MULTIPLE PASS STRATEGIES FOR IMPROVING ACCURACY IN A VOICE SEARCH APPLICATION

Tianhe Zhang

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

January 2011

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of M.Eng.

2011/01/07

Abstract

This thesis describes a set of techniques for improving the performance of automated voice search services intended for mobile users accessing these services over a range of portable devices. Voice search is implemented as a two stage search procedure where string candidates generated by automatic speech recognition (ASR) system are re-scored in order to identify the best matching entry from a potentially very large application specific database. The work in this thesis deals specifically with user utterances that contain spoken letter sequences corresponding to spelled instances of search terms. Methods are investigated for identifying the most likely database entry associated with the decoded utterance. An experimental study is presented describing the characteristics of actual user utterances obtained from a prototype voice search service.

Abrégé

Cette thèse décris un ensemble de techniques pour améliorer la performance de services de recherche par la voix automatisés destinés à des utilisateurs de téléphone cellulaire qui accèdent ces services à l'aide de différents appareils portatifs. La recherche vocale est implémentée en deux étapes où la phrase candidate est générée par un système de reconnaissance de la parole automatique (Automatic Speech Recognition ou ASR) et ensuite réévaluée pour identifier le meilleur résultat provenant d'une base de donnée spécifique à l'application. Cette thèse traite principalement des énoncés d'utilisateurs qui contiennent des séquences de lettres correspondant à des instances de termes de recherche. Différentes méthodes sont utilisées afin d'identifier l'entrée de la base de données la plus semblable à l'énoncé décodé. Une étude expérimentale est présentée décrivant les caractéristiques d'un énoncé utilisateur réel obetnu à partir d'un prototype de service de recherche voix.

Acknowledgments

My thesis work was graciously supported by the joint grant offered by NSERC, McGill University and Nuance Communication Inc. I would like to acknowledge my supervisor in McGill University, Professor Richard Rose, for his support and guidance during my Master's study. Also I would like to convey my gratitude to Mr.Jean-Guy Dahan at Nuance Communication Inc, for his suggestions and supervision on my internship and thesis work. Finally, I would like to thank my wife Yi Zhang and my parents for their support throughout my studies. I will always be grateful to McGill for the experience provided there for graduate studies and to Nuance in Montreal for providing a nurturing environment for my internship.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ASR	Automatic Speech Recognition
DMC	Discriminative Model Combination
WER	Word Error Rate
LER	Letter Error Rate
PER	Phone Error Rate
DP	Dynamic Programming
LM	Language Model
SLM	Statistical Language Model
LBLM	Letter Based Language Model
SSM	Statistical Semantic Model
OOV	Out Of Vocabulary
HMM	Hidden Markov Model
NL	Natural Language
FSM	Finite State Machine
FST	Finite State Transducer

Chapter 1

Introduction

This chapter contains an introduction to an application domain in the speech technology area and a set of modeling approaches developed as part of this research. It also provides a description of the organization of this thesis documents. The specific application being addressed involves speech recognition scenarios where it is not possible to integrate all available knowledge sources available for a task in a general purpose speech recognition system.

The approaches that are investigated here integrate multiple knowledge sources to produce word strings that are consistent with models available to the speech recognition system and also consistent with application specific models that are not directly available to automatic speech recognition (ASR). With the rapid development of information technology, researchers are dealing with search and retrievals including DNA sequence alignment, fuzzy Internet search and many others. In these applications, the goal is to retrieve an optimal string from a potentially very large database, for example, looking for the best DNA match given a non-perfect (incomplete or partially missing) DNA string [1] or looking for a criminal's identity from partial information [2]. This thesis tries to present the techniques for re-ordering strings hypotheses produced by an ASR system relying on external knowledge sources.

In Section 1.1, a description of the problem addressed in the thesis will be introduced, and the problem motivation will also be discussed. Section 1.2 discusses the approaches

that are investigated to address the problems. Finally, an outline of the organization of this thesis document will be presented in Section 1.3.

1.1 Problem Description and Problem Motivation

This section introduces the problem of voice search on mobile devices and motivates the need for the approaches developed in this thesis. A large number of tasks involve accessing to online information that are easy to perform on a work station with a keyboard and large display but are more difficult to perform on a mobile device with more limited input and display modalities. In recent years, cell phones have experienced strong growth. On current cellphones, people are allowed to send text messages, to browse Internet and to search certain content. People are increasing their electronic information access via web search engines which bring lower costs for distributing information to a global audience and providing tons of information access. However for people who do not have a QWERTY type keyboard, for example mobile phone users, typing web search queries from their keypad becomes very inconvenient. As the most robust way to interact, voice has been applied to web search on mobile devices in order to overcome input constraints. Voice recognition software allows real-time transcription of the user's voice query. However the vocabulary size in automatic speech recognition (ASR) is limited. The vocabulary associated with text documents is vast and contains many special terms. A significant percentage of these words are not contained in the ASR vocabulary. Since ASR vocabulary is comparatively limited to that of a text based search engine, in the context of an open web search, recognition of out of vocabulary (OOV) words becomes a very important issue.

There are many voice enabled services for mobile users offered by communications and information technology companies [3] [4]. Nuance Communications Inc. offers a speech recognition service to all registered mobile users. By registering in this service, mobile users are allowed to take advantage of speech recognition technology to dictate their emails and text messages, to do voice Internet search, or even to do voice GPS route directing. In voice Internet search, mobile users speak one of a set of search category names, for example business names, and then speak the query term for that category. The utterance is presented to an ASR system and decoded into word strings. The word string is then presented as a text based query string to a database that may contain anywhere from approximately

10,000 to over one million entries depending on the search category. Utterances where an ASR based confidence measure does not exceed a threshold are directed to a human operator. Although this solution is functional, it is far from optimal, since it does not attempt to correct the many errors in the hypotheses and too much rely on human intervention.

When this service covers a set of areas (e.g. business names, stock quote, website names, etc.), recognition of OOV words becomes a major problem. For instance, if a mobile user searches for the term "Vuvuzela", which is a traditional African musical instrument, the term will most likely not be contained in the ASR vocabulary and will result in a recognition error. However, in practice, it is observed that users often spell out a word as is often done in human-human interaction, hoping that a spelled letter sequence would provide enough information for the search engine to retrieve a relevant database entry. In this example, the mobile user will repeat the term for several times, since the ASR is doing so poor on this OOV word, the user will likely give up speaking "Vuvuzela" at the end and instead, he might search for "musical instrument, V U V U Z E L A". Examples of "go to website S O G O U" or "stock quote B S P M" show that sometimes mobile users speak spell mode utterances "naturally". This happens when the spoken query is not even an English word, rather a website or a stock quote which as a whole do not have special meaning nor a pronunciation. As for these terms, people have already learned to speak them in spell mode from daily use. Dealing with "spell mode" utterances will be in major focus of this thesis. Analysis of 3900 utterances collected from users' interactions with this service has provided the motivation for improving "spell mode" recognition accuracy.

1.2 Approach Investigated in this Thesis

This section introduces the approaches taken to address the problems of automatic speech recognition for spell mode utterances. The general problem of spell mode ASR is that recognition performance for utterances containing OOV words or confusable words is low. In the application domain considered in this thesis, the utterances are obtained from Nuance's mobile device voice web search data. The registered users are familiar with the syntax of making a mobile voice search query which involves first speaking one of a set of search category names, and then speaking the query term for that category.

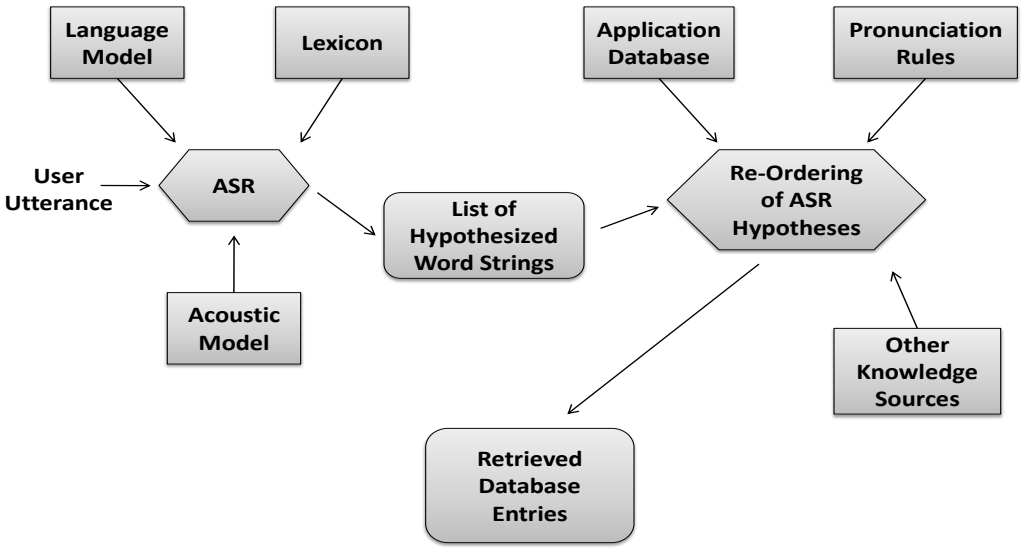

Fig. 1.1 Block diagram of Re-Ordering ASR Hypotheses

The block diagram shown in Figure 1.1 provides a description of the steps involved in obtaining a retrieved database entry in response to a user utterance containing a spoken query. First, the ASR engine decodes the user utterance into list of hypothesized word strings. The ASR system is assumed to be "trust independent" with statistical acoustic model parameters trained for utterances from mobile devices and transmitted on a wireless communication channels. In a second pass, these word hypotheses are re-ordered by exploiting additional knowledge sources including application database, the linguistic knowledge (pronunciation rules) and others. The goal is for mobile voice search to become more robust and be able to provide more precise results from the specific database associated with a particular application. The additional knowledge sources provide application specific constraints that are applied to the word string hypotheses obtained in the first pass decoding. These constraints can be applied using many different formalisms. The work described in this thesis is mainly concerned with instances of spell mode utterances. Techniques are investigated for improving performance of recognizing spelled letters in the first pass and associating these, possibly errorful, spelled word string with relevant database entries in the second pass. For spell mode utterances, the re-ordering process is essentially

a sequence alignment. This consists of transforming one sequence (ASR hypothesis) into another (retrieved database entry) [5].

1.3 Thesis Outline

This section provides an outline of this thesis document. To present the methodology used to generate and assess a multiple pass re-ordering process, the rest of this thesis has been divided into four chapters.

In Chapter 2, important concepts that are used throughout this thesis are introduced. In particular, the components of a typical ASR system will be introduced along with dynamic programming and multiple pass search techniques. Grapheme to phoneme estimation techniques will be introduced for converting from text representations of words to acoustic phonetic expansions. A technique for combining multiple statistical models using a discriminative approach is presented. Additionally it summarizes the literature published in the area of voice web search and ASR for spell mode utterances.

In Chapter 3 the design of multiple pass re-scoring process for identifying the most likely database entry associated with the utterances that contain spoken letters is described. The implementation of efficient similarity measures is presented as they are used in matching decoded N-best letter string hypotheses obtained from the ASR system with entries in the textual database. An overall explanation of how the entire system (efficient grapheme/phoneme search) is integrated is presented in detail. Finally, the application of discriminative model combination (DMC) for obtaining an optimum linear combination of the log probabilities from the multiple knowledge sources given in Figure 1.1 is described.

Chapter 4 describes the performance of the multiple pass re-scoring process. An experimental study is presented describing the characteristics of actual user utterances obtained from a prototype voice search service. The impact of the multiple pass re-scoring process on word error rate is presented. Based on the experiment results, it is believed that the multiple pass spoken term retrieval process improves the recognition accuracy and word retrieval accuracy for utterances containing spoken letter sequences.

Finally Chapter 5 concludes the design, implementation, experimental study and suggests possible future work. It is believed that work on improving spell mode utterance recognition, specifically using large textual database entries as external knowledge sources can be very helpful in terms of improving efficiency and accuracy in many other applications.

Chapter 2

Background

This chapter provides an introduction to the theoretical tools that are used in this thesis work. The development will be at a level so it is clear how these techniques are applied to the overall approach discussed in Figure 1.1. A review of recent literature in voice web search will also be presented.

A introduction of the basic components of a typical ASR system will be provided in Section 2.1 including feature analysis, acoustic models and language models. In Section 2.2, dynamic programming for string alignment will be presented in detail. This technique is utilized in this thesis work to look for the best database entry. Later in this chapter, Section 2.3 discusses multiple pass search and its application in ASR research. Then Section 2.4 presents two grapheme to phoneme estimation algorithms which are used for producing phonemic expansions of spelled letter sequence. The two algorithms include N-Gram based and Decision Tree based estimations. After that, in Section 2.5 a discriminative method for log-linear model combination will be presented. This method will be used in obtaining a log-linear weighted combination of different model probabilities obtained from multiple pass re-ordering process. Finally Section 2.6 presents a literature review of voice web search and ASR for spell mode utterances, and it also discussed the limitations of ASR when applied to spell mode recognition.

2.1 A typical ASR system

This section will present a brief introduction of ASR. ASR refers to a system which converts spoken words to text. The automatic speech recognition system is based on statistical models of acoustics and language trained from a large population of speakers to enable speaker independent operation. Common commercial ASR applications include "Voice search" for mobile devices and "Call routing" services for automating call center operations. A typical ASR system usually consists of several key components including feature analysis, acoustic model and language model. Each of these components will be introduced in this section.

Overall speaking, speech recognition is a statistical problem of deriving the word sequence that has the highest likelihood of corresponding to the observed sequence of acoustic vectors [6].

$$\hat{W} = \arg\max_{W} P(W|X) \tag{2.1}$$

Here, $W = w_1, w_2, \cdots, w_S$ is a sequence of S words and $X = x_1, x_2, \cdots, x_T$ is a sequence of X acoustic observation vectors. The above equation may be read as $\hat{W}$ is the particular word sequence W which has a maximum posterior probability given the observation sequence X. Using Bayes' Rule, the equation maybe re-written as:

$$\hat{W} = \arg\max_{W} \frac{P(X|W)P(W)}{P(X)} \tag{2.2}$$

$P(X|W)$ denotes the probability of the acoustic vector sequence X given the word sequence W. $P(W)$ denotes the probability with which the word sequence W occurs in the language, which will be introduced later. $P(X)$ denotes the prior probability of the acoustic sequence. $P(X)$ is independent of the word sequence, therefore $\hat{W}$ can be computed without knowing $P(X)$. Thus the above equation may be re-written as:

$$\hat{W} = \arg\max_{W} P(X|W)P(W) \tag{2.3}$$

The set of signal processing algorithms that convert the speech signal into the acoustic vector sequence X is commonly referred to as the $front\ end$, or $feature\ analysis$. The quantity $P(X|W)$ is generated by evaluating an *acoustic model*. The term $P(W)$ is generated from a *language model*.

2.1.1 Feature Analysis

Feature analysis is one of the principal components in speech recognition process which is to provide accurate and reliable parameters to the ASR system for subsequent acoustic matching [6]. Although choosing the appropriate feature set depends on the task being performed, the most widely used ASR feature extraction techniques, are based on filterbank and linear predictive models of the speech spectrum [6]. The signal processing front end reduces the spectral characteristics of the speech waveform into a sequence of acoustic vectors that are suitable for processing by the acoustic model.

The Mel-Frequency Cepstrum (MFC) is a filterbank based technique that is used as feature anallysis method in this thesis work. The Mel-Frequency Cepstrum Coefficients (MFCCs) are frequency-domain features which have been proven to be beneficial for speech recognition [7]. The feature extraction procedure assumes that the speech waveform is wide-sense stationary over short intervals of approximately 10-25 msec. In order to compute the cepstrum coefficients of speech segments, a short-time Hamming window usually slides along the speech signal. Each windowed speech segment is pre-emphasized and goes through a Fast Fourier Transformation (FFT) based spectrum analysis which computes the discrete spectrum of input segment. The spectrum magnitude is then being processed by using a filterbank with M triangular filters. The average spectral energy is derived around the center frequency of each filter [6]. The center frequency and the boundary points of these filters are linearly spaced in a non-linear Mel-Frequency scale. This Mel-Frequency scale is developed based on human auditory perception and on the other hand approximately models the sensitivity of human ears [6].

