Efficient Error Correction for Speech Systems using Constrained Re-recognition

by

Gregory T. Yu

S.B., Massachusetts Institute of Technology (2007)

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

July 2008

Author______________________________
Department of Electrical Engineering and Computer Science
July 17, 2008

Certified by______________________________
James R. Glass
Principal Research Scientist
Thesis Supervisor

Certified by______________________________
Lee Hetherington
Research Scientist
Thesis Co-Supervisor

Accepted by______________________________
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Efficient Error Correction for Speech Systems using Constrained Re-recognition

by

Gregory T. Yu

Submitted to the
Department of Electrical Engineering and Computer Science

July 17, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Efficient error correction of recognition output is a major barrier in the adoption of speech interfaces. This thesis addresses this problem through a novel correction framework and user interface. The system uses constraints provided by the user to enhance re-recognition, correcting errors with minimal user effort and time. In our web interface, users listen to the recognized utterance, marking incorrect words as they hear them. After they have finished marking errors, they submit the edits back to the speech recognizer where it is merged with previous edits and then converted into a finite state transducer. This FST, modeling the regions of correct and incorrect words in the recognition output, is then composed with the recognizer's language model and the utterance is re-recognized. We explored the use of our error correction technique in both the lecture and restaurant domain, evaluating the types of errors and the correction performance in each domain. With our system, we have found significant improvements over other error correction techniques such as n-best lists, re-speaking or verbal corrections, and retyping in terms of actions per correction step, corrected output rate, and ease of use.

Thesis Supervisor: James Glass
Title: Principal Research Scientist

Thesis Co-Supervisor: Lee Hetherington
Title: Research Scientist

Acknowledgments

First of all, I would like to thanks my advisors, Jim Glass and Lee Hetherington, for their guidance, their support, both technical and logistical, and their insightful feedback. When I needed direction after hitting a snag in my experiments, they patiently found a way for me to move forward. I am very appreciative of the time they took each week to work with me in spite of their hectic schedules. This thesis has been a pleasure to work on, and I'm sincerely grateful for the opportunity provided by my advisors and all of SLS.

Since joining SLS two years ago, I've worked on a number of exciting and challenging projects that have broadened my knowledge and given me the chance to collaborate with some of the many brilliant and welcoming researchers in the lab. I would like to thank Scott Cyphers for his tireless technical assistance on all matters, no matter how arcane. I would also like to thank Sean Liu and Alex Gruenstein for their assistance with the restaurant domain.

I would also like to thank my friends at MIT for keeping me motivated outside of lab, telling me I should be working on my thesis instead of playing computer games, as well as just being there to listen when I was stressed or unsure of what to do.

Finally, I want to thank my parents for their patience with the repeatedly delayed move-out date, and their constant support and sacrifices throughout my life.

This research was sponsored by Nokia as part of a joint MIT-Nokia collaboration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid growth of computing power and the steady increase in recognition accuracy of speech recognition systems, speech interfaces are becoming more attractive and accessible. Despite this, many users are still abandoning speech recognition systems soon after trying them, going back to conventional interfaces. The problem is not a lack of accuracy or speed, but rather one of usability. Any recognition system, be it handwriting, vision, or speech, will naturally have errors. With speech systems thus far, recovering from these errors has not been graceful, and at times, efforts to correct recognition mistakes lead to further errors. A chief source of user frustration is in the detection and correction of errors. In order for speech recognition to be widely adopted and fulfill its potential as a natural, high-bandwidth interface, there must be improvements in error handling.

There are a number of error correction techniques currently used in speech recognition systems. The most commonly used approaches are n-best lists, verbal corrections, and nothing at all, forcing the user to type in the correct text or manually choose the intended action. Correction schemes of these types are either extremely slow or inefficient. With n-best lists, the user is left to search through many lines of hypotheses with the high chance that the desired transcription is not in the list at all. Re-speaking leads to lowered recognition accuracy due to hyper-articulation, the type of over-emphasizing that people do when they are misunderstood by another person [1]. The nature of re-speaking is problematic to most users, since without knowledge of the system model, how should one speak to be understood better? That two seemingly identically spoken statements might be recognized differently leads to confusion and distrust. Finally, the hassle of manually correcting and retyping the utterance results in users spending more time than if they had chosen a different modality in the first place. This is especially true for mobile devices, an area where speech systems are gaining increasing interest, where manual corrections would be done with a numeric keypad, a slow input modality.

The Spoken Language Systems group has produced a range of user applications to demonstrate and evaluate its speech technologies, most recently in the lecture, address, and restaurant domains [2, 5]. These applications have some features in common—they share the same core recognition framework, SUMMIT [7], and they are all web systems using server-based recognition. This work focuses on a flexible error correction system integrated into SUMMIT with a web interface, allowing for fast, accurate corrections of recognition output across domains and platforms.

1.1 Motivation

There are two major factors driving the relevance of speech interfaces—the increasingly universal presence of computer interfaces across the spectrum of environments and situations, and the reduction in size and input modalities of many modern computer systems. We now find people interacting with computers in new environments, such as home media centers, their cars, and virtually anywhere else via cellphones and other mobile devices. In addition to the increased distractions present in these new environments, both physical and mental, the systems used all lack the full input capabilities of standard desktop computers. The interfaces to these systems typically rely on limited sets of buttons, touch screens, or numeric keypads. While these interfaces may be sufficient for a limited set of predefined tasks, they are ill-suited for more open tasks like searching through vast collections of data.

Speech recognition systems are ideally suited for these new environments and devices. Aside from learning the command set, no additional training is required, making system interactions feel natural for both novice and expert users. Speech is hands-free—the output channel used does not intersect with those used for critical tasks in its usage environments. Speech has a potentially high bandwidth, with output in the range of 100-200 words per minute. This bandwidth remains a potential however, as it has not been realized with existing speech interfaces. Even with relatively high accuracy rates, the time required to review and correct a speech utterance or the incorrect command that resulted from it can drive down input rates to well below that of other conventional input techniques. One study [8] found users could dictate at an uncorrected rate of 102 words per minute. After factoring in the time for correction, the output rate dropped to less than 10 words per minute. For speech systems that use re-speaking as their correction mechanism, users often find themselves correcting their corrections resulting in an inescapable error spiral [9].

Aside from hampering a user's input rate, errors lower subjective satisfaction with the interface, and speech systems in general. In one study, users with disabilities that prevented them from using a standard keyboard and mouse were given a speech recognition system, and after 6 months, almost all users abandoned the speech interface in favor of alternative input methods [10]. Addressing errors by attempting to increase recognition accuracy is insufficient—there needs to be a fast and easy way for users to correct recognition output.

1.2 Goals

The goal for this work is to create an effective error correction interface for speech recognition systems that will allow users to quickly and easily correct recognition output. The interface must be general, easy for users to interact with, and allow for improved output rates compared to other error correction techniques.

The correction interface must be general, both in terms of its front-end and its back-end implementation. As computer interfaces appear in new environments, so too will speech interfaces. No matter the environment or speech system, recognition errors will affect user performance and satisfaction. Thus, it is essential that error correction is not tied to a specific modality, such as a keyboard or mouse. In particular, mobile devices like cellphones and tablet PCs must deal with the constraint of awkward text entry and small screen size. Secondly, this approach should not be tied to a particular domain or recognition architecture. Our technique should be as widely applicable as alternates lists and re-speaking are now.

The correction technique must be easy to use, reducing the user's cognitive load during the correction process. With re-speaking, the user must think about what the correct word is, how they're supposed to say it differently so it will not be mis-recognized again, and pay attention to the system response to watch for the start of an error spiral. With alternates lists, the user must scan through each alternate, trying to identify the correct phrase. If the correct utterance is not there, the user must go through the list again, figuring out the differences between each output and the original recognition result and then deciding which one is closest. The guiding principle of usability that we will follow in our approach is that recognition is much easier than recall—identifying errors can be done faster than correcting them.

Finally, the most important metric for evaluating our approach to correction is overall output rate. Speech systems should not be relegated to hands-free environments, but rather they should be a first-class input technique. In order for this to happen, the corrected output rate must match or exceed that of other modalities. To accomplish this, we will investigate bringing the recognizer into the loop during correction to use constraints provided by both the user and system to improve correction time.

1.3 Outline

The remainder of this work is organized as follows:

An overview of related work in the field of error correction for speech systems is covered in Chapter 2. Background information on finite state transducers and their application to our error correction technique is given. Finally, the lecture and restaurant domains are discussed, highlighting their challenges and general characteristics.

Chapter 3 covers our approach to error correction, describing the design of our constraint FSTs used in re-recognition. In addition, we discuss the challenges encountered implementing this approach and how they were resolved.

Chapter 4 covers the web-based user interface, reviewing its design and features. The system flow for a sample user interacting with our interface is shown.

Our evaluation methodology, experimental setup, and results are presented in Chapter 5. Statistics on the errors found in our experiments are also covered. The results of oracle and speaker adaptation experiments are discussed. An analysis of theoretical efficiency is presented using predictive evaluation.

Chapter 6 provides a summary and discussion of future work.

Chapter 2

Background

This chapter first gives a brief survey of previous work in error correction for speech systems, highlighting the characteristics of each with respect to our goals. This is followed by a discussion of the technologies which form the basis of our approach and the speech system we will be working with. The domains used in evaluation are presented, in addition to their related challenges. Finally, the last section covers the two usage scenarios that our approach will attempt to address.

2.1 Error Handling in Speech Systems

A number of approaches to error handling have been explored in the past. In this section we will review these approaches and discuss their strengths and weaknesses.

2.1.1 Alternates Lists

Alternates lists, or n-best lists are one of the most basic forms of error handling. An n-best list contains the top n recognition hypotheses ranked by their score, calculated through a weighted sum of its acoustic probability and language model probability. As we can see in Figure 2-1, n-best lists can be hard for users to read. Often, hypotheses will be differentiated by small differences—the entire list may just be variations of a filler word like "a," "the," "uh," or "um." A user might take several minutes reading the n-best list below, carefully reviewing each candidate, and still not pick the one which minimizes WER.

by d n a d if i give you the matrix let me come back now for them a boy or i might <cough> little the pieces and i might ask you something about the matrix for example
by d n a d if i give you the matrix let me come back now for them anymore or i might <cough> little the pieces and i might ask you something about the matrix for example
find the f b d if i give you the matrix let me come back now for them a boy or i might <cough> little the pieces and i might ask you something about the matrix for example
find the f b d if i give you the matrix let me come back now for them anymore or i might <cough> little the pieces and i might ask you something about the matrix for example
by d s p d if i give you the matrix let me come back now for them a boy or i might <cough> little the pieces and i might ask you something about the matrix for example
by d n a d if i give you the matrix let me come back out of the main or or i might <cough> little the pieces and i might ask you something about the matrix for example
by d n a d if i give you the matrix let me come back now for them a boy or i might give you the pieces and i might ask you something about the matrix for example
by b s e d if i give you the matrix let me come back now for them a boy or i might <cough> little the pieces and i might ask you something about the matrix for example
find the n b d if i give you the matrix let me come back now for them a boy or i might <cough> little the pieces and i might ask you something about the matrix for example
by d s p d if i give you the matrix let me come back now for them anymore or i might <cough> little the pieces and i might ask you something about the matrix for example

Figure 2-1: A sample 10-best list for an utterance in the lecture domain.

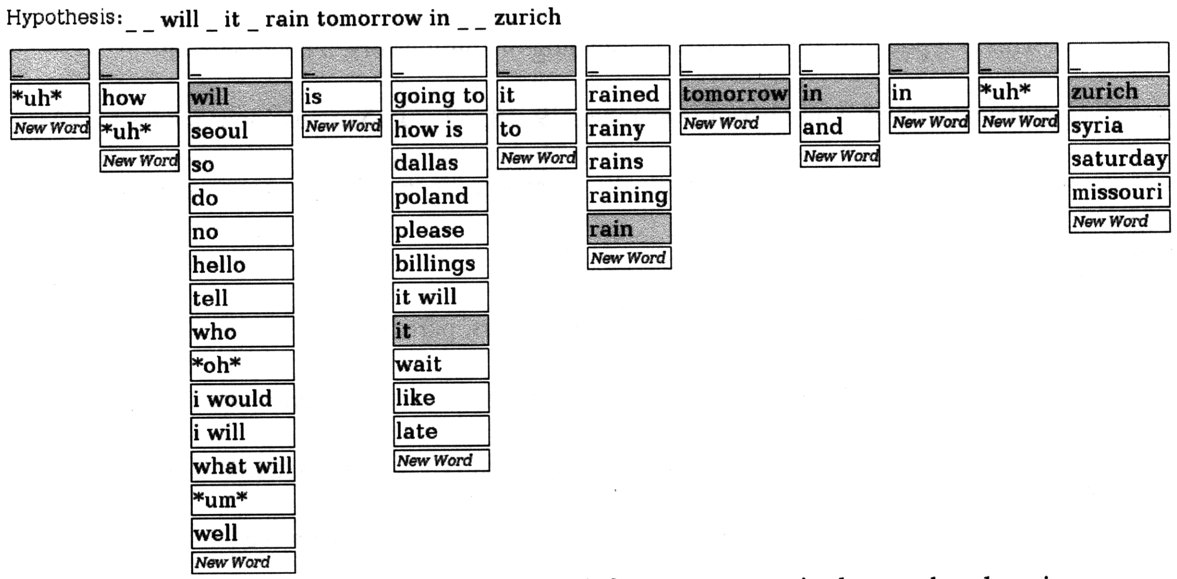
N-best lists most commonly have an utterance level granularity, although some speech interfaces use word-level alternates lists, accessed by selecting a specific word. Generally, these lists are limited to 5-10 entries as it becomes difficult for users to scan through any more than that. With utterance level alternates and the number of possible alternates roughly exponential with the number of words in the utterance, it becomes unwieldy to use this approach for sentences with a moderate amount of errors.

Modifications to improve alternates lists have typically come in the form of re-ranking algorithms. The Open Mind Common Sense Project has collected over 700,000 "common sense" facts to re-rank candidate hypotheses [11]. This approach helps to disambiguate between phonetically similar words by choosing the most semantically probable. Semantic context takes advantage of long-range dependencies, exceeding the 2-3 word limit of bigrams and trigrams. This approach reduces word error rate by 17% relative, and reduces dictation time by 7.5%.

Stolcke et al explored optimizing the hypotheses ranking algorithm to minimize word error rate [12]. While hypotheses are typically ranked by their posterior probability, in effect minimizing sentence error rate, this does not result in optimal word error rates. The expected word error rate of a given hypothesis is computed by summing the error rates relative to each of the other hypotheses and weighing it by its posterior. This method results in a modest decrease in word error rate of .5% absolute.

