chen, and H. M. Wang. On using entropy information to improve posterior probability-based confidence measures. In *in Proceedings ISCSLP*, Dec 2006.

[5] Mark Clements, Peter S. Cardillo, and Michael S. Miller. Phonetic searching vs. lvcsr: How to find what you really want in audio archives.

[6] Steve Young et al. *The HTK Book (for HTK Version 3.3)*. Cambridge University Engineering Department, 2005.

[7] Thomas Hain et al. The ami meeting transcription system: Progress and preformance. In *Proc. NIST RT06 evaluations*, 2006.

[8] Michal Fapšo. Search engine for access to information from speech recognition. Bachelor thesis, 2005.

[9] Mark Finlay. Effective search of large audio archives using structured mining.

[10] Szöke Igor. Keyword detection in speech data. Concept of Doctoral Thesis, April 2005.

[11] Odell J. *The Use of Context in Large Vocabulary Speech Recognition*. PhD thesis, Queens' College, March 1995.

[12] Hui Jiang. Confidence measures for speech recognition: A survey. In *Speech Communication*, volume 45, pages 455–470. Science Direct, 2005.

[13] Corinna Ng, Ross Wilkinson, and Justin Zobel. Experiments in spoken document retrieval using phoneme n-grams.

[14] Petr Schwarz, Pavel Matejka, and Jan Cernocký. Towards lower error rates in phoneme recognition. In *Proceedings of 7th International Conference Text,Speech and Dialoque 2004*, page 8. Springer Verlag, 2004.

[15] Igor Szöke, Michal Fapso, Martin Karafiát, Lukás Burget, Frantisek Grézl, Petr Schwarz, Ondrej Glembek, Pavel Matejka, Stanislav Kontár, and Jan Cernocký. But system for nist std 2006 - english. In *Proc. NIST Spoken Term Detection Evaluation workshop (STD 2006)*, page 26. National Institute of Standards and Technology, 2006.

[16] Igor Szöke, Petr Schwarz, Lukás Burget, Michal Fapso, Martin Karafiát, Jan Cernocký, and Pavel Matejka. Comparison of keyword spotting approaches for informal continuous speech. In *Interspeech'2005 - Eurospeech - 9th European Conference on Speech Communication and Technology*, pages 633–636, 2005.

[17] Jean-Manuel Van Thong, Pedro J. Moreno, Beth Logan, Blair Fidler, Katrina Maffey, and Matthew Moores. Speechbot: An experimental speech-based search engine for multimedia content in the web. Technical report, June 2001.