In addition to feature sets derived from speech spectrum, articulatory based features become a hot topic in recent years. An articulatory based representation characterizes the underlying state of the speech production system. In fact articulatory based features can be obtained from acoustic correlates of articulatory phenomena, for example nasality, voicing or place of articulation[8] [9].

2.1.2 Acoustic Model

An acoustic model is created by taking audio recording of speech and their transcriptions which are usually taken from a speech corpus, and compiling them into a statistical representations of the sounds that make up each word [6]. In practice the acoustic model evaluates the similarity between an input feature vector sequence and a set of acousitc word models to determine what words were most likely spoken. Equation 2.3 needs the quantity $P(X|W)$, the probability of an acoustic vector sequence X given a word sequence W, to find the most probable word sequence.

The statistical models are trained for sub-word units. The subword models are the representation of the fundamental speech units used as the building blocks for words, phrases, and sentences. The most widely used set of subword units describe phonemes in context. An example of the definition of sub-word unit is the tri-phone which is a phoneme in the context of the phonemes that occur to the left and the right of the phoneme [10]. The most common modeling formalism used is the hidden Markov model (HMM). In English phonology, there are usually 40-50 phoneme symbols [11] [12]. The speech units are modeled left-to-right, continuous density hidden Markov models, and within each state of the HMM, the random spectral vecotr is represented by a Gaussian mixture density [10]. The training of subword unit models consists of estimating the HMM parameters from a labeled training set of continuous speech utterances where all of the relevant subword units are known to occur sufficiently often. Different ways in which training is performed affects greatly the overall recognition performance.

2.1.3 Language Modeling

Language models describe the underlying structure of sentences in a language and pronunciation models describe the acoustic realization of this underlying structure. An ideal language model which has a broad coverage of the formal grammar could play the role of syntax and semantics of a language for speech recognition systems. However, for ASR applications where naturally spoken language is in particular interest, taking a probabilistic approach is more efficient. To achieve this purpose, statistical language models (SLMs) are employed in which the probabilistic relationship between a sequence of words is modeled [6].

Bayes decision rule, used in typical ASR systems to classify speech segments as belonging to different word or phoneme classes, justifies the probabilistic approach for language modeling in ASR. If $X = x_1, x_2, ..., x_T$ represents a sequence of acoustic observation vectors and $W = w_1, ..., w_S$ represents a word or phoneme sequence, the observation sequence X can be labeled as the word sequence W if:

$$\begin{aligned} \forall W' \neq W : & logp(W|X) - logp(W'|X) \geq 0 \\ & log[p(W)p(X|W)] - log[p(W')p(X|W')] \geq 0 \end{aligned} \tag{2.4}$$

In above equation, $p(X|W)$ represents the acoustic model probability and $p(W)$ represents the prior probability of generating the corresponding word or phoneme string and is called the language model probability.

However, to be able to compute the prior probability of any word or phoneme string $p(W)$, the input utterances have to be broadly covered by the language model which is not possible except for very constrained application domains. Moreover, spoken language does not usually follow grammatical rules. Statistical language modeling is a technique which provides ASR systems with a probabilistic description of the language. The basic goal is for the language model probability to accurately reflect occurrence of word strings in actual utterances.

A statistical language model (SLM) controls word recognition in an application. It is automatically generated from a training file, and it improves recognition accuracy by describing the probability of each valid utterance. A very simple form of an SLM is a list of words with probabilities assigned to each; this is commonly called a unigram. A unigram is overly simplistic model for many reasons, primarily because the probability of a word is independent of its position in a sentence. In this model, for example, a sentence such as "go to website G O O G L E" is considered as likely as any permutation of its words, such as "G O O go to L E website". In contrast, an N-Gram SLM which will be discussed later, is one in which the probability of a word depends on the previous $N-1$ words. The value N is called the order of the model.

N-Gram Language Models: Computing the prior probability of a given phoneme or word sequence, $p(W)$, corresponds to determining the frequency of uttering that word sequence as a sentence. If $W = w_1, ..., w_S$ represents the sequence of words spoken, $p(W)$ can be written as:

$$\begin{aligned} p(W) &= p(w_1, w_2, ...w_S) \\ &= p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_S|w_1, w_2, ...w_{S-1}) \\ &= \prod_{i=1}^{S} p(w_i|w_1, w_2, ..., w_{i-1}) \end{aligned} \tag{2.5}$$

where $p(w_i|w_1, w_2, ..., w_{i-1})$ is the probability of speaking word w_i given that the word sequence $w_1, w_2, ..., w_{i-1}$ has been previously uttered. However for a vocabulary size of υ, complete specification of all possible $p(w_i|w_{i-1}, w_{i-2}, ..., w_{i-k})$ requires the estimation of υ^k probabilities. In practice, even for average values of i, estimating all of the probabilities from a finite corpus is not achievable. A practical solution to this problem is taking advantage of N-gram language models. In N-gram language models, it is assumed that each word only depends on $N-1$ previous words and not on the entire past history. The trigram language model, where each word only depends on the previous two words, is particularly powerful. This is due to the fact that there usually exists a strong dependency between a word and the previous two words and second, given a training corpus of reasonable size, trigram probabilities can be robustly estimated. The trigram estimation is based on maximum likelihood principle which is computed by counting the frequencies of the word pair $C(w_{i-2}, w_{i-1})$ and triplet $C(w_{i-2}, w_{i-1}, w_i)$ in a given training corpus as follows:

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} \tag{2.6}$$

Using the probabilities assigned to each triphone, the prior probability of every given word sequence can be computed based on the above equation. Even with this constrained model, it is still necessary to smooth the estimated probabilities. For this reason, there has been a large amount of research done to explore the advantages of using different smoothing methods [6].

At any moment in a dialog, some sentences are more likely to be spoken than others.

An SLM models what users might say by defining a vocabulary of acceptable words, and describing the probability that a given word will be used based on other words in a sentence. For example, utterances beginning with the phrase "go to website" might be completed with any combination of words or spelled words constituting a website name. For example, spell letter sequences like the following may be uttered:

- "G O O G L E"
- "N U A N C E"
- "Y A H O O"

If the SLM has been trained using letter sequences obtained from text from a similar domain, "well formed" letter sequences like these would be more likely than misspelled words or nonsense words. In theory, an SLM may still accept any combination of vocabulary words even if they occur in an order that does not make sense. For example, it could accept the nonsense phrase: "N N U U A N C E"; words in the vocabulary can be recognized at any time, in any combination. However, in practice, the SLM likelihood for this letter sequence should be far lower than a well formed letter sequence and, if this is one of many string candidates obtained from an ASR system, this string would be rejected.

Overall speaking, SLMs get started on the understanding free-style speech by recognizing an wider range of user utterances. An SLM alone can confirm whether the user said something recognizable. However, recognizing the utterance is only half the battle. The other half is interpeting the meaning of the utterance. SLMs can be used alone, however they may need to be combined with a hand tuned grammar when there is insufficient training from a particular domain. In Nuance's voice web search service where the ultimate goal is to literally rather than semantically understand users' query in which may contain ungrammatical sentence, disfluencies (e.g., repeated words, repairs or false starts), it had better combine the SLM with another grammar, for example a robust parser which may be robust to both ill-formed spoken language and under-specified semantic grammars. Such a combination provides powerful natural language understanding. With this combined approach, people can use a single, generalized language model with more than one robust parsing grammar: the SLM handles the word combinations, and the robust parser interprets meanings in response to different situations and prompts.

To address this problem, a robust natural language (NL) parsing capability has been introduced which only consider the meaningful phrases in an utterance, ignoring the parts that don't matter [13]. Typically natural language interpretation has 2 modes: *full* and *robust*. In the conventional operating mode, the full mode, the recognition engine and the NL engine are driven by the same grammar. This grammar both defines the valid phrases. In robust mode, it is possible to use two different grammars. The first grammar drives the recognition phase and the second drives the interpretation phase. For example, the recognition could use an SLM grammar, allowing the application to recognize a large range of user's speech. The text output by the recognition engine could then by processed by the NL engine running a grammar that parses certain phrases only–the meaningful ones [14]. For example, consider a grammar for a typical flight reservation application and the following user's sentence: *I'd like to um I want to go to boston tomorrow*. The speech recognition engine, driven by the SLM, recognizes the sentence and sends the result to the NL engine, which tries to interpret the text. In full mode, the NL engine would not parse the text, because the sentence cannot be completely parsed by the grammar; consequently, it would reject the sentence. However, in robust mode, the engine could ignore the babbling at the beginning of the sentence and fill the destination and day slots with *boston* and *tomorrow*, respectively.

2.2 Dynamic Programming

According to Figure 1.1, after the ASR system decodes the input user utterance and generates a list of hypothesized word strings, there is a second pass re-ordering process which takes advantage of external knowledge sources and re-orders the list of ASR hypotheses. This is done by evaluating the similarity of sequences associated with ASR hypotheses and sequences generated by the knowledge sources in the second stage. To achieve this, a score that reflects how similar the two sequences are will be calculated. Similarity is defined in terms of a weighted count of symbol substitutions, insertions, and deletions. The goal of the re-scoring procedure in the second stage is to optimally align the hypothesized sequences with the reference sequences and to compute their similarity.

Dynamic programming [15] is a method of solving complex problems by breaking them

down into simpler steps. It is applicable to problems that exhibit the properties of overlapping subproblems and optimal substructure. It is both a mathematical optimization method, and a computer programming method. In both contexts, it refers to simplifying a complicated problem by breaking it down into simpler subproblems in a recursive manner.

In a sequence alignment example, it is very difficult to score all possible alignments and pick the best one. This is not practical because according to [16], there are about $\frac{2^{2N}}{\sqrt{\pi N}}$ different alignments for two sequences of length N. Assume the two sequences x and y, they are of length M and N letters respectively. The i^{th} letter in x is x_i and the j^{th} letter of y is y_j. An efficient way to process the alignment is to build a cost matrix $\sigma(a, b)$ for aligning two letters a, b to each other, and a insertion cost γ_i or deletion cost γ_d for every time such an error takes place.

A dynamic programming algorithm consists of four parts: a recursive definition of the optimal score; a dynamic programming matrix for remembering optimal scores of subproblems; a bottom-up approach of filling the matrix by solving the smallest subproblems first; and a traceback of the matrix to recover the structure of the optimal solution that gave the optimal score.

- Recursive definition of the optimal score

The problem breaks into independently optimizable pieces, as the scoring system is strictly local to one aligned column at a time. That is for instance, the optimal alignment of $x_1 \cdot x_{M-1}$ to $y_1 \cdot y_{N-1}$ is unaffected by adding on the aligned letter pair x_M, y_N and likewise the score $s(x_M, y_N)$ being added on is independent of the previous optimal alignment. Thus a general recursive definition of all our partial optimal alignment scores $S(i, j)$ can be written as:

$$S(i,j) = max \begin{cases} S(i-1, j-1) + \sigma(x_i, y_j) \\ S(i-1, j) + \gamma_d \\ S(i, j-1) + \gamma_i \end{cases} \tag{2.7}$$

- The dynamic programming matrix

Some subproblems are already occurring more than once and this wastage gets exponentially worse as it moves deeper into the recursion. Clearly one sensible thing to do is to keep track of which subproblems have already been operated on. This is the key difference between dynamic programming and simple recursion: a dynamic programming algorithm memorizes the solution of optimal subproblems in an organized tabular form, called a dynamic programming matrix, so that each subproblem is solved just once.

For the pairwise sequence alignment algorithm, the optimal scores $S(i, j)$ are tabulated in a two dimensional matrix with i running from $0 \cdots M$ and j running from $0 \cdots N$. And for subproblems $S(i, j)$, their optimal alignment scores are stored in the appropriate (i, j) cell of the matrix.

- A bottom-up calculation to get the optimal score

Once the top row and left column of $S(i, j)$ is initialized, the rest of the matrix can be filled in by using the recursive definition to calculate any cell whose values are known. The calculation operates on three adjoining cell to the upper left $(i-1, j-1)$, above $(i-1, j)$, and to the left $(i, j-1)$. There are several different ways to do this. One is to iterate two nested loops, $i = 1 \cdots M$ and $j = 1 \cdots N$, so the matrix is filled up from left to right and from top to bottom.

- A traceback to get the optimal alignment

Once the matrix is completed, the score of the optimal alignment of the complete sequences is the last score calculated $S(M, N)$. But the optimal alignment itself is not known yet. Thus a recursive *traceback* of the matrix is used. Starting in cell (M, N), it determines which of the three cases used to get the given cell. It records that choice as part of the alignment, and then it follows the appropriate path for that case back into the previous cell on the optimum path. It keeps doing that one cell in the optimal path at a time, until cell $(0, 0)$ is reached, at which point the optimal alignment is fully reconstructed.

Dynamic programming is guaranteed to give the minimum cost difference between the strings. However when the two strings are too long that there are too many alignment possibilities, dynamic programming becomes computationally demanding. Filling in the matrix

requires computation proportional to MN which may be unacceptable for some applications. This is why there is so much research devoted to finding good, fast approximations to dynamic programming alignment [17].

2.3 Multiple pass Search

An ideal search algorithm for speech recognition integrates all knowledge sources including acoustic, lexicon, and language knowledge in the search. The search complexity introduced with increasing vocabulary size, large n-gram language models, and other complicated knowledge sources, will not always allow the simultaneous integration of all possible information in a single-pass search algorithm [6].

An alternative to the single-pass search exists: integrating multiple knowledge sources at different stages. First of all, a first-pass search is performed, then in following stages new sets of knowledge sources can be used to further constrain the search space generated in the first pass. However, the order in which knowledge sources have been implemented is important; discriminant knowledge sources which are believed to retain more important information and are easier to employ in the first pass are initially used to constrain the search space to a limited number of hypotheses. Afterward, more powerful sources of knowledge can be applied in order to find the optimal transcription among the available hypotheses. One point to be made here is that since the correct transcription may be wrongly pruned due to lack of higher level information in the initial passes, the multiple pass search algorithm is not guaranteed to provide us with the optimal transcription. In other words, the multiple pass search algorithm is not admissible. However, the reduction in computation complexity gained by using the multiple pass search represents such an important advantage that the non-admissibility of the algorithm is generally accepted in practice [6] [18].

One widely used multiple pass search strategy in ASR involves the first recognition pass generating a list of the N-best scoring word strings. These hypothesized strings can then be re-ordered by re-scoring the hypotheses with an additional knowledge source as part of a second recognition pass.

A more compact and efficient way for characterizing the result of the first pass search is the lattice. A word lattice is a finite state network that consists of word hypotheses as labels on the arcs while nodes represent inter-word transitions.

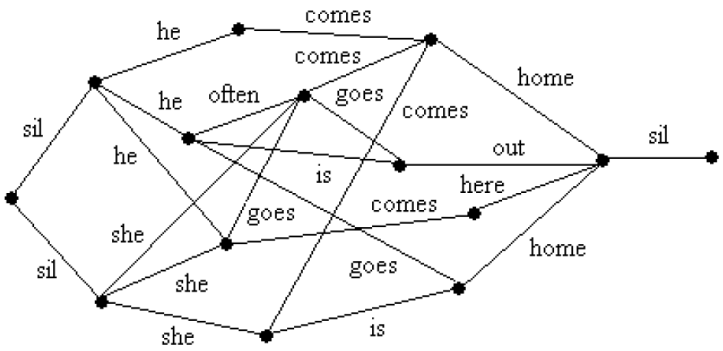

Fig. 2.1 A simple word lattice

Figure 2.1 shows a typical example of a word lattice in which each arc contains a word label and each node corresponds to a transition between words. Arcs may also contain weights that correspond to the acoustic and language probability for the word. Nodes may also contain the time instant associated with the word transition.