2.1.2 Confusion Networks

Confusion networks are similar to word level alternates lists, but alternate candidates for each word are displayed. They are an algorithmic transformation of the weighted word lattice generated during recognition, and can be used to display more candidates than a standard n-best list [13, 14]. As seen in Figure 2-2, a dense word lattice can turned into a very compact confusion network.


Hypothesis: _ _ will _ it _ rain tomorrow in _ _ zurich
_ *uh* New Word
_ how *uh* New Word
_ will seoul so do no hello tell who *oh* i would i will what will *um* well New Word
_ is New Word
_ going to how is dallas poland please billings it will it wait like late New Word
_ it to New Word
_ rained rainy rains raining rain New Word
_ tomorrow New Word
_ in and New Word
_ in New Word
_ *uh* New Word
_ zurich syria saturday missouri New Word
Figure 2-2: A sample confusion network for an utterance in the weather domain.

Whereas an n-best list only has n alternatives, a confusion network m words long with n candidates per word would have n^m alternatives. Confusion networks are used as the error correction mechanism to edit podcast transcripts in the PodCastle web service [17].

Confusion networks are aligned to form word "sausages" where arcs always go from state i to i+1. As a result, if a multisyllabic word is recognized as multiple monosyllabic words, it must either be aligned with the first word of the sequence, or the entire sequence must be aligned with the multisyllabic word. Either way can result in a confusion network that is hard for a user to parse, making correction a cognitively challenging process.

2.1.3 Spoken Corrections

Spoken corrections take the form of repeating the entire utterance or word(s) that need to be corrected, entering a correction mode or performing a correction through a spoken command, or spelling out the incorrect word ("speak and spell"). While spoken corrections are the only choice in a hands-free environments, outside of that context, they are one of the weaker correction modalities. The biggest danger of using speech for corrections is the potential to misrecognize the correction. This is a phenomenon known as cascading or spiral errors, a phenomenon characterized and analyzed by a number of researchers [9, 18, 19, 20]. Not only does it cause users to get incredibly frustrated, it also dramatically increases the time it takes to correct an utterance.

Another common phenomenon with spoken corrections, particularly re-speaking, is hyperarticulation, a manner of speaking characterized by more pauses between words, longer pauses, and less disfluencies [1]. This "clearer" type of speech is a person's reaction to another human misunderstanding what was said. While stronger enunciation may improve human understanding, it does the opposite for speech recognition systems. Standard acoustic models are not trained on this type of speech and thus recognition performance suffers, leading to higher error rates and cascading errors. Some attempts have been made at predicting and modeling hyperarticulate speech [21], however the best solution is to switch modalities and prevent the problem entirely. Despite the difficulties of speech corrections, users will favor the modality if it is present, and stick with it despite much higher error rates [8]. Consequently, the best approach to multimodal error correction is to eliminate speech as an option when other modalities are available.

2.1.4 Keyboard/Keypad corrections

One of the simplest and most effective error correction techniques is to allow keyboard input during the correction phase. Retyping is especially attractive because it is not a recognition-based input, unlike alternates lists, or re-speaking. What the user types will not be interpreted and thus its contents are predictable. Errors in the correction input, or typos, are easily fixed through the same modality. Keyboard input bandwidth, in the range of 50-100+ wpm, is relatively high compared to that of corrected speech [22].

On the other hand, one of the primary benefits of speech interfaces is that the only required input modality is audio, allowing for greater flexibility in their use. Current applications of speech recognition include mobile devices, kiosks, automated telephone systems, and car-based computer systems. All of these systems do not support full keyboards, providing a keypad or a virtual keyboard at best. Input rates for virtual keyboards are significantly less, averaging around 10 wpm or less with triple-tap or T9 [23].

Using virtual keyboards for constraints, rather than the full correction shows promise. One study [39] investigated the use of a virtual keyboard for re-recognition, comparing it against re-speaking and alternates lists. The virtual keyboard was used to select the first letter of the correct word, constraining the lexicon during re-recognition. This approach was effective 76.4% of the time, with alternates lists effective 36.2% of the time, and re-speaking effective only 11.2% of the time. In this study, the speech interface was for a form used to obtain the timetable for a railway system, with the vocabulary constrained to times, dates, and a set of towns.

2.1.5 Error Detection and Automatic Correction

A large amount of research has been done on error detection and automated correction, looking for ways to improve recognition accuracy and take the burden of error detection off of the user. Error detection techniques typically rely on confidence measures estimated from a set of features which are used by a classifier to determine if a word or utterance is correct or not. The features are derived from two sources—recognizer independent and recognizer dependent information [24]. Examples of recognizer independent features include linguistic information like part of speech or content words, acoustic and language model scores, and posterior word probabilities. Examples of recognizer dependent features reply on intermediate data structures generated by the recognizer like word lattices, confusion networks, and n-best lists. Approaches to error detection often rely on long range dependencies that are not examined in standard n-gram language models. One approach [25] performs a co-occurrence analysis on the training data to determine the context words associated with a given keyword. They then locate instances where a word phonetically similar to a keyword was recognized, but that keywords context words appear around it, signifying a possible error. This type of error detection would be particularly useful for information retrieval tasks where keyword accuracy is important.

Despite the attraction of doing error correction automatically, it would not solve the need for user intervention. Error detection and automated correction should be treated as another element of the recognition pipeline that improves recognition performance. Performing error detection alone as a way of saving the user time during error correction can be dangerous. Users must detect false positives and false negatives—if a marked word is actually correct and if an unmarked word is actually incorrect. While the number of selections may be lowered, the number of decisions remains the same and the types of decisions doubles.

2.1.6 Other Approaches

A number of multimodal approaches use speech for initial recognition and then switch to handwriting for corrections. While this approach leads to a number of issues—a second recognition step using a different recognizer, the need for a writing instrument, the attention that writing requires, etc.—there are some potential benefits as well. Generally speaking, speech errors are orthogonal to handwriting errors. That is, words which are confusable acoustically are usually not confusable orthographically. One speech interface [26] performs corrections through handwriting and written gestures. It uses word and character level editing gestures for each of the error types—substitutions, deletions, and insertions—as well as a gesture to "accept" part of a sentence to prune and re-search the word graph generated during recognition.

2.2 Finite State Transducers

Finite state transducers are a mathematical framework used to represent state-based transitions with input and output labels between states. Transitions can be weighted according to statistical probabilities, allowing one to search for the best path amongst a dense number of alternatives. FSTs are used throughout the SUMMIT recognition pipeline to allow for uniform representation, efficient composition, and optimization. The SUMMIT system makes use of a single FST in the first pass of the form opt(C o P o L o G). FSTs for context-dependent phonetic models, phonological rules, the lexicon, and the n-gram language model are composed together and then optimized to form the recognition framework. In the error correction method we investigated, a fifth FST, the error FST E, is generated and then composed with the output of the n-gram language model. FST operations are done using the MIT FST Toolkit, developed specifically for the use of FSTs in speech recognition [27].

2.2.1 Word Graphs

Word graphs are a data structure used to efficiently search through recognition candidates. Word graphs are constructed as FSTs during recognition, where each arc is given a word label and weighted by its posterior probability. Exhaustively searching all possible candidates is generally far too costly, thus the search space must be constrained. The size of the word graphs is controlled by a number of parameters used during Viterbi beam search. The score threshold, vprune, is the maximum difference in log probability between the top hypothesis and the current one. The size threshold, vprunenodes, is the maximum number of possible candidates allowed at each step. Another size threshold, maxpopspersec, controls the maximum numbers of arcs that can be created each second. In our experiments, we compare the accuracy and computational cost of composing the error FST with the language model and re-recognizing versus composing it with the word graph generated after initial recognition.

2.3 Lecture Domain

The lecture corpus is composed of over 500 hours of speech, with lectures from 13 different courses and over 100 seminars on various topics. The style, topic matter, and speaker characteristics vary widely among the data, making the task of transcribing lectures a difficult one. Lectures, 7,000 words on average, typically contain only 500 to 1,100 unique words [2]. Some fraction of these words are those found in standard conversational speech and can be modeled through training on other lectures or other conversational speech corpora like Switchboard. The remainder of a lecture's unique vocabulary is subject specific keywords (like "eigenvalue" or "matrix"), and can be difficult to model without additional related data such as other lectures in the same course or course-related materials like textbooks.

Lectures are typically recorded with an omni-directional microphone as part of a video recording. Audio quality was less than ideal in a number of lectures, due to the type of microphone (compared to a close-talking headset), audience noise, and room acoustics. Typically each lecture has a main speaker, but some lectures have two or more speakers as well as audience interaction, adding additional acoustic challenges. The lecture domain is challenging for speech recognition, with word error rates as high as 30-50%, making a compelling case for error correction.

2.4 Restaurant Domain

The restaurant test corpus was created through a web-based multimodal dialog system in the restaurant information domain known as City Browser [5]. Roughly 2,200 utterances were recorded through user testing of the restaurant system. Possible queries include specifying preferences based on price, cuisine, or location relative to streets, metro regions, or landmarks, as well as asking for information about a specific restaurant, like phone number, hours, or address. User queries typically take one of two forms—either part of a series of cumulative constraints, narrowing down the set of qualifying restaurants, or a new query which does not build upon the context of prior requests.

In this system, queries are built from static and dynamic vocabularies. The dynamic vocabulary is updated at each dialog turn and contains topic words like the names of streets and restaurants. The dynamic classes were populated by crawling restaurant databases on the web, and pronunciations were derived from existing data or through a letter-to-sound module. Data were collected across seven metropolitan areas with roughly 3,000 to 17,000 restaurant aliases (each restaurant has one or more aliases), 1,100 to 2,200 street names, and 50 to 250 cities and neighborhoods. The recognizer language model was trained on user queries generated automatically by a simulation server [28]. The simulated queries were built using a rule-based recursive text generator. The resulting training set contained 8000 sentences with a static lexicon of 1400 words.

2.5 Usage Scenarios

As a user interface component, error correction must be put in the context of its common usage scenarios. Different ASR systems. domains, user backgrounds, and overall goals will all affect the types of errors that can occur and the needs of the user in repairing them.

2.5.1 Real-time Query Corrections

The traditional use of speech in user interfaces is as an input modality to issue commands or queries. When using such a speech system, corrections are issued by the speaker immediately after recognition. The goal of correction in this case is not necessarily an entirely correct utterance, but rather a correct action. City Browser, for example, has a semantic decoding step after recognition, interpreting the command intended by the user. In this way, both misrecognized and out of grammar utterances can still result in the correct action. Thus, the goal of correction is to minimize concept rather than word error rate. Certain classes of words, like proper nouns, are important to identifying the correct concept, making sequential correction systems [30] inefficient. Random-access correction schemes allow for users to jump to relevant error gaps in the recognition output. This is especially useful for this usage case because the speaker is doing the corrections, and should still remember what was said, and consequently, know what is wrong in the output without having to listen to a recording.

2.5.2 Third-party Corrections

This usage scenario could most commonly be found in the lecture domain with students correcting a professor's lecture transcription. In this scenario, the professor's lecture would be recorded and then transcribed by an ASR system. The output transcription would then be reviewed and collaboratively edited by the students in the class to use as study notes. In this case, users would not necessarily remember what the professor said, so an audio recording to accompany the recognition output would be necessary. Alternatively, the lecture may be automatically transcribed in real-time, and students correct it as they listen to it. Wald et al. investigated a number of keyboard-based interfaces for real-time correction of lecture transcripts generated by an ASR system [31]. They found corrected output rates ranging from 10 to 19 WPM depending on the level of the typist. Our correction scheme will attempt to address both real-time and off-line transcript editing. Unlike the previous usage scenario, students would attempt to minimize word error rate, looking to create as legible transcripts as possible. Perfect accuracy is not required, however, as one study [32] quizzed students on the contents of a lecture accompanied by transcripts with varying error rates. Word error rates of 25% or less resulted in better performance compared to the control group which received no transcript.

Chapter 3

Approach

Our approach to error correction was partially motivated by the work of Kumaran et al. [29] on automatic error detection and correction. In their approach, they noted that humans disambiguate conversational speech through long range context. This context works in both directions, so after hearing a portion of unintelligible speech, a person will use their short-term memory to recall what was said before and after that portion of speech to resolve it. In the same fashion, a speech recognizer's hypotheses can be divided into regions of high and low confidence—"islands" of reliability and error "gaps." The search space for LVCSR systems is far too large to search exhaustively, so there must be score-based pruning. In doing so, a large portion of the search space may be alternate candidates for a highly confident segment, pruning out the correct candidates for a gap. By identifying islands and gaps, one can guide pruning decisions as well as rescore gaps using nearby islands to calculate LM scores.

Kumaran et al. built three word confidence classifiers using boosted decision trees, SVMs, and neural nets to identify the island and gap regions in a recognition hypothesis. Their best classifier had an overall accuracy of 63.47%, identifying islands correctly 60.85% and gaps 80.58% of the time. While promising, this accuracy is insufficient as a user interface component. As with other error classifiers, false positives create a confounding effect that requires extra user attention. Consequently, we adopted a supervised approach to this technique in which users flag sections of recognition output as correct or incorrect, followed by constrained re-recognition.