To generate ASR lattices, a simple modification is required to be done on the forward-backward search algorithm [6]. First, the modified forward probability $\alpha_t(w)$ is computed using a forward time synchronous Viterbi search. This modified forward probability is the probability of observing the word label w ending at time t. When the forward search is performed, the best score a^T is stored to be used later for pruning. Next, a backward time-synchronous Viterbi search is done and the modified backward probability $\beta_t(w)$ is estimated. The modified backward probability is defined as the probability of observing the word label w beginning at time t. To generate the lattice, the forward-backward probability is checked at each time instant t. If the probability is inside the pruning beam width, the word transition $w_i - w_j$ is included in the word lattice. This condition is illustrated by the following equation for word w_i ending at time t and word w_j beginning at time t:

$$\alpha_t(w_i) + \beta_t(w_j) + [-logP(w_j|w_i)] < \alpha^T + \theta \tag{2.8}$$

where $P(w_j|w_i)$ is the language model probability and θ is the pruning threshold. The

lattice generation is continued backward in time by looking for the next word pair until the search reaches the beginning of the utterance.

2.4 Grapheme to Phoneme Estimation

According to Figure 1.1, grapheme to phoneme estimation can be used for generating phonetic expansions of word strings. These phonetic expansions can then be used as an added knowledge source for re-ordering ASR hypotheses. Grapheme to phoneme estimation obtains a phonemic expansion associated with the pronunciation of a given letter string. It it used here for obtaining a phoneme expansion of a spelled letter sequence contained in an ASR word string hypothesis. These phoneme expansions will be compared with phonemic expansions of database. In this section, the motivation of grapheme to phoneme estimation in an ASR system will be presented and then two main grapheme to phoneme estimation algorithms will be introduced including N-Gram based estimation and Decision Tree based estimation.

2.4.1 General Purpose of Grapheme to Phoneme Estimation

In large vocabulary speech recognition systems, the pronunciation of a word is usually given by a phonemic transcription of the word. This transcription can be obtained from large background lexicons for general purpose words [19]. But in some cases there is no entry for a word in the background lexicon. Especially databases for proper names are usually not large enough to be complete and also special vocabulary might not be covered by a general purpose background lexicon. Another situation where transcriptions are not available for all words is during the initial construction of a background lexicon. In all cases an algorithm is desirable which automatically generates a transcription of a word from its spelling only. This works obviously only if the spelling of a word is linked to its pronunciation. This is usually true for languages using the Roman alphabet, but it is not as obvious for languages using e.g. Chinese characters [20].

Different approaches for getting a phonemic transcription from the spelling of a word have been proposed and implemented [21] [22] [23]. The class of rule based algorithms uses manually written rules to find a transcription [24]. The other class of algorithms is data driven. They propose a probabilistic model and estimate the free model parameters from

training data. A phonemic expansion is obtained for a word by taking the most probable transcription. Among those algorithms, a popular one is the N-Gram approach [25] [26]. Another frequently described approach is based on decision trees [27] [28].

2.4.2 N-Gram based G2P Estimation

An automatic grapheme-to-phoneme algorithm is based on the Bayes decision rule with an N-Gram approximation. Following Bayes decision rule, the sequence $p_1 \cdots p_N$ of phoneme units should be selected which gives the highest conditional probability given the grapheme unit sequence $g_1 \cdots g_N$ of the word. Rewriting the conditional probability and ignoring terms that are constant to the maximization:

$$\begin{aligned} \hat{p} &= \arg\max_{p_1 \cdots p_N} Pr(p_1 \cdots p_N | g_1 \cdots g_N) \\ &= \arg\max_{p_1 \cdots p_N} Pr(p_1 \cdots p_N, g_1 \cdots g_N) / Pr(g_1 \cdots g_N) \\ &= \arg\max_{p_1 \cdots p_N} Pr(p_1, g_1 \cdots p_N g_N) \end{aligned} \tag{2.9}$$

where the $\hat{p}$ refers to the optimum phoneme sequence. Now the joint probability may be re-written as product of conditional probabilities.

$$Pr(p_1, g_1 \cdots p_N g_N) = \prod_{i=1}^{N} Pr(p_i, g_i | p_1, g_1 \cdots p_{i-1} g_{i-1}) \tag{2.10}$$

If we approximate those conditional probabilities by n-grams we obtain the N-Gram model:

$$Pr(p_i, g_i | p_1, g_1 \cdots p_{i-1} g_{i-1}) \approx P_{ng}(p_i, g_i | p_{i-n+1}, g_{i-n+1} \cdots p_{i-1} g_{i-1}) \tag{2.11}$$

There are different possibilities to estimate the N-Gram probabilities. In this thesis a simple linear interpolation of the maximum likelihood estimators $P_{ng(i)}$ of the lower order N-Gram is used:

$$P_{ng} = \prod_{i=0}^{n} \alpha_i P_{ng(i)} \tag{2.12}$$

The sum of the weights α_i must be equal to 1.

The training of an N-Gram model amounts to counting and efficiently storing of all N-Grams occurring in the aligned training lexicon. The decoding for the N-Gram model is performed by a "time-synchronous" beam search. The strength of the N-Gram approach is the relatively high accuracy, especially for irregular languages as e.g. English. This is achieved by a huge number of parameters, which is one of the major disadvantages of this algorithm in commercial real-time application, e.g. Nuance's voice web search service [26] [23].

Tunable model parameters comprise the maximal considered context length n, the interpolation weights and the probability threshold for discarding grapheme-phoneme sequence pairs. Once the model is trained, the pruning threshold for the beam search is the only remaining tunable parameter.

2.4.3 Decision Tree based G2P Estimation

The approach to automatic grapheme-to-phoneme transcription which is most commonly described in literature and which does not rely on hand-written rules is decision trees based G2P [27] [28]. Decision trees are very much like the rules used in the previous section and there are well understood algorithms to train those trees automatically.

Assuming a binary tree is considered, each node represents a set of samples where a sample is a grapheme-unit phoneme-sequence pair from the aligned training lexicon. The information about the left and right context of this production is still available.

For example, the word "*more*" with the phonemic expansion "$m\ O:\ r$" would present 4 grapheme to phoneme mappings: 1. $m \rightarrow$ "m"; 2. $o \rightarrow$ "$O:$"; 3. $r \rightarrow$ "r"; 4. $e \rightarrow$ "" ($null$). In practice, the mapping takes context letters into account. For example, the 2nd mapping refers to take grapheme "o" with left context letter "m" and right context letter "r", and then map into phoneme "O:".

A question about the context of the letters is associated with each non-leaf node and the two child nodes represent the positive and negative cases to this question. Using a decision-tree an individual sample S can now be classified. The tree will be traversed starting at the

root by answering for the sample the associated questions and selecting the appropriate children. Finally a leaf-node $C(S)$ will be reached. For each node the probability of a certain phoneme-sequence $p_1 \cdots p_n$ being produced given that node will be estimated. The maximum likelihood estimate of the probability of this phone sequence in this node C is:

$$P_{dt}(p_1 \cdots p_n | C) = N(p_1 \cdots p_n, C)/N(C) \tag{2.13}$$

where $N(C)$ is the total number of samples in the node C and $N(p_1 \cdots p_n, C)$ is the number of samples in the node producing the phonemes sequence $p_1 \cdots p_n$.

A common criterion for how well the training data (represented by samples S with associated phoneme sequences $p(S)$) is the entropy h:

$$\begin{aligned} h: &= -\sum_S log P_{dt}(p(S)|C(S)) \\ &= -\sum_S log N(p(S), C(S))/N(C(S)) \\ &= -(\sum_C \sum_p N(p,C) log N(p,C) - \sum_C N(C) log N(C)) \end{aligned} \tag{2.14}$$

Fortunately this criterion is in the sense of local nature, that for calculating the difference between the original entropy h and the entropy h' obtained by a split of one certain node X into two new nodes X_1 and X_2, only the samples of these nodes must be taken into account:

$$\begin{aligned} h - h' = &\sum_p N(p, X_1) log N(p, X_1) - N(X_1) log N(X_1) \\ &+ \sum_p N(p, X_2) log N(p, X_2) - N(X_2) log N(X_2) \\ &- \sum_p N(p, X) log N(p, X) + N(X) log N(X) \end{aligned} \tag{2.15}$$

The basic idea of the decision-tree training is to find a tree which describes the training data best. Starting with some binary tree the tree is grown by successively splitting a leaf-node using a question from a set of given questions. At each split that leaf-node and that question are selected which results in the largest gain of the optimization criterion h. The considered questions usually ask for a grapheme unit at a relative position, but also combined questions and questions for a phoneme sequence may be used. Several parameters

control the size of the tree. First the maximal number of nodes of the tree may be specified. Furthermore, it can be specified how many samples a node must have in order to be considered for splitting and how many nodes must remain at least in each node after a split.

Once a decision tree is trained, transcribing a word is very simple. The individual grapheme units of the word are interpreted as samples and the node in the decision tree belonging to this sample is searched (in the case of phoneme questions, the already transcribed phonemes must also be taken into consideration). The most probable phoneme sequence of the found node is finally taken as transcription of the grapheme unit.

The decision tree approach is very fast and has only a small amount of parameters [28]. It achieved good transcription accuracy, also the size of decision tree is under control to suit the computational constraints in speech recognition systems.

2.5 Model Combination Using a Discriminative Approach

In Figure 1.1, the second pass re-ordering process takes advantage of external database associated with the specific decoded utterance, as well as linguistic pronunciation rules and other knowledge sources. An approach pursued in automatic speech recognition for optimal integration of multiple models is *Discriminative Model Combination* (DMC). Using this paradigm, the models are integrated into one general log-linear posterior probability distribution [29]. To justify the model integration approach introduced by DMC, it can be started with the application of combining an acoustic model with a language model.

Given an observation vector x, the so-called Bayes decision rule is usually applied for the purpose of classification. The observation vector belongs to class k if:

$$\forall k' = 1, ..., K; k' \neq k : log\pi(k|x) - log\pi(k'|x) \geq 0 \tag{2.16}$$

where $\pi(k|x)$ is the posterior probability of class k given observation vector x . The function $g(x, k, k') = log(\pi(k|x)/log\pi(k'|x))$ is called the discriminant function. Since the posterior probabilities $\pi(k|x)$ in Equation 2.16 cannot be estimated directly, they are usually approximated using some probability distribution.

Thus if $x_1^{T_n}(n) = (x^n(n), ..., x^{T_n}(n))$ for $n = 1, ..., N$ and $w_1^S = (w^1, ..., w^S)$ represent a sequence of observation vectors and the corresponding word sequence respectively, to apply the above decision rule to ASR, Equation 2.16 can be rewritten as follows:

$$\forall {w'}_1^{S'} \neq w_1^S : logp(w_1^S|x_1^{T_n}(n)) - logp({w'}_1^{S'}|x_1^{T_n}(n)) \geq 0 \tag{2.17}$$

Consequently, the discriminant function will be equal to:

$$\begin{aligned} g(x_1^{T_n}(n), w_1^S, {w'}_1^{S'}) &= logp(w_1^S|x_1^{T_n}(n)) - logp({w'}_1^{S'}|x_1^{T_n}(n)) \\ &= log[p(w_1^S)p(x_1^{T_n}(n)|w_1^S)] - log[p({w'}_1^{S'})p(x_1^{T_n}(n)|{w'}_1^{S'})] \end{aligned} \tag{2.18}$$

where $p(w_1^S)$ is the language model probability and $p(x_1^{T_n}(n)|w_1^S)$ represents the acoustic model probability. However, since these probabilities do not provide us with the true distributions, the classifier will deviate from the optimal one. To address this problem, a discriminative method can be used. One well known example of applying a discriminative method is the estimation of the language model scale factor λ which is used to compensate for the inherent difference between acoustic and language model probabilities. This results in a discriminant function of the following form:

$$g(x_1^{T_n}(n), w_1^S, {w'}_1^{S'}) = log[p(w_1^S)^{\lambda} p(x_1^{T_n}(n)|w_1^S)] - log[p({w'}_1^{S'})^{\lambda} p(x_1^{T_n}(n)|{w'}_1^{S'})] \tag{2.19}$$

Based on this general model, now assume that M different acoustic-phonetic and language models are going to be used to find the best word sequence matching the given observation vectors. The combined log-linear posterior probability distribution of M different acoustic and language models, $p_j(w_1^S|x_1^{T_n}(n)), j = 1, ..., M$, will have the following form:

$$p_{\Lambda}^{\Pi}(w_1^S|x_1^{T_n}(n)) = exp\{logC(\Lambda) + \sum_{j=1}^{M} \lambda_j logp_j(w_1^S|x_1^{T_n}(n))\} \tag{2.20}$$

where $\Lambda = \lambda_1, ..., \lambda_M$ and $C(\Lambda)$ is a normalization factor.

In the DMC approach, the coefficient Λ are optimized based on the decision error rate

of the following discriminant function:

$$g(x_1^{T_n}(n), w_1^S, {w'}_1^{S'}) = \sum_{j=1}^{M} \lambda_j (logp_j(w_1^S|x_1^{T_n}(n)) - logp_j({w'}_1^{S'}|x_1^{T_n}(n))) \tag{2.21}$$

To solve the above optimization problem, first we define the "ideal" discriminant function as $f(\mathscr{L}(k_{nk}, k_{n0}))$ where $\mathscr{L}(k_{nr}, k_{n0})$ is the $Levenshtein - distance$ between the correct phoneme sequence k_{n0} and the competing phoneme sequence of k_{nr}, and $f(.)$ is a sigmoid function. Since the discriminant function given in Equation 2.21 actually represents the class boundaries, optimizing the coefficients, Λ, of log-linear model combination involves minimizing the mean squared distance between this discriminant function and the "ideal" discriminant function on the training data to satisfy the DMC criteria. Equation 2.22 shows the mean squared error function to be minimized:

$$D(\Lambda) = \frac{1}{K \times N} \sum_{n=1}^{N} \sum_{k=1}^{K} (log \frac{p_\Lambda^\Pi(k_{n0}|x_1^{T_n}(n))}{p_\Lambda^\Pi(k_{nk}|x_1^{T_n}(n))} - f(\mathscr{L}(k_{nk}, k_{n0})))^2 \tag{2.22}$$

To optimize the mean squared distance, $D(\Lambda)$, the values λ_j can be computed by taking the derivative of $D(\Lambda)$ with respect to Λ and setting it equal to zero. The final closed-form solution is given as,

$$\Lambda = Q^{-1}P \tag{2.23}$$

where Q is the $M \times M$ autocorrelation matrix of the discriminant functions of the M different models and P is the correlation vector between the discriminant functions of the M given models and the sigmoid function of the Levenshtein-distance $f(\mathscr{L}(k_{nk}, k_{no}))$,

$$Q_{ij} = \frac{1}{KN} \sum_{n=1}^{N} \sum_{k=1}^{K} \{log \frac{p_i(k_{no}|x_1^{T_n}(n))}{p_i(k_{nk}|x_1^{T_n}(n))}\} \{log \frac{p_j(k_{no}|x_1^{T_n}(n))}{p_j(k_{nk}|x_1^{T_n}(n))}\}, \qquad (i, j = 1, ..., M) \tag{2.24}$$

$$P_i = \frac{1}{KN} \sum_{n=1}^{N} \sum_{k=1}^{K} \{log \frac{p_i(k_{no}|x_1^{T_n}(n))}{p_i(k_{nk}|x_1^{T_n}(n))}\} f(\mathscr{L}(k_{nk}, k_{n0})), \qquad (i = 1, ..., M) \tag{2.25}$$

Using the above equations, the optimal weights for the log-linear model combination will

be estimated where each estimated weight depends on the correlation of its discriminant function to the ideal discriminant function and also to the discriminant functions of all the models. The feature integration scheme presented in Chapter 3 is optimized using this discriminative model combination approach.

2.6 Previous Work Done On Voice Web Search

This section is written to give an overview of previous work done on voice web search and particular in ASR of spell mode utterances. It will firstly introduce recent research work in automated voice search services on mobile devices [30] [31] [32] [33]. After that current research in ASR for spell mode utterance will be discussed.

2.6.1 Voice Search on Mobile Devices

A. Franz and B. Milch [4] investigated language modeling techniques for ASR in a prototype voice search system. Specifically, they explored a method to increase the predictive power of the unigram model by adding collocations of words which is any sequence of words that co-occurs more often than chance to its vocabulary. And according to [34], a collocation is defined as an expression of two or more words that corresponds to some conventional way of saying things.

More recently, the multi-modal and hand-held nature of cellphone communication creates numerous challenges as well as opportunities for speech-enabled search applications. For example the primary problem is the design of the user interface: users are not initially aware of the capabilities and limitations of speech recognition, and an intuitive interface must be provided for displaying information and soliciting feedback from the user. Acero et al [3] designed and implemented Windows Mobile Live which allows users to interact with web information portals. Their work includes integrating voice-enabled data input functionality with search functionality to provide local-business related information. The key challenge encountered in [3] is the low signal-to-noise ratio (SNR) caused by the fact that users often speak at arms length while looking at the screen, which in turn causes the ASR word error rate to increase. According to [3], utterances with low ASR scores are forwarded to human transcribers. A retrieval accuracy of 43% was obtained using the fully automated system when the user is given the option to select from the nine highest scoring

entries retrieved by the search engine.

Keith el al [31] investigated how to recognize and correct voice web search queries. Spoken web search queries are difficult to recognize since it is likely to be used in noisy environments. Therefore in [31], an mobile phone interface is developed. And the interface is based on a word confusion network built from the recognition lattice. When recognition is complete, the best hypothesis is displayed on screen. If a word is a recognition error, the user can change it to alternative words. In addition to selecting alternative words, the user can also type corrections on an on-screen keyboard. [31] found that on average, a user is able to speak and correct a web search query in about 18 seconds despite a high overall WER of 48%. And they showed that using a search-specific vocabulary with automatically generated pronunciations achieved better performance than using a vocabulary limited to a fixed pronunciation dictionary.

Since 2008, Google has been continuously working on their complete voice search service (GOOG-411) in the US and Canada for several types of smartphones [32]. The acoustic models are triphone decision-tree tied 3-state HMMs with currently up to 10k states total. The service uses between 200k and 1.5M words in the dictionary which are extracted automatically from the web-based query stream. Google has collected roughly 250k of spoken queries using an Android application to train an initial system. To approximates user experience, the ASR hypothesis and reference will be sent to a search backend and compare the links it gets back. In [32], it assumes that the reference generates the correct search results, thus in this way, Google knows whether the search result for the hypothesis is within the first three reference results such that the user can see the correct results on the smartphone screen. However, there are many challenges. These include optimizing protocols for minimum latency, dealing with special cases like numbers/dates/abbreviations correctly and avoid showing offensive queries.

2.6.2 ASR for Spell Mode Utterance

Another approach is to incorporate additional constraints in ASR decoding from other possibly non-acoustic inputs in order to improve spell mode ASR in mobile applications. An approach was proposed that constrains speech recognition using DTMF based key-

pad input when ASR for spell mode confidence is low. Parthasarathy [35] suggested that recognition can be constrained by keypad input. Figure 2.2 represents an unweighted grammar representing all possible letter sequences which correspond to the keypad input 78726 (representing SUSAN).

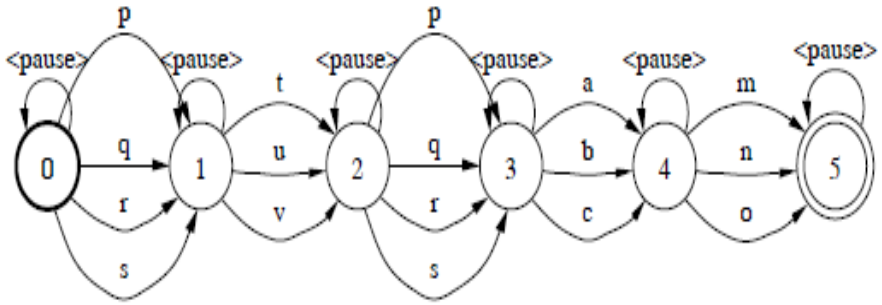

Fig. 2.2 Unweighted grammar generated from the keypad input 78726 (SUSAN)

If one had access to spellings that characterize the domain, such as a directory of business names for the name recognition task, one could use an N-gram letter model L_N trained on this data, and construct a weighted grammar

$$K_w = K \bigcap L_N \tag{2.26}$$

where K is the unweighted keypad grammar and $\bigcap$ is the intersection operation. The K_w is the output weighted grammar. Such a grammar could be a significant advantage in spell mode recognition. The output word lattice obtained by utilizing such grammar in spell mode recognition, can be later followed by a lookup in a database (of valid words, names, etc.) in order to find out the best valid letter sequence.

$$r_D = bestpath(R_{nc} \cdot D) \tag{2.27}$$

where R_{nc} is the word lattice obtained from ASR recognition without a database constraint, and D is finite state network that accepts only valid letter strings. By implementing database lookup as a separate step from ASR recognition has the following advantages.

- The complexity of the recognizer does not grow with the size of the database/directory.

- The vocabulary (allowed letter strings) as well as domain dependent language models (such as frequency of requested names) could be updated independent of the recognizer, thereby simplifying service deployment.

An almost perfect recognition 99% is reported in Parthasarathys work [35], with a scheme of speech input first, followed by keypad entry. G. Chung and S. Seneff have created a finite-state transducer (FST) that maps phonetics to graphemics, which is later composed with an FST derived from the keypad input to reduce the search space [36].

Another approach on ASR for spell mode utterances is proposed by Chung and Seneff [37]. The problem of multi-stage speak and spell recognition has been properly addressed hoping a second recognition pass for spellings will help system handle unknown words. Chung and Seneff [37] dealt with new word acquisition for applications where the input utterances corresponded to an unknown word followed by the spelling of the word. Their procedure involved using an automatic procedure for mapping the letter sequence to a phoneme sequence and using this phoneme sequence as the phonetic expansion for the new word. Specifically the first-stage extracts the spelled letters from the spoken utterance, treating the pronounced portion of the word as a generic out of vocabulary (OOV) word. Then it is followed by an intermediate stage where the hypotheses of the letter recognition are used to construct a pruned search space for a final sound-to-letter recognizer which will directly output grapheme sequences.

2.6.3 Limitation of Spell Mode Recognition

Although keypad constraint methods discussed previously in this section have achieved a significant improvement in spell mode recognition, but keypad input may not be very natural in a spoken language system and the design of a user interface to incorporate keypad and speech may be a challenge. Also when utterances can contain a combination of spoken words and spoken letters, it can be difficult to model the transition between the word sequence and the letter sequence.

Additionally, in the recognition of spell mode utterances, dedicated statistical language models can be trained from training text that has been expanded into letter sequences. However, it is difficult to train statistical language models for utterances that may contain

a combination of word and letter sequences. Because of potentially high ASR error rates for these utterances and the potential of misspellings being made by the user, the ASR word accuracy can be low for spell mode utterances. Therefore, in this thesis a second pass re-ordering process is introduced to deal spell mode utterances.

Chapter 3

Multiple Pass Spoken Term Retrieval

As discussed in previous chapters, this thesis focuses on improving recognition accuracy of OOV words by dealing with utterances in spell mode. After a first pass ASR decoding which produces a set of hypothesized word strings for an utterance, a second pass re-ordering process is performed by exploiting additional knowledge sources, such as information from the application specific database linguistic pronunciation rules and others. This chapter investigates multiple pass techniques for both reducing the effective WER for spell mode utterances and for improving the information retrieval performance in this voice search application.

This chapter provides a detailed description of the second pass re-ordering process which is summarized in the block diagram given in Figure 3.1. In Section 3.1, the overview of the second pass re-ordering system as well as the data flow will be introduced. The remainder of this chapter presents the implementation of how re-ordering process exploits external knowledge sources. Specifically, Section 3.2 and 3.3 will introduce the structure of networks used to perform letter based and phoneme based re-scoring of ASR hypotheses. Section 3.4 explains how a letter based statistical language model (LBLM) can be applied to estimating the probabilities of the entries in the external database and then used as a score when re-ranking ASR hypotheses. Dynamic Programming based search for grapheme/phoneme alignment will be discussed in Section 3.5. A modified beam search algorithm which achieves a good trade-off between search accuracy and efficiency will be discussed in Section 3.6. Section 3.7 presents the implementation of all search algorithms

used in this thesis work, and it also describes the parameter selection for the search. Finally Section 3.8 presents a discriminative model combination algorithm which estimates the weights of a log linear model that combines the scores associated with the above knowledge sources. The output of this log model is used for re-ordering hypothesized word strings obtained from the ASR system.

3.1 Overview of Re-scoring ASR Hypotheses

This section provides an overview of the second pass re-ordering process shown in Figure 3.1. There are several procedures that are presented in this section for implementing this re-ordering process. First, an ASR system produces a list of the N most likely hypothesized

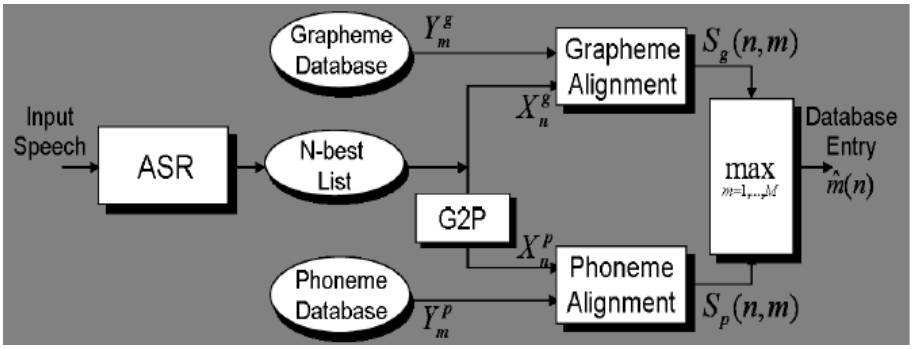

Fig. 3.1 Re-ordering N-best ASR candidates obtained from spell mode utterances

word sequences. A spoken letter sequence, if any, is extracted from each candidate word string. It is assumed that the hypothesized letter sequences produced by the ASR system may contain either recognition errors produced by the ASR system or the query term may have been misspelled by the speaker.

Second, the N-best letter sequence is re-ordered with respect to both grapheme and phoneme expansions of the entries in the text database as well as to the specific language model associated with the particular application. An recognition error could happen in the ASR system described in Section 2.1 due to inaccurate modeling, it is assumed that the recognized letter sequence may contain simple letter substitutions, insertions, and deletions. On the other side, the utterances themselves may contain misspellings made by users, in such cases, it is assumed that the hypothesized letter sequence corresponds to

a word or words with similar pronunciation to the correct spelling. To solve these two classes of errors, the solution is given by the second pass re-ordering process that a dynamic programming based string alignment discussed in Section 2.2 will be performed. Specifically for the recognition error, in Figure 3.1 a DP based grapheme alignment of the hypothesized letter sequence X_n^g with respect to graphemic expansions of the entries in the database Y_m^g is performed. This process will obtain $S_g(n,m)$ which is a set of grapheme based distances between the nth ASR candidate and the mth database entry. For the misspelling errors, a phonetic expansion of the hypothesized grapheme sequence X_n^p will be obtained using grapheme-to-phoneme estimation described in Section 2.4. and a DP based phoneme alignment of the resulting phoneme sequence with respect to phonetic expansions of the database entries will be performed. This process will yield a set of phoneme based distances, $S_p(n,m)$, shown in Figure 3.1.

Other than the distances discussed in previous paragraph, additional letter sequence constraints are applied in the form of the log probabilities associated with a letter based language model (LBLM). This is done by training trigram probabilities for letter sequences by first obtaining word transcriptions taken from the domain of a given search category and then expanding the words as grapheme sequences, which will be presented in Section 3.4. Though not shown in Figure 3.1, the LM probabilities are included in external database and in the criterion based on a log linear combination. The log-linear combination will obtain the optimum database entry, $\hat{m}(n)$. This criterion is described in more detail in Section 3.8.

3.2 Network Representation of a Grapheme Database

Given a letter sequence contained in an ASR string hypothesis, a dynamic programming procedure is used to compute a score for that string with respect to grapheme expansions of database entries. This procedure is made more efficient by organizing these grapheme expansions in a finite state network and the actual string alignment process is implemented in C language which usually saves memory consumption. This section describes the structure and creation of this network. The grapheme database represents the graphemic expansion of all valid word entries from a specific vocabulary. In other words, such a grapheme database is capable of converting a string of letters to the corresponding word entry in the

associated database. The structure of the grapheme database model includes nodes with input letter, output word entry and transition arcs connecting each node together. The input letters are indeed in the order of composing a word in the specific vocabulary. The output word entry is actually the word itself from the vocabulary. These basic structures enables the network to convert letter string to the corresponding word entry. The structure of transition arcs describes the arrangements of possible letters and the language model (LM) cost that used to differentiate between letters with similar ones. In practice, the LM cost corresponds to the negative logarithm of the statistical language model probability, and the language model is trained on the transcription of historical search queries along with the specific external vocabulary. The method used to compute the cost will be discussed in detail in Section 3.4. The LM cost will be used as one of the criterion to look for the optimum database entry.

3.2.1 Basic Method for Grapheme Network Creation

The grapheme database transduces an input letter sequence to the corresponding word entry in the vocabulary. The basic method used to create a grapheme database begins with a single node which is both a departure and arrival node. For each word in the dictionary, a set of arcs is added to form a loop from the initial node. The inputs of the arcs which form the loop are in order of letter sequence followed by the special symbol "#wb#" which refers to "end-of-word" or "word-boundary". The arcs with input letters have output label "#eps#" referring to "epsilon" with a LM cost of zero. But the last arc with the "#wb#" input has output label corresponding to the word symbol and a letter based LM cost associated with that word entry. Figure 3.2 shows a grapheme database that includes two words "nuance" and "nance". Indeed, the costs on the arcs are the negative logarithm of the statistical language model probability. In Figure 3.2, the negative logarithm of LM probability for the word "NUANCE" is 6.38 as shown, and for the word "NANCE", the negative logarithm of LM probability is 7.23. The process of training a LBLM and computing LM costs has the following steps:

- Label user utterances with the spoken work string and identify utterances corresponding to spoken letter sequences.
- Train a SLM: Estimate counts and compute probabilities for a trigram LBLM as described in Section 3.4.

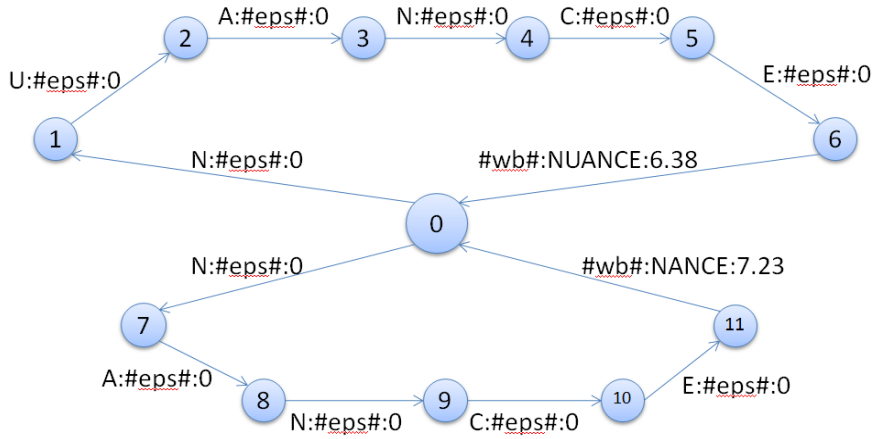

Fig. 3.2 Closed loops for words "nuance" and "nance" in unsimplified model

- Obtain LM Cost of a word entry: The LM cost is the negative logarithm of the LBLM probability of the word entry.