3.1 Overview

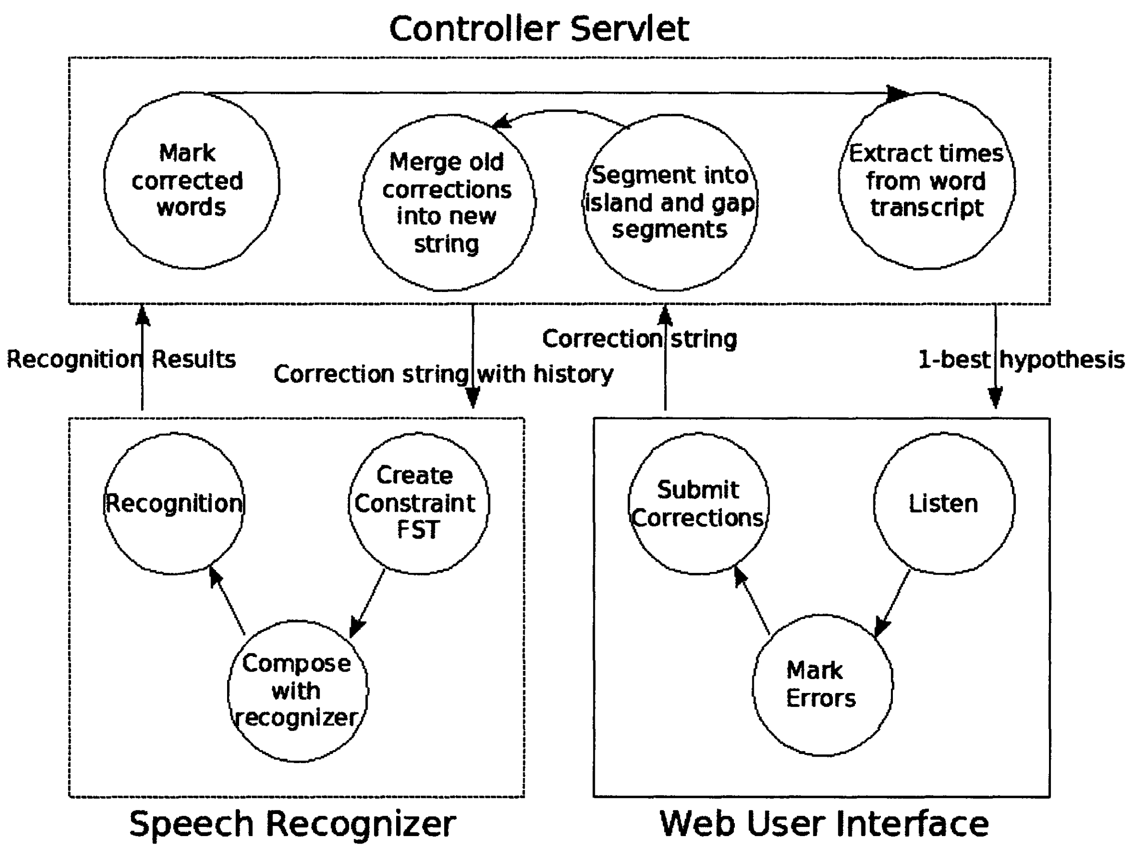


Figure 3-1: The system architecture for our error correction system.

Our error correction system is composed of a web-based graphical user interface, a web server, and a speech recognition server, a diagram of it can be seen in Figure 3-1. Audio, either streamed or prerecorded, is sent to the web server running the controller servlet and then forwarded to the recognizer server via XML-RPC for initial recognition. The n-best list is sent to the web server, and the top candidate is passed to a JavaServer Page which then renders the correction interface. Users manipulate the utterance via a Javascript framework, flagging incorrect regions, through either click or drag gestures. Deletion errors are marked by clicking between words, and substitution or insertion errors are marked by clicking or dragging over words. Users may listen to the audio recording sent to the recognizer through audio controls above the utterance, using a Javascript to Flash interface.

After marking the regions to be corrected, users then submit it back to the web server running the controller servlet. If this is the first correction pass on a given utterance, the correction string is sent to the recognizer server. The recognizer takes the correction string, generates a correction FST and composes it with the right side of the n-gram language model. Acoustic scores are kept from the initial recognition pass, and language scores are recomputed using the new language model. The new n-best is sent back to the servlet and re-rendered on the UI. For subsequent corrections on the same utterance, the servlet keeps an internal history of the previous corrections made so that they can be merged with the current correction string.

3.2 Error Constraint FST

In the following sections, the design, implementation, and challenges of building and using FSTs for correction are discussed.

3.2.1 Design

To enforce the constraints of the correct regions and re-recognize the marked incorrect regions of the speech utterance, we generate a finite state transducer encoding the error constraints. We chose this representation as it allowed efficient integration into the SUMMIT recognition framework, which is composed of a series of FSTs, using composition and optimization operations developed in the MIT FST toolkit. The constraint condition for the correct word segments can be described as fulfilling the following transformation: given a word segment sequence s_1 t_1 s_2 t_2... s_n t_n where all s_i represent a segment of one or more correct words and all t_i represent a segment of one or more incorrect words (however either s_1 and t_n may be empty), output after re-recognition must contain all s_i, and s_i comes before s_j if and only if $i < j$.

For the incorrect word segments, the new candidate text between correct word sequences s_i and s_{i+1} cannot contain t_i or any of its ordered subsets. For example given an incorrect word sequence t_i composed of words $w_1\ w_2 \dots w_n$ the re-recognized segment cannot contain any sequences containing $w_i\ w_{i+1} \dots w_j$ where $j - i \leq 1$. The rationale behind this is that we wish to prevent word sequences which do not conform with the marked corrections. For example, given the sequence "a b c d" with "b c" marked as incorrect, and the error segment was re-recognized as just "b" giving "a b d" then it would imply that "b" was correct in the first place, despite being marked as an error.

However, some permutations of the error sequence are possible and should be permitted. The phrase "to the two, too" might be recognized as "too a to two" which contains a rearrangement of a subset of the incorrect words and it would be valid to mark the entire sequence as incorrect. Alternative ways of marking that phrase are possible as well—one could mark "too a" as incorrect and then indicate a deletion error between "to" and "two" or one could just mark "a to two" as incorrect and then indicate an insertion error before "too." These alternate corrections, while appropriate from a purely textual analysis, are unlikely to align with the corresponding hypothesis phone sequence. This can get arbitrarily complex as the error sequences grow in length, resulting in increasingly dense constraint FSTs to represent them. To avoid this problem of valid and invalid permutations of the error word sequence, our constraint FST does not allow any of the incorrect words in the new candidate sequence. As a result, some word sequences cannot be corrected and must be addressed through an alternative correction scheme. These instances are rare, however, and do not detract from the efficacy of this approach.

3.2.2 Implementation

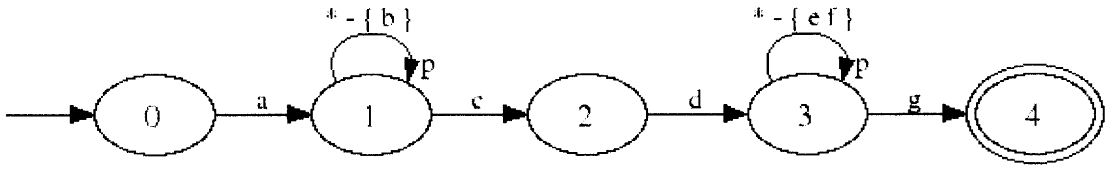


Figure 3-2: The error constraint FST generated for the correction encoding "a (b) c d (e f) g."

The error constraint FST for the sequence "a b c d e f g" where "b" and "e f" are marked as incorrect is shown in Figure 3-2. The constraint FST is encoded as a string when it is passed from the UI to the controller servlet to the recognizer. In our encoding, words marked as incorrect are enclosed in parentheses as follows: "correct (incorrect) correct" and "correct (   )" denotes a deletion error after a correct word. All words within a parentheses are excluded from the outgoing arcs corresponding to that error state in the FST. States with a single outgoing arc from it correspond to a correct word and states with multiple outgoing arcs correspond to a error sequence. The label "* - { word_1, ... , word_n }" corresponds to the arcs leaving the error state, signifying the set of words in the recognizer vocabulary minus the set of words found in that error segment.

Error states contain a self loop as the number of correct words is unknown. Self loops can be problematic as they allow for any error state to consume arbitrarily many phones. As the entire vocabulary is permitted within this state, the optimal language model scores could be obtained through remaining in the error state. Due to pruning during search, it may not be discovered that one ended in a non-final state until all paths out of the error state have already been eliminated. To fix this, a penalty must be added for each time the self loop is traversed, signified by the penalty *p* in the diagram. We examined the effects this penalty parameter has on both WER and re-recognition output in our experiments.

3.2.3 Optimizations

A challenge we ran into was the sheer size of these constraint FSTs—with a vocabulary of over 37,000 words, each error state was enormous. Every sequence of incorrect words represented in the constraint FST essentially contains the entire language model. Additionally, they could not be reused, as each error state has its own set of words excluded from the vocabulary. Our initial implementation, direct composition of the constraint FST with the recognizer, consumed enormous amounts of memory and computational resources, taking minutes to complete composition, or not finishing at all.

The gap states in the error constraint FST contain outgoing arcs corresponding to V - T where V is the set of all words in the vocabulary and T is the set of all words in the error gap. More precisely however, the set of allowable transitions is $\{ V \mid w_i, w_{i-1} \}$ where w_i and w_{i-1} are the words preceding the start of the gap. Only a small subset of words are reachable at any point in time, on the order of hundreds, rather than thousands. To generate the same results as full composition, the constrained recognizer FST is built in place incrementally. Using lookahead to find the set of all words allowable from the current states, dead or invalid arcs are trimmed, reducing space and time requirements.

3.2.4 Error History and Merging

In order for corrections to be made iteratively, their effect must be cumulative, requiring some internal state to record the history of corrections. The purpose of the history is to avoid hypothesizing a word that was previously marked incorrect within that same error sequence. Using our definition of the correction transformation, given the following sequence: $s_1\ t_1\ s_2$, all subsequent hypotheses for words falling between s_1 and s_2 may not contain any word in t_1. That is, with the history $s_1\ h_1\ s_2\ h_2 \ldots s_n\ h_n$, h_i contains the union of all t_i in previous correction passes. This is a somewhat restrictive view as it does not allow certain corrections which may be valid, such as permutations of the error sequence, but this is consistent with our earlier design choice.

When a new correction is submitted by the user, it must be merged with the correction history before it is sent to the recognizer for FST construction, composition, and re-recognition. When merging, one must keep track of the initial set of correct word sequences, the "islands" $S = \{ s_1 \ldots s_n \}$, rather than the correct word sequences as marked in the new correction $S' = \{ s'_1 \ldots s'_m \}$. As errors are corrected, the gaps will shrink and/or split, resulting in a different number of islands. To keep track of the original islands, we mark each word in a re-recognized gap with a text delimiter. After the recognizer sends back the new candidate, it is compared with the correction string and the new words are identified and marked. The new words are marked by going through the new recognition output and searching for each island in S. In our encoding, this is represented as "correct new! new! correct" where words marked with a "!" are the new re-recognition results. A "!" without a word indicates that all of the words in that gap were deleted. These markings are preserved through the next correction pass and incorporated into the subsequent correction string. A correction string such as "correct new! (new!) correct" indicates that the first new word is correct, but the second is not. To ensure the incorrect word formerly in that gap is not given during re-recognition, the new correction string needs to be merged with the correction history before it is converted into a constraint FST.

Using the correction delimiters, we go through the new correction, segmenting it into islands and gaps (rather than segmenting based on the correction groupings which are based on the new islands in S'). The history is then segmented into islands and gaps using the correction groupings. A sample segmentation can be seen in Figure 3-3. We then align the islands and gaps by iterating through each segment in the new correction and matching it with one or more segments in the history.

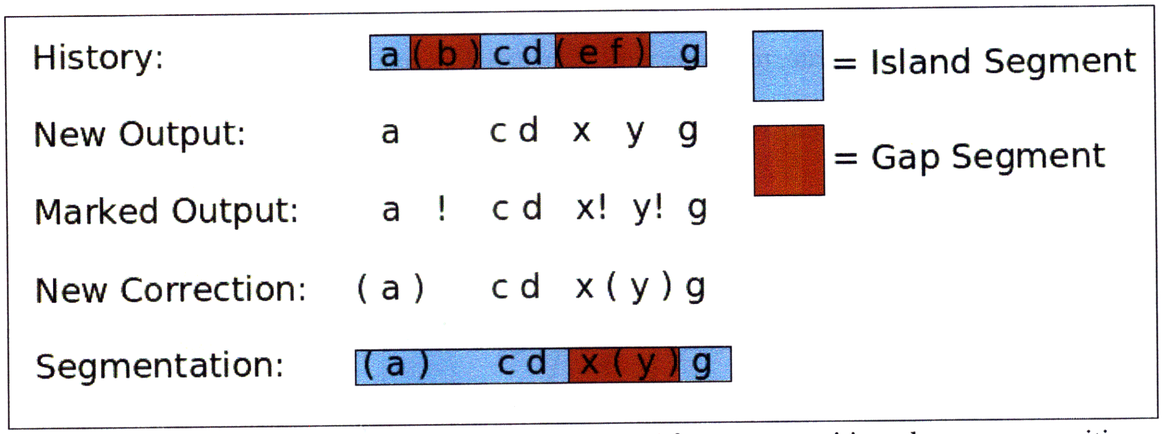


Figure 3-3: The marking and segmentation process: After re-recognition, the new recognition results in gap regions are marked, using the history to identify previous gaps. The new output is sent to the user who then makes a new set of corrections. Finally, the new corrections are segmented using the history markings, with marked words corresponding to gaps, and unmarked words corresponding to islands.

There are five possible effects to the island-sequence that can occur during re-recognition: there can be no change at all (no effect), a gap is deleted from the start (initial segment type is different, one less segment), a gap is deleted from the middle (two islands are merged into one, two fewer segments), a gap is deleted from the end (final segment type is different, one less segment), or a combination of the above (from 2n segments to as few as 1). Alignment is performed with these effects in mind.

Once aligned, as in Figure 3-4, the gap segments are checked for new gaps. If there is a new gap t_i that aligns with a history gap h_i, then $t_i = h_i \cup t_i$. A new gap may align with an old island, this is the case when a user may not have noticed an error earlier and decided to mark it in a later correction pass. In this case, the new gap would not be modified. After merging is completed, the updated correction string is sent to the recognizer, and re-recognition continues as normal. The history management portion of our system—marking, aligning, and merging—takes place in the controller servlet. One advantage of this is that it allows for the handling of history exceptions separately from possible recognizer exceptions. History exceptions occur when the recognizer ends in a non-final state, returning a result which does not contain all members of S, causing the marking process to fail. When this happens, the controller can choose the appropriate response to the user.

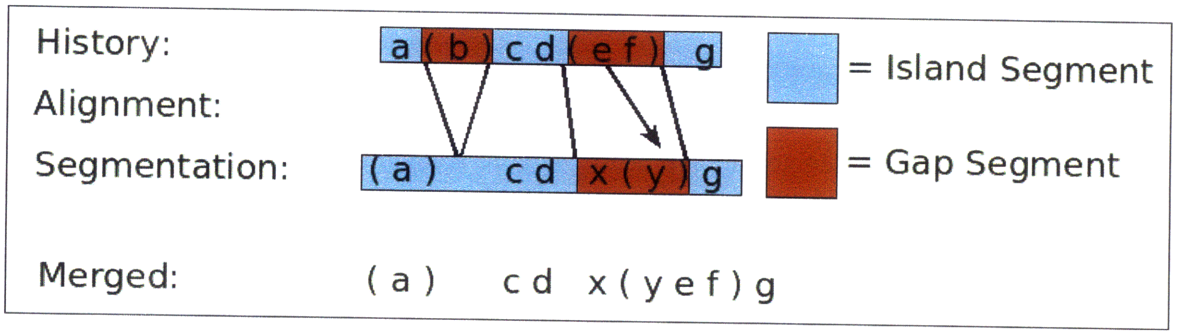


Figure 3-4: Merging of gaps after alignment between the history and new correction: After the new correction string is segmented using the island and gap segments in the history, old corrections are then merged in. In the above example, the "b" gap is eliminated, combining "a" and "c d" into a single island. In the new correction string, the user has marked "a" as incorrect, indicating they did not notice or forgot to mark it in the previous correction pass. If a word in the new string is marked as incorrect and it is in a gap segment, then the set of incorrect words from the history is merged with the new gap. In this case, the segment "x (y)" is aligned with "(e f)" so "e f" is merged with "(y)" resulting in "x (y e f)."