Once the network has been created, it is necessary to reduce the number of paths in the network by doing minimization then determinization operations to the network. The determinization operation in finite state automata reduces the number of arcs and nodes by combining the arcs that originate from the same node while having the same input letter [38][39][40]. Figure 3.3 shows the network from Figure 3.2 after having been determinized and with redundant states removed. Unfortunately, this basic creation method will create

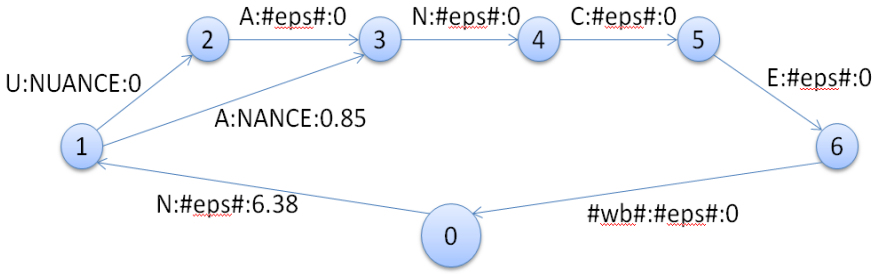

Fig. 3.3 Closed loops for words "nuance" and "nance" in simplified model

a loop for each single word. When over millions of word entries will be included, this basic method for creating grapheme database is way far from being efficient in processing time and memory usage.

3.2.2 Algorithm Used for Grapheme Database Creation

In order to simplify the creation procedure and especially the memory usage, the grapheme database creation and simplification can be carried out simultaneously. The algorithm used to achieve this is to treat the word entries from the vocabulary in alphabetical order and maximize the common letter-span sharing. It is concluded into the following algorithm.

Algorithm 1 Algorithm used for Grapheme Database Creation

Read the word entries from dictionary in alphabetical order and determine the associated LM costs.
Create initial node 0, as the arrival and departure node.
Create a "next node" and initialize node number to 1.
Create an empty list of states.
for all word entries read from the dictionary **do**
  for each letter of the word **do**
    Assign i to be the letter position of the word (i = 1 if it is the first letter).
    if the list contains at least "i" states and the "ith" element contains the current letter **then**
      Continue with the next letter.
    else
      Keep only the first "i-1" states.
      Find the word entries which shares the first "i" letters.
      For these word entries, find the minimal cost.
      Assign x to be the difference between the minimal cost and the cost of the last state in the list.
      if only one word shares these letters and it's not marked **then**
        From the node corresponding to the last state in the list, add an arc with the current letter as input, the current word as output and variable x as cost, to the next node.
        Mark the current word.
      else
        From the node corresponding to the last state in the list, add an arc with the current letter as input, the empty symbol "#eps#" as output and variable x as cost, to the next node.
      end if
      Add a state containing the next node, the minimal cost and the current letter to the list.
      Increment the next node number by 1.
    end if
  end for
  Conduct Algorithm 2
end for

Algorithm 2 (Continued) Algorithm used for Grapheme Database Creation

Assign x to be the difference between the cost associate with the current word entry and the the cost of last state in the list.

if the current word entry is not marked **then**

From the node corresponding to the last state in the list, add an arc with word boundary "#wb#" as input, the current word as output and variable x as cost, to the arrival node (node 0).

else

From the node corresponding to the last state in the list, add an arc with word boundary "#wb#" as input, the empty "#eps#" as output and the variable x as cost, to the arrival node (node 0).

end if

The process involves consolidating all paths with common letter sequence prefixes into shared paths. This is similar to the operation of determinization in finite state automata [38][39][40]. When creating an arc, a state is added to the list of states. The list of states contains the last treated letter, the arrival node number of this arc and the total cost associated with the path terminating at this node. This cost is the minimum cost of all word entries that have the same characters till that point. In fact, finding which words shares letters is relatively easy to achieve since the dictionary word entries are placed alphabetically. For example, when the arc with the input letter "a" is added to departure node, assuming the lowest cost among complete paths from here, is 4.2505 associated with the path "and". In such case, the cost associated with this arc "a" is 4.2505 which equivalent to the LM cost of word "and".

Another operation is moving LM costs forwards as much as possible to the departure node. To do this, the LM cost associated to an arc corresponds to the "net cost", that is the difference between the total LM cost of an complete path and the LM cost till this node. The "cost till this node" is calculated by summing all cost from departure node to the current node. To continue the example in previous paragraph, assuming an arc with the input letter "r" is added after the arc "a", and the cost associated with the word "are" is 6.6031. Since a cost in arc "a" is already 4.2505, therefore the "net cost" of this current arc "r" is the difference between 6.6031 and 4.2505, which is 2.3526.

For a dictionary of over 1 million entries, the creation time using the above procedure is about 50 seconds. The resulting model contains 4.6 million arcs and 3.5 million nodes whereas the original network has over 6 million arcs and over 5 million nodes. The effect of the above procedure includes carrying the creation as well as the simplification simultaneously. Since a binary search is used to find word entries that share the common letters in the alphabetically sorted word list, the complexity of the algorithm is about $n * log(n)$ based on the number of word entries. In addition, the memory usage is only proportional to the number of word entries, since the arcs are not kept in memory while creating the model. Indeed, only the number of nodes that are retained in the list of states, is used to generate the next arcs. By using this algorithm, different grapheme database models have been created for corresponding application, i.e. "business name", "stock quote", "website names" and others.

3.3 Network Representation of a Phoneme Database

Section 3.2 describes the creation of network for the grapheme expansion of the database entries. This section describes the process for creating the network for the phonemic expansion of database entries. The phoneme database can convert a series of phonemes into a word entry by aligning input phoneme string with the phonetic expansion of database word entries. The phoneme database creation requires a pronunciation dictionary that maps the phoneme sequences to corresponding word entries. The Nuance pronunciation dictionaries have been utilized in this thesis work. However, a large number of words in the training set are absent, i.e. OOV words. The pronunciation of these OOV words must be estimated. Grapheme to phoneme (G2P) estimation is therefore utilized. Throughout the development of phoneme database, more than 20,000 pronunciations have been estimated using G2P estimation. In this section, a combination of N-Gram and Decision Tree based G2P estimation will be introduced. Then the creation of phoneme database will be presented.

3.3.1 Grapheme to Phoneme Estimation Used in the Thesis Work

The G2P techniques discussed in Section 2.4 address the problem of finding and improving algorithms for automatically transcribing words written in a roman alphabet (or similar) where a special focus is taken on proper names. Although speed and memory consumption

of the algorithms are also important issues, the accuracy of the automatic transcription is the main objective in this study. The study does not treat the pronunciation of words containing digits, abbreviations or those words partially spelled. According to Section 2.4, both N-gram and decision tree approaches have their own strengths and weaknesses. Therefore it may be a good idea to combine the two approaches. The following way is proposed to realize this combination. It is noted that the approach of N-gram search algorithm is utilized again in this combined algorithm, however the original N-gram optimization criterion is replaced by a combined decision-tree and N-gram criterion.

Assuming a decision tree and an N-gram model using the algorithms discussed in Section 2.4 has already been trained . And the decision tree model must be trained in a way such that the probability distributions:

$$P_{dt}(p_1 \cdots p_N | g_1 \cdots g_N) = \prod_{i=1}^{N} P_{dt}(p_i | g_1 \cdots g_N) \tag{3.1}$$

are still available. In addition we have the joint probability from the N-gram model:

$$P_{ng}(p_1 \cdots p_N | g_1 \cdots g_N) = \prod_{i=1}^{N} P_{ng}(p_i, g_i | p_{i-n+1}, g_{i-n+1} \cdots p_{i-1} g_{i-1}) \tag{3.2}$$

From here, a conditional probability can be calculated as:

$$P_{ng}(p_1 \cdots p_N | g_1 \cdots g_N) = P_{ng}(p_1 \cdots p_N, g_1 \cdots g_N) / P_{ng}(g_1 \cdots g_N) \tag{3.3}$$

A new combined model could be build from this by linear interpolation:

$$P_{ngdt}(p_1 \cdots p_N | g_1 \cdots g_N) = \alpha P_{ng}(p_1 \cdots p_N | g_1 \cdots g_N) + (1 - \alpha) P_{dt}(p_1 \cdots p_N | g_1 \cdots g_N) \tag{3.4}$$

where the interpolation factor α is an additional parameter which can be chosen to optimize

the performance. An alternative way is the log-linear interpolation:

$$\begin{aligned} logP_{ngdt}(p_1 \cdots p_N | g_1 \cdots g_N) &= \alpha logP_{ng}(p_1 \cdots p_N | g_1 \cdots g_N) + (1-\alpha)P_{dt}(p_1 \cdots p_N | g_1 \cdots g_N) \\ &= \alpha logP_{ng}(p_1 \cdots p_N, g_1 \cdots g_N) - \alpha logP_{ng}(g_1 \cdots g_N) \\ &\quad +(1-\alpha)P_{dt}(p_1 \cdots p_N | g_1 \cdots g_N) \end{aligned} \tag{3.5}$$

This heuristic interpolation approach is not as satisfactory as from a theoretical point of view, but it has proven to provide more accurate prediction of phonemic pronunciations in similar language modeling experiments conducted in Nuance Communication Inc. It has another benefit that is we do not need to calculate the aprior N-gram probability of the grapheme sequence, since this is a constant term that can be neglected when only looking at the $argmax$. Finally it fits smoothly in the general search architecture of the N-gram algorithm such that we may apply exactly the same search algorithm as used in the N-gram approach.

This combination of N-Gram and Decision Tree based algorithm has been utilized in estimating OOV words pronunciation when creating the network of phonemic expansions for database entries. Specifically in Figure 3.1, it has been utilized in the second pass re-ordering process, as to estimate the phonemic expansion of letter sequences obtained from ASR hypotheses.

3.3.2 Creation of Phoneme Database

The structure of a phoneme database is similar to the one of grapheme database. The major difference between these two, is that a phoneme database can convert a series of phoneme input to the corresponding word entry, rather a grapheme database takes letter sequence and convert it to the corresponding word entry. The following Algorithm 3 shows the procedure to create the phoneme database.

Algorithm 3 Algorithm used for Phoneme Database Creation

Read the word entries from the dictionary as well as from OOV word collections, sort them by alphabetical order and determine the associated LM costs.
Read the pronunciation dictionary.
Create an empty list for the pronunciation to be treated.
for each of the word entry in dictionary **do**
  Search its pronunciation in the pronunciation dictionary.
  Add each pronunciation and the LM cost associated with the word entry to the pronunciation list.
end for
Create the initial node 0, which is both the departure node and arrival node.
Create a variable "next node" and initialize the node number to 1.
Create an empty list of states.
for each of the pronunciation of the list to be treated **do**
  Perform the steps of algorithm 1 using phoneme input instead of letter input.
end for

In other words, a grapheme database takes letters as input, and the corresponding outputs are word entry in the specific dictionary. However in a phoneme database, the inputs are actually the phoneme from the phonemic expansion of the dictionary, while the outputs are still the corresponding word entry. By doing so, a phoneme database validates the pronunciation of a word. Again although not shown explicitly in Figure 3.1, the LM cost attributes to give weight on the phoneme inputs.

3.4 Training Letter-Based Statistical Language Model

In this section, the procedure of training a letter based SLM will be presented and some key procedures will be given as examples and explained in detail. As previously discussed in this chapter, a word entry in an external database has a LM cost associated with it. This LM cost is actually the negative logarithm of the probability of a letter based SLM trained for this specific application. Specifically Figure 3.4 shows the process for generating the SLM for "website names" application from a training file.

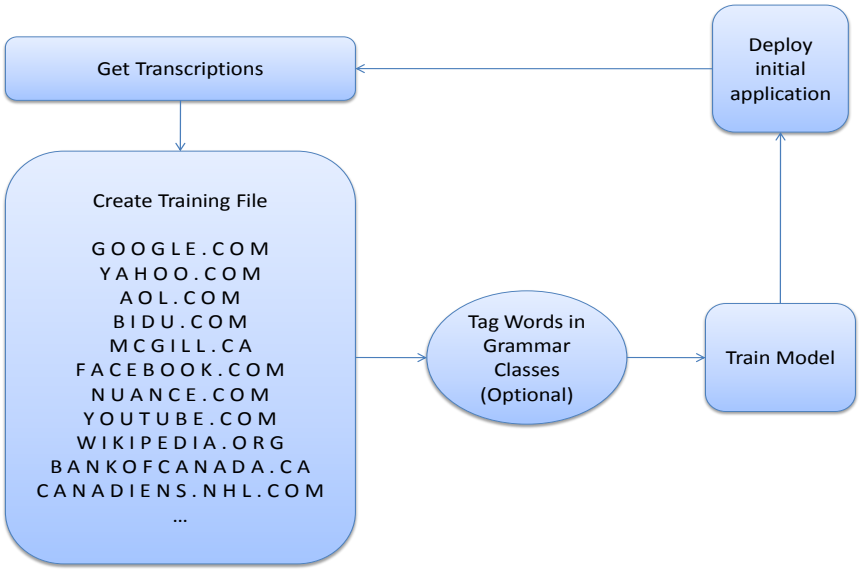

Fig. 3.4 Generating an SLM from a training file

According to Figure 3.4, training a SLM consists of several steps:

- Create a training corpus by expanding words into sequences of letters and special symbols.
- Determine the order n of the model (discussed later in Section 3.4.3).
- Compute counts and estimate probabilities of the LBLM.
- Use the SLM grammar in application, and test it during early deployment with small number of test sets. During the test, evaluate the speed and recognition accuracy, generate new transcriptions, augment the training file and make changes if needed to configuration parameters.

3.4.1 Tuning SLM

Once deployed, an application can always be improved and fine-tuned. For SLMs in particular, real-life data can be used to augment the training set. We can do several iterations to fine-tune an SLM. The steps in tuning process include:

- Begin with the model initially trained.

- Exercise this model as early as possible during the application development life cycle. These early phases of the life cycle include usability testing, pre-deployment (low-load usage) and full deployment (increasing load).
- At runtime, the re-ordering process produces new letter sequences that to be included in the next training file.
- Refine configuration parameter settings.
- With the additional sentences added, recompile the training file. The result will be a new compiled model that can be used for the next iteration of the process. We can repeat the entire process several times, as recognition accuracy improves with each iteration.
- Optionally, create an additional set of test sentences as an evaluation set. The purpose of the evaluation set is to provide an independent validation of trained and tuned models. These test sentences can not be used for tuning iterations. Instead, they are kept separately until the tuning is nearly complete.

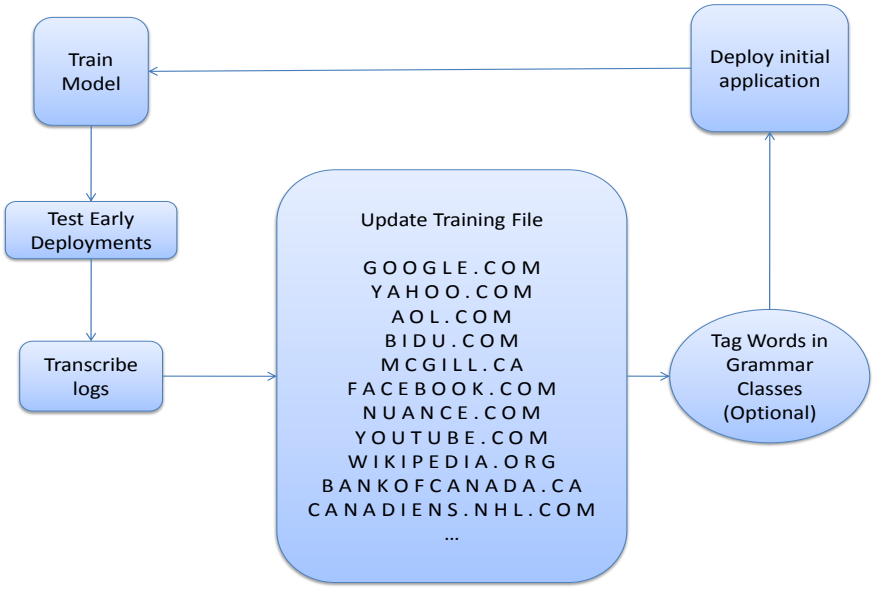

Fig. 3.5 Tuning an SLM

3.4.2 Language Model Performance - Perplexity

The measure used for measuring the performance of an SLM is test set *perplexity*. Perplexity is closely related to the average sentence score achieved on a set of test sentences. Computing perplexity of any statistical language model requires the existence of a held-out text corpus. For example, one common way of looking at perplexity is in terms of word-level perplexity. The perplexity of a text with respect to the word "water" represents the number of words which could potentially follow it. There are some given number of words which may follow the word "water" (gun, ball, and cube), and similarly, some words which would most likely not follow "water" (nuclear, peripheral, and happily). Since the number of possible words which could follow "water" is relatively high, its perplexity is high. In the example of a LBLM for stock symbol, a letter can be followed by another letter, as long as the letter combination is a valid stock symbol. For example, at the letter-level, the string "NUAX" where X is a letter, has a low perplexity since there are only 26 letters possible, and only the letter "N" could be found in this position in any stock symbol corpus.
Mathematically, the perplexity of a text (or word, corpus, or language) is defined as 2 raised to the power of the entropy [6]:

$$PP_{corpus} = 2^{H_{corpus}} \tag{3.6}$$

Another way of looking at this is in comparing different language models with respect to their usefulness for a particular text or corpora. The perplexity of a language model with respect to text x is the reciprocal of the geometric average of the probabilities of the predictions in text x. So, if text x has k words, then the perplexity of the language model with respect to that text is $Pr(x)^{-1/k}$ [41]. The distribution of the sentences in the held-out text corpus should reflect the actual distribution of sentences in the application. High perplexity implies a language model with a high level of uncertainty and is associated with poor performance in modeling text sequences. When estimating language model parameters, the goal is always to reduce the test set perplexity. In the thesis, language model order has been take into account to achieve low perplexity.

3.4.3 Order Determination

Assuming N is the order of the language model. The larger the N is, the larger context is used to assign a probability to a word, therefore the more powerful the $N-Gram$ model is. However, the number of probabilities needed in the model grows as a power of N, and is therefore more difficult to estimate at large N values. Other than that, by increasing N value, when the training samples are limited, the model may experience over-training; that is, the model may memorize the training set and hence lose its ability to model sentences that the training set does not cover. The optimum value of N is usually empirically determined by training a number of different $N-Gram$ SLMs and measure their performance.

In practice, after a few times of tuning SLM, a N-Gram language model is finally obtained which gives relatively low perplexity and relatively good performance on a specific application domain. And this LM will be used to generate LM cost for each word entry in the dictionary. In this thesis work, depending on different application task, both bigram and trigram language models have been trained for use. Till this point, the external database is completed.

3.5 Dynamic Programming Based Sequence Alignment

As discussed in previous sections, external database models define the validity of a letter string or phoneme string, and also contain the LM cost associated with the inputs. This section describes the search algorithm for finding the database entry that is the closest to a given N-best spell mode ASR hypothesis. In is done by sequence alignment where dynamic programming is used. The problem consists of transforming one sequence into another using an edit operations that replace, insert or remove an element. Each operation has an associated cost and the goal is find the sequence of edits with the lowest cost. In this thesis work, the search used on grapheme database is the same as it is used on phoneme database. In this section, the edit distance matrix will firstly be introduced, and then the usage of dynamic programming search will be presented.

3.5.1 Model of Edit Distance Matrix

To execute a sequence alignment between N-best ASR hypothesis and external knowledge sources, the second pass re-ordering process allows the insertion, deletion and conversion of grapheme or phoneme element. It is done by introducing an edit distance matrix which defines the cost when an insertion, deletion or substitution taking place. This section presents the calculation of those costs in such a matrix. Specifically, this section divides the edit distance cost into the cost of grapheme confusion and phoneme confusion.

Edit Distance Cost of Grapheme Confusion: For the edit cost used in grapheme alignment, it has been calculated from our training set as well as using estimated values from articles on the correction of spelling errors [42] [43]. The cost is essentially the negative logarithm of confusion probability, $P(h, r)$. The confusion probability is estimated from counts computed over training data:

$$P(h, r) = \frac{C(h, r)}{\sum_{k \in h} C(k, r)} \tag{3.7}$$

where $C(h, r)$ refers to the number of occurrences of the hypothesized letter h being aligned with the reference letter r, in held-out training data. In this thesis work, all negative logarithm of confusion probability $P(h, r)$ are normalized with respect to a maximum value of 13. Because it has been observed that with a maximum value of 13, the distribution of negative logarithm of $P(h, r)$ is spread out enough such that the confusion matrix is working properly in distinguishing errors. Since not all confusion errors have occurred in training set nor can be seen in reference articles, the counts for non-occurred errors have been assumed to be a very small number, and as a result the edit costs for these non-occurred errors have been computed very large. In this thesis, these very large values have been approximated to the maximum value of 13.

Edit Distance Cost of Phoneme Confusion: For confusion between the phonemes, not much information were found in literature. And the phonology is differently used from one software to another, thus it would be very difficult to learn edit costs of phoneme confusion from literature or from manuals of commercial speech recognition software. For simplicity, in this thesis only the training data has been used to compute the edit cost

of phoneme confusion. Specifically, the phoneme sequence obtained from ASR engine is compared with the phonemic expansion of the correct database entry. Counts and probabilities are then computed accordingly. For those non-occurred errors, the counts have been assumed to be very small and in turn, the corresponding costs have been computed to be very large and approximated to 10. Here the approximation value is 10 instead of 13 as used for grapheme confusion, is because having a phoneme error is more likely than having a spelling error.

3.5.2 Dynamic Programming Search

As introduced in Section 2.2, dynamic programming can simplify a complicated problem by breaking it down into simpler subproblems in a recursive manner. In this subsection, a dynamic programming search on grapheme database is taken as an example to explain how DP is being used in this thesis work. Later in this subsection, implementation of such a DP based search algorithm will be presented in detail including conversion of input, introduction of search token, calculation of edit distance costs and potential limitation in applications whose database is very huge.

Assuming there are N hypothesized letter sequence generated by ASR engine. According to Figure 3.1, the nth hypothesized sequence is given as $X_n^g = \{x_{1,n}^g, x_{2,n}^g, ..., x_{I_n,n}^g\}$ where $x_{i,n}^g$ is the ith letter or grapheme in the sequence and I_n is the length of the nth. For each search category there are M sequences in the category specific database. The grapheme sequence associated with the mth database entry is $Y_m^g = \{y_{1,m}^g, y_{2,m}^g, ..., y_{J_m,m}^g\}$ where $y_{j,m}^g$ is the jth grapheme and J_m is the length of the mth reference sequence. The distance between the grapheme sequence hypothesis, X_n^g and Y_m^g is given by $S_g(n,m)$ and corresponds to a Levenshtein distance computed using a dynamic programming (DP) algorithm. The distance, $L_{n,m}(i,j)$, between i length subsequence of X_n^g and j length subsequence of Y_m^g is computed by induction from subsequences of length $i-1$ and $j-1$ as:

$$L(i,j) = max \begin{cases} L(i-1,j) \cdot P(\phi, x_i) \\ L(i-1,j-1) \cdot P(y_j, x_i) \\ L(i,j-1) \cdot P(y_j, \phi) \end{cases} \tag{3.8}$$

In above equation, $P(y_j, x_i)$ is the probability of the hypothesized letter index y_j being confused with the reference letter index x_i, $P(y_j, \phi)$ refers to insertion probability, and $P(\phi, x_i)$ is the probability of deletion. These probabilities are estimated from counts computed over training data as discussed in previous Subsection 3.5.1. In the example in Figure 3.6, the

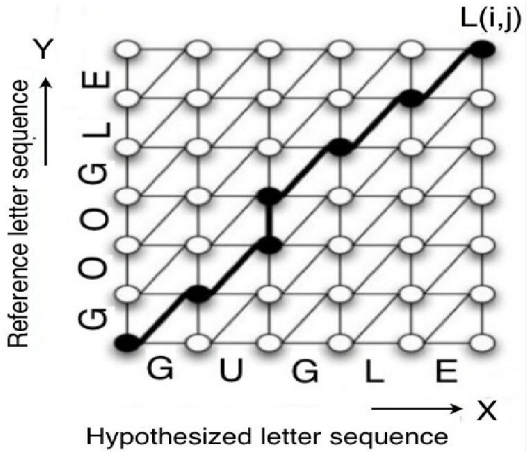

Fig. 3.6 Dynamic Programming Explanation

optimum Levenshtein distance $L(i, j)$ path is given in bold. It can be understood as two edits were made. The two edits consist of one substitution on the second input letter between "O" and "U", and another missing on the third input letter "O".

In the thesis work, before making the DP based search, either letter sequence or its phonetic expansion, phoneme sequence will be converted to a sequence of input tokens in order to be processed by the search algorithm. The conversion of input includes:

Conversion for Letter Sequence Input: The conversion for letter inputs is simple. We simply need to look for the letter number corresponding to each input letter and place them in the same order (i.e. "a" corresponds to the number 1 and "z" corresponds to the number 26, etc). Spaces are identified by a special empty token "#eps#". The conversion ensures that the a special token "#wb#" is added at the end of the sequence as the "word boundary".

Conversion for phoneme Input: For conversion of phoneme inputs, the process is very similar to conversion of letters except that it is possible that a word has more than

one pronunciation. An option to choose is that the re-ordering process performs a search for each of the pronunciations and keep the optimum result. An other option to choose is that one of the pronunciations will be chosen randomly.

After input phoneme string is converted into input token sequence, the search is underway while a search token used in search is introduced. A search token contains the following information about a current trip in the network:

- The current node number
- The cost of edit
- The LM cost of the current output
- The list of output encountered so far in the network

At first, a token is placed at the departure node of the network. The token then moves under certain conditions, from one node to another through transition arcs. It is noted that the path that the token travels will not be kept in memory, rather only the information that the token tracks will be saved.

To enable edit during the search in external database network, three types of transitions are considered. These correspond to the three basic operations of calculating the Levenshtein distance [44] [45]. First, the insertion allows the token to jump an arc without consuming the input. The cost associated with this insertion edit is based on the element to be inserted and its context. Second, in contrast with the insertion, deletion allows the token to stay in the same node while consuming the input. In this case, the cost is calculated based on the item to delete and the context of this operation. Third, the substitution allows the token to travel an arc leaving the node while consuming the input. When the input element is identical to that associated with the arc, the cost of conversion is null. If they are different, the cost of substitution is read from the corresponding position of confusion matrix. In this case, the context is not taken into account, since the model would be too complex.

Now that the database networks are compiled and the confusion matrix already defines the edit costs, the DP based search algorithm is capable to provide a sequence alignment.

By ranking the total number of possible transition paths by overall cost, we can always find the global optimum path. However, some databases are so huge that they contain over 4 millions internal nodes and 5 millions transition arcs. Since not only all nodes and all arcs must be traversed before the search is completed, but also the insertion, deletion and substitution greatly increases the computation time and memory usage by allowing arcs to be traversed several times. In such an application which will ultimately be using in real-time recognition, the current algorithm needs to be more efficiently modified.

3.6 Beam Search Pruning

This section presents a modified beam search pruning which successfully achieved good search efficiency while losing recognition accuracy only by a small amount [46] [47] [48]. Beam search is a heuristic search algorithm that is an optimization of best-first search that reduces its memory requirement. Best-first search is a graph search which orders all partial solutions according to some heuristic which attempts to predict how close a partial solution is to a complete solution. In beam search, only a predetermined number of best partial solutions (also called beam width) are kept as candidates. In other words, it uses breadth-first search to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in order of increasing heuristic values. The smaller the beam width is, the more states are pruned. Therefore, with an infinite beam width, no states are pruned and beam search is identical to breadth-first search. In theory, the beam width bounds the memory required to perform the search, at the expense of risking completeness and optimality. The main reason for this risk is that the goal state could potentially be pruned [49] [50].

According to Figure 3.1 and the discussion made in previous section regarding dynamic programming based search, the distance between input string candidate X_n^g and database entry Y_m^g can be written in terms of the DP score as $S_g(n,m) = L_{n,m}(I_n, J_m)$. The computation associated with evaluating $S_g(n,m)$ for all $m = 1, ..., M$ database entries can be prohibitive to real time operation if the input string X_n^g is aligned to one reference string at a time as suggested by Equation 3.8. The computational load associated with computing $S_g(n,m)$ can be reduced by pruning partial path matches so that the induction in Equation 3.8 is not performed for subsequences terminating at letter pairs (i,j) when the partial path

score $L_{(n,m)}(i,j)$ falls below a threshold. A beam search strategy has been implemented where a pruning threshold, $T(i)$, is obtained by applying a fixed pruning beam width, BW, to the maximum alignment score obtained for the input subsequence of length i, where $T(i) = max_j(L(i,j)) - BW$.

In different steps of the search, the beam width can either be fixed or variable. In a fixed beam width, a maximum number of successor states is kept. In a variable beam width, a threshold is set around the current best state. All states that fall outside this threshold are discarded. Thus, in places where the best path is obvious, a minimal number of states is searched. In places where the best path is ambiguous, many paths will be searched. In this thesis work, a more efficient beam search strategy has been used for computing the distances $S_g(n,m)$ for input string X_n in a way that facilitates a more aggressive pruning strategy using an adaptive pruning threshold. The $i-1$ step of this procedure involves aligning all reference strings $Y_m^g, m = 1, \cdot, M$ to an $i-1$ length substring of X_n^g. Low scoring partial alignments are pruned and all surviving alignments are extended to a length i length substring of X_n^g for all M reference strings using the induction in Equation 3.8. This is continued until $i = I_n$. The important aspect of this procedure is that the pruning beam width can be adapted at each step based on information derived from the partial alignment obtained from all reference strings to the ith input substring.

At the ith step of the above procedure, a dynamic pruning beam width $BW(i)$ is obtained from the beam width at step $i-1$ as

$$BW(i) = BW(i-1) * s(i) \tag{3.9}$$

where $s(i)$ is a scale factor that has the effect of reducing the beam width as the length of the input substring increases. The scale factor that is applied to $BW(i)$ is itself dependent on a minimum threshold applied to the value of the beam width, BW_{min} and to a minimum threshold, NP_{min} applied to the number of active substring alignments:

$$s(i) = \begin{cases} as(i-1) & BW(i) > BW_{min}, NP > NP_{min} \\ 1.0 & otherwise \end{cases} \tag{3.10}$$

The value of $s(i)$ is initialized to 1.0 and a value of $\alpha = 0.9$ was found to provide a good trade-off between performance and efficiency. Equation 3.9 and 3.10 have the effect of continually decreasing the beam width for successively longer input subsequences.

3.7 Implementation of Search Algorithms

This section provides detail implementation of the search algorithms used in the thesis work. Since the approaches of dynamic programming based search and modified beam pruning have already been explained in previous sections, in this section the parameter selection criterion will be firstly introduced and then the algorithm implementation will be presented in pesudo-code level.

3.7.1 Parameter Selection

The DP based search algorithm with modified beam pruning divides costs into two distinct categories, that is edit cost and LM cost. To clarify the selection criteria, active tokens use three beams: the cost of edit distance, the cost of transition arc or equivalently the LM cost and the total cost. The beam width greatly influences the time taken to process search. A too wide beam width leads to unnecessary processing nodes while a too narrow width prevents error tolerance.

In the implementation of algorithms, several parameters were used to constrain the active tokens. For each of the three costs – cost of edit, LM cost and the total cost, there is a base width, a minimum width, a maximum cost and a reduction factor. In addition, the maximum number of active tokens is also imposed. It is also possible to specify the method when checking the validity of the tokens. It is possible to specify whether one, two or three beams must be respected in order to keep an active token.