Chapter 4

User Interface

As the portion of our system which is in direct contact with the user, our goals of simplicity and efficiency drove the design of our interface. To make a simplified user experience, user input and output must be simple. On the input side, some correction interfaces have large command sets in which the user uses some combination of gestures, writing, typing, and clicking. Each time a user must think which command they need to use and how to perform it, or switch from one modality to another, it takes a significant amount of time. In terms of output, many correction interfaces inundate the user with recognition data. N-best lists are especially bad as they force the user to read through many long sentences that seem unordered from the perspective of the user, who has no notion of acoustic and language model scores. Not only can the amount of output on the screen slow down users, but the number of different outputs can also be a factor. Each time a user switches focus from one part of the interface to another, it again takes a significant amount of time. With simplicity in mind, we focused on a set of controls and output which would allow the user to perform their corrections with a minimum number of actions.

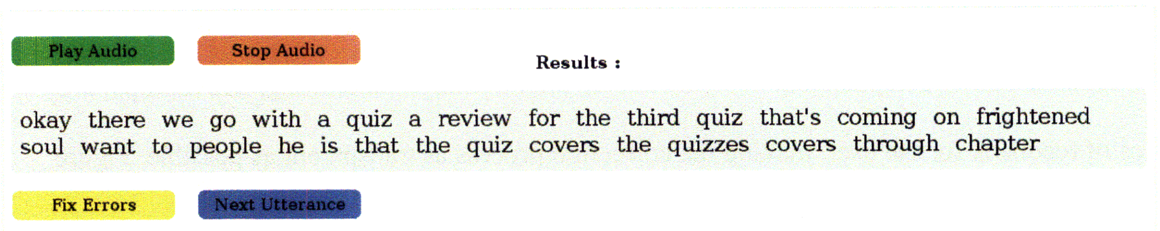
Our interface, designed using a combination of JSP, HTML, Javascript, and Flash, was built with platform independence in mind, and web technologies are a natural fit for this. The command language of our interface is extremely simple, and relies on a single modality—mouse selection and clicking. This makes it universal across multiple platforms and devices. The mouse actions can be easily translated to the selection modality of choice on the target platform such as touch or stylus.

The servlet and web interface are currently configured to work with the second usage scenario, correcting prerecorded lectures. The instructions to use it are as follows:

> "After beginning a lecture, results will be shown one utterance (approximately 10-20 seconds of speech) at a time.
> After an utterance is recognized, controls to listen to the audio and the recognition results will be displayed below.
> If you cannot hear the audio when you click the Play Audio button, try closing other audio applications and restarting your browser.
> Click on a word to mark it as incorrect, or click between two words to indicate that a word should be inserted.
> You can also click and drag from one word to another to mark multiple words at once.
>
> When you have marked corrections, click 'Fix Errors' to allow the recognizer to reprocess the utterance. When you are done correcting, click 'Next Utterance' to advance to the next clip."

From there, users can begin with the first utterance in a lecture or jump to a specific utterance number if they want to resume correcting a lecture from where they left off. The utterances time boundaries are determined through an initial segmentation step, based on silence durations. Since the interface works on a per-utterance level, as opposed to full transcript editing tools like [31, 32], it could be easily reconfigured to work in a query or command driven speech interface system as in the first usage scenario.

After choosing a starting point, the first utterance is recognized and the user is presented with the correction interface. Figure 4-1 shows the UI for a sample utterance on the first correction pass before any correction marks are made. The "Play Audio" and "Stop Audio" buttons control audio playback using the SoundManager Javascript to Flash Sound API. SoundManager allowed us to easily incorporate reliable sound playback into our interface.

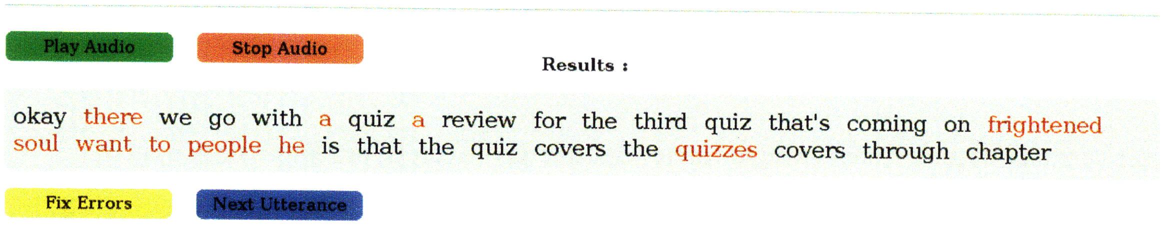


okay there we go with a quiz a review for the third quiz that's coming on frightened soul want to people he is that the quiz covers the quizzes covers through chapter

Figure 4-1: The web-based user interface for a sample utterance after initial recognition, before the user marks any words as incorrect.

Other methods of sound playback either work across limited subsets of browsers and browser versions, require a heavyweight framework like Java applets, or require deep system permissions like with XPCOM interfaces. The last method has a Javascript binding, however all code must be run locally from a Firefox extension. To assist with sound playback, there are also synchronized visual cues indicating the currently playing word. As the recording plays back, the word hypothesized to correspond with the current time is underlined, giving users a clearer notion if the recognition hypothesis is correct or not. Without this information, it can become easy to get lost in a long series of incorrect words, and be unable to find where the transcript becomes correct again. To allow for this feature, we generate a word transcript for the best candidate with corresponding start and end times for each word. The times are parsed out and passed to the web UI during initialization for use with playback.

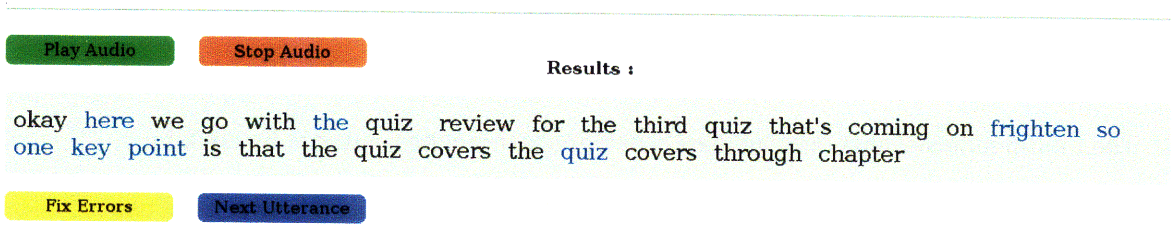
The results pane is set between the two pairs of controls, containing the recognition output and acting as the correction workspace. The user's current focus within the results pane is shown by underlining if they hovering over a word, or a tooltip if they are hovering between words. The tooltip indicates that one can mark a deletion error in the transcript by clicking between two words. When marking a deletion error, an underscore is placed between the two words and highlighted red. Clicking on a word highlights it red, indicating either a substitution or insertion error. All correction marks can be toggled by clicking them again, bringing them back to their default color. In addition, clicking and dragging allows one to specify multiple errors at once. From a usability perspective, the visual effects (highlighting, underlining, and tooltips) are sources of feedback for the user, making the correction process as transparent as possible. Figure 4-2 shows the interface after a series of corrections.



okay there we go with a quiz a review for the third quiz that's coming on frightened soul want to people he is that the quiz covers the quizzes covers through chapter

Figure 4-2: The user interface after the recognition output has been marked for correction.

After the user is satisfied with their set of corrections marks, they may click the "Fix Errors" button to begin the correction process. The correction marks are encoded to our correction string format and sent back to the web server where history merging is performed, if necessary, and the error correction process proceeds as previously mentioned. After the new words are marked with delimiters, the re-recognition 1-best is sent back to the web UI. The delimiters are translated into visual cues, highlighting the new words blue, allowing the user to quickly jump to what changed, marking any words which are still incorrect. Figure 4-3 shows the interface after one correction pass.



okay here we go with the quiz review for the third quiz that's coming on frighten so one key point is that the quiz covers the quiz covers through chapter

Figure 4-3: The user interface after corrections are marked, sent to the recognizer for re-recognition, and a new 1-best is returned to the user.

Chapter 5

Evaluation

To evaluate our error correction method and compare it against other correction techniques, we need to determine its maximum efficacy through synthetic experiments. To achieve this, we conducted a number of oracle experiments in which correction strings were generated using the reference transcript. In our oracle experiments, we tuned the gap transition penalty parameter, examined the durational and word length statistics of gaps, determined the reduction in WER, SER, and types of errors corrected, compared performance against composing the constraint FST with per-utterance word graphs, and examined applications of our technique for speaker adaptation.

To estimate potential user performance, we use predictive evaluation to compare our approach against other correction techniques. We will use the Keystroke-Level Model (KLM) to generate action sequences and times for each model [34]. KLM allows a interface designer to predict a user's efficiency by breaking down complex operations into a series of primitive operations. While using predictive models cannot gauge attributes like cognitive load, subjective satisfaction, or how easy the interface is to learn, it serves as a way to estimate the performance of an experienced user. We conclude with a brief user test, to gauge user timings and accuracy.

5.1 Experimental Setup

For our oracle experiments, we worked with a set of 7 lectures, representing approximately

30,000 words and 3 hours of audio. Two lectures were in the recognizer's training set, taken from an MIT course on linear algebra, containing approximately 5000 words. The remaining five consisted of two lectures from a seminar series and three lectures from an MIT speech recognition course. The test set lectures each had different speakers, while the two lectures from the training set had the same speaker. Word error rates ranged from 22.5% to 49.8% and sentence error rates ranged from 69% to 100%.

For the restaurant domain, we worked with a test corpus mentioned earlier, generated during user testing. We configured our recognizer for restaurant guide queries containing dynamic classes. This recognizer has a smaller static vocabulary (and a correspondingly smaller language model) than the lecture domain, roughly 1400 words compared to 30,000. The controller servlet was modified to send dynamic updates to the recognizer. Experiments in this domain were intended to compare performances across the two usage scenarios, as well as contrast the different size vocabulary of each recognizer. Given that the language model training corpus was derived from a rule based generator, one would expect to see smaller variation in recognition output, and hence constrained re-recognition output. The chance of an unseen n-gram or OOV appearing in the user data are higher than with the lectures, potentially proving an obstacle to our correction technique.

Using the reference transcripts, we scored each recognition pass with SCLITE [35], using the SGML output to construct an island-gap construct which was then converted into a correction string. This correction string contains the optimal correction marks for the given utterance at that time. The correction string is then sent to the recognizer for constrained re-recognition. Experiments were conducted through batch recognition, performing recognition on an entire lecture, generating the full set of correction strings, then re-recognizing using the corrections. This process was repeated over a number of iterations to determine the most efficient user behavior. In addition we examined the types of errors that were eliminated and those that persisted, and tracked the histogram of gap word length through multiple passes.

5.2 Gap Transition Penalty

Penalty - Pass	WER	SUB	DEL	INS	SER	# Non-Final
Initial	35.2	24.1	3.4	7.6	100	
0.5 - 1	24.6	16.1	4.0	4.5	100	21
0.5 - 2	18.4	12.2	4.0	2.2	100	17
2.5 - 1	20.0	11.9	6.2	1.9	100	2
2.5 - 2	17.3	9.2	5.9	2.1	99.3	6
3.0 - 1	19.9	11.5	6.8	1.6	100	0
3.0 - 2	17.1	8.8	6.5	1.7	97.9	5
5.0 - 1	19.9	11.5	6.8	1.6	100	0
5.0 - 2	17.4	8.0	8.2	1.1	97.9	3

Table 5-1: Error rate results of gap transition penalty experiments over two correction passes. The leftmost column indicates the correction penalty used followed by the correction pass. Experiments were conducted using one lecture in the evaluation set with a high incident of outputs ending in non-final states. The lecture had 141 total utterances.

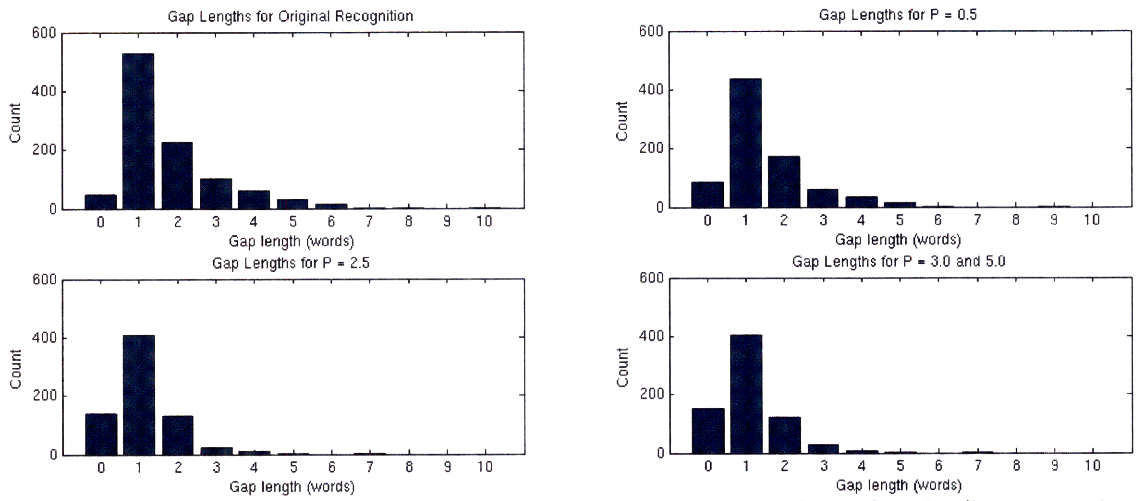
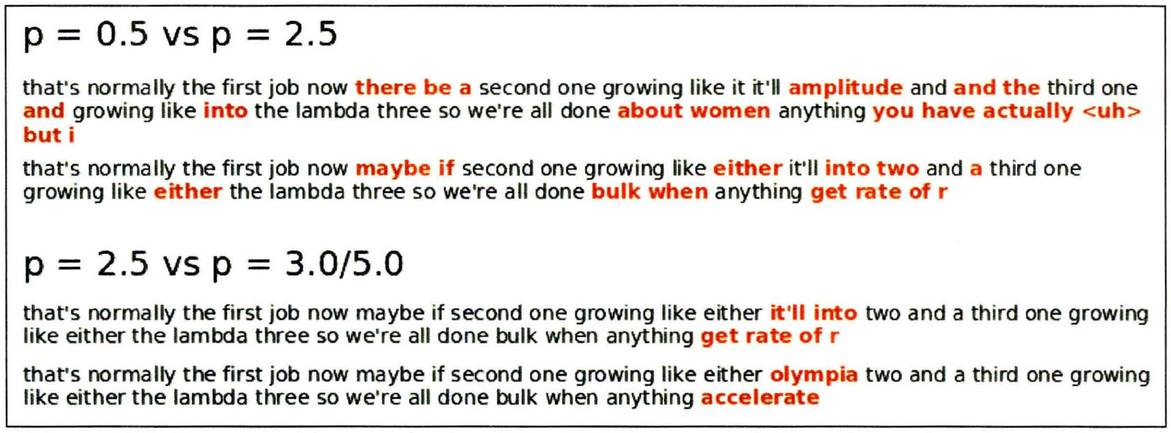


Figure 5-1: Gap word length histograms for various transition penalties after the first correction pass. The histograms track the count of gaps containing n-words for n = 1 to 10. Increasing the penalty increases the number of 0-word errors (deletions) as well as decreases the number of multi-word gaps.