To expedite the process, fixed beam widths are initialized for all search. For example, in grapheme alignment, the width is 6 for the cost of edit distance, 10 for the cost of traveling paths (LM cost) and 8 for the total cost. To keep an active token, two of the three beams must be respected. While in phoneme alignment, the width is 11 for all beams. With these experimental parameters, the search time is on average of 20 milliseconds per word.

These parameters are concluded to achieve the best overall performance from a number of experiments. More experimental results will be discussed in Chapter 4.

3.7.2 Algorithms Used to Browse The Models

In this subsection, algorithms will be presented to explain the implementation of the main part of second pass re-ordering process. Algorithm 4 shows the modified beam pruning. This treatment applies search to models of spelling and pronunciation. At the start of this algorithm, the letter or phoneme input have already been transformed into sequence of input tokens using the "conversion for inputs" procedure described in Subsection 3.5.2.

Algorithm 4 Algorithm used in modified beam search

Create a variable containing the last treated element.
Assign the symbol "wb" (end of a word) to it.
Create an empty list of active tokens.
Add a token associated with the start node in the model to the list.
Since only active token is the departure token, the minimum cost is zero for: the cost of edit distance, the cost of travel and total cost.
Browse the graph without consuming an input of the request (see Algorithm 5).
for each of the input element **do**
  Erase all data on the reference token.
  Browse the graph with consuming a single element entry (see Algorithm 6).
  if As the number of tokens exceeds the maximum number allowed and it is possible to reduce beam with **then**
    Reduce the width of beam search.
    Delete tokens that no longer respect the selection criteria of active tokens.
  end if
  Assign the current input element to the "last treated element".
  Browse the active tokens to find the minimum cost for: the cost of edit distance, the cost of traverse and total cost.
  Browse the graph without consuming an input of the request (see Algorithm 5).
  Treat the intersection between active tokens.
end for
Only the active tokens that are at the end node will be kept.
Delete duplicate tokens.
Sort and descent the token list by total cost and keep only the specified number of tokens.

The methodology behind this algorithm is to separate the insertion operations (without consumption of input element) from those of deletion and substitution (including consumption). The algorithm performs two types of treatment. To cover all possibilities, an insertion is made at the beginning and at the end of each search. Each move that consumes an input element is followed by filtering tokens. This filtering operation will reduce the width of the dynamic width, and reduce the maximum number of active tokens. Finally, only the tokens that successfully get to the arrival node are considered as valid results,

since they are the only tokens that have traveled across the model. These tokens are then sorted in descended order by their total cost.

The Algorithm 5 below shows the procedure used to move the tokens without consuming the input element.

Algorithm 5 Algorithm used to move the search tokens in the graph model without consuming an input

```
for each of the active tokens do
  for each of the arcs leaving the node where an active token is do
    Copy the current token to a new token which will be used temporarily for the rest
    of the treatment on this arc.
    Add the LM cost of this path.
    Add the cost of insertion if necessary (if applicable).
    Update the current node of temporary token.
    Add the output associated with the arc (if applicable).
    if the criterion of selecting active tokens are not respected then
      Delete the temporary token.
    else if there is a reference token at the same node and having the same output
    then
      if it has a lower total cost than the temporary token then
        Delete temporary token.
      else
        Replace the reference token by the temporary token.
        Add the temporary token to the end of the active token list.
      end if
    else
      Add the temporary token as the reference token.
      Add the temporary token to the end of active token list.
    end if
  end for
end for
```

The procedure is to move search tokens without consuming an input, in other words, it

allows insertion happen. It involves the insertion of the input associated with the traveled arc, since no input has been consumed. Enabling insertion is based on two criteria. First, the criteria of selecting active tokens must be respected. Second, it should not have another token called reference token which refers to a token has the same output or at the same node while having lower total cost than the token to be added.

The Algorithm 6 shows the procedure used to move the tokens by consuming the input element. The processing performed by the Algorithm 6 is similar to Algorithm 5 except it allows deletion and substitution happen. The possibility of a deletion error has been dealt with early in the algorithm as a special case. The proposed substitution is to add a token for each arc leaving the node where an active token is. In addition, the list to which tokens are added, is independent of active token list such that it ensures only one element is consumed.

After creating and implementing these algorithms, the dynamic programming based search with modified beam pruning enables edits in the sequence alignment in second pass re-ordering process. Not only the optimum path of a correct input sequence will be found, but also the most likely path corresponding to an erroneous input sequence will be found too. According to Figure 3.1, this second pass re-ordering process consults on external textual grapheme database, phoneme database and SLM trained for a specific database entry. The sequence alignment function takes care of substitution, insertion or deletion, and based on appropriate cost calculation, the top N-best paths will be found. By applying the search algorithm discussed in this chapter, the re-ordering process is capable to find the N-best database entry based on spelling, pronunciation and language model probability.

3.8 Discriminatively Re-scoring ASR Candidates

This section describes finding the optimum database entry by using discriminative model combination method discussed in Section 2.5. For each of the N-best spoken letter string candidates generated by the ASR system, the last block in re-ordering procedure described in Figure 3.1 finds an optimum database entry, $\hat{m}(n)$, that optimizes a criterion based on a linear combination of three measures. These include grapheme alignment score, $s_g(n) = S_g(n, \hat{m})$, phoneme based alignment score, $S_p(n) = S_p(n, \hat{m})$, and the letter sequence language model probability, $p(Y^g_{\hat{m}})$. The alignment scores can be normalized and

Algorithm 6 Algorithm used to move the search tokens in the graph models with consumption of an input

```
Create a new list of tokens.
for each of the active tokens do
  Copy the current token into a temporary token which will be used for the rest treatment.
  Add the deletion cost to the cost of edit distance correction (if applicable).
  if the parameter of selecting active tokens are not respected then
    Delete the temporary token.
  else if there is a reference token at the same node and having the same output then
    if it has a lower total cost than the temporary token then
      Delete the temporary token.
    else
      Replace the reference token by the temporary token.
      Add the temporary token to the end of the new token list.
    end if
  else
    Add the temporary token as the reference token.
    Add the temporary token to the end of the new token list.
  end if
  for each of the arcs leaving the node where the current token is do
    Copy the current token into a temporary token which is to be used for the rest treatment of this arc.
    Add the travel cost of this arc to the travel cost.
    Add the substitution cost to the edit distance cost (if applicable).
    Update the current node of temporary token.
    Add the element of output associated with the arc traveled (if applicable).
    if the parameters of selecting active tokens are not respected then
      Delete the temporary token.
    else if there is a reference token at the same node and having the same output then
      if it has a lower total cost than the temporary token then
        Delete temporary token.
      else
        Replace the reference token by the temporary token.
        Add the temporary token to the end of the new token list.
      end if
    else
      Add the temporary token as the reference token.
      Add the temporary token to the end of the new token list.
    end if
  end for
end for
Remove all items from the active token list.
Copy the new token list to the active token list.
```

approximated as the probabilities $p(X_n^g|Y_{\hat{m}}^g)$ and $p(X_n^p|Y_{\hat{m}}^p)$ respectively. The overall optimization criterion is given in terms of the weight vector $\Lambda = \lambda_1, \lambda_2, \lambda_3$ as

$$\lambda_1 logp(Y_{\hat{m}}^g) + \lambda_2 logp(X_n^g|Y_{\hat{m}}^g) + \lambda_3 logp(X_n^p|Y_{\hat{m}}^p) \tag{3.11}$$

These weights are estimated discriminatively by minimizing the continuous function of the string error generated by the ASR system on a held out training set using method in Section 2.5.

Chapter 4

Experimental Study

This chapter aims to demonstrate the performance improvement achieved by utilizing the second pass re-ordering process discussed in Chapter 3. Experiments have been conducted by using real-life mobile voice search data obtained from Nuance Communication Inc. In order to give a clear and convincing demonstration, this chapter has been divided into three parts. Section 4.1 will introduce the experimental setup including the task domain of Nuance's mobile search service. In Section 4.2, a baseline system performance will be introduced. This baseline performance will be later used to compare with the performance achieved by the additional second pass re-ordering process. Finally in Section 4.3, an detailed experimental analysis will be presented including the relative performance improvement of the entire system with respect to the baseline system.

4.1 Experiment Setup

This section briefly introduces the voice search task domain and motivates the approaches investigated in this work. The voice search task being addressed here involves spoken queries entered by users of mobile devices. The mobile voice search service we have worked on provides users of any mobile handset with speech driven access and control of on-device features as well as popular connected services like search (Web, music, etc.), messaging, and navigation. Users can simply use their own voice for a quick and easy access to applications and content. It is given that users have already known the basic syntax of making a voice search query by reading service manual. Users perform directed queries by speaking one of a set of 5 category names at the beginning of an utterance. Examples of search categories

include business names, stock quotes, and website names. Table 4.1 describes a pilot corpus containing 3935 labeled utterances that has been collected from a population of 601 users of the service. While the utterances begin with a search category name, they are otherwise relatively unconstrained. Analysis of the data shows that there are approximately 6 words per utterance. As shown in Table 4.2, approximately 37 percent of the utterances contain spoken letter sequences which correspond to spell mode versions of a query term.

Data set statistics

	Utterances per Speaker	Words per Speaker
Mean	6.5	40.7
Total	3935	24480

Table 4.1 Statistics of a transcribed data set

Utterances in Spell Mode

Category	Number of Utterances	Increase in WER w.r.t. baseline WER
Go to website	560	+7.3%
Stock quote	512	+8.6%
Business name	349	+7.6%

Table 4.2 Utterances in spell mode and their recognition performance

This situation is problematic because as shown in the last column of Table 4.2, the WER obtained for utterances containing spoken letters was found to be approximately 8 percent higher than the WER for utterances that did not contain spoken letters. However, spell mode utterances also create an opportunity in multiple pass recognition scenarios where potentially erroneous letter sequences obtained from a first pass ASR decoder can be used to detect query terms in a larger speech corpus. This is important because for some search categories the number of entries can be extremely large. For example, the website name category in the voice search search service investigated in this work contains approximately 1.3 millions entries [51]. As a result, the number of utterances presented to the ASR system that contain out-of-vocabulary (OOV) words can be expected to be quite high. The main reason that the performance in this thesis work has been shown with respect to the baseline WER, is to emphasize the importance of techniques described in this thesis work that improve the WER for spell mode utterances, as well as respect Nuance Communication's policy.

The experimental study which will be presented in this chapter, was performed to evaluate the effect of the approaches described in Section 3 on ASR performance for spell mode utterances. The data set consisted of a total of 8000 test utterances collected from 735 speakers performing voice search under one of three categories including "website", "stock quote" and "business names". These include 2000 utterances in the website category, 4000 in the stock quote category, and 2000 in the business name category. Statistical letter based language model for each category has been trained and iteratively tuned using the method discussed in Section 3.4. Confusion matrices were trained by aligning decoded and reference letter and phoneme sequences on held out training utterances. Grapheme databses were compiled for each of the three search categories. The phonemic expansions were obtained and then the phoneme databases were also compiled for the three categories. The database for the website category was originally defined as the names of the top one million websites as measured by user tracking information provided by Alexa. The database for the stock quote category consists of 9500 entries originally taken from the NYSE, AMEX, and NASDAQ exchanges. Finally, the database for the business names category consists of approximately 40,000 entries acquired from various corpora and augmented by customer searches in this category.

4.2 Baseline System Performance

This section provides a brief introduction of the baseline system performance. The baseline system is the original ASR system used in this mobile voice search service, which is without running the second pass re-ordering process. The number of candidate sequences in the ASR N-best list is $n = 10$. The baseline WER refers to the recognition accuracy of the top ASR candidate. The "real time spent" refers to the time spent on decoding an utterance by the ASR engine. Both the actual baseline WER and real time spent is not given here due to intellectual property protection. In order to demonstrate the performance improvement of the additional second pass re-ordering process, in this chapter both the WER reduction and the execution time of re-ordering process will presented in terms of relative improvement with respect to baseline system performance.

4.3 Experimental Analysis

In this section, a detailed experimental analysis will be presented. With the additional re-ordering process, the voice search system is implemented as a two stage search where string candidates generated by an ASR system are re-ordered by consulting potential very large application database corresponding to the specific decoded utterance, pronunciation rules and LM probabilities associated with the search category. By doing so, external knowledge sources are used to help identify the best database entry according to the given ASR candidate.

4.3.1 Experiment Results of Manual Input

Before conducting experiment on the entire test set, potential erroneous letter sequences have been input manually. The first experiment is to examine the performance of grapheme alignment. Table 4.3 demonstrate the performance on voice search queries in "stock quote"

Input Word	Correct Word	Top 3 Reordered Candidates	Edits made
I B N	I B N	I B N	No edits made
		I B M	1 Substitution
		A B M	2 Substitutions
N U V N	N U A N	N U V M	1 Substitution
		N U A N	1 Substitution
		N U V A	1e Substitution
B I I D	B I D	B I I B	1 Substitution
		B I D	1 Deletion
		B I D U	1 Deletion, 1 Insertion

Table 4.3 Grapheme Alignment Results of manual inputs

category. The grapheme alignment function first takes erroneous stock symbols as input, then finds the optimum stock quote database entry based on spell correctness, and finally outputs the the best match. By doing so, all outputs are ensured to be valid stock symbols. According to confusion matrix of stock quote category, substitution cost is usually smaller than deletion or insertion cost. It is obvious that re-ordering process preferably substitutes mis-spelled letters than looks for any deletions or insertions. The result is satisfactory since

the correct words are contained in top 3 re-ordered candidates.

The second experiment is to examine the performance of phoneme alignment. Table 4.4 shows the top 3 candidates suggested by phoneme alignment for the same potential erroneous input used in Table 4.3. The results shows phoneme alignment is also working properly. Some common ASR errors can be proven in Table 4.4, such as "I" and "A" or "B" and "D".

Input Word	Correct Word	Top 3 Reordered Candidates	Edits made
I B N	I B N	I B N	No edits made
		A B M	2 Substitutions
		I B M	2 Substitutions
N U V N	N U A N	N U V M	1 Substitution
		N U V A	1 Substitution
		N U A N	1 Substitution
B I I D	B I D	B I I B	1 Substitution
		B I D	1 Deletion
		B L D	1 Deletion,1 Substitution

Table 4.4 Phoneme Alignment Results of manual inputs

The last experiment is enabling all of grapheme, phoneme alignment and LM score, to find the optimum database entry given by the discriminative model combination discussed in Section 3.8. Table 4.5 shows the additional effect given by LM cost. The result boosts the confidence that the whole second pass re-ordering process is working properly and giving desired results. Now that the second pass re-ordering process performs as expected, the experiment can be augmented by testing the system on other service categories with a much bigger test set.

4.3.2 Experiment Results of Entire Test Set

Table 4.5 presents the performance of the second pass re-ordering system which has been described in Section 3 on three different search categories. The results are given in the format of the relative reduction in WER with respect to the WER obtained by the baseline system with refers to the "ASR" system in Figure 1.1. The number of N-Best ASR candidates is "n = 10". The actual baseline WER is not given here. The relative WER

Input Word	Correct Word	Top 3 Reordered Candidates	Edits made
I B N	I B N	I B N	No edits made
		I B M	1 Substitutions
		A B M	2 Substitutions
N U V N	N U A N	N U A N	1 Substitution
		N U V M	1 Substitution
		N U V A	1 Substitution
B I I D	B I D	B I I B	1 Substitution
		B I D	1 Deletion
		B I D U	1 Deletion, 1 Insertion

Table 4.5 Performance of second pass re-ordering process on manual inputs

reductions for each category is presented separately in corresponding place.