The gap transition penalty determines how costly it is to self loop in the multi-arc gap state before transitioning back to a forced path island state. The penalty is applied to the log-probability language model scores. We tuned the penalty on one of the lectures that had a particularly high number of recognitions ending in non-final states. Table 5-1 shows the performance of each penalty value tested, with its associated WER (broken down by error type), SER, and number of non-final results for two correction iterations. Figure 5-1 shows the gap word length histograms for the different penalty values. We see the increase in 0-word gaps, or deletion errors, as a result of the higher penalty. Multi-word gaps are not as reduced with the low penalty value because results ending in a non-final state typically retain the original errors, as well as alter the originally correct words. In addition, replacing a multi-word gap with an equally long word sequence during re-recognition would be costly due to the penalty incurred for each word.

Analyzing the results, we see that the gap penalty is directly proportional to WER and number of deletions errors while inversely proportional to the number of non-final results and average gap word length. Figure 5-2 compares sample output for each penalty value from the same correction pass.



p = 0.5 vs p = 2.5

that's normally the first job now **there be a** second one growing like it it'll **amplitude** and **and the** third one **and** growing like **into** the lambda three so we're all done **about women** anything **you have actually <uh> but i**

that's normally the first job now **maybe if** second one growing like **either** it'll **into two** and **a** third one growing like **either** the lambda three so we're all done **bulk when** anything **get rate of r**

p = 2.5 vs p = 3.0/5.0

that's normally the first job now maybe if second one growing like either **it'll into** two and a third one growing like either the lambda three so we're all done bulk when anything **get rate of r**

that's normally the first job now maybe if second one growing like either **olympia** two and a third one growing like either the lambda three so we're all done bulk when anything **accelerate**

Figure 5-2: Re-recognition results for the same utterance and correction string given different penalties.

As expected, increasing this penalty makes it costly to remain in the gap state, resulting in shorter word sequences in gaps after re-recognition. Aside from making for less words per re-recognized gap, there are also more multisyllabic words. For this particular domain, college level courses, topic words are often multisyllabic and wind up segmented into multiple smaller words for lower penalties. After evaluating these results, we chose a gap penalty of 3.0 for the remainder of our experiments as it offered the best balance of minimizing non-final results and WER.

5.3 Lecture Domain Experiments

The following experiments on the lecture domain are intended to evaluate both the efficiency of our technique as well as characterize the types of errors that occur and the ones that are fixed.

5.3.1 Error Duration and Length Statistics

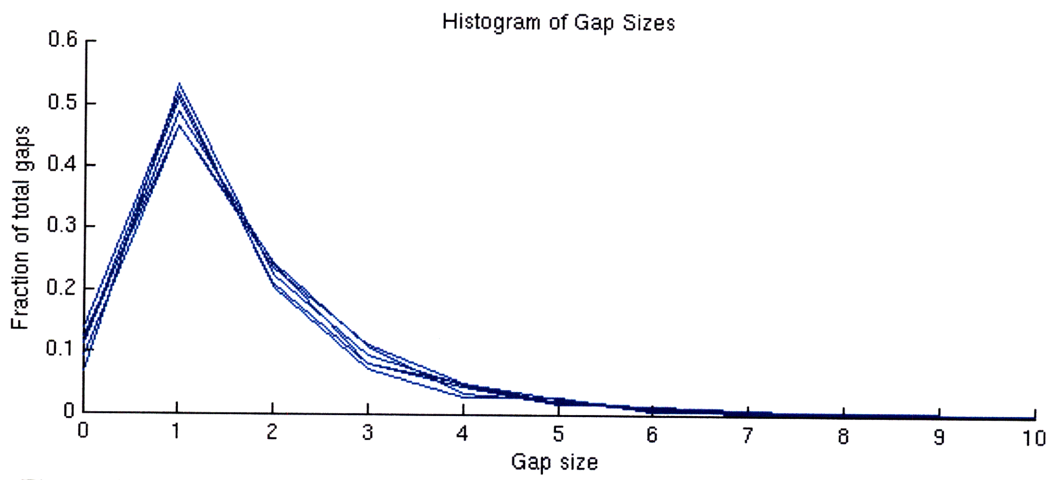


Figure 5-3: Distribution of gap word lengths after initial recognition for each lecture in the evaluation set, indicating the frequency of a given gap length.

In order to better understand the types of errors found in this domain, we measured the duration and word length statistics of the gaps in our evaluation set. Figure 5-3 shows the histogram of gap word lengths for each lecture, measured after initial recognition, but before any correction passes. The lectures share the same distribution of error lengths despite varying speakers, initial word error rates, and topic. The vast majority of errors are only a single word, ranging from 46% to 53% of the total error sequences, and are surrounded on either side by either correct words or sentence markers. One's initial intuition might be that any error correction mechanism should focus on single word error correction, like word-level alternate lists, confusion networks, or re-speaking with isolated word recognition. While single word errors are important, one needs to look at the histogram weighted for each error sequence's contribution to the word error rate. Figure 5-4 shows the histogram with each sequence weighted by its length. This data reveals that multi-word errors are very prominent, accounting for 60-70% of the total transcript errors. Error correction schemes that correct one word at a time are slower for users, and less effective, as they fail to take advantage of information about other local errors.

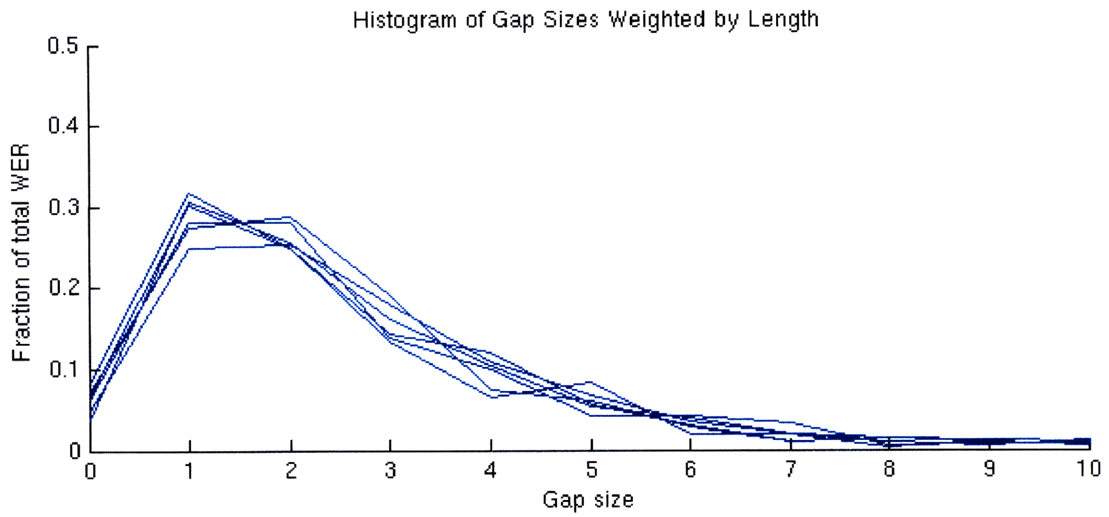


Figure 5-4: Distribution of gap word lengths after initial recognition for each lecture in the evaluation set, weighted by the number of words in the gap, indicating the contribution of a given gap length towards WER.

The duration distributions for gaps of one to five words in length is shown in Figure 5-5. Examining the distributions, one sees a second "bump" after the main probability mass, due to multisyllabic words. The distribution for five word gaps is very wide, as the number of possible syllables increases exponentially. A more informative choice if this information were to be used for constraint purposes (see Chapter 6) would be to model phone duration distributions. Given enough data one could generate diphone duration models to predict the number of words in a gap accurately.

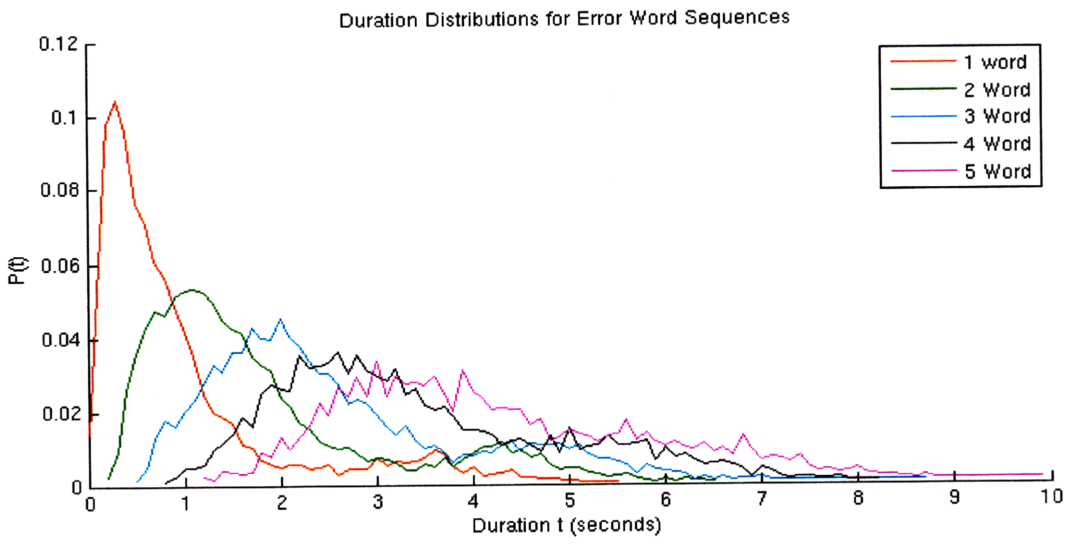


Figure 5-5: Distribution of durations (in seconds) for gaps of one to five words.

5.3.2 Word Graph Constraint Composition

Our initial implementation of composition was computationally and memory inefficient, requiring us to investigate alternative approaches. One candidate we explored was composing the error constraint FST with the word graph generated for each utterance during recognition. Word graphs contain the candidate search space at the time recognition completes. Word graph composition should perform very close to that of full re-recognition given sufficiently large word graphs. Given infinite space, one could create a word graph containing the entire possible search space and achieve the same performance as re-recognition. Real world constraints require the recognition search space to be constantly pruned as arcs are added and states are explored. The pruning is dictated by the parameters described in Chapter 2, eliminating paths based on their probability according to various size criterion and score thresholds. For our experiments with word graphs, we explored the pruning parameter space, to see what gains could be achieved as word graph size increased.

vp / vpn / mpps	Input/output symbols	States	Arcs
24 / 2000 / 2000	369.4	579.7	3573.4
24 / 2000 / 10000	407.1	1037.8	13971
24 / 2000 / 20000	410.1	1020.7	14331.6
24 / 5000 / 20000	409.2	930.4	14050.5
24 / 20000 / 20000	410.7	930.3	14110.7
40 / 20000 / 40000	410.4	858	13009.8
50 / 20000 / 40000	410.4	858	13009.8

Table 5-2: The size of word graphs for increasing values of vprune (vp), vprunenodes (vpn), and maxpopspersec (mpps) for a random set of 10 utterances in a lecture.

Table 5-2 shows the size of the word graphs as vprune, vprunenodes, and maxpopspersec are increased. Interestingly, we find that after optimizing the word graphs, their sizes level off despite increasing the pruning parameters. The reason for this is that before optimization, a given word might appear more than once at slightly shifted times. Each instance of that word represents different search candidates as there will be a different probability associated with its alignment with the acoustic information, but the paths leading from it may still result in the same output word sequences. Optimization collapses these identical words into a single path. As a result, increasing the pruning thresholds tends to increase the number of paths for the same subset of words, making the resulting optimized word graph roughly the same size as with smaller thresholds.

To compute a correction pass, the corrected results are generated by composing each utterance's word graph with its oracle constraint FST. Outgoing arcs in gap states contain all words in the vocabulary of the word graph instead of the recognizer's full vocabulary. Table 5-3 show the results of first pass composition with small, medium, and large word graphs on a subset of our evaluation set. We see modest improvements in the initial correction pass, approximately 10% absolute reduction in WER. This technique is still a post-recognition step, and does not use information obtained through user correction in the recognition process. As such, it will be limited to the search space at the time of recognition, like confusion networks and alternates list are, making it impossible to retrieve pruned word sequences without moving to free form correction methods like keyboard-based editors. Since some states may only have one outgoing word arc, composing the constraint often gives an empty FST. Given these limitations and our improvements to runtime with full re-recognition, word graph composition is an inferior technique. In situations where re-recognition is not possible (on a mobile device using a server-based recognition architecture) then improvements to this technique might be of interest.

	WER	SUB	DEL	INS	SER
Original	35.2	24.1	3.4	7.6	100
First Pass small	25.2	17.2	4.3	3.7	100
First Pass med	24.6	16.8	4.4	3.4	100
First Pass large	28.2	18.7	5.7	3.8	100

Table 5-3: First pass composition with small (vprune = 24, vprunenodes = 2000, maxpopspersec = 2000), medium (vprune = 24, vprunenodes = 2000, maxpopspersec = 20000), and large (vprune = 50, vprunenodes = 20000, maxpopspersec = 40000) word graphs.

5.3.3 Iterative Error Reduction

An important aspect of our correction technique is that, barring any out of vocabulary words, we are guaranteed to converge on the correct hypothesis in a finite number of iterations. Figure 5-6 shows the convergence of iterative corrections for one lecture. WER levels off at around 5%, due to OOV words (1.6% WER contribution) and the issue of not forcing non-null replacements for deletions errors (3.3% WER contribution). While a sole user may not want to mark a misrecognized word more than two or three times, if our correction technique were to be incorporated into a collaborative interface for lecture viewing and editing, a class's joint efforts could quickly produce a near-perfect transcript. The increasing demand for access to multimedia resources in education make this scenario especially pertinent and an ideal fit for our correction system.

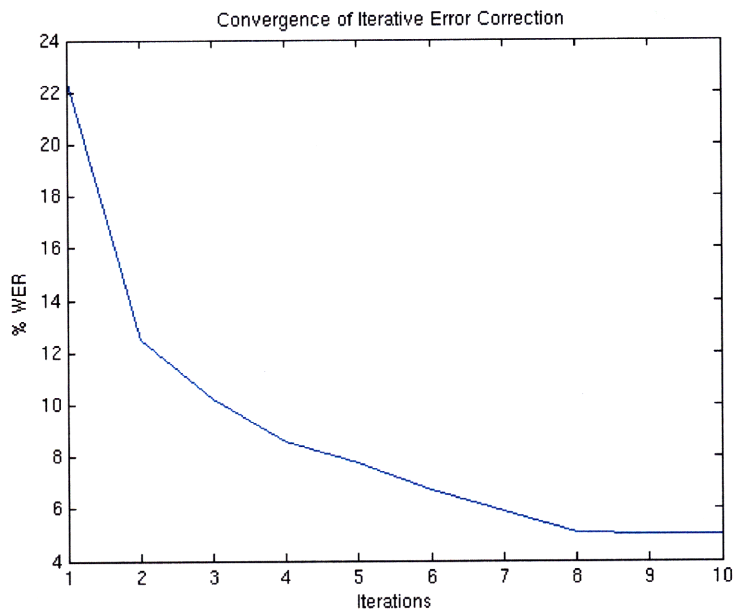


Figure 5-6: Demonstration of the convergence of error correction over a large number of iterations. Evaluated on a lecture in the test set which contained OOV errors, so 0% WER cannot be achieved.

	SUB	DEL	INS	WER	Rel. ≤WER
Original	21.0	4.6	5.7	31.2	
Pass 1	10.0	8.0	1.0	19.1	38.78
Pass 2	7.4	7.5	0.8	15.8	17.28
Pass 3	5.9	7.1	0.7	13.7	13.29
Pass 4	4.9	7.1	0.6	12.6	8.03

Table 5-4: Substitution/Deletion/Insertion statistics, WER, and relative WER reductions on a subset of our evaluation corpus to determine the maximum number of effective iterations.

To evaluate the effect of iterative correction passes, we first determined the optimal number of correction passes using a subset of our evaluation set. Table 5-4 shows the error statistics and results for each correction pass, followed by the relative word error rate reduction in the right-hand column. The effect of subsequent passes diminishes quickly, however we still see greater than a 1% absolute WER improvement through the third pass, so in our full evaluation and speaker adaptation experiments, we will only explore up to three iterations. In practice, the most efficient user strategy is probably one or two correction passes, followed up with an alternate method on the few remaining errors.

Lecture	Rel. ≤WER, first pass	Rel. ≤WER, second pass	Rel. ≤WER, third pass
1	43.2	17.0	8.9
2	53.6	20.7	12.2
3	32.5	20.1	9.8
4	39.2	13.9	19.8
5	37.0	17.6	15.0
6	44.4	22.6	18.6
7	37.0	22.9	13.7

Table 5-5: Relative word error rate reductions after each iteration. For the second and third passes, the WER reduction is calculated relative to the error rate of the previous pass.

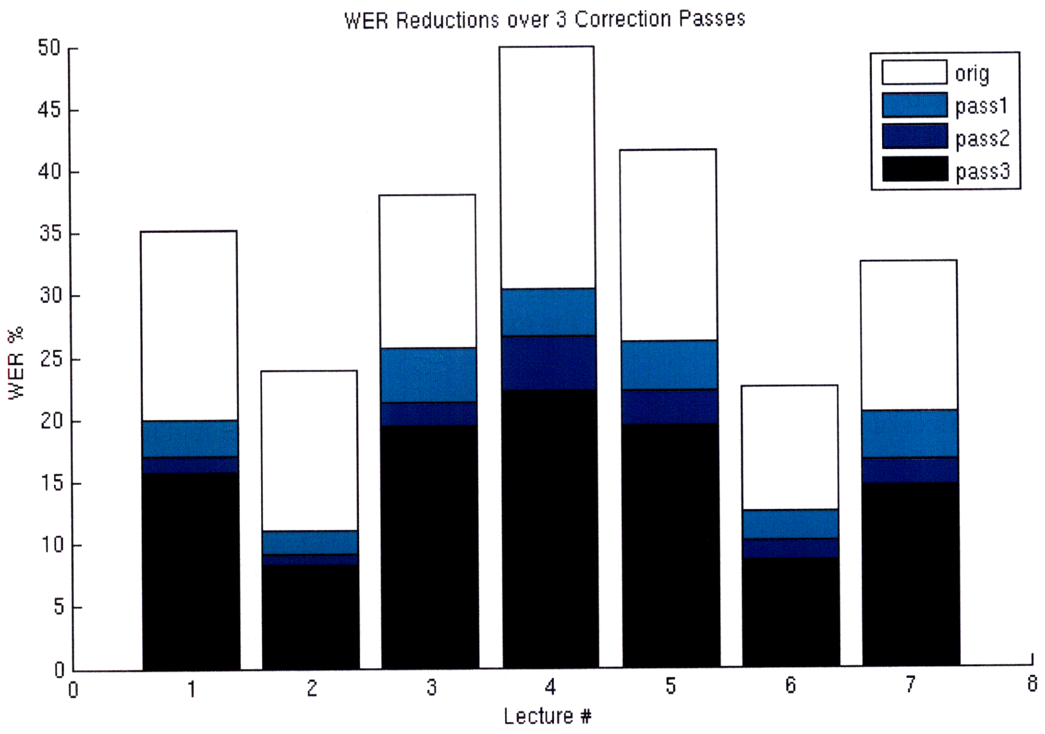


Figure 5-7: Word error rates for the lectures in the evaluation set after each correction pass. Large reductions in WER are obtained in the first pass, followed by modest reductions in later passes.

Figure 5-7 shows the word error rates for the lectures in our evaluation set after each correction pass. Table 5-5 summarizes relative WER improvements between each iteration. The first correction pass fixes approximately 30-50% of all errors, and the second pass reduces the remaining errors further, usually leaving less than half of the original errors. Figure 5-8 shows the progression of gap length histograms through each iteration. As we saw in our analysis of gap lengths, initially, most gaps are only a single word. The first correction pass reduces 1-word gaps by 45%, but increases 0-word gaps (deletion errors) by 276%. Overall, 0-word and 1-word gaps decrease by 17% and multi-word gaps decrease by 43%. The increase in deletions is due to our penalty inhibiting gap self-traversal, causing many 1-word gaps to simply be deleted. Currently, our constrained re-recognition process does not forbid epsilon transitions (null output) when a deletion error is marked, so some 0-word gaps are never corrected. Since an epsilon arc was the best path in initial recognition, the only way to get a word in re-recognition is if words are corrected close enough to the gap to modify its LM weight.

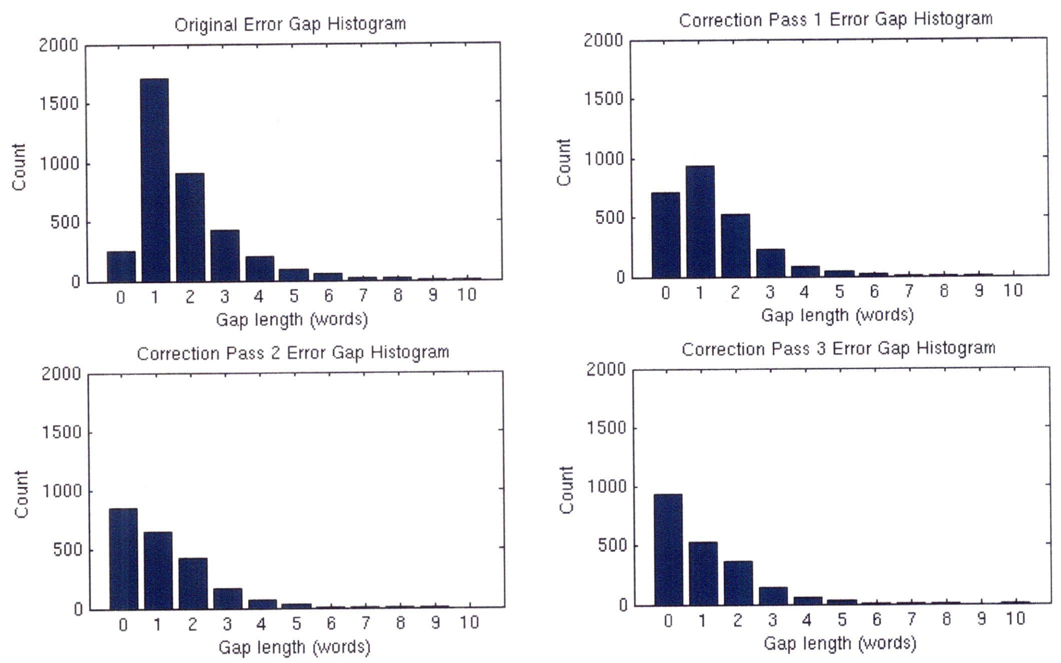


Figure 5-8: Aggregated gap length histograms for each correction pass. Each correction pass reduces the multi-word and single word errors, and increases the deletions errors due to the gap penalty.

We divided the data by lecture to see if the effectiveness of our technique was correlated with initial WER. Table 5-6 shows the correlations between initial WER and the relative WER reduction for each correction pass. The amount corrected in the first and second pass has a moderate negative correlation with the initial WER, indicating that lectures with high initial WERs will be tougher to correct, as contributing factors like low audio levels, reverb, and out of vocabulary words are not correctable with our method. There is no correlation between the relative WER reduction and the WER between the second and third pass indicating that, because the amount of errors corrected is close to random, re-recognition is picking from a set of roughly equiprobable candidates which are not affected by the language model constraints introduced by the correction FST. These findings support our prior intuition that performing two correction passes is the optimal strategy. In summary, we achieved consistent results across all lectures—about half of the errors are corrected after the first two passes—demonstrating the potential of this method in the lecture domain.

	Init. WER and Relative WER reduction in Pass 1	Pass1 WER and Relative WER reduction in Pass 2	Pass2 WER and Relative WER reduction in Pass 3
Corr. Coeff.	-.631	-.690	.161

Table 5-6: Correlations between initial WER and ≤WER.

5.4 Restaurant Domain Experiments

The experiments in the restaurant domain demonstrate our correction method on another test set as well as highlight the differences between the two domains. The restaurant domain is characterized by its dynamic classes and smaller grammar compared to the lecture domain. With more than half of the vocabulary being class nouns, and the static vocabulary (from which carrier phrases are composed) only 5% of the size of the lecture domain's grammar, variation in sentence structure is very low. Another contributing factor is that much of the data was collected from a kiosk-based system running the recognizer where users could interact with it. User behavior typically begins with a target query followed by a number of repetitions or rephrases when misrecognition occurs. Given this behavior, unrecoverable errors, like out of vocabulary words, poor audio, and unseen n-grams, are amplified in the test corpus.

5.4.1 Gap Transition Penalty

We re-tuned the penalty parameter as optimal values should be domain dependent. Intuitively, avoiding results in non-final states requires balancing the penalty against average utterance duration and phone length, and the average LM score of each word in the vocabulary. In addition to preventing non-final results, the penalty also shortens the average word length of a gap's re-recognition hypothesis, favoring words which contain more phones. Each transition made within a gap state incurs the penalty, thus a gap sequence will be replaced with m words instead of n words, where $m < n$, if $(n - m) * p > LM(m) - LM(n)$. Table 5-7 shows the results of our tuning experiment, giving an optimal penalty of 1. Unlike the lecture domain, increasing the penalty results in strictly poorer performance, and with relatively little change in the number of non-final results.

Penalty	WER	SUB	DEL	INS	SER	# Non-Final
Base	29.6	16.9	9.4	3.3	66.0	
1	25.2	12.2	11.9	1.1	59.8	44
3	25.6	10.2	14.7	0.6	60.6	37
7	26.7	7.9	18.4	0.4	61.1	40

Table 5-7: Error results for one correction pass in the restaurant domain for different penalty values. Evaluated on 1161 utterances, approximately half of the test corpus.

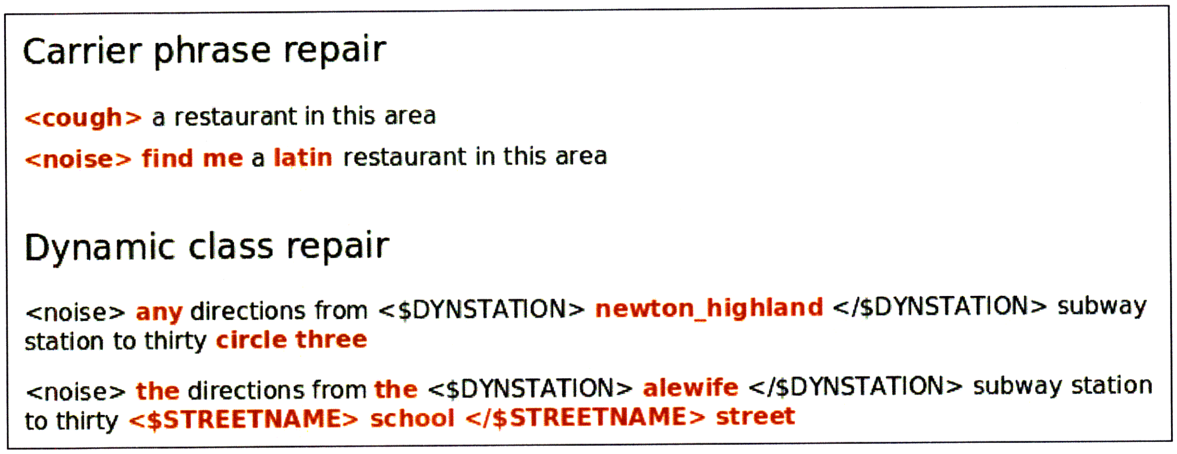
5.4.2 Iterative Error Reduction

In our full iterative correction experiments, we found modest improvements to WER, Table 5-8 summarize the results. Some of the recurring errors can be attributed the unrecoverable errors mentioned previously.