Re-ordering method	Stock Quote	Websites	Bussiness Names
GA	18.3%	24.1%	17.2%
GA+ PA	19.5%	25.2%	21.0%
GA + PA + LM	22.6%	25.9%	23.0%

Table 4.6 Relative reduction in WER obtained using grapheme alignment (GA), phoneme alignment (PA) and language model (LM) scores

There are several observations that can be concluded from Table 4.6. First, website names normally contains more than 4 letters, some of them even contain over 10 letters; therefore grapheme constraints have the biggest effect. Column 3 in Table 4.6 shows that by only utilizing grapheme alignment, website name test has the best performance among three, which comes at 24.1% relative improvement over baseline WER. Second, most users periodically check popular stock quotes. According to column 2 in Table 4.6, a LM helps improve WER 22.6% against the WER baseline. Third, it can be observed that pronunciation rules for stock quote and website tests didn't help significantly. It is because that most stock symbols and websites names are not regular English words, instead it is a list of letters which are normally acronyms for a listed company or a group of letter combination that hasn't be registered on Internet. However for business name test set, we found that pronunciation rules did a better job than the other two. This is understood that business names are usually a combination of English word which ought to be easy memorized and pronounced. Overall speaking, it is clear from the second row of Table 4.6 that the

grapheme based alignment results in relative WER reduction of 17-24 percent. The effect of this procedure is most pronounced for the website category. The third row of the table shows that with a combination of grapheme and phoneme alignment, it results in a small additional reduction in WER. This effect is less pronounced for the stock quote and website categories since many of the query terms in these categories are acronyms or unusual proper names. The last observation relates to the inclusion of the language model (LM) scores in the second pass re-ordering of the ASR hypotheses. The forth row of Table 4.6 shows that the letter based language model (LBLM) in second pass re-ordering had a significant impact on the stock quote and business name search categories.

4.3.3 Modified Beam Search

Another issue discussed in previous chapters, is the trade-off between search efficiency and recognition accuracy. According to Section 3.6, beam pruning saves memory in search; however there is no guarantee that it will find the best solution, since the optimal path could potentially be pruned. The following figure provides a comparison of the execution times and WER reduction associated with the static beam pruning (SBP) and dynamic beam pruning (DBP) DP search strategies described in Section 3.6. The two curves show the relative WER reduction and the execution time as a percentage of real time (%RT) for the "GA + PA + LM" approach applied to the website search category. It is clear from the figure that the DBP strategy provides significantly higher improvements in WER over a range of execution times (%RT=[0.1,0.5]).

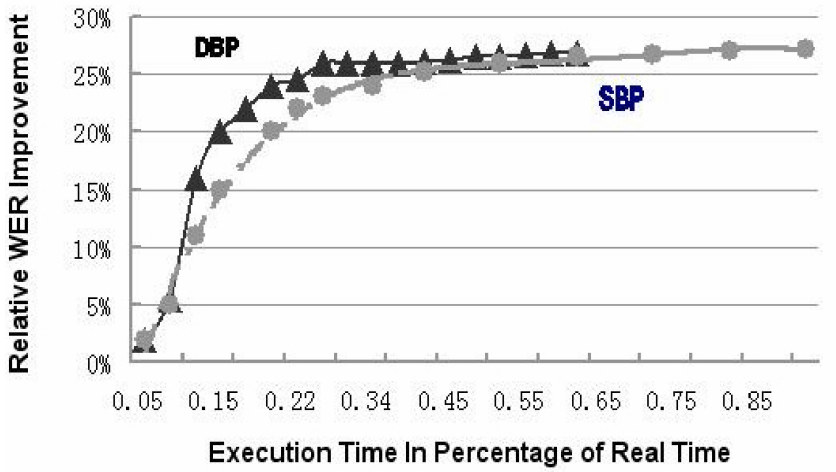

Fig. 4.1 Speed / accuracy comparison of static beam pruning (SBP) and dynamic beam pruning (DBP) implementations of DP rescoring

Chapter 5

Conclusion

This chapter concludes the entire thesis and suggests promising future works that can be done by using techniques developed in this thesis work. The main goal of this thesis is to make the the many external knowledge sources accessible to users of hand-held mobile devices in mobile voice search service.

5.1 Summary of Second Pass Re-ordering Process

This section summaries the problem motivation and the improvement achieved by applying the techniques developed in this thesis. In the context of a voice web search service, there are several problems related to the use of speech recognition software. One problem is, the vocabulary databases of textual web search is vast and contains many names, new words and special terms. However these words are likely unknown words to an ASR system. In other words, the vocabulary that an ASR system supports is much too small to accommodate the millions of words in current web search world. Another problem is, the databases contain many ASR errors and common spelling errors. Because for the majority of the data entered by users, there is no method that allows an ASR system to comprehend the data in a very robust way.

This thesis has presented techniques for improving the performance of a service providing voice search capability on mobile devices. These techniques were evaluated using utterances collected from users of an actual voice search service. Since the ultimate ob-

jective of this research is to optimize the performance of spell mode utterances, those utterances containing spoken letter sequences have been particularly kept for analysis. It was observed that in over a third of the interactions, users uttered spoken letter sequences as part of their query. The multiple pass procedures presented in the thesis were shown to have the effect of reducing WER by as much as 25.9% for website names. And it performs better on longer letter spans which are common English words and follow regular pronunciation rules. It is also shown that the procedures could be efficiently implemented even when the database being searched contains over a million entries. According to the dynamic beam pruning implementation, the system obtains good accuracy without loosing too much of process speed. Overall this thesis provides a good example of how additional domain specific knowledge sources can be used with a domain independent ASR system to facilitate voice access to online search index.

5.2 Potential Future Work

This section lists a number of issues which have been limitations in the thesis, as well as possible approach which has not been investigated in the thesis but however could be worked on in continuing research. This section also proposes potential application of the techniques developed in this thesis in other projects.

First of all, the training data and test data is not perfect. In Chapter 4, the results for our experiments have been presented. All of the results are based on letter recognition accuracy measured on Nuance's mobile voice web search test corpus. Since there are not enough training data for letter based confusion matrix, those non-occurred unpopular mistakes used the statistics obtained from literature. Moreover, the statistical language model require a very large training set in order to calculate bi-gram and tri-gram probabilities. Due to the lack of training data and test data, the models and experiment results discussed in this thesis may have limitations and inaccuracies. In future, as we collect more pseudo-truth data (transcription), we will be able to obtain a more precise confusion matrix and train better language models. At that time, an improved SLM and confusion matrix will improve the overall performance of the second pass re-ordering process.

Secondly, there are several approaches which may be worth investigating in future. For example, in Chapter 2 where recent literature in speech recognition field has been reviewed, some of the research proposed extra constraints can further improve recognition accuracy. Particularly if the ASR system with additional second pass re-ordering process can send a N-best candidate list to the user, and meanwhile be able to retrieve the correct candidate that the user selects, that will save a lot of time consumed by transcribing voice search queries. And the feedback data can be used in training session. Once the entire system is automated, it becomes a self adaptive system which updates models automatically and periodically.

Additionally the techniques developed in this thesis can be applied in other services. For instance, it can be applied very well in automatic booking system where user makes reservation for flights including city names and human names. Moreover if the relationship between the size of database entry and recognition accuracy for each service category can be properly modeled, the second pass re-ordering process then can be improved. Specifically if the WER can not be significantly improved after the size of database entry reaches a certain level, it is unnecessary to keep adding OOV words into the database. However if adding OOV words into database does improve overall performance, it is suggested to do so. The ultimate goal of this thesis work is to improve retrieval accuracy for OOV words and comprehend voice query literally. Based on experiment results shown in Chapter 4, I am very much confident that with continuous research, the second pass re-ordering process will eventually be used in commercial speech recognition service and in the near future, the mobile voice web search service offered by Nuance Communication will become more successful and convenient by contributing to people's daily life.

References

[1] J. Hein, "An algorithm combining DNA and protein alignment," *Journal of Theoretical Biology*, vol. 167, no. 2, pp. 169 – 174, 1994.

[2] G. Wang, H. Chen, and H. Atabakhsh, "Automatically detecting deceptive criminal identities," *Communication of the ACM*, vol. 47, no. 3, pp. 70–76, 2004.

[3] A. Acero, N. Bernstein, R. Chambers, Y. Ju, X. Li, J. Odell, P. Nguyen, O. Scholz, and G. Zweig, "Live search for mobile: Web services by voice on the cell phone," in *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing*, pp. 5256–5259, March 31-April 4 2008.

[4] A. Franz and B. Milch, "Searching the web by voice," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 2*, (Taipei,Taiwan), pp. 1–5, 2002.

[5] J. Li, Y. Tsao, and C.-H. Lee, "A study on knowledge source integration for candidate re-scoring in automatic speech recognition," in *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing - Volume 1*, pp. 837–840, March. 2005.

[6] X. Huang, A. Acero, and H. W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. New Jersey: Prentice Hall PTR, 2001.

[7] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, pp. 357 – 366, Aug. 1980.

[8] K. Kirchhoff, *Phonus 5*. Institute of Phonetics, University of the Saarland, 2000.

[9] J. Sun, X. Jing, and L. Deng, "Data-driven model construction for continuous speech recognition using overlapping articulatory features," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP) - Volume1*, pp. 437–440, 2000.

[10] R. P. C.H. Lee, L.R. Rabiner and J.G.Wilpon, "Acoustic modeling for large vocabulary speech recognition," *Computer Speech and Language*, vol. 4, no. 2, pp. 127–165, 1990.

[11] N. Chomsky and H. Morris, *The Sound Pattern of English*. New York: Harper & Row, 1968.

[12] J. Harris, *English Sound Structure*. Wiley-Blackwell, 1994.

[13] J. Allen, *Natural Language Understanding*. Benjamin-Cummings Publishing Co., Inc., 1995.

[14] P. Jackson and I. Moulinier, *Natural Language processing for online applications: text retrieval, extraction, and categorization*. MIT Press, 2002.

[15] R. Giegerich, C. Meyer, and P. Steffen, "A discipline of dynamic programming over sequence data," *Science of Computer Programming*, vol. 51, pp. 215–263, 2004.

[16] A. K. Richard Durbin, Sean Eddy and G. Mitchison, *Biologic Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[17] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA: The MIT Press, 2004.

[18] G. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[19] U. Reichel, H. R. Pfitzinger, and H. Hain, "English grapheme-to-phoneme conversion and evaluation," *Speech and Language Technology*, vol. 11, pp. 207–212, 2008.

[20] A. W. Black, K. Lenzo, and V. Pagel, "Issues in building general letter to sound rules," in *3rd ESCA Workshop on Speech Synthesis*, (Jenolan Caves, Australia), pp. 77–80, 1998.

[21] M. Divay and A. J. Vitale, "Algorithms for grapheme-phoneme translation for English and French: applications for database searches and speech synthesis," *Computational Linguistics*, vol. 23, no. 4, pp. 495–523, 1997.

[22] A. F. Llitjos and A. W. Black, "Knowledge of language origin improves pronunciation accuracy of proper names," in *Proceedings of Eurospeech*, (Aalborg, Denmark), pp. 1919–1922, 2001.

[23] M. Bisani and H. Ney, "Joint-sequence models for grapheme-to-phoneme conversion," *Speech Communication*, vol. 50, no. 5, pp. 434–451, May. 2008.

[24] J. M. G. Patrizia Bonaventura, Fabio Giuliani and I. Ortin, "Grapheme-to-phoneme transcription rules for Spanish, with application to automatic speech recognition and synthesis," in *Proceedings of the Workshop on Partially Automated Techniques for Transcribing Naturally Occurring Continuous Speech*, Transcribe '98, (Montréal, Québec, Canada), pp. 33–39, 1998.

[25] A. G. Xiao Li and A. Acero, "Adapting grapheme-to-phoneme conversion for name recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, (Kyoto, Japan), pp. 130–135, 2007.

[26] L. Galescu and J. Allen, "Pronunciation of proper names with a joint n-gram model for bi-directional conversion grapheme-to-phoneme conversion," in *Proceedings of International Conference on Spoken Language Processing*, (Denver, Colorado), pp. 109–112, Sep. 2002.

[27] J. Suontausta and J. Hakkinenen, "Decision tree based text-to-phoneme mapping for speech recognition," in *Proceedings of International Conference on Spoken Language Processing*, (Beijing, China), pp. 831–834, 2000.

[28] A. K. Kienappel and R. Kneser, "Designing very compact decision trees for grapheme-to-phoneme transcription," in *Proceedings of the International Conference on Spoken Language Processing*, (Aalborg, Denmark), pp. 1911–1915, Sep. 2001.

[29] P. Beyerlein, "Discriminative model combination," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, (Seattle,WA,USA), pp. 481–484, May 12-15 1998.

[30] M. Kamvar and S. Baluja, "A large scale study of wireless search behavior: Google mobile search," in *Proceedings of the SIGCHI Conference on Human Factors in computing systems*, (Montréal,Québec,Canada), pp. 701–709, 2006.

[31] K. Vertanen and P. O. Kristensson, "Recognition and correction of voice web search queries," in *Proceedings of the International Conference on Spoken Language Processing*, (Brighton, UK), pp. 1863–1866, Sep. 2009.

[32] M. Schuster, "Speech recognition for mobile devices at Google," in *PRICAI 2010: Trends in Artificial Intelligence*, vol. 6230, pp. 8–10, 2010.

[33] G. Heigold, G. Zweig, X. Li, and P. Nguyen, "A flat direct model for speech recognition," in *IEEE International Conference on Acoustic, Speech and Signal Processing, 2009. ICASSP 2009*, pp. 3861–3864, April 2009.

[34] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[35] S. Parthasarathy, "Experiments in keypad-aided spelling recognition," in *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing - Volume 1*, pp. 873–876, May 17-21 2004.

[36] G. Chung and S. Seneff, "Integrating speech with keypad input for automatic entry of spelling and pronunciation of new words," in *Proceedings of the the 7th International Conference on Spoken Language Processing*, (Denver,Colorado,USA), pp. 2061–2064, Sep 16-20 2002.

[37] G. Chung, S. Seneff, and C. Wang, "Automatic acquisition of names using speak and spell mode in spoken dialog systems," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, (Edmonton,Canada), pp. 32–39, 2003.

[38] E. Roche and Y. Schabes, *Finite-state Language Processing*. MIT Press, 1997.

[39] M. Mohri, "Finite-state transducers in language and speech processing," *Association of Computational Linguistics*, vol. 23, pp. 269–311, June 1997.

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

[41] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational Linguistics*, vol. 18, pp. 467–479, Dec. 1992.

[42] K. W. Church and W. A. Gale, "Probability scoring for spelling correction," vol. 1, pp. 93–103, 1991.

[43] S. Singh, "The black chamber-letter frequencies." [Available Online] http://www.simonsingh.net/The_Black_Chamber/frequencyanalysis.html (Consulted on 17 October 2008).

[44] E. Brill and R. C. Moore, "An improved error model for noisy channel spelling correction," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, (Hong Kong), pp. 286–293, 2000.

[45] S. Cucerzan and E. Brill, "Spelling correction as an iterative process that exploits the collective knowledge of web users," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Barcelona, Spain), pp. 293–300, 2004.

[46] L. Xie and L. Du, "Efficient Viterbi beam search algorithm using dynamic pruning," in *Proceedings of the 7th International Conference on Signal Processing*, vol. 1, (Beijing,China), pp. 699–702, 2004.

[47] H. Ney, R.Haeb-Umbach, B.H.Tran, and M.Oerder, "Improvements in beam search for 10000-word continuous speech recognition," in *Proceedings of the IEEE International*

Conference on Acoustics, Speech and Signal Processing, vol. 1, pp. 9–12, March 23-26 1992.

[48] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," in *Readings in Speech Recognition*, pp. 159–165, 1990.

[49] D. Lorens, F. Casacuberta, C. Martinez, S. Molau, F. Nevado, H. Ney, M. Pastor, D. Pico, A. Sanchis, E. Vidal, and J. Vilar, "Speech-to-speech translation based on finite-state transducers," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, (Salt Lake City, Utah, USA), pp. 613–616, 2001.

[50] A. Mikheev, "Document centered approach to text normalization," in *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and development in information retrieval*, SIGIR '00, (Athenes, Greece), pp. 136–143, 2000.

[51] "Top 1 million websites." [Available Online] http://www.Alexa.com/ (Consulted on April 2009).