	WER	SUB	DEL	INS	Rel. ≤WER
Original	29.6	16.9	9.4	3.3	
Pass 1	25.2	12.2	11.9	1.1	14.9
Pass 2	24.4	11.2	12.0	1.2	3.2

Table 5-8: Error results and relative WER reductions for two correction passes. Evaluated on 1161 utterances, approximately half of the test corpus.

While gains were not as large as with the lecture domain, constrained re-recognition still proved effective with a number of utterances. Figure 5-9 shows some sample sentences before and after re-recognition. We see effective repairs made to both the carrier phrase portion of the query and missing instances of dynamic classes. Our correction method could be of use as an initial correction mechanism, however, additional improvements would have to be made before it could be used as a primary correction method; for ideas on extensions of our technique, see Chapter 6.



Carrier phrase repair

<cough> a restaurant in this area

<noise> find me a **latin** restaurant in this area

Dynamic class repair

<noise> **any** directions from <$DYNSTATION> **newton_highland** </$DYNSTATION> subway station to thirty **circle three**

<noise> **the** directions from **the** <$DYNSTATION> **alewife** </$DYNSTATION> subway station to thirty **<$STREETNAME> school </$STREETNAME> street**

Figure 5-9: Sample re-recognition results from the iterative re-recognition experiments in the restaurant domain.

5.5 Speaker Adaptation

In our interface, the user explicitly marks the incorrect words, providing the system with free supervised data. A natural application of this information is language model or acoustic model adaptation, further boosting re-recognition performance beyond that provided by the error constraint FST. The SUMMIT recognizer uses speaker independent acoustic models for recognition, trained on hundreds of hours of lecture audio from variety of male and female speakers, so interpolating speaker dependent models should improve recognition performance.

We tested speaker adaptation on three lectures, each with different speakers, building speaker independent acoustic models for each lecture. The models were trained off-line in batch mode after each recognition pass, using the reference transcripts. Adaptation was performed using the method described in [3], with ≤set to 50. Phone models were trained through aligning the hypothesized phone sequence with the hypothesized word boundaries, discarding data that corresponded to incorrect words.

Table 5-9 shows the performance of our iterative correction process with and without speaker adaptation performed between each iteration. Our results show that off-line batch adaptation can be used effectively to boost our correction technique. The applications of this range from performing adaptation after each lecture transcript is corrected in a lecture series, increasing recognition performance in later lectures, to building a speaker dependent model for a user of a query interface, doing batch adaptation after an interval of corrections. Additional research could be done in the area of batch adaptation, with Chapter 6 exploring some of those ideas.

	WER	SUB	DEL	INS	SER
Original	33.7	22.9	5.9	4.9	75.5
Pass 1	21.4	11.0	9.7	0.7	60.5
Pass 1 with SA	19.8	9.5	9.3	1.0	57.7
Pass 2	17.8	8.1	9.0	0.6	54.8
Pass 2 with SA	16.3	6.6	8.9	0.8	52.3
Pass 3	16.0	6.6	8.9	0.5	51.7
Pass 3 with SA	14.5	5.2	8.7	0.6	50.2

Table 5-9: Correction performance for three iterations with and without batch-mode speaker adaptation after each pass.

5.6 Predictive Evaluation

Predictive evaluation allows for the objective measurement of efficiency given an expert user of the current task. The particular predictive evaluation model we will be using, the Keystroke-Level Model, breaks down any task into a sequence of elementary operations—pressing a key or clicking a button, pointing to a target with the mouse, moving ones hands from the mouse to the keyboard and vice versa, drawing a straight-line segment, and mental preparation (K, P, H, D, M, respectively) [34]. We will modify this model slightly to account for static click speed and variable typing rate. This average time required for each operator is shown in Table 5-10.

Operator	Description	Time
K	Typing a key on the keyboard	.08s (best), .28s (average), 1.2s (worst)
B	Clicking a mouse button	.1s
P	Pointing with the mouse	1.1s
D	Drawing a line	1.1s
H	Homing from the mouse to keyboard	.36s
M	Mental operations	1.35s

Table 5-10: Elemental operations used in KLM analysis and their associated durations for the average person.

The times assigned to each operation were obtained from years of user testing, representing the speed of the average human [34]. While pointing and drawing a line are dependent on distance, we will make the simplifying assumption of uniformity, as calculating a more detailed distance model would rely on a large number of factors like monitor size and resolution, font size, average word length, etc. For our evaluation we will also ignore computational costs of re-recognition as it varies based on the domain, utterance length, number of errors, length of the error gaps, hardware used, recognizer architecture, and network latency. In our experiments we found history operations, constraint composition, and re-recognition typically took on the order of seconds, with smaller recognizers, such as that used in the restaurant domain, taking only a fraction of a second. For an analysis of computation time, see Table 5-11 which breaks down the durations of each operation performed in re-recognition for a long utterance (~20 seconds) and a short utterance (~5 seconds). Re-recognition was performed using a quad core computer with 4GB of RAM. We see that total computation time is highly influenced by the number of gaps in the correction. Increasing the number of words in each gap reduces constraint time, since all words in a single gap are converted into a single state in the constraint FST. Recognition time goes up as gap length increases due to the greater number of choices in each gap state.

	Gaps	Initial Rec	Construct	Trim	Re-Rec	N-Best	**Total**	States	Arcs
Long	1 x 1w	12.33	1.97	1.14	0.83	0.61	**4.55**	270525	337070
	1 x 3w		1.84	1.06	0.94	1.13	**4.97**	263811	354527
	1 x 5w		1.78	1.06	1.07	1.75	**5.66**	263184	367925
	3 x 1w		4.73	3.29	2.55	0.87	**11.44**	806639	913531
	3 x 3w		4.55	3.24	1.95	1.41	**11.15**	799681	730904
	5 x 1w		7.63	5.46	1.91	0.94	**15.93**	1344182	1057468
Short	1 x 1w	2.89	1.51	1.1	0.16	0.11	**2.87**	269222	131002
	1 x 3w		1.43	1.06	0.24	0.37	**3.10**	265787	186566
	1 x 5w		1.39	1.06	0.31	0.48	**3.23**	265638	223390
	3 x 1w		4.27	3.27	0.44	0.28	**8.25**	806825	401482
	3 x 3w		3.75	3.36	0.80	0.69	**8.59**	807780	579614
	5 x 1w		6.70	5.51	0.73	0.41	**13.35**	1346651	626383

Table 5-11: Computation time (in seconds) for each operation involved in constraint construction (Construct and Trim) and re-recognition (Re-Rec and N-Best) for a 20 second and a 5 second utterance in the lecture domain. Gaps of the form n x mw consist of n separate gaps, each w words in length.

While this model does not account for user mistakes while performing the task, users can undo correction marks through clicking on the word again, making recovery very simple. Given this, true user's results should not deviate significantly from those presented in our analysis. Table 5-12 shows the time required to identify and mark all of the errors in recognition output of n words with m sequences of errors. Note that the operations are grouped by description, and may not necessarily follow that order in time (ex: M P B M P D = M*2 P*2 B D). At the start of the task, we assume the user has either listened to the audio recording or remembers what was said.

Operator	Description	Time
M * n	Read a word and decide if it is correct	1.35s * n
P * m	Move the mouse to the incorrect word	1.1s * m
B * m * .7	Click the error (0-word or 1-word gap)	.07s * m
D * m * .3	Drag select the error sequence (multi-word gap)	.33s * m
P	Move the mouse to the Fix Errors button	1.1s
B	Click the button	.1s
Total		1.2s + 1.35s * n + 1.5s * m

Table 5-12: Decomposition of our error correction method using KLM

In the next correction pass, there are ≤ m words to read and m * (1 - p) errors to mark where p is the fraction of errors that were corrected. (If we did not highlight the words that were modified through re-recognition, the user would have to read all n words again).

To compare our approach with a competing correction method, we will analyze the time required to correct an utterance using a keyboard, currently the fastest modality (although one unavailable on many platforms). Table 5-13 breaks down keyboard correction using the KLM model, assuming the the interface is structured such that users must click to move the editing cursor before typing and each error sequence is c characters long on average.

Operator	Description	Time
M * n	Read a word and decide if it is correct	1.35s * n
P * m	Move the mouse to the incorrect word	1.1s * m
B * m	Click to move the cursor before the error	.1s * m
M * m	Recall how to spell and type the correct sequence	1.35s * m
H * m	Move hands from the mouse to the keyboard	.36s * m
K * c * m	Retype	.28s * m * c
Total		1.35s * n + (2.91s + .28s * c) * m

Table 5-13: Decomposition of keyboard corrections using KLM

One important difference between the two techniques is that in our error correction interface, the user does not need to recall how to create the correct output, he or she must only recognize that the word is incorrect. All correction interfaces share the initial step of reading each word and determining if it is correct, making the cost to use our method only clicking to select the words and then clicking the Fix Errors button. The average character length of an error sequence in the lecture domain was around 8 letters, making the total time for keyboard corrections (neglecting initial error detection), 5.15s * m compared to 1.2s + 1.5s * m for our method. Given our correction rates of 30-50% in the first pass, our technique should be superior as an initial correction method. For a second correction pass and a conservative estimate of only 30% corrections, the total time for our method becomes 1.2s + (1.35s + 1.5s + 1.5s * .3) * m = 1.2s + 3.3s * m, making it the more effective method if 60% or more errors can be corrected in both passes, or if the user is a particularly slow typer. For ASR systems where a full keyboard is not available, like a kiosk or mobile device, our correction method is clearly the most efficient.

5.7 User Test

Given time limitations, we were unable to conduct a full-scale user test, however we obtained some usage data for a small set of users (n = 7). Users were drawn from the population of lab researchers and students, and all possessed a high level of computer proficiency. Before testing began, users were read the instructions to the interface (see Chapter 4) and were told to correct consecutive utterances from a single lecture in the lecture test set. After users were given 10 utterances to practice and familiarize themselves with the correction interface, they were instructed to correct 50 utterances, containing 126 word errors. The web interface was modified to log user actions, recording the number of times the user played the audio, their correction string, and the current utterance id, along with timestamps for each action.

Table 5-14 shows the results of our user test, demonstrating the efficiency of our correction system. With an average correction time of 10.09 seconds, users required less than twice real time to mark errors. Users demonstrated an extremely high precision rate, indicating it was clear to users when a word was correct. Lower recall rates were a result of either uncertainty or leniency with the recognition output. Users tended to ignore stemming errors ("wait" instead of "waited" or "things" instead of "thing"). Ambiguous errors, such as "um" instead of "uh" or "a," as well as non-speech tags like <noise> and <laugh> were difficult for users to identify.

Subjective user impressions were favorable, indicating that the interface was easy to use with a low cognitive overhead. Some users encountered difficulty with longer utterances, as the length of the audio would exceed their auditory memory, requiring multiple playings. UI improvements we could look into to facilitate this include pausing the audio when a correction is made, random-access playback, or variable rate playback (see Chapter 6).

	Audio Clicks	Total Time	Average Time	Recall	Precision
User 1	1.02	328	6.56	0.84	0.98
User 2	2.00	653	13.06	0.78	0.99
User 3	1.10	517	10.34	0.70	0.96
User 4	1.30	519	10.38	0.74	0.98
User 5	1.22	574	11.48	0.82	0.94
User 6	1.48	486	9.72	0.67	0.97
User 7	1.78	762	15.24	0.77	0.98
Mean	1.41	548.43	10.97	0.76	0.97
Standard Dev	0.36	136.35	2.73	0.06	0.02

Table 5-14: User test statistics for each user, the mean, and standard deviation for the number of times the user listened to the audio for each utterance, their total and average time to complete the task, and their recall and precision in identifying errors.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we sought to create an easy to use, widely applicable error correction system that would boost user output rates. As ASR systems move from the research space to the consumer environment, the need for efficient, user-friendly error correction is apparent. Effective interactions with speech systems of any domain rely both on accurate recognition and the means to recover when it is inevitably wrong. Through our approach of constrained re-recognition, we have developed a correction system which works effectively alone, correcting more than half of the errors in a minimal amount of actions, and can seamlessly integrate with alternate correction techniques when used as a preliminary correction mechanism after the user identifies recognition mistakes. This work is a significant first step towards efficient error correction, demonstrating the effectiveness of our design, but there is still significant improvements and extensions that could be made. Our error correction framework can be extended in a number of directions that we will explore below, directly addressing usability and system performance, adding additional constraint to the re-recognition process, and investigating further uses of data generated through user annotated corrections.

6.2 Future Work

Our error correction technique shows great promise over existing error correction methods, and there are a number of improvements which could enhance the usability and performance of our framework. As the goal of our error correction system has been to combine constraints provided by the recognizer and the user, future extensions target both user and system performance.

In our user interface, changes to enhance user control and further increase user-generated constraints would assist user-based metrics like time per correction step and overall output rate, as well as user satisfaction. These changes include better synchronizing audio playback with the recognition output, allowing for variable rate playback, visualization of word-based confidence, and efficient supplemental correction methods.

Changes to our system back-end to enhance recognition time and the accuracy of results returned would again reduce the time required to repair recognition results. Interpolating a larger language model vocabulary during re-recognition, adding additional phonetic or timing constraints to the correction FST, and fixes to prevent results ending in non-final states are all possible areas of future research.

6.2.1 Full User Study

Due to time limitations we were unable to conduct a large user study to determine the behavioral patterns and characteristics of real users working with our correction interface in comparison with other techniques. To more rigorously validate our correction scheme and interface, we would like to measure a user's corrected word output, and obtain more subjective impressions on its usability and user-friendliness. Other statistics of interest include the amount of time it takes for users to find errors, as well as their recall and precision in error detection. User interface design could be further refined with user feedback and data on correction strategies. With our goal of platform independence, we would like to conduct tests on a range of devices to see how performance is affected by small screen size or limited correction modalities.

6.2.2 Supplemental Correction Methods

While user-constrained re-recognition can correct all errors, it is not guaranteed to if the correct word is out of the recognizer's vocabulary, or if its probability is so low that it would take a large number of correction steps to reach it. Thus, there needs to be effective alternate methods when re-recognition fails to correct the utterance in a reasonable amount of time. One possibility would be to present a number of correction strategies at any time, and allow users to choose the most convenient scheme. If the modality is available, the interface could allow for keyboard or touch-pad input, display an alternatives list below the top hypothesis, and display a confusion network below each word.

One problem with this approach is that users often favor a correction strategy even if it is less efficient as some studies have shown [8, 19]. Thus, it may be wise to only allow for constrained re-recognition for the first two iterations where the technique is most efficient. Another issue is that of cognitive overload—one driving motivation behind our work was to make it easy for users to work with. Displaying multiple correction schemes adds another step in the critical path of the correction process.

6.2.3 Guided Error Detection

Automated error detection techniques are not accurate enough to be combined with our correction technique. False positives in detection would become even more harmful in later correction steps as the correction rate drops, while falsely detected errors will always be modified from the correct word. This confounding effect of deciding if there is a recognition error or a detection error would increase a user's cognitive load, lowering correction throughput.

Despite the dangers of error detection, an error classifier could be used to guide user error discovery. If we are displaying confusion networks as an alternate correction method, we could use statistics derived from them as features for the classifier. These features include the posteriors of the most probable word, the difference of the posteriors of the most and the second most probable word, the entropy of the posteriors, and the number of alternatives within each word slot of the confusion network [29]. The ROC curve for a classifier developed by a researcher in SLS using these features can be seen in Figure 6-1.

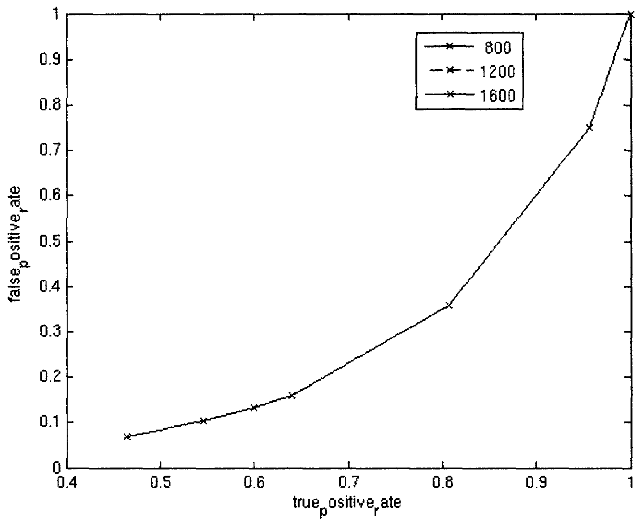


Figure 6-1: The ROC curve for a gap classifier using posterior-based features which can be used for automatic error detection. The classifier was evaluated on data in the lecture corpus.

Experiments would have to be done on various detection thresholds to find out the levels of precision and recall that are most useful for error correction. As was mentioned earlier, forcing the user to decide if something is a false positive or false negative is mentally taxing, so a high detection threshold would be preferred.

6.2.4 Audio/Transcript Synchronization and Variable Rate Playback

When correcting another person's prerecorded speech, such as a student correcting a professor's spoken lecture, listening to the audio is critical to successfully correcting the current utterance. Two key improvements could be made to audio playback in our error correction interface—better synchronization between the audio and the transcript, as well as the ability to slow down the audio. Currently audio is on an utterance-based granularity whereas a word-based granularity would be more effective. One could use the word segmentation boundaries proposed by the recognizer to align the audio.

In addition, it can be difficult to follow fast speakers, especially coupled with the false starts, filler words, and filled pauses common to the lecture domain and other spontaneous, conversational speech data. One way to help users better follow the audio recordings would be to lower playback rate (independent of pitch) to decrease the speaking rate. These audio-based interface improvements would prevent users from unnecessarily listening to the audio multiple times per correction pass.

6.2.5 Additional Constraints During Re-recognition

A number of constraints can be added during the re-recognition phase to further improve performance. One key constraint would be using the length of the error gap to weight the probability of new hypotheses of various lengths. The duration distributions for error gaps based on their word length was measured in Chapter 5. Staying within a gap state of the error FST incurs a penalty each transition, favoring shorter hypotheses. Using timing information would reduce the likelihood of multi-word phrases being completely deleted during re-recognition due to this gap transition penalty. Timing information could be obtained from the time alignments proposed in the previous recognition pass. Phonetic information could also be used as a constraint for this task, using simple durational distributions based on phone counts, or using phone sequences as a feature for a word boundary classifier. A soft classifier could be developed using these timing and phonetic features to generate a new penalty score that could be easily incorporated into the error FST.

6.2.6 Avoiding Results in Non-Final States

When constrained re-recognition ends in a non-final state, it can be particularly harmful for error correction. The user's correction marks may be partially ignored, previously correct words wind up incorrect, or the utterance may raise an exception during the history process, all resulting in longer correction times. There are a few directions one could go to help prevent this problem. One way would be dynamic gap penalties that start low and increase each time search ends in a non-final state. This would result in repeated recognitions, possibly increasing overall correction time, but it would guarantee that the lowest gap penalty within the granularity of our increments was chosen. Another way would be to increase the pruning limits so that when a non-final state is reached, the search can backtrack to earlier decision points. Any method chosen would require additional time and space during search, so considerations for the target platform would need to be kept in mind.

6.2.7 Supervised Methods

All error correction schemes either implicitly or explicitly generate annotated data that can be used for supervised training of classifiers or adapting models. Since generating annotated data can be extremely time consuming and expensive, it would be wasteful not to harness the data retrieved through user corrections. In our experiments we investigated using correction data for supervised acoustic model adaptation, demonstrating the merit of the approach. More work could be done investigating more sophisticated interpolation schemes and tuning interpolation parameters. We could also examine on-line speaker adaptation and its performance in a live speech system.

Language model adaptation is another area that can be explored using supervised correction data. This could be done either off-line in batch mode, as in our speaker adaptation experiments, or on-line, using a cache-based language model. In the lecture domain, topic words are particularly important and are often referred to in the same n-gram context throughout the lecture, making an on-line approach helpful for IR tasks. Correction data from users can be applied to a variety of supervised approaches, improving recognition performance and the usability of speech interfaces. Error correction need not be a dead end in ASR applications, separated from recognition, but rather it should be recognized as a key component in the advancement of speech systems.

References

1. Shriberg, E., Wade, E., and Price, P. 1992. Human-machine problem solving using spoken language systems (SLS): factors affecting performance and user satisfaction. Proc. Workshop on Speech and Natural Language, Harriman, New York, February 1992. Human Language Technology Conference. Association for Computational Linguistics, Morristown, NJ.
2. Glass, J., Hazen, T., Hetherington, L., and Wang, C. 2004. Analysis and processing of lecture audio data: Preliminary investigations. Proc. HLT-NAACL 2004 Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval, Boston, MA, May, 2004.
3. Park, A., Hazen, T., and Glass, J. 2005. Automatic Processing of Audio Lectures for Information Retrieval: Vocabulary Selection and Language Modeling. Proc. ICASSP, Philadelphia, March 2005.
4. Gruenstein, A. and Seneff, S. 2006. Context-Sensitive Language Modeling for Large Sets of Proper Nouns in Multimodal Dialogue Systems. Proc. IEEE/ACL 2006 Workshop on Spoken Language Technology, Palm Beach, Aruba, December 2006.
5. Gruenstein, A., Seneff, S., and Wang, C. 2006. Scalable and Portable Web-Based Multimodal Dialogue Interaction with Geographical Database. Proc. Interspeech, Pittsburgh, Pennsylvania, September 2006.
6. Gruenstein, A. and Seneff, S. 2007. Releasing a Multimodal Dialogue System into the Wild: User Support Mechanisms. Proc. of the 8th SIGdial Workshop on Discourse and Dialogue, Antwerp, Belgium, pp. 111-119, September 2007.
7. Glass, J. 2003. A probabilistic framework for segment-based speech recognition.

Computer Speech and Language, Vol. 17 pp. 137-152.

8. Larson, K. and Mowatt, D. 2003. Speech Error Correction: The Story of the Alternates List. International Journal of Speech Technology Vol. 6 No. 2.
9. Oviatt, S. and VanGent, R. 1996. Error resolution during multimodal human-computer interaction. Spoken Language, Vol. 1 pp. 204-207, October 1996.
10. Koester, H. 2003. Abandonment of speech recognition by new users. Proc. RESNA'03.
11. Lieberman, H., Faaborg, A., Daher, W., and Espinosa, J. 2005. How to wreck a nice beach you sing calm incense. Proc. 10th international Conference on Intelligent User interfaces, San Diego, California, USA, January 2005.
12. Stolcke, A., Konig, Y., and Weintraub, M. 1997. Explicit word error minimization in N-best list rescoring. Proc. 5th European Conference on Speech Communication and Technology, Rhodes, Greece, 1997.
13. Mangu, L., Brill, E., Stolcke, A. 2000. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. Computer Speech and Language Vol. 14 No. 4 pp. 373-400, October 2000.
14. Mangu, L., Padmanabhan, M. 2001. Error corrective mechanisms for speech recognition. Proc. ICASSP 2001.
15. Ogata, J., Goto, M. 2005. Speech repair: quick error correction just by using selection operation for speech input interfaces. Proc. Interspeech 2005.
16. Ogata, J., Goto, M., Eto, K. 2007. Automatic Transcription for a Web 2.0 Service to Search Podcasts. Proc. Interspeech 2007 Antwerp, Belgium.
17. Ogata, J., Goto, M., Eto, K. 2007. PodCastle: A Web 2.0 Approach to Speech Recognition Research. Proc. Interspeech 2007 Antwerp, Belgium.
18. Halverson, C., Horn, D., Karat, C., and Karat, J. 1999. The beauty of errors: patterns of

error correction in desktop speech systems. Proc. INTERACT '99 Edinburgh, September 1999, IOS Press, pp. 133-140.

19. Karat, J., Horn, D. B., Halverson, C. A., and Karat, C. M. 2000. Overcoming unusability: developing efficient strategies in speech recognition systems. In CHI '00 Extended Abstracts on Human Factors in Computing Systems, The Hague, The Netherlands, April 2000.
20. Karat, C., Halverson, C., Horn, D., and Karat, J. 1999. Patterns of entry and correction in large vocabulary continuous speech recognition systems. Proc. SIGCHI Conference on Human Factors in Computing Systems: the CHI Is the Limit, Pittsburgh, Pennsylvania, United States, May 1999.
21. Oviatt, S., MacEachern, M., and Levow, G. 1998. Predicting hyperarticulate speech during human-computer error resolution. Speech Communication Vol. 24 Issue 2, May 1998, pp. 87-110.
22. Kolsch, M., Turk, M. 2002. Keyboards without Keyboards: A Survey of Virtual Keyboards. Technical Report 2002-21. University of California, Santa Barbara.
23. Cockburn, A. and Siresena A. 2003. Evaluating Mobile Text Entry with the Fastap Keypad. In British Computer Society Conference on Human Computer Interaction, England, 2003.
24. Zhou, L., Shi, Y., Zhang, D., and Sears, A. 2006. Discovering Cues to Error Detection in Speech Recognition Output: A User-Centered Approach. Journal of Management Information Systems Vol. 22 No. 4 pp. 237-270, April 2006.
25. Sarma A. and Palmer D. 2004. Context-based speech recognition error detection and correction. Proc. HLT-NAACL 2004.
26. Liu, P. and Soong, F. K. 2006. Word graph based speech rcognition error correction by

handwriting input. Proc. 8th international Conference on Multimodal Interfaces, Banff, Alberta, Canada, November 2006.

27. Hetherington, L. 2004. The MIT Finite-State Transducer Toolkit for Speech and Language Processing. Proc. ICSLP, Jeju, South Korea, October 2004.
28. Chung, G., Seneff, S., Wang, C., and Hetherington, L. 2004. A Dynamic Vocabulary Spoken Dialogue Interface. Proc. Interspeech, Jeju, South Korea, October 2004.
29. Kumaran, R., Bilmes, J., and Kirchhoff, K. 2007. Attention-Shift Decoding for Conversational Speech Recognition. Proc. Interspeech 2007, Antwerp, Belgium.
30. Hsu, B.J., Mahajan, M., and Acero, A. 2005. Multimodal Text Entry on Mobile Devices. ASRU, San Juan, Puerto Rico, December 2005.
31. Wald, M., Boulain, P., Bell, J., Doody, K. and Gerrard, J. 2007. Correcting Automatic Speech Recognition Errors in Real Time. International Journal of Speech Technology.
32. Munteanu, C., Penn, G., Baecker, R., and Zhang, Y. 2006. Automatic speech recognition for webcasts: how good is good enough and what to do when it isn't. Proc. 8th International Conference on Multimodal Interfaces, Banff, Alberta, Canada, November 2006.
33. Schiller, S. SoundManager: Javascript to Flash Sound API. http://www.schillmania.com/projects/soundmanager/
34. Card, S. K., Moran, T. P., and Newell, A. 1980. The keystroke-level model for user performance time with interactive systems. Commun. ACM Vol. 23 Num. 7 pp. 396-410, July 1980.
35. NIST 1998. SCLITE Scoring Package Version 1.5. http://www.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm
36. Yongmei, S., Lina, Z. 2006. Examining knowledge sources for human error correction.

Proc. Interspeech 2006.

37. Glass, J., Hazen, T., Cyphers, S., Malioutov, I., Huynh, D., and Barzilay, R. 2007. Recent Progress in the MIT Spoken Lecture Processing Project. Proc. Interspeech, Antwerp, Belgium, August 2007.
38. Bourguet, M. 2006. Towards a taxonomy of error-handling strategies in recognition-based multi-modal human-computer interfaces. Signal Processing Vol. 86 Issue 12, December 2006.
39. Suhm, B., Myers, B., and Waibel, A. 2001. Multimodal error correction for speech user interfaces. ACM Transactions on Computer-Human Interaction Vol. 8 Issue 1, March 2001.
40. McNair, A.E., and Waibel, A. 1994. Improving Recognizer Acceptance Through Robust, Natural Speech Repair. ICSLP 1994, Yokohama Japan.
41. Sturm, J. and Boves, L. 2004. Effective error recovery strategies for multimodal form-filling applications. Speech Communication, Vol. 45 Issue 3.
42. Suhm, B., Myers, B., and Waibel, A. 1999. Model-based and empirical evaluation of multimodal interactive error correction. Proc. SIGCHI Conference on Human Factors in Computing Systems: the CHI Is the Limit, Pittsburgh, Pennsylvania, United States, May 1999.
43. Deng, L. and Huang, X. 2004. Challenges in adopting speech recognition. Commun. ACM Vol. 47 Issue 1, January 2004.
44. Shneiderman, B. 2000. The limits of speech recognition. Commun. ACM Vol. 43 Issue 9 September 2